h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

Evaluation vortrainierter neuronaler Netzwerke zur Anwendung auf die autonome Zählung von Personen mit Objektdetektion

Fachbereich Informatik und Fachbereich Mathematik

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

vorgelegt von

Oskar Rudolf

Matrikelnummer: 763479

Referentin : Prof. Dr. Elke Hergenröther

Korreferent : Prof. Dr. Sebastian Döhler

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 13. Mai 2024

Oskar Rudolf

Zusammenfassung

- **Motivation**: Die fortlaufende Entwicklung der Prozessortechnologie bietet immer bessere Möglichkeiten der Anwendung von Computer Vision und neuronalen Netzwerke in der Praxis. Angesichts des technologischen Fortschritts ist es Ziel dieser Arbeit, die Anwendbarkeit und Effizienz von neuronalen Netzwerkarchitekturen allgemein, unter hardwareeingeschränkter Systemumgebung und im Realszenario zu untersuchen. Mit dieser Studie soll ein Beitrag zur Nutzung dieser Technologien zur Entwicklung intelligenter und nachhaltiger Umgebungssteuerungssysteme beizutragen.

- **Inhalt**: Verschiedene vortrainierte neuronale Netzwerke werden hinsichtlich ihrer Eignung für die Zählung von Personen unter Verwendung von Objektdetektion analysiert. Es wird das methodische Vorgehen und die resultierende Evaluation von 45 verschiedenen Netzen zur Objekterkennung, eine Auswertung der Leistung einiger Netzwerke an begrenzte Hardwarekapazitäten und die Anwendung in einem simulierten Realszenario beschrieben. Zusätzlich erfolgt eine subjektive Bewertung von high-level Benutzerschnittstellen für neuronale Netze auf Benutzerfreundlichkeit und Kompatibilität in realen Szenarien anhand der Beobachtungen und gesammelten Erfahrungen.

- **Ergebnisse**: Die Evaluation der Netzwerkarchitekturen identifiziert YOLO-Modelle im Vergleich als beste Gruppe von Netzwerkarchitekturen. Es hat sich gezeigt, dass sie im Vergleich zu anderen Architekturklassen bezüglich Genauigkeit und Inferenzzeit überlegen sind und selbst mit eingeschränkten Hardwarekapazitäten und ungewöhnlichen Kameraperspektive im Bürosetting in der Lage sind, zuverlässig Personen zu detektieren. Die Studie bestätigt das Potenzial von KI-gestützten Systemen zum Einsatz in Gebäudeautomation und -management über Objekterkennung. Zukünftige Forschung könnten sich darauf konzentrieren, weitere Modelle in die Analyse einzubeziehen und weitere Möglichkeiten zu untersuchen, neuronale Netze in Kleingeräte und eingebette Systeme einzubauen, um die Einsatzmöglichkeiten dieser Technologien im Alltag zu verbessern.

Abstract

- **Motivation**: The ongoing development of processor technology and parallel computing capabilities provides new opportunities for the application of computer vision and neural networks in practice. Given this technological advancement, this study aims to investigate the applicability and efficiency of neural network architectures in general, under hardware-constrained system environment and in a real-world context. The aim of this study is to contribute to the use of these technologies for the development of intelligent and sustainable environmental control systems.

- **Content**: Various pre-trained neural networks are analyzed with respect to their suitability for counting people using object detection. The methodological procedure and the resulting evaluation of 45 different networks for object detection is described, as well as the evaluation of some chosen networks on limited hardware capacities and the application in a simulated real life scenario. In addition, a subjective evaluation of high-level user interfaces for neural networks in terms of user-friendliness and compatibility in real scenarios is carried out on the basis of observations and experience gained.

- **Results**: The evaluation of the network architectures identifies YOLO models as the dominant group of network architectures regarding performance in an overall comparison. It has been shown that they are superior to other architecture classes in terms of accuracy and inference time and are able to reliably detect people even with limited hardware capacities and unusual camera perspectives. The study confirms the potential of AI-supported systems for use in building automation and management via object detection. It becomes clear that integrating advanced computer vision technologies into the infrastructure is an effective way to solve energy consumption and security challenges. Future research could focus on incorporating more models into the analysis and investigating further ways of incorporating neural networks into small devices and embedded systems to improve the application possibilities of these technologies in everyday life.

Danksagungen

Wir sind wie Zwerge auf den Schultern von Riesen, damit wir weiter sehen als sie, nicht
weil unsere Sicht schärfer ist oder unsere Statur größer, sondern weil sie uns emporheben
und uns ihre gewaltige Größe verleihen.

- Gelehrter und Philosoph Bernard von Chartres, 12. Jahrhundert

Meine Wertschätzung und mein Dank gehen an die Mühen und Errungenschaften aller vergangenen forschenden Geister unserer Wissensgemeinschaft. Ohne auf deren Schultern zu stehen, wären wir heute nicht in der Lage, so weit in die Ferne zu blicken. Darüber hinaus gilt der Dank meiner Frau Maria für ihre fortwährende und bedingungslose Unterstützung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CV Computer Vision

ML Machine Learning

KI Künstliche Intelligenz

CNN Convolutional Neural Networks

YOLO You Only Look Once

GPU Graphics Processing Unit

CPU Central Processing Unit

RAM Rapid Access Memory

LiDAR Light Detection and Ranging

TF TensorFlow

IoT Internet of Things

API Application Programming Interface

JPEG Joint Photographic Experts Group

RGB Rot-Grün-Blau

IoU Intersection of Unions

FPS Frames Per Second

COCO Common Objects in Context

EfficientDet Efficient Detection

CenterNet Center Network

ResNet Residual Network

MobileNet Mobile Network

ReLU Rectified Linear Unit

MSE Mean Squared Error

DNN Deep Neural Network

COCO Common Objects in Context

Teil I

Thesis

1 Einleitung

Aufgrund voranschreitender technologischer Fortschritte ist die Verfügbarkeit und Anwendbarkeit von KI-System heutzutage so sehr gegeben wie noch nie. In dieser Masterarbeit soll die Möglichkeit der Anwendbarkeit bereits etablierter KI-Systeme im Bereich der Computer Vision untersucht werden. Diese Masterarbeit befasst sich mit einem Forschungsprojekt zur Erforschung der Nutzung bereits frei verfügbarer KIs im Alltag am Beispiel des Anwendungsfalls der Menschenzählung in einem Büro.

Zunächst wird in Kapitel 1 der Hintergrund und die Motive der Arbeit genauer erläutert und dessen Ziele ausformuliert. Anschließend folgt eine Beschreibung der zugrundeliegenden Technologien und der dazugehörigen aktuellen Forschungslage und best-practise Verfahren (s. Kapitel 2). Nach der Erläuterung der theoretischen Grundlagen wird im nächsten Abschnitt ein Überblick über die angewandte Methodik der Untersuchung gegeben (Kapitel 3), gefolgt von einer sachlichen Darstellung der Ergebnisse (Kapitel 4). Im Diskussionskapitel 5 danach werden die Ergebnisse hinsichtlich ihres Informationsgehalts unter Berücksichtigung weiterer Studien und eigener Erkenntnisse analysiert, ausgewertet und interpretiert. Zuletzt werden sämtliche aus dieser Studie resultierende Forschungsergebnisse, dessen Limitationen und Anstöße für zukünftige Forschung in einem Abschlusskapitel (6) zusammengefasst.

1.1 Hintergrund und Motivation

Wir erleben derzeit in fast allen Bereichen der Technik konsistent immer neue Durchbrüche bei immer kürzeren zeitlichen Abständen zwischen den Entwicklungsstufen. Diese Entwicklung lässt sich gut anhand der Beobachtung technologischer Meilensteine in der Geschichte zeigen (Abbildung 1.1). Ein in diesem Zusammenhang ursprünglich oft zitiertes Posulat ist das **Mooresche Gesetz**[1], nach dem sich die Anzahl Transistoren auf integrierten Schaltkreisen in regelmäßigen Abständen verdoppelt. Diese Aussage wird zwar nicht für alle Zeit haltbar sein, da ab einem bestimmten Punkt die physikalische Grenze der Größe eines Teilchens nicht mehr unterschritten werden kann, sie dient aber

als illustratives Beispiel für die exponentielle Entwicklung des technologischen Fortschritts.

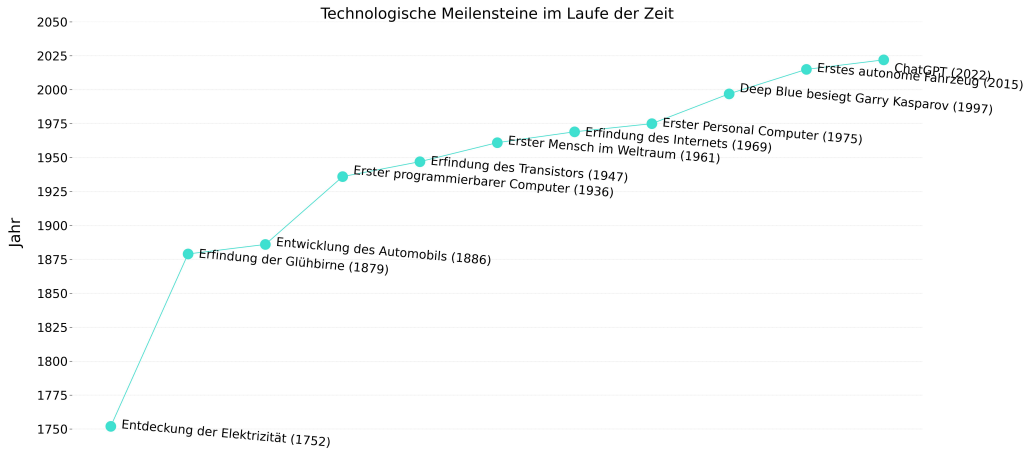


Abbildung 1.1: Illustration des technologischen Fortschritts anhand von Meilensteinen

Der u.a. nach Moor prognostizierte Trend in der Entwicklung der Leistungsfähigkeit von Computerprozessoren hielt sich bis zum Jahr 2020 und führt zu einer deutlichen Reduktion der Anschaffungskosten für leistungsfähige Prozessoren (so beispielsweise die Kosten von Intel-Prozessoren[2]). Dementsprechend sind leistungsstarke Prozessoren immer zugänglicher für die breite Masse geworden und werden heutzutage auch in Mobiltelefonen in der breiten Menge eingesetzt. Insbesondere Computerchips, die für parallele Berechnung von Fließkommaoperationen optimiert wurden (üblicherweise GPUs, die zunächst vor allem für Videosimulationen und Videospiele verwendet wurden), etablierten sich im Laufe der letzten Jahre als nützliche Werkzeuge zur Berechnung von aufwendigen mathematischen Problemen, wie zum Beispiel dem sogenannten Mining bei Kryptowährungen[3] oder der iterativen Parameterberechnung beim Training von Künstliche Intelligenz (KI)-Systemen, in denen sie im Vergleich zum CPU deutlich effizienter sind[4].

Parallel zur Entwicklung von besserer Hardware wurde auch die Implementierung komplexer Algorithmen im Laufe der Zeit immer effizienter. Ein anschauliches Beispiel etwa ist die Evolution von Sortieralgorithmen vom einfachen *Bubble Sort* hin zum *Quick Sort*[5]. Unter Anwendung von KI-Technologie (AlphaDev) gelang es vor kurzer Zeit einer Forschergruppe sogar, den schon als maximal optimiert geltenden Sortieralgorithmus noch einmal deutlich zu verbessern[6].

Der Einsatz von KI wird im Kontext des fortgeschrittenen Stands der Technik immer attraktiver, denn je nach Parameterzahl können inzwischen großdimensionierte Matrixmultiplikationen binnen kurzer Zeit berechnet werden, welche in nicht allzuweiter Vergangenheit noch Wochen gedauert hätten. Das Forschungsgebiet der KI hat sich infolgedessen rasch erweitert. KI dient oft als Überbegriff für technische Systeme, die überwiegend autonom agieren und für Aufgaben eingesetzt werden, die menschliche Intelligenz erfordern. Zu solchen Prozessen gehören lernen, schlussfolgern, Problemlösung, Verstehen natürlicher Sprache und Wahrnehmung von Umweltreizen[7].

Fasst man den Begriff der KI nach Definition in den Datenwissenschaften enger, so stellt er im Wesentlichen ein statistisches Modell dar, dessen Ziel es ist, anhand bereits bekannter Daten Muster zu extrahieren und aus diesen eine möglichst genaue Prognose für zukünftige Ereignisse mit neu auftretenden Datenpunkten abzuleiten. Der Bezug zum allgemeinen Verständnis des Intelligenzbegriffs ist dadurch gegeben, dass man den Prozess des Annäherns der Modellparameter an die Daten in diesem Zusammenhang als Lern- oder Trainingsprozess (engl. Machine Learning (ML)) bezeichnet. Je nach Anwendungsfeld und Trainingsprozedur kann ML in weitere Unterkategorien untereilt werden: Beim überwachten (supervised) Lernen kennt das Modell im Trainingsprozess die dem Datensatz zugehörige Kategorie (Label), wogegen beim unüberwachten Lernen keine solche Zuordnung stattfindet, sodass das Modell gezwungen ist, anhand der Gesamtdaten eigenständige Datengruppen (Cluster) zu bilden. Im letzteren Falle bleibt die Interpretation der Clustergruppen derzeit noch dem menschlichen Benutzer überlassen. Über diese beiden Teilbereiche hinaus gibt es noch das an die operante Konditionierung nach Skinner[8] angelehnte verstärkte Lernen (Reinforcement Learning). Nach der Theorie der operanten Konditionierung nimmt die Wahrscheinlichkeit eines Verhaltens zu oder ab, je nachdem, ob es eine Belohnung oder Bestrafung als Konsequenz hat. Entsprechend wird bei dieser Form der KI ein initial unwissender Agent über ein Punktesystem mit Belohnungs- und Bestrafungsreizen trainiert.

Berücksichtigt man die explosive technologische Entwicklung in den zahlreichen Sektoren wird die Vorstellung praktischer Anwendungen von KI auf kleinen Geräten und sogar in autonomen eingebetten Systemen auf Mikrocontrollern immer greifbarer. Die Erforschung der Anwendung komplexer Algorithmen in kleine autonome Systeme bringt erhebliche Vorteile in diversen Branchen. Sie können in zahlreichen Anwendungsfällen zur Sicherheit, Erhöhung des Lebensstandards und Schonung von Ressourcen hinsichtlich Personal, Energie

und Material beitragen. Studien zur Anwendung in Sicherheitssystemen[9], Fahrassistenzsystemen[10], Medizin[11] oder Produktionsprozessen[12] belegen dies.

Der Nutzen wird umso höher, wenn aufgrunddessen teure Hardware durch simplere Systeme ausgetauscht werden kann. Ein besonders interessantes Forschungsgebiet, in dem dieser Fall eintritt, ist der Bereich der Computer Vision (CV). Bei CV handelt es sich um die Verarbeitung von elektronischen Bilddaten für die maschinelle Auswertung. Analog zur visuellen Wahrnehmung beim Menschen hat in diesem Zusammenhang CV das Ziel eine Form des maschinellen Sehens umzusetzen[13]. Der Einsatz von CV ermöglicht es, herkömmliche Videokameras zu verwenden und somit teure Wärmebildkameras oder Light Detection and Ranging (LiDAR)-Systeme durch den Einsatz von Objekterkennungsalgorithmen überflüssig zu machen. Bildaufnahmen sind kosteneffizienter, ressourcenschonender und weniger anfällig für spezifische Sensor-Störquellen. Darüber hinaus sind Bildaufnahmen nicht zweckgebunden, sondern können simultan für mehrere Funktionen eingesetzt werden. Der hohe Nutzen von Objekterkennungsalgorithmen spiegelt sich bereits in ihrem breiten Einsatz im Alltag für Extraktion von Informationen über anwesende Personen, Gegenstände und Abläufe wieder, beispielsweise beim autonomen Fahren[14] oder Verifizierung von Personen über eine Gesichtserkennung mit dem Smartphone[15].

Diese Systeme sind nützlich, um mehr Prozesse im alltäglichen Leben zu automatisieren und das sogenannte Internet of Things (IoT) auszubauen. Die Automatisierung trägt dazu bei, Fehlerraten zu reduzieren (Beispiel: Medizin[16]), ist aber vor allem sparsam und kann die globalen Anstrengungen bei der Einsparung von Ressourcen und folglich der Reduzierung von Treibstoffemissionen unterstützen. Umso mehr Prozesse autonom gesteuert werden können, desto ressourcensparender und somit nachhaltiger werden sie. Oft sind die Auswirkungen nur indirekt beobachtbar, etwa durch eine Reduzierung der Laufzeit von Produktionsprozessen. Im Gegensatz dazu hat die Implementierung in Smarthome-Systemen zur Optimierung von Steuerung der Strom- und Wärmeressourcen einen beobachtbaren Einfluss auf die Energiestatistik und macht sich auf der Stromrechnung direkt bemerkbar. Studien[17,18] haben gezeigt, dass durch den Einsatz von Smarthome-Systemen erhebliche Mengen an Energie eingespart werden können (von 15% bis hin zu 70 %). Insbesondere deshalb ist dieses Anwendungsgebiet öffentlichkeitswirksam und kann der Skepsis gegenüber Anwendung von KI im Alltag einiger Teile der Bevölkerung entgegenwirken. Es lohnt sich damit als ein Fokusobjekt der wissenschaftlichen Auseinandersetzung. Diese Untersuchung beschäftigt sich mit einer möglichen konkreten Umsetzung der Anwendung von

CV-Methoden zur Erkennung der Personenzahl in Räumlichkeiten über die Bildaufnahme mit einer handelsüblichen Kamera und einem kostengünstigen Hardware-Setup.

Die konkrete Auswahl des optimalen Algorithmus bei der Implementierung von KI wird durch die Vielzahl an inwzischen erforschten ML-Algorithmen erschwert. Zu den am häufigsten verwendeten Algorithmen[19,20] zählen die lineare und logistische Regression, Entscheidungsbäume, Zufallswälder, Support-Vektor-Maschinen, K-Means-Clustering, K-Nächste-Nachbarn, Neuronalen Netze und das Naive Bayes-Verfahren. Jeder Algorithmus enthält zahlreiche sogenannte Hyperparemter. Diese stellen bestimmte Einstellungen bezüglich der Eigenschaften des statistischen Modells dar und dienen meistens als Stellschrauben zum Feinjustieren der Modelle. Im Falle eines neuronalen Netzes etwa kann man die Anzahl an Neuronen in den jeweiligen Schichten einstellen (s. Kapitel 2.2). Es nicht unüblich, die heutzutage verfügbare Prozessorleistung voll auszunutzen und mehrere Modellansätze auszutesten, um über ein Suchraster die idealle Konfiguration von Modelltyp und dessen Hyperparametern explorativ zu finden. Obwohl diese Methode für KI-affine AnwenderInnen und WissenschaftlerInnen sinnvoll sein kann, benötigt sie Zeit und statistische Kenntnisse über Funktionsweise der Hyperparameter, sowie fachspezifische Programmierkenntnisse. Zudem ist es essentiell, eine große Menge an Trainingsdaten zu akquirieren und den exponentiellen Faktor für Berechnungszeit mit Aufnahme eines jeden Hyperparameters im Suchraster zu berücksichtigen. Darüber hinaus ist man darauf angewiesen, in jedem neuen Anwendungsfall ein für diesen Zweck optimiertes ML-Modell zu trainieren. Folglich ist es trotz dem derzeitigen Entwicklungsstand der Technik nötig, eine signifikante Menge an Ressourcen für die Integration von KI in eine Infrastruktur aufzuwenden. Diese Hürde kann einerseits KI-Technologie trotz ihrer enormen Möglichkeiten nicht nur unattraktiv für den Einsatz in der Industrie machen, sondern ist im Sinne eines nachhaltigen Managements von Ressourcen zum Umweltschutze nicht zu empfehlen.

Es bietet sich daher an, außerhalb vom Forschungskontext in der Praxis auf bereits verfügbares Wissen über performante KI-Modelle zuzugreifen. Es stehen für diesen Zweck bereits high-level Benutzerschnittstellen - engl. Application Programming Interface (API) - zur Verfügung, die vorkonfigurierte, optimierte und trainierte KI-Modelle öffentlich zugänglich anbieten. Je nach Bedarf können diese Schnittstellen genutzt werden, um KI direkt in Systeme zu implementieren oder auf den speziellen Anwendungsfall feinzujustieren. Speziell für den Fall

des Einsatzes von Objekterkennungsalgorithmen sind zwei Möglichkeiten solcher Schnittstellen der **Model Zoo** von Googles open-source-Plattfrom für Machine Learning Anwendungen TensorFlow (TF)[21] und die Plattform Ultralytics[22]. Diese Benutzerschnittstellen stellen eine noch sehr junge Entwicklung im Bereich KI dar, bieten aber erhebliches Potential für die Einbindung intelligenter Systeme ohne spezifisches Vorwissen oder Trainingsaufwand. Aus diesem Grund sind die zwei obengenannten Schnittstellen ebenfalls im Rahmen dieser Untersuchung oberflächlich in Bezug auf ihre Eignung in verschiedenen Kontexten exploriert werden. Die Forschungsergebnisse können damit als Grundlage für Implementierungen im Bereich Einsatz von KI und CV in autonomen Systemen dienen.

1.2 Ziele der Untersuchung

Das Ziel dieser Arbeit ist die Überprüfung dessen, inwiefern eine effiziente Anwendung von Machine Learning Algorithmen im Bereich CV mit heutigen Technologien im Feld unter Berücksichtigung von allen Herausforderungen des machinellen Sehens und eingeschränkter Hardwarekapazitäten realisierbar ist. Für diesen Zweck wird eine Evaluation verschiedener bereits vortrainierter neuronaler Netze für den Einsatz der Objekterkennung unter Berücksichtung der Faktoren Ressourcenanforderungen, Modellkomplexität, Inferenzzeit und Messgenauigkeit anhand eines öffentlichen Bilddatensatzes sowie einer Feldstudie im Labor durchgeführt. Für die Einbindung der Modelle werden die von TF und Ultralytics zur Verfügung gestellten Schnittstellen eingesetzt. Die Ergebnisse dienen dem Ziel, die Realisierbarkeit der Implementierung eines effizienten neuronalen Netzes zur Detektion von Personen in einem Raum zu evaluieren. Im regulären Anwendungsfall ist davon auszugehen, dass die Detektionsumgebung nicht den idealen Umständen entspricht. Die Testdaten werden daher zusätzlich augmentiert, um die Störanfälligkeit und Anpassungsfähigkeit der neuronalen Netze in die Analyseergebnisse einfließen zu lassen. Um neben der Leistung der Netze auf öffentlich zugänglichen Daten deren Generalisierbarkeit auf die Anwendung in Büroräumen zu testen, soll über eine Erhebung von zusätzlichem Videomaterial im Labor zur Simulation einer Realumgebung mit einer Weitwinkelkamera eine fundierte Aussage über die Praxisnähe der neuronalen Netze getroffen werden.

Um den Einsatz in einem System mit vergleichsweise niedrigen Ressourcen (Low-End) zu prüfen, wird zunächst eine Vorselektion über die Evaluation in einem leistungsstarken (High-End) System der bestgeeignetesten Modellarchitekturen

vorgenommen. Anschließend wird die Leistung dieser Algorithmen in Bezug zum Low-End-System gesetzt. Die Forschungsergebnisse sollen zudem Erkenntnisse über technische und praktische Beschaffenheiten und Herausforderungen der Implementierung von neuronalen Netzen in Kleingeräten und eingebetteten Systemen liefern. Ferner sollen auf Grundlage dieser Untersuchung Aussagen über die Generalisierbarkeit von derzeit frei verfügbaren vortrainierten KI-Modellen zur Anwendung im Bereich CV ohne zusätzlichen Trainingsaufwand getroffen werden können. Neben diesen Hauptzielen erfolgt auch eine oberflächliche Beurteilung der Nutzerfreundlichkeit und Kompatibilität mit unterschiedlichen Programiermodulen und Betriebsystemen bezüglich der Implementierbarkeit der Modelle, die von den Benutzerschnittstellen zur Verfügung gestellt werden.

2 Theoretischer Hintergrund

In diesem Kaptiel folgt eine ausführliche Beschreibung der bis heute etablierten Technologien mit Relevanz für die Untersuchung der Forschungsziele. Darunter fällt zunächst die Beschreibung von CV im Allgemeinen. Anschließend wird das Prinzip der Funktionsweise neuronaler Netze erläutert, die sich als *state of the art* ML-Algorithmus im Bereich CV etabliert haben. Unter den neuronalen Netzen wird insbesondere auf eine spezielle Art von Netzwerken, Faltungsnetzen (Convolutional Neural Networks (CNN)), eingegangen und auf die Besonderheiten der Architekturen verschiedener Modelle, die von den high-level API's von TF und Ultralytics angeboten werden, eingegangen.

2.1 Definition von Computer Vision

Neurologisch betrachtet ist die visuelle Wahrnehmung bei Lebewesen ein komplexes System aus Prozessen, in denen Lichtstrahlen über Rezeptoren im Auge aufgenommen und über Nervensignale soweit verarbeitet und abstrahiert werden, dass sie als eine Abffolge von Bildern in Echtzeit wahrgenommen werden können[23]. Das wahrnehmende Lebewesen nutzt diese Bildinformationen für eine Interpretation der eingehenden Reize und kann entsprechend reagieren. Möchte man diese Fähigkeit in die Welt der Computersprache übertragen, so ergibt sich bei CV, wie auch in anderen Bereichen der Informatik, ein großes Sortiment an Möglichkeiten. Es liegt in der Verantwortung der Anwender zu entscheiden, welche Methode unter Berücksichtigung der verfügbaren Ressourcen das beste Verhältnis von Kosten und Leistung bringt. Obwohl KI-Werkzeuge in diesem Kontext immer prävalenter werden, sollte man beachten, dass auch alternative Algorithmen zur Lösung von CV-Anwendungsfällen existieren, die manchmal den effizienteren Ansatz darstellen. Diese Kategorie von CV-Implementierungen steuert die maschinelle Verarbeitung von Bildern auf Pixel- oder Eigenschaftsbasis ohne Einsatz von iterativen ML-Verfahren. Beispiele für spezielle Anwendungsgebiete in denen sie angewendet werden sind beispielsweise die Detektion von Kanten (Canny-Algorithmus[24]) oder der Abgleich von Bildern mit einer $N \times N$ Pixelmatrix zur Detektion von Mustern (Template Matching[25]). Auch in der Unterscheidung von Farbtönen können pixelbasierte Verfahren eingesetzt werden, um schnell

und effizient gewünschte Systemantworten zu produzieren, zum Beispiel bei der Verarbeitung von Pixelwerten zur Unterscheidung von für Menschen schwer erkennbaren Farbkontrasten. Siehe dazu Abbildung 2.1, in der durch eine einfache Transformation eines Bildes[26] mit für Menschen schwer unterscheidbaren Farbkontrasten von Farbkanläen zu Schwarzweiß-Pixeln deutlich bessere Konturen sichtbar werden, die sonst für manche Menschen eher schwer zu Unterscheiden sind.

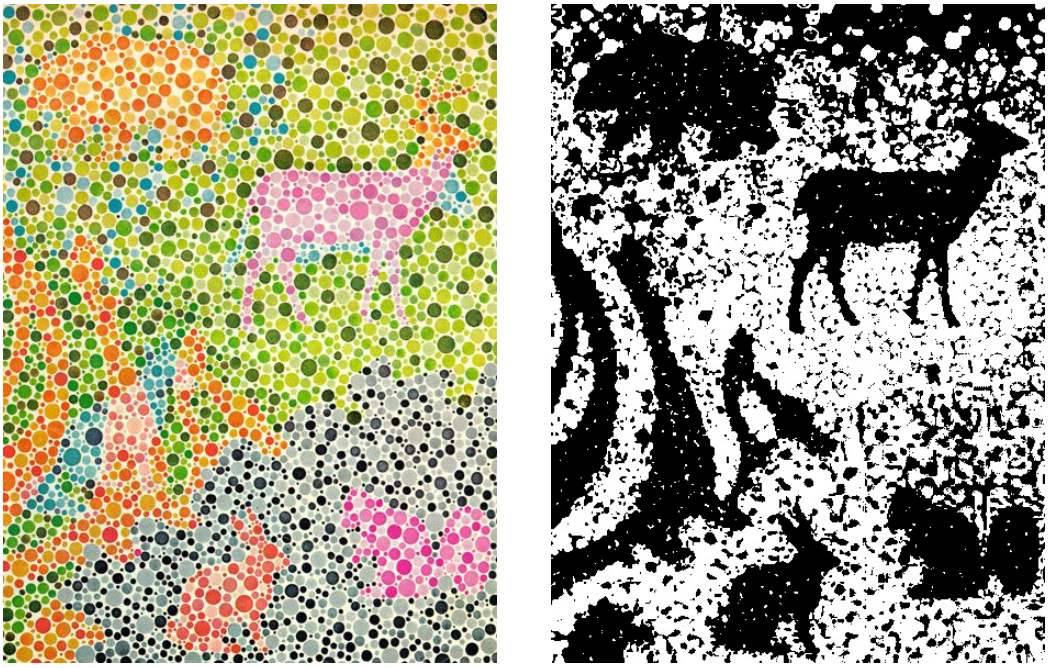


Abbildung 2.1: S-W-Farbtransformation anhand von Pixelwerten.

In Bereichen, in denen sich das Eingabeformat im Wesentlichen kaum ändert (z.B. Produktionsprozesse in der Industrie), ist diese Form des maschinellen Sehens sehr gut zur Detektion von Abnormalitäten einsetzbar. Ungeachtet dessen kann diese Form des menschlichen Sehens allerdings nicht gut in Präsenz von flexiblen Bildveränderungen oder der Komplexität einer Aktivumgebung wie dem Straßenverkehr angewendet werden. Sie sind daher meistens nicht für die Extraktion von Features von Bildern mit viel Bewegung oder häufigen Abweichungen (Größenveränderungen, Rotationen und Verschiebungen) geeignet.

Darüber hinaus weicht diese Form der Bildverarbeitung vom allgemeinen menschlichen Verständnis der visuellen Wahrnehmung ab. Möchte man analog zur menschlichen visuellen Wahrnehmung Computern über die Eingabe von Daten das Sehen beibringen, so können die traditionellen Algorithmen diesem Anspruch nicht mehr gerecht werden. Es entwickelte sich dementsprechend mit

dem rastanten Wachstum und der Verfügbarkeit von Computerprozessorleistung und der Entwicklung von KI ein neuer Pfad im Berich CV: Die breite Anwendung und Erforschung von ML- Algorithmen, darunter neuronalen Netze, die konzeptuell an das natürliche Nervensystem angelehnt sind und deren Weiterentwicklung - Faltungsnetze (engl. CNN). Je nach Wahl und Feinjustierung des zu trainierenden ML-Modells und der Größe des Trainingsdatensatzes kann dieses in der Anwendung Muster in neuen (unbekannten) Datensätzen erkennen, abstrahieren, einordnen, interpretieren und auf dessen Grundlage Entscheidungen treffen. Auf CV angewendet bedeutet das, Maschinen visuelle Signale *verstehen* zu lassen. Ein klassisches Beispiel, das für diese Form des maschinellen Sehens herangezogen wird, ist die Klassifizierung von handgeschriebenen Zahlen auf Basis des MNIST-Datensatzes[27]. Aufgrund des intuitiven Verständnisses der Funktionalität von CNNs anhand dieses Beispiels wird es manchmal auch als das *Hello World* des MLs bezeichnet. Im Gegensatz zur Klassifizierung, in der ein ganzes Bild mit einem einzigen Label versehen wird (z.B Katze vs. Hund), werden für Objektdetektion überlicherweise rechteckige Rahmen (Bounding Boxes) verwendet, für die Klassen mit einer bestimmten Konfidenz vorhergesagt werden (s. blauer Rahmen im linken Bild von Abbildung 2.2). Um die Genauigkeit solcher Modelle zu messen, vergleicht man typischer Weise die überlappenden Flächeninhalte über die IoU der vorgelabelten Bounding Box (Ground Truth) und der vorhergesagten Bounding Box.

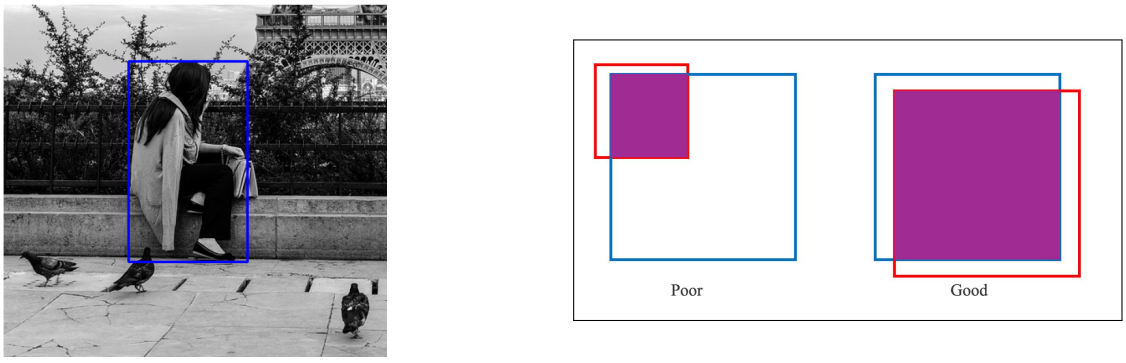


Abbildung 2.2: Beispiel für Bounding Box-Label aus dem COCO-Datensatz und IoU-Kalkulation.

$$\text{Berechnung IoU} = \frac{\text{Schnittmenge}}{\text{Vereinigungsmenge}} = \frac{|A \cap B|}{|A \cup B|}$$

Wie bereits angedeutet, haben sich CNN-basierte Verfahren in der Anwendung bei Problemen mit komplexen Bildgegebenheiten als besonders wirkungsvoll etabliert. Sie reduzieren die Probleme herkömmlicher Methoden hinsichtlich Skalierung, Rotation und Translation und bringen zudem ein gewisses Ausmaß an interpretierbarer Semantik in Form der Featuremaps und Generalisierbarkeit von verschiedener Bilderkennungsumgebungen. Aufgrunddessen wird die Funktionsweise von neuronalen Netzen, insbesondere CNNs und unterschiedlicher Varianten von neuronalen Netzwerkarchitekturen im Folgenden genauer erläutert.

2.2 Neuronale Netze

In diesem Abschnitt wird genauer auf das Konzept und die Funktionsweise von neuronalen Netzen zum Verständnis von deren Effektivität bei der Lösung von Objektdetektion eingegangen.

2.2.1 Ursprung und Funktionsweise neuronaler Netze

Ursprung neuronaler Netze. Laut aktuellem Wissensstand geht man beim durchschnittlichen menschlichen Gehirn von 85 bis über 105 Milliarden[28] Nervenzellen aus, die über mehr als 100 Billionen ($1 * e^{14}$) Verbindungen (Synapsen) miteinander verbunden sind. In diesem Netzwerk dienen Nervenzellen sowohl als Signalgeber, als auch Signalempfänger. Je nach Art der für die Übertragung eingesetzten Botenstoffe (Transmitter) kann ein Signal exzitatorisch (auslösend) oder inhibitorisch (hemmend) sein. Die elektrische Spannung aller an den Nervenenden (Dendriten) gleichzeitig eintreffenden hemmenden und auslösenden Nervensignale läuft an der Nervenzellmembran der Empfängerzelle zusammen und wird durch einen Spannungsausgleich zu einem Gesamtwert verrechnet. Überschreitet dieser elektrische Gesamtspannungswert eine bestimmte Grenze ($-40mV$), so gibt die Empfängerzelle infolge ein neues Signal (Aktionspotential) weiter. Auf diese Art und Weise werden Signale durch verschiedene Schichten und Gehirnareale propagiert. Umso häufiger bestimmte Signalketten zusammen Aktionspotentiale aussenden, desto stärker werden die entsprechenden Verbindungsstellen (Synapsen) zwischen den Nerven ausgebaut, was im Wesentlichen den Prozess des menschliche Lernens ausmacht (nach dem Prinzip *fire together, wire together*. Viele dieser Vorgänge wurden bereits im späten 19. und frühen 20. Jahrhundert entdeckt und beschrieben[29] [,30]. Obwohl die hier

beschriebene Darstellung der neurologischen Prozesse und Strukturen extrem stark vereinfacht ist, bot sie eine Grundlage für das Konzept der Funktionsweise neuronaler Netze in der Mitte des 20. Jahrhunderts und dient als eine gute Analogie zu deren Verständnis.

Die erste Prototyp-Version eines neuronalen Netzes namens Perzeptron wurde 1943 von Warren McCulloch und Walter Pitts entwickelt[31] und in den 1950er Jahren von Frank Rosenblatt[32] zum ersten Mal implementiert. Es besteht in seiner einfachsten Form aus je einer Eingabeschicht, einer Zwischenschicht und einer Ausgabeschicht (s. Abbildung 2.3). Wie auch im Gehirn besteht jede Schicht im Konzept der neuronalen Netze aus Zellen, den (künstlichen) Neuronen. Die Zellen zwischen den Schichten sind, analog zu Synapsen im Gehirn, mit Verbindungen untereinander ausgestattet, deren Verbindungsstärken über Gewichte in Fließkommazahlen repräsentiert werden. Ist jedes künstliche Neuron einer Schicht mit jedem Neuron der nächsten Schicht verbunden, so bezeichnet man diese auch als vollständig verbundene Schicht (fully connected layer). Analog zur statischen Spannung jeder Nervenzellmembran (Ruhepotential), trägt im neuronalen Netz jedes Neuron außerhalb der Eingangsschicht einen statischen Wert (Bias). Die initialen Verbindungsstärken und Anfangswerte im untrainierten Netz sind gewöhnlich zufällige Fließkommazahlen mit einem Mittelwert von 0 und werden auch als *weights and biases* bzw. *Modellparameter* bezeichnet[1]. Die genaue Konfiguration des Grundgerüsts eines neuronalen Netzwerks, darunter die Anzahl der künstlichen Neuronen, sowie Anzahl und Typ der Schichten wird allgemein als Netzwerkarchitektur bezeichnet. Hat ein neuronales Netz eine Architektur mit mehreren Hundert oder gar tausenden Schichten, so spricht man von einem Deep Neural Network (DNN). Manchmal wird im Zusammenhang von Netzwerkarchitekturen der Hauptteil eines Netzes auch als *Backbone* bezeichnet. Damit ist das Grundgerüst des Netzes aus Schichten gemeint, das im Anschluss modifiziert über das Basismodell hinaus modifiziert wird.

1 Es hat sich gezeigt, dass die Anwendung bestimmter Algorithmen und Zufallsverteilungen für den Initiierungsprozess die Konvergenz des Trainings beschleunigen und das typisches Problem von neuronalen Netzen der explodierenden oder verschwindenden Gradienten reduzieren kann[33]

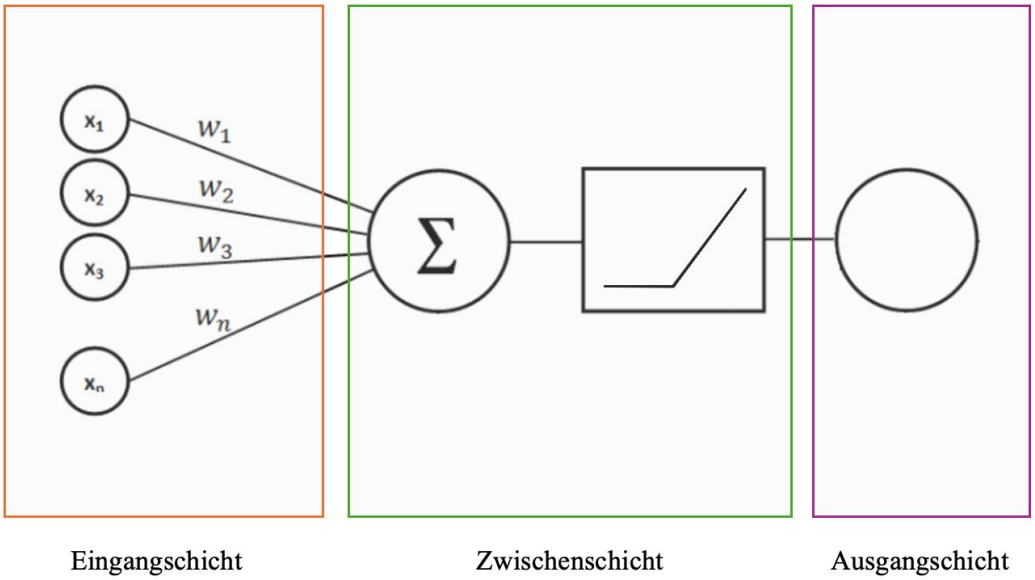


Abbildung 2.3: Graphische Illustration des Perzeptrons mit einer Zwischenschicht.

Im stark vereinfachten Applikationsbeispiel des MNIST-Datensatzes, dessen handgeschriebene Bilder (Abbildung 2.4) aus **28 x 28** Pixel mit Werten von 0 (schwarz) bis 255 (weiß) bestehen, könnte eine Architektur folgendermaßen aussehen: Eine Eingabeschicht aus $28 * 28 = 784$ Neuronen, zwei Zwischenschichten mit je 10 Neuronen und eine Ausgabeschicht aus 10 Neuronen (jeweils pro eines pro Ziffer). Bei der Selektion der Zwischenschichten ist der Gedanke, dass diese jeweils ein Maß an Abstraktion der Daten lernen und repräsentieren können sollten. Am Beispiel der MNIST-Zahlen wären das zunächst kleine Bildelemente, die in jeder Schicht zu größeren Elementen abstrahiert werden. Leider zeigt sich in der Praxis, dass die Zwischenschichten für Menschen eher weniger nachvollziehbare Abstraktionen lernen, die anstatt an Bildabschnitte zu erinnern, eher einem zufälligen Rauschen von Pixeln gleichen. Aus diesem Grund werden neuronale Netze manchmal auch als Black Box-Verfahren bezeichnet. Die erschwerte Interpretierbarkeit wird bei DNN noch gravierender, wenn die Anzahl von Parametern in den zwei- oder dreistelligen Millionenbereich wächst. Es ist aber zu erwähnen, dass es durchaus erfolgreiche Projekte zur Visualisierung von neuronalen Netze gibt[34]. Insbesondere in einer Weiterentwicklung der einfachen neuralen Netzen (Multi-Schicht-Perzeptrons), den CNN (s. 2.2.2) lassen sich *Feature Maps* von den einzelnen Knoten exrahieren, die die Interpretierbarkeit und Nachvollziehbarkeit der Neuronfunktionalitäten verbessern.

Abbildung 2.4: Handgeschriebene **5** aus dem MNIST-Datensatz.

Trainingsprozess[2]. Nachdem ein neuronales Netz über die Eingabeschicht Werte aus einer Beobachtung der Trainingsdaten erhalten hat, kann es mittels eines Prozesses, der als *feed forward*-Propagation bezeichnet wird, die Eingabewerte bis zur letzten Schicht durchpropagieren. Da neuronale Netze überlicher Weise über hoch optimierte Matrixmultiplikationen[36] implementiert werden, macht es Sinn, die dahinterliegende Mathematik in der Matrixschreibweise darzustellen. In einem Propagationsschritt von Schicht 1 (z.B. der Eingabeschicht) zu Schicht 2 sind die Zellwerte demnach definiert als $[a_1^{(1)}, a_2^{(1)}, \ldots, a_i^{(1)}]^T = \vec{a}^{(1)}$ und $[a_1^{(2)}, a_2^{(2)}, \ldots, a_i^{(2)}]^T = \vec{a}^{(2)}$. Die Gewichte von Schicht 1 zu Schicht 2 lassen sich als Matrix $\boldsymbol{W}$ darstellen, wobei der Index die Verbindung von Schicht 1 (i) zu Schicht 2 (j) darstellt. Berücksichtigt man noch den bias in Schicht 2 $[b_1^{(2)}, b_2^{(2)}, \ldots, b_i^{(2)}]^T = \vec{b}^{(2)}$, so lassen sich die Endwerte in Schicht 2 folgendermaßen berechnen:

$$\underbrace{\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1j} \\ w_{21} & w_{22} & \cdots & w_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & \cdots & w_{ij} \end{bmatrix}}_{\mathbf{W}} * \underbrace{\begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_i^{(1)} \end{bmatrix}}_{\vec{a}^{(1)}} + \underbrace{\begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ \vdots \\ b_j^{(2)} \end{bmatrix}}_{\vec{b}^{(2)}} = \underbrace{\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_j \end{bmatrix}}_{\vec{z}}$$

Bis zu diesem Punkt handelt es sich bei der Berechnung um eine für ML typische lineare Transformation, die beispielsweise in der linearen Regression verwendet wird. Da in der Realität allerdings nur selten lineare Zusammenhänge bestehen, hat sich gezeigt, dass es zusätzlich einer nicht-linearen Transformation bedarf, damit neuronale Netze Zusammenhänge besser abstrahieren können. Es wird daher zusätzlich eine sogenannte Aktivierungsfunktion auf die Endwerte angewendet. Zwar gibt es eine Reihe von geeigneten Aktivierungsfunktionen,

2 Die mathematischen Formeln und Notationen aus diesem Unterkapitel wurden übernommen aus dem Lehrmaterial über neuronale Netze von Mathematiker Grant Sanderson[35]

doch hat sich insbesondere die Sigmuid-Funktion $\sigma(x) = \frac{1}{1+e^{-x}}$ in der Literatur als nützliche Aktivierungsfunktion etabliert[37], unter anderem weil sie eine nichtlineare Transformation ist, die einen unendlich großen Wertebreich auf einen Wertebereich zwischen 0 und 1 abbildet. Eine weitere aufgrund ihrer Effektivität[38] besonders häufig verwendete Aktivierungsfunktion ist die Rectified Linear Unit (ReLU)-Stufen-Funktion: $r(x) = argmax(0, x)$. Die endgültigen Werte einer Schicht im neuronalen Netz bei der ***forward propagation*** berechnen sich unter Anwendung der ReLU-Aktivierungsfunktion folglich über

$$ReLU(W * \vec{a}^{(1)} + \vec{b}^{(2)}) = ReLU(\vec{z}) = \begin{bmatrix} max(0, z_1) \\ max(0, z_2) \\ \vdots \\ max(0, z_j) \end{bmatrix} = \vec{a}^{(2)}.$$

Da ein untrainiertes Modell zunächst mit zufälligen Parametern initiiert wird, weicht die endgültige Ausgabe in der letzten Schicht ($a^{(L)}$) höchstwahrscheinlich vom erwarteten Wert ab. Um die Diskrepanz zwischen der Erwartung und den berechneten Werten numerisch zu beschreiben, benötigt der Algorithmus während des Trainingsprozesses zu jeder Eingabe eine dazugehörige zu erwartende Ausgabe $\vec{y}$. Hat man beispielsweise eine handgeschriebene 5 als Eingabe und die Ausgabeschicht 10 künstliche Neuronen, in denen jedes einer Zahl von 0 bis 9 entspricht, wird der Vektor $\vec{y} = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]^T$ als erwarteter Zielvektor an den Algorithmus übergeben. Die quadrierte Differenz zwischen dem ermittelten Ergebnis in der letzten Schicht $a^{(L)}$ und dem tatsächlich erwarteten Vektor $\vec{y}$ wird daher für jede Beobachtung als Fehlerwert C für den ganzen Datensatz mit n Beobachtungen gemittelt[3]:

$$Kosten\ des\ Modells = \frac{1}{n} * \sum_{k\,=\,0}^{n-1} (\vec{y}_k - \widehat{\vec{y}})^2 = \frac{1}{n} * \sum_{k\,=\,0}^{n-1} C_k.$$

Das Ziel des vollständigen Trainingsprozesses ist die Minimierung dieser Kostenfunktion. Da diese eine multidimensionale Funktion mit p Paramtern darstellt, wird die Ermittlung des absoluten Minimums über die Ableitung angestrebt. Meistens ist die analytische Lösung dieses Optimierungsproblems aufgrund der Nichtlinearität und Modellkomplexität unmöglich. Daher

3 Die quadratische Kostenfunktion - auch Mean Squared Error (MSE) genannt - ist nur eine Möglichkeit der Fehlerberechnung. Dabei werden über die Quadrierung größere Abweichungen stärker bestraft als kleinere. Weitere Kostenfunktionen sind die Kreuzentropie[39] oder Huber Loss[40]

berechnet man die durchschnittlichen Kosten für jede Beobachtung im Trainingsdatensatz und passt anschließend die Parameter anhand des über partielle Ableitungen berechneten Gradienten ∇ in Richtung des Minimums an (Gradientenabstiegsverfahren). Die Berechnung der partiellen Ableitungen jeweils nach Gewicht und nach Bias, über die verschiedenen Schichten zurück bis zur Eingabeschicht erfolgt unter Anwendung der Kettenregel und wird als *Backpropagation* bezeichnet (s. hier[41] für eine detailierte mathematische Beschreibung). Die Berechnung des Fehlers für eine Beobachtung oder eines Bündels an Beobachtungen (Batch) und Anpassung der Parameter in Richtung des Minimums wird als Trainingschritt bezeichnet. Eine vollständige Anpassung aller Modellparameter unter Iteration aller Beobachtungen eines Datensatzes wird als Epoche bezeichnet. Nach Anpassung der Parameter in entgegengesetzter Richtung des Gradienten (∇) wird eine neue Epoche gestartet. Dieser Prozess wird solange iteriert, bis die Kosten zu einem bestimmten Wert konvergiert sind oder ein vorher festgelegtes Abbruchkriterium (z.B. Anzahl von Epochen) erreicht worden ist.

Da man in der letzten Schicht eines neuronalen Netzes (Ausgabeschicht) im Falle einer Kategorisierung eine aussagekräftige Metrik für die Klassifikation benötigt, wird an dieser Stelle oft die Softmax-Funktion angewandt, die alle Werte einer numerischen Liste proportional zueinander auf den Wertebereich [0,1] mit der Summe 1 abbildet. Angenommen $\vec{z}$ sei ein Vektor mit $\vec{z} = [1, 2, \ldots, K]^T$, wobei K die Anzahl an Klassen in der letzten Schicht des Netzes ist. Dann berechnen sich die entsprechenden Werte mit der Softmax-Funktion σ durch

$$\sigma(z)_i \;=\; \frac{e^{z_i}}{\sum_{j\,=\,1}^{K} e^{z_j}}.$$

Es ergibt sich eine Liste mit jeweiligen Werte pro Klasse, die als Wahrscheinlichkeit interpetiert werden können.

Hyperparameter. Über die Anzahl von Epochen beim Training hinaus gibt es zahlreiche Konfigurationsvariablen, die die Trainingsgeschwindigkeit und Genauigkeit beeinflussen können. Diese Einstellungen werden als Hyperparameter bezeichnet. Obwohl sich im Laufe der Zeit bestimmte Daumenregeln für gute Hyperparametereinstellungen herausgebildet haben, ist es trotzdem noch gängige Praxis, die beste Konfiguration für das eigene Modell experimentell druch Iteration einer Vielzahl an Kombinationen in einer Rastersuche zu ermitteln und anzupassen

(*Tuning*). Oft wird für diesen Zweck die vorbereitete Datenquelle in drei Teile separiert: Ein Trainingsteil mit dem größten Anteil (70-80% der Daten), ein Validerungsteil, dessen Daten verwendet werden um Modelle mit verschiedenen Hyperparametern miteinander zu vergleichen und ein Testteil mit Daten, der verwendet wird, um die entgültige Modellleistung zu ermitteln. Für eine Übersicht häufig genutzter Hyperparameter, siehe Tabelle 2.1. Da sich diese Arbeit mit dem Vergleich bereits trainierter Modelle befasst, wird auf eine detaillierte Beschreibung der Verfahren zur Optimierung des Trainingsprozesses verzichtet.

Tabelle 2.1: Hyperparameter von neuronalen Netzen im Überblick

Hyperparameter	Beschreibung	Beispielwerte
Anzahl an Trainingsepochen	Eine Epoche entspricht einer Iteration des gesamten Datensatzes beim Training	10-100
Lernrate	Ein Faktor für die Schrittgröße des Gradientenabstiegs (typischer Weise als η bezeichnet)	0.001, 0.0001
Batch-Größe	Verarbeitung mehrerer Datensätze pro Trainingsschritt	8, 16, 32, 64, 128[42]
Anzahl Schichten	Anzahl (versteckter) Zwischenschichten	2-1000
Aktivierungsfunktion	Transformation gewichteter Eingänge und Bias in eine Neuronenschicht	ReLU, Sigmoid, Tanh
Regularisierung	Bestrafung von Modellkomplexität zur Vermeidung von Überanpassung (overfit)	L1[43], L2[43], Dropout[44]
Optimierer	Verbesserung des Gradientenabstiegsverfahrens und des Lernprozesses	SGD[45], Adam[46]

2.2.2 Convolutional Neural Networks

CNN sind eine Erweiterung neuronaler Netze um sogenannte Faltungsschichten (convolutional Layers) in Kombination mit Schichten zur Dimensionsreduktion (Pooling Layers). Die Faltung in der Bildverarbeitung beschreibt den Prozess, ein Bild über eine kleine Kernel- oder Faltungsmatrix anzupassen, in dem diese sukzessive über eine Pixelgruppe geschoben und durch elementweise Multiplikation der Kernel-Werte mit den entsprechenden Bildpixeln und Aufsummation eine Filteroperation erfolgt. Je nach Kernel können solche Filter Bilder glätten, schärfen oder bestimmte Arten von Kanten identifizieren (s. Abbildung 2.5). Letztere können daher auch als Feature Detektoren bezeichnet werden, beispielsweise wenn sie dazu eingesetzt werden, vertikale Kanten zu finden.

Abbildung 2.5: Faltungsprozess mit einem Gauss-Kernel[47].

Da in CNNs überlicherweise Farbbilder verwendet werden, die drei Farbkanäle - Rot-Grün-Blau (RGB) - haben, ist das Eingabebild eine drei-dimensionale Matrix (**3-D-Tensor**). Dementsprechend wird jeder Kanal des Farbbildes mit einem einzigartigen Kernel gefaltet. Die Werte in den Kernels entsprechen in CNNs analog zu den herkömmlichen neuronalen Netzen den Gewichten, die es zu trainieren gilt. Die resultierenden berechneten Pixelwerte der gefalteten Bilder werden, wie auch im herkömmlichen Netz, zur Aufsummierung aller eingehenden Signale gemeinsam einschließlich des Bias aufsummiert. Eine weitere Gemeinsamkeit mit Netzen ohne Faltung ist die Transformation der Werte in den Feature Maps mit einer Aktivierungsfunktion wie ReLU, um die Nichtlinearität der Realität in das Model einzubringen. Würde man dies nicht tun, so würden tiefe CNNs mit der Zeit zu äquivalenten Faltungsschichten konvergieren[34]. In der Folge ergibt sich aus

der Summe der Kanäle in der vorhergehenden Schicht ein neues Farbbild pro künstlichem Neuron in der Faltungsschicht. Die neu enstandenen Bilder werden auch als *Feature Map* oder *Activation Map* bezeichnet.

Im Gegensatz zu den numerischen Werten in herkömmlichen neuronalen Netzen ermöglichen die Feature Maps über die Darstellung von deren Pixelwerten, die Funktionsweise der CNN-Netze zu visualisieren und zu verstehen. Die Faltungsschichten können als Abstraktionsebenen gesehen werden, die zur Erkennung von Mustern dienen. Umso tiefer die Schicht im neuronalen Netz, desto abstrakter werden die von den Feature Maps repräsentierten Muster. Während die Feature Maps in einer frühen Schicht eher Kanten und Ecken erkennen, tragen solche in tieferen Ebenen Gemeinsamkeiten der eingehenden Informationen zusammen, d.h. sie suchen nach zusammenhängenden Objekten wie Rechtecken, Kreisen oder in den letzten Schichten ganzen Objekten wie Gesichtern[48].

In CNN-Architekturen korrelieren typischer Weise die Anzahl der Feature Maps pro Schicht mit der Netztiefe und erinnert somit an eine pyramidenähnliche Struktur. Die Ursache liegt in der ansteigenden Anzahl an Kombinationsmöglichkeiten aus den simplen Formen und Texturen aus den anfänglichen Schichten zu immer abstrakteren Feature Maps. Es entsteht eine hierarchische Form der Propagierung durch CNNs. Um dem damit einhergehenden Anstieg des Rechenaufwands entgegenzuwirken werden Faltungsschichten üblicherweise zusammen mit Pooling-Schichten in CNN-Architekturen verwendet. Beim Pooling wird ebenfalls ein Kernel verwendet, dessen Aufgabe es ist, beim Entlanggleiten am Bild die Pixelzahl in den Feature Maps über das Ersetzen von Pixelgruppen durch repräsentative Statistiken zu reduzieren. Zwei häufig verwendete Varianten des Poolings sind der Einsatz von Maximum (MaxPooling) und Mittelwert(AveragePooling)[49]. Über den Effekt der Reduktion der Komplexität hinaus helfen Pooling-Schichten dabei, CNNs resilienter gegen Verschiebung zu machen (Translationsinvarianz).

Sobald die Feature Maps abstrakt genug sind, folgt in CNN-Architekturen üblicherweise am Ende eine oder mehrere vollverbundene gewöhnliche Neuronenschicht ohne Faltung. Deren Aufgabe ist es, aus den gegebenen Features einen Klassifikator oder Detektor zu bilden, beispielsweise unter Anwendung der Softmax-Funktion (s. oben).

Zusätzlich zu den Hyperparamtern in herkömmlichen neuronalen Netzen (s. Tabelle 2.1) können für CNNs weitere spezielle Hyperparamter eingestellt werden. Dazu zählen die Größe und Form des Kernels, der Verschiebungsabstand (Stride) und das Padding (hinzufügen von Pixeln am Bildschirmrand). Diese Einstellungen

können bei Bedarf sowohl für jede Faltungsschicht als auch jede Poolingschicht unabhängig konfiguriert werden.

CNNs bieten einige entscheidende Vorteile gegenüber mehrschichtigen Perzeptron- Architekturen. Die Verwendung der Filterkernel auf alle Pixel eines Kanals reduziert deutlich die Anzahl benötigter Trainingsparameter. Anstatt für jeden Pixel ein Neuron mit einem Gewicht zu verbinden, kann ein 3x3-Kernel einen ganzen Bildkanal mit der Folgeschicht verbinden. Achtet man auf eine vielfältige Datenbasis, so kann es außerdem lernen, mit Translation, Rotation und Skalierung von Objekten umzugehen. Zuletzt ist der Aufbau von CNN-Architekturen für Menschen durch deren hierarchische Struktur intuitiver nachzuvollziehen und damit leichter zu modellieren.

Aufgrund der beschriebenen Eigenschaften haben sich CNNs besonders im Bereich der Bild- und Audioverarbeitung als Wegweiser im Bereich Klassifizierung- und Objekterkennung herausgestellt. Wie auch in der Anwendung von neuronalen Netzen insgesamt gilt es bei CNNs, unter Anwendung von verschiedenen Architekturkonfigurationen und der Anwendung spezieller Bildverarbeitungsmethoden explorativ die bestmögliche Kombination zu ermitteln.

Dabei gilt die Abwägung von Ressourcenlast und Trainingsgeschwindigkeit gegen Genauigkeit. Betrachtet man beispielsweise das oben genannte MNIST-Beispiel (s. Abbildung 2.4) so wäre mit einer relativ kleinen mehrschichtigen Perzeptron-Netzarchitektur (784 - 10 - 10 - 10) bereits eine Menge von 8070 verstellbaren Parametern erreicht. Benutzt man Farbbilder mit drei Kanälen statt Graustufenbildern, so wird die Anzahl an Parametern bereits in der Eingangsschicht um den Faktor 3 vergrößert. Für den Anwendungsfall bedeutet dies den dreifachen Rechenaufwand für die Inferenz. Erhöht man die Anzahl an Schichten, so wie die Anzahl Neuronen pro Schicht, so kann das Netz zwar feinere Unterschiede erkennen, aber es erhöht sich auch exponentiell die Ressourcenlast und birgt zudem das Riskio des *Overfittings*, einer Überangepasstheit des statistischen Modells an die Daten. Im letzteren Falle würde das Modell bei Eingaben, die sich auch nur leicht von den Trainingsdaten unterscheiden, bereits starke Einbußen in der Genauigkeit zeigen. Aufgrund dieser Beschaffenheit ist es insbesondere für die Anwendung in CV, wo Kameras in speicher- und verarbeitungsschwache Systeme integriert sind, wichtig, der Modellgröße und Parameterzahl eine gesonderte Rolle bei der Auswahl des passenden ML-Modells einzuräumen.

Über die Berechnung innerhalb der Netzwerkarchitektur hinaus muss für die Rechenöknomoie auch die Anpassung der Daten an die Eingabeschicht des Netzes

berücksichtigt werden. Bilddaten müssen etwa manchmal zunächst in ein anderes Bildformat konvertiert oder auf eine vordefinierte Auflösung skaliert werden.

Die Rastersuche nach einer guten Hyperparameterkonfiguration stellt, abgesehen von der nötigen Expertise und Infrastruktur, einen nicht unwesentlichen Arbeitsaufwand dar, der viele Anwender zögern lässt, KI-Technologien in ihre Systeme zu integrieren. Es wäre daher von Vorteil, das bis dato angesammelte verfügbare Wissen und die Erkenntnisse zu leistungsfähigen Netzwerkarchitekturen in eine anwendungsfreundlichere Abstraktionsebene zu transportieren und solche Modelle im Vergleich zueinander zu stellen. Dafür bieten sich die zahlreichen vortrainierten Netze, die inzwischen von KI-Forschungsunternehmen in speziellen Benutzerschnittstellen zur Verfügung gestellt werden, an. Neben dem offensichtlichen finanziellen Vorteil erübrigt sich eine fachspezifische Kenntnis über die Funktionsweise des Netzwerktrainings und der oft nicht trivialen anwendungsspezifischen Aquise von Datenmaterial für das Training. Folglich ließe sich der Aufwand zum Einsatz von KI in Industrie und Alltag deutlich reduzieren und eventuelle Vorbehalte abbauen. In der nachfolgenden Sektion werden die Funktionsweisen der gängisten frei verfügbaren Netzarchitekturen zur Objekterkennung im Detail erläutert.

2.2.3 Funktionsweise gängiger Netzwerkarchitekturen

YOLO. YOLO ist eine leistungsstarke und frei verfügbare Architektur für Objekterkennungssysteme, die sich durch ihre Fähigkeit sowohl in Erkennungsgeschwindigkeit und Generalisierbarkeit, als auch Genauigkeit auszeichnet. Es wird in aktueller Literatur als *state of the Art* Technologie für Objektdetektion angesehen[50,51,52] und existiert derzeit in neun verschiedenen Versionen[53]. Die erste Version dieser Architektur wurde von Joseph Redmon et al.[54] vorgestellt. Traditionell haben Objekterkennungssysteme wie rekurrente CNNs[55] und seine Varianten den Bildraum in mehreren Iterationen untersucht, was sie zwar genau, aber langsam machte. Das Ungewöhnliche am Konzept der YOLO-Architektur war die Betrachtung von Objekterkennung als Regressionsproblem anstelle einer kategorialen Klassifikation und der einmaligen Propagierung durch das Netz (daher der Name *You Only Look Once*).

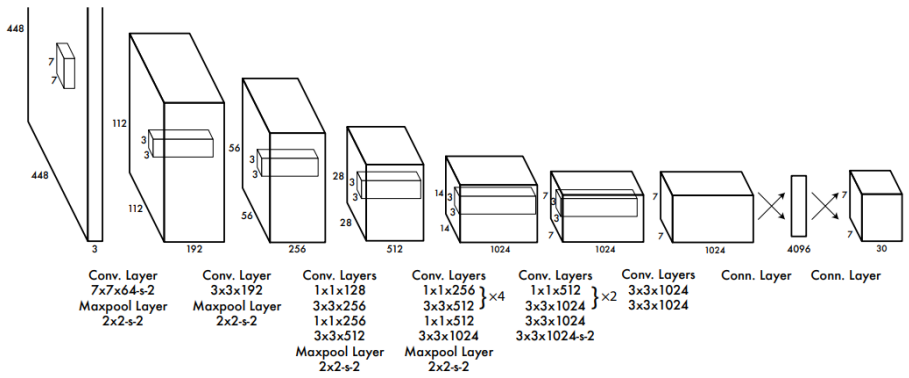


Abbildung 2.6: YOLO v1-Architektur aus dem Veröffentlichungsartikel[54].

In seiner Ursprungsversion erfordert das Eingangsbild eine Auflösung von 448 × 448 Pixeln und hat insgesamt 24 Faltungsschichten und 4 MaxPooling-Schichten, gefolgt von zwei vollständig verbundenen Netzwerkschichten (s. Abbildung 2.6). Speziell zur Reduktion der Schichtgröße werden Faltungen mit 1×1 im Wechsel mit 3×3 großen Kernels vorgenommen. Ausgenommen von der letzten Schicht wird ausschließlich die ReLU-Aktivierungsfunktion verwendet.

Zunächst wird das Bild auf ein quadratisches Format skaliert und in ein Raster von $S \times S$ Zellen aufgeteilt. Jede Zelle hält fünf Informationen über B Bounding Boxes vor:

$$\begin{aligned} \text{Zelle 1: } & \left[x_1,\ y_1,\ \sqrt{w_1},\ \sqrt{h_1},\ C_1\right] \\ \text{Zelle 2: } & \left[x_2,\ y_2,\ \sqrt{w_2},\ \sqrt{h_2},\ C_2\right] \\ & \vdots \\ \text{Zelle B: } & \left[x_B,\ y_B,\ \sqrt{w_B},\ \sqrt{h_B},\ C_B\right] \end{aligned}$$

Als Mittelpunkt der Bounding Box im Verhältnis zur Zelle sind x und y definiert. Liegt dieser beispielsweise horizontal zentriert und vertikal im unteren Drittel, so wären $x = 0.5$ und $y = 0.66$. w und h repräsentieren die relative Größe der Bounding Box im Verhältnis zum geamten Bild. C gibt einen Konfidenzwert im Wertebereich [0,1] darüber aus, ob diese Bounding Box tatsächlich ein Objekt ist. Zusätzlich zu den Bounding Box Informationen hat jede Zelle Informationen über die prognostizierten Klassenwahrscheinlichkeiten $[p(c_1),\ p(c_2), \ldots, p(c_n)]$ mit $n =$ Anzahl an Klassen. Alle Zellinformationen werden gemeinsam in einem Vektor der Größe $B \times 5 + n$ gespeichert, in dem jedes Element je mit einem künstlichen

Neuron in der letzten Schicht der Netzarchitektur korrespondiert. Folglich wird nach der Eingabe des Bildes die CNN-Architektur durchlaufen, um diese Paramter in der letzten Schicht zu schätzen. Die Autoren haben in der Erstpublikation für das Modell[54] die Hyperparamter $S = 7, B = 2$ und $n = 20$ gewählt, so dass die finale Vorhersage mit einem $7 \times 7 \times 30$ großen Tensor mit je 30 Informationen zu den 49 Zellen repräsentiert wird. Da im Trainingsprozess die realen Bounding Boxes bekannt sind, werden diejenigen Gitterzellen, die den Mittelpunkt der Bounding Box beinhalten, mit den tatsächlichen Bounding Box Parametern, einer Klassenwahrscheinlichtkeit von 1 und einem Konfidenzwert entsprechend der IoU (s. Abbildung 2.2) der tatsächlichen und der vorhergesagten Bounding Box versehen. So kann die Kostenfunktion den Fehler in dieser Gitterzelle zwischen dem vorhergesagten und der tatsächlichen Bounding Box berechnen und die Parameter über Backpropagation anpassen. Die Kostenfunktion setzt sich zusammen aus mehreren Teilen: $L = L_{Klasse} + L_{BoundingBoxes}$[4].

Da das Modell B Boxes pro Zelle vorhersagt, werden bei der Inferenz diese zunächst anhand eines Grenzwerts IoU $> .50$ gefiltert. Anschließend wird die Bounding Box als detektiertes Objekt ausgegeben, die den höchsten Konfidenzwert hat. Diese Methode wird als *Non max suppression* bezeichnet[56].

Obwohl bereits die erste Version der YOLO-Architektur einen neuen Standard für Objektdetektion gesetzt hat, unterliegt sie einigen Mängeln. Jede Gitterzelle kann nur maximal eine Klasse als Ausgabe produzieren. Das bedeutet in der Praxis, dass mehrere Objekte (z.B. zwei Menschen oder ein Hund und ein Fahrrad) innerhalb einer Zelle nicht erkannt werden. Zudem nimmt das Modell an, dass sich alle Klassen gegenseitig ausschließen. In der Realität kann ein Objekt allerdings mehrere Bezeichnungen haben (z.B. *Hund* und *Tier*). Außerdem geben die Autoren an, dass die Vorhersage bei Bildern mit ungewöhnlichen Seitenverhältnissen schwerer fällt und die Größe der fehldetektierten Bounding Boxes beim Training nicht ins Gewicht fällt, weshalb die Parameter für Höhe und Breite der Bounding Boxes innerhalb der Kostenfunktion quadriert werden sollten.

In den Folgeversionen wurden diese Probleme schrittweise angepasst. Mit der Einführung neuer Features zur Optimierung von Teilen des Prozesses, sowie Verbesserungen der Architektur wurde die Genauigkeit und Geschwindigkeit des Modells fortschreitend verbessert und einige Limitationen der ersten Version aufgehoben (Beispiel: YOLO v3 ermöglicht die mehrfache Klassifizierung eines Objekts mit zwei unterschiedlichen Klassen). In Appendix A werden die Versionsunterschiede der YOLO-Modelle in einer Übersicht zusammengefasst.

4 für eine detailierte mathematische Darstellung, s.[54]

Obwohl es nahe liegen würde, ausschließlich die neuesten Modelle für die Evaluierung zu berücksichtigen, gibt es auf der praktischen Seite aufgrund des Aussetzens der Unterstützung alter Systeme immer öfter Versionskonflikte zwischen den angeforderten Modulversionen der Architektur und den unterstützen Modelversionen für rudimentäre Systeme, insbesondere wenn die Prozessorarchitektur von typischen Normen abweicht. Aus diesem Grund wird zum Einen auf die Anwendung der experimentellen Version YOLOv9 verzichtet und zum anderen wird zusätzlich zu YOLOv8 auch YOLOv5 als Stellvertreter für eine ältere Version inkludiert, da sich diese Variante in vorab-Implementierungsversuchen zu dieser Studie als stabile Alternative herausgestellt hat.

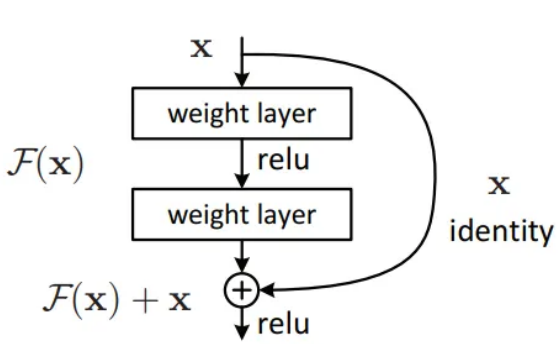
Resnet. Ein Kerngedanke der ResNet-Architektur ist der Befund, dass die Implementierung von immer tieferen Netzwerkarchitekturen nicht immer bessere Resultate bringt. Der Grund dafür liegt in dem typischen Problem für tiefe Architekturen, dass über die vielen aneinandergeketteten Ableitungen der Gradient in der Backpropagation (s. Kapitel 2.2.1) entweder sehr klein (*vanishing Gradient*) oder zu groß (*exploding Gradient*) wird. Um dieses Problem zu addressieren hat Microsoft im Jahr 2015 die ResNet-Architektur publiziert.[57].

Im Wesentlichen wird ein gewöhnliches CNN-Backbone verwendet. Die Besonderheit besteht in der Einführung von *Skip Connections* bzw. *Shortcut Connections*, die zusätzlich zu den Faltungsschichten eingebaut werden und Informationen von einer deutlich früheren Schicht zu einer späteren Schicht als Zusatzinformation transportieren. Eine von einer Skip Connection umschlossene Einheit wird als *Residual Building Block* bezeichnet. Eine typische ResNet-Architektur besteht aus vielen solcher aneinander gereihten Blöcken und konnte durch seine Tiefe größere Genauigkeit erzielen (s. Abbildung 2.9).

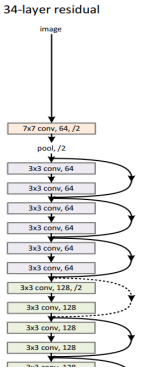
Die Implementierung der eingehenden Daten jeder Schicht wird so umgebaut, dass zusätzliche Output-Daten einer früheren Schicht zur in die Eingabe einer späteren Schicht über eine elementweise Addition hinzugefügt werden können:

$$y = F(x, \{W_i\}) + x$$

Obwohl die Skip Connections tiefere Netze ermöglichen, erhöhen sie die Komplexität der Modelle und, damit verbunden, den Rechenaufwand drastisch. Damit zusammenhängend ist bei tiefen Netzwerken eine Feinjustierung der optimalen Hyperparameter zeit- und kostenintensiv. Es hat sich außerdem gezeigt, dass ab einer bestimmten Tiefe kaum noch Mehrwert für die Prognosegeschwindigkeit erzielt werden kann[58].



Eine Residual Block - Einheit.



Ausschnitt der ResNet34-Architektur.

Abbildung 2.9: Grundgerüst für ResNet - Architekturen[57].

Es gibt vortrainierte Modelle für mehrere verschiedene Architekturen, die ein ResNet-Backbone verwenden und darauf dann zusätzliche Objekterkennungsalgorithmen implementieren. Die Faster-R CNN Architekturen[59] nutzen zwei hintereinander geschaltete Netzwerke: Das Region Proposal Network (RPN) generiert zunächst Vorschläge für Objektkoordinaten, welche dann als Teilbild-Eingaben zur Klassifikation in ein Regionales CNN übergeben werden. Im Konstrast dazu wurde von Facebook AI Research das RetinaNet[60] vorgestellt, das ebenfalls ein ResNet-Backbone verwendet, aber nur eine Einstufen-Detektion verwendet. Das Modell versucht durch ein Feature Pyramid Network (FPN) mit unterschiedlich groß skalierten Feature Maps Größenunterschieden in Bildern zu begegnen. Zur Untersuchung der Bedenken bezüglich der Anwendbarkeit von Architekturen mit ResNet-Backbone auf weniger performanten Systemen werden für diese Studie verschiedene Varianten der Faster R-CNN und RetinaNet in die Evaluation integriert.

MobileNet und EfficientDet. MobileNet-Architekturen wurden mit dem Hintergrund für den Einsatz in kleinen Geräten und Mobilgeräten entwickelt und sind deshalb besonders interessant für diese Studie. Laut den Entwicklern der Architektur[61] ist das Netzwerk quantisierungsfreundlich. Quantisierung beschreibt im Allgemeinen den Prozess der Diskretisierung oder Reduzierung einer Datenstruktur. Ein einfaches Beispiel dazu ist die Konvertierung von speicherintensiven Fließkommazahlen (float) zu ganzen Zahlen (integer).

Mobile Network (MobileNet) verwendet in seinen Faltungsschichten eine abgewandelte Form der Faltung. Zum Einen werden *Depthwise Convolutions* verwendet, um Eingabekanäle Schichtweise zu glätten, anstelle eines mehrdimensionalen Kernels. Anschließend werden mit *Pointwise Convolutions* 1×1 Faltungen verwendet, um die Kanäle der Depthwise Convolution zu kombinieren.

Dieses Verfahren dient dem Zweck, parallele Rechenlast des Systems zu reduzieren. Es bietet darüber hinaus zwei Hyperparameter zu einer noch besseren Feinjustierung für die Anwendung im speziellen Endgerät an:

(a) Der *Width Multiplier* α im Wertebereich [0,1] ist ein Skalierungsfaktor, der die Anzahl der Kernel pro Schicht skaliert.

(b) Der *Resolution Multiplier* ρ skaliert die Dimension des Eingabebildes.

Efficient Detection (EfficientDet)-Netze sind eine Weiterentwicklung der MobileNet-Architekturen, die ebenfalls Depthwise und Pointwise Convolutions nutzen. Zusätzlich nutzen EfficientDet-Architekturen *Compound Scaling*, eine Methode zur gleichmäßigen Herunterskalierung von neuronalen Netzen bezüglich Tiefe, Breite und Auflösung der Eingangsschicht[62] und BiFPN (Bidirectional Feature Pyramid Networks), einem Algorithmus, der darauf abzielt, verschieden skalierte Feature Maps für eine bessere Inferenz bei unterschiedlich groß skalierten Objekten zu erzeugen. In den letzten Schichten werden über eine Regression die Parameter der Objekt-Bounding Boxes geschätzt. Laut einer aktuellen Studie[63] bieten EfficientDet-Netze im Vergleich zu MobileNet-Architekturen einen Gewinn an Genauigkeit im Austausch für Rechenleistung und Größe.

Obwohl Recheneffizienz wichtig ist, bleibt zu untersuchen, inwiefern vortrainierte Modelle mit MobileNet- und EfficientDet- Architekturen im Vergleich zu anderen Modellen in dieser Studie einen lohnenswerten Kompromiss zu Geschwindigkeit und Genauigkeit finden können.

Weitere vortrainierte Netze. Neben den aufgelisteten Netzen bietet die TF-Benutzerschnittstelle noch weitere vortrainierte Modelle an, die zur Vollständigkeit als Vergleichswerte in die Evaluation eingegliedert werden sollten. Darunter sind Netze mit CenterNet-Architektur mit einer Hourglass-Architektur[64] als CNN-Backbone, die sich auf die Identifizierung von deren zentralen Punkten (Zentroiden) von Objekten im Kontrast zu Bounding Boxes konzentriert. Hourglass-Architekturen fügen zusätzlich zu herunterskalierenden Schichten (Downsampling) wie den Pooling-Schichten in Richtung der Ausgabeschicht hochskalierende Schichten hinzu, um dem Modell durch zusätzliche räumliche Auflösung mehr Möglichkeiten zur Kombination der Feature Maps zu geben(s. Abbildung 2.10).

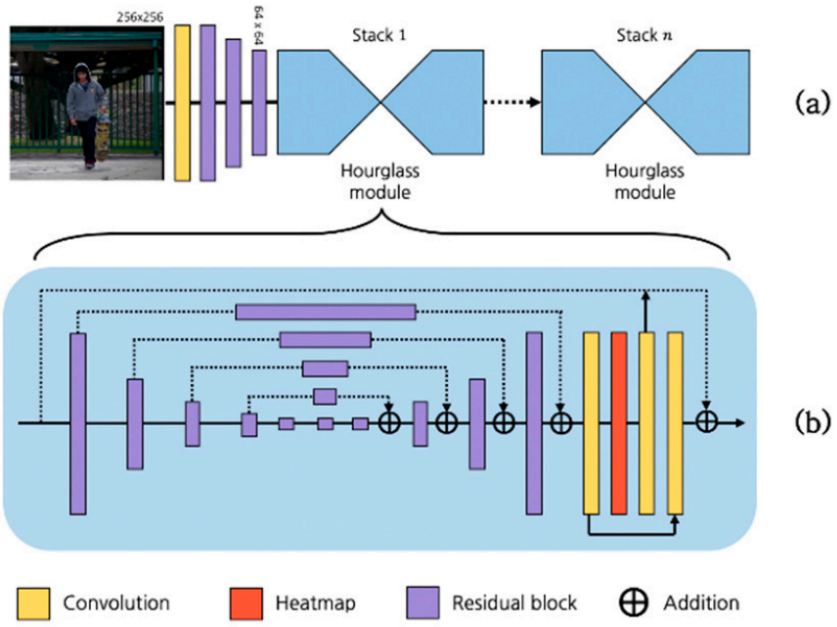


Abbildung 2.10: Hourglass-Architektur aus dem Veröffentlichungsartikel[64].

Eine vollständige Übersicht aller Netze der TensorFlow-Benutzerschnittstelle in dieser Studie befindet sich in Kapitel 3.2.4.

3 Untersuchungsmethodik und Implementierung

In diesem Kapitel wird die methodische Vorgehensweise zur Bearbeitung der gesetzten Forschungsschwerpunkte aus Kapitel 1 erläutert. Dies beinhaltet zunächst das gesamte Studienkonzept und Forschungsaufbau und anschließend eine Erläuterung zu dessen Umsetzung mit Hilfe der in Kapitel 2 beschriebenen Technologien. Zuletzt erfolgt eine technische Beschreibung der Einstellungen und technischer Hilfsmittel.

3.1 Studienkonzept

3.1.1 Überblick

Nach einer umfassenden Recherche der Möglichkeiten über die Möglichkeiten der Anwendung vorhandener vortrainierter neuronaler Netze soll zur Feststellung der Anpassungsfähigkeit und Tauglichkeit für den Praxisbetrieb von Personenerkennung eine Evaluation dieser Technologien hinsichtlich Ressourcenbedarf, Geschwindigkeit und Genauigkeit stattfinden. Da es zunächst um die Identifizierung besonders leistungsfähiger Netze und deren Generalisierbarkeit im Allgemeinen geht, erfolgt diese Analyse auf einem vertikal hoch skalierten Hardwaresystem. Durch die gegebene Rechenleistung kann eine größere Menge an Netzen untersucht werden und zusätzliche Iterationen mit Bildaugmentierungen durchgeführt werden, die Störquellen simulieren sollen. Nach dieser Analyse kann aufgrund der Zwischenbilanz eine Auswahl von Modellen erfolgen, die in einem weiteren Test auf einem herunterskalierten Low-End-System gegeneinander abgewogen werden, um zu ermitteln, inwiefern sie auch in Umgebungen mit eingeschränkten Hardwareressourcen noch annehmbare Ergebnisse liefern. Da die Evaluationsdaten auf Grundlage eines frei verfügbaren gelabelten Datensatzes mit unabhängigen Bildern erhoben werden, wird in einem letzten Schritt eine Laborstudie zur Überprüfung der Anwendbarkeit auf einen konkreten Kontext, der Personenzählung in einem Raum mit Top-Down-Perspektive und Weitwinkelkamera, durchgeführt. Parallel zur Durchführung der Studie erfolgt eine subjektive Beurteilung der Benutzerschnittstellen.

3.1.2 Konkrete Vorgehensweise

Vorbereitung: Auswahl geeigneter vortrainierter Modelle. Es wird nach aktuellem Stand der Wissenschaft eine Auswahl an vortrainierten Netzen mit der Funktionalität der Objektdetektierung vorgenommen. Unter dem Aspekt der Untersuchung von praxisnaher Umsetzbarkeit von KI-Anwendungen werden insbesondere Modelle in die Evaluation aufgenommen, die (a) direkt über eine API die Anwendung vortrainierter Netze erlauben und (b) zur freien Verfügung bereit gestellt werden. Weitere Auswahlkriterien umfassen die berichteten Ergebnisse in vorausgegangen Studien und einer besonderen Eigung für die Anwendung in Geräten mit wenig Hardwareressourcen. Parallel dazu erfolgt die Identifizierung eines geeigneten frei verfügbaren Evaluierungsdatensatzes mit bereits zugeordneten Bounding Boxes zu Objekten im Bild. Darüber hinaus ist es notwendig, dass die Klasse *Person* in Abgrenzung zu anderen Kategorien in ausreichend großer Zahl gegeben und korrekt bezeichnet ist. Sofern eine Auswahl an passenden Bilddaten vorliegt, wird zusätzlich eine händische Selektion vorgenommen, um fehlkategorisierte Objektrahmen zu entfernen und eine gleichmäßige Anzahl von Umgebungskontexten zu gewährleisten. Unter der Abwägung von Generalisierbarkeit und Kontrolle von Versuchsbedingungen soll auf diese Art und Weise eine gleichmäßige Verteilung verschiedener Bildtypen gewährleistet werden - darunter Innenräume, Außenräume, Altersgruppen, Entfernung, Aktivität und Personenzahl. Nach der Vorbereitungsergebnisse wird die Experimentalphase mit drei verschiedenen Experimenten eingeleitet.

Experiment 1: Vergleich neuronaler Netze. Eine Beschreibung der übergeordneten Netzwerkarchitekturen findet sich in Kapitel 2 wieder. Zu jeder Architektur existieren mehrere Versionen von vortrainierten Netzwerken, die jeweils seperat in die Evaluation eingehen. Eine vollständige Übersicht über alle Modelle findet sich in Appendix D. In der Evaluationsphase wird jedes Modell mit den vortrainierten Parametern geladen und soll anschließend alle gelabelten Personen iterativ Bild für Bild detektieren. Zusätzlich zur Detektion der Standardbilder werden diese noch augmentiert. Diese Vorgehensweise hat zwei Vorteile: Erstens kann die Größe des Evaluationsdatensatzes angehoben werden ohne zusätzliche Daten zu aquirieren und zweitens kann über eine Augmentierung die Anpassungsfähigkeit von Netzen gegenüber Störungen im Bild beurteilt werden. Die vorgenommenen Augmentierungen werden in Kapitel 3.2.5 detailiert

beschrieben. Die Evaluierung der Ergebnisse dieser Inferenzen erfolgt anhand der in Kapitel **??** beschriebenen Metriken zur Auswertung der Modellgüte.

Experiment 2: Überprüfung von Störfaktoren über Augmentierung. Sobald mit dem High-End System eine Auswahl an leistungseffizienten Modellen selektiert worden ist, kann ein *proof of concept*-Experiment über die Skalierbarkeit der Modelle auf ein kleineres Endgerät vorgenommen werden. Um die Vergleichbarkeit zwischen den Laborumgebungen zu gewährleisten, werden die im vorhergehenden Schritt 3.1.2 verwendeten Bilder und Netzeinstellungen für den Vergleich mit dem Low-End System übernommen. Darüber hinaus wird die Kompatibilität der High-Level KI-API mit einem rudimentäreren Betriebssystem beiläufig für eine Beurteilung dokumentiert.

Experiment 3: Aufzeichnung und Evalierung der Leistung in praxisnahem Kontext. Im Anschluss an die Untersuchung mit zufälligen Bilddaten aus einer frei Verfügbaren Quelle aus dem Internet folgt eine auf einen konkreten Anwendungsfall zugeschnittene Untersuchung. Es soll die Modellleistung in der Anwendung auf Videomaterial in einem simulierten Bürokontext untersucht werden. Für den Laboraufbau führen zwei Versuchspersonen bürotypische Bewegungen (Maus- und Tastaturbedienung, Raum verlassen/betreten, Dehnübungen) während eine zentral an der Decke montierte Kamera mit einem Weitwinkelobjektiv Videodaten mit einer Bildrate von 30 Frames Per Second (FPS) aufzeichnet. Die Aufnahme umfasst sowohl statische als auch dynamische Bewegungsniveaus. Eine Skizze des räumlichen Aufbaus ist in Abbildung 3.1 dargestellt.

3.2 Technische Umsetzung

Diese Sektion behandelt technische Hilfsmittel und Ressourcen zur Umsetzung der Forschungsziele.

3.2.1 Eingesetzte Geräte

Computerhardware. In den folgenden Tabellen sind die Spezifikationen der verwendeten Computersysteme aufgeteilt nach Leistungskapazitäten aufgelistet:

Abbildung 3.1: KI-generierte Skizze des Versuchsaufbaus

Tabelle 3.1: CPU-Details

	High-End	Low-End
Modell	Intel Core i9-10900X	ARM Cortex-A53
Architektur	x86_64	ARMv8
Betriebsmodus	64-bit	64-bit
Anzahl Kerne	10	4
Threads pro Kern	2	1
Taktfrequenz pro Kern	3,70 GHz	1,20 GHz
Cache	19,25 MB	512 KB
Hersteller	Intel	Raspberry Pi Foundation

Tabelle 3.2: GPU-Details

	High-End	Low-End
Modellbezeichnung	NVIDIA RTX A5000	VideoCore IV
Grafikpeicher	24 GB	-
max. Leistung	230 W	3,7 W
Hersteller	Nvidia	Raspberry Pi Foundation

Tabelle 3.3: RAM-Details

	High-End	Low-End
Typ	DDR4	LPDDR2-SDRAM
Speicher	2x 32 GB	1,0 GB
Hersteller	Micron	Raspberry Pi Foundation

Kamera. Für die Aufnahme im Laborraum wird eine Kamera von der Firma Conceptronic vom Modell *AMDIS08B* verwendet. Die Kamera verfügt über eine Auflösung von 3840 x 2160 Pixeln (8.3 Megapixel), Autofokus-Feature, eine 120° breite Linse und zeichnet Videos mit einer Auflösung Bildrate von 30 Bildern pro Sekunde auf. Die Daten werden über eine USB 2.0-Schnittstelle übertragen. Außerdem verfügt die Kamera über zwei Mikrophone zur Stereo-Aufzeichnung von Geräuschen.

3.2.2 Eingesetzte Software und Programmiertools

Für alle Implementierungen im Rahmen dieser Arbeit wurde die Programmiersprache Python[65] (Version 3.10.12) verwendet. Darunter fällt der Import einer Anzahl von bereits effizient implementierter Algorithmen in Form von Modulen und Skriptbibliotheken. In Tabelle 3.4 sind die entsprechenden importierten Abhängigkeiten in einer Übersicht aufgelistet. Das operative System für die Anwendung für das High-End System ist *Ubuntu* Version 22.04.1. Das Low-End-System verfügt über das operative System *Debian* Version 6.6.20-1. Alle Prozesse auf beiden Systemen wurden innerhalb einer gemeinsamen virtuellen Umgebung ausgeführt.

Tabelle 3.4: Übersicht der verwendeten Python-Module

Modul	Technische Bezeichnung	Referenz	Version	Beschreibung
Opencv	opencv-python	[66]	4.9.0.80	Verarbeitung von Bildern
PIL	pillow	[67]	10.3.0	Verarbeitung von Bildern
cvzone	cvzone	[68]	1.6.1	Verarbeitung von Videos
Matplotlib	matplotlib	[69]	3.8.4	Visualisierung von Diagrammen
Numpy	numpy	[70]	1.26.4	Effiziente Berechnung von Listen- und Matrixoperationen
Pandas	pandas	[71]	2.2.1	Verarbeitung von strukturierten Daten im Tabellenformat
Tensorflow	tensorflow	[72]	2.15.1	Framework zur Anwendung verschiedener Machine Learning Algorithmen
Keras	tf_keras	[73]	2.15.1	Modul zur Implementierung von ML-Algorithmen; Tensorflow backend
Ultralytics	ultralytics	[74]	8.1.45	Framework zur Anwendung verschiedener Machine Learning Algorithmen
Torch	torch	[75]	2.2.2	Modul zur Implementierung von ML-Algorithmen; Ultralytics backend
Torchvision	torchvision	[75]	0.17.2	Modul zur Implementierung von CV-ML-Algorithmen; Ultralytics backend
PyCocoTools	pycocotools	[76]	2.0.7	Import von Bildern und Annotationen von Bildern im COCO-Format

3.2.3 Erhebung und Beschreibung der Daten

Bilddaten aus öffentlich zugänglichen Quellen. Für die Untersuchung wird eine Teilmenge des COCO-Datensatzes[77] aus dem Jahr 2017 verwendet. Die Wahl des Datensatzes fällt aus mehreren Gründen auf diese Datenquelle. Erstens ist sie für Forschungszwecke an keine Kosten gebunden und kann frei abgerufen werden. Zweitens ist bei der Aufbereitung von Datenquellen für ML-Zwecke, insbesondere bei Objektdetektion und Segmentierung die Anotation (labeling) ein sehr zeitintensiver Prozess. Der COCO-Datensatz bietet zusammen mit den Bilddaten auch die entsprechend kategorisierten Bildkoordinaten für Objekte in über 80 Kategorien, sodass dem/der AnwenderIn dieser Prozess erspart bleibt. Die COCO-Daten werden daher oft als Ausgangspunkt für Wettbewerbe für den Vergleich verschiedener CV-Anwendungen herangezogen. Letztlich ist der Datensatz schon in die für ML üblichen Teilmengen, namentlich den Trainings-, Test- und Validierungsdaten aufgeteilt. Es kann damit sicher gestellt werden, dass vortrainierte Modelle auf diesen Daten lediglich den Trainingsdatensatz kennen und eine unverzerrte Auswertung mittels der Testdaten erfolgen kann. Der vollständige Test- und Validierungsdatensatz aus dem Jahr 2017, der in dieser Studie verwendet wird, kann kostenlos auf der Website der Forschungsgruppe[78] unter dem Link https://cocodataset.org/ heruntergeladen werden und umfasst 5000 Bilder im Joint Photographic Experts Group (JPEG)-Format. Die Bildauflösung variiert von 200×145 bis 640×640 Pixel (Mittelwert: 574×483; Standardabweichung 94×98). Die häufigste Bildauflösung (Median) entspricht einer Auflösung von 640×480 Pixeln (21,22 % der Fälle). Nach einer Ausselektion anhand der Kategorie *person* wurde der Datensatz auf 2693 Bilder reduziert. Da insgesamt 46 Modelle mit 8 verschiedenen Augmentierungsvarianten untersucht werden sollen, wurde dieser Datensatz zur Verkürzung der allgemeinen Laufzeit der Evaluation unter der Berücksichtigung, dass zusätzlich noch fehlannotierte Bilder entfernt werden müssen, auf eine Zufallsstichrobe mit $n = 130$ pseudorandomisierten Bildern (random.seed = 126) verkleinert. Die Fehlmarkierungen der COCO-Daten entstehen, indem beispielsweise ganze Personenverbände als Einzelpersonen markiert, Personenspiegelungen inkludiert oder nur Hände und Finger markiert wurden. In Abbildung 3.2 sind zwei Beispiele für als ungültig deklarierte Bilder zu sehen. Aufgrund der unzulässigen Label-Qualität musste die Zufallsstichprobe anhand der in Kapitel 3.1.2 beschriebenen Gütekriterien händisch weiter gefiltert werden. Der entgültige Datensatz mit einer Größe von $n = 67$ stellte einen annehmbaren Kompromiss zwischen erwarteter Laufzeit und Menge an gültig

markiertem Bildmaterial dar. Für eine vollständige Ansicht aller verwendeten Bilder für diesen Teil der Analyse s. Appendix C.1.

Abbildung 3.2: Beispiele für ungültige Bilder.

Laboraufzeichnung. Für eine geringere Verarbeitungszeit wird mittels eines automatisierten Skripts die für diese Studie nicht relevante Tonspur von der Videospur getrennt. Anschließend wird das Video auf eine Auflösung von *1920 x 1280* herunterskaliert, um den Speicherbedarf und Berechnungszeit für die Vorverarbeitung der Netze zu reduzieren. Es werden zwei Videos zu jeweils einer Minute Laufzeit aufgezeichnet und in insgesamt 3600 Bilder aufgeteilt. Die Aufzeichnung wird unter kontrollierten, gleichbleibenden Lichtbedingungen durchgeführt. Da keine großen Abweichungen zwischen zwei aufeinanderfolgenden Bildern zu erwarten sind, wurde die Rate der extrahierten Bilder auf zwei Bilder pro Sekunde reduziert. Der endgültige Datensatz für die Laboraufzeichnung beläuft sich damit auf 240 Bilder, die im Anschluss an die Aufzeichnung händisch mit entsprechenden Bounding Boxes mit einem Annotierungswerkzeug https://www.makesense.ai/ markiert wurden. Für eine Teilausschnitt der verwendeten Bilder für diesen Teil der Analyse s. Appendix C.2.

3.2.4 Überblick der verwendeten Netzwerkarchitekturen

Insgesamt werden sieben verschiedene Architekturklassen untersucht. Zu jeder Architekturklasse stehen jeweils unterschiedlich viele für Objektdetektion vortrainierte Modelle mit jeweils nochmal unterschiedlichen Hyperparamter-konfigurationen zur Verfügung. Da Modellparameter vor dem Training festgelegt werden, wird entsprechend keine manuelle Modifikation an jeglichen Parametern in Form eines erweiterten Trainings vorgenommen. Für den Zugriff auf die entsprechenden Modelle werden High-Level-Frameworks von TensorFlow

und Ultralytics verwendet. Insgesamt ergibt sich eine Auswahl von 46 verschiedenen vortrainierten Modellen zur Objektdetektion, die miteinander verglichen werden. Es folgt eine kurze Übersicht der trainierten Netze und deren Unterscheidungsmerkmale. Zur Funktionsweise der einzelnen Architekturen, s. Kapitel 2.2.3. Um eine objektive Vergleichbarkeit zwischen allen Architekturen sicherzustellen, werden nur solche inkludiert, die auf dem gleichen Datensatz trainiert worden sind. Die Grundlage der ausgewählten neuronalen Netze ist daher für alle gemeinsam eine Teilmenge der Trainingsdaten des COCO-Datensatzes (s. oben).

Ultralytics-Framework Architekturvariationen. Die Ultralytics-API stellt auf der eigenen Website https://docs.ultralytics.com/models/ eine umfassende Auswahl an vortrainierten Modellen zur freien Verfügung bereit. Diese beinhalten Netze sowohl Versionen verschiedener YOLO-Architekturen von YOLOv3 bis YOLOv9. Neben der versionsbedingten Unterschiede in den Architekturfunktionalitäten gibt es eine Auswahl der Modelle nach Anwendungsbreich (Detektion, Segmentierung, Keypoint-Detektion, Orientierte Detektion und Klassifikation) und Modellgröße (**n**ano, **s**mall, **m**edium, **l**arge, e**x**tra large). Für die Analyse in dieser Arbeit werden die für verwandte Anwendungsbereiche im Bereich der CV verfügbaren Modelle, so zum Beispiel. das Segment Anything Model[79] und das Mobile Segment Anything Model[80] nicht berücksichtigt. Um den Umfang an zu evaluierenden Netzen weiter einzuschränken wird eine Selektion der verfügbaren Netze getroffen, darunter eine etwas ältere Architektur (YOLOv5[81] und eine neuere Architektur(YOLOv8[82], jeweils in unterschiedlichen Größen gemessen an der Anzahl von trainierbaren Parametern. In Tabelle 3.5 sind die wesentlichen Unterschiede der Architekturvariationen dargestellt. Jedes der Modelle ist für Objektdetektion auf 80 verschiedene Kategorien trainiert. Das Backend des Ultralytics-Frameworks ist die ML-Bibliothek Torch[75].

Tabelle 3.5: Ultralytics-Framework Architekturvariationen im Vergleich

Modellbezeichnung	**Parameter in Millionen**	**Validierungsmetrik (mAP)**
yolov5nu	2.6	34.3
yolov5s6u	15.3	48.6
yolov5m6u	41.2	53.6
yolov5l6u	86.1	55.7
yolov5xu	97.2	53.2
yolov5x6u	155.4	56.8
yolov8n	3.2	37.3
yolov8s	11.2	44.9
yolov8m	25.9	50.2
yolov8l	43.7	52.9
yolov8x	68.2	53.9

TensorFlow Detection Model Zoo-Framework Architekturvariationen. Wie auch das Ultralytics-Framework bietet der Tensorflow Detection Model Zoo eine Auswahl verschiedener Architekturen von ML-Modellen sowohl zu Anwendungen im Bereich CV, als auch anderen Teilbereichen an. Eine Übersicht der mit dem COCO-2017-Datensatz vortrainierten Netze mitsamt der Validierungs-**mAP!** (**mAP!**) lässt sich auf der Plattform git unter dem Link https:/github.com/tensorflow/.../tf2_detection_zoo.md oder unter Appenidx B.1 wiederfinden. Für diese Studie werden alle kompatiblen Modelle berücksichtigt, die der Tensorflow Detection Model Zoo beinhaltet. Diese sind im wesentlichen die in Kapitel 2.2.3 beschriebenen Architekturklassen *Center Network (CenterNet)*, *EfficientDet*, *ResNet* und *MobileNet*.

3.2.5 Implementierung

Implementierungsschritte. Die konzeptuelle Implementierung für die Umsetzung der Forschungsziele wird in Tabelle 3.6 visualisiert. Für eine bessere Übersicht ist der Quellcode in einzelne Sektionen unterteilt, die chronologisch nacheinander ausgeführt werden. Der vollständige Quellcode mitsamt Kommentaren kann auf Nachfrage zur Verfügung gestellt werden.

Tabelle 3.6: Implementierungsschritte

Sektion	Beschreibung
1	Einbidung notwendiger Bibliotheken (s. Tabelle 3.4)
2	Definition von konstanten Einstellungen (Datenpfade, Konfidenzwerte, Grenzwerte)
3	Definition von Hilfsfunktionen (s. Tabelle 3.7)
4	Import der Bild-IDs mit der Kategorie **person**
5	Import der Annotationen (Bounding Box-Eckpunkte)
6	Definition der zu verwendenden Modelle
7	Messung über eine iterative Wiederholung der folgenden Schritte auf dem High-End System:
	(a) Laden eines Modells
	(b) Laden eines Bildes
	(c) Iterative Augmentierung des Bildes
	(d) Objektdetektierung auf jedem augmentierten Bild unter Messung der Inferenzzeit
	(e) Vergleich der detektierten Objektkoordinaten mit den tatsächlich markierten Objektkoordinaten
	(f) Extraktion der Messwerte
	(g) Persistieren der Ergebnisse auf einem lokalen Datenträger im csv-Format
8	Strukturierung und Visualisierung aller relevanten Zwischenergebnisse
9	Identifizierung der leistungsfähigsten Netze (n = 3)
10	Wiederholung der Evaluationsschritte auf dem Low-End System mit den leistungsfähigsten Netzen
11	Wiederholung der Evaluationsschirtte mit der Videoaufzeichnung aus dem Labor mit den leistungsfähigsten Netzen
12	Strukturierung und Visualisierung aller relevanten Gesamtergebnisse

Augmentierungen.

Um die Anpassungsfähigkeit der Modelle auf unterschiedliche Bildgegebenheiten zu testen, werden die Testbilder unterschiedlichen Augmentierungen unterzogen. Diese beinhaltet eine JPEG-Komprimierung in zwei Qualitätsstufen, einer Konvertierung in Grauwerte, einer Aufhellung und Abdunklung, einer Weichzeichnung mit einem Gauskern und einer vertikalen Spiegelung. Für die Verwendung der Augmentierungsschritte werden bereits effizient implementierte Versionen dieser Algorithmen in unterschiedlichen Python-Bibliotheken, darunter opencv, verwendet.

JPEG-Komprimierung. JPEG-Komprimierung ist ein für die Komprimierung von Bilddaten entwickelter Algorithmus zur Reduzierung der Größe von Bildformaten. Er wird angewendet, um die Größe von Bildformaten auf bis zu 10 % der ursprünglichen Größe zu reduzieren, ohne dass sich dabei eine sichtbare Qualitätsänderung mit dem menschlichen Auge feststellen lässt[83]. Der Prozess besteht aus mehreren Schritten, unter denen einer der wesentlichsten die Konvertierung des Farbraums darstellt. Der RGB Farbraum, in dem ein Bildformat typischer Weise auf Computern repräsentiert wird, wird in einen anderen Farbraum übertragen: Y Cb Cr. Diese Farbkanäle für stehen für Helligkeit (Y), Blautöne (Cb - Chroma blue) und Rottöne (Cr - Chroma red). Die Wahrnehmungsforschung hat gezeigt, dass Menschen evolutionsbedingt besonders

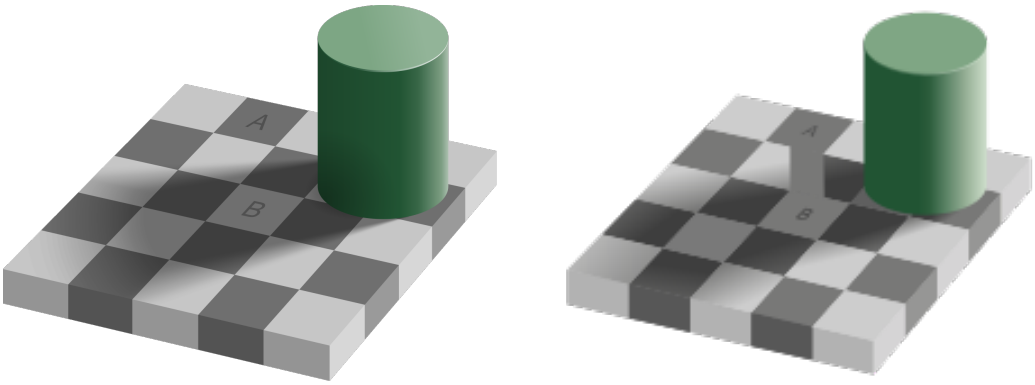


Abbildung 3.3: Schachbrett-Schatten-Illusion[86]

empfindlich gegen Helligkeitsdifferenzen sind, allerdings Änderungen im Farbton kaum auffallen[84]. Ein anschauliches Beispiel für diesen Prozess ist die sogenannte Schachbrett-Schatten-Illusion[85] (s. Abbildung 3.3).

Nach der Konvertierung des Farbraums kann der Anteil der Chrominanzkomponenten im Verhältnis zur Y-Komponente deutlich reduziert werden. So wird Speicherplatz gespart, ohne dass eine merkliche Änderung im Bild erfolgt. Über diesen Konvertierungsschritt hinaus, wird die Kompressionsrate unter Anwendung weiterer Algorithmen, darunter der diskreten Kosinustransformation, erhöht[87].

Grauwertbildung. Die Umwandlung von einem RGB-Bild zu einem Grauwertbild reduziert die Komplexität des Bildes, indem die drei Farbkanäle (Rot, Grün und Blau) in einen einzigen Grauwertkanal zusammenfasst werden. Üblicherweise funktioniert diese Konvertierung basierend auf der pixelweisen Berechnung eines gewichteten Mittelwerts der drei Farbkanäle (Luminanzmethode). Diese verschiedenen Gewichtungen berücksichtigen die gewichtete unterschiedliche Wahrnehmung des menschlichen Auges von Farbtönen:

$$Grauwert = 0.299R + 0.587G + 0.114B$$

Das resultierende Ergebnisbild besteht nur noch aus Grautönen mit Helligkeitsinformation des ursprünglichen Farbbildes.

Weichzeichnen. Der Prozess der Weichzeichnung wurde bereits in Kapitel 2.2.2 beschrieben. Das Bild wird mit einer kleinen symmetrischen Matrix pixelweise geglättet, um Kanten zu verwischen.

Abdunkeln und Aufhellen. Um die Helligkeit eines Bildes anzupassen, wird es zunächst vom RGB-Farbraum in den LAB-Farbraum überführt. Dieser teilt

das Bild in eine Helligkeitsskala (L), einer Rot-Grün-Skala (A) und einer Gelb-Blau-Skala(B). Nach der Konvertierung kann die Helligkeit des Bildes über die L-Komponente reguliert werden. Die Sklierungsfaktoren für diese Komponente werden auf 1.5 für das Aufhellen und auf 0.5 für das Abdunkeln festgelegt.

Spiegeln. Beim Spiegeln wird jedes Pixel des Bildes entlang einer festgelegten Linie (vertikal oder horizontal) spiegelverkehrt umgedreht. Diese Operation erfolgt iterativ, bis alle Pixel entsprechend ihrer Position umgeordnet werden. Diese Augmentierung dient in den meisten Fällen, so wie auch in dieser Studie, dazu, die Menge an Datenpunkten durch Variation der initialen Datenstruktur künstlich zu erhöhen.

Evaluationsmetriken. In der Objektdetektion werden Leistungsmetriken definiert, um die Genauigkeit und Effizienz eines Detektionsmodells zu bewerten[88]. Diese Metriken basieren auf der Klassifikation von Ergebnissen als wahr positiv, wahr negativ, falsch positiv und falsch negativ. Im allgemeinen Verständnis bedeuten die Begriffe **wahr** und **falsch** in der Wissenschaft eine Form der Prognose, also ob eine Klassifikation oder Diagnose festgestellt wurde. **Positiv** und **negativ** bezieht sich auf die tatsächliche Begebenheit, unabhängig von der Prognose. **Wahr positiv** bedeutet also, dass eine Prognose getätigt wurde, die tatsächlich stimmt. Im Folgenden werden diese Begriffe für den Kontext der Obektdetektion spezifiziert:

Ein Ergebnis wird als *wahr positiv* klassifiziert, wenn das Modell ein Objekt korrekt erkennt und die vom Modell erstellte Bounding Box eine IoU von mehr als 50% mit der tatsächlichen Bounding Box (Ground Truth) aufweist.

Ein *falsch positiv*-Fall tritt auf, wenn das Modell ein Objekt identifiziert, das entweder nicht existiert oder dessen detektierte Bounding Box eine IoU von weniger als 50% mit jeder tatsächlichen Bounding Box aufweist. Diese Klassifikation bezieht sich auf Fehler durch Überdetektion oder erhebliche Ungenauigkeiten in der Lokalisierung.

Ein *falsch negativ*-Fall liegt vor, wenn das Modell ein vorhandenes Objekt nicht erkennt. Diese Situation entsteht, wenn ein Objekt im Bild existiert, aber vom Modell keine entsprechende Bounding Box erstellt wird. Falsch negativ-Fehler sind besonders wichtig in Anwendungen, bei denen die vollständige Erkennung jedes Objektes erforderlich ist, wie in sicherheitskritischen Szenarios.

Wahr negativ-Klassifikationen treten in Objektdetektionsfällen nicht auf, da dieser Begriff die korrekte Nicht-Erkennung von Objekten implizieren würde.

Meistens sind die nicht-relevanten Objekten (Tische, Stühle, Tiere, Gegenstände) aber nicht in Form von Bounding Box-Labels gegeben.

Mithilfe dieser Werte werden in ML-gängige Metriken zur Validerung der Modellgüte verwendet. Die am häufigsten berechneten Metriken sind Genauigkeit, Recall und Präzision. Sie werden benötigt, um Aussagen über Modelle aus verschiedenen Kontexten tätigen zu können.

Die Genauigkeit (engl. Accuracy) gibt an, welcher Anteil der Vorhersagen eines Modells insgesamt korrekt ist. Sie wird in dieser Studie als Hauptkriterium für die Modellgüte herangezogen.

$$\text{Genauigkeit} = \frac{TP}{\text{Alle Vorhersagen}}$$

Der Recall (auch Sensitivität genannt) gibt den Anteil der tatsächlich positiven Fälle an. Ein hoher Recall bedeutet, dass das Modell die meisten positiven Fälle erfasst, berücksichtigt dabei aber kein falsch positiv-Fälle.

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

Präzision misst das Verhältnis der korrekt erkannten positiven Fälle (**wahr positiv**) zur Gesamtzahl der vom Modell als positiv erkannten Fälle. Eine hohe Präzision bedeutet, dass ein großer Anteil der vom Modell als positiv klassifizierten Fälle tatsächlich positiv ist, berücksichtigt jedoch nicht, inwiefern positive Fälle nicht erkannt wurden.

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

Eine weitere relevante Metrik in diesem Kontext ist die *Mean Average Precision (mAP)*. Sie berechnet die durchschnittliche Präzision über verschiedene Schwellenwerte der IoU. Die mAP wird oft für Benchmarks und Leistungsvergleiche in wissenschaftlichen Studien verwendet, da sie ein umfassendes Bild der Modellleistung bietet.

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i$$

Hilfsfunktionen. Neben den importierten Skript-Bibliotheken (s. 3.4) werden Hilfsfunktionen implementiert. Diese Hilfsfunktionen dienen im Wesentlichen dazu, Programmierteile in kleinere Codeblöcke aufzuzuteilen, auch als Refactoring bezeichnet. Die Funktionalität der Programmierschritte wird dabei nicht modifiziert. Diese Vorgehensweise verbessert die Lesbarkeit und Wartbarkeit des Codes und kapselt bestimmte Variablen in die Laufzeit der Funktionen ein, sodass kein unnötiger Arbeitsspeicher über die Lebensspanne des Prozesses hinaus beansprucht wird. Für eine Übersicht der implementierten Hilfsfunktionen s. Tabelle 3.7.

Funktionalität	**Beschreibung**
Tensorflow Hilfsfunktionen	Einheitlicher Zugriff auf Tensorflow Model Garden Modelle inklusive Modellimport und Inferenz
JPEG Kompression	Kompression von Bilder auf ein kleineres Dateiformat under leichtem Informationsverlust
LAB Adjustierung	Konvertierung in vom RGB-Farbraum in den LAB-Farbraum und anpassung der Bildhelligkeit
IoU-Berechnung	Berechnung der Überschneidung von zwei Bounding Boxes in Menge pro Hundert Einheiten
Bounding Box Extraktion	Konvertierung der Eckpunkt-Koordinaten aus der Annotationsdatei zu Bounding Boxes
ML-Hilfsfunktionen	ML-Verarbeitungsschritte wie Modellimport, Prognose und Visualisierung
Metrikberechnung	Auswertung der Modellleistung anhand der detektierten Bounding Boxes
Augmentierung	Grauwertkonvertierung, Weichzeichnung, aufhellen, abdunkeln, JPEG-Komprimierung (leicht, stark), spiegeln

Tabelle 3.7: Übersicht zusätzlicher Hilfsfunktionen

4 Ergebnisse

Dieses Kapitel dient einer wertfreien und objektiven Darstellung der Studienergebnisse. Zunächst wird eine umfassende Vergleichsanalyse der verschiedenen neuronalen Netze hinsichtlich ihrer Geschwindigkeit, Genauigkeit und Modellgröße durchgeführt. Anschließend erfolgt eine Darstellung der Resultate unter Einfluss von Störfaktoren wie Lichtverhältnissen, Bildkompression und Weichzeichnung. Die Modelle mit den besten Leistungswerten werden danach insbesondere im Zusammenhang mit unterschiedlicher Leistung in High-End- vs. Low-End-Systemen betrachtet. Letztlich werden die Ergebnisse des Laborexperiments ausgewertet. Eine Interpretation und Erklärung der Resultate erfolgt im anschließenden Diskussionskapitel (s. Kapitel 5).

4.1 Allgemeine Modellevaluation mit dem COCO-Datensatz

4.1.1 Deskriptive Statistiken

Insgesamt wurden im ersten Schritt 46 Modelle evaluiert. Nach Ausschluss ungeeigneter Bilder anhand qualitativer Mängel blieb ein Datensatz mit n = 67 gelabelten Bildern aus dem COCO-Datensatz übrig (s. Appendix C.1). Jedes Modell sollte eine Personendetektion für jedes der Bilder, sowie der jeweils 7 augmentierten Versionen der Bilder vornehmen. Damit wurden 536 Detektionen pro Modell durchgeführt. Folglich wurde ein Gesasmtumfang von $n = 24656$ Iterationen an Inferenzen gerechnet. Das Modell mit der Bezeichnung *centernet_mobilenetv2_fpn* alias *Modell 3* wurde mit einer Genauigkeit von 0 in allen Detektionsschritten als Ausreißer von der weiteren Ergebnisauswertung ausgeschlossen. Die Summe der Inferenzzeiten betrug 34.57 Minuten (Mittelwert 86 ms, Median 70 ms, Standardabweichung 78,86 ms). Die Dauer des Ladevorgangs der Modellparameter in den Arbeitsspeicher ist dabei nicht in die Laufzeit eingerechnet worden. Durchschnittlich waren in den Bildern 3.06 Personen pro Bild markiert (Standardabweichung 2.63, Median 2.0), von denen wiederum durchschnittlich 2.04 Personen erfolgreich detektiert wurden (Standardabweichung 1.40, Median 1.45). Die dazugehörigen relevanten Detektionsmetriken (s. Kapitel 3.2.5) dieser Personen sind in Tabelle 4.1 dargestellt.

Tabelle 4.1: Detektionsergebnisse über alle Modelle hinweg; $n = 24121$

	Median	Mittelwert	Standardabweichung
Accuracy	0.82	0.76	0.17
Recall	0.82	0.76	0.17
Precision	0.90	0.87	0.10

4.1.2 Evaluationsergebnisse bezüglich der Architekturen

Ergebnisse auf Modellebene. In Diagramm 4.1 sind die Ergebnisse der Evaluation aufgeteilt nach Modellen in Abhängigkeit der durchschnittlichen Inferenzzeit in ms und Genauigkeit visualisert. Es lässt sich feststellen, dass kein Modell langsamer als 400 ms war und die meisten Modelle eine relativ hohe Genauigkeit aufweisen, wenn man berücksichtigt, dass für eine vorhergesagte Bounding Box einen einen IoU-Wert von über .5 erreichen musste, um als erfolgreiche Detektion zu gelten. Darüber hinaus ist nur ein schwacher Zusammenhang zwischen Modellparameterzahl und Infernzzeit beobachtbar, was der relativ geringe Korrelationskoeffizient nach Pearson[89] von 0.169 bestätigt. Eine vollständige tabellarische Übersicht der Ergebniswerte kann in Appendix D eingesehen werden.

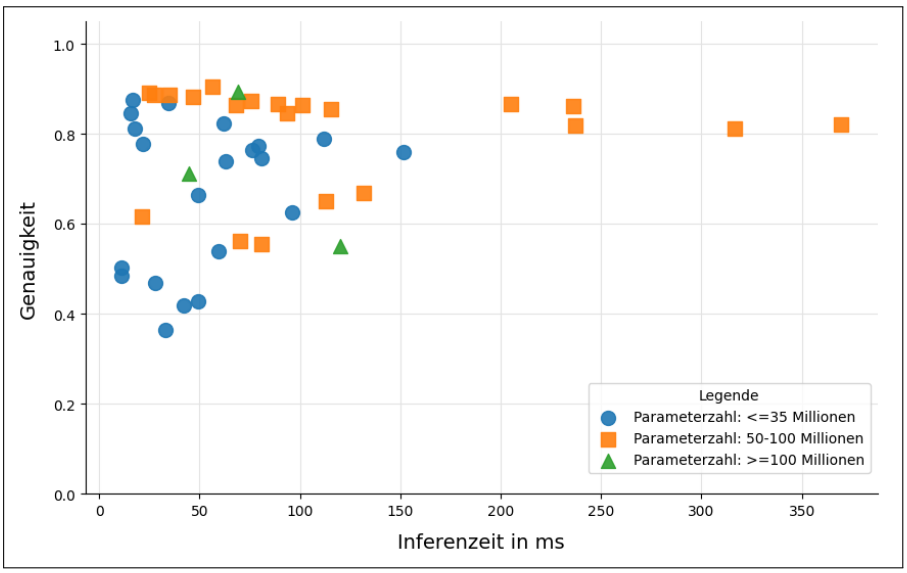


Abbildung 4.1: Modellevaluation mit $n = 45$ Modellen

Ergebnisse auf Modellklassenebene. In Abbildung 4.2 ist die gleiche Evaluationsgrafik um die Dimension der Architekturklasse erweitert worden.

Es lässt sich deutlich feststellen, dass die Modelle mit YOLO-Architektur im allgemeinen Vergleich das beste Verhältnis von durchschnittlicher Inferenzzeit zu Genauigkeit erreichen konnten.

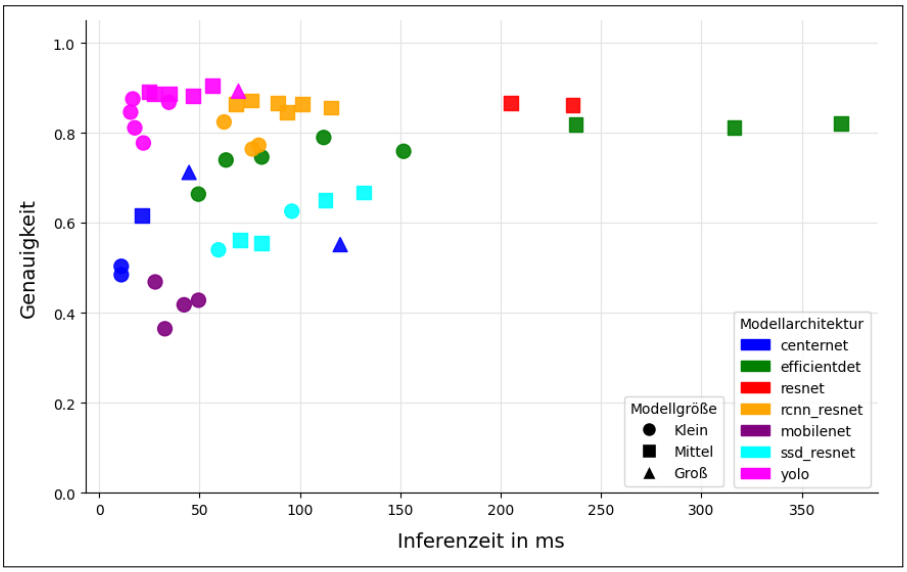


Abbildung 4.2: Modellevaluation farblich separiert nach Modellarchitekturen

4.1.3 Einfluss von Störfaktoren

In Abbildung 4.3 sieht man die Auswirkungen der Augmentierungen gemittelt über alle untersuchten Modelle hinweg. Es ist kein statistisch signifikanter Unterschied zwischen den Werten bezüglich Genauigkeit festzustellen. Allerdings ist deutlich zu sehen, dass die Gruppe ohne Augmentierung (*normal*) eine größere durchschnittliche Inferenzzeit hat. Ein Varianzanalyse (ANOVA) auf signifikante Gruppenunterschiede bestätigt diese Beobachtung. Es gibt einen statistisch signifikanten Unterschied zwischen den Gruppen mit Teststatistik $F = 13.198$ und $p < 0.001$. Der darauffolgende paarweise Vergleich der Gruppen nach Tukey HSD-Algorithmus[90] zeigt statistisch signifikante Unterschiede zwischen der Gruppe *normal* und alle anderen Gruppen (s. Appendix D.2).

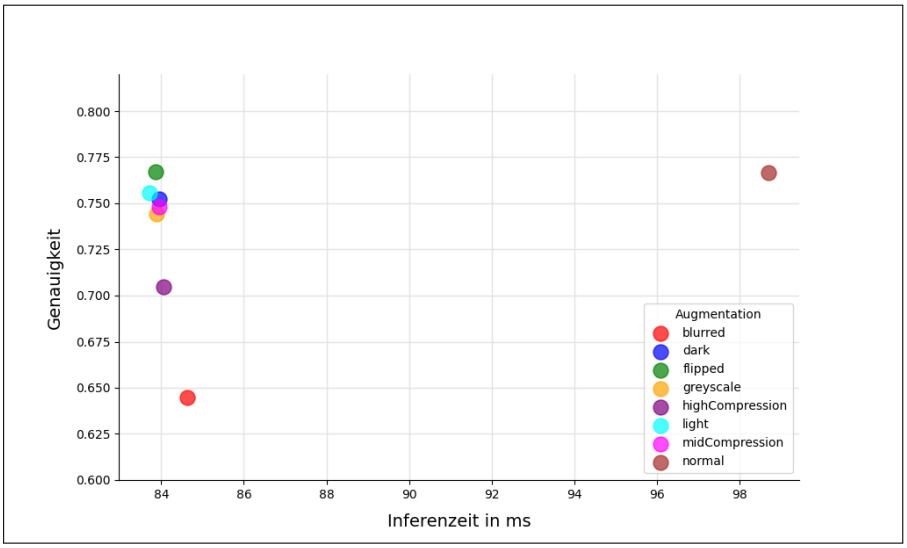


Abbildung 4.3: Modellperformance gruppiert nach Augmentierungen

Eine Gruppierung nach Modellarchitekturen unterstützt dieses Muster. In Abbildung 4.4 sind analog dazu die durschnittlichen Inferenzmetriken als Punkte in einem Punktediagramm dargestellt, die Farbkodierung des Augmentierungstyps wurde zur Darstellung der Modellarchitekturunterschiede entsprechend abgeändert. Aus Gründen der Übersichtlichkeit wird nur noch die Gruppe mit der Augmentierung *normal* (ohne Augmentierung) als Label angezeigt. Die restlichen Gruppen korrespondieren mit minimalen Abweichungen jeweils mit dem in Abbildung 4.3 dargestellten Muster. Darüber hinaus allerdings offenbaren die gruppierten Daten keine zusätzlichen Erkenntnisse.

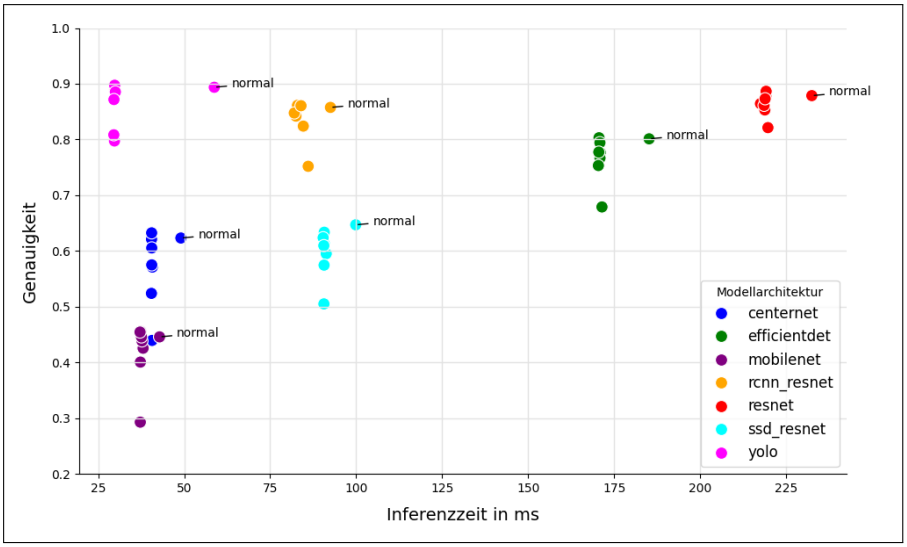


Abbildung 4.4: Modellleistung gruppiert nach Augmentierungen und Architektur

Betrachtung der Bestperformer. Da für den nächsten Schritt Modelle mit den besten Kompromiss zwischen Inferenzzeit und Genauigkeit selektiert werden sollen, werden die Werte mit einer durchschnittlichen Inferenzzeit genauer betrachtet (s. Abbildung 4.5). Für diesen Zweck werden die Ergebnisse auf eine Genauigkeit größer als 0.70 und Inferenzzeit kleiner als 80 ms gefiltert und genauer betrachtet.

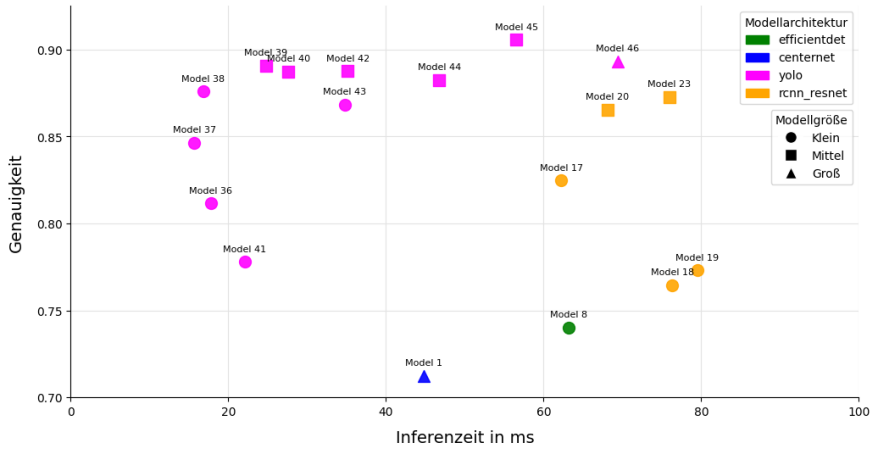


Abbildung 4.5: Gefilterte Modellmetriken nach guter Genauigkeit und Inferenzzeit.

Für den nächsten Schritt in der Evaluationsphase werden drei leistungsstarke Modelle dieser Modellgruppen für eine Untersuchung der Inferenzzeiten auf einem leistungsschwächeren System ausgesucht. Um darüber hinaus eine Aussage über den Zusammenhang zur Modellgröße tätigen zu können, fällt diese Wahla auf ein jeweils kleines, mittelgroßes und großes Modell mit guten Metriken: **Modell 38**, **Modell 40** und **Modell 46** erfüllen diese Anforderungen. In Tabelle 4.2 sind die Leistungsmetriken dieser Modelle einzusehen.

Tabelle 4.2: Übersicht der drei besten Modelle mit unterschiedlich großer Modellparameterzahl

Alias	Modellname	Genauigkeit	Recall	Präzision	Zeit in ms	Modellgröße
Modell 38	yolov8m	0.796	0.902	0.8721	16.791	25.89
Modell 40	yolov8x	0.813	0.915	0.879	27.630	68.20
Modell 46	yolov5xu	0.823	0.918	0.889	69.402	155.48

4.2 Modellevaluation in ressourcenarmer Umgebung

Vergleicht man die Inferenzzeiten des High-End Systems in der vorhergehenden Phase gegen mit den Ergebnissen aus Tabelle 4.3, so zeigt sich der erwartete

Unterschied in der Inferenzzeit. Die weiteren Leistungsmetriken stimmen im Vergleich mit dem High-End System erwartungsgemäß überein und werden daher nicht separat gezeigt.

Tabelle 4.3: Modellleistung auf rechenschwachem System

Alias	Modellname	Zeit in ms	Zeitzuwachs
Model 38	yolov8m	18.892	112.513 %
Model 40	yolov8x	112.006	405.378 %
Model 46	yolov5xu	103.189	148.683 %

4.3 Einsatz in Realumgebung

Die Anwendung der Evaluationssieger im Laborsetting zeigte trotz der ungewöhnlichen Top-Down Perspektive und der Linsenverzerrung durch das Weitwinkelobjektiv durchgängig verlässliche Personendetektionen. Für Beispiel mit roten Bounding Boxes als Vorhersage und blauen Bounding Boxes als Ground Truth, s. Abbildung 4.6.

Abbildung 4.6: Beispiel einer Detektion in der Laborumgebung.

Die Detektionsergebnisse für die drei in Sektion 4.1.3 ausgewählten und in Sektion 4.2 verwendeten Modelle zeigen zuverlässige Vorhersagen über den Zeitverlauf. In Abbildung 4.7 sind die entsprechenden Detektionen im Zeitverlauf im Vergleich zum Ground Truth visualisiert.

Tabelle 4.4: Modellleistung auf den Videoaufnahmen

Alias	Modelname	TP	FP	FN	Genauigkeit	Recall	Präzision
Model 38	yolov5xu	431	0	13	0.971	0.971	1.0
Model 40	yolov8m	426	2	18	0.955	0.959	0.995
Model 46	yolov8x	435	12	9	0.954	0.980	0.973

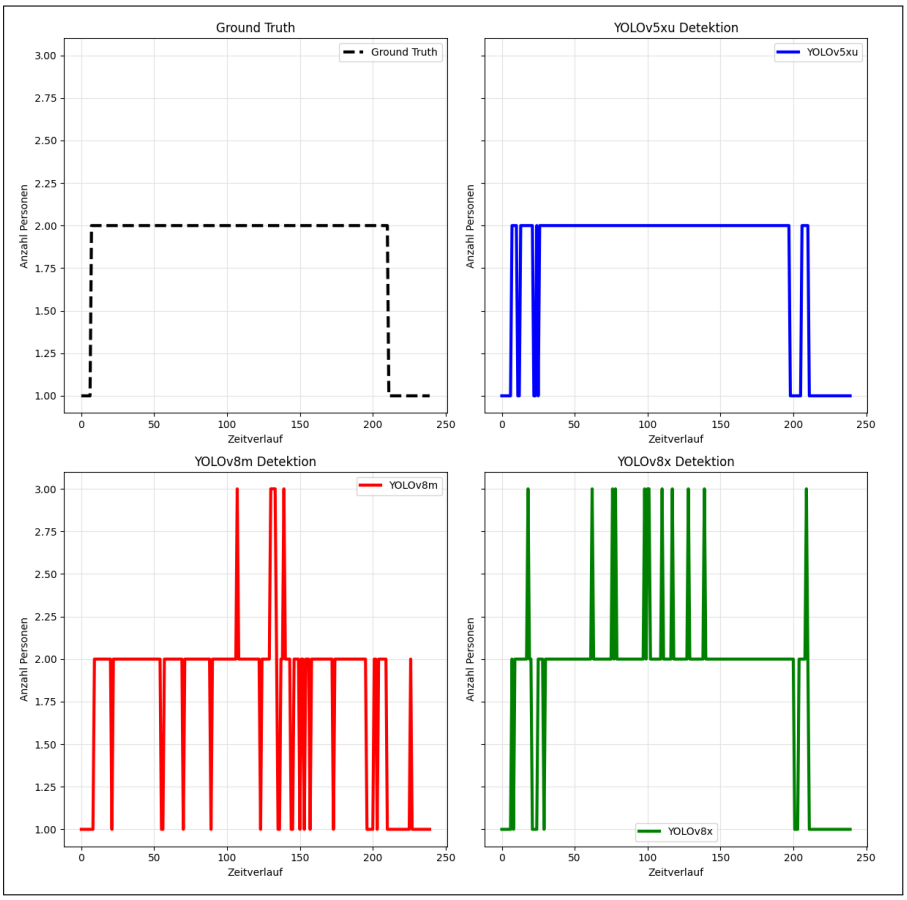


Abbildung 4.7: YOLO-Modell-Personenerkennung im Zeitverlauf während der Laboraufzeichnung

5 Erklärung und Interpretation

In diesem Kapitel werden die Resultate aus Kapitel 4 auf inhaltlicher Ebene analysiert, erklärt und bewertet, sowie anschließend in Bezug zu aktuellen Entwicklungen in Wissenschaft und Technik gesetzt und mögliche Implikationen für Anwendungsfälle abgeleitet. Anschließend wird eine inhaltliche Auseinandersetzung mit Limitationen dieser Studie ausgeführt.

5.1 Interpretation

5.1.1 Deskriptive Statistiken

Extrapoliert man die durchschnittliche Inferenzzeit von 86 ms über alle Modelle hinweg auf Sekunden so ergibt sich ein Wert von 11.63 Detektionen pro Sekunde. Eine Untersuchung hat gezeigt, dass die von Menschen wahrnehmbare Bildrate bei bewegten Prozessen zwischen 15 und 120 Hertz beträgt[91]. Spätestens ab einer Bildrate von 30-40 Bildern pro Sekunde sind Bilder nicht mehr voneinander separierbar und werden vom menschlichen Auge als fließende Bewegung registriert. Damit ist bereits mit der durchschnittlichen Modellleistung eine Detektion fast in Echtzeit gegeben, trotz der relativ hohen Verzerrung dieser Statistik durch die langsameren Modelle. Beschränkt man sich auf die fünf schnellsten Modelle in dieser Studie, so erhöht sich dieser Wert auf durchschnittlich 69.11 Inferenzen pro Sekunde, was definitiv dem Anspruch einer Echtzeiterkennung gerecht wird. In einer Studie zur Detektion von Flugobjekten[92] konnten Forscher mit einem YOLOv8 Modell ähnlich hohe Detektionen pro Sekunde ermitteln (durchschnittlich 50 FPS). Auch bezüglich der Genauigkeit ist eine weit überzufällige Trefferquote von Überschneidungen der vorhergesagten Bounding Box mit den tatsächlichen Bounding Boxes erkennbar. Betrachtet man die Verteilung der Genauigkeiten über alle 24121 Iterationen (s. Abbildung[1] 5.1), so sind die Detektionen überwiegend korrekt. Berücksichtigt man, dass alle Modelle auf vortrainierten neuronalen Netzen basieren und man erwarten dürfte, dass die technische Grenze an Leistungsfähigkeit von Prozessoren in Zukunft weiter steigt, so sind diese Zahlen ein starker Indikator

1 Genauigkeitswerte unter 0 und über 1 sind nicht möglich. Sie existieren in dieser Darstellung lediglich aufgrund der Kernelbreite zur Darstellung der Verteilung als Dichtefunktion

dafür, dass KI in zukünftigen Systemen zur Erkennung von Personen seinen Platz als industriefähige Technologie einnehmen wird. Es ist allerdings an dieser Stelle zu betonen, dass bei der Interpretation der absoluten Geschwindigkeiten Vorsicht geboten ist. Die Inferenzzeit ist extrem stark vom ausführenden System abhängig und wurde in diesem Fall in Hardware mit vergleichsweise großen Rechenkapazitäten durchgeführt (s. Tabelle 3.1). Eine direkte Verallgemeinerung über die dargestellten absoluten Inferenzzeiten ist daher nur bedingt gültig.

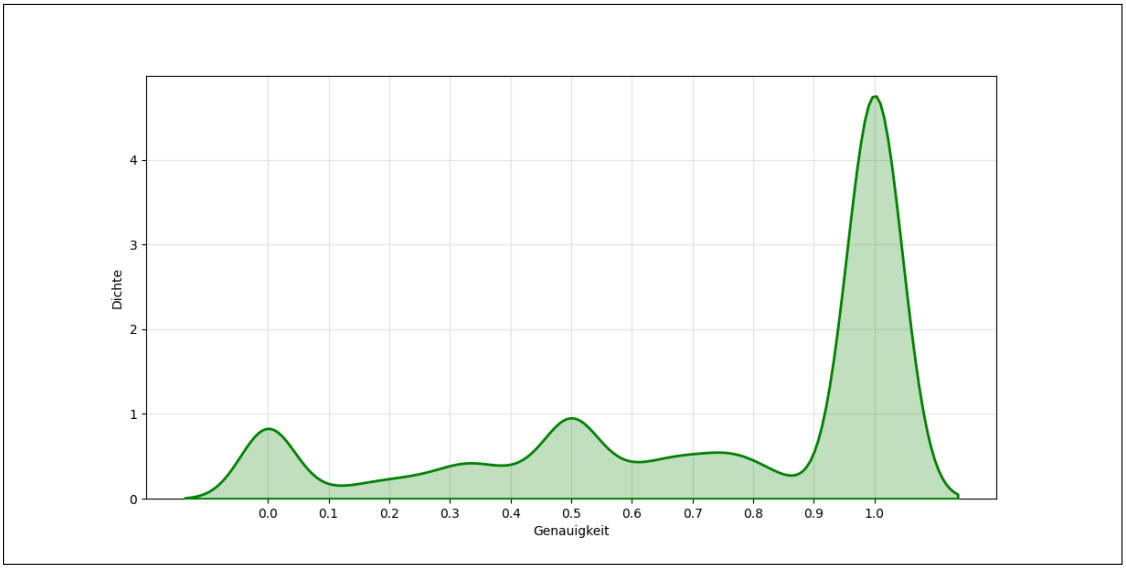


Abbildung 5.1: Dichteverteilung der Vorhersagegenauigkeiten.

Trotz der hohen durchschnittlichen Leistungsmetriken lohnt sich ein Blick auf die Bilder, die weniger gut detektiert werden konnten (s. Tabelle 5.1). Ein Blick auf die Fehlerquoten zeigt, dass Bilder mit größeren Personengruppen größere Einbußen in der Genauigkeit aufweisen.

Tabelle 5.1: Nach Bild ID gruppierte und nach Fehler (absteigend) sortierte Daten

Bild ID	Personen	TP	FN	FP	Summe der Fehler
338905	11	4.503	6.497	0.341	6.838
238410	9	3.605	5.395	1.148	6.543
6471	10	5.540	4.460	0.707	5.167
429690	10	5.142	4.858	0.224	5.082
2685	6	3.099	2.901	1.017	3.918

Eine Prüfung der Bilder (zwei Beispiele sind in Abbildung 5.2 zu sehen) ergab, dass es sich um Seitenaufnahmen von Personengruppen handelt. Abgesehen

vom kostenintensiven Training mit einem deutlich größeren Datensatz und der Weiterentwicklung bestehender Architekturen lässt sich diesem Problem relativ leicht lösen, indem man die Kameraposition und Ausrichtung expliziet an den Anwendungsfall anpasst. Zur Registrierung von Personen in Büroräumen wäre eine mittig platzierte Deckenkamera für diesen Zweck gut geeignet, da die Kamera an dieser Stelle kein Hindernis darstellt und sich Menschengruppen aus der Deckenperspektive gut auseinanderdifferenzieren lassen.

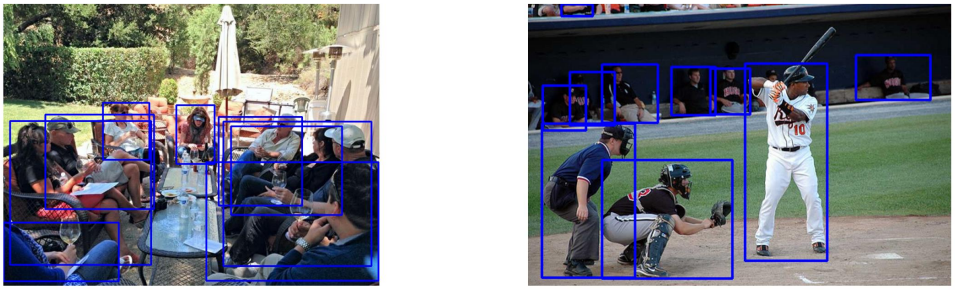


Abbildung 5.2: Bilder mit häufigsten Fehldetektionen

5.1.2 Evaluationsergebnisse bezüglich der Architekturen

Evaluation auf Modellebene. Diese Studie hat gezeigt, dass eine hohe Modellkomplexität und eine große Anzahl an Paramtern nicht zwangsläufig eine wünschenswerte Leistung erbringt. Selbst bei Ausblendung der Inferenzzeit gab es nur ein einziges großes Modell (YOLOv5x6u), das in dieser Studie mit der Genauigkeit der kleineren Netze mithalten konnte. Neben der Tatsache, dass große Netze mehr Rechenleistung beanspruchen und deutlich schwerer zu erklären, entwerfen und interpretieren sind[93], deckt sich dieser Befund mit Studien, die die Auswirkung von Modellkomplexität auf Vorhersagegüte untersucht haben und feststellten, dass zu große Modelle eher zu schlechteren Resultaten führen[94].

Obwohl die absoluten Zahlen dieser Studie wenig allgemeine Gültigkeit haben, lassen sich Erkenntnisse über die relative Modelleistung der vielen Modelle untereinander gewinnen. Es gibt zwar viele Publikationen zum Vergleich einzelner Modelle miteinander[95,96], es lassen sich aber in der wissenschaftlichen Literatur kaum Benschmarks zum Vergleich vieler Netze für Objekterkennung unter den gleichen Versuchsbedingungen finden. Möglichlierweise liegt die Ursache darin,

dass die breite Verfügbarkeit dieser Objekterkennungsmethoden eine eher neue Entwicklung darstellt oder aufgrund der Dominanz der YOLO-Modellarchitektur wenig Bedarf in einer Metaanalyse besteht. Dennoch besteht ein Mehrwert in einer solchen Vergleichsstudie. Zum Einen kann durch die Gewährleistung einer gleichen Datenbasis und Hardware eine Verzerrung durch Interaktionen mit Außenkriterien minimiert werden. Zum Anderen bietet sie die Möglichkeit, eine solide Entscheidungsbasis für den speziellen Anwendungsfall zu bieten, indem Modelle anhand nach Größe, Geschwindigkeit und Genauigkeit in einer Rangfolge angeordnet werden können. Auch lässt sich im Falle, dass bestimmte Architekturen aufgrund von Kompatibilitätsproblemen als Möglichkeit ausscheiden, eine passende Alternative finden. Konkret auf diese Untersuchung ließen sich mit den Evaluationsergebnissen drei Gewinnermodelle mit einem angemessenen Kompromiss von Geschwindigkeit, Genauigkeit und Größe für die zwei Folgeexperimente ermitteln.

Evaluation auf Modellarchitekturebene. Anhand der Daten wird ersichtlich, dass YOLO-Modellarchitekturen im Vergleich mit anderen Gruppen die domante Gruppe bezüglich Geschwindigkeit und Inferenzzeit sind. Die nächstbeste Architektur ist demnach die Klasse von faster R-CNN-Modellen. Alle weiteren Modellarchitekturen lassen weniger präzise Aussagen über die Gruppeninterne Leistung zu, da sie stärkere Abweichungen je nach Modellgröße aufweisen. Bei der graphischen Darstellung der Modelleistung in Abbildung 4.2 sind deutliche Muster zwischen diesen Klassen erkennbar. Es zeigt sich, dass die Art und Weise, wie neuronale Netze im Innern konzeptuell strukturiert sind wichtiger ist, als die tatsächliche Modellgröße oder weitere Hyperparameter, die die individuellen Modellunterschiede ausmachen. Innerhalb jeder Klasse lassen sich tendenziell leichte Zusammenhänge zwischen Genauigkeit, Größe und Inferenzzeit vermuten, allerdings sind die entsprechenden Korrelationskoeffizienten[89] eher schwach: $-0.02 < r_{ij} < 0.2$, wobei i und j für die Modellarchitekturklassen stehen.

Empfindlichkeit gegenüber Bildaugmentierungen.

Die geringen Unterschiede in der Genauigkeit zwischen den Augmentierungsformen sprechen für die allgemein hohe Anpassungsfähigkeit von CNN-basierten Architekturen für Objektdetektion, begründet in deren Robustheit gegenüber Bildveränderungen (Translation, Rotation, Skalierung). Der signifikante Unterschied zwischen den nichtaugmentierten Bildern und den augmentierten Bildern bezüglich der Inferenzzeit lässt sich dadurch erklären, dass in der

Implementierung die nichtaugmentierte Version des Bildes in der Iteration immer als erstes geladen wurde. Das bedeutet, dass sowohl das Bild selbst, als auch die Annotationen von der Festplatte geladen werden mussten. In den darauffolgenden Iterationen, in denen das Bild augmentiert wurde, sind diese Informationen bereits im deutlich schneller zugreifbaren Arbeitsspeicher des Computers gepuffert. Diese Erklärung konnte über einen Posthoc-Test bestätigt werden, in dem die erste Iteration in der Augmentierungsschleife doppelt ausgeführt wurde und erst die zweite Inferenz als tatsächlicher Messwert erfasst wurde. Dieser Erklärungsansatz gilt auch für die nach Architekturklassen gruppierten Messergebnisse.

5.1.3 Modellevaluation in ressourcenarmer Umgebung

Zwar zeigen die Inferenzzeiten auf dem Raspberry Pi einen Zuwachs in der Inferenzzeit an, allerdings skaliert die Berechnungszeit bei weitem nicht so stark mit der Rechnerkapazität wie erwartet. Berücksichtigt man, dass die Inferenz auf dem leistungsstarken Computersystem mit der Unterstützung einer Graphikkarte gerechnet wurde, die besonders auf die parallele Verarbeitung von Fließkommazahlen optimiert ist, so sind die Inferenzzeiten auf dem Raspberry Pi vergleichsweise gut. Der deutliche Unterschied zwischen **Modell 38** und den anderen Modellen entsteht durch die Begrenzung des RAM-Speichers des Geräts. Während **Modell 38** klein genug war, um alle Parameter und Bildinformationen in den Arbeitsspeicher zu laden, war dies bei **Modell 40** und **Modell 46** nicht mehr gegeben. Neben den Hintergrundprozessen des Betriebssystems blieb nicht genug Speicherplatz im RAM des Raspberry Pi für alle Modellparameter. Üblicherweise ist für diesen Fall ein Teil der Festplattenspeichers als virtueller Zusatzspeicher (*swap*) reserviert, der im Austausch für den zusätzlichen Speicherplatz deutlich höhere Latenzzeiten hat[97]. Wird auch dieser Speicher ausgelastet, so greift gewöhnlicherweise ein Sicherheitsmechanisums des Betriebssystems ein, um einen Absturz des Computersystems zu verhindern und beendet umgehend Prozesse mit hoher Arbeitsspeicherauslastung. Da im Fokus des Experiments die Überprüfung der unmodifizierten vortrainierten Modelle steht, wurde an dieser Stelle keine Reduzierung der Modellgröße unternommen. Um die Modelle **40** und **46** dennoch auf dem Raspberri Pi lauffähig zu machen, wurde der zugewiesene swap-Speicher manuell auf 1024 MegaByte erhöht. Die damit einhergehende Latenz der Datenübertragung wird dennoch zur Inferenzzeit gezählt, damit der Speicherbedarf des Modells in speicherarmen Systemen als Nachteil bei der Bewertung ins Gewicht fällt. Die durchschnittliche Inferenzzeit von 18.892 ms des Modells YOLOv8m im

Raspberry Pi ist vergleichbar mit der Inferenzzeit des High-End-Systems (16.791 ms) und impliziert damit die Möglichkeit der Anwendung einer Objektdetektion in beinahe Echtzeit trotz geringer Rechenkapazität. Unter Anwendung von Netzskalierungsmethoden[61] und Quantisierung[98] ist es damit sogar vorstellbar dieses leistungsfähige Netzwerk in eingebettete Systeme zu implementieren.

5.1.4 Einsatz in Realumgebung

In der Labormessung haben alle Modelle sehr gute Messwerte erzielt. Das ist insbesondere beeindruckend, weil im Trainingsdatensatz für diese Modelle fast gar keine Bilder aus dieser Top-Down-Perspektive vorgekommen sind. Dieses Ergebnis unterstreicht nicht nur die sehr gute Generalisierbarkeit dieser Modelle, sondern bedeutet auch, dass ein neues Training der Modellparameter speziell für den Anwendungsfall der Personenzählung in Büroräumen nicht erforderlich ist. Über den Zeitverlauf hinweg betrachtet, hat das größte Modell (**46**) die beste Vorhersageprognose erreicht. Für den Transfer in eine Realumgebung würde man sich vermutlich dennoch für die YOLOv8m-Variante entscheiden, da die Modellgröße von 155.48 Millionen Paramter und Inferenzzeit von 69.40 ms des Gewinnermodells im Vergleich zu lediglich 25.89 Millionen Parametern und 16.80 ms Inferenzzeit in keinem guten Verhältnis zum erwarteten Gewinn von 1.7 % Genauigkeit steht, vor allem da diese Datensatzstichprobe relativ klein ist und die Messgenauigkeiten daher kleineren Schwankungen unterliegen.

5.1.5 Bewertung der Benutzerschnittstellen

Bei der Anwendung der Ultralytics-Schnittstelle zur Anwendung der YOLO-Architektur gab es leichtere Versionsinkompatibilitäten mit den Python-Bibliotheken auf dem Raspberry Pi, die durch Nutzung von virtuellen Umgebungen behoben werden konnten. Auf dem High-End System traten diese Schwierigkeiten nicht auf. Die Unterstützung für CUDA[99] und CUDNN[100] Bibliotheken zur Optimierung der Geschwindigkeit mit der Graphikkarte erfolgte ebenfalls problemlos. Darüber hinaus ist die Ultralytics-Modellbibliothek leicht im Internet unter dem Weblink https://docs.ultralytics.com/models/ auffindbar, gut strukturiert und dokumentiert, inklusive verschiedener Modellformate und begleitenden Beispielen und Videomaterial zur Anwendung. Zusätzlich kann der Import der Modelle auch ohne externes Herunterladen der Modellparameter

direkt über die Python-Programmiersprache erfolgen. Der **Model Zoo** von TensorFlow, mit dem alle nicht-YOLO-Modelle importiert wurden, ist unter unterschiedlichen Weblinks in Form von git-Repositories ansteuerbar (https://github.com/tensorflow/models/). Ein Testversuch der Implementierung auf dem Raspberry Pi schlug aufgrund von unauflösbaren Konflikten der Keras-, Tensorflow-, Pythonversionen und der ARM-Prozessorarchitektur fehl. Die Anwendung der Modelle auf dem High-End-System konnte nur über das Einbinden von Hilfsfunktionen implementiert werden und benötigte neben der Datei für die Modellarchitektur eine pipeline-config Datei und die Checkpoint-Datei mit den persistierten vortrainierten Modellparametern. Es gestaltete sich schwieriger die Dokumentation zur Nutzung der Benutzerschnittstelle nachzuvollziehen, auch weil diese teilweise nicht mehr aktuell war. Positiv ist zu bemerken, dass die Auswahl an unterschiedlichen Modellarchitekturen deutlich größer als als bei Ultralytics und über Objektdetektion hinaus auf weitere Modelle zugegriffen werden kann, zum Beispiel zur Audio- und Spracherkennung, Reinforcement Learning oder Textklassifizierung. Bei einem Vergleich dieser beiden Schnittstellen überwiegen für den Anwendungsfall der Objekterkennung die Vorteile bezüglich Benutzerfreundlichkeit bei der Ultralytics-Schnittstelle.

5.2 Evaluation der Gesamtstudie

Die Studienergebnisse bieten solide Erkenntnisse zum Vergleich von verschiedenen neuronalen Netzarchitekturen auf verschiedener Hardware, sowohl in Anwendung auf öffentlichen Bilddaten mit Augmentierungen als auch im Kontext der Personenzählung in einem Bürosetting. Sie können für NutzerInnen eine Entscheidungsgrundlage zur Selektion eines geeigneten Modells für den individuellen Anwendungsfall dienen und bestätigen, dass Objekterkennugns-algorithmen in der Praxis realisierbar sind. Allerdings unterliegt die Anwendung der Ergebnisse einigen Limitationen.

Erstens erfolgte die Modellevaluation mit der Einschränkung auf eine Teilmenge von vortrainierten Modellen, die über Benutzerschnittstellen abrufbar sind. Obwohl sich darunter *state of the Art* Objektdetektor-Modelle befinden, kann nicht ausgeschlossen werden, dass es darüber hinaus noch bessere Architekturen gibt, die in keine der hier untersuchten Modellkategorien fallen.

Zweitens wurde eine high-level Programmiersprache für die Implementierung der Modelle verwendet. Umso abstrakter eine Programmiersprache, desto höherwertige Operationen lassen sich damit ausführen. Die entsprechenden Anweisungen

müssen letztendlich aber in Maschinensprache übersetzt werden, was bei höheren Abstraktionsniveaus mehr Latenzzeiten bedeutet. Python-Module für ML-Anwendungen und Matrixmultiplikationen werden daher im Backend über hoch effiziente und maschinennahe Implementierungen in c übersetzt, um die Performanz dieser Algorithmen zu erhöhen. Dennoch könnte eine Implementierung in einer maschinennäheren Programmiersprache Leistungsgewinne erzielen[101].

Drittens haben nicht viele Systeme eine GPU, die besonders effizient für parallele Bereichnung von Parametern geeignet sind. In dieser Studie wurde daher bewusst ein zweites System getestet, das nicht auf eine GPU oder ähnliche für ML-Optimierte Prozessoren (**TPU**, **NPU**) zugreift.

Viertens kann keine Aussage zur Leistung bei anderen Klassen wie Fahrzeugen oder Tieren allein anhand dieser Studienergebnisse gemacht werden, da sie nicht im Fokus dieser Studie standen. Da jedoch alle Modelle theoretisch ebenfalls auf diese Klassen trainiert worden sind, sollte man davon ausgehen können, dass die Genauigkeit für andere Klassen ähnlich hoch ist, vorrausgesetzt, dass der Trainingsdatensatz, auf dem die Modelle basieren, eine ausreichend große Repräsentation dieser Klassen vorweist.

Letztlich sollte auch erwähnt werden, dass bei der Verarbeitung von Bilddaten, auf dem sich Personen befinden, insbesondere wenn deren Bewegungen zwischen mehreren aufeinanderfolgenden Bildern getrackt werden, strenge Datenschutzauflagen zu berücksichtigen sind. Die Speicherung und Verarbeitung solcher Daten sollte unter allen Umständen nur im Wissen und mit dem Einverständnis der entsprechenden Personen erfolgen.

6 Zusammenfassung und Fazit

Dieses Kapitel dient der Refklektion der gesamten Studie. Es wird ein kleiner Ausblick in Möglichkeiten für zukünfte Untersuchungsmöglichkeiten gegeben und ein abschließendes Fazit gezogen.

6.1 Zusammenfassung

In dieser Studie wurde eine Metaanalyse über geeignete vortrainierte neuronale Netze im Bereich CV zur Erkennung von Personen durchgeführt und anhand von zwei Folgeexperimenten die Eignung der leistungsfähigsten Modelle zum Einen in einem System mit wenig Rechenleistung und zum Anderen in einer Büroumgebung zur Simulation eines echten Anwendungsfalles durchgeführt. Die Motivation dieser Studie richtet sich auf die Annäherung von KI-Systemen im Allgemeinen und der Anwendung im Bereich von CV im Speziellen zur Erhöhung des allgemeinen Lebensstandards und der Umrüstung zur Automatisierung im Sinne einer nachhaltigeren Gesellschaft, beispielsweise über eine autonome Steuerung von Thermalanlagen. Zu diesem Zweck wurden zunächst die Forschungsziele formuliert und zum Verständnis der Thematik eine wissenschaftlich fundierte Basis relevanter Konzepte und Technologien aufgearbeitet. Anschließend wurde die methodische Vorgehensweise im Detail einschließlich des Studienkonzepts und der Implementierung beschrieben. Die drei Ergebnisteile der Studie umfassen aufgearbeitete Analysedaten zur Modellleistung von 45 unterschiedlichen vortrainierten neuronalen Netzen mit 7 verschiedenen Netzarchitekturen auf dem COCO-Datensatz bezüglich Geschwindigkeit, Genauigkeit und Modellgröße, Messdaten der Personenzählung von drei besonders leistungsfähigen Modellen mit unterschiedlichen Modellgrößen mit einem Raspberry Pi 3b und Messdaten aus einer Laborstudie mit Videodaten aus einer an der Decke angebrachten Weitwinkelkamera. Die Evaluationsergebnisse bieten eine gute Entscheidungsgrundlage für AnwenderInnen, anhand von Größe, Genauigkeit und Geschwindigkeit ein passendes vortrainiertes Modell für den speziellen Anwendungsfall auszusuchen und in bestehende Systeme zu integrieren. Sowohl bei der Anwendung bei einem Low-End-System als auch bei der Videoaufnahme im Labor bestätigt sich, dass vortrainierte

neuronale Netze zur Objekterkennung eine realistische Lösung zur Erfüllung von CV-Aufgaben darstellen. Unter Anwendung der Benutzerschnittstellen benötigen sie vergleichsweise wenig Fachexpertise zur Einbettung in bestehende Systeme, sind recheneffizient, robust gegenüber Bildadjustierungen und funktioneren in Realumgebungen auch bei Kameraperspektiven, die von den Bildperspektiven in den Trainingsdaten abweichen. Möchte man noch bessere Ergebnisse, so lässt sich die Modellarchitektur leicht an eigene Daten anpassen. Die Ergebnisse aus allen drei Teilstudien verdeutlichen die Realisierbarkeit der Implementierung in Alltagssysteme zu Objekterkennungsaufgaben, insbesondere zur Zählung von Menschen. Über die Verfolgung der Untersuchungsziele hinaus erfolgte eine subjektive Beurteilung der Anwendernähe der Benutzerschnittstellen, die diese Netze kostenlos zur Verfügung stellen anhand der gesammelten Erfahrungen und Beobachtungen mit der Technologie während der Durchführung der Studie.

6.2 Ausblick auf zukünftige Forschung

Zum untersuchten Forschungsgebiet ließen sich noch eine Vielzahl von Einflussvariablen untersuchen, die aufgrund von Ressourcen- und Zeitlimitationen nicht untersucht werden konnten. Obwohl eine Vielzahl an Modellen in die Evaluation aufgenommen wurde, ist die Auswahl der Modelle trotzdem limitiert. Um die Ergebnisse noch aussagekräftiger zu machen, könnte man die nicht inkludierten neuronalen Netze ebenfalls in die Metaanalyse aufnehmen. Dazu gehören unterschiedliche Varianten der hier untersuchten Architekturen mit unterschiedlichen Versionen, weiteren Optimierungsfeatures, Hyperparameterkonfigurationen und Modellgrößen, sowie unterschiedlichen Trainingsdatensätzen. Zudem ist es nicht unwahrscheinlich dass neben den bestehenden Netzen im Laufe der Zeit weitere Architekturen publiziert werden oder in die Benutzerschnittstellen weitere Varianten von bereits bestehenden Architekturen aufgenommen werden, deren Messwerte in die Sammlung der bereits evaluierten Modelle aufgenommen werden müssten. Auch die Untersuchung in Low-End-Systemen ließe sich in gleichem Maße erweitern. Obwohl das Raspberry Pi 3 bereits relativ wenig Rechenleistung im Vergleich zu modernen Computern oder Mobilgeräten mit sich bringt, sollte die Anwendung auf noch rudimentäreren Systemen ausgiebig exploriert werden, sodass die Nutzung von Autonomisierung und KI im industriellen Kontext noch lukrativer wird und das Motiv der Nachhaltigkeit in diesem Sinne vorangetrieben wird. Denn parallel mit der Entwicklung von immer leistungsfähigeren Kleingeräten bis hin zu Mikrocontrollern

erfolgen könnte ein Punkt in der Zukunft erreicht werden, an dem das Verzichten auf nachhaltige autonome Systeme keine rentable Option mehr darstellt.

Zum Einsatz in bestimmten Kontexten kann die Analyse weiterer Einflussgrößen zusätzliche Erkenntnisgewinne bringen. Für das Feldexperiment wurden Daten für den *Proof of Conzept* der Realisierbarkeit von Personenzählung in einem Realsetting erhoben. Die Erweiterung des Basisdatensatzes würde die Aussagevalidität der Ergebnisse deutlich verbessern. Man könnte die kontrollierte Umgebung mit Störvariablen beeinflussen, um die Reaktionen der Netze darauf zu beobachten, darunter Haustiere und unterschiedliche Lichtverhältnisse. Auch die anwesenden Personen könnten in weiteren Erhebungen mehr Aussagekraft über Generalisierbarkeit ermöglichen, da sie über Kleidungsvielfalt, Accessoires, Geschlecht, Hautfarbe und weiteren Attributen eine größere Variabilität in die Analyse bringen. Zusätzlich könnten auch unterschiedliche Kameraauflösungen und Linsen eingesetzt werden. Eine entsprechende Studie[102] konnte beispielsweise zeigen, dass YOLO-Netze sogar gut mit Fischaugenkameras, eingesetzt in Fahrerkabinen von PKWs performen. Die importierten Modelle können außerdem an den speziellen Fall angepasst werden, indem das vortrainierte Netz mit Daten aus dem Anwendungsfeld über **transfer learning**[103] noch einmal nachtrainiert wird oder fortgeschrittene Algorithmen wie **Student-Teacher-Systeme**[104] angewendet werden, bei denen größere Netze (**Teacher**) gelerntes Wissen auf kleinere Netze (**students**) übertragen. Auch könnte untersucht werden, inwiefern die Integration mehrerer unterschiedlicher Datenquellen, so zum Beispiel über Mikrofone aufgezeichnete Audiospuren- und Lokalisationen, sowie die Detektion über mehrere Bilder über einen kleinen Zeitraum hinweg zum Ausgleich von Fehldetektionen in Einzelaufnahmen die Genauigkeit der exakten Personenzahl erhöhen könnte.

6.3 Schlussfolgerung

Aus der ausführlichen Analyse der bestehenden Technologien in dieser Arbeit lässt sich ableiten, dass der Einsatz von Objektdetektion zur Personenzählung und für andere Anwendungsbereiche gut realisierbar ist. Unter dem Einsatz von kostengünstigen Endsystemen wie dem Raspberry Pi lässt sich gut auf die Eignung von neuronalen Netzen für rudimentärere Systeme schließen. Allerdings nehmen die Herausforderungen bezüglich Kompatibilität zu, je kleiner das System ist. Eine High-Level Schnittstelle wie Ultralytics ist zwar gut in gängigen Computern und dem Raspberry Pi umsetzbar, ist aber für Kleingeräte mit sehr rudimentären Betriebssystemen nicht vorgesehen. Trotzdem erleichtern sie die Vorverarbeitung und den Zugriff auf bestehende Technologien und erlauben, diese an den individuellen Anwendungsfall anzupassen.

Es ist zudem bemerkenswert, dass die Algorithmen und Konzepte, die hinter den Netzwerkarchitekturen wichtiger für eine gute Leistung sind, als die eingestellten Hyperparameter wie Modellgröße oder Anzahl der künstlichen Neuronen pro Schicht. Es bleibt damit bei der Kreativität und dem Wissen der Menschen, das volle Potential des mächtigen Werkzeugs der Zukunft namens künstliche Intelligenz auszuschöpfen.

Teil II

Appendix

A Appendix: YOLO-Versionen

Tabelle A.1: YOLO-Versionen im Überblick

YOLO Version	Neue Features	Verbesserungen	Referenz
YOLOv1	Objekterkennung als Regression	Schnelle Erkennung; jedoch weniger präzise bei kleinen Objekten.	[54]
YOLOv2	Einführung von Anchor Boxes; Batch Normalization; Höhere Auflösung der Eingagsschicht.	Bessere Lokalisierung und Erkennung; höhere Auflösung.	[105]
YOLOv3	Einführung von 3 verschiedenen Skalen. Detektion mehrerer Klassen pro zelle	Verbesserte Erkennung bei kleinen Objekten.	[106]
YOLOv4	Verwendung von Darknet-53 als Backbone; Einführug von Optimierungsfeatures, unter anderem Self-adversarial-training (SAT).	Verbesserungen in Geschwindigkeit und Genauigkeit.	[107]
YOLOv5	Implementierung in PyTorch; Modellskalierung; automatische Batch-Größe.	Einfachere Nutzung und flexible Anpassungen; schnellere Trainingzeiten.	[108]
YOLOv6	Verbesserungen in der Architektur; Weitere Einführug von Optimierungsfeatures	Schnellere und genauere Erkennung; bessere Performance auf verschiedenen Hardwaretypen.	[109]
YOLOv7	Verbesserungen der Architektur und Einführug weiterer Optimierungsfeatures.	Verbesserte Erkennungsgenauigkeit und Effizienz im Vergleich zu YOLOv6.	[110]
YOLOv8	Optimierung der Netzwerkarchitektur; Verbesserungen im Training und Datenverarbeitung.	Erhöhte Genauigkeit und Geschwindigkeit bei der Objekterkennung.	[111]
YOLOv9	Derzeit experimentelle Version mit Implementierung neuer Konzepte (PGI & GELAN)	Erhöhte Genauigkeit und Geschwindigkeit bei der Objekterkennung.	[111]

B Appendix: Modellmetriken

Tabelle B.1: TensorFlow Detection Model Zoo Architekturvariationen

Modellbezeichnung	Parameter in M	mAP bei Validierung
CenterNet HourGlass104 512x512	191.29	41.9
CenterNet HourGlass104 1024x1024	191.29	44.5
CenterNet Resnet50 V1 FPN 512x512	25.28	31.2
CenterNet Resnet101 V1 FPN 512x512	44.30	34.2
CenterNet Resnet50 V2 512x512	29.85	29.5
CenterNet MobileNetV2 FPN 512x512	2.39	23.4
EfficientDet D0 512x512	5.57	33.6
EfficientDet D1 640x640	8.32	38.4
EfficientDet D2 768x768	10.00	41.8
EfficientDet D3 896x896	14.16	45.4
EfficientDet D4 1024x1024	23.32	48.5
EfficientDet D5 1280x1280	36.75	49.7
EfficientDet D6 1280x1280	55.51	50.5
EfficientDet D7 1536x1536	55.51	51.2
SSD MobileNet V2 FPN 320x320	6.07	20.2
SSD MobileNet V1 FPN 640x640	12.08	29.1
SSD MobileNet V2 FPNLite 320x320	2.63	22.2
SSD MobileNet V2 FPNLite 640x640	2.63	28.2
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	32.84	34.3
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	32.84	38.3
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	51.83	35.6
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	51.83	39.5
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	67.48	35.4
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	67.48	39.6
Faster R-CNN ResNet50 V1 640x640	28.46	29.3
Faster R-CNN ResNet50 V1 800x1333	29.19	31.6
Faster R-CNN ResNet50 V1 1024x1024	28.46	31.0
Faster R-CNN ResNet101 V1 640x640	47.45	31.8
Faster R-CNN ResNet101 V1 800x1333	48.18	36.6
Faster R-CNN ResNet101 V1 1024x1024	47.45	37.1
Faster R-CNN ResNet152 V1 640x640	63.09	32.4
Faster R-CNN ResNet152 V1 800x1333	63.82	37.4
Faster R-CNN ResNet152 V1 1024x1024	63.09	37.6
Faster R-CNN Inception ResNet V2 640x640	59.47	37.7
Faster R-CNN Inception ResNet V2 1024x1024	60.02	38.7

C Appendix: Datenbeispiele

Abbildung C.1: Auf 200 x 200 Pixel herunterskalierte Collage der verwendeten Bilder für die Evaluation der neuronalen Netze; $n = 67$

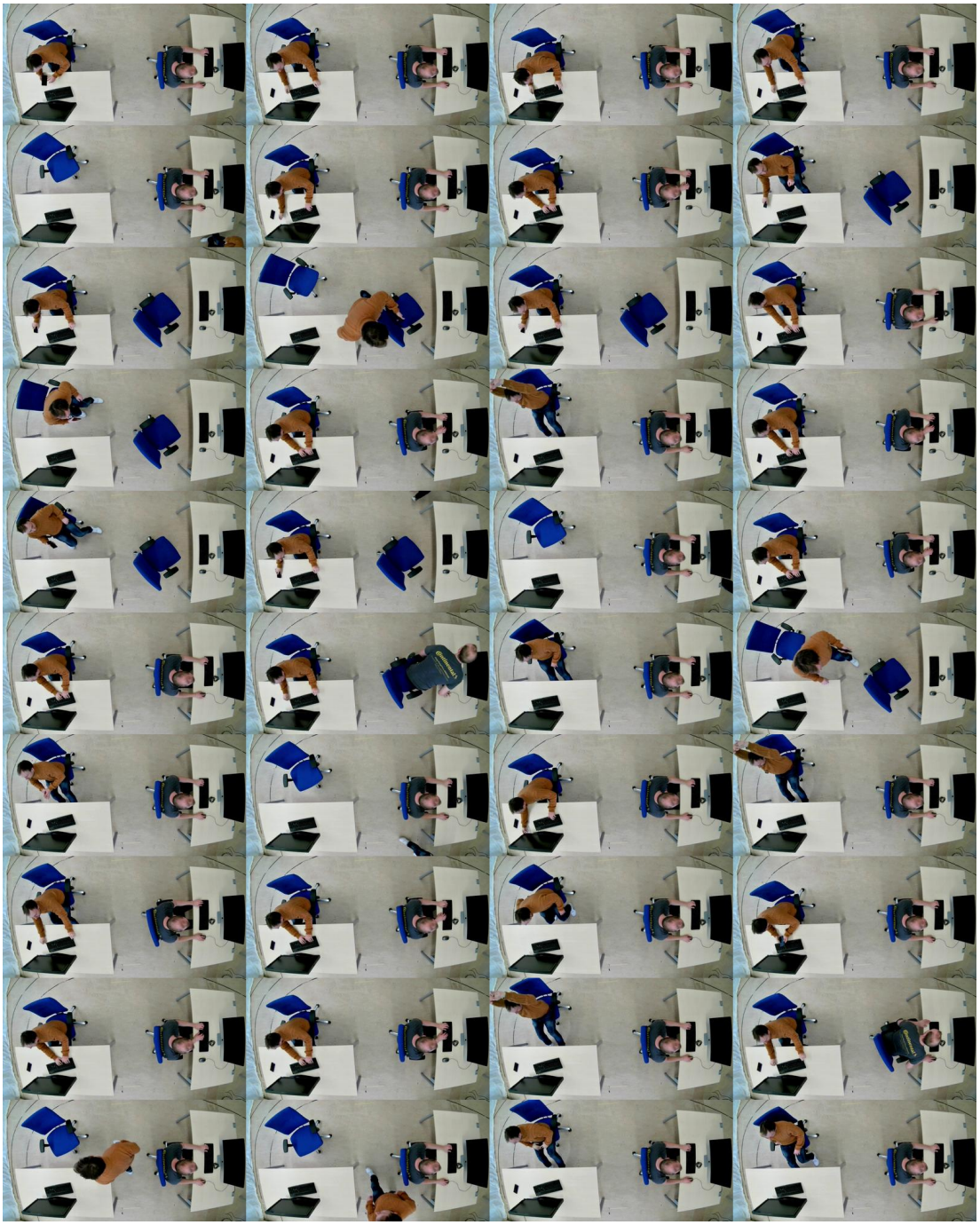


Abbildung C.2: Auf 400 x 200 Pixel herunterskalierte Collage mit Bildern aus der Laboraufzeichnung; $n = 40$

D Appendix: Ergebnisse

Tabelle D.1: Vollständige Ergebnistabelle der Evaluation aggregiert nach Modell

Alias	Modellname	Modellgröße	Accuracy	Recall	Precision	Zeit in ms
Model 1	centernet_hg104_512x512	191,29	0,713	0,716	0,992	44,813
Model 2	centernet_hg104_1024x1024	191,29	0,552	0,564	0,979	120,112
Model 4	centernet_resnet50_v1_fpn_512x512	25,28	0,485	0,498	0,965	11,063
Model 5	centernet_resnet50_v2_512x512	29,85	0,504	0,515	0,973	11,045
Model 6	centernet_resnet101_v1_fpn_512x512	44,30	0,617	0,627	0,965	21,157
Model 7	efficientdet_d0	5,57	0,664	0,664	1,000	49,478
Model 8	efficientdet_d1	8,32	0,740	0,741	0,994	63,228
Model 9	efficientdet_d2	10,00	0,747	0,747	1,000	81,007
Model 10	efficientdet_d3	14,16	0,790	0,791	0,996	111,884
Model 11	efficientdet_d4	23,32	0,759	0,760	0,998	151,698
Model 12	efficientdet_d5	36,75	0,818	0,820	0,993	237,425
Model 13	efficientdet_d6	55,51	0,813	0,815	0,993	316,381
Model 14	efficientdet_d7	55,51	0,821	0,824	0,989	369,757
Model 15	faster_rcnn_inception_resnet_v2_640x640	59,47	0,866	0,891	0,955	205,131
Model 16	faster_rcnn_inception_resnet_v2_1024x1024	60,02	0,863	0,903	0,921	235,933
Model 17	faster_rcnn_resnet50_v1_640x640	28,46	0,825	0,880	0,915	62,257
Model 18	faster_rcnn_resnet50_v1_800x1333	29,19	0,764	0,895	0,827	76,325
Model 19	faster_rcnn_resnet50_v1_1024x1024	28,46	0,773	0,843	0,869	79,534
Model 20	faster_rcnn_resnet101_v1_640x640	47,45	0,865	0,897	0,947	68,097
Model 21	faster_rcnn_resnet101_v1_800x1333	48,18	0,846	0,900	0,906	93,731
Model 22	faster_rcnn_resnet101_v1_1024x1024	47,45	0,867	0,890	0,958	89,086
Model 23	faster_rcnn_resnet152_v1_640x640	63,09	0,873	0,900	0,954	75,970
Model 24	faster_rcnn_resnet152_v1_800x1333	63,82	0,857	0,908	0,920	115,709
Model 25	faster_rcnn_resnet152_v1_1024x1024	63,09	0,864	0,893	0,947	101,063
Model 26	ssd_mobilenet_v1_fpn_640x640	12,08	0,428	0,428	1,000	49,534
Model 27	ssd_mobilenet_v2_320x320	6,07	0,469	0,474	0,964	27,948
Model 28	ssd_mobilenet_v2_fpnlite_320x320	2,63	0,365	0,365	0,980	32,780
Model 29	ssd_mobilenet_v2_fpnlite_640x640	2,63	0,418	0,418	1,000	42,351
Model 30	ssd_resnet50_v1_fpn_640x640	32,84	0,540	0,542	0,994	59,459
Model 31	ssd_resnet50_v1_fpn_1024x1024	32,84	0,626	0,627	0,995	96,007
Model 32	ssd_resnet101_v1_fpn_640x640	51,83	0,562	0,564	0,991	70,336
Model 33	ssd_resnet101_v1_fpn_1024x1024	51,83	0,650	0,651	0,994	112,724
Model 34	ssd_resnet152_v1_fpn_640x640	67,48	0,556	0,558	0,992	80,989
Model 35	ssd_resnet152_v1_fpn_1024x1024	67,48	0,668	0,670	0,993	131,660
Model 36	yolov8n.pt	3,15	0,812	0,869	0,907	17,761
Model 37	yolov8s.pt	11,16	0,847	0,910	0,900	15,690
Model 38	yolov8m.pt	25,89	0,876	0,941	0,913	16,791
Model 39	yolov8l.pt	43,67	0,891	0,940	0,925	24,776
Model 40	yolov8x.pt	68,20	0,887	0,947	0,920	27,631
Model 41	yolov5nu	2,65	0,778	0,850	0,883	22,071
Model 42	yolov5xu	97,23	0,887	0,940	0,924	35,149
Model 43	yolov5s6u	15,29	0,868	0,920	0,920	34,757
Model 44	yolov5m6u	41,19	0,882	0,941	0,921	46,772
Model 45	yolov5l6u	86,05	0,905	0,953	0,936	56,530
Model 46	yolov5x6u	155,48	0,893	0,951	0,926	69,403

Tabelle D.2: Tukey HSD-Paarweiser Vergleich auf statistische Signifikanz

Group 1	Group 2	Mean diff	p-adj	lower	upper	reject
blurred	dark	-0.693	1.0	-6.838	5.452	False
blurred	flipped	-0.779	0.9	-6.924	5.366	False
blurred	greyscale	-0.772	0.9	-6.918	5.372	False
blurred	highCompression	-0.560	1.0	-6.706	5.585	False
blurred	light	-0.905	0.9	-7.051	5.240	False
blurred	midCompression	-0.683	1.0	-6.828	5.462	False
blurred	normal	14.082	0.0	7.9374	20.22	True
dark	flipped	-0.086	1.0	-6.231	6.059	False
dark	greyscale	-0.079	1.0	-6.225	6.065	False
dark	highCompression	0.1327	1.0	-6.012	6.278	False
dark	light	-0.212	1.0	-6.357	5.933	False
dark	midCompression	0.01	1.0	-6.135	6.155	False
dark	normal	14.776	0.0	8.6306	20.92	True
flipped	greyscale	0.0066	1.0	-6.138	6.152	False
flipped	highCompression	0.2189	1.0	-5.926	6.364	False
flipped	light	-0.126	1.0	-6.271	6.019	False
flipped	midCompression	0.0962	1.0	-6.049	6.241	False
flipped	normal	14.862	0.0	8.7169	21.00	True
greyscale	highCompression	0.2123	1.0	-5.933	6.357	False
greyscale	light	-0.132	1.0	-6.278	6.012	False
greyscale	midCompression	0.0896	1.0	-6.055	6.235	False
greyscale	normal	14.855	0.0	8.7102	21.00	True
highCompression	light	-0.344	1.0	-6.490	5.800	False
highCompression	midCompression	-0.122	1.0	-6.268	6.022	False
highCompression	normal	14.643	0.0	8.498	20.78	True
light	midCompression	0.2222	1.0	-5.923	6.367	False
light	normal	14.988	0.0	8.8429	21.13	True
midCompression	normal	14.766	0.0	8.6207	20.91	True

Teil III

Quellenverzeichnis

Literatur

[1] Gordon E Moore. “Cramming more components onto integrated circuits”. In: *Electronics* 38.8 (1965).

[2] Kenneth Flamm. “Measuring Moore’s law: evidence from price, cost, and quality indexes”. In: *Measuring and Accounting for Innovation in the 21st Century*. University of Chicago Press, 2019.

[3] Jega Anish Dev. “Bitcoin mining acceleration and performance quantification”. In: *2014 IEEE 27th Canadian conference on electrical and computer engineering (CCECE)*. IEEE. 2014, S. 1–6.

[4] Saide Isilay Baykal, Deniz Bulut und Ozgur Koray Sahingoz. “Comparing deep learning performance on BigData by using CPUs and GPUs”. In: *2018 Electric Electronics, Computer Science, Biomedical Engineerings’ Meeting (EBBT)*. IEEE. 2018, S. 1–6.

[5] Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani und Nabeel Imhammed Zanoon. “Review on sorting algorithms a comparative study”. In: *International Journal of Computer Science and Security (IJCSS)* 7.3 (2013), S. 120–126.

[6] Daniel J Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern u. a. “Faster sorting algorithms discovered using deep reinforcement learning”. In: *Nature* 618.7964 (2023), S. 257–263.

[7] Marvin Minsky. *Heuristic aspects of the artificial intelligence problem*. Ed. Services Technical Information agency:[Springfield, Va.]: distiributed . . ., 1956.

[8] BF Skinner. *Skinner-Operant Conditioning*. 1953.

[9] S Mohanasundaram, V Krishnan und V Madhubala. “Vehicle theft tracking, detecting and locking system using open CV”. In: *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*. IEEE. 2019, S. 1075–1078.

[10] Yen-Lin Chen, Hsin-Han Chiang, Chuan-Yen Chiang, Chuan-Ming Liu, Shyan-Ming Yuan und Jenq-Haur Wang. "A vision-based driver nighttime assistance and surveillance system based on intelligent image sensing techniques and a heterogamous dual-core embedded system architecture". In: *Sensors* 12.3 (2012), S. 2373–2399.

[11] Alok Prakash, Nirmala Ramakrishnan, Kratika Garg und Thambipillai Srikanthan. "Accelerating computer vision algorithms on heterogeneous edge computing platforms". In: *2020 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE. 2020, S. 1–6.

[12] Göksel Dedeoğlu, Branislav Kisačanin, Darnell Moore, Vinay Sharma und Andrew Miller. "An optimized vision library approach for embedded systems". In: *CVPR 2011 WORKSHOPS*. IEEE. 2011, S. 8–13.

[13] Erik G Learned-Miller. "Introduction to computer vision". In: *University of Massachusetts, Amherst* (2011).

[14] Bhaskar Barua, Clarence Gomes, Shubham Baghe und Jignesh Sisodia. "A self-driving car implementation using computer vision for detection and navigation". In: *2019 International conference on intelligent computing and control systems (ICCS)*. IEEE. 2019, S. 271–274.

[15] Harshit Nigam, Nida Hasib, Mohammad Nabigh Abbas, Mohneesh Tiwari und Himanshu Mali Shalaj. "Facial biometric-based authenticator system: Secure ID". In: *Emerging Trends in IoT and Computing Technologies*. Routledge, 2022, S. 187–192.

[16] Ravi Aron, Shantanu Dutta, Ramkumar Janakiraman und Praveen A Pathak. "The impact of automation of systems on medical errors: evidence from field research". In: *Information systems research* 22.3 (2011), S. 429–446.

[17] MA Erfani Moghaddam und I Konstantzos. "Towards a novel intelligent and fully interactive IoT framework for residential buildings". In: *Journal of Physics: Conference Series*. Bd. 2600. 7. IOP Publishing. 2023, S. 072009.

[18] Helen Stopps und Marianne F Touchie. "Residential smart thermostat use: An exploration of thermostat programming, environmental attitudes, and the influence of smart controls on energy savings". In: *Energy and Buildings* 238 (2021), S. 110834.

[19] Christopher M Bishop. "Pattern recognition and machine learning". In: *Springer google schola* 2 (2006), S. 645–678.

[20] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[21] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[22] Glenn Jocher u. a. *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements*. Version v3.1. Okt. 2020. DOI: 10.5281/zenodo.4154370. URL: https://doi.org/10.5281/zenodo.4154370.

[23] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, Sarah Mack u. a. *Principles of neural science*. Bd. 4. McGraw-hill New York, 2000.

[24] John Canny. "A computational approach to edge detection". In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), S. 679–698.

[25] Frédéric Jurie und Michel Dhome. "A simple and efficient template matching algorithm". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Bd. 2. IEEE. 2001, S. 544–549.

[26] Darren Bagley. *What does it mean to be colorblind?* Zugriff am 26. April 2024. 2020. URL: https://www.canr.msu.edu/news/what-does-it-mean-to-be-colorblind.

[27] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), S. 141–142.

[28] Suzana Herculano-Houzel. "The human brain in numbers: a linearly scaled-up primate brain". In: *Frontiers in human neuroscience* 3 (2009), S. 857.

[29] Max Bielschowsky. *Allgemeine Histologie und Histopathologie des Nervensystems*. Springer, 1910.

[30] Ernst Adolf Spiegel. "Die Zentren des autonomen Nervensystems:(Anatomie· Physiologie und topische Diagnostik)". In: (1928).

[31] Warren S McCulloch und Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), S. 115–133.

[32] Frank Rosenblatt u. a. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Bd. 55. Spartan books Washington, DC, 1962.

[33] Xavier Glorot und Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop und Conference Proceedings. 2010, S. 249–256.

[34] Zijie J Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng und Duen Horng Polo Chau. "CNN explainer: learning convolutional neural networks with interactive visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2020), S. 1396–1406.

[35] Grant Sanderson. *Neural Networks*. Zugriff am: 18. Februar 2024. 2017. URL: https://www.3blue1brown.com/topics/neural-networks.

[36] Anthony Breitzman. "A Modified Strassen Algorithm to Accelerate Numpy Large Matrix Multiplication with Integer Entries". In: (2023).

[37] Heny Pratiwi, Agus Perdana Windarto, S Susliansyah, Ririn Restu Aria, Susi Susilowati, Luci Kanti Rahayu, Yuni Fitriani, Agustiena Merdekawati und Indra Riyana Rahadjeng. "Sigmoid activation function in selecting the best model of artificial neural networks". In: *Journal of Physics: Conference Series*. Bd. 1471. 1. IOP Publishing. 2020, S. 012010.

[38] Yongbin Yu, Kwabena Adu, Nyima Tashi, Patrick Anokye, Xiangxiang Wang und Mighty Abra Ayidzoe. "Rmaf: Relu-memristor-like activation function for deep learning". In: *IEEE Access* 8 (2020), S. 72727–72741.

[39] Yangfan Zhou, Xin Wang, Mingchuan Zhang, Junlong Zhu, Ruijuan Zheng und Qingtao Wu. "MPCE: a maximum probability based cross entropy loss function for neural network classification". In: *IEEE Access* 7 (2019), S. 146331–146341.

[40] Qiqi Li, Ruoxi Wei, Han Sun, Zan Yang, Wei Nai u. a. "Huber Loss Function Based on t-Distribution Yin-Yang-Pair Optimization Algorithm". In: *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*. Bd. 6. IEEE. 2022, S. 1895–1899.

[41] Grant Sanderson. *Neural Networks*. Zugriff am 18. Februar 2024. 2017. URL: https://www.3blue1brown.com/lessons/backpropagation-calculus.

[42] Shuang Wu, Guoqi Li, Lei Deng, Liu Liu, Dong Wu, Yuan Xie und Luping Shi. "$L1$-norm batch normalization for efficient training of deep neural networks". In: *IEEE transactions on neural networks and learning systems* 30.7 (2018), S. 2043–2051.

[43] Andrew Y Ng. "Feature selection, L 1 vs. L 2 regularization, and rotational invariance". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, S. 78.

[44] Ekachai Phaisangittisagul. "An analysis of the regularization between L2 and dropout in single hidden layer neural network". In: *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*. IEEE. 2016, S. 174–179.

[45] Gergely Neu, Gintare Karolina Dziugaite, Mahdi Haghifam und Daniel M Roy. "Information-theoretic generalization bounds for stochastic gradient descent". In: *Conference on Learning Theory*. PMLR. 2021, S. 3526–3545.

[46] P Kingma Diederik. "Adam: A method for stochastic optimization". In: *(No Title)* (2014).

[47] Mark Van der Wilk, Carl Edward Rasmussen und James Hensman. "Convolutional gaussian processes". In: *Advances in neural information processing systems* 30 (2017).

[48] Keunwoo Choi, George Fazekas und Mark Sandler. "Explaining deep convolutional neural networks on music classification". In: *arXiv preprint arXiv:1607.02444* (2016).

[49] Florentin Bieder, Robin Sandkühler und Philippe C Cattin. "Comparison of methods generalizing max-and average-pooling". In: *arXiv preprint arXiv:2103.01746* (2021).

[50] Wei Fang, Lin Wang und Peiming Ren. "Tinier-YOLO: A real-time object detection method for constrained environments". In: *Ieee Access* 8 (2019), S. 1935–1944.

[51] Guohe Zhang, Kepeng Zhao, Bin Wu, Yiqun Sun, Li Sun und Feng Liang. "A RISC-V based hardware accelerator designed for Yolo object detection system". In: *2019 IEEE International Conference of Intelligent Applied Systems on Engineering (ICIASE)*. IEEE. 2019, S. 9–11.

[52] Wangpeng He, Zhe Huang, Zhifei Wei, Cheng Li und Baolong Guo. "TF-YOLO: An improved incremental network for real-time object detection". In: *Applied Sciences* 9.16 (2019), S. 3225.

[53] Ultralytics. *Models - Ultralytics Documentation*. Zugriff am 02. Mai 2024. 2024. URL: https://docs.ultralytics.com/models/.

[54] Joseph Redmon, Santosh Divvala, Ross Girshick und Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, S. 779–788.

[55] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang u. a. "Sparse r-cnn: End-to-end object detection with learnable proposals". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, S. 14454–14463.

[56] Mingming Zhu, Yuelei Xu, Shiping Ma, Shuai Li, Hongqiang Ma und Yongsai Han. "Effective airplane detection in remote sensing images based on multilayer feature fusion and improved nonmaximal suppression algorithm". In: *Remote Sensing* 11.9 (2019), S. 1062.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren und Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, S. 770–778.

[58] Sergey Zagoruyko und Nikos Komodakis. "Wide residual networks". In: *arXiv preprint arXiv:1605.07146* (2016).

[59] Yun Ren, Changren Zhu und Shunping Xiao. "Object detection based on fast/faster RCNN employing fully convolutional architectures". In: *Mathematical Problems in Engineering* 2018 (2018), S. 1–7.

[60] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He und Piotr Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, S. 2980–2988.

[61] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen und Mickey Aleksic. "A quantization-friendly separable convolution for mobilenets". In: *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE. 2018, S. 14–18.

[62] Mingxing Tan, Ruoming Pang und Quoc V Le. "Efficientdet: Scalable and efficient object detection". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, S. 10781–10790.

[63] Munir Majdalawieh, Shafaq Khan und Md T Islam. “Using deep learning model to identify iron chlorosis in plants (October 2022)”. In: *IEEE Access* (2023).

[64] Seung-Taek Kim und Hyo Jong Lee. “Lightweight stacked hourglass network for human pose estimation”. In: *Applied Sciences* 10.18 (2020), S. 6497.

[65] Guido Van Rossum und Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[66] Itseez. *Open Source Computer Vision Library*. https://github.com/itseez/opencv. 2015.

[67] P Umesh. “Image Processing in Python”. In: *CSI Communications* 23 (2012).

[68] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), S. 90–95. DOI: 10.1109/MCSE.2007.55.

[69] Charles R. Harris u. a. “Array programming with NumPy”. In: *Nature* 585.7825 (Sep. 2020), S. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[70] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[71] cvzone. *cvzone: Computer Vision Library*. GitHub repository. 2023. URL: https://github.com/cvzone/cvzone.

[72] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[73] Francois Chollet u. a. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[74] Glenn Jocher u. a. *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements*. Version v3.1. Okt. 2020. DOI: 10.5281/zenodo.4154370. URL: https://doi.org/10.5281/zenodo.4154370.

[75] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga und Adam Lerer. “Automatic differentiation in PyTorch”. In: *NIPS-W*. 2017.

[76] COCO Consortium. *COCO API: Tools for loading, parsing, and visualizing the annotations in COCO*. GitHub repository. 2023. URL: https://github.com/cocodataset/cocoapi/tree/master.

[77] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r und C. Lawrence Zitnick. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[78] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár und C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, S. 740–755.

[79] Alexander Kirillov u. a. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV].

[80] Chaoning Zhang, Dongshen Han, Yu Qiao, Jung Uk Kim, Sung Ho Bae, Seungkyu Lee und Choong Seon Hong. "Faster Segment Anything: Towards Lightweight SAM for Mobile Applications". In: *arXiv preprint arXiv:2306.14289* (2023).

[81] Glenn Jocher. *Ultralytics YOLOv5*. Version 7.0. 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

[82] Glenn Jocher, Ayush Chaurasia und Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics.

[83] Chad F Baker, Christopher J Rudnisky, Matthew TS Tennant, Paul Sanghera, Bradley J Hinz, Alexander R De Leon und Mark DJ Greve. "JPEG compression of stereoscopic digital images for the diagnosis of diabetic retinopathy via teleophthalmology". In: *Canadian journal of ophthalmology* 39.7 (2004), S. 746–754.

[84] John-Dylan Haynes, R Beau Lotto und Geraint Rees. "Responses of human visual cortex to uniform surfaces". In: *Proceedings of the National Academy of Sciences* 101.12 (2004), S. 4286–4291.

[85] Edward H Adelson. "Checkershadow illusion. 1995". In: *URL http://web. mit. edu/persci/people/adelson/checkershadow_illusion. html* (2005).

[86] *Checker Shadow Illusion*. https://en.wikipedia.org/wiki/File:Checker_shadow_illusion.svg. Zugriff am 13. April 2024.

[87] Gregory K Wallace. "The JPEG still picture compression standard". In: *Communications of the ACM* 34.4 (1991), S. 30–44.

[88] Jianlong Zhou, Amir H Gandomi, Fang Chen und Andreas Holzinger. "Evaluating the quality of machine learning explanations: A survey on methods and metrics". In: *Electronics* 10.5 (2021), S. 593.

[89] R Artusi, P Verderio und EJTIjobm Marubini. "Bravais-Pearson and Spearman correlation coefficients: meaning, test of hypothesis and confidence interval". In: *The International journal of biological markers* 17.2 (2002), S. 148–151.

[90] Anita Nanda, Bibhuti Bhusan Mohapatra, Abikesh Prasada Kumar Mahapatra, Abiresh Prasad Kumar Mahapatra und Abinash Prasad Kumar Mahapatra. "Multiple comparison test by Tukey's honestly significant difference (HSD): Do the confident level control type I error". In: *International Journal of Statistics and Applied Mathematics* 6.1 (2021), S. 59–65.

[91] Alex Mackin, Fan Zhang und David R Bull. "A study of subjective video quality at various frame rates". In: *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2015, S. 3407–3411.

[92] Dillon Reis, Jordan Kupec, Jacqueline Hong und Ahmad Daoudi. "Real-time flying object detection with YOLOv8". In: *arXiv preprint arXiv:2305.09972* (2023).

[93] Tameru Hailesilassie. "Rule extraction algorithm for deep neural networks: A review". In: *arXiv preprint arXiv:1610.05267* (2016).

[94] Udo Seiffert. "Training of large-scale feed-forward neural networks". In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. IEEE. 2006, S. 5324–5329.

[95] Yiding Li, Shunsheng Zhang und Wen-Qin Wang. "A lightweight faster R-CNN for ship detection in SAR images". In: *IEEE Geoscience and Remote Sensing Letters* 19 (2020), S. 1–5.

[96] Arun Narenthiran Veeranampalayam Sivakumar, Jiating Li, Stephen Scott, Eric Psota, Amit J. Jhala, Joe D Luck und Yeyin Shi. "Comparison of object detection and patch-based classification deep learning models on mid-to late-season weed detection in UAV imagery". In: *Remote Sensing* 12.13 (2020), S. 2136.

[97] Hyoseong Choi, Jiwon Lee, Jeonghoon Choi und Won Woo Ro. "Analysis of DRAM-based Network of DRAM Swap Space Adopting Latency Hiding Technique". In: *2022 37th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. IEEE. 2022, S. 239–242.

[98] Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin und Hongkai Xiong. "Deep neural network compression with single and multiple level quantization". In: *Proceedings of the AAAI conference on artificial intelligence*. Bd. 32. 1. 2018.

[99] NVIDIA, Péter Vingelmann und Frank H.P. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: https://developer.nvidia.com/cuda-toolkit.

[100] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro und Evan Shelhamer. "cudnn: Efficient primitives for deep learning". In: *arXiv preprint arXiv:1410.0759* (2014).

[101] Ross Smith. "Performance of MPI Codes Written in Python with NumPy and mpi4py". In: *2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*. IEEE. 2016, S. 45–51.

[102] Yen-Sok Poon, Chih-Chun Lin, Yu-Hsuan Liu und Chih-Peng Fan. "YOLO-based deep learning design for in-cabin monitoring system with fisheye-lens camera". In: *2022 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2022, S. 1–4.

[103] Bipul Neupane, Teerayut Horanont und Jagannath Aryal. "Real-time vehicle classification and tracking using a transfer learning-improved deep learning network". In: *Sensors* 22.10 (2022), S. 3813.

[104] Hanting Chen, Yunhe Wang, Chang Xu, Chao Xu und Dacheng Tao. "Learning student networks via feature embedding". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2020), S. 25–35.

[105] Joseph Redmon und Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, S. 7263–7271.

[106] Joseph Redmon und Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[107] Alexey Bochkovskiy, Chien-Yao Wang und Hong-Yuan Mark Liao. “Yolov4: Optimal speed and accuracy of object detection”. In: *arXiv preprint arXiv:2004.10934* (2020).

[108] Glenn Jocher. *Ultralytics YOLOv5*. Version 7.0. 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

[109] Chuyi Li, Lulu Li, Yifei Geng, Hongliang Jiang, Meng Cheng, Bo Zhang, Zaidan Ke, Xiaoming Xu und Xiangxiang Chu. *YOLOv6 v3.0: A Full-Scale Reloading*. 2023. arXiv: 2301.05586 [cs.CV].

[110] Chien-Yao Wang, Alexey Bochkovskiy und Hong-Yuan Mark Liao. “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors”. In: *arXiv preprint arXiv:2207.02696* (2022).

[111] Glenn Jocher, Ayush Chaurasia und Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics.