DLM Semesterprojekt: Bestimmung der 3D-Boundingbox eines Roboters

GruppeNR: 1, Namen: Tim Eisenacher (1870755) und Paul Manea(1870391)
EMail: 1870755@stud.hs-mannheim.de

Abstract

The motivation of this project is the 3D bounding box estimation of a roboter using deep learning algorithms. Large groups of patients can benefit from the great potential of modern computer vision , for example in radiological diagnostics or in automated surgery. Due to increasingly high-resolution images and increasing data volumes, Convolutional Neural Networks (CNN) are particularly well suited to solve this problem due to their high performance. Therefore, the YOLO (You Only Look Once) network architecture as a state of the art real-time object detection system is introduced in this work. With small architectural modifications and a custom loss function, YOLO's high performance and accuracy in solving the problems of this work can be demonstrated. For this purpose, the object recognition was first implemented for a 2D problem and then extended to recognize the robot in a 3D to 2D projection. All in all it is determined that the 3D bounding box estimation can significantly benefit from using YOLO as a fast and accurate bounding box regression system. Also the YOLO implementation is well suited for real-time image processing.

1 Einleitung

Objekterkennung ist einer der vielversprechendsten und am stärksten im Forschungsfokus stehenden Bereiche von *Computer Vision* und *Machine Learning* [10]. Besonders im Bereich der medizinischen und industriellen Innovationen konnten dadurch bereits jetzt schon große Fortschritte erzielt werden [15, 9]. So können beispielhaft Operationsroboter stark von einer auf *Deep-Learning*-Algorithmen basierenden Erkennung und Lokalisation der Operationsinstrumente profitieren [14]. Auch in der Tumordiagnostik kann so die Erkennungsgenauigkeit gegenüber einer menschlichen Klassifikation bei gleichzeitig signifikant niedrigeren Kosten verbessert werden [2].
Die Anforderungen an die Bildverarbeitung sind dabei in diesen Bereichen besonders groß. Zum einen sorgen immer hochauflösendere Bilder für enorme Datenmengen und damit Hardwareanforderungen. Zum anderen ist die Anzahl an möglichen Klassen und damit Featurekombinationen bei der Objekterkennung enorm. Aufgrund ihrer hohen Trainingsperformanz und der Fähigkeit, herausragend effizient Features in Bildern zu erkennen, eignen sich *Convolutional-Neural-Networks (CNN)* besonders gut als Lösungsansatz der beschriebenen Probleme [10].
In der vorliegenden Arbeit wird ein *Deep-Learning*-Modell zur Bestimmung der *3D-Bounding-Box* eines Roboters implementiert und evaluiert. Dafür erfolgt anfangs die Abgrenzung und grobe Erläuterung des *Deep-Learnings* (DL) im Kontext des *Machine-Learnings* (ML). Der Fokus liegt dabei auf dem verwendeten *DL*-Konzept der CNN's und damit verbundener Probleme und Herausforderungen. Anschließend erfolgt eine Abhandlung der für die Implementierung verwendeten Netzstruktur und Metriken. Die Implementierung wird zunächst zur Lösung eines zweidimensionalen Problems erstellt und dann anschließend auf drei Dimensionen erweitert. Die Entwicklung erfolgt in Python 3.7 unter Verwendung des Tensorflow Frameworks in der Spider IDE. Die Evaluation der Implementierung geschieht anhand eines Vergleiches der geschätzten *Bounding-Box* mit der gelabelten *Bounding-Box* der Testdaten. Als Datengrundlage für Training und Evaluation dient ein synthetisch generierter und vorgelabelter Datensatz von RGB-Bildern, welcher vom Institut für eingebettete Systeme und Medizintechnik zur Verfügung gestellt wird.

2 Grundlagen und Stand der Technik

Salvaris beschreibt das *Machine Learning* (ML) als einen Zweig der Computerwissenschaften, bei dem Computern beigebracht wird anhand von Trainingsdaten Entscheidungen zu treffen. Typische Anwendungsgebiete des ML sind Klassifikation, Regression und Clustering. Das *Deep Learning* (DL) ist ein Teilgebiet des ML (Abb. 1) bei dem besonders komplexe neuronale Netze mit vielen Schichten und Neuronen Verwendung finden. Eine weitere wesentliche Abgrenzung stellt die Merkmalsextraktion dar. Also die Extraktion der Eigenschaften eines Objekts, die ausschlaggebend für etwaige Klassenzugehörigkeiten sind. Diese entscheidenden Merkmale müssen dem DL-Modell nicht vorgegeben werden, sondern werden von dem Algorithmus selbst gefunden. Dieser Umstand stellt mit die größten Herausforderungen aber auch Chancen beim DL dar [7, S.32-47]. Im Folgenden werden nur die für die vorliegende Arbeit besonders relevanten und speziell angepassten Methoden und Aspekte des DL erläutert. Für weiterführende grundlegende Informationen zum Beispiel zu Neuronen, Schichttypen, Aktivierungsfunktionen und zum Gradientenabstiegsverfahren wird auf [5] verwiesen.

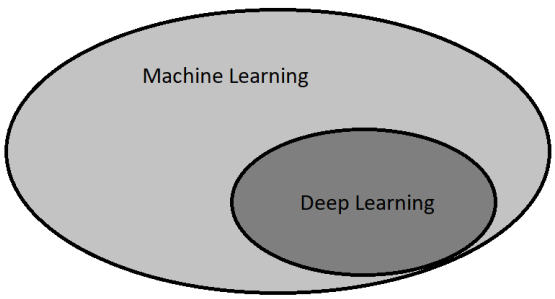

Abbildung 1: Abgrenzung Deep Learning zu Machine Learning.

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) sind eine spezielle Art von neuronalen Netzen, die sich für das verarbeiten von gitterartig beschaffenen Daten eignen. Hierzu zählen zum Beispiel auch Bilddaten, deren Pixelraster sich als Gitter oder Matrix interpretieren lassen. Ein typisches CNN besteht dabei aus einem oder mehreren Paaren von *Convolutional-* und *Pooling-Layern*, gefolgt von einem oder mehreren *Fully-Connected-Layern*. Die Folgenden Darstellungen richten sich im wesentlichen nach Goodfellow [4, S.326-366]

Convolutional Layer Bei einem *Convolutional Layer* wird schrittweise ein Filterkernel K über eine Eingabematrix I mit den Dimensionen n und m bewegt (Abb. 2). Der Input der folgenden Neuronen $S(i, j)$ berechnet sich dann aus einer Faltungsoperation der jeweils übereinanderliegenden Kernel- und Bildelemente (Gleichung 1).

$$S(i,j) = (I * K)(i,j) = \sum_n \sum_n I(i-m, j-n)K(m,n) \tag{1}$$

Der so berechnete Input eines Neurons wird anschließend abhängig von der verwendeten Aktivierungsfunkton in den Output verwandelt. Zu bemerken ist, dass alle Neuronen eines *Convolutional Layers* die gleichen Gewichte haben (sog. *Parameter Sharing*). Dadurch ist es möglich Speicher gegenüber anderen Netzstrukturen einzusparen, die häufig eine große Gewichtungsmatrix verwenden. Ein weiterer großer Vorteil sind die sog. *Sparse Interarctions*. Durch die Verwendung eines Filterkernels der meist nur einen Bruchteil der Größe des zu analysierenden Bildes aufweist, werden nur die Features extrahiert, die wirklich entscheidend sind für die Zugehörigkeit zu einer Klasse. Dies führt ebenso zu einer weiteren Speicher- und Performanzoptimierung.

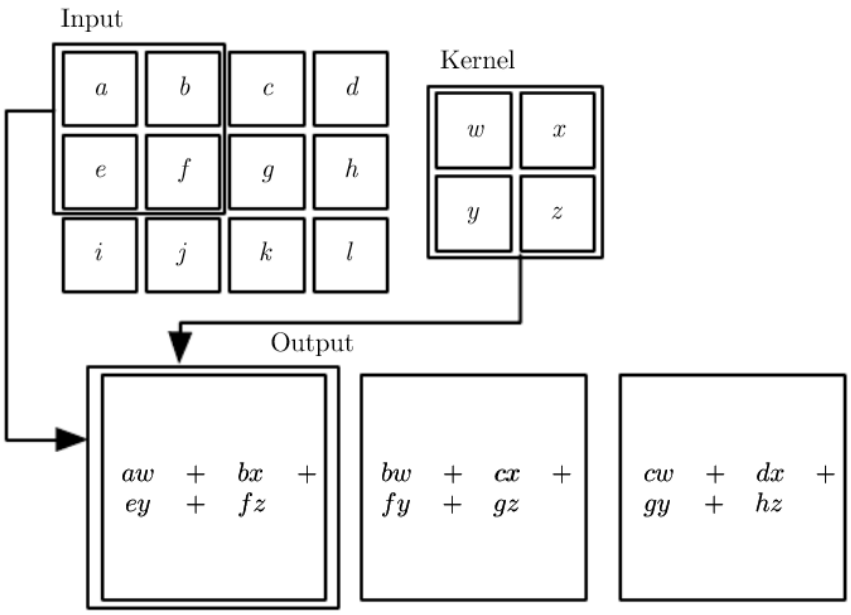

Abbildung 2: Prinzip eines Convolutional Layers [4, S.330].

Pooling Layer *Pooling Layer* sorgen dafür, dass Features einer Klasse in einem Bild nahezu ortsinvariant gelernt werden können. Ein weitverbreitetes Pooling Verfahren ist das sog. $2X2$ *Max-Pooling*, bei dem aus jedem $2X2$ Quadrat der Neuronen des *Convolutional-Layers* nur das aktivste Neuron an die nächste Schicht weitergeleitet wird. Abbildung 3 verdeutlicht dieses Funktionsprinzip. Es werden von den jeweils benachbarten Neuronen nur die mit den höchsten Gewichten an die nächste Schicht durchgeschaltet.

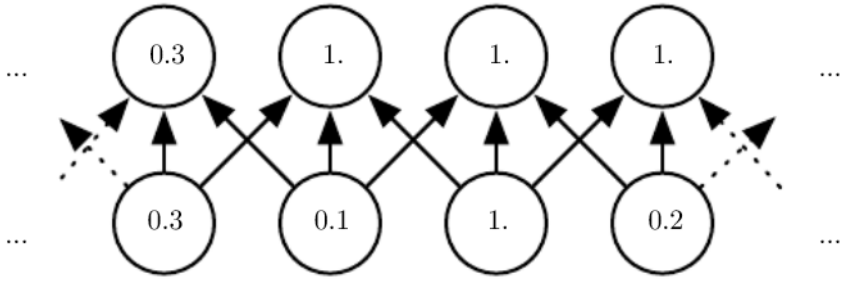

Abbildung 3: Prinzip eines Pooling Layers [4, S.337].

Fully Connected Layer *Fully-Connected-Layer* oder in *Keras* sog. *Dense-Layer* stellen einen Schichtentyp dar, bei dem jedes Neuron mit jeweils jedem Neuron der vorigen Schicht verschaltet ist. Es ist so möglich, die Ausgaben des letzten *Pooling-Layers* über ein oder mehrere *Fully-Connected-Layer* mithilfe von Aktivierungsfunktionen zum Beispiel in eine Wahrscheinlichkeitsverteilung der Klassenzugehörigkeit zu überführen. Die Anzahl Neuronen in der letzten Schicht entspricht dann der Anzahl zu lernender Klassen oder auch der Anzahl vorherzusagender Features.

2.2 Loss-Funktionen und Metriken

DL Netze optimieren ihre Gewichte und Neuronenaktivitäten während des Trainings selbst durch einen Vergleich der geschätzten Ergebnisse y_{pred} mit den entsprechenden Zielgrößen y_{target}. Dies geschieht über sog. *Loss-Functions*. Dabei zeigen diese Funktionen bei großen Abweichung von den Zielwertebereichen typischerweise auch hohe Werte [4, S.271-279]. Ein weit verbreitetes Beispiel hierfür ist der mittlere quadratische Fehler (*Mean-Squared-Error (MSE)*), der häufig als Maß zur Evaluation des Trainingserfolges eingesetzt wird. Der MSE berechnet sich entsprechend Gleichung 2. Die MSE-Loss-Funktion wird auch als L2-Loss bezeichnet. Diese Möglichkeit zur Beurteilung des Erfolges eines DL-Modells wird in Keras auch als Metrik bezeichnet. Als Metriken werden meist ebenso die beschriebenen Loss-Funktionen verwendet. Metriken haben einen rein informativen Zweck und werden nicht direkt für das Training verwendet [1]. Loss-Funktionen spiegeln so einen entscheidenden Faktor für ein erfolgreiches Training wieder. Dabei sollte je nach Anwendungsfall individuell eine passende und sinnvolle Loss-Funktion zur Validierung gewählt werden. Ein ausführliche Abhandlung hierüber findet sich in [5] und [7].

$$MSE = \sum_{n} \frac{(y_{pred} - y_{target})^2}{n} \tag{2}$$

2.3 Stand der Technik

In der Literatur werden viele Möglichkeiten zur Objekterkennung mittels DL beschrieben. Laut Zhao haben sich aktuell jedoch drei Hauptverfahren in der Industrie etabliert [17]. *Fast-Region-Based-CNNs* ermöglichen gute Detektionsergebnisse, sind aber relativ rechenaufwendig [3]. Das *You Only Look Once* Modell (YOLO) erreicht eine höhere Performanz bei verbesserter Präzision [12]. Ein weiteres Modell zur Lösung des Problems ist das von Liu vorgestellte *Single-Shot-Detection* (SSD) Modell vor. Beim SSD können Performanz und Genauigkeit zu Lasten einer komplexeren Netzarchitektur noch weiter verbessert werden [6]. Die Gemeinsamkeit aller vorgestellten Modelle besteht in der Verwendung von *Convolutional Layern*.

3 Methoden

Zu Lösung der Aufgabenstellung wird eine modifizierte Version der sogenannten YOLO (*You Only Look Once*) Netzstruktur in Python unter Nutzung der Keras API implementiert. YOLO Netze zeichnen sich durch gute Detektionsergebnisse bei einer sehr hohen Trainings- und Klassifikationsperfomanz aus. So ist mithilfe des YOLO-Modells sogar auf vergleichsweise kostengünstiger Hardware eine Echtzeitschätzung der Bounding-Boxen von Objekten möglich [12].

3.1 Lossberechnung und Intersection over Union

Rezatofighi1 beschreibt die *Intersection over Union* (IOU) als eine performante und präzise Möglichkeit den Loss bei der Objekterkennung zu bestimmen (s. Kap 2.2). Dabei wird die IOU aus dem Quotienten der sich überlappenden Fläche *(Intersection / Overlap)* und der gemeinsam gebildeten Fläche (*Union*) zweier Regionen berechnet:

$$IOU(y_{target}, y_{pred}) = \frac{y_{target} \cap y_{pred}}{y_{target} \cup y_{pred}} \tag{3}$$

y_{pred} steht hierbei für die geschätzte und y_{target} für die reale Bounding-Box der Objekte. Die graphische Interpretation der Berechnung wird in Abbildung 4 deutlich. Zur Berechnung der jeweiligen Flächen wird der von Rezatofighil in [13] beschriebene Algorithmus implementiert. Dort ist zusätzlich eine hier nicht genutzte Erweiterung zur *Generalized-IOU* beschrieben.

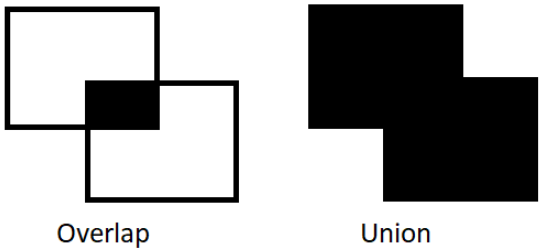

Abbildung 4: Intersection over Union

Somit ergibt sich für den 2D-Fall als resultierende Loss-Funktion:

$$L_{2D} = 1 - IOU(y_{target}, y_{pred}) \tag{4}$$

Es ist darauf hinzuweisen, dass aufgrund der hohen Komplexität auf die Erweiterung der IOU Berechung für den 3D-Fall im Rahmen dieser Arbeit verzichtet wird. Möglich Ansätze für 3D-IOU Berechungen finden sich jedoch in [16] und [8]. Im 3D-Fall wird der Loss lediglich über den in Kapitel 2.2. beschriebenen L2-Ansatz bestimmt. In Abbildung 5 ist schematisch dargestellt wie der L2 Loss berechnet wird. Hierfür wird lediglich der Abstand zwischen einem, vom Netz prädizierten Punkt und einem Zielpunkt berechnet. Dieser Abstand etspricht dem Ausdruck $y_{target} - y_{pred}$ aus Formel 2.

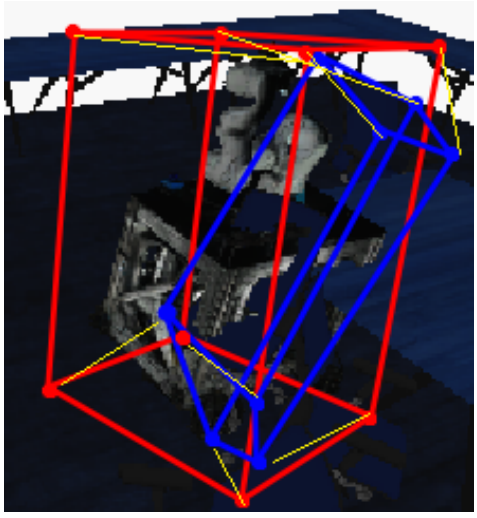

Abbildung 5: Prädizierte und gelabelte Punkte der Bounding-Box mit Prädiktonsfehler (Gelbe Linien)

3.2 Netzstruktur

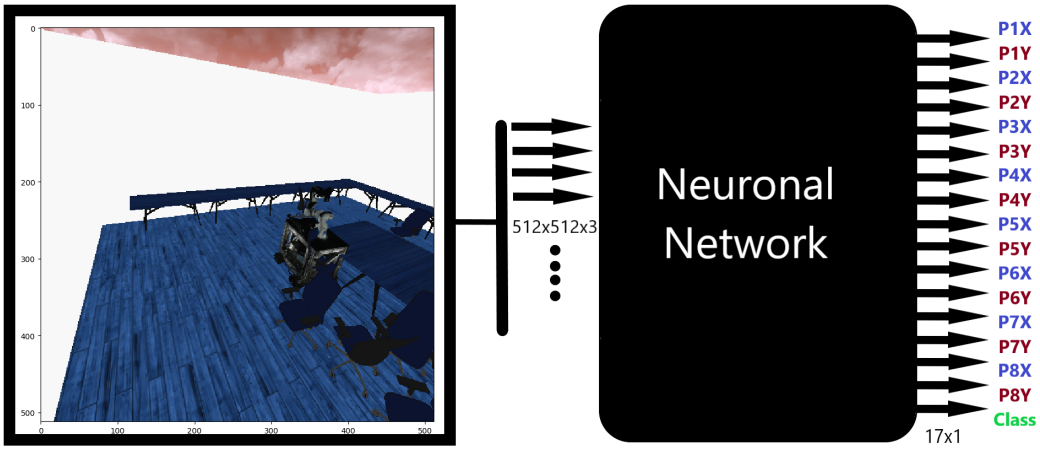

Abbildung 6: Vereinfachte Darstellung der Netzstruktur zur Prädiktion einer 3D-Bounding-Box

Die Netzstruktur, welche für die Prädiktion einer 3D-Bounding-Box verwendet wird orientiert sich sehr stark an dem sogenannten YOLO Netz. Die ersten Schichten des Netzes bestehen aus *Convolutional Layern*, welche die Aufgabe haben, Features aus dem Bild zu extrahieren. Die hintere Schicht des Netzes besteht aus *Fully Connected Layern*, welche die Koordinaten der Bounding-Box Punkte sowie die Klassenzugehörigkeit ausgeben [12].
Die YOLO Netzarchitektur muss jedoch für die Prädiktion einer projizierten 3D Bounding-Box angepasst werden. Diese besitzt im Gegensatz zu einer 2D-Bounding-Box 8 anstatt 2 erforderlichen X/Y-Bildpunkten. Ein Vergleich einer 2D und einer 3D-Bounding-Box ist in Abbildung 7 dargestellt. Aus den 8 Punkten ergeben sich 16 Koordinaten. Zu den 16 Koordinaten kommt ein weiterer Parameter, welcher die Klasse eines Objektes angibt. Diese Angabe wird implementiert, um das Netz modular und skalierbar zu halten, ist jedoch für die Lösung dieser Problemstellung nicht zwingend notwendig, da es nur die eine Klasse *Roboter* gibt. In Abbildung 6 ist eine vereinfachte Darstellung des implementierten Netzes als Blackbox aufgezeigt. Zu sehen sind die 17 Ausgänge, welche aus 16 Koordinaten für die Punkte der 3D-Bounding-Box, sowie der Klassenzugehörigkeit bestehen.

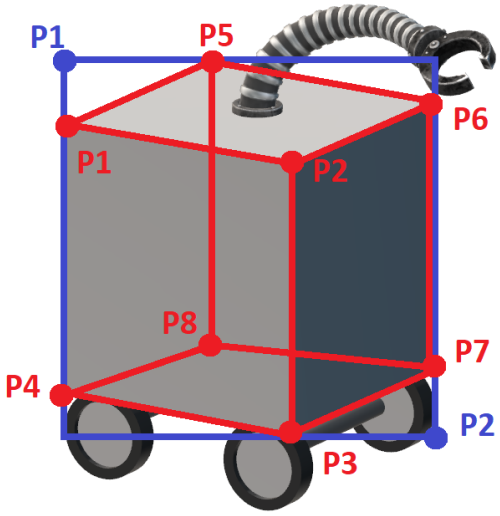

Abbildung 7: 3D-Bounding-Box mit 8 Punkten (rot) und im Vergleich dazu eine 2D-Bounding-Box mit 2 Punkten (blau)

3.3 Training

Für das Training wird ein synthetischer vorgelabelter Datensatz verwendet, welcher vom Institut für eingebettete Systeme und Medizintechnik der Hochschule Mannheim bereitgestellt wird. Es liegen insgesamt 2000 gelabelte Samples vor, von denen 90% für das Training und 10% für die Validierung verwendet werden. Alle weiteren Trainingsparameter sind Tabelle 1 zu entnehmen. Diese optimalen Hyperparameter wurden dabei in mehreren Trainingsexperimenten empirisch ermittelt.

Tabelle 1: Trainingsparameter

Parameters	**Values**
Convolutional Layer	9
Fully connected Layer	2
Anzahl der Trainingssamples	1800
Epochen	10
Batch Size	1
Optimizer	Adam
Learning Rate	0,0001
Loss Function	MSE (siehe 3.1)

Auf einer NVIDIA GTX960M Grafikkarte mit einer *NVIDIA Compute Capability* von 5, dauert das Training des Netzwerks mit den in Tabelle 1 beschriebenen Parametern lediglich 25 Minuten. Diese relativ geringe Trainingszeit ist auf die hohe Performanz der YOLO Netzstruktur zurückzuführen.

3.4 Beschreibung der Softwaremodule

Für die Realisierung eines Netzwerkes zur 3D-Bounding-Box-Schätzung werden mehrere Softwaremodule implementiert und Im Folgenden beschrieben.

3.4.1 Laden der Daten

Das Laden der Trainingsdaten wird durch eine eigens dafür entwickelte Klasse realisiert. Die Bilder liegen im .png Format und die Labels im .json Format vor. Die Klasse lädt diese Daten, teilt diese in Trainings- und Testdaten auf und gibt die Daten als *numpy arrays* zurück, sodass diese direkt für das Training verwendet werden können. In Abbildung 8 ist das zu der Klasse gehörige UML Diagramm mit allen Methoden dargestellt.

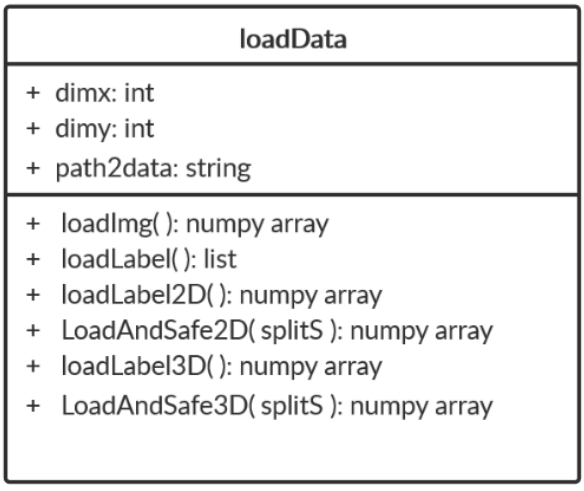

Abbildung 8: UML Diagramm loadData

3.4.2 Objekt Lokalisierung

In der Klasse *ObjectLocalizer3D* befindet sich der für die Objekterkennung und Bounding-Box-Schätzung relevante Programmcode. Hier wird unter anderem das Netz erzeugt und die Loss-Funktion definiert. Die Klasse ist abgeleitet von der *Keras functional API*. In dieser Klasse werden auch vorimplementierte Methoden der *Keras functional API* aufgerufen oder überladen. Ein Beispiel dafür ist die *fit*-Funktion. Ebenfalls ist es möglich die trainierten Modelle zu speichern und diese wieder zu laden. In Abbildung 9 wird ein UML-Klassendiagramm des *ObjectLocalizer3D* dargestellt.

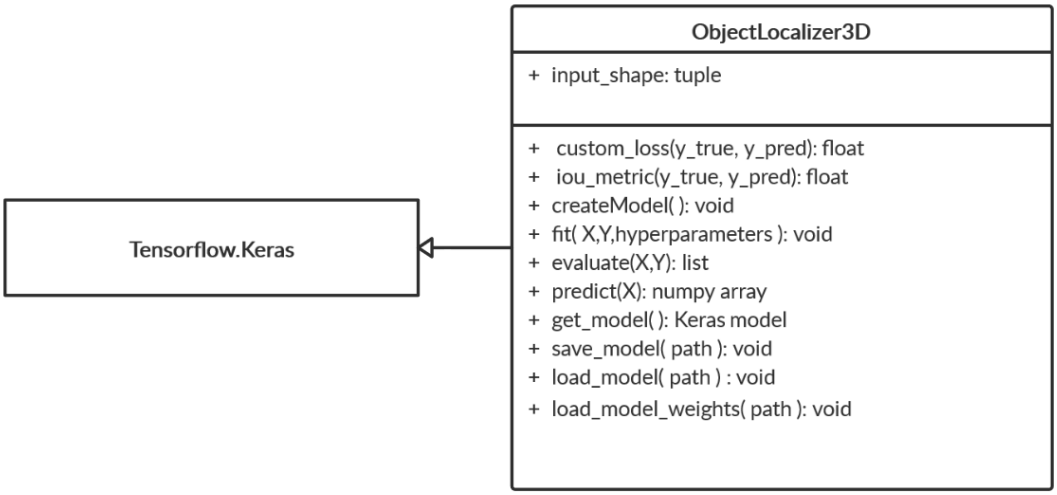

Abbildung 9: UML Diagramm der Klasse ObjectLocalizer3D (abgeleitet von *Keras Functional API*)

4 Ergebnisse

Die in Abschnitt 3.2 beschriebene Netzstruktur wird mit dem in Abschnitt 3.3 vorgestelltem Datensatz und Parametern trainiert und validiert. Die optimalen Trainingsparameter wurden dabei aus mehren Experimenten empirisch ermittelt. Dabei dient die in Abschnitt 3.1 erarbeitete *Loss*-Metrik zur jeweiligen Quantisierung des Trainingserfolges.
Abbildung 10 zeigt den Verlauf des *Loss* jeweils für die Trainings- und Validationsdaten über die Trainingsepochen im 3D-Fall. Dabei fällt dieser zunächst relativ schnell und nähert sich dann ungefähr $1 * 10^{-3}$ an. Dies deutet auf ein erfolgreiches Training ohne *Overfitting* hin. In Abbildung 11 und 12 werden beispielhaft zwei prädizierte Bounding-Boxen der Testdaten dargestellt (blau). Zum Vergleich sind zusätzlich die manuell vorgelabelten Bounding-Boxen eingezeichnet (rot). Oben im Bild steht der jeweils errechnete *Loss*. Es ist zu erkennen, dass das rechte Bild mit dem höchsten *Loss* auch die sichtbar größte Abweichung der prädizierten Bounding-Box mit der vorgelabelten aufweist. Der *Loss* im linken Bild ist relativ gering und spiegelt so auch die sichtbar geringe örtliche Abweichung der prädizierten Bounding-Box zur vorgelabelten Bounding-Box wieder.

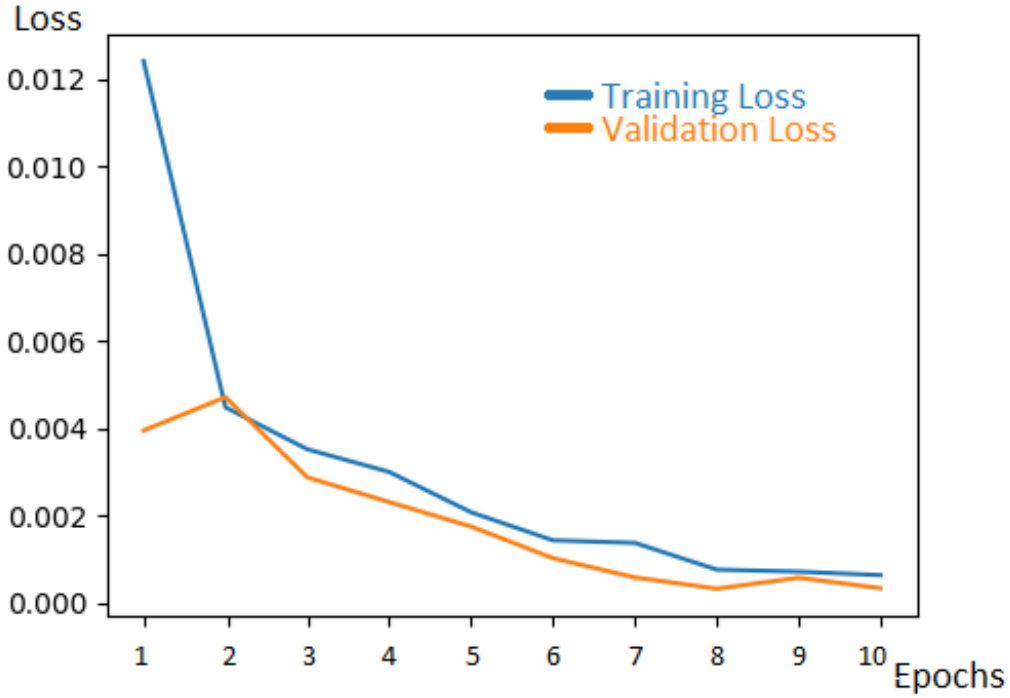

Abbildung 10: Trainings- und Validations-Loss über die Trainingsepochen

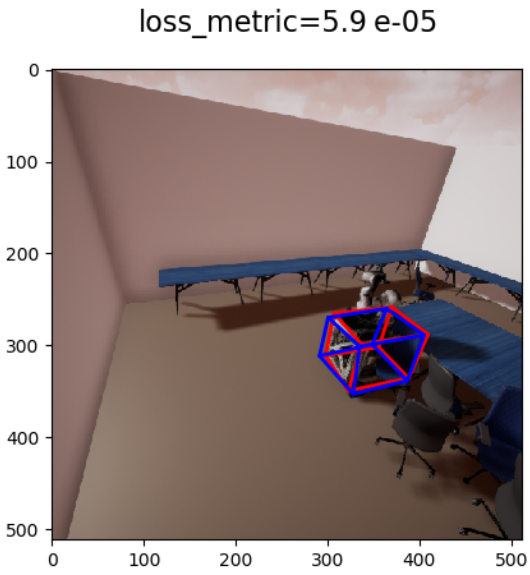

Abbildung 11: Beispiel einer gut prädizierten Bounding-Box mit geringem Loss. Zu sehen ist die gelabelte Bounding-Box (rot), sowie die prädizierte Bounding-Box (blau).

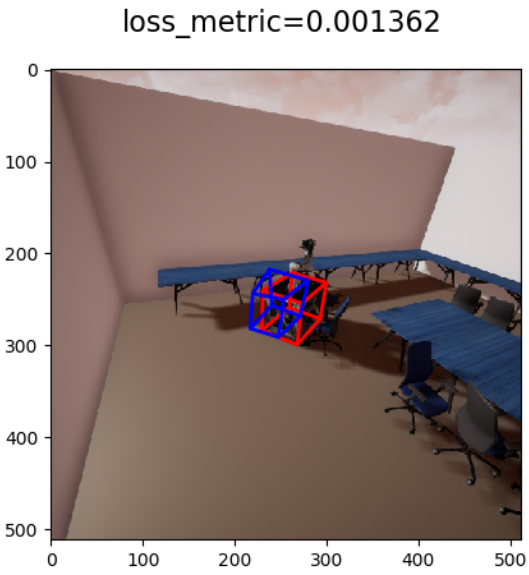

Abbildung 12: Beispiel einer schlecht prädizierten Bounding-Box mit hohem Loss. Hier könnte speziell der Stuhl die Schätzung negativ beeinflusst haben.

5 Fazit und Ausblick

In der vorliegenden Arbeit wurde erfolgreich ein *Deep-Learning* Netzwerk zur Detektion einer 3D Bounding-Box trainiert und validiert. Als Netzstrukur dient dabei eine modifizierte Version des *YOLO*-Netzes. Zur Validierung im 2D-Fall dient die *IOU*-Metrik. Im 3D-Fall wird der *Loss* mittels der L2-Metrik ermittelt. Die optimalen Hyperparameter sind empirisch aus verschiedenen Experimenten hervorgegangen. Zusätzlich ergeben sich vielversprechende Anknüpfungspunkte für weitere Arbeiten. Atiqur beschreibt zum Beispiel eine Möglichkeit die IOU Berechung im 2D-Fall noch weiter zu optimieren in [11]. Weiterhin wäre es interessant die IOU auch für die Berechnung der Loss Funktion im 3D Fall zu benutzten. Möglich Ansätze für 3D-IOU Berechungen finden sich in [16] und [8]. Eine weitere Möglichkeit zur Verbesserung der Trainingsergebnisse liegt in der Augmentation der Daten. Darauf wird im Rahmen dieser Arbeit jedoch verzichtet.

6 Literatur und Bilder

Literatur

[1] François Chollet et al. Keras. `https://keras.io`, 2015.

[2] Angel Cruz-Roa, Hannah Gilmore, Ajay Basavanhally, Michael Feldman, Shridar Ganesan, Natalie N.C. Shih, John Tomaszewski, Fabio A. González, and Anant Madabhushi. Accurate and reproducible invasive breast cancer detection in whole-slide images: A deep learning approach for quantifying tumor extent. *Scientific Reports*, 7(1), apr 2017.

[3] Ross Girshick, Microsoft Research, and rbg@microsoft.com. Fast r-cnn. In *arXiv:1504.08083v2*, 2015.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning. Das umfassende Handbuch: Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze (mitp Professional)*. mitp Professionals, 2018.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *arXiv:1512.02325v5*, 2016.

[7] Wee Hyong Tok Mathew Salvaris, Danielle Dean. *Deep Learning mit Microsoft Azure*. Rheinwerk Computing, 2019.

[8] Arsalan Mousavian, Dragomir Anguelov, John Flynn, George Mason University Zoox, Inc. Zoox, and Inc. 3d bounding box estimation using deep learning and geometry. In *arXiv:1612.00496v2*, 2017.

[9] Ahmad Zaib Muhammad Imran Razzak, Saeeda Naz. Deep learning for medical image processing: Overview, challenges and future. In *CPHHI*, 2016.

[10] Wanli Ouyang, Xiaogang Wang, Xingyu Zeng, Shi Qiu, Ping Luo, Yonglong Tian, Hongsheng Li, Shuo Yang, Zhe Wang, Chen-Change Loy, and Xiaoou Tang. Deepid-net: Deformable deep convolutional neural networks for object detection. In *CVPR 2015 p.2403-2412*, 2014.

[11] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *Department of Computer Science, University of Manitoba, Canada*, 2015.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *arXiv:1506.02640v5*, 2016.

[13] Hamid Rezatofighi, Nathan Tsoi JunYoung, and Gwak Amir Sadeghian. Generalized intersection over union: A metric and a loss for bounding box regression. In *arXiv:1902.09630v2*, 2019.

[14] Robot-Assisted Surgery, Using Deep, and Learning. Automatic instrument segmentation in. In *arXiv:1803.01207v2*, 2018.

[15] Jonathan Tremblay, Thang To, and Balakumar Sundaralingam. Deep object pose estimation for semantic robotic grasping of household objects. In *arXiv:1809.10790v1*, 2018.

[16] Jun Xu, Yanxin Ma, Songhua He, and Jiahua Zhu. 3d-GIoU: 3d generalized intersection over union for object detection in point cloud. *Sensors MDPI*, 19(19):4093, sep 2019.

[17] Zhong-Qiu Zhao, Peng Zheng, Shou tao Xu, and Xindong Wu. Object detection with deep learning: A review. In *arXiv:1807.05511v2*. IEEE, 2019.

Abbildungsverzeichnis