DLM Semesterprojekt: Bestimmung der 3D-Boundingbox eines Roboters

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

Abstract

This Latex and Style file are modified versions from NeurIPS 2018.[1]

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 [13, 9]. So können beispielhaft Operationsroboter stark von einer auf *Deep-Learning*-Algorithmen basierenden Erkennung und Lokalisation der Operationsinstrumente profitieren [12]. Auch in der Tumordiagnostik kann so die Erkennungsgenauigkeit gegenüber einer menschlichen Klassifikation bei gleichzeitig signifikant niedrigeren Kosten verbessert werden [3].
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-Boundingbox* 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 und Eclipse IDE. Die Evaluation der Implementierung geschieht anhand eines Vergleiches der geschätzten *Boundingbox* mit der gelabelten *Boundingbox* der Testdaten. Als Datengrundlage für Training und Evaluation dient ein synthetisch generierter und vorgelabelter Datensatz von RGB-Bildern.

2 Grundlagen und Stand der Technik

Salvaris beschreibt *Machine Learning* 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. DL ist ein Teilgebiet des ML (Abb. 1) bei dem besonders komplexe Neuronale Netze mit vielen Schichten und Neuronen Verwendung finden. Ein 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 [8, 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 [6] verwiesen.

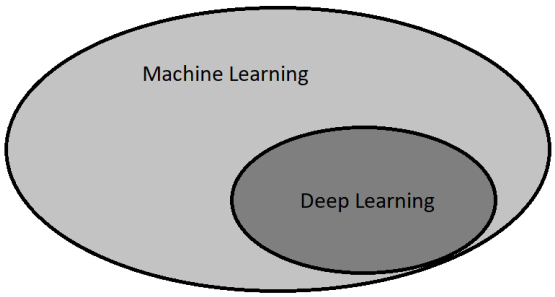

Abbildung 1: Abgrenzung Deep Learning zu Machine Learning.

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 [5, 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.

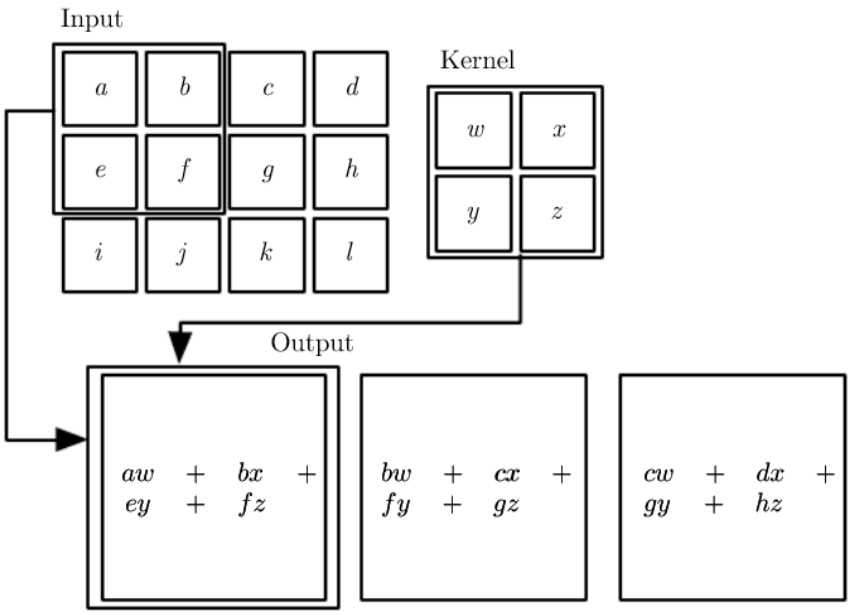

Abbildung 2: Prinzip eines Convolutional Layers [5, S.330].

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.

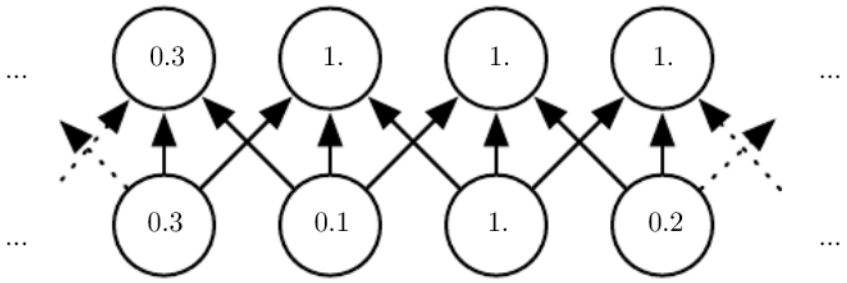

Abbildung 3: Prinzip eines Pooling Layers [5, S.337].

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 [5, 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 [2]. Loss-Funktionen stellen so einen entscheidenden Teil für ein erfolgreiches Training dar. Dabei sollte je nach Anwendungsfall individuell eine passende und sinnvolle Loss-Funktion zur Validierung gewählt werden. Ein ausführliche Abhandlung hierüber findet sich in [6] und [8].

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

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 [14]. *Fast-Region-Based-CNNs* ermöglichen gute Detektionsergebnisse, sind aber relativ rechenaufwendig [4]. Das *You Only Look Once* Modell (YOLO) erreicht eine höhere Performanz bei verbesserter Präzision [11]. Ein weiteres Modell zur Lösung des Problems ist das von Liu vorgestellte *Single-Shot-Detection* (SSD) Modell vor. Beim SSD konnten Performanz und Genauigkeit zu Lasten einer komplexeren Netzarchitektur noch weiter verbessert werden [7]. Die Gemeinsamkeit aller vorgestellten Modelle besteht in der Verwendung von *Convolutional Layern*.

3 Methoden

Zu Lösung der Aufgabenstellung wird die sogenannte 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 günstiger Hardware eine Echtzeitschätzung der Bounding-Boxen von Objekten möglich [11].

3.1 Loss mittels Interception over Union

3.2 Netzstruktur

@Paul: hab das Paper gefunden, von dem unsere Netzstruktur inspiriert ist, da kannst du mal einen Blick rein werfen wenn du weitere infos brauchst: [11]

3.3 Klassendiagramm

Falls du bock drauf hast wäre es vlt noch sinnvoll grob das Klassendiagramm mit den Funktionen zu zeigen (aus python). Aber auf keinen Fall details. aber wenn du so schon genug hast reicht das auch ohne

4 Ergebnisse

5 Fazit und Ausblick

ganz kurz

6 Literatur und Bilder

Literatur

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

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

[3] 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.

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

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

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

[7] 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.

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

[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*, 2014.

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

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

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

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

Abbildungsverzeichnis