Studienarbeit - Deep Learning

Erkennung und Umrandung von Bienen durch Neuronale Netze

Ahmet Sahin
Munich University of Applied Sciences
Department of mechanical, automotive and aeronautical engineering
Munich, Germany
asahin@hm.edu
Matr. Nr.: 23159816

Oliver Scholz
Munich University of Applied Sciences
Department of computer science and mathematics
Munich, Germany
scholz.oliver@hm.edu
Matr. Nr.: 07991619

Josef Kaffl
Munich University of Applied Sciences
Department of computer science and mathematics
Munich, Germany
jkaffl@hm.edu
Matr. Nr.: 26260218

***Abstract*—In dieser Studienarbeit wird die Erkennung (detection) und Umrandung (bounding boxes) von Bienen beschrieben. Hintergrund der Arbeit ist ein Forschungsprojekt (EnBeeMo), für das Bienen gezählt werden müssen. Hierfür eignet sich der Einsatz von Deep Learning. Es bestehen bereits vortrainierte Modelle wie die Masked Regionbased Convolutional Neural Networks (Mask R-CNN) und You only look once (YOLO). Ersterer wird in dieser Studienarbeit eingesetzt.**

I. Einleitung

Hintergrund der Projektarbeit ist das Forschungsprojekt „EnBeeMo“ (Environment and Bee Monitoring), welches die Entwicklung eines Pareto-Optimalen Algorithmus' zur Erkennung von Bienen zum Ziel hat. Als Datengrundlage und Ausgangspunkt stehen 2000 gelabelte Bilder mit einer Auflösung von 1613x371 zur Verfügung, die von einer an einem Bienenstock angeschlossenen Monitoring-Box aufgezeichnet wurden. Um die Zusammenhänge zwischen dem Bienenbestand und Umweltfaktoren wie Luftfeuchtigkeit und Temperatur besser beschreiben zu können, soll ein performanter Zähl- und Detektionsalgorithmus zur Erkennung des Objekts 'Biene' mit Hilfe von Bounding Boxes von den Studententeams entwickelt werden. Hierbei soll eine Implementierung von CNN-Architekturen, eine Parametrisierung der Netzwerke und eine Ermittlung von Precision, Recall und processing time erfolgen. Außerdem sollen insbesondere auch die relevanten Hyperparameter in entsprechenden Intervallen identifiziert werden, deren Zusammenhänge untereinander beschrieben werden und letztlich ein Tuning der gefundenen Hyperparameter vorgenommen werden, um die Leistung im erstellten Netz zu verbessern.

II. Theoretischer Hintergrund

Die Umsetzung der Projektarbeit soll mit dem Einsatz von Deep Learning erfolgen. Deep Learning als Unterdisziplin von Machine Learning verwendet Layer von Algorithmen, um Daten zu verarbeiten und dadurch beispielsweise Objekte zu erkennen oder menschliche Sprache zu verstehen. Dabei wird über die Layerkette (vom Input-Layer über Hidden-Layer zum Output-Layer) die betrachtete Information weitergeleitet und verarbeitet. Die für diese Arbeit relevanten Netzklassen sind die um das Convolutional Neural Network (CNN). CNNs gehören zu den Feedforward-Netzen, welche eine vorwärts gerichtete Flussrichtung besitzen. Eine Unterform stellt die Familie um die Regionbased CNNs (R-CNNs) dar, bei der mit Hilfe eines Selective-Search-Algorithmus' eine Auswahl an Regionen (Region of Interest, RoI) des Inputs generiert wird, die in die weitere Klassifizierung mittels eines vortrainierten CNNs eingeht [1] [2].

Innerhalb der Netze treten dabei verschiedene Layer auf: Convolutional Layer (CL) erkennen lokale Konjunktionen von Features der vorherigen Schicht und ordnen sie einer Feature Map zu. Dabei ergeben mehrere Feature Maps letztendlich ein Objekt. Außerdem kommt es in Pooling Layer (PL) zu einer Bereichsaufteilung der Feature Maps. In Activation Layer (AL) wird die Aktivierungsfunktion auf alle gegebenen Werte angewendet. Am Ende eines CNNs klassifiziert das Fully Connected Layer (FCL) die Ergebnisse aus den vorangegangenen Daten [3].

Historisch gesehen geht das erste CNN auf Kunihiko Fukushima zurück: Er entwarf ein CNN mit mehreren CL und PL. Sein Netz Neocognitron (1979) kann visuelle Muster erkennen und lernen. Das Neocognitron inspirierte die weitere Forschung an CNN in den 90ern und brachte weitere Besserungen hervor, beispielsweise die Handschrift-Erkennung durch Yann LeCun. Über die letzten Jahre hinweg konnten erhebliche Performanceverbesserungen erzielt werden bei der Erkennung natürlicher Bilder, der Gesichtserkennung und der Analyse medizinischer Bilder. So konnten die CNNs die klassischen Hardcoding-Methoden ab Anfang der 2000er Jahre outperformen [1] [4].

III. Problembeschreibung

Anhand des gegebenen Bilddatensatzes sollen nun eine Klassifikation und Regression durchgeführt werden, um zu erkennen, ob auf dem betrachteten Bild Bienen zu sehen sind, und wo sie sich auf dem Bild befinden. Die Implementierung eines eigenen Netzes erweist sich dabei als sehr komplex, weswegen die Projektgruppe sich für die Verwendung eines vortrainierten Netzes entscheidet. Hierbei

wird auf einen bestehenden Algorithmus zurückgegriffen, der mittels architektonischer Anpassungen adaptiert wird. Mögliche Netze, die zur Bearbeitung verwendet werden können, sind dabei beispielsweise R-CNN, Fast R-CNN, Faster R-CNN und Mask R-CNN sowie Single-Shot-Verfahren wie SSD oder YOLO. Das Training wird auf einer Amazon Web Services (AWS) Instanz durchgeführt.

IV. Anwendung: Mask R-CNN

Im Folgenden hat sich die Projektgruppe für die Arbeit mit Mask R-CNN entschieden, welches sich über architektonische Verbesserung in der Kette von R-CNN über Fast R-CNN und Faster R-CNN am besten bewährt hat.

A. Architektur des Mask R-CNN

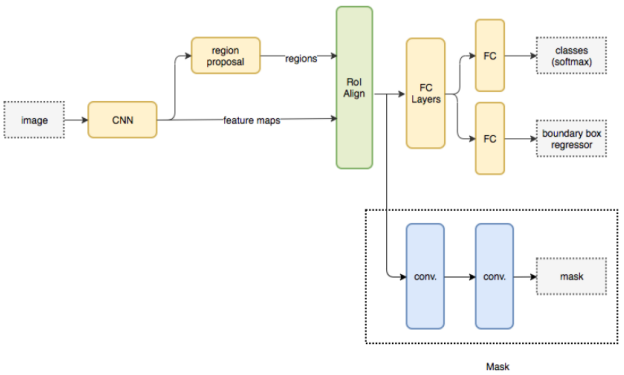

Fig. 1. Struktur der Mask R-CNN Architektur [5]

Mask R-CNN basiert auf einer Weiterentwicklung des Faster R-CNN. Dabei setzt sich Faster R-CNN aus zwei Stufen zusammen: Die erste Stufe ist das Region Proposal Network (RPN), welches den betrachteten Objekten aus der RoI Bounding Boxes vorschlägt. Darauf folgt die zweite Stufe, die im Wesentlichen ein Faster R-CNN ist. Hierbei werden von jeder Bounding Box Features mittels RoI-Pool extrahiert und eine Klassifikation und Bounding-Box-Regression wird durchgeführt. Im Gegensatz zum Faster R-CNN wird beim Mask R-CNN nach identischem ersten Schritt (RPN) im zweiten Schritt neben der Klassifikation und Regression parallel eine binäre Maske für jede RoI ausgegeben. Durch diese Maske kann ein deutlich feineres räumliches Layout des Input Objekts abgebildet werden [6].

Innerhalb des Mask R-CNN werden zwei Architekturen voneinander abgegrenzt: Zum Einen die Convolutional Backbone Architektur, welche zur Feature-Extraktion über ein gesamtes Bild verwendet wird und zum Anderen das Head-Netzwerk zur Erkennung von Bounding Boxes und zur Vorhersage von Masken, welche auf jede RoI angewendet werden. Das Convolutional Backbone basiert grundlegend auf zum Beispiel ResNet (Einführung einer Verknüpfung von der vorherigen zur nächsten Ebene) oder ResNeXt (Stapeln gleicher Topologie Blöcke) Netzen, mit einer Tiefe von 50 oder 101 Layer. Beispielsweise kann ein Backbone mit ResNet-50-C4 genutzt werden, mit einem 50 Layer ResNet und Feature-Extraktion vom letzten CL der vierten Stufe (C4). Noch bessere Schnelligkeit und Genauigkeit können beim Verwenden eines Feature Pyramid Networks (FPN) erreicht werden, einer Top-Down Pyramiden Architektur mit seitlichen Verbindungen. Der Head des Netzes ist eine Erweiterung bestehender Architekturen um einen Fully-Convolutional Masken-Vorhersage Ast. Hierbei wird beim Faster R-CNN Box Head vom beispielsweise ResNet-C4 Backbone ein Masken-Ast hinzugefügt. Diese Abzweigung für die Maske kann entweder einfach strukturiert sein oder auch komplexer, um die Performance weiter zu verbessern. So kann parallel zur Regression und Klassifikation eine Maske ausgegeben werden [6] [7].

B. Implementierung

Für das Training des Mask R-CNN wird eine zur Verfügung gestellte AWS Sagemaker-Instanz genutzt. In dieser wird ein Python 3.6 Kernel gestartet, der Tensorflow in der Version 1.13 inklusive GPU-Unterstützung bereitstellt. Als Ausführungsinstanz wird der Typ `ml.g4dn.xlarge` gewählt. Dieser Instanztyp bringt 4 virtuelle Kerne, 16 GiB RAM sowie eine GPU mit. Bei der GPU handelt es sich um eine NVIDIA Tesla T4, die auf einer Turing-Architektur basiert und für KI-Aufgaben optimiert ist [8]. CUDA liegt in der Version 11 vor. Unter diesen Voraussetzungen ist ein schnelles und effizientes Training des Mask R-CNN möglich.

Das herangezogene Mask R-CNN Modell basiert auf einem bereits bestehenden GitHub-Repository [9]. Das Modell ist auf Basis von Keras und Tensorflow gebaut. Durch die Weiterentwicklung von Keras und Tensorflow kommt es teilweise zu Deprecations-Warning, wobei mit Tensorflow 1 die Ausführung des Netzes kein Problem darstellt. Für die Verwendung des Modells im Rahmen der vorliegenden Projektarbeit wird lediglich eine kleine Veränderung vorgenommen: Zum Trainieren des Modells wird die Keras-Methode `tf.keras.Model#fit_generator` genutzt, die ein History-Objekt als Rückgabewert hat. Dieses wird in der vorliegenden Implementierung ignoriert. Die Änderung besteht nun darin, das History-Objekt auszugeben, um einen Verlauf der erreichten Metriken über die Epochen hinweg zu erhalten.

Für das Training des Modells werden zunächst zwei wesentliche Datasets erstellt, train und test. Diese sind Instanzen der Klasse `BeeDataset`, die die Klasse `Dataset` erweitert. Die Klasse `BeeDataset` überschreibt Methoden zum Laden der Bilder aus einem S3-Storage, dem Verarbeiten der Manifest-Datei, die die Ground-Truth-Daten beinhaltet und zum Extrahieren der entsprechenden Bounding-Boxes aus diesen Informationen. Das Trainings-Dataset beinhaltet ca. 75% der Daten, die zum Training genutzt werden. Zum Validieren wird das Test-Dataset genutzt, was die übrigen 25% der Bilder beinhaltet. Weiterhin gibt es ein Sample-Dataset, welches ein einziges Bild enthält, für das keine Annotation in der Manifest-Datei vorhanden ist.

Im Anschluss an diese Datenaufbereitung erfolgt das Erstellen einer Netz-Konfiguration, die für das Trainieren verwendet wird. Diese beinhaltet verschiedene Parameter, darunter insbesondere die einstellbaren Parameter. Die Konfiguration basiert auf der Klasse `Config` (GitHub-Repository: `./mrcnn/config.py`) und überschreibt dort die gewünschten Parameter. Grundlegend wird die Anzahl der gesuchten Klassen auf 2 gesetzt. Dies begründet sich in der Arbeitsweise des Mask R-CNN, welches mit einer Background-Klasse arbeitet. Dadurch ergeben sich die Klasse `bee`, der die Bienen zuzuordnen sind, und eine `background`-Klasse. Zudem wird der Konfiguration ein sprechender Name gegeben (`bee_cfg`) und die Anzahl der Schritte (1000) je Epoche (5) festgelegt. Diese Parameter bleiben über alle Trainingsläufe hinweg unverändert, um vergleichbare Ergebnisse bei den Variationen der Hyperparameter zu erhalten. Als Backbone-Netzwerk wird unverändert ein ResNet-101 Netz genutzt. Dies könnte ebenfalls in der Konfigurationsklasse geändert werden.

Für das Training selbst wird nun diese Konfiguration genutzt, um eine Instanz der Klasse `MaskRCNN` zu erstellen, für die der Trainingsmodus aktiviert wird. Da ein Transfer-Learning durchgeführt wird, werden nun die vortrainierten Gewichte geladen. Hierfür nutzt die Projektgruppe Gewichte, die auf Basis des COCO-Datensatzes ermittelt wurden. Dabei werden jedoch verschiedene Layer, darunter insbesondere die der Bounding Box-Regression des Mask R-CNN ausgelassen, damit diese Gewichte im Training auf den vorliegenden Use-Case angepasst werden können. Diese Modellinstanz wird nun angewiesen, mit dem vorliegenden Trainings- und Validierungs-Dataset ein Training durchzuführen.

Das Training mittels GPU benötigt für eine Epoche zwischen 17 und 25 Minuten. Anschließend wird das zuvor angesprochene History-Objekt ausgegeben und gemeinsam mit der verwendeten Konfiguration (händisch) gespeichert, um den Verlauf der Metriken später zu bewerten.

Weiterhin wird eine Modell-Konfiguration erstellt, die eine Prediction ermöglicht. Dabei werden ebenfalls die Klassen auf 2 gesetzt, darüberhinaus werden jedoch keine gravierenden Änderungen vorgenommen und das Modell instanziiert. Im Gegensatz zum vorherigen Laden der vortrainierten Gewichte des COCO-Datensatzes erfolgt nun ein Laden der eigenen Gewichte, die für jede durchgeführte Epoche gespeichert werden. Eine Methode plottet nach der Prediction die ermittelten Bounding Boxes auf dem Bild, um einen visuellen Eindruck des Ergebnisses zu erhalten.

Abschließend ermittelt ein Algorithmus die Überlappung der Ground Truth-Daten mit den vorhergesagten Bounding Boxes, um diese Überlappung als Maß für die Genauigkeit nutzen und eine Bewertung der gewählten Hpyerparameter zu ermöglichen.

C. Verbesserung des Modells durch Hyperparamtertuning

Um die Performance des Modells zu verbessern sind die Hpyerparameter des Modells zu tunen. In dieser Studienarbeit wird konkret die Lernrate, Weight Decay und das Learning Momentum des Mask R-CNN varriert, um so festzustellen welche Werte für das Problem optimal sind. Neuronale Netze werden mithilfe des Gradientenabstiegsverfahrens trainiert. Das bedeutet, dass die Gewichte des Netzes immer dem Fehler entsprechend angepasst werden. Die Lernrate begrenzt dabei die Veränderung der Gewichte. Ist sie zu hoch, kann es passieren, dass beim Training das globale Minimum des Fehlerwertes verfehlt wird. Ist die Lernrate jedoch zu niedrig, kann es passieren, dass das Modell in einem globalen Minimum verharrt. Daher ist die Wahl einer passenden Lernrate für die Lösung des Problems essenziell [10].

In unterschiedlichen Trainingsläufen werden verschiedene Lernraten ausprobiert. Dabei stellt sich heraus, dass eine Lernrate > 0.05 reproduzierbar dazu führt, dass u. a. keine Berechnung des Losses mehr möglich ist und dass bei der Vorhersage von Bounding Boxes keine einzige Biene erkannt wird. Bei einer Lernrate < 0.05 und > 0.005 hingegen werden Bienen erkannt, jedoch mit einer Genauigkeit (mAP, mean average precision), ermittelt über 100 zufällig ausgewählte Bilder, von lediglich ca. $mAP = 70\%$. Bei einer Lernrate im Intervall $[0.001, 0.003]$ erzielt das Modell die besten Resultate mit einer Genauigkeit von $mAP = [80\%, 82\%]$. Wird die Lernrate weiter verringert, nimmt die Genauigkeit wieder ab und der ermittelte Loss zu.

Eine weitere Möglichkeit, um das Feststecken in lokalen Minima zu vermeiden ist das Momentum. Durch das Momentum werden die Optimierungsgeschwindigkeiten aus den vorherigen Schritten miteinbezogen, wodurch vermieden wird, dass sich der Gradientenabstieg im lokalen Minimum festsetzt. Der verstellbare Hyperparameter bestimmt dabei, wie schnell sich das Momentum abschwächt [11]. Für die Eingrenzung des Learning Momentum auf ein optimales Intervall wird die Lernrate auf 0.003 festgesetzt, damit vergleichbare Ergebnisse erzielt werden. Bei den Versuchen wird der Wert für das Momentum auf das Intervall $[0.75, 0.95]$ eingegrenzt. Bei einem Wert von 0.9 wird die höchste mAP erreicht, weshalb dieser Wert für die Optimierung des Weight Decay im Folgenden weiterverwendet wird.

Wenn das Modell für das Problem zu komplex ist und es somit zwar einen geringen Trainings-, aber einen hohen Test- bzw. Generalisierungsfehler aufweist, spricht man Overfitting. Ein Hinweis auf Overfitting bieten hoch eingestellte Gewichte. Um dieses Problem zu vermeiden wird das sogenannte Weight Decay oder L^2 Regularisierung eingesetzt. Dabei wird an die Loss-Funktion ein zusätzlicher Wert, das Gewicht im Quadrat multipliziert mit einem Weight Decay Faktor, angehängt. Durch die Einstellung des Weight Decay Faktors werden somit hohe Werte für Gewichte, abhängig von der Größe des Faktors mehr oder weniger, bestraft und regularisiert [12]. Bei der Suche nach einem optimalen Weight Decay wird ermittelt, dass dieser Hyperparameter optimalerweise im Intervall $[0.0005, 0.001]$ liegt. Außerhalb dieses Intervalls verringert sich die mAP, jedoch sind die Unterschiede geringer

als bei der Lernrate.

V. Diskussion der Ergebnisse

Die Lernrate des Mask R-CNN im Intervall von 0.001 bis 0.003 erzielt die besten Ergebnisse. Die konstant übereinstimmenden Ergebnisse in diesem Intervall lassen sich durch die vortrainierten Gewichte begründen. Dadurch, dass das Modell nicht mit zufällig initialisierten, sondern mit bereits bewehrten Gewichten trainiert wird, sind die Ergebnisse im besagten Intervall unabhängig von der Lernrate. Aus dem selben Grund ist auch kein großer Weight Decay nötig. Da dessen Zweck die Vermeidung von Overfitting durch die Abschwächung großer Gewichte ist, muss dieser nicht groß eingestellt werden, weil die Gewichte bereits sinnvolle Werte darstellen. Dennoch sind auch hier Veränderungen der Werte in den Resultaten sichtbar. Das Learning Momentum liegt bei dem vorliegenden Mask R-CNN und der geschilderten Aufgabenstellung optimalerweise nahe, aber unter 1. Ein abschließendes Training wird mit einer Lernrate von 0.002, einem Weight Decay von 0.001 und einem Momentum von 0.95 durchgeführt. Dieses erzielt eine mAP von 79.6%. Das Resultat einer Erkennung durch das trainierte Netz ist in Abbildung 2 zu sehen.

Fig. 2. Ergebnis der Prediction auf einem zufällig gewählten Bild. An jeder Biene ist zudem ein Wert angegeben, wie sicher sich das Modell ist, dass es sich um eine Biene handelt.

Für die Erkennung der Bienen auf einem Bild benötigt das Mask R-CNN circa 2 Sekunden. Dies bedeutet, dass das Modell für bewegte Bilder ungeeignet ist, weil die Erkennung eines Bildes mehr Zeit benötigt, als das Bild zu sehen ist. Im Rahmen der Studienarbeit beschäftigt sich die Projektgruppe parallel mit einem weiteren Algorithmus, dem YOLO (v4). Dieser wird in Google Colab trainiert und erzielt nach einem Hyperparametertuning einen mAP von 89%. Der große Vorteil des YOLOv4 gegenüber dem Mask R-CNN ist die Geschwindigkeit. Für die Erkennung eines Bildes (ebenfalls mittels Tesla T4) benötigt YOLOv4 etwa 20ms, was bestätigt, dass YOLO für Videos geeignet ist. Der Grund für die Verwendung des Mask R-CNN in dieser Studienarbeit ist dem Fakt geschuldet, dass dieses mit Tensorflow und Keras implementiert ist, während das verwendete YOLOv4 (Darknet) in C und CUDA geschrieben ist und somit wenig Einsicht auf die Architektur und Einfluss auf die Hyperparameter zulässt. Unabhängig des gewählten Modells könnte die Performance verbessert werden, indem Data Augmentation eingesetzt wird, um einerseits mehr Daten durch Bildbearbeitung (Spiegelung, Änderung des Kontrasts, Zoom) für das Training zu generieren. Andererseits könnte durch Umwandlung der RGB-Bilder in eindimensionale Graustufen Bilder die Trainings und Ausführzeit verbessert werden. Außerdem wurde festgestellt, dass bei einigen Bilder die Label einiger Bienen gänzlich fehlen, was kontraproduktiv für das Training und die Berechnung des mAP ist.

VI. Zusammenfassung und Ausblick

Zusammenfassend lässt sich sagen, dass das Ziel der Arbeit, nämlich die Detektion von Bienen auf Bildern, mit dem Mask R-CNN bei sehr geringer Trainingsdauer und guten Ergebnissen, erreicht wird. Mit den optimierten Parametern werden die besten Ergebnise erzielt. Die Performance des Algorithmus kann durch Data Augmentation, also die künstliche Erweiterung der vorhandenen Daten durch Veränderung, weiter verbessert werden. Außerdem sind die Label der vorhandenen Daten teilweise fehlerhaft, was zu Fehlern in der Gewichtsanpassung führt. Deshalb sollten die Label der Daten für das finale Modell im Projekt angepasst werden. Zudem ist erwähnenswert, dass es weitere Modelle zur Bilderkennung gibt. Sofern die Erkennung auf bewegten Bildern erfolgen soll, wird ein Single Shot Algorithmus wie YOLO empfohlen, welcher in sehr kurzer Rechendauer ähnliche Ergebnisse wie Mask R-CNN liefert.

References

[1] K. D. Foote, “A brief history of deep learning,” 07.02.2017. [Online]. Available: https://www.dataversity.net/brief-history-deep-learning/#

[2] R. Gandhi, “R-cnn, fast r-cnn, faster r-cnn, yolo – object detection algorithms,” 09.07.2018. [Online]. Available: https://bit.ly/3tAJvO2

[3] M. Ziegler, “Object recognition with convolutional neural networks,” 2018. [Online]. Available: https://cogsys.uni-bamberg.de/teaching/ws1718/sem_m2/Seminararbeit_Kognitive_Systeme_Michael_Ziegler.pdf

[4] R. Draelos, “The history of convolutional neural networks,” 13.04.2019. [Online]. Available: https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/

[5] J. Singh and S. Shekhar, “Road damage detection and classification in smartphone captured images using mask r-cnn,” *arXiv preprint arXiv:1811.04535*, 2018.

[6] H. Kaiming, G. Georgia, D. Piotr, and G. Ross, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969. [Online]. Available: https://ai.facebook.com/research/publications/mask-r-cnn/

[7] E. Ma, “Enhancing resnet to resnext for image classification,” 24.03.2020. [Online]. Available: https://medium.com/dataseries/enhancing-resnet-to-resnext-for-image-classification-3449f62a774c

[8] NVIDIA, “Nvidia tesla t4 tensor-recheneinheiten zur inferenzbeschleunigung,” 29.01.2021. [Online]. Available: https://www.nvidia.com/de-de/data-center/tesla-t4/

[9] GitHub, “matterport/mask_rcnn,” 29.01.2021. [Online]. Available: https://github.com/matterport/Mask_RCNN

[10] J. Brownlee, “Understand the impact of learning rate on neural network performance,” *Machine Learning Mastery*, 24.01.2019. [Online]. Available: https://bit.ly/3jnxkzh

[11] Prof. Dr. David Spieler, “Deep learning 06: Optimization,” lecture for deep learning, University of Applied Sciences Munich, 04.12.2018.

[12] ——, “Deep learning 04: Regularization,” lecture for deep learning, University of Applied Sciences Munich, 19.11.2019.

Hinweis: Bei der Erstellung des Dokumentes haben alle Projektmitglieder zu gleichen Teilen mitgewirkt.