Echtzeit Positionsbestimmung von Fahrzeugen im „Mikrowunderland“ mit Object Detection

Jan Poth

Fakultät Technik und Informatik
Department Informatik
Hochschule für Angewandte Wissenschaften Hamburg
Hamburg, Deutschland

Abstract In dieser Arbeit wird eine Lösung vorgestellt und evaluiert, um die Positionen der Fahrzeuge im „Mikrowunderland“ der HAW Hamburg in Echtzeit zu bestimmen. Dafür wird eine Kamera über dem „Mikrowunderland“ befestigt, welche die Fahrzeuge aus der Vogelperspektive aufnimmt und die Bilder auf einem Raspberry Pi verarbeitet. Die Bilder werden dafür auf eine Auflösung von $300 \times 300 \times 3$ Pixel herunterskaliert und mit einem neuronalen Netz verarbeitet. Das neuronale Netz basiert auf einem MobileNetV2 SSD, wird mit Transfer Learning und eigenen Bildern trainiert und ist mit TFLite auf einem Raspberry Pi lauffähig. Für die Erweiterung der Trainingsdaten wurde zusätzlich eine Vielzahl von „Data Augmentation“-Methoden verwendet.

Keywords: Convolutional Neural Networks, Object Detection, Data Augmentation, Car Tracking, Image Processing, Embedded Machine Learning, TFLite

1 Einleitung

Das „Mikrowunderland“ Projekt der HAW Hamburg (siehe Abbildung 1) beschäftigt sich mit autonom fahrenden Fahrzeugen im Maßstab 1:87. Die Bestimmung der exakten Fahrzeugpositionen eröffnet neue Anwendungsmöglichkeiten. Dazu gehören beispielsweise die Analyse von Fahrverhalten, optimierter Verkehrsfluss, besseres autonomes Fahren durch zusätzliche Informationen und vieles mehr.

Abbildung 1: Bild vom Mikrowunderland (eigene Aufnahme)

Diese Arbeit beschäftigt sich mit der Positionsbestimmung von Fahrzeugen im „Mikrowunderland“. Dazu wird eine Kamera die Fahrzeuge aus der Vogelperspektive aufnehmen und die Bilder an ein eingebettetes System weitergeben. Das Bild wird herunterskaliert und als Eingabe für ein neuronales Netz verwendet werden. In Echtzeit werden viele Fahrzeuge gleichzeitig als Ausgabe des neuronalen Netzes klassifiziert und lokalisiert.

2 Maschinelles Lernen

Diese Arbeit verwendet maschinelles Lernen, welches ein Teilbereich der künstlichen Intelligenz ist. Genauer wird sich auf das überwachte Lernen bezogen. Beim überwachten Lernen werden die Trainings- und Testdaten mit zusätzlichen Metainformationen verknüpft. Während des Trainings wird ein Fehler berechnet, wie gut das neuronale Netz Vorhersagen treffen kann. Der Fehler wird aus den Vorhersagen des neuronalen Netzes und den Metainformationen mithilfe einer Fehlerfunktion berechnet. Mithilfe dieses Fehler und Backpropagation kann das das neuronale Netz lernen. Fehlerfunktionen werden auch „Loss-Functions“ genannt und verwenden unterschiedliche Berechnungsgrundlagen für den Fehler. [1, vgl.]

Die sogenannte „Object Detection“ (Objekterkennung) beschreibt eine Bildverarbeitung, bei der ein Bild als Eingabe verwendet wird und als Ausgabe eine Liste mit detektierten Objekten ausgegeben wird. Dabei werden die Ergebnisse meist als bunte Rechtecke über dem erkannten Objekt dargestellt, wie zum Beispiel in Abbildung 9b zu sehen ist. Dazu wird dann die jeweilige Klasse angegeben, die erkannt wurde. [2, vgl.]

„Transfer Learning“ ist eine Methode, bei der ein neuronales Netz verwendet wird, welches bereits mit einem meist generelleren Datensatz trainiert wurde. Anschließend werden weitere Bilder, die den speziellen Anwendungsfall repräsentieren, verwendet und das neu-

ronale Netz damit trainiert. „Transfer Learning“ hat den Vorteil, dass ein großer Teil der Trainingszeit bereits erfolgt ist und darauf aufgebaut werden kann. In den ersten Schichten eines neuronalen Netzes werden meist einfache Formen und Strukturen abgebildet, die generell für viele Anwendungsfälle hilfreich sind. [3, vgl.] [4, vgl.]

2.1 Modellierung

Das zu erstellende neuronale Netz soll auf eines Raspberry Pi lauffähig sein und gegebenenfalls mit einer Coral-TPU (siehe 2.3) beschleunigt werden. Die Erkennung der Fahrzeuge auf Basis eines Bildes wird mit Object Detection durchgeführt. [1, vgl.] Dabei wird dem neuronalen Netz als Eingabe das zu analysierende Bild übergeben. Die Ausgabe enthält alle erkannten Objekte, in diesem Fall Fahrzeuge. Dazu wird die Position des Fahrzeuges mit einem Rechteck visualisiert und die Art des Fahrzeuges klassifiziert.

Abbildung 2: Spielzeugfahrzeug: DHL Truck [5]

Diese Arbeit beschränkt sich auf die Erkennung von einem gelben DHL-Truck (siehe Abbildung 2). Für die interne Verarbeitung wird dies die Klasse „truck“ genannt. Die Klassen sollen technisch nicht nur auf die Klasse „truck“ limitiert sein, sondern auch offen für neue Klassen wie „car“, „tractor“ oder andere sein.

2.2 Neuronales Netz

Als neuronales Netz, welches die Anforderungen aus Abschnitt 2.1 erfüllt, wird ein „SSD MobileNet v2 300×300“ [6] aus dem „TensorFlow 2 Detection Model Zoo“ [7] genutzt. Das neuronale Netz ist bereits auf den „COCO 2017 Datensatz“ trainiert. Dieser Datensatz besteht aus mehr als 330.000 Bildern mit 1.500.000 Objekten in 91 Objekttypen. [8] Mithilfe von Transfer Learning (siehe Abschnitt 2) soll dieses Netz auf den Anwendungsfall der Erkennung und Lokalisierung von Fahrzeugen im „Mikrowunderland“ trainiert werden.

Das „SSD MobileNet v2 300×300“ wird für diese Arbeit ausgewählt, da es einen guten Kompromiss aus Geschwindigkeit und Genauigkeit findet. [6, vgl.] Die Auflösung von

300×300×3 Pixel wird genutzt, um die Geschwindigkeit zu erhöhen, da die Originalauflösung der Kamera mit 1920×1080×3 Pixel deutlich zu rechenaufwendig wäre. Die ×3 steht für die Verwendung von den Farbinformationen (Rot, Grün, Blau). Das Netz verwendet dementsprechend 300×300×3 = 270.000 Neuronen in der Eingabeschicht.
Die Besonderheit vom SSD (Single Shot Detector) ist, dass für die Klassifikation und die Lokalisierung von Objekten nur ein einzelner Durchlauf durch das neuronale Netz nötig ist. Es können dabei auch, wie in diesem Anwendungsfall nötig, mehrere Objekte gleichzeitig klassifiziert und lokalisiert werden. [9, vgl.] [6, vgl.]

2.3 Beschleunigung

Um die Laufzeit für die Verarbeitung eines Bildes zukünftig reduzieren zu können, ist es angedacht, einen Coral „Google Coral USB Accelerator“ zu verwenden. Ein „Google Coral USB Accelerator“ ist ein für maschinelles Lernen optimierter Koprozessor, der via USB an einen Raspberry Pi angeschlossen werden kann. [10, vgl.] Mit diesem könnte eine Performance von 30 Bildern pro Sekunde erreicht werden, welche für diesen Anwendungsfall ausreichend ist. Laut Benchmarks auf der Website wird mit einem „Dev Board: Quad-core Cortex-A53 @ 1.5GHz + Edge TPU“ (wird im Raspberry Pi 3 verwendet [11]) eine Performance von ca. 71 Bildern pro Sekunde bei einem „MobileNet v2 SSD (224×224)“ erreicht. [12] Folglich ist eine Performance von 30 Bildern pro Sekunde mit unserem „MobileNet v2 SSD (300×300)“ auf einem Raspberry Pi 4 realistisch, müsste aber noch einem Praxistest unterzogen werden.

3 Versuchsaufbau

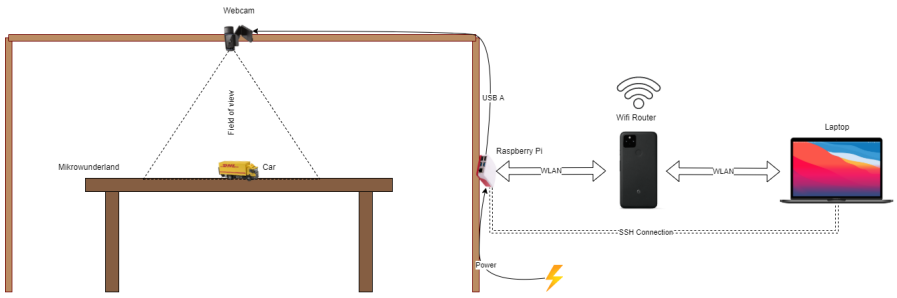


Abbildung 3: Darstellung des gesamten Versuchsaufbaus

Für die Bildaufnahmen wird eine Kamera an einem Holzgestell über dem „Mikrowunderland“ befestigt. Die Kamera ist in diesem Fall eine Webcam und nimmt das Geschehen im „Mikrowunderland“ aus der Vogelperspektive auf. Die Webcam wird per USB mit einem Raspberry Pi der vierten Generation verbunden.

Um die Bilder aufzunehmen und zu bearbeiten, wird ein mobiles Wi-Fi mit einem Telefon erstellt, in welches sich sowohl der Raspberry Pi als auch ein Laptop einloggen. Zwischen den beiden Geräten wird eine SSH-Verbindung hergestellt, um von dem Laptop auf den Raspberry Pi zuzugreifen und diesen zu steuern.
Mit einem selbst geschriebenen Skript werden automatisiert alle Bilder aufgenommen, die später für das Training und die Auswertung verwendet werden. Anschließend werden die Bilder vom Raspberry Pi mittels SSH (Secure Shell) auf den Laptop kopiert.

4 Trainings- und Testdaten

Die Bilder für das Training und die Auswertung werden mit einer Webcam aufgenommen, wie schematisch in Abbildung 3 zu sehen ist. Mit einem selbst geschriebenen Pythonskript wird automatisiert alle zwei Sekunden ein Bild mit der angeschlossenen Webcam aufgenommen und auf dem Raspberry Pi gespeichert. Die Aufnahmeauflösung beträgt 1920×1080 Pixel, um im späteren Verlauf die Möglichkeit zu haben, die Bilder beliebig herunter zu skalieren.

Abbildung 4: Eigene Aufnahmen mit der Webcam (1920×1080 Pixel)

In dem Beispielbild in Abbildung 4 ist links im Bild der DHL Truck aus Abbildung 2 zu erkennen.
Es werden mit diesem automatischen Aufnahmeskript nun fortlaufend Bilder von dem „Mikrowunderland“ aus der Vogelperspektive gespeichert. Da zum Aufnahmezeitpunkt die Fahrzeuge nicht autonom fahrbereit sind, muss der „truck“ manuell weiterbewegt werden und ein realistisches Fahrverhalten nachgeahmt werden. Insgesamt werden 500 Bilder aufgenommen.
Im nächsten Schritt werden alle Bilder händisch aussortiert und gelöscht, auf denen ein Arm oder eine Hand die Fahrzeuge verdeckt. Nach der Bereinigung bleiben 386 Bilder übrig.

4.1 Verarbeitung der Bilder

Für das Hinzufügen von Metainformationen zu den einzelnen Bildern wird das Tool „LabelImg“ [13] verwendet. Dabei wird für jedes aufgenommene Bild in Originalauflösung (1920×1080) in der Software ein Rechteck für den DHL Truck mit der Zuordnung zu der Klasse „truck“ hinzufügt (siehe Abbildung 5). Diese Metainformationen werden pro Bild als .xml-Datei gespeichert.

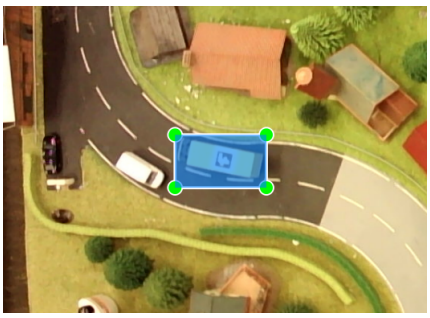


Abbildung 5: labelImg: Rechteck für die Position und Klassenzuordnung „truck“

Eine effiziente Verwendung der Bilder für maschinelles Lernen erfordert deutlich niedrig aufgelöstere Bilder. Dafür wird ein selbst geschriebenes Pythonskript verwendet, welches alle 386 Bilder auf einmal verkleinern kann. Die Bilder werden alle auf eine Auflösung von 300×300 Pixel reduziert. Dadurch wird auch das Seitenverhältnis von 16:9 auf 1:1 verändert.
Da Metainformationen für die Bilder absolute Positionen beinhalten, verändert das eigene Skript zusätzlich die .xml-Dateien der Bilder und berechnet die Positionen der Rechtecke neu und schreibt die .xml-Dateien entsprechend um.

Abbildung 6 zeigt drei beispielhafte Bilder in der Auflösung 300×300 Pixel, die für das Training verwendet werden.

Abbildung 6: Beispielbilder für den Trainingsprozess

4.2 Unterteilung des Datensatzes

Die 386 Bilder werden in drei disjunkte Datensätze aufgeteilt. Dafür sind 70 % der Bilder Teil des Trainingsdatensatzes, 20 % der Bilder Teil des Testdatensatzes und 10 % der Bilder werden für den Validierungsdatensatz ausgewählt.

Die Datensätze sind abschließend folgendermaßen aufgeteilt:

- 270 Bilder im Trainingsdatensatz
- 78 Bilder im Testdatensatz
- 38 Bilder im Validierungsdatensatz

4.3 Data Augmentation

Für eine künstliche Erhöhung der für das Training zur Verfügung stehenden Datenmenge wird sogenanntes „Data Augmentation“ eingesetzt. Durch diese Techniken wird die Anzahl der Bilder, die für das Training zur Verfügung stehen, deutlich erhöht. Auch die Varianz wird erhöht und das neuronale Netz ist weniger von den Originaldaten abhängig. [14, vgl.]

Dabei werden auf Basis des Ursprungsdatensatzes neue Bilder unter gegebenen Voraussetzungen erzeugt.

Es wurden insgesamt folgende Data Augmentation Methoden verwendet:

- Zufälliges Zuschneiden
- Zufällige Anpassung in Kombination aus:
  - Helligkeit
  - Hue
  - Kontrast
  - Sättigung
- Zufälliges horizontales Spiegeln
- Zufälliges vertikales Spiegeln
- Zufälliges drehen um 90 Grad

Alle diese Methoden werden dabei auch in Kombination verwendet.

(a) Zuschneiden und Rotation

(b) Zusätzliche Färbung

Abbildung 7: Beispiele für durchgeführte Data Augmentation

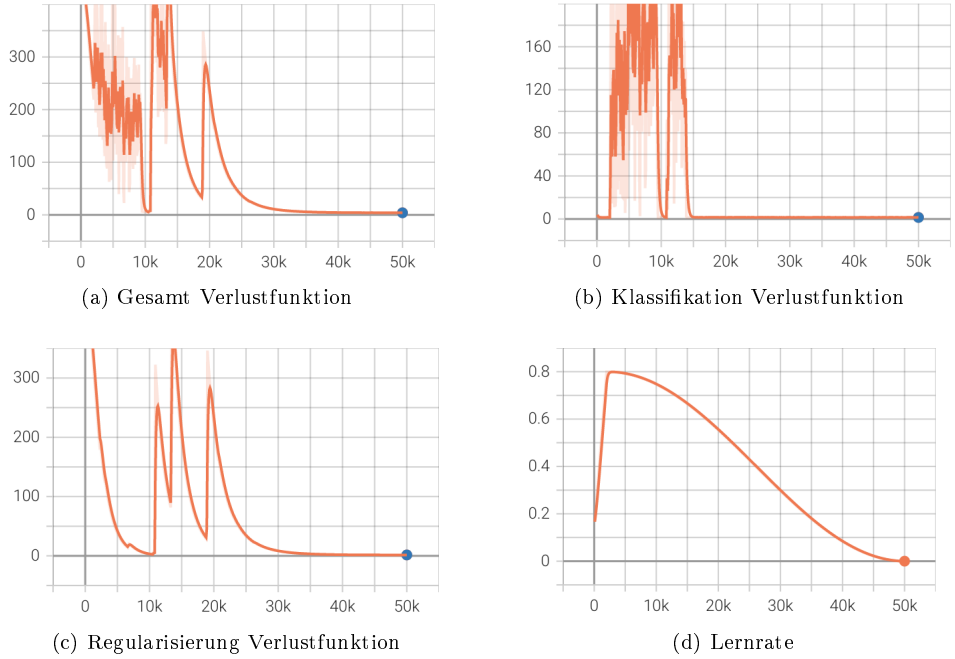
In Abbildung 7 ist zu erkennen, dass bei der durchgeführten Data Augmentation neue Bilder erzeugt werden, die von den ursprünglichen Bildern relativ stark abweichen. Wie in Abbildung 7a zu erkennen, wir damit auch erzielt, dass nicht in jedem Bild ein „truck“ enthalten ist. Somit kann das neuronale Netz auch lernen, dass nicht immer ein „truck“ zu suchen ist und ist nicht wie in den eigenen Aufnahmen auf Bilder beschränkt, die immer einen „truck“ enthalten und die Kamera optimal positioniert ist.
Durch die zusätzlichen zufälligen Färbungen in Abbildung 7b wird versucht zu verhindern, dass das neuronale Netz sich ausschließlich auf einzelne Farbkontraste verlassen kann.

5 Trainingsprozess

Für das Training mit TensorFlow 2 werden alle Trainings- und Testdaten in ein .record-Dateiformat umgewandelt. [15] Dazu werden alle Trainingsbilder mit den zugehören .xml-Dateien verknüpft und in eine einzelne *training.record* Datei gebündelt. In gleicher Weise wurden alle Testbilder in eine *test.record* gebündelt.
Die beiden .record-Dateien werden anschließend zusammen mit einer „LabelMap“ für das Training verwendet. Eine „LabelMap“ beinhaltet jedes Label (Klasse), welche durch das neuronale Netz trainiert und erkannt werden soll. In diesem Fall enthält sie nur das Label „truck“ mit der ID *1*. Für das Training wurde eine Batch Größe von *64* festgelegt.

Nach einem kurzen Testlauf auf einem MacBook Pro 2020 [16], welcher pro Trainingsschritt ca. 10 Sekunden gebraucht hat, wird das Training auf die Informatik Compute Cloud (ICC) der HAW Hamburg [17] portiert. Die in der ICC verwendete Grafikkarte ist die NVIDIA Tesla V100. [18] Für das Training werden alle für das benötigten Dateien auf die ICC hochgeladen. Anschließend wird das Training dort neu gestartet. Auf der ICC beträgt die Zeit pro Trainingsschritt im Durchschnitt ca. 1,01 Sekunden. Das entspricht

einer Geschwindigkeitserhöhung von ungefähr *890 %*. Am Anfang des Trainings beträgt die Zeit pro Trainingsschritt sogar nur 0,67 Sekunden. Für die ersten 20.000 Schritte entspricht das sogar einer Geschwindigkeitserhöhung von ca. *1.390 %*.



(a) Gesamt Verlustfunktion

(b) Klassifikation Verlustfunktion

(c) Regularisierung Verlustfunktion

(d) Lernrate

Abbildung 8: Trainingsverlauf

In den Abbildungen 8 lässt sich der Verlauf von mehreren Werten über das Training hinweg beobachten. Für den Verlust der Lokalisierung wird die *Smooth* L_1 Verlustfunktion verwendet. [19] Für den Verlust der Klassifikation wird eine *gewichtete Sigmoid Focal* Verlustfunktion verwendet. „Focal“ Verlustfunktionen sind eine für Object Detection verbesserte Version von „Cross-Entropy Verlustfunktionen“. [20, vgl.]

In der Übersicht der gesamten Verlustfunktion lässt sich in Abbildung 8a gut erkennen, dass das trainierte Modell bis ungefähr 10.000 Trainingsschritte starke Variationen im Verlust hat, aber kontinuierlich besser wird. Anschließend ist sowohl bei ca. 10.000 Trainingsschritten als auch bei ca. 20.000 Trainingsschritten zu erkennen, dass der Verlust stark ansteigt und danach wieder absinkt. Dies lässt sich auf den Verlust der Regularisierung in Abbildung 8c zurückführen.

Der Verlust der Klassifikation in Abbildung 8b ist bis ca. 15.000 Schritte recht hoch und fällt dann schlagartig auf ungefähr 1,5 herunter.

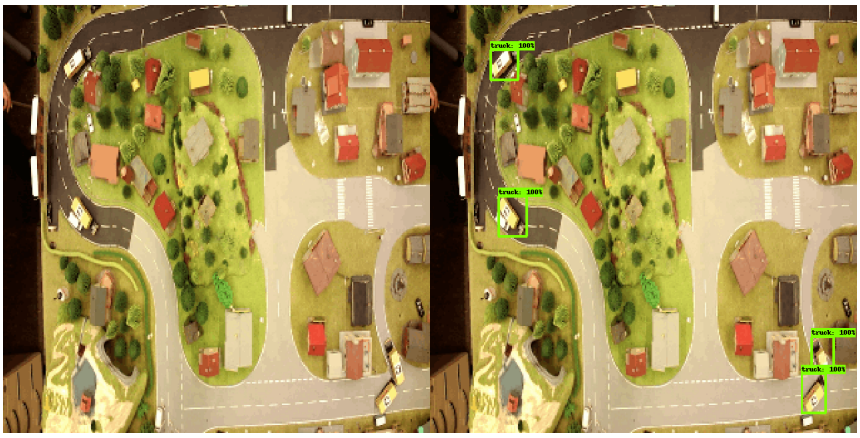
In Abbildung 8d ist zu beobachten, wie die Lernrate sich dem Trainingsfortschritt dynamisch anpasst. Anfangs ist der Gesamtverlust in Abbildung 8a sehr hoch, weshalb die Lernrate stark bis auf 0,8 ansteigt. Hiernach ist zu beobachten, dass die Lernrate erst

langsam, danach relativ konstant abnimmt und kurz vor Ende des Trainings dann gegen 0 konvergiert.

6 Auswertung

Die Ergebnisse dieser Arbeit lassen sich in den Abbildungen 9 gut erkennen. In den Beispielbildern in Abbildung 9 wird jeweils ein Eingabebild (links dargestellt) in das neuronale Netz, welches in Abschnitt 2.2 beschrieben wurde, hinein gegeben. Auf der rechten Seite ist dann die dazugehörige Ausgabe zu erkennen. Dafür werden über dem Eingabebild grüne Rechtecke für jedes erkannte Objekt dargestellt, mit der jeweiligen Zuordnung der erkannten Objektklasse. Zu beachten ist, dass alle diese Bilder von dem neuronalen Netz *nicht* für das Training oder das Testen verwendet werden und somit völlig neu für das Netz sind.
Diese Bilder sind Teil des Validierungsdatensatzes, welcher in Abschnitt 4.2 eingeführt wurde. Alle 38 Bilder des Validierungsdatensatzes wurden korrekt lokalisiert und klassifiziert.
So ist in Abbildung 9a zu erkennen, wie ein „truck“ erfolgreich lokalisiert und klassifiziert wird. In Abbildung 9b ist zu sehen, dass auch mehrere „trucks“ gleichzeitig bestimmt und korrekt dargestellt werden.
Die Anzeige der Wahrscheinlichkeit für die jeweiligen Fahrzeuge, die als „truck“ deklariert werden, wird mit 100 % angegeben. Dies ist darauf zurückzuführen, dass sich dieser Prototyp auf eine einzige Klasse („truck“) beschränkt und das neuronale Netz gelernt hat, dass nur die Klassifikation „truck“ korrekt ist. Technisch ist dies nicht limitiert und es ist experimentell getestet, dass auch andere Klassen erkannt werden könnten. Da es für die anderen Klassen keine ausreichenden Trainingsdaten zur Verfügung stehen, wird in dieser Arbeit, wie eingangs beschrieben, nur die Auswertung für die Klasse „truck“ dargestellt.

(a) Beispiel mit einem einzelnen „truck“ (links: Eingabe, rechts: Ausgabe)



(b) Beispiel mit mehreren „trucks“ (links: Eingabe, rechts: Ausgabe)

Abbildung 9: Beispiel Objekt Erkennungen

Abbildung 10: Ausschnitt aus einem Beispiel (keine Interpolation der Pixel)

In Abbildung 10 lässt sich gut erkennen, dass die ArUco Marker, die auf den DHL Truck geklebt sind, nicht mehr zu identifizieren sind. Aufgrund der niedrigen Auflösung von 300×300 Pixel lassen sich die Details aus der Entfernung von der Kamera zum Fahrzeug nicht erkennen. Eine klassische Erkennung der Fahrzeugpositionen würde sich so nicht mehr realisieren lassen. [21, vgl.]

Die Performance für die Verwendung auf einem Raspberry Pi 4 ist mit ca. 3 bis 5 verarbeiteten Bildern pro Sekunde akzeptabel. Dabei ist zu beachten, dass diese Werte ohne einen „Google Coral USB Accelerator“ erreicht werden, welcher in Abschnitt 2.3 beschrieben wird.

7 Schluss

7.1 Fazit

Diese Arbeit hat sich mit der Positionsbestimmung von Fahrzeugen im „Mikrowunderland“ der HAW Hamburg mit Object Detection beschäftigt. Dabei wurde gezeigt, dass mittels Transfer Learnings ein guter Erfolg für die Positionsbestimmung eines Fahrzeugs erreicht werden konnte.
Mit einer geringen Auflösung von 300×300 Pixel können für diesen Anwendungsfall kostengünstige Komponenten (Kamera, Mikroprozessor) für die Verarbeitung verwendet werden. Eine herkömmliche Methode, wie die ArUco Marker in Abschnitt 6 beschrieben, würde nicht mit dieser Auflösung funktionieren können.
Zudem wurde dargestellt, dass das Modell mit weniger als 400 Trainingsbildern unter Verwendung von zusätzlicher „Data Augmentation“ trainiert werden kann.

7.2 Ausblick

Für die reelle Anwendung würde die Kamera, wie in dieser Arbeit beschrieben, im „Mikrowunderland“ der HAW Hamburg fest installiert werden. Dabei würde die Position der

Kamera genau kalibriert werden müssen, um auf Basis der Daten des neuronalen Netzes eine exakte Berechnung der tatsächlichen Positionen zu ermöglichen.
Zudem ist die Verwendung eines „Google Coral USB Accelerator“ sinnvoll, wie in Abschnitt 2.3 beschrieben, um die Performance deutlich zu erhöhen.
Es könnten neue Klassen, wie „car“ oder „tractor“, und neue Fahrzeuge für alle Klassen hinzugefügt werden. Dabei ist es entscheidend, dass für jedes Fahrzeugmodell genug Trainingsbilder zur Verfügung stehen, um ein hinreichend gutes Ergebnis zu erzielen.

Literatur

[1] E. Alpaydin, *Maschinelles Lernen*. De Gruyter Oldenbourg, 2019, online abrufbar unter https://www.degruyter.com/document/doi/10.1515/9783110617894/html; abgerufen am 27. Februar 2022.; ISBN: 978-3-11-061789-4.

[2] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. N. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *CoRR*, vol. abs/2104.11892, 2021. [Online]. Available: https://arxiv.org/abs/2104.11892

[3] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *CoRR*, vol. abs/1911.02685, 2019. [Online]. Available: http://arxiv.org/abs/1911.02685

[4] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," *CoRR*, vol. abs/1808.01974, 2018. [Online]. Available: http://arxiv.org/abs/1808.01974

[5] EMEK, "Dhl truck," 2015, online abrufbar unter https://images-eu.ssl-images-amazon.com/images/I/41AgDzsyAgL._SR600%2C315_PIWhiteStrip%2CBottomLeft%2C0%2C35_SCLZZZZZZZ_FMpng_BG255%2C255%2C255.jpg; abgerufen am 27. Februar 2022.

[6] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[7] TensorFlow, "Tensorflow 2 detection model zoo," 2021, online abrufbar unter https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md; abgerufen am 05. Januar 2022.

[8] Microsoft, "Microsoft coco: Common objects in context," 2015, https://cocodataset.org/; Paper online abrufbar unter https://arxiv.org/pdf/1405.0312.pdf; abgerufen am 18. Januar 2022.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[10] Google, "Coral usb accelerator," 2020, online abrufbar unter https://coral.ai/products/accelerator/; abgerufen am 25. Februar 2022.

[11] E. Upton, "Raspberry pi 3," 2016, online abrufbar unter https://www.raspberrypi.com/news/raspberry-pi-3-on-sale/; abgerufen am 21. Februar 2022.

[12] Google, "Coral usb accelerator benchmarks," 2020, online abrufbar unter https://coral.ai/docs/edgetpu/benchmarks/; abgerufen am 25. Februar 2022.

[13] D. „tzutalin", "Labelimg," https://github.com/tzutalin/labelImg, 2015-2022.

[14] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, Jul 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0

[15] TensorFlow, "Tfrecorddataset," online abrufbar unter https://www.tensorflow.org/api_docs/python/tf/data/TFRecordDataset; abgerufen am 01. März 2022.

[16] Apple, "Macbook pro (13 zoll, 2020)," online abrufbar unter https://support.apple.com/kb/SP819; abgerufen am 01. März 2022.

[17] HAW-Hamburg, "Informatik compute cloud (icc)," online abrufbar unter https://userdoc.informatik.haw-hamburg.de/doku.php?id=docu:informatikcomputecloud; abgerufen am 26. Februar 2022.

[18] Nvidia, "Nvidia tesla v100," online abrufbar unter https://www.nvidia.com/de-de/data-center/tesla-v100/; abgerufen am 08. Februar 2022.

[19] N. Khanzhina, A. Lapenok, and A. Filchenkov, "Towards robust object detection: Bayesian retinanet for homoscedastic aleatoric uncertainty modeling," *CoRR*, vol. abs/2108.00784, 2021. [Online]. Available: https://arxiv.org/abs/2108.00784

[20] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

[21] OpenCV, "Detection of aruco markers," 2022, online abrufbar unter https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html; abgerufen am 16. Februar 2022.