Erheben einer Verkehrsstatistik durch Klassifizierung von Verkehrsobjekten unter Verwendung eines neuronalen Netzes

Studienarbeit - T3201

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Florian Drinkler, Luca Stanger

11. Juni 2021

Bearbeitungszeitraum	06.10.2020 - 11.06.2021
Matrikelnummer, Kurs	6653948, 7474265, TINF-18B
Ausbildungsfirma	Balluff GmbH, camos GmbH, Stuttgart
Betreuer	Sebastian Trost, Telefónica Germany

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anderslautende Genehmigung der Ausbildungsstätte vorliegt.

Stuttgart, 11. Juni 2021

Florian Drinkler, Luca Stanger

Erklärung

Wir versicheren hiermit, dass wir unsere Studienarbeit - T3201 mit dem Thema: *Erheben einer Verkehrsstatistik durch Klassifizierung von Verkehrsobjekten unter Verwendung eines neuronalen Netzes* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versicheren zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 11. Juni 2021

Florian Drinkler, Luca Stanger

Abstract

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CPU	Central Processing Unit
CNN	Convolutional Neural Network
GPU	Graphical Processing Unit
HTML	Hypertext Markup Language
PDF	Portable Document Format
MSE	Mean Squared Error
SVM	Support Vector Machine
TPU	Tensor Processing Unit
OvR	One-vs.-Rest
OvO	One-vs.-One
XLA	Accelerated Linear Algebra

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In der heutigen Zeit werden Algorithmen immer häufiger eingesetzt, um verschiedenen Personengruppen die Auswertung von Daten leichter zu machen. Maschinelle Lernverfahren bieten mit der Zeit immer fortgeschrittenere Möglichkeiten, unterschiedlichste Alltagssituationen zu analysieren. Im Zusammenhang mit dem Thema Verkehrsanalyse bietet das maschinelle Lernen vielfältige Möglichkeiten zur Verbesserung des öffentlichen Raums. Hauptbestandteil dieser Arbeit soll es sein, eine Verkehrsstatistik unter Verwendung eines Klassifikationsalgorithmus zu erstellen. Die daraus gewonnenen Erkenntnisse werden in dieser Arbeit wiedergegeben. Aus den resultierenden Ergebnissen wird eine einfache Ableitung über das Verkehrsaufkommen erreicht.

1.1 Motivation und Problemstellung

1.2 Zielsetzung

1.3 Methodik und Aufbau der Arbeit

2 Grundlagen und Stand der Forschung

2.1 Implementierungsumgebung Jupyter

Jupyter Notebooks ist eine von der non-profit Organisation Project Jupyter entwickelte Open-Source Lösung zur interaktiven Arbeit mit Dutzenden Programmiersprachen [Jup21b]. Der Name Jupyter leitet sich dabei von den drei primären Programmiersprachen Julia, Python und R ab. Jupyter Notebooks ist sprachunabhängig und unterstützt, unter Verwendung des IPython Kernel, die Programmiersprachen Julia, R, Haskell, Ruby und Python [Jup21a]. Darüber hinaus werden unterschiedlichste Export Möglichkeiten wie Hypertext Markup Language (HTML), Portable Document Format (PDF) und LaTeX unterstützt. Die in diesem Projekt verwendete Variante von Jupyter Notebooks ist Google Colab, eine speziell für die Python-Entwicklung entworfene Umgebung. Colab Notebooks führen Code auf Cloud-Servern aus und bieten somit unabhängige Vorteile gehosteter Hardware, wie Graphical Processing Units (GPUs) und Tensor Processing Units (TPUs) [Col21].

2.2 Bildverarbeitungsalgorithmen mit OpenCV

2.3 Maschinelle Lernverfahren

Maschinelle Lernverfahren lassen sich in drei Bereiche unterteilen. Sejnowski beschreibt in seinem Buch *Unsupervised Learning - Foundations of Neural Computation* das Unüberwachte Lernen *(unsupervised learning)* als maschinelles Lernverfahren, das ohne zuvor bekannte Werte oder Belohnungen, Abweichungen vom strukturlosen Rauschen erkennt [Sej99]. Ferner wird von Duda et al. die automatische Segmentierung (Clustering) und die Komprimierung von Daten zur Dimensionsreduktion erwähnt, die zum fortwährenden Erfolg der Lernmethode beitragen [DH+73, S. 51 f.; CC08, S. 51 f.]. Als eine weitere maschinelle Lernmethode führt Cord et al. das Überwachte Lernen *(supervised learning)* an. Das Überwachte Lernen beinhaltet das Lernen einer Abbildung zwischen einem Satz von Eingangsvariablen X und einer Ausgangsvariablen Y, sowie die Anwendung dieser

Abbildung zur Vorhersage der Ausgabe für ungesehene Daten [CC08, S. 21 ff.]. Cord et al. nennt 2008 als verbreitestes Modell die Support Vector Machine (SVM), die ihre Stärken besonders in der Verarbeitung von Multimedialen Daten besitzt. Im Paradigma des überwachten Lernens besteht das Ziel darin, eine Funktion $f : X \longrightarrow Y$ aus einem Beispiel- oder Trainingssatz A_n abzuleiten [CC08, S. 22]. Sei hierzu $x_i \in X$ und $y_i \in Y$:

$$A_n = ((x_1, y_1), \ldots, (x_n, y_n)) \in (X \times Y)^n. \tag{2.1}$$

Ein weiterer Fundamentaler Bestandteil des überwachten Lernens ist der Begriff des Verlusts zur Messung der Übereinstimmung zwischen der Vorhersage $f(x)$ und der gewünschten Ausgabe y. Hierfür wird eine Verlustfunktion $L : Y \times Y \longrightarrow \mathbb{R}^+$ zur Evaluierung des Fehlers eingeführt. Die Wahl der Verlustfunktion $L(f(x), y)$ hängt von dem zu lösenden Lernproblem ab [CC08, S. 22].

Zu unterscheiden sind hierbei drei verschiedenen Arten - *Binäre*, *Multi-Class* sowie *Multi-Label* Klassifikation. Für erstere ist es das Ziel, die Ausgabe eines messbaren zufälligen Klassifikators, parametrisiert durch $f : X \to [0, 1]$, der zwischen positiven ($Y = 1$) und negativen ($Y = 0$) Instanzen unterscheidet, zu erzeugen [MW18, S. 2 f.]. Ein zufälliger Klassifikator sagt jedes $x \in X$ mit der Wahrscheinlichkeit $f(x)$ als positiv voraus; die Qualität eines solchen Klassifikators wird durch ein statistisches Risiko $R(\cdot; D) : [0, 1]^X \to \mathbb{R}_+$ bewertet [MW18, S. 3]. Die standardmäßig gewählte Verlustfunktion $L(f(x), y)$ für Binäre Klassifikation ist die Binäre Kreuzentropie (*binary cross entropy*), definiert als

$$CE = -\sum_{i=1}^{C'=2} t_i \log(s_i) = -t_1 \log(s_1) - (1 - t_1 \log(1 - s_1). \tag{2.2}$$

Es wird von zwei Klassen C_1 und C_2 ausgegangen. $t_1 \in [0, 1]$ und s_1 sind die Grundwahrheit und der Wert für C_1; $t_2 = 1 - t_1$ sowie $s_2 = 1 - t_2$ für C_2 [RK14].

Sollte der vorgegebene Raum $[0, 1]$ für die Klassifizierung nicht ausreichen, kann die *Multi-Class Classification* eingesetzt werden. Hierfür wird die Binäre Klassifikation durch eine Auswahl verschiedener Strategien an das vorgegebene Lernproblem angepasst. Eine hierbei verwendete Methode ist die Transformation der Problemstellung in den binären Raum, welche mit Hilfe der One-vs.-Rest (OvR) oder One-vs.-One (OvO) Strategie umgesetzt werden kann. Die OvR-Strategie beinhaltet dabei das Training eines einzelnen Klassifikators pro Klasse, wobei die Proben dieser Klasse als positive Proben und alle anderen Proben als negative Proben gelten. Diese Strategie erfordert, dass die Basisklassifikatoren einen realwertigen Konfidenzwert für ihre Entscheidung erzeugen und nicht nur ein Klassenetikett; diskrete Klassenetiketten allein können zu Mehrdeutigkeiten führen,

bei denen mehrere Klassen für eine einzelne Probe vorhergesagt werden [Bis06, S. 182]. Bei der OvO-Strategie hingegen werden $K(K-1)/2$ binäre Klassifikatoren für ein K-Wege-Mehrklassenproblem trainiert. Jeder Klassifikator K erhält die Proben eines Klassenpaares aus der ursprünglichen Trainingsmenge und muss lernen, diese beiden Klassen zu unterscheiden [Bis06, S. 339]. Wie OvR leidet auch OvO unter Mehrdeutigkeiten, da einige Regionen des Eingaberaums die gleiche Anzahl von Stimmen erhalten können [Bis06, S. 183]. Neben dem Einsatz der Transformation in den binären Raum, kann ein binärer Klassifikator auch zur Lösung von Mehrklassen Problemen erweitert werden. Aufgrund der Bedeutsamkeit dieser Anpassungstechnik in Kontext dieser Arbeit, wird hierauf in Kapitel 2.4.2 tiefer eingegangen.

Optimierungsverfahren für maschinelle Lernmethoden

Damit eine Vielzahl von optimalen Lösungen für verschiedene Fragestellungen ermittelt werden kann, kommen Optimierungsverfahren zum Einsatz. Der Grundgedanke dieser ist es, die Parameter iterativ so zu verändern, dass eine Kostenfunktion minimiert wird.

Gradientenverfahren

Das Gradientenverfahren ist anwendbar, um eine differenzierbare Funktion $f : \mathbb{R}^n \to \mathbb{R}$ mit einem reellen Wert zu erhalten, die minimiert werden soll:

$$\min_{x \in \mathbb{R}^n} f(x). \tag{2.3}$$

Das Verfahren berechnet den lokalen Gradienten der Fehlerfunktion, entsprechend dem Parametervektor θ und bewegt sich in Richtung eines abnehmenden Gradienten. Sobald die Steigung Null annimmt, ist ein Minimum erreicht. Zu Beginn wird θ mit Zufallszahlen initialisiert, welche anschließend in kleinen Schritten verbessert werden um die Kostenfunktion[1] zu senken. Dieser Schritt wird wiederholt, bis der Algorithmus bei einem lokalen Minimum konvergiert [Gér17, S. 112 f.] (siehe Abbildung 2.1).

Bei der iterativen Suche eines Minimums ist dabei die Schrittgröße zu beachten. Diese wird durch die zuvor definierte Lernrate ermittelt, die als Hyperparameter der Gleichung anzusehen ist. Ist die Lernrate zu klein, steigen die Schritte des Algorithmus, was ihn langsam und ineffizient werden lässt. Eine zu große Lernrate kann hingegen zur Divergenz des Algorithmus führen [Gér17, S. 114].

[1] Als Kostenfunktion kann z.B. der Mean Squared Error (MSE) eingesetzt werden.

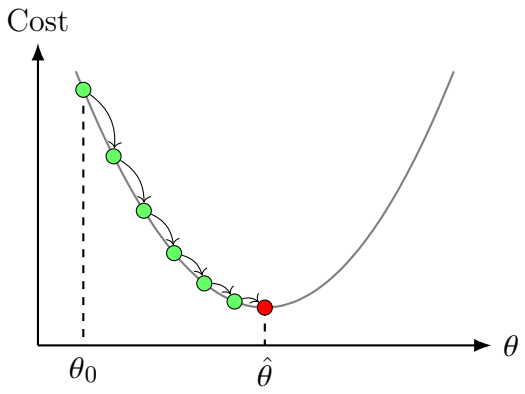

Abbildung 2.1: Gradientenverfahren mit zufälligem Startvektor θ_0 und lokalem Minimum $\hat{\theta}$

2.4 Datenstromorientierte Programmierung mit TensorFlow

TensorFlow ist eine Open-Source-Bibliothek für maschinelles Lernen, die von Google seit 2017 angeboten wird. Sie wurde vom Google Brain Team entwickelt und ermöglicht dank vieler Application Programming Interfaces (APIs) Deep-Learning, um Aufgaben effizient zu lösen. Seit Ende 2019 ist TensorFlow 2 mit Keras-Integration verfügbar. Der Kern von TensorFlow ist in der Programmiersprache C++ implementiert und ermöglicht die Ausführung von Operationen auf unterschiedlichen Hardware-Basen (CPU, GPU und TPU) und Betriebssystemen. Parallel dazu wurde zur Verbesserung der Ausführungszeiten und der Speicheroptimierung Accelerated Linear Algebra (XLA) entwickelt, ein Compiler, der spezifische mathematische Funktionen für TensorFlow optimiert und diese unabhängig von der Hardware ausführen kann [DN19, S. 139 f.].

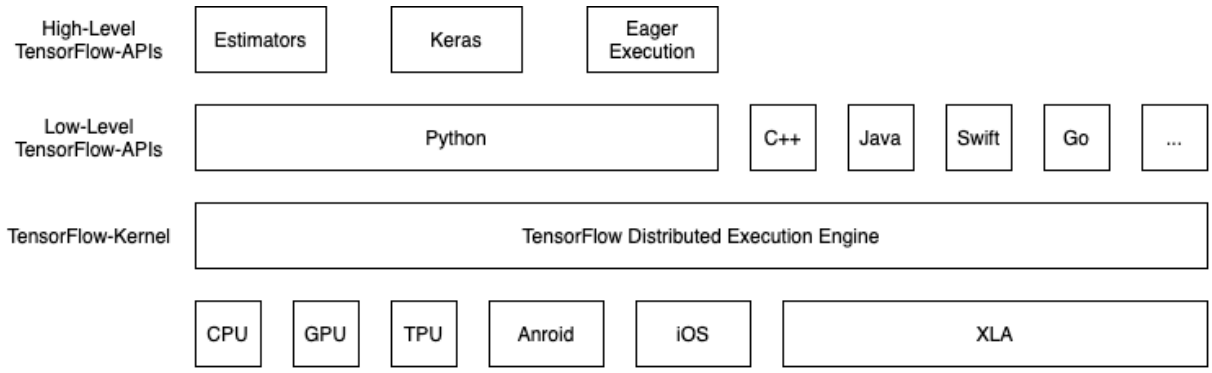

Abbildung 2.2: API-Struktur von TensorFlow 2 (grafisch adaptiert von [DN19, S. 140][Tea17])

2.4.1 Tensoren

Grundlegender Bestandteil TensorFlows sind die schon im Namen enthaltenen Tensoren.

2.4.2 Künstliche neuronale Netze

Die Ursprünge der künstlichen neuronalen Netze lassen sich auf McCulloch et al. im Jahre 1943 zurückführen [MP43]. Eine von Donald O. Hebb 1949 formulierte Lernregel stellt seither in ihrer allgemeinen Form die Grundlage der künstlichen neuronalen Lernverfahren dar [Mai97]. Ein künstliches neuronales Netz besteht aus einer Eingabeschicht von Neuronen, 1..n versteckter Schichten und einer letzten Schicht von Ausgangsneuronen. Ein einzelnes Neuron nimmt überlicherweise mehrere Werte $x_1, ..., x_n$ und einen Bias-Term w_0 als Eingabe und berechnet daraus die Ausgabe $y = h(z)$.

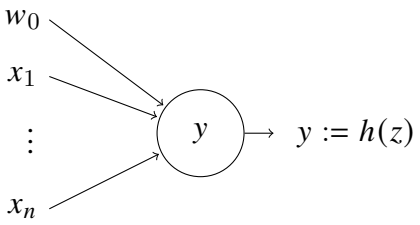

Abbildung 2.3: Einzelnes Neuron mit dessen Eingangsvariablen. Die Aktivierungsfunktion ist beschrieben als h und wird auf die tatsächlichen Eingabe z angewandt. $x_1, ..., x_n$ repräsentieren die Eingabe von anderen Neuronen innerhalb des Netzes. w_0 wird Bias genannt und repräsentiert ein externes Gewicht [Stu14].

Die Ausgabe h_i des Neurons i in der versteckten Schicht wird beschrieben durch

$$h_i = \varphi(\sum_{j=1}^{N} V_{ij} x_j + \theta_i^{hid}) \tag{2.4}$$

wo $\varphi(\cdot)$ die Aktivierungsfunktion, N die Anzahl der Eingangsneuronen, V_{ij} die Gewichte, x_j die Eingabe zum Neuron und θ_i^{hid} der Schwellenwertterm der versteckten Neuronen ist [Wan03, S. 81–100; NMS95, S. 195–201]. Die Intention der Aktivierungsfunktion $\varphi(\cdot)$ neben der Einführung von Nichtlinearität in das neuronale Netz ist, den Wert eines Neurons zu begrenzen, damit das neuronale Netz nicht durch divergierende Neuronen gelähmt wird. Eine gängige Aktivierungsfunktion ist die Sigmoid Funktion $\sigma(\cdot)$, wie definiert in 2.5.

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{2.5}$$

Weitere sigmoide Aktivierungsfunktionen sind der Arkustangens (arctan) und Tangens Hyperbolikus (tanh) [NMS95, S. 195–201]. Sie haben ein ähnliches Ansprechverhalten auf die Eingangswerte wie die Sigmoidfunktion, unterscheiden sich aber in den Ausgangsbereichen.

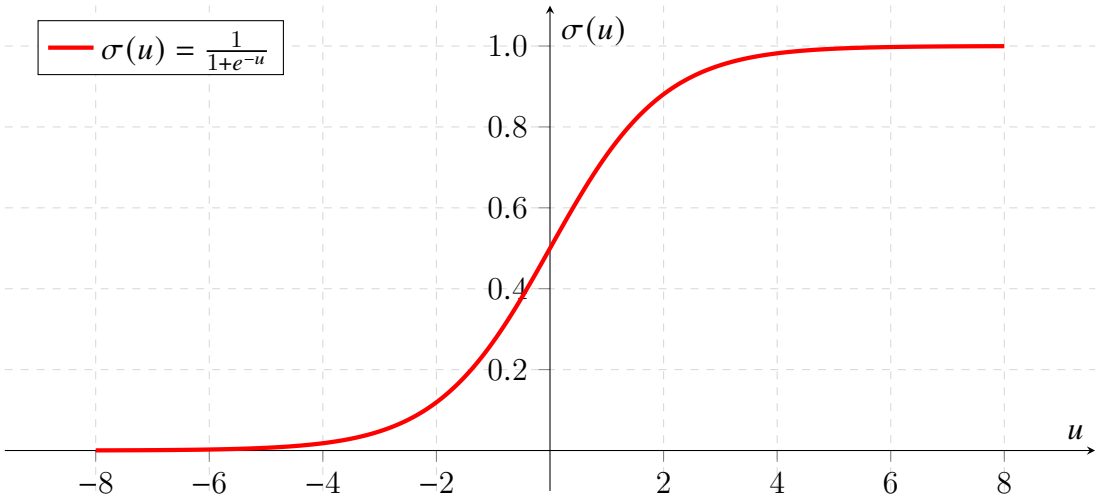

Abbildung 2.4: Darstellung der Sigmoid Aktivierungsfunktion

2.4.3 Convolutional Neural Networks

Seitdem AlexNet im Jahre 2012 eine Auszeichnung beim jährlichen Wettbewerb der Benchmark-Datenbank ImageNet erzielte [Ima12], hat sich der Forschungsfokus auf das Themengebiet *Deep Learning* zubewegt [KSH12; Ras+16; Rus+15]. Zuvor waren SVMs der prävaliernde Ansatz zur Erkennung von Mustern.

Convolutional Neural Networks (CNNs) werden seit 1995 in der digitalen Bildverarbeitung eingesetzt und sind fester Bestandteil des *Deep Learnings*. Es werden Faltmatrizen der Größe 3x3, 5x5, 7x7 bzw. 9x9 eingesetzt, um Bereiche der Eingabematrix sukzessiv zu analysieren. Die dabei verwendeten *convolutional operations* (Faltoperationen) erzeugen rezeptive Felder, die eine Merkmalskarte (*feature map*) des CNN generieren [Rus+15]. Die rezeptiven Felder korrespondieren mit einer Region aus dem Originalbild [Yan20].

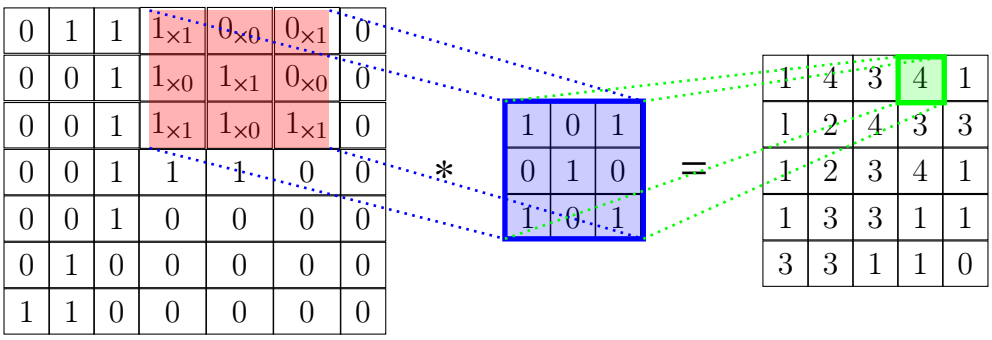

Abbildung 2.5: Erzeugen einer Merkmalskarte durch schrittweise Faltung

In der Mathematik wird die Faltung als eine Operation auf zwei Funktionen f, g beschrieben, die eine dritte Funktion $f * g$ erzeugt. Die dritte Funktion beschreibt, wie die Form von f durch g verändert oder gefiltert wird. Für eine Position $z_{i,j}$ in der Ausgabe gilt

$$z_{i,j} = b + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} x_{i+u,j+v} \cdot w_{u,v} \tag{2.6}$$

worin $z_{i,j}$ die Position innerhalb der Matrix z beschreibt und b der Bias ist [Kar20, S. 6]. Betrachtet man nun die Position $z_{i,j}$ in der Ausgabe eines Layers gilt

$$z_{i,j} = b + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} x_{i',j'} \cdot w_{u,v} \qquad \begin{cases} i' = i \cdot s_h + u \\ j' = j \cdot s_w + v \end{cases} \tag{2.7}$$

worin s_h der vertikale und s_w der horizontale Stride sind [Kar20, S. 13 f.]. Der Stride ist eine Komponente des CNN, abgestimmt auf die Kompression des Eingabedatensatzes. Weitergehend wird er als Parameter des CNN-Filters bezeichnet, der die Bewegung über die Eingabematrix bestimmt.

Die darauffolgende *Volume Convolution* erweitert die Gleichung um einen Parameter k , der die Anzahl der Farbräume in die Gleichung einbezieht [Kar20, S. 25; Gér17, S. 365]. Es gilt

$$z_{i,j,k'} = b_{k'} + \sum_{c=1}^{k} \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} x_{i',j',c} \cdot w_{u,v,c,k'} \qquad \begin{cases} i' = i \cdot s_h + u \\ j' = j \cdot s_w + v \end{cases} \tag{2.8}$$

Die Anzahl der Parameter eines CNN ist unabhängig von der Eingabe, jedoch abhängig von der Größe des Filters [Kar20, S. 28]. Allgemein gilt daher

$$\#\,\text{Params}_{conv} = (f_w \cdot f_h \cdot k^{l-1} + 1) \cdot k^l \tag{2.9}$$

2.5 Deep Learning mit Keras

2.6 Verwandte Arbeiten

3 Analyse der Datenströme

3.1 Anforderungen an die Analyse

3.2 Datenaufbereitung

3.2.1 Datenerhebung und Integration

3.2.2 Datenberechnung

3.2.3 Datenaggregation

3.2.4 Datenbereinigung

4 Entwicklung des Modells

4.1 Vorverarbeitung der Daten

4.2 Entwurf eines Netzwerks zur Klassifikation von Objekten

4.3

4.4 Erheben einer Verkehrsstatistik

4.5 Statistische Bewertung des Modells

4.5.1 Determination aussagekräftiger Metriken

5 Prototypische Implementierung

5.1 Aufbau des Prototypen

5.2 Modellierung

5.3 Deployment

5.4 Anpassbarkeit

6 Evaluation des Prototypen

6.1 Metriken zur Bewertung der Klassifikation

6.2 Optimierung des neuronalen Netzes

6.3 Evaluierung der Ergebnisse

7 Abschluss

7.1 Fazit

7.2 Ausblick

Literatur

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer, 2006. ISBN: 978-0-387-31073-2.

[CC08] Matthieu Cord und Pádraig Cunningham. *Machine Learning Techniques for Multimedia - Case Studies on Organization and Retrieval*. Berlin Heidelberg: Springer Science und Business Media, 2008. ISBN: 978-3-540-75171-7.

[Col21] Google Colab. *Willkommen bei Colaboratory - Colaboratory*. 2021. URL: https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=OwuxHmxllTwN. (abgerufen am 01.02.2021).

[DH+73] Richard O Duda, Peter E Hart et al. *Pattern classification and scene analysis*. Bd. 3. Wiley New York, 1973.

[DN19] Matthieu Deru und Alassane Ndiaye. *Deep Learning mit TensorFlow, Keras und TensorFlow.js*. Bonn: Rheinwerk, 2019. ISBN: 978-3-836-26509-6.

[Gér17] Aurélien Géron. *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow : Konzepte, Tools und Techniken für intelligente Systeme* -. Sebastopol: O'Reilly, 2017. ISBN: 978-3-960-09061-8.

[Ima12] ImageNet. *ImageNet Large Scale Visual Recognition Competition*. 2012. URL: http://www.image-net.org/challenges/LSVRC/2012/results.html. (abgerufen am 01.02.2021).

[Jup21a] Project Jupyter. *Jupyter Kernels - jupyter/jupyter*. 2021. URL: https://github.com/jupyter/jupyter/wiki/Jupyter-kernels. (abgerufen am 01.02.2021).

[Jup21b] Project Jupyter. *Project Jupyter - Home*. 2021. URL: https://jupyter.org/. (abgerufen am 01.02.2021).

[Kar20] Michael Dr. Karl. *Grundlagen Maschineller Lernverfahren - Convolutional Neural Networks*. DHBW Stuttgart. 2020.

[KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. „ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Hrsg. von F. Pereira et al. Bd. 25. Curran Associates, Inc., 2012, S. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[Mai97] Klaus Mainzer. „Komplexität neuronaler Netze“. In: *Gehirn, Computer, Komplexität*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, S. 143–161. ISBN: 978-3-642-60524-6. DOI: 10.1007/978-3-642-60524-6_9. URL: https://doi.org/10.1007/978-3-642-60524-6_9.

[MP43] Warren McCulloch und Walter Pitts. „The Linear theory of Neuron Networks: The Static Problem“. In: *Bulletin of Mathematical Biology* 4.4 (1943), S. 169–175.

[MW18] Aditya Krishna Menon und Robert C Williamson. „The cost of fairness in binary classification“. In: *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*. Hrsg. von Sorelle A. Friedler und Christo Wilson. Bd. 81. Proceedings of Machine Learning Research. New York, NY, USA: PMLR, 23–24 Feb 2018, S. 107–118. URL: http://proceedings.mlr.press/v81/menon18a.html.

[NMS95] International Workshop on Artificial Neural Networks, Jose Mira und Francisco Sandoval. *From Natural to Artificial Neural Computation - International Workshop on Artificial Neural Networks, Malaga-Torremolinos, Spain, June 7-9, 1995 : Proceedings*. Berlin Heidelberg: Springer Science und Business Media, 1995. ISBN: 978-3-540-59497-0.

[Ras+16] Mohammad Rastegari et al. *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*. 2016. arXiv: 1603.05279 [cs.CV].

[RK14] R.Y. Rubinstein und D.P. Kroese. *The Cross-Entropy Method*. Springer, 2014. ISBN: 9781475743227. URL: https://books.google.de/books?id=7hj4sgEACAAJ.

[Rus+15] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[Sej99] Howard Hughes Medical Institute Computational Neurobiology Laboratory Terrence J Sejnowski. *Unsupervised Learning - Foundations of Neural Computation*. Cambridge: MIT Press, 1999. ISBN: 978-0-262-58168-4.

[Stu14] David Stutz. „Introduction to Neural Networks“. In: *RWTH Aachen University* (März 2014).

[Tea17] The TensorFlow Team. *Google Developers Blog: Introduction to TensorFlow Datasets and Estimators*. 2017. URL: https://developers.googleblog.com/2017/09/introducing-tensorflow-datasets.html. (abgerufen am 09.03.2021).

[Wan03] Sun-Chong Wang. „Artificial Neural Network“. In: *Interdisciplinary Computing in Java Programming*. Boston, MA: Springer US, 2003, S. 81–100. ISBN: 978-1-4615-0377-4. DOI: 10.1007/978-1-4615-0377-4_5. URL: https://doi.org/10.1007/978-1-4615-0377-4_5.

[Yan20] Wei Qi Yan. *Computational Methods for Deep Learning - Theoretic, Practice and Applications*. Singapore: Springer Nature, 2020. ISBN: 978-3-030-61081-4.

Glossar

AlexNet

Ein von Alexander Krizhevsky entworfenes CNN.

Bias

Unabhängiges Gewicht eines neuronalen Netzes.

Hyperparameter

Wert zur Steuerung des Lernprozesses.

Kernel

Ein Programm, das den Code des Anwenders ausführt und introspektiert.