Erkennung und Auslesen von Nummernschildern aus Bilddateien mithilfe von Deep Learning und Optical Character Recognition

Christian Peters

3. März 2021

Veranstaltung:	Fallstudien II
Dozent:	Prof. Dr. Markus Pauly
Gruppe:	Anne-Sophie Bollmann, Susanne Klöcker, Pia von Kolken, Christian Peters

Inhaltsverzeichnis

1 Einleitung

Die automatisierte Erkennung von Nummernschildern aus Bilddateien ist ein wichtiger Bestandteil vieler moderner Verkehrssysteme und kommt beispielsweise in Parkhäusern, Mautstellen oder bei der Identifikation gestohlener Fahrzeuge zum Einsatz [7].

Das Ziel dieses Projektes ist es, einen Prototypen für ein solches Erkennungssystem zu entwickeln, welcher in der Lage sein soll, erfolgreich Nummernschilder aus Bilddateien erkennen und auszulesen zu können.

Zu diesem Zweck kommen bei der Entwicklung des Prototypen Methodiken aus dem Bereich des Deep Learning, sowie der Optical Character Recognition zum Einsatz, die in Abschnitt 3 näher beschrieben werden. Ein Überblick über das vorliegende Datenmaterial, welches zum Training dieser Algorithmen wervendet wurde, wird in Abschnitt 2 gegeben. Abschließend werden die erzielten Ergebnisse in Abschnitt 5 beschrieben und die Stärken sowie die Schwächen des Prototypen diskutiert.

Der gesamte Quellcode dieses Projekts ist Open Source und kann unter `https://github.com/cxan96/license_plate_detection` abgerufen werden.

2 Datenbeschreibung

Der vorliegende Datensatz besteht aus insgesamt 949 Bildern von Autos mit Nummernschildern.[1] Die Bilder wurden in den Regionen Brasilien, Europa, Rumänien und USA aufgenommen.

Zu jedem Bild liegen außerdem Informationen zur Position des Nummernschildes innerhalb des Bildes anhand von Pixel Koordinaten vor. Die Position des Nummernschildes ist dabei durch die Koordinaten $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$ relativ zur linken oberen Ecke des Bildes eindeutig spezifiziert.

In Abbildung 1 sind drei Beispielbilder aus dem Datensatz dargestellt. Die Koordinaten der Nummernschilder sind anhand der roten Rechtecke eingezeichnet.

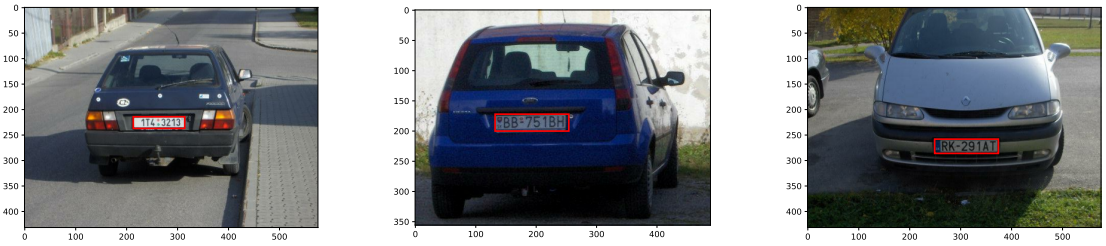

Abbildung 1: Drei Beispiele aus dem vorliegenden Datensatz. Die Nummernschilder sind anhand ihrer Koordinaten rot umrandet.

[1] Die Originaldaten sind unter `https://github.com/phibuc/Lab_FS_Data`, sowie unter `https://github.com/RobertLucian/license-plate-dataset` einsehbar.

3 Grundlagen

3.1 Neuronale Netze

Neuronale Netze sind eine Modellklasse, welche zur Lösung des bereits ausführlich in [5] beschriebenen Problems des statistischen Lernens eingesetzt werden können. In dieser Problemsituation wird angenommen, dass sich der Zusammenhang zwischen beobachtbaren Prädiktorvariablen $X_1, ..., X_p$, welche sich durch einen Vektor $X = (X_1, ..., X_p)$ zusammenfassen lassen, und einer Zielvariable Y durch eine Funktion f^* mit $Y = f^*(X)+\epsilon$ modellieren lässt. Hierbei kann ϵ als eine zufällige Störgröße angesehen werden, die hier im weiteren Verlauf aber keine wichtige Rolle spielt. Das Ziel von Neuronalen Netzen ist es, die unbekannte Funktion f^* zu approximieren.

Die folgenden Erklärungen zum Aufbau und zum Training neuronaler Netze stützen sich wesentlich auf [4] und sind hier auf das grundlegendste reduziert worden. Obwohl es der Name *neuronale* Netze suggeriert, wird auch hier genau wie in [4] ebenfalls auf jegliche biologische Motivation verzichtet, damit nicht der fälschliche Eindruck entstehen kann, dass es sich bei neuronalen Netzen um Modelle von echten biologischen Gehirnen handeln könnte. Das Ziel von neuronalen Netzen ist es viel eher, unbekannte Funktionen anhand von Trainingsdaten zu approximieren und die Ergebnisse dann auf ungesehene Daten zu generalisieren.

3.1.1 Aufbau

Wie in 3.1 angedeutet, definiert ein neuronales Netz also eine Abbildung f, welche den Zusammenhang zwischen einer Eingabe X und einer Ausgabe Y approximieren soll. Eine Hauptcharakteristik von neuronalen Netzen ist es, dass die Funktion f durch die Verkettung weiterer Funktionen gebildet wird. Ist ein neuronales Netz beispielsweise durch $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ gegeben, so setzt es sich aus der Verkettung der einzelnen Funktionen $f^{(1)}$, $f^{(2)}$ und $f^{(3)}$ zusammen. Wie genau diese Funktionen aussehen können, soll an dieser Stelle bewusst ersteinmal offen bleiben.

Solche Ketten von Funktionen können gut durch azyklische Graphen beschrieben werden. Hierbei wird jedes Zwischenergebnis durch einen Knoten repräsentiert, jede Kante zwischen zwei Knoten beschreibt die Operation, die von einen Ergebnis zum nächsten geführt hat. Der Beispielgraph zu $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ ist in Abbildung 2 zu sehen.

Im Kontext von neuronalen Netzen wird jede der Funktionen $f^{(1)}$, $f^{(2)}$ und $f^{(3)}$ auch als eine **Schicht** im neuronalen Netz bezeichnet. Da $f^{(1)}$ und $f^{(2)}$ im Inneren des Netzwerks liegen, bezeichnet man diese Schichten auch als **versteckte Schichten**. Die Gesamtanzahl der Schichten wird als die **Tiefe** des neuronalen Netzes bezeichnet.

Zusammenfassend lässt sich also sagen, dass sich ein neuronales Netz als eine Verkettung beliebiger Funktionen auffassen lässt, welche auf eine Eingabegröße X angewendet werden. Wie genau diese Funktionen, also die Schichten des neuronalen Netzes, aussehen können, wird nun im nächsten Abschnitt beschrieben.

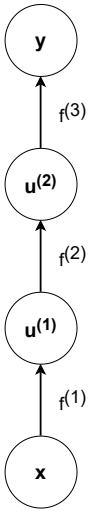

Abbildung 2: Verkettung dreier Funktionen dargestellt als azyklischer Graph.

3.1.2 Schichten

Eine Schicht eines neuronalen Netzes ist eine Funktion f, die eine Eingabe $x \in \mathbb{R}^n$ auf eine Ausgabe $f(x) \in \mathbb{R}^m$ abbildet. Häufig hat f dabei die folgende Form:

$$f(x) = g(Wx + b) \tag{1}$$

$W \in \mathbb{R}^{m \times n}$ ist hierbei eine sogenannte Gewichtsmatrix, die die Eingabe x der Schicht linear transformiert. $b \in \mathbb{R}^m$ wird auch Bias genannt und wird auf das Ergebnis der Multiplikation von W mit x addiert. Die Funktion $g : \mathbb{R}^m \to \mathbb{R}^m$ heißt Aktivierungsfunktion.

Oftmals ist es so, dass Aktivierungsfunktionen elementweise auf das Resultat von $Wx + b$ angewendet werden. Hierzu kann man sich eine Funktion $\Phi : \mathbb{R} \to \mathbb{R}$ definieren und dann $g_i(u) = \Phi(u_i),\ i = 1, ..., m$ setzen. Hierbei beschreibt g_i das i-te Element der vektorwertigen Funktion g und u_i das i-te Element des Vektors $Wx + b$.

Die Parameter W und b einer Schicht können während der Trainingsphase des neuronalen Netzwerks optimiert werden. Dieses Vorgehen wird in Abschnitt 3.1.4 näher beschrieben. Die Aktivierungsfunktion einer Schicht hingegen ist ein sogenannter **Hyperparameter** des Netzwerks. Dies bedeutet, dass dieser Parameter vor dem Training vom Anwender spezifiziert werden muss.

3.1.3 Aktivierungsfunktionen

Im Folgenden werden ausschließlich elementweise Aktivierungsfunktionen $\Phi : \mathbb{R} \to \mathbb{R}$ betrachtet, da diese die größte praktische Relevanz haben. Zwei sehr häufig eingesetzte Aktivierungsfunktionen, die auch in diesem Projekt verwendet wurden, sind die **Rectified Linear Unit (ReLU)** Funktion und die **Sigmoid** Funktion. Beide werden im Folgenden kurz beschrieben.

ReLU Die ReLU Funktion ist nach [4] quasi eine Standardempfehlung für die Aktivierungsfunktion in den versteckten Schichten moderner tiefer neuronaler Netze. Sie ist

durch folgenden Ausdruck gegeben:

$$\Phi_{\mathrm{ReLU}}(x) = \max\{0, x\}$$

Wird diese Funktion elementweise auf einen Vektor angewendet, so werden alle negativen Elemente auf Null gesetzt. Die restlichen Elemente bleiben unberührt.

Sigmoid Die Sigmoid Funktion ist durch folgenden Ausdruck gegeben:

$$\Phi_{\mathrm{Sigmoid}}(x) = \frac{1}{1 + \exp(-x)}$$

3.1.4 Training

3.1.5 Convolutional Neural Networks

3.2 OpenCV

3.3 Tesseract

4 Pipeline

5 Ergebnisse

6 Zusammenfassung

Literatur

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from `http://www.tensorflow.org`.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] F. Chollet. *Deep Learning with Python*. Manning Publications Co., 2017.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning*. Springer-Verlag New York, 2013.

[6] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. `http://neuralnetworksanddeeplearning.com/`.

[7] S. M. Silva and C. R. Jung. License plate detection and recognition in unconstrained scenarios. In *2018 European Conference on Computer Vision (ECCV)*, pages 580–596, Sep 2018.

[8] R. Smith. An overview of the tesseract ocr engine. *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 2:629–633, 2007.