End-to-end-Learning Ansatz für autonomes Fahren im Miniatur Wunderland

Nils-Ole Bickel, Michel Brüger

27. Februar 2020

Inhaltsverzeichnis

ToDo:
evtl Abstract schreiben
- end-to-end learning erklären
– netzt sagt für übertragene Bilder lenkwinkel voraus
– übermittelte Lenkwinkel als Label, Netz sagt für übertragene Bilder Lenkwinkel voraus

1 Hardware

1.1 Raspberry Pi 4

Die Hardware in dem Fahrzeug selbst ist nicht stark genug um aus den Kamerabildern den entsprechenden Lenkwinkel zu berechnen. Daher ist das Ziel, die Berechnung der Lenkwinkel auszulagern, auf einen Rechner, der im besten Fall unauffällig im *Miniatur-wunderland* "getarnt"werden kann.

Der *Raspberry Pi 4* hat im Verhältnis zu seiner Größe potente Hardware und verfügt vor allem auch über zwei USB-3-Anschlüsse. Diese sind wichtig um das volle Potential aus dem *Google Coral USB Accelerator* herauszuholen. Außerdem bietet er die Möglichkeit vollwertige Linux-Betriebssysteme zu installieren, was die Verwendung der für die AI-Lenkwinkel-Voraussagung notwendigen Software ermöglicht.

Die Möglichkeit, den *Raspberry Pi* über eine Powerbank per USB-C zu laden, macht es noch leichter ihn in der Kulisse zu verstecken, da man nicht noch ein Kabel zu einem Stromanschluss verlegen muss.

1.2 Google Coral USB Accelerator

Der *Google Coral USB Accelerator* ist ein Tensor-Prozessor in Form eines USB-Sticks, der an einen Rechner wie den *Raspberry Pi* angeschlossen werden kann, um die Machine Learning Operationen schnell und besonders energieeffizient durchzuführen. Dies kann der *Raspberry Pi 4* zwar auch alleine bewerkstelligen, der *Google Coral USB Accelerator* ist dabei aber deutlich schneller. Bei spontanen Vergleichen konnten wir eine 14-fache Geschwindigkeit bei der Objekterkennung in Bildern feststellen.

1.3 Das Auto

darüber wissen wir eigentlich nichts, vielleicht Section lieber wieder löschen?

1.4 Der Trainingsrechner

Das Netz wurde Trainiert auf einem der Laborrechner in Raum *???*. Dieser verfügt über einen *???* Prozessor und eine *Nvidia Geforce GTX 1050 Ti* Grafikkarte, welche den Trainingsprozess deutlich beschleunigt, im Vergleich zum Training mit dem Prozessor.

2 Software

2.1 Auf dem Raspberry Pi verwendete Software

2.1.1 Raspbian

Als Betriebssystem auf dem *Raspberry Pi 4* wird *Raspbian Buster* verwendet. Dabei handelt es sich um eine auf Debian basierende Linux-Distribution.

2.1.2 Python

Standardmäßig ist *Python 3.7* in *Raspbian Buster* enthalten. Dies ist für die Verwendung des *Google Coral USB Accelerator* mit *Tensorflow Lite* ausreichend.

2.1.3 Edge TPU runtime

Die *Edge TPU runtime* wird für die Kommunikation mit dem *Google Coral USB Accelerator* benötigt.

Während der Installation der *Edge TPU runtime* kann man auswählen, ob man den *Google Coral USB Accelerator* mit Standardgeschwindigkeit oder maximaler Geschwindigkeit (2x Standard) betreiben möchte. Da sich bei unseren Versuchen mit Standardgeschwindigkeit herausgestellt hat, dass der *Raspberry Pi 4* bei der Vorbereitung der Bilder überhitzt, ist eine Installation mit maximaler Geschwindigkeit überflüssig.

2.1.4 Tensorflow Lite Library für Python

Da auf dem *Raspberry Pi 4* lediglich *Tensorflow Lite* Modelle ausgeführt werden sollen reicht die Installation des *Tensorflow Lite interpreter* anstelle aller Tensorflow Pakete.

2.2 Auf dem Auto verwendete Software

- irgendwie schickt es einen Videostream...

2.3 Auf dem Trainingsrechner verwendete Software

Auf den Labor-PCs ist *Anaconda* installiert.

2.3.1 Anaconda

In *Anaconda* können unter Verwendung von *Conda* virtuelle Umgebungen (Environments) erstellt werden, in denen voneinander unabhängige *Python*-Installationen existieren können.

In unserer virtuellen Umgebung wird *Python 3.7* und die aktuelle Version von *Tensorflow* mit GPU Support installiert.

Beim Training wird ein normales *Tensorflow* Modell trainiert. Da auf dem *Raspberry Pi 4* nur *Tensorflow Lite* Modelle ausgeführt werden sollen wird dies mit Hilfe des *TFLiteConverter* von *Tensorflow* in ein *Tensorflow Lite* Modell umgewandelt.

- Virtual Environments
- Python
- Tensorflow
- Tensorflow lite converter (heisst der so?)

3 Das Netz

3.1 Implementierung

Das Netzwerk besteht aus 11 Layern. Bei 6 davon handelt es sich um Convolutional Layer. Die anderen 5 sind fully-connected Layer. Die Convolutional Layer sind für die Featureerkennung verantwortlich. Die ersten 4 haben einen Kernel der Größe 5x5 und Strides der Größe 2x2. Die anderen Convolutional Layer haben einen Kernen der Größe 3x3 und Strides der Größe 1x1. Anschließend kommen die 5 fully-connnnected Layer. Das Ergebnis beschreibt den Lenkwinkel in einer abstrakten Form. Genauer ist es die Länge des Vektors (X, Y), die der Hall Sensor, an der Vorderachse des Auto, bei diesem Lenkeinschlag misst.

Als Vorlage für das Netzwerk wurde das Netzwerk aus dem Paper [BTD+16] verwendet. Unsere Bilder haben eine Größe von 300x800. Dies ist deutlich größer als im Paper, weshalb das Netzwerk um weitere Layer, auf das oben beschriebene, erweitert wurde.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 398, 32)	2432
conv2d_1 (Conv2D)	(None, 72, 197, 64)	51264
conv2d_2 (Conv2D)	(None, 34, 97, 64)	102464
conv2d_3 (Conv2D)	(None, 15, 47, 64)	102464
conv2d_4 (Conv2D)	(None, 13, 45, 64)	36928
conv2d_5 (Conv2D)	(None, 11, 43, 64)	36928
flatten (Flatten)	(None, 30272)	0
dense (Dense)	(None, 1000)	30273000

```
_________________________________________________________________
dense_1 (Dense)              (None, 100)               100100
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 30,711,151
Trainable params: 30,711,151
Non-trainable params: 0
_________________________________________________________________
```

3.2 Training

3.2.1 Trainingsdaten

Die Trainingsdaten wurden mit Hilfe des Autos aufgenommen. Das Auto ist entlang eines Drahtes nach dem Faller Car-System durch das Miniatur Wunderland gefahren. Dabei wurden Video-Stream und zugehörige Werte des Hall Sensors aufgezeichnet. Die obere Hälfte des Bildausschnitts wird verworfen. Dies führt dazu, dass großteils die Straße auf den Bildern zu sehen ist [Abbildung 1]. Die Hoffnung ist, dass dadurch das Netzwerk nicht durch Häuser oder Deckenelemente im Miniatur Wunderland abgelenkt wird und somit allgemeiner eingesetzt werden kann.

Die Bilder werden als BGR-Bilder mit 3 Kanälen zusammen mit den Daten zum Lenkwinkel in einer H5-Datei gespeichert. Dabei ist zu beachten, dass die Bilddaten bereits als float32 Werte gespeichert werden. Sonst muss dieser rechenaufwändige Schritt während des Trainings durchgeführt werden. Eine Normierung der Bilddaten auf einen Wert zwischen 0 und 1 könnte zu einer Verbesserung der Ergebnisse vom Netzwerk führen. Allerdings würde es die Zeit, bis ein Bild bearbeitet wurde, deutlich erhöhen. Aufgrund dessen haben wir uns gegen eine Normierung entschieden.

3.2.2 Optimierung

Das Netzwerk wird dahingehend Trainiert die durchschnittlichen Quadratischen Abweichung (mse) zwischen dem Ergebnis des Netzes und den gemessenen Daten zum Lenkwinkel zu minimieren. Dies führt dazu, dass große Abweichungen besonders stark Gewichtet werden und kleinere nur Geringer.

Abbildung 1: Beispiele aus den Trainingsdaten mit eigeblendeten Metadaten.

3.2.3 ...

4 Ergebnisse/Fazit

- Genauigkeit der Berechneten Lenkwinkel noch nicht erprobt...

Literatur

[BTD+16] Mariusz Bojarski, Davide Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Larry Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. 04 2016.