Abgabe KI-Projekt: Verkehrszeichenerkennung

Team: Tim Kaiser, Klemens Heinrich

Betreuende Lehrkraft: Kai Metzger

Abbildung 1
https://www.bing.com/images/search?view=detailV2&ccid=lQOchN2C&id=987D51FA65F1B634B1C825C91919820A23E5AED3&thid=OIP.lQOchN2Cra5IWjGJjDMK-wHaEY&mediaurl=https%3a%2f%2fwww.autoscout24.de%2fcms-content-assets%2f2AmudNED1dNWGnwsSDrcG5-0ca1ff0b3ec51e010859fae

Projektziel

Ziel des Projekts war es, ein Programm zu entwickeln, das mittels Machine Learning in der Lage ist, deutsche Verkehrszeichen zu erkennen und zu klassifizieren. Die Umsetzung erfolgt mit dem Framework TensorFlow.Keras unter Verwendung öffentlich verfügbarer Datensätze aus dem Internet. Zusätzlich wird ein Inferenzprogramm erstellt, dass das trainierte Modell zur Laufzeit nutzen kann, um neue, unbekannte Bilder automatisiert zu analysieren.

Das Projekt soll demonstrieren, wie maschinelles Lernen zur automatisierten Erkennung visueller Informationen eingesetzt werden kann – eine Schlüsseltechnologie für moderne Fahrerassistenzsysteme und autonome Fahrzeuge.

Auswahl eines Datensatzes

Um den Fokus des Projekts auf die Entwicklung eines Machine-Learning-Algorithmus zu legen, wurde bewusst entschieden, auf einen bereits vorhandenen, öffentlich zugänglichen Datensatz zurückzugreifen. Dies ermöglichte eine erhebliche Zeitersparnis bei der Datenerhebung und -aufbereitung.
Nach eingehender Recherche wurden zwei geeignete Datensätze identifiziert:

Der erste Datensatz wurde im Rahmen einer Studienarbeit erstellt und ist über GitHub verfügbar (Link). Die Trainingsdaten sind bereits nach Klassen sortiert, wobei der Ground Truth direkt aus dem Ordnernamen abgeleitet werden kann. Die Bezeichnungen wurden hier den offiziellen VZ-Nummern des amtlichen Katalogs der Verkehrszeichen entnommen, was eine eindeutige Zuordnung ermöglicht.

Der zweite Datensatz stammt von der Plattform Kaggle (Link). Auch hier sind die Trainingsbilder nach Klassen in Ordnern strukturiert, allerdings erfolgt die Beschriftung nummerisch von 0 bis 42. Der zweite Datensatz ist deutlich umfangreicher und enthält pro Klasse zwischen 210 und 2220 Bildern.

Aufgrund der überschaubareren Datenmenge wurde zunächst der erste Datensatz für die Entwicklung und das Training verwendet, um die Trainingsdauer zu verkürzen und erste Ergebnisse schneller erzielen zu können.

Ablauf des Projekts

Nachdem ein passender Datensatz ausgewählt worden war, konnte mit der eigentlichen Programmierung begonnen werden. Zu Beginn wurde ein GitHub-Repository eingerichtet, um sicherzustellen, dass beide Teammitglieder jederzeit auf dem aktuellen Stand arbeiten und Änderungen effizient versionieren können.

Als Grundlage für den Machine-Learning-Algorithmus diente das von Herrn Metzger bereitgestellte „Training-symbols-dataset“. Dieses musste jedoch zunächst angepasst werden. Insbesondere die Erstellung des Ground Truth erforderte eine Überarbeitung: Während im Beispielprogramm der Ground Truth über eine separate *.txt*-Datei pro Bild definiert wurde, wurde in der von uns überarbeiteten Variante stattdessen die Python-Funktion os.path.normpath() verwendet. Diese ermöglicht es, den Namen des letzten Ordners im jeweiligen Dateipfad zu extrahieren und diesen automatisch dem Ground-Truth-Array zuzuordnen. Dadurch konnte der Annotierungsprozess deutlich vereinfacht und automatisiert werden. Um die Anzahl der vorhandenen Bilder automatisch anzupassen, wurde die glob.glob() Funktion genutzt. Diese Zählt bei uns alle Images in den Ordnern.

Um die Komplexität des Programms zu Beginn gering zu halten und potenzielle Fehlerquellen schneller identifizieren zu können, wurde zunächst mit einer reduzierten Anzahl von Klassen gearbeitet. Aus den insgesamt 43 verfügbaren Klassen wurden vier ausgewählt: Vorfahrt, Vorfahrt gewähren, Achtung und 50 km/h.

Zur Evaluation des trainierten Modells wird für jede dieser Klassen ein Bild verwendet, das weder im Trainings- noch im Testdatensatz enthalten ist. Diese Bilder dienen der manuellen Prüfung, wobei das Modell die Klasse vorhersagt und zusätzlich die ermittelte Wahrscheinlichkeit für jede Vorhersage ausgibt.

Während des Trainingsprozesses werden mehrere Modelle mit unterschiedlichen Hyperparametern trainiert. Dabei lassen sich unter anderem die Batchgröße, die Anzahl der Epochen sowie die Anzahl der Neuronen pro Schicht variieren. Jedes fertige Modell wird im Ordner chpt gespeichert. Ergänzend dazu wird eine Textdatei erstellt, in der die verwendeten Hyperparameter, die Klassifikationsergebnisse der Testbilder sowie die erzielte Accuracy und der Loss dokumentiert werden.

Zusätzlich werden Verlaufsdiagramme generiert, die die Entwicklung von Accuracy und Loss während des Trainings grafisch darstellen. Dies ermöglicht einen übersichtlichen Vergleich der einzelnen Trainingsdurchläufe und unterstützt bei der Auswahl der besten Modellkonfiguration.

Als nächster Verbesserungsschritt wurde die Anzahl der erkannten Klassen erhöht und ein Inferenzprogramm zur Live-Erkennung implementiert. Dieses Programm nutzt das Bild einer über USB angeschlossenen Kamera, die an einem Raspberry Pi betrieben wird. Das aktuelle Kamerabild wird in Echtzeit ausgewertet und die jeweils wahrscheinlichste Verkehrszeichenklasse ausgegeben.

Während der ersten Tests mit den Inferenzprogramm zeigte sich, dass das Modell häufig nur zwei Klassen zuverlässig erkennt. Um die Genauigkeit zu verbessern, wurde das neuronale Netzwerk daraufhin erweitert: Es wurden drei zusätzliche Convolutional Layers sowie Average Pooling Layers integriert. Die Anzahl der Filter und Faltungen orientierte sich dabei an bewährten Konfigurationen aus einschlägigen Quellen und Fachartikeln. Durch diese strukturelle Anpassung sollte das Modell in der Lage sein, komplexere Muster besser zu erfassen und zwischen mehreren Klassen zuverlässiger zu unterscheiden.

Im vorletzten Verbesserungsschritt wurde die Anzahl der Bilder pro Klasse erhöht, indem ein alternativer Datensatz verwendet wurde. Nun stehen pro Klasse etwa 2000 Bilder zur Verfügung. Gleichzeitig wurde der Code so angepasst, dass nicht mehr Graustufenbilder, sondern Farbbilder verwendet werden. Dies führte zu einer signifikanten Erhöhung der Trainingszeit, die von anfänglich etwa 40 Minuten auf mehrere Stunden pro Modell anstieg. Diese längeren Trainingszeiten erschweren die Arbeit zuhause, da ein leistungsfähiger Rechner erforderlich ist. Um solche Modelle im professionellen Rahmen zu trainieren, ist der Einsatz einer leistungsstarken GPU unverzichtbar.

Als letzte Verbesserungsmaßnahme wurde die Architektur des Netzwerks an die aktuelle LeNet-Struktur angepasst. Dazu wurde die bisherige Kombination aus einer Conv2D-Schicht mit integrierter Aktivierungsfunktion (z. B. *tanh* oder *ReLU*) und anschließender MaxPooling2D-Schicht in vier separate Layer aufgeteilt. Die Aktivierungsfunktion wurde dabei aus der Conv2D-Schicht entfernt, stattdessen wurde eine BatchNormalization-Schicht dazwischengeschaltet, um das Training zu stabilisieren und zu beschleunigen. Wichtig ist, dass die Normalisierung **vor** der ReLU-Aktivierungsfunktion erfolgt.

Durch den Einsatz der ReLU-Funktion wird das Modell nichtlinear, wodurch es in der Lage ist, komplexere Muster und Strukturen in den Eingabedaten besser zu erkennen.

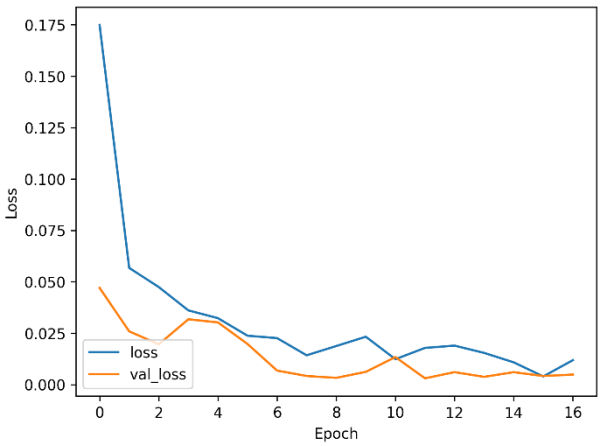
```
model.add(layers.Conv2D(6, (5,5), activation=None, padding='same', input_shape=img_shape))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2,2)))
```

Zusätzlich wurde noch Early-Stoping implementiert, um overfitting zu umgehen. Um die einzelnen Trainierten Modelle noch Besser miteinander vergleichen zu können wurde ebenfalls eine Confusion Matrix eingefügt. Diese wird nach dem Training ebenfalls in dem chpt-Ordner abgespeichert.

Auswahl des Besten Modells

Nachdem das Modell Fertig optimiert war, mussten nun die Idealen Hyperparameter gefunden werden. Bei allen Modellen zeigt sich die größte Schwachstelle beim Erkennen des Stoppschildes. Dies liegt daran, dass es für diese klasse am wenigsten Trainingsdaten gibt.

Training 62:



Bildauflösung:64x64

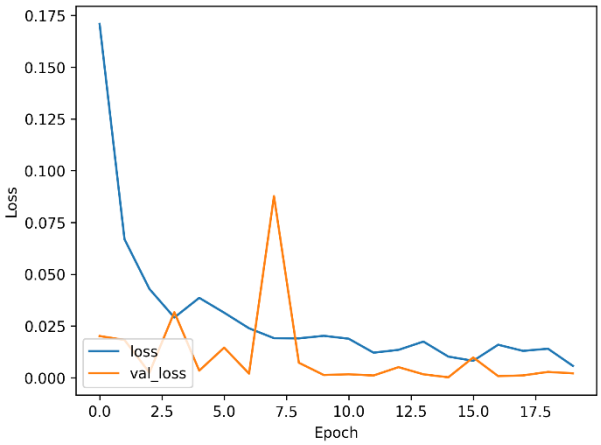
Anzahl Neuronen: 256

Batchsize:8

Es zeigt sich eine Erhöhung des val_loss nach der 8. Epoche. Im Anschluss wird der Minimalwert nichtmehr erreicht. -> Stoppen bei Epoche 8.

Der Test mit sechs Beispielbilder ergab: 5 von 6 Bilder richtig erkannt jedoch Stoppschild zu 8,75%

Training 63:



Bildauflösung: 64x64

Anzahl Neuronen: 256

Batchsize:8

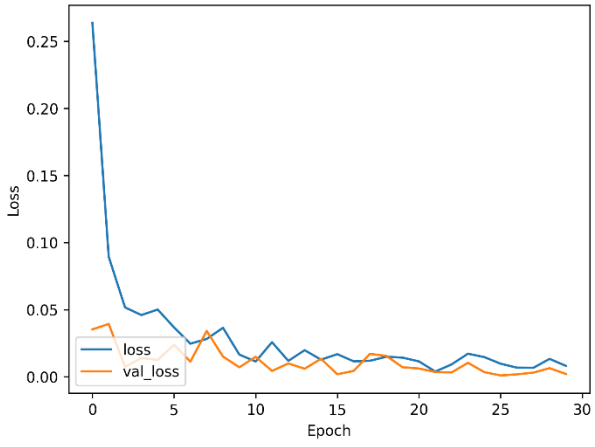
Während der loss konstant sinkt schwankt der val_loss sehr stark. Niedrigster Punkt bei Epoche 10.

Es wurden ebenfalls 5 von 6 Bildern richtig erkannt. Das Stoppschild wurde zu 0% erkannt.

Training 64 bis 67:

Die Trainings wurden mit immer kleiner werdender Neuronen zahl durchgeführt. Die anderen Hyperparameter waren identisch zu den Trainings 62 und 63. Hierbei konnte beobachtet werden das je kleiner die Anzahl der Neuronen ist, desto geringer ist die Wahrscheinlichkeit das die Testbilder richtig erkannt wurden und desto früher beginnt ein Overfitting des Modells.

Training 68:



Bildauflösung: 32x32

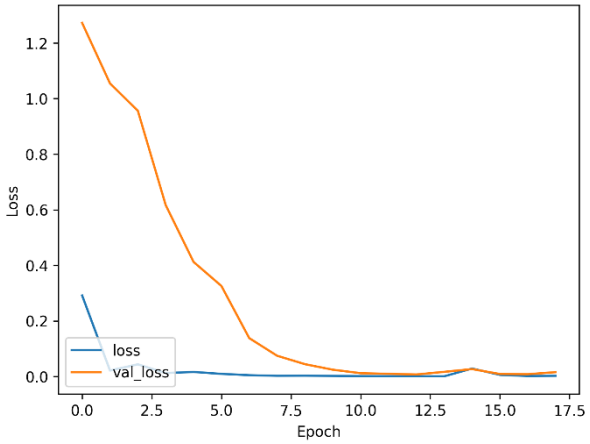
Anzahl Neuronen: 256

Batchsize:8

Es zeigt sich, dass das Modell deutlich höhere Schwankungen hat bei einer geringeren Auflösung. Wir würden das Training nach der 11 Epoche stoppen.

Bei dem Test mit sechs Bildern wurden nur 3 richtig erkannt. Daher werden höhere Neuronen zahl für künftige Modelle bevorzugt. Die Trainings 69 bis 73 zeigten die gleichen Ergebnisse

Training 74



Bildauflösung: 64x64

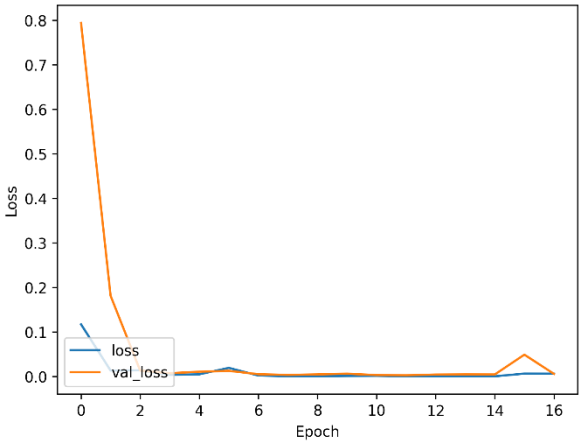
Anzahl Neuronen: 256

Batchsize: 256

Durch die höhere Batchsize zeigt sich eine geglättete val_loss kurve.

Bei dem Test mit 6 Bildern wurden jedoch nur 3 korrekt Erkannt.

Training 75



Bildauflösung: 64x64

Anzahl Neuronen: 256

Batchsize: 64

Abbruch des Trainings nach der 13. Epoche.

Mit einer etwas niedrigeren Batchsize bekommen wir ähnliche Ergebnisse wie bei Training 62. Bei dem Test mit 6 Bildern wurden 5 richtig erkannt. Dieses Mal wurde jedoch nicht Stopp, sondern Achtung nicht richtig erkannt.

Auswahl des Modells

Nachdem mit verschiedenen Hyperparametern Modelle trainiert wurden, kamen die Modelle 75 und 62 in die nähere Auswahl. Beide wurden mit dem Inferenzprogramm geöffnet und ausgiebig getestet.

Hierbei zeigte sich, dass das Modell 75 am besten die Livebilder erkannt hat. Daher wurde dieses Modell final gewählt.