Dokumentation zu WeatherPredicter

Autor: Shazil Khan

GitHub Link: https://github.com/Bro-tec/WeatherPredicter.git

Inhaltsverzeichnis

Einleitung

Hauptteil

* *Datensammlung*
* *Daten Extraktion*
* *Maschinelles Lernen*
* *Visualisierung*

Schluss

Einleitung

In diesem Projekt ging es darum, Wetterdaten zu erfassen und mithilfe eines Long Short-Term Memory (LSTM) Modells ausgewählte Parameter vorherzusagen.

Hierbei wurden Daten vom Deutschen Wetterdienst gesammelt und in das LSTM-Modell eingespeist.

Ziel war es, Prognosen für verschiedene Werte wie Temperatur, Windgeschwindigkeit und Wetterverhältnisse für die zokünftige Tage zu generieren.

Mehrere Modelle wurden trainiert, die in zwei Varianten unterteilt wurden und jeweils unterschiedliche Dateneingaben und Ausgaben verwenden.

Diese Modellvarianten werden als "Hourly" und "Daily" bezeichnet.

Hourly ist eine weniger rechenintensive und daher ressourcenschonendere Variante, die Vorhersagen basierend auf einer begrenzten Anzahl von Eingabewerten trifft.

Sie erzeugt sechs Ausgaben, darunter Temperatur, Windrichtung, Windgeschwindigkeit, Sichtweite sowie Wetterverhältnisse 1 und 2.

Daily hingegen ist eine rechenintensivere und aufwendigere Variante, die eine Vielzahl von Eingabewerten verwendet, was den Trainingsprozess verlangsamt.

Sie bietet wiederum sechs Ausgabewerte, darunter minimale und maximale Temperatur, minimale und maximale Windgeschwindigkeit sowie Wetterverhältnisse 1 und 2.

Die Motivation hinter der Verwendung beider Verfahren bestand darin, dass Hourly weniger Ressourcen benötigt, um trainierte Modelle zu generieren.

Diese sind jedoch weniger präzise in ihren Vorhersagen.

Daily hingegen erfordert mehr Ressourcen, was den Trainingsprozess verlangsamt, ermöglicht jedoch genauere Vorhersagen.

Hauptteil

Die Idee bestand darin, mithilfe von maschinellem Lernen eine Wettervorhersage zu erstellen, wobei Python als Hauptsprache verwendet wurde.

Zunächst wurden Wetterdaten gesucht und in Datensätze (Features) umgewandelt.

Ebenso wurden die Daten, zu welchen Prognosen gestellt werden sollten, in Datensätze (Labels) umgewandelt.

Diese wurden in zwei verfahren weitergegeben, die sich Hourly und Daily nennen.

Daraufhin sollten eigene LSTM Modelle erstellt werden, wobei mehrere Trainingsdateien erstellt wurden.

Anschließend wurden die Datensätze verwendet, um die Modelle zu trainieren.

Während des Trainings wurde der Loss-Wert bewertet und in einem Diagramm dargestellt.

Des Weiteren wurde die Genauigkeit (Accuracy) der Vorhersagen bestimmt.

Bei Verwendung von Labels wurden diese auch in Form einer Konfusionsmatrix dargestellt.

Zum Schluss wurden die Vorhersagen zusammen mit en Labels Bildlich dargestellt um schauen zu können wie genau die Modelle sind.

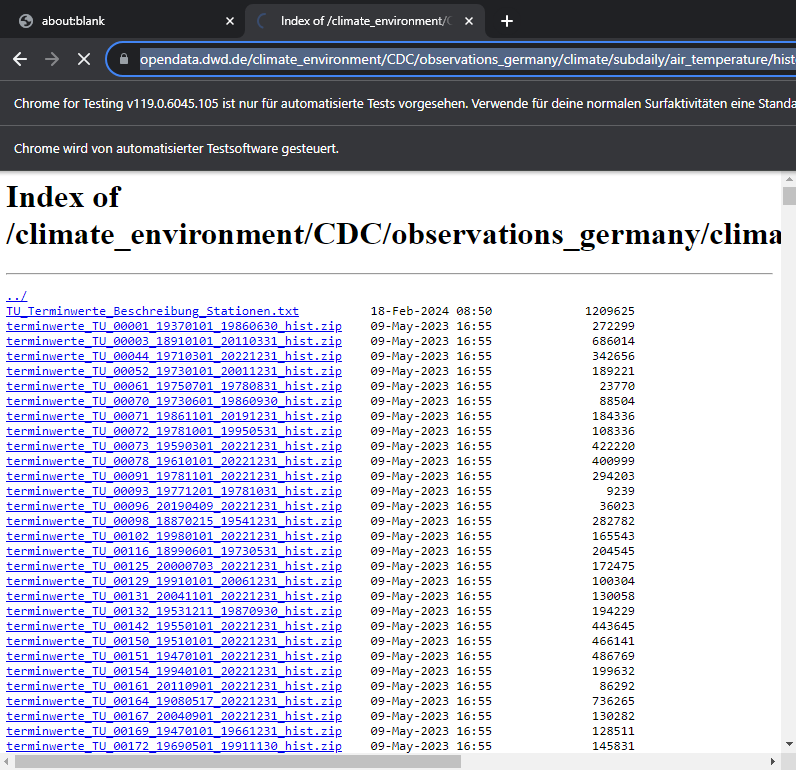
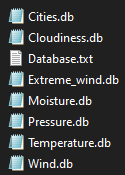
Zukunftsvorhersagen ohne Labels sind auch möglich.

Datensammlung

Beim Sammeln von Daten traten zwar wenig, jedoch starke Verzögerungen auf.

Es wurden verschiedene Methoden zur Datengewinnung ausprobiert.

Zunächst wurden Daten von der Website des Deutschen Wetterdienstes (DWD) mithilfe eines Webcrawlers in Node.js gesammelt.

Die gesammelten Daten mussten in Datenbanken zusammengeführt werden, da es zu umständlich war, sie individuell zu extrahieren.

Während der Arbeit mit den Daten wurde festgestellt, dass viele Daten fehlten und es zu vielen Nullwerten kam, die aus verschiedenen Gründen nicht ausgefüllt wurden.

Auf der Suche nach qualitativ und quantitativ hochwertigeren Daten wurde die Programmierschnittstelle (API) von OpenWeather ausprobiert.

Allerdings stellte sich heraus, dass diese kostenpflichtig war und daher für das Projekt nicht geeignet war.

Nach zahlreichen weiteren Auswertungen wurde letztendlich Bright Sky verwendet, eine API vom DWD.

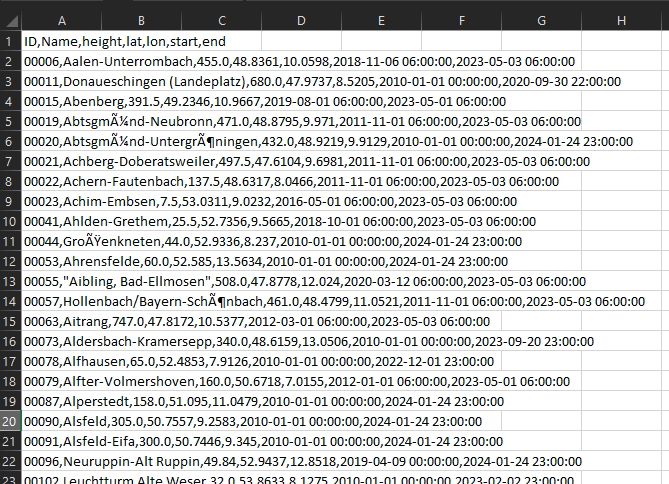
Obwohl sie viele Nullwerte lieferte, gestaltete sich der Zugriff auf die Datensätze weniger umständlich.

Diese mussten nicht zwischengespeichert werden, und es standen viele Daten zur direkten Verfügung.

Daten Extraktion

Um die daten zu generieren wird zu erst die ID der Stadt und das Datum benötigt.

Die IDs der Städte wurde durch den Start der Datei get\_DWD\_data in dem Ordner CollectData gesammelt und in eine Exceltabelle mit den Daten zur Stadt gespeichert. In dieser Datei befinden sich auch das Anfangsdatum der gesammelten Daten zu jeder Stadt, womit man das Startdatum setzen konnte.



Die erhaltenen Daten wurden in Datensätze umgewandelt und mit einem fertigen Keras-Programm trainiert.

Dieser Vorgang dauerte für eine Epoche mit 200 Daten rundum 30 Minuten oder gar länger.

Aus diesem Grund wurde Asyncio verwendet, um die Datensammlung zu parallelisieren.

Durch diesen Schritt konnte die Zeit für das Sammeln aller 1561 Daten zu einem bestimmten Datum und deren Umwandlung in verwendbare Datensätze auf eine bis fünf Minuten reduziert werden, abhängig von der Qualität der Internetverbindung zum Zeitpunkt des Trainings.

Die Datensammlung wurde mit zwei verschiedenen Ansätzen realisiert: dem Hourly-Verfahren, das die Werte der nächsten Stunde und wenn möglich auch die des nächsten Tages vorhersagen soll, und dem Daily-Verfahren, das die Werte der nächsten sieben Tage vorhersagen soll.



Beim Hourly-Verfahren handelt es sich um eine weniger rechenintensive Modellsammlung, die auf weniger Merkmalen basiert und schneller trainiert werden kann. Die Features für jede Stadt belaufen sich auf 17 werte, die mit den Werten, der vier nächsten Städte zusammengesetzt werden. Da der tag 24 h hat werden 24 Features zu jeder erfolgreich ausgegebenen Stadt ausgegeben. Neben diesen 85 Daten wird die stunde zu jedem Feature hinzugefügt, worauf sich die Features auf 86 werten beläuft. Die Labels hingegen setzen sich zusammen aus den sechs werten: Temperatur, Windrichtung, Windgeschwindigkeit, Sichtweite sowie Wetterverhältnisse 1 und 2.

Beim Daily-Verfahren handelt es sich hingegen um ein rechenintensiveres Modellverfahren, da es auf mehr Merkmalen basiert, was jedoch zu längeren Trainingszeiten führt. Die Features für jede Stadt belaufen sich wie bei hourly auf 17 werte, die diesmal mit den werten von insgesamt acht Städten in vier Himmelsrichtungen in sowohl kürzerer als auch fernerer distanz zusammengesetzt werden. Da der tag 24 h hat werden alle 24 Features zu jeder erfolgreich ausgegebenen Stadt ausgegeben zusammengesetzt. Somit beläuft sich die Anzahl dieser Featrure pro Stadt auf 7752 Werten. Die Labels hingegen setzen sich zusammen aus den sechs werten: minimale Temperatur, maximale Temperatur, minimale Windgeschwindigkeit, maximale Windgeschwindigkeit, sowie Wetterverhältnisse 1 und 2.

Maschinelles Lernen

Da nun zu viele Daten auf einmal verwendet wurden, sollte das Training der Daten auf einer Nvidia-Grafikkarte durchgeführt werden, wobei dieser Prozess mit CUDA realisiert werden sollte.

Da der Trainings-Algorithmus in Keras realisiert wurde, wurde über zwei Wochen lang versucht, CUDA mit TensorFlow zum Laufen zu bringen.

Dabei wurden mehrere Conda Environments unbrauchbar gemacht, und sowohl Treiber als auch Anaconda mussten mehrmals neu installiert werden.

Aufgrund des Zeitverlusts wurde auf PyTorch als Alternative zurückgegriffen.



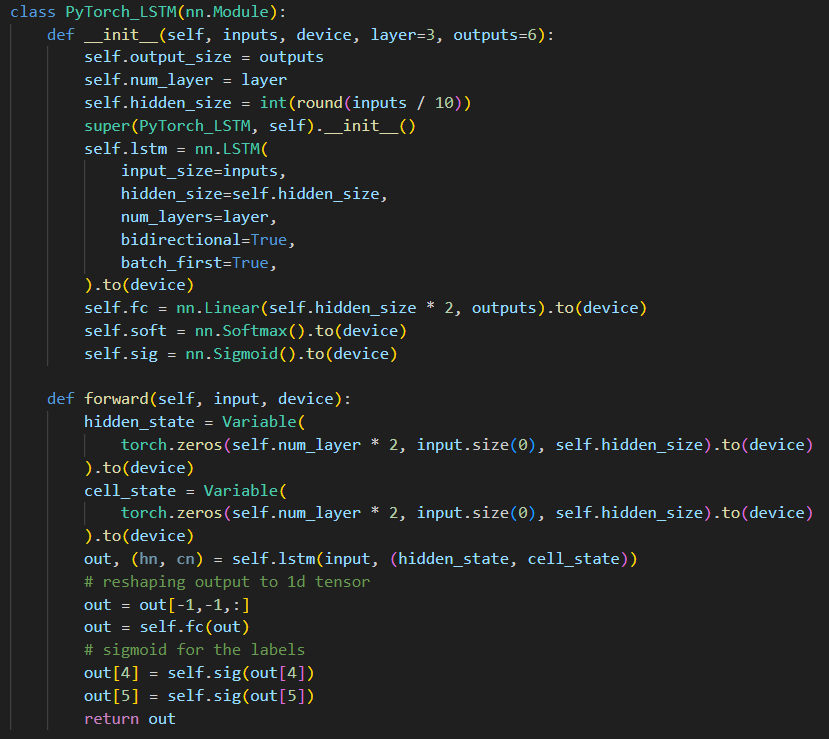
Wie bei Keras wurde ein LSTM verwendet, das die Datensätze in Form von Tensoren erhält und auf der Grafikkarte trainiert.

Anfangs wurde ein Kullback-Leibler-Divergenz-Verlust verwendet, jedoch wurde nach mehreren Versuchen letztendlich auf einen Cross-Entropy-Verlust gewechselt.

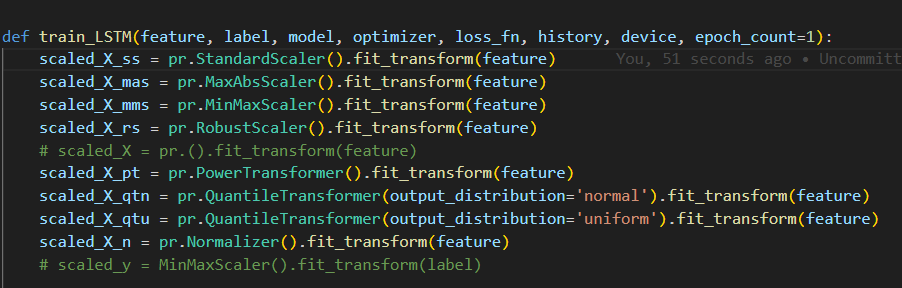
Die Accuracy wird dabei prozentual als Bewertung der Summe der richtig vorhergesagten Werte berechnet.

Für die Labels wurde beim Trainieren lange der Fehler gemacht Softmax zu verwenden, was einen wert maximiert und alle anderen minimiert. Der Fehler liegt darin dass unter den sechs Klassen, nicht die mit der höchsten Wahrscheinlichkeit gewählt wird, sondern dass aus den werten skalierte lösungswerte ausgegeben werden, wofür der Softmax nicht gedacht ist.

Die Werte Innerhalb der letzten beiden Tensoren des Datensatzes, sind Labels aber dies sollen positive Zahlen ausgeben, was mit Sigmoid Realisiert wurde.



Die Datensätze mussten außerdem skaliert werden. Während verschiedene Versuche unternommen wurden, die Eingabedaten zu skalieren, wurde entschieden diese unverändert zu lassen, da es sich um verschiedene Wertegrößen handelte.



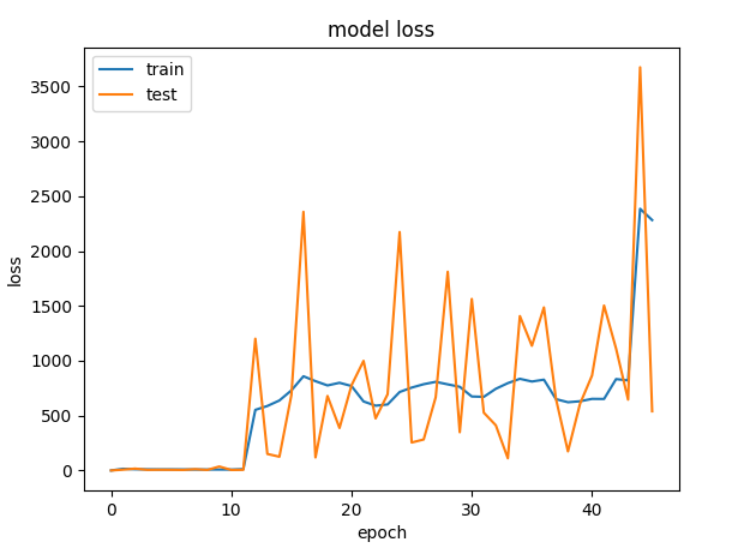
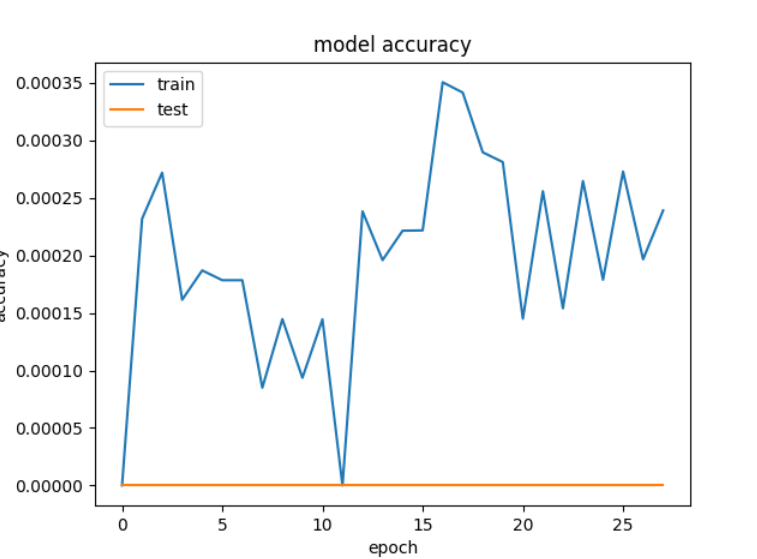
Die Labels hingegen wurden mit einer eigenen Funktion skaliert.

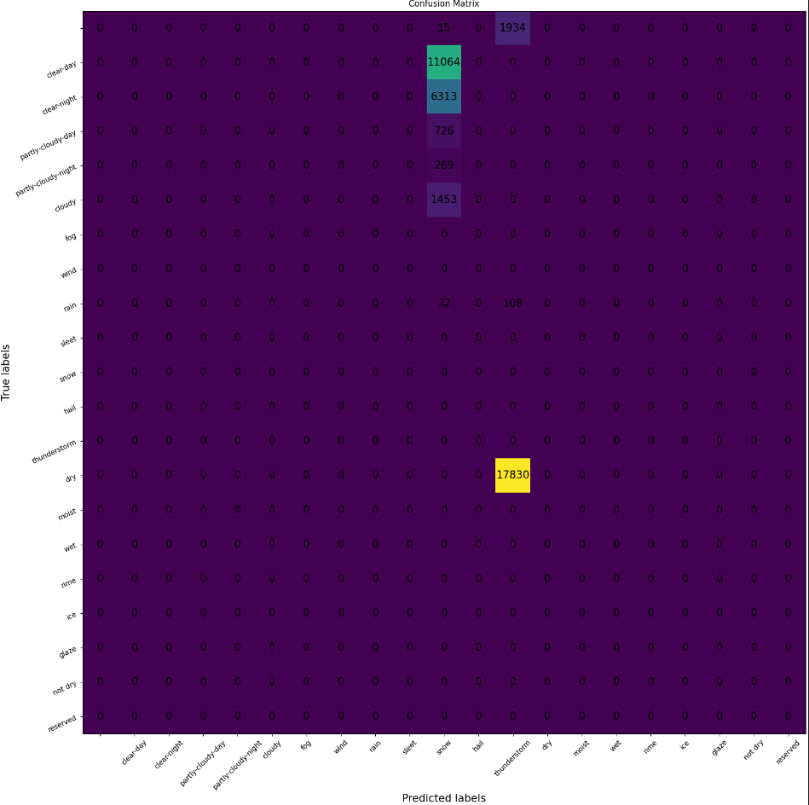
Die vorhergesagten Ausgaben wurden ebenfalls durch eine eigene Funktion in verwendbare Werte umgewandelt.

Visualisierung

Um die Evaluation zu visualisieren, wurden die Accuracy- und die Loss-Werte in Diagrammen dargestellt.

Die klassifizierbaren Werte in der Ausgabe wurden als Multiclass-Classification dargestellt.

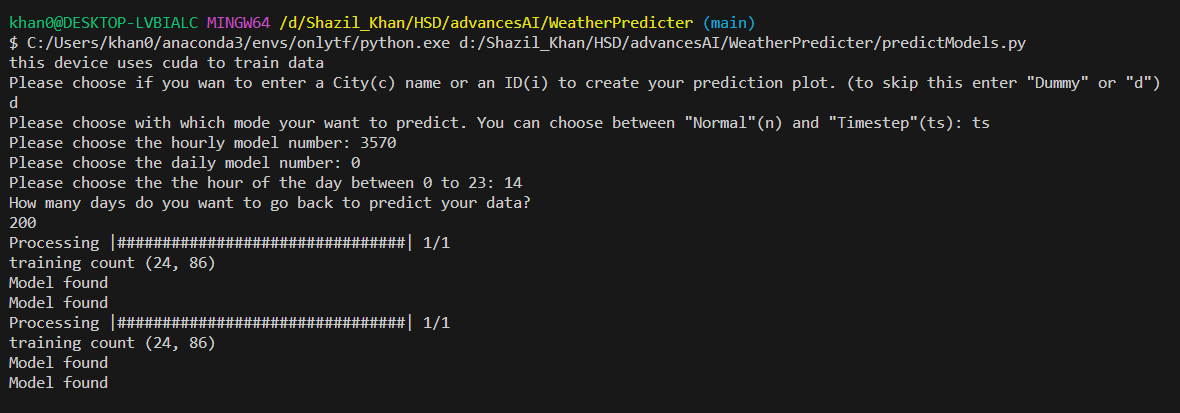
 



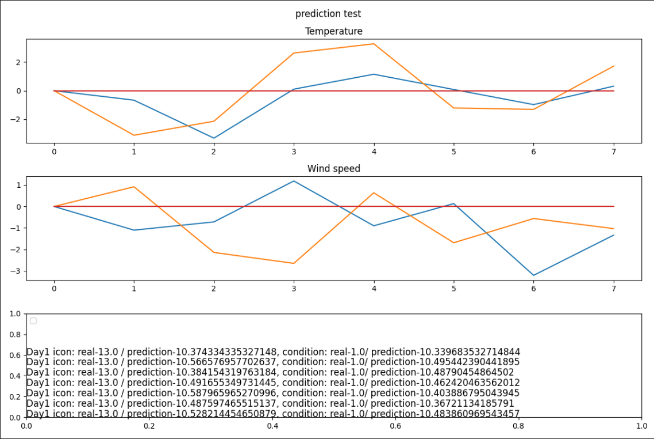
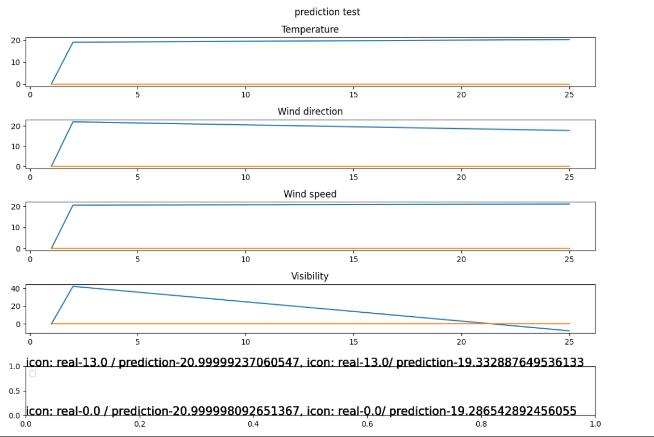
*Dies sind schlechte Ausgaben und werden sobald das training abgeschlossen ist verbessert*

Anschließend wurden die Ausgaben separat mit einem ausgewählten Modell in Diagrammen dargestellt.

Um den Anforderungen gerecht zu werden, dass ein Benutzer diese Modelle trainieren kann, wurden die Dateien so bearbeitet, dass man durch Interaktion mit der Konsole die Modelle trainieren und eigene Vorhersagen treffen kann.



Alle Diagramme werden im Ordner "Plots" gespeichert und können dort eingesehen werden.

*Hourly Daily*

*Dies sind schlechte Ausgaben und werden sobald das training abgeschlossen ist verbessert*

Falls genügend Speicherplatz auf dem Endgerät vorhanden ist, können alle Modelle im Timeseries-Modus nach jeder Epoche separat nummeriert gespeichert werden, um dem Verlust der Genauigkeit beim sogenannten Overfitting entgegenzuwirken.

Schluss

Das Projekt wurde erfolgreich abgeschlossen und erfüllt alle Voraussetzungen, um Modelle zu trainieren, die gute Vorhersagen treffen.

Die Dateien können auch von Personen ohne grundlegendes Wissen im Bereich des Maschinellen Lernens ausgeführt werden, wodurch auch Laien in der Lage sind, diese Modelle zu trainieren.

Allerdings wurden die Modelle nicht vollständig trainiert, sodass nur unfertige Modelle evaluiert und getestet werden konnten.

Für zukünftige Projekte empfiehlt es sich, APIs zu verwenden, da sie viele Daten ohne großen Aufwand liefern können.

Des Weiteren sollten die fertigen Modelle für den Einsatz bereitgestellt werden. Wenn große Datenmengen trainiert werden müssen, ist es ratsam, von Anfang an auf PyTorch zu setzen, da damit erfolgreich und ohne viele Probleme auf CUDA-fähigen Geräten trainiert werden kann.