Convolutional LSTM zur Bewegungsvorhersage von Verkehrsteilnehmern

Anwendungen der Künstlichen Intelligenz
Sommersemester 2021

Armin Straller

13. Juli 2021

Zusammenfassung

Um vorausschauendes Fahren für Autonome Fahrzeuge zu ermöglichen, ist es notwendig die Bewegungen anderer Verkehrsteilnehmer vorherzusagen. Da das Verhalten von Fußgängern, Radfahreren und Autofahrern oft auf Interaktion basiert oder von Ortspezifischen Mustern abhängt, bietet es sich an ein Netzwerk mit historischen Informationen über das Verhalten der Verkehrsteilnehmer aufzubauen. Diese Arbeit beschäftigt sich mit einem solchen Netzwerk zur Bewegungsvorhersage von Verkehrsteilnehmern. Als Datengrundlage wurde der im Frühjahr 2021 veröffentlichte Waymo Open Motion Datensatz verwendet. Das entwickelte Netzwerk verwendete bildbasierte Eingangsdaten welche eine gerasterte Darstellung der unterschiedlichen Szenarien verwendet. Ein LSTM bietet dann die Möglichkeit Daten eines vergangen Zeitraums in Kontext zu bringen und ein eine bildliche Repräsentation eines zukünftigen Zustands auszugeben. Das aus der Arbeit resultierende Netzwerk ermöglicht grundlegende vorhersagen der Bewegungsrichtung ist aber zum aktuellen Entwicklungsstand nicht für den Einsatz in einem Autonomen Fahrzeug geeignet.

1 Problemstellung

Die Problemstellung für die entwickelte Künstliche Intelligenz ist aus der Waymo Motion Prediction Challenge abgeleitet. Hierbei soll auf Basis der Verkehrsdaten aus einer vergangenen Sekunde eine prädiktion für die folgenden acht Sekunden erfolgen. Der dafür bereitgestellte Datensatz ist das Waymo Open Motion Dataset [1]. Dieses umfasst 574 Stunden und 1750 km an Verkehrsinformation und bietet somit genug Informationen für das Training eines Neuronalen Netzwerks.

1.1 Datengrundlage

Im Datensatz von Waymo sind verschidene Informationen über die aktuelle Verkehrssituation enthalten. Diese können im Detail dem scenario.proto aus dem Anhang entnommen werden. Teil des Scenarios ist auch eine Karte deren Details in der map.proto Datei definiert sind. [2]

Auf die für diese Arbeit wichtigen Inhalte der Szenarien wird Kapitel über die Vorverarbeitung der Daten eingegangen.

2 Related Work und Entstehung der Projektidee

Es haben sich bereits andere Arbeiten mit der Vorhersage von Bewegungen im Straßenverkehr beschäftigt. Aufgrund der aktualität des Waymo Datensatzes sind zu diesem allerdings noch keine Veröffentlichungen bekannt. Ein Datensatz mit ähnlichem Inhalt wurde für die Prediktion von anderen Fahrzeugen in der Arbeit von Jeong, Yonghwan verwendet. Hierbei kam ein LSTM-RNN Netzwerk zum Einsatz. [3]

Auch die Arbeit von Wu, Jingyuan beschäftigt sich mit der Bewegungsvorhersage. In diesem Fall geht es um Fußgänger und eine Karten- beziehungsweise Modellbasierte Methodik zur Vorhersage. Diese verwendet ein Raster welches die Aufenthaltswahrscheinlichkeit der Fußgänger darstellt. [4]

2.1 Convolutional LSTM basierte Bewegungsvorhersage

Auf Basis der aufgeführten Arbeiten und der zur verfügbaren Daten ist die grundlegende Idee für diese Arbeit entstanden. Die Verkehrsinformationen werden in ein Raster eingetragen und in Bildform als Eingangsdaten für das Convolutional-LSTM verwendet. Hierdurch können die Karten- und Positionsdaten der Verkehrsteilnehmer in Relation gebracht werden. Die Historie der Bewegung wird dann durch die Eingangsbreite des LSTMs dargestellt. Das Keras Beispiel “Next-Frame Video Prediction with Convolutional LSTMs“ [5] dient als Grundlage für die Umsetzung. Da das Raster nur mit begrenzter Auflösung arbeitet kann es theoretisch vorkommen, dass das Autonome Fahrzeug im Szenario das Raster verlässt. Um dies zu umgehen wird ein Ansatz eine Transformation des Szenarios in die Ego-Perspektive des autonomen Fahrzeugs beinhalten. Im Folgenden wird daher von einer statischen oder dynamischen Darstellung gesprochen, wobei die dynamische Darstellung die Transformation in die Ego-Perspektive beinhaltet. Um eine Prediktion von bis zu 8 Sekunden zu ermöglichen, soll das Modell erneute Prediktionen auf basis der bereits predizierten Bilder erstellen. Dies wird wiederholt bis der benötigte Zeithorizont abgedeckt ist.

3 Vorverarbeitung der Daten

Um die Informationen aus dem Waymo Open Motion Dataset in die Darstellungsform des Rasters zu bringen sind einige Schritte notwendig. Diese werden im Folgenden dargestellt.

3.1 Einlesen des Protobuf

Zunächst muss das Szenario aus dem scenario.proto Format eingelesen werden. Hierfür werden Auszüge aus dem Waymo Open Dataset Motion Tutorial [6] verwendet. Dabei werden allerdings nur die Einträge geladen die für die Rasterisierung der Objekte notwendig sind. Es werden immer alle Zeitschritte geladen und anschließend konvertiert. Dies beinhaltetd die vergangenen, den aktuellen und die zukünftigen Elemente des Protobufs.

3.2 Konvertierung der Karte

Die Karteninformation ist Teil des scenario.proto und wird in Form des map.proto in jedem Szenario bereitgestellt. Ein Auszug aus der map.proto Definition der in Listing 1 dargestellt wird zeigt diese Liste und die Definition der Punkte. Hierbei ist zu beachten,

dass die Punkte in Metern zu einem beliebigen Ursprung gegeben sind. Die “polyline“ definiert dann ein Feature des Straßennetzwerks (im Beispiel für “LaneCenter“). Diese ist eine Liste von “MapPoints“ welche Segemente zwischen aufeinanderfolgenden Punkten definieren. Da die Auflösung der Punkt groß genug ist werden diese Einfach in das Raster übertragen. Sobald ein “MapPoint“ innerhalb eines Rasterelements liegt wird dieses entsprechend des Linientypen eingefärbt.

```
message LaneCenter {
    // ...
    repeated MapPoint polyline = 2;
}
message MapPoint {
    optional double x = 1;
    optional double y = 2;
    optional double z = 3;
}

```

Listing 1: Auszug aus map.proto

Im Abbildung 1 wird die Karte aus dem Szenario 38 des Validierungsdatensatzes dargestellt. Die statische Repräsentation das Scenarios erfordert eine einmalige Konvertierung der Karte. Diese bleibt dann für den gesamten Zeitraum des Szenarios konstant.

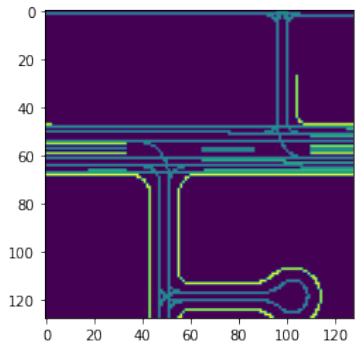
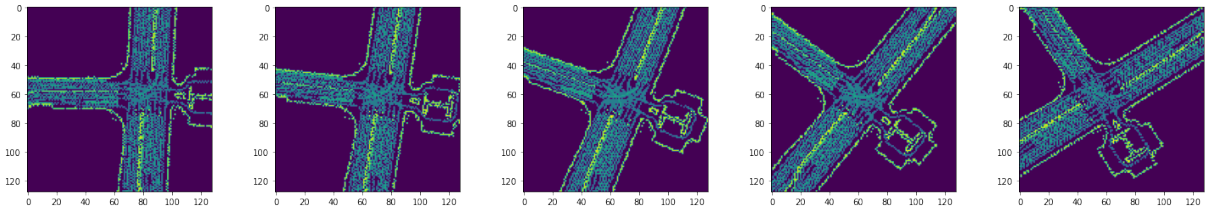


Abbildung 1: Beispiel einer in das Raster übertragenen Karte

Dynamische Darstellungsform der Karte Für den Fall der dynamischen Darstellung der Karte muss diese bei jeder Bewegung des autonomen Fahrzeugs in dessen Ego-Perspektive transformiert werden. Dadurch ergibt sich zum Beispiel bei einer Kurvenfahrt eine Rotation und Translation der Karte. Dies wird in Abblidung 2 veranschaulicht. Hier wird die Karte für die Zeitschritte -1 s also dem Start der Aufzeichnung, 0 s entsprechend dem gegenwärtigen Zeitschritt, sowie den zukünftigen Zeitschritten +1 s und +2 s dargestellt. Datengrundlage ist das zweite Szenario des Validierungsdatensatzes.



(a) -1 s (b) 0 s (c) +1 s (d) +2 s (e) +3 s

Abbildung 2: Beispiel für eine dynamische gerasterte Karte

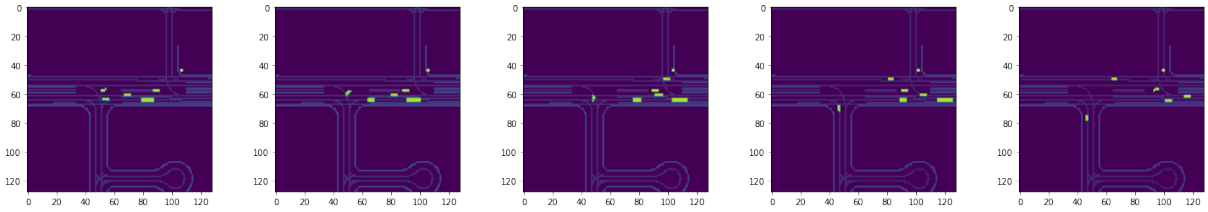
3.3 Konvertierung der Objekte

Der Protobuf enthält Informationen über die Größe sowie globale Position und Orienierung der Objekte. Dadurch können die vier 2D-Eckkoordinaten der Objekte berechnet werden. Um diese in das 2D-Raster zu übertragen wird die Grundfläche der Box entlang der Diagonale in zwei Dreiecke aufgeteilt und anschließend in das Raster übertragen. Wenn Objekte kleiner als ein Rasterelement sind, werden sie auf ein ganzes Rasterelement übertragen. So kann sichergestellt werden, dass alle Objekte in der Darstellung repräsentiert werden. Die Definition der Objekte im Protobuf wird in Listing 2 aufgeführt. Das “valid“ Flag gibt an ob die Daten verwertbar sind. Ist dieses für einen Zeitschritt nicht wahr, wird das Objekt für den zugehörigen Schritt nicht in das Raster übertragen.

```
1         message ObjectState {
2             optional double center_x = 2;
3             optional double center_y = 3;
4             optional double center_z = 4;
5
6             optional float length = 5;
7             optional float width = 6;
8             optional float height = 7;
9
10            optional float heading = 8;
11
12            optional bool valid = 11;
13        }
14
```

Listing 2: Auszug aus scenario.proto

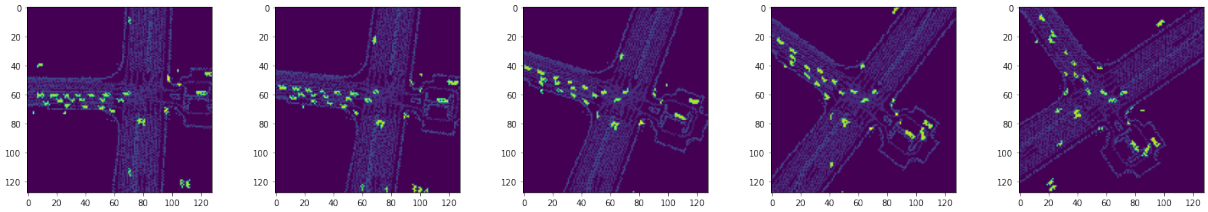
Im Abbildung 3 werden Ausschnitte aus dem gesamten Szenario 38 des Validierungsdatensatzes dargestellt. Die Objekte werden entsprechend ihrer globalen Position dem Raster hinzugefügt. Da bei der Erstellung der Bilder für die Einfärbung eine Heatmap verwendet wir, werden stets die höchsten Werte in der hellsten Farbe dargestellt. Deshalb ist die Karte in Abbildung 1 auch deutlich heller eingezeichnet als in Abbildung 3.



(a) -1 s (b) 0 s (c) +1 s (d) +2 s (e) +3 s

Abbildung 3: Beispiel für Objekte auf statischer Karte

Dynamische Darstellungsform für Objekte Bei der dynamischen Darstellungsform, müssen alle Objekte auf Basis der aktuellen Position des autonomen Fahrzeugs in dessen lokales Koordinatensystem transformiert werden. Die Information darüber welches Fahrzeug das autonom agierende ist, kann ebenfalls dem Protobuf entnommen werden. Die dynamische Darstellung der Objekte wird in Abbildung 4 anhand des zweiten Scenarios aus dem Validierungsdatensatz gezeigt.



(a) -1 s (b) 0 s (c) +1 s (d) +2 s (e) +3 s

Abbildung 4: Beispiel für Objekte in dynamischer Darstellungsform

3.4 Speicherung der Trainings- und Validierungsdaten

Die konvertierten Daten werden für das Training des ConvLSTM Netzwerks in einer Ordnerstruktur abgelegt. Diese beinhaltet einen Ordner für jedes Szenario. Ordner bei denen die statische Darstellungsform verwendet wurde tragen den prefix “static_“.

3.5 Probleme bei der Datengenerierung

Fehler durch Rasterisierung Die Rasterisierung dazu führt, dass Objekte nur mit der Rasterauflösung von 0.5 m positioniert sein können. Deshalb kann es vorkommen, dass Objekte von einem Zeitschritt zum nächsten um die Rasterauflösung springen. Dies führt bei einer nur geringen Bewegung, die allerdings die grenze zum nächsten Feld des Rasters überschreitet, zu großen Fehlern.

Konvertierungszeit Durch die große Datenmenge der Protobufs und die ineffiziente Programmierung des Konvertierungsalgorithmus, benötigt dieser zwischen 150 s und 200 s für die Konvertierung eines Szenarios.

4 ConvLSTM für Bewegungsvorhersage

Das Convolutional LSTM bietet die Möglichkeit die Informationen auf Bildebene miteinander zu assoziieren und in einen Zeitlichen Kontext zu bringen. Aufgrund dieser Eigen-

schaften bietet es eine gute Grundlage für den Anwendungsfall.

4.1 Modell

Für die Erstellung des Modells wurden tensorflow und keras eingesetzt. Diese beiden Bibliotheken bieten die Möglichkeit mit einfachen Interfaces komplexe funktionalitäten umzusetzen.

4.1.1 Architektur

Das Netzwerk folgt der Architektur die in Listing 3 dargestellt ist. Es besteht aus mehreren Convolutional LSTM Layern die über eine jeweils kleinere Fenstergröße verfügen. Um die generalisierung des Netzwerks zu verbessern und geringere Trainingszeiten zu ereichen werden Normalisierungs Layers eingefügt. Die Netzwerkarchitektur wurde Aufgrund der guten Ergebnisse, die im Beispiel von Keras [5] erziehlt wurden, gewählt.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d (ConvLSTM2D)    (None, 10, 128, 128, 128) 1651712
_________________________________________________________________
batch_normalization (BatchNo (None, 10, 128, 128, 128) 512
_________________________________________________________________
conv_lst_m2d_1 (ConvLSTM2D)  (None, 10, 128, 128, 128) 1180160
_________________________________________________________________
batch_normalization_1 (Batch (None, 10, 128, 128, 128) 512
_________________________________________________________________
conv_lst_m2d_2 (ConvLSTM2D)  (None, 10, 128, 128, 128) 131584
_________________________________________________________________
batch_normalization_2 (Batch (None, 10, 128, 128, 128) 512
_________________________________________________________________
conv3d (Conv3D)              (None, 10, 128, 128, 1)   3457
=================================================================
Total params: 2,968,449
Trainable params: 2,967,681
Non-trainable params: 768
_________________________________________________________________
```

Listing 3: Ausgabe des model.summary() Aufrufs

Insgesamt beinhaltet das Modell 2967681 trainierbare Parameter. Auf dem System welches zum Training verwendet werden soll werden diese Parameter durch float32 repräsentiert welcher 4 Byte im Speicher benötigt. Dies führt auf Basis der Berechnung in Formel 1 zu einem minimal benötigten Speicherbedarf von 11.87 GB. Dieser muss auf der zum Training verwendeten Platform Platz im Arbeitsspeicher finden Können.

$$2967681 \cdot 4\ Bytes = 11.87\ GB \tag{1}$$

4.2 Training

Beim Training wurde zunächst die dynamische Darstellungsform angewandt. Diese lieferte allerdings transformierte und rotierte Bilder bei denen für eine erneute Prediktion die Transformation und Rotation bestimmt werden musste. Um diesen Prozess zu vereinfachen wurde deshalb das Training auf die statische Darstellungsform umgestellt.

4.2.1 Spezifische Loss-Funktion

Da die Trainingsbilddaten viele Bereiche enthalten die für die Performance des Netzwerks nicht relevant sind, wurde eine für das Netzwerk zugeschnittene Loss funktion Implementiert. Diese Analysiert den Mean-Squared Error in verschiedenen Bereichen des predizierten Bildes.

Zukünftige Aufenthaltsorte von Objekten Hier wird der Mean-Squared Error für die Pixel in denen Sich das Objekt aufhalten sollte berechnet. Dieser wird mit einem Verhältnis von 80 % gewichtet.

Hintergrund und Karte Über den Rest des Bildes wird ebenfalls ein Mean-Squared Error Wert berechnet. Dies ist notwendig, da das Netzwerk sonst das gesamte Bild in Objektfarbe einfärben würde und dadurch eine hohe Genauigkeit erreicht. Der Wert wird dann mit 20 % gewichtet.

4.2.2 Generierung der Trainingsdaten zur Laufzeit

Um das Training mit der gesamten Menge der Szenariodaten auf einer Platform mit beschränkten Ressourcen zu ermöglichen wurde ein Gernerator für Trainingsdaten implementiert. Dieser verwendet eine Datenstruktur die Dateipfade enthält und lädt die Trainingsdaten immer erst zur Lauftzeit in den Arbeitsspeicher.

4.2.3 Gradient Accumulation

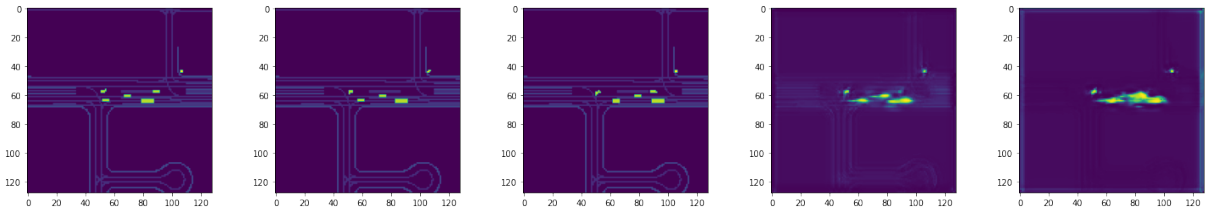
Zudem wurde das Prinzip der Gradient Accumulation eingesetzt um den auf der Grafikkarte benötigten Speicherplatz zu reduzieren. Hierbei werden die Batches die zum Training verwendet werden aufgeteilt. Diese werden dann nacheinander trainiert. Der resultierende Gradient wird dann im Anschluss akkumuliert. Dadurch entsteht beim Training der gleiche Effekt wie bei größeren Batch Größen. Der Speicherbedarf bleibt jedoch konstant klein. [7]

5 Ergebnisse

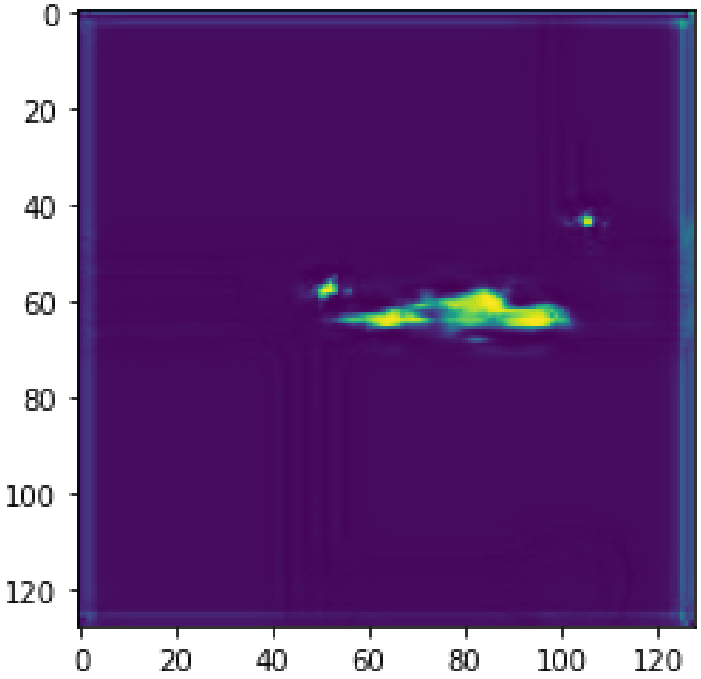
Der Ansatz ein Convolutional LSTM für die Bewegungsvorhersage einzusetzen zeigt erste vielversprechende Ergebnisse. Alle Prediktionen von Abbildung 5 bis Abbildung 7 dienen der Veranschaulichung der Performance des ConvLSTMs. Besser Sichtbar sind diese jedoch in Form von Animationen. Diese können in dem beigefügten Jupyter Notebook inference.ipynb erzeugt werden. Hierbei wird die Datenkonvertierung nicht mehr durchgeführt.

Für die Erstellung der Prediktionen wurde das Trainierte Convolutional LSTM verwendet. Für Prediktionen die über 100 ms hinaus gehen, wurde das Ergebnis als jeweils 10tes Bild wieder in das Netzwerk eingespeist.

Geringe Objektdichte In Szenarien mit gerninger Fahrzeugdichte passen die Ausgaben des Modells grundlegend zu den Refernzdaten des Validierungsdatensatzes. Dies ist im folgenden Beispiel erkennbar. Besonders in Abblidung 5 bei +1 s ist gut zu sehen wie die Bewegung der drei größeren Objekte weiter prediziert wird.



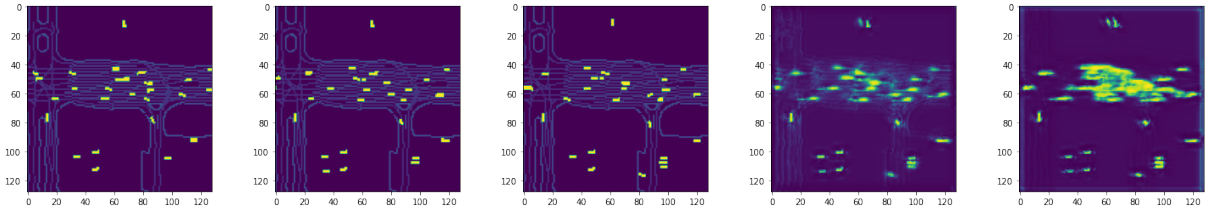
(a) -1 s (b) -0.5 s (c) 0 s (d) +0.1 s (e) +1 s



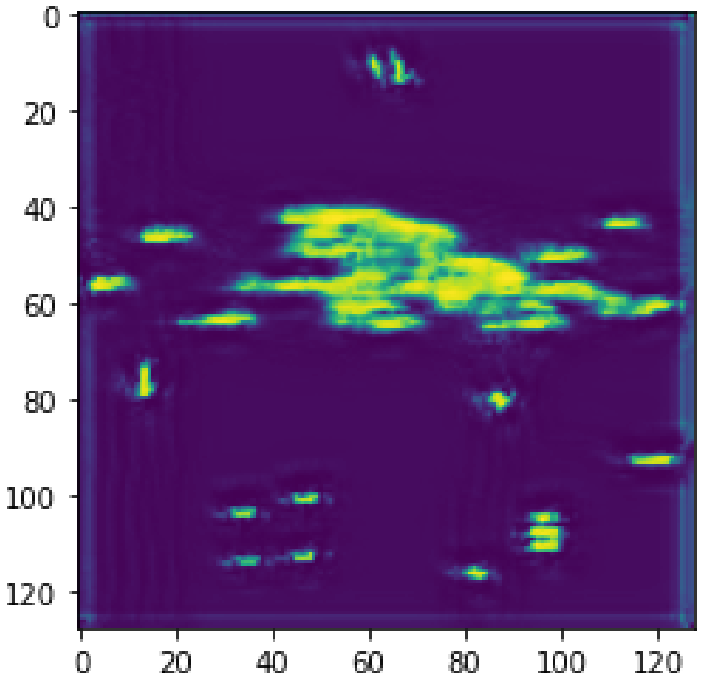
(f) Vergrößerte Darstellung der Prediktion bei +1 s

Abbildung 5: Beispiel für Szenario mit geringer Objektdichte

Komplexes Szenario Für den Fall eines komplexeren Szenarios mit vielen Objekten, unterschiedlichen Fahrtrichtungen und komplexeren Kartensituationen, liefert das Modell unzuverlässige Ergebnisse. Im folgenden Beispiel werden alle Objekte in die gleiche Richtung prediziert.



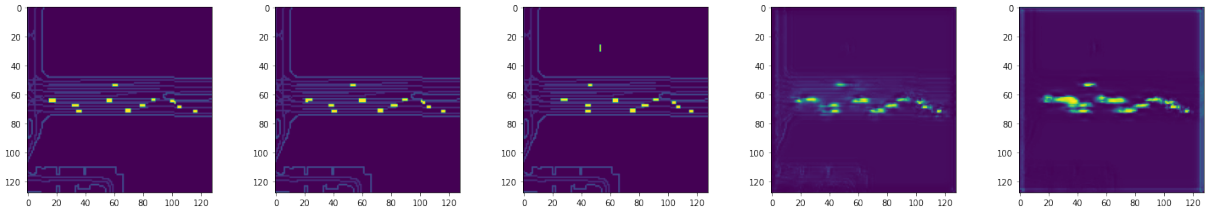
(a) -1 s (b) -0.5 s (c) 0 s (d) +0.1 s (e) +1 s



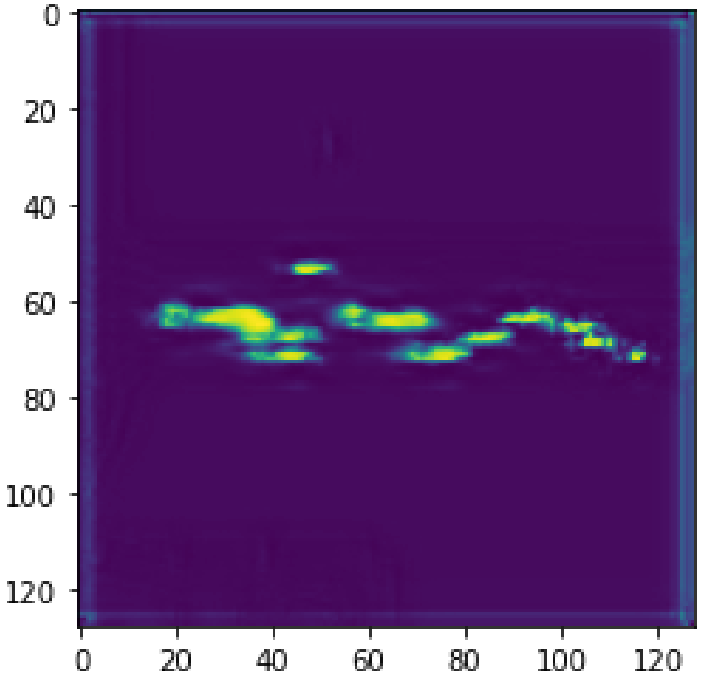
(f) Vergrößerte Darstellung der Prediktion bei +1 s

Abbildung 6: Beispiel für komplexes Szenario mit Gegenverkehr

Statisches Hindernis Ein Beispiel bei dem bessere Ergebnisse erziehlt wurden ist das folgende Szenario bei dem die Fahrzeuge für eine statisches Hindernis am Straßenrand bremsen müssen. Hier liegen die Prediktionen relativ nah an den Refernzdaten.



(a) -1 s (b) -0.5 s (c) 0 s (d) +0.1 s (e) +1 s



(f) Vergrößerte Darstellung der Prediktion bei +1 s

Abbildung 7: Beispiel für Szenario mit Geschwindigkeitsdifferenz

Probleme durch nichtlineare Einfärbung bei Rasterisierung Da bei der Datenverarbeitung eine nichtlineare Funktion zum einfärben der Objekte und Kartendaten verwendet wurde, unterscheiden sich die Trainingsdaten zum Teil stark in der Intesität. Dies fürhrt dazu, dass das Decoding der Bildinformationen in Positionsdaten auf Basis von klassischer Bildverarbeitung nur aufwändig möglich ist. Daher kann im Rahmen dieses Projketes keine aussagekräftige Auswertung über die Genauigkeit der Prediktionen erfolgen.

Durch inkonsequente Einfärbung der Trainingsdaten ist es zudem nicht möglich die Daten aufzubereiten und mehrfache Prediktionen durchzuführen. Hierdurch sollte ursprünglich eine gesamte Vorausschau von 8 Sekunden erreicht werden.

Verbesserungsmöglichkeiten bei der spezifischen Loss-Funktion Die Kartendaten hinterlassen Fragmente bei der Vorhersage. Diese können vermutlich durch die Bear-

beitung der Loss-Funktion beeinflusst und reduziert werden.

Training mit längeren Zeitabschnitten Aufgrund der limiterten Hardwareleistung wurden nur beim Training mit dem ersten Zeitschritte der 1000 Scenarios verwertbare Ergebnisse erziehlt. Bei einem Training mit allen Zeitschritten der 1000 Szenarios ist zu erwarten, dass bessere Ergebnisse erziehlt werden können.

5.1 Lessons Learned und Future Work

- Gradient Accumulation ermöglicht den Speicherbedarf für Zwischenergebnisse auf der GPU zu verringern.
- Der Rechenaufwand für die Rasterisierung der Daten, sowie der benötigte Speicherbedarf für Convolutional LSTMs sollte nicht unterschätzt werden.
- Die Inference Zeit ist deutlich schlechter als bei einer Bild-Klassifikation. Dies liegt an der größe des Netzwerks und der Menge der Eingangsdaten. Dies sollte für die praktische Verwendbarkeit des ConvLSTM+Raster Ansatzes berücksichtigt werden.
- Durch höhere Auflösung des Rasters könnten Objekte besser unterschieden und unabhängig voneinander prediziert werden. Dies führt allerdings zu einer geringeren betrachtbaren Kartengröße oder zu einem größeren und langsameren Netzwerk.

Literatur

[1] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov, "Large scale interactive motion forecasting for autonomous driving : The waymo open motion dataset," 2021.

[2] Waymo-Research, "Waymo-open-dataset/protos." `https://github.com/waymo-research/waymo-open-dataset/tree/master/waymo_open_dataset/protos` [2021-07-13], 2021.

[3] Y. Jeong, S. Kim, and K. Yi, "Surround vehicle motion prediction using lstm-rnn for motion planning of autonomous vehicles at multi-lane turn intersections," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 1, pp. 2–14, 2020.

[4] J. Wu, J. Ruenz, and M. Althoff, "Probabilistic map-based pedestrian motion prediction taking traffic participants into consideration," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1285–1292, 2018.

[5] A. Joshi, "Next-frame video prediction with convolutional lstms." `https://keras.io/examples/vision/conv_lstm/` [2021-07-13], 2021.

[6] Waymo-Research, "Waymo open dataset motion tutorial." `https://github.com/waymo-research/waymo-open-dataset/blob/master/tutorial/tutorial_motion.ipynb` [2021-07-13], 2021.

[7] R. Haleva, "What is gradient accumulation in deep learning?." https://towardsdatascience.com/what-is-gradient-accumulation-in-deep-learning-ec034122cfa [2021-07-13], 2021.