Motion Prediction for Autonomous Cars with Interaction Prediction

Roshan Benefo rbenefo@seas.upenn
Rahul Maganti rmag@seas.upenn
Sheil Sarda sheils@seas.upenn

Abstract

Motion prediction is one of the few outstanding unsolved tasks of autonomous driving, and is critical to the deployment of safe autonomous vehicles (AV)s that are able to navigate complex environment. We evaluate a number of deep learning approaches to motion prediction, and propose a novel combination of a convolutional social pooling and BEV convolutions to predict vehicle trajectories.

1 Introduction

In order to navigate complex environments, autonomous vehicles need to be able to generate accurate predictions of the motion of other agents. Robust motion prediction is an essential part of a successful autonomous driving system: it can help the autonomous vehicle (AV) decide when to change lanes, react to unexpected situations, and plan efficient paths around other agents. However, motion prediction remains one of the few remaining unsolved tasks in autonomous driving, lagging far behind that of perception, planning, and control.

1.1 Contributions

- We propose a novel combination of Convolutional Social Pooling and Convolutional Birds Eye View (BEV) Encoding to model the interactions between agents and learn road geometry to predict agent trajectories.
- Propose open problems for future research, and provide a thorough analysis of several existing motion prediction methods.

2 Background

Broadly speaking, motion prediction systems for self driving cars can be split into two categories: classical, and deep learning-based methods. The simplest classical method is physics-based: simply taking the historical physical state of the agent (its position, velocity, and acceleration), inputting that into a physics model, and using that predict the trajectory. Physics-based models, however, are only really reliable on short timescales– they cannot hope to predict agent trajectories that are governed by agent decision-making.

Another classical method is maneuver recognition, where the AV attempts to recognize the maneuver an agent is currently engaged in, and projects out its trajectory based on this maneuver. The maneuver can then be projected out either by inputting a custom physics model based on this maneuver (e.x. the AV might have a physics model for the maneuver "turn left"). This, however, is a limited approach given the diversity of potential maneuvers in driving: it becomes difficult to recognize all potential maneuvers and convert these labels into trajectories through a rules-based approach.

Researchers and AV engineers more recently have turned to deep learning to help develop robust algorithms for motion prediction. A wide variety of deep learning frameworks have been proposed for the problem, including convolutional architectures, recurrence, reinforcement learning, and more recently, graph-based networks. We go into these proposals in greater depth in the following section.

3 Related Work

Modern deep learning approaches follow an "end-to-end" learning methodology that attempts to minimize *a priori* assumptions on the data. While this has been wildly successful in the past few years, especially in domains like image classification, it is not necessarily suited to multi-agent systems with complex interactions. Understanding the interactions between agents in the system is therefore critical to devising methods that can accurately predict their motion. Recent approaches to solving the interaction prediction problem assume an *a priori* representation on the data and leverage this representation to produce better trajectory estimates. Alahi et. al (1) propose social LSTMs, which jointly model and predict the motion of pedestrians in dense crowds through the use of a social pooling layer. Deo, Trivedi (2) propose a similar model with a social pooling layer but also incorporate the spatial structure of freeways. Graph neural networks offer a more general approach to joint interaction prediction. Lee et. al (3) model the relationships among agents as vertices and edges on a graph to classify vehicle-vehicle interactions and predict their trajectories.

4 Dataset

Our data comes from the Motion Prediction Challenge posed by Lyft in 2020. The dataset provided in the challenge includes more than 1,000 hours of driving data collected with over 16,000 miles of driving in Palo Alto, California. The competition required using the L5Kit toolkit to preprocess data, train models, and evaluate results.(4)

The L5Kit toolkit uses `numpy` structured arrays that pull from data stored in a `zarr` format. The important fields from these arrays that we used from the dataset include:(i) `scenes`: driving episodes acquired from a given vehicle; (ii) `frames`: snapshots in time of the pose of the vehicle, consisting of target; (iii) `agents`: a generic entity captured by the vehicle's sensors. Each agent state describes the position, orientation, bounds, and type.

5 Approach

We elected to focus on two main tasks in our exploration: **interaction prediction**, and **road geometry recognition**. We believe that these aspects are two important predictors of future motion planning.

Interaction prediction is important for motion prediction because in many cases the future trajectory of agents depends on the future trajectories of its neighboring agents, including vehicles and pedestrians. Consider the case where a pedestrian can either yield to a vehicle or the vehicle yields to the pedestrian. Only predicting the marginal future trajectory of one of these actors ignores the possible modes of interaction between then, yielding an inaccurate set of future trajectories. (3)

Road geometry recognition is important because on-road vehicles must obey road geometry, i.e. run within the constraint of road shapes. An example of where trajectories can be deemed infeasible because of road geometry is presented in Figure 7 of the Appendix. For our road geometry models, we passed in to our models historical birds eye view (BEV) rasterizations of the road, which contained images for the past 10 timesteps of the surrounding road geometry (including lane markings and crosswalks). These images also contained rasterizations of the focus vehicle (the one we are predicting the trajectory of), and its surrounding neighbors. Three of our models were explicitly dedicated towards road geometry recognition– our two convolutional ConvNets, and our linear SimpleNet. We discuss these models in greater detail in the experimental results section.

For our interaction prediction models, we passed in information about the historical locations, velocities, and orientations of the focus vehicle and its neighbors. Two of our models were explicitly focused on interaction prediction– our Convolutional Social Pooling (CSP) network, and our GraphNet graph-based neural network.

5.1 Data Preprocessing

The BEV rasterizations needed no preprocessing– when we needed them, we simply accessed them through the provided Lyft SDK and fed them into the network. The position and orientation data of neighboring vehicles, however, was not readily available in the datastructure provided by Lyft's SDK, and heavy modifications had to be made of the SDK to access it. Rather, only information about the

target car (the one we are predicting) is readily available. As such, we needed to modify the SDK to access the frame and scene number associated with the target car data we received, and efficiently search through the dataset to match the frame and scene number with the data of the surrounding neighbors. Once acquired, position and orientation information of the surrounding neighbors was transformed from the world reference frame to the target vehicle reference frame via a transformation matrix we created based on the target vehicle orientation and position.

The AV's perception system is not perfect, however, and some agents in the scene were not seen during certain time steps, either because they were occluded or because data association failed to identify them in the scene. As such, the data comes with an auxiliary observation Boolean list which notes out for which time steps each agent was or was not observed. We used this Boolean list to remove datapoints for either the target vehicle or neighboring vehicles where they were not observed by the AV, replacing them with a padding value of 0.

5.2 Accuracy Metric

To evaluate our models, we used the competition accuracy metric, negative multi-log likelihood, and our own qualitative metric of how accurate the rendered output trajectories seemed. To calculate negative multi-log likelihood (NML), we use the following equation:

$$L = -log(\sum_{k} exp(log(c^{(k)} - \frac{1}{2}\sum_{t}(\overline{x}_t^{(k)} - x_t)^2 + (\overline{y}_t^{(k)} - y_t)^2)) \quad (1)$$

Where we are outputting proposed trajectories $k = 1, 2...n$ each with confidence c and parameterized by coordinates $\overline{x}_t, \overline{y}_t$. The ground truth coordinates of the observed trajectory are x_t and y_t. With this metric, a lower score means a better result, one that is closer to the ground truth. Qualitatively, we looked for evidence that the outputted trajectory was reasonable (e.x. it did not go off of the road, more or less followed the ground truth, and did not result in a collision with another vehicle).

5.2.1 Loss Functions and Training Implementation

To train our models, we mainly used the same metric we were using for our accuracy metric, NML. We believed that NML was an accurate metric that correctly optimized for the variables we wished to optimize for–it penalized trajectories that strayed from the ground truth, but also allowed for uncertainty in the output trajectories. We also, however, experimented with MSE loss for models where we only produced one trajectory– experimentally, for these models, we found that MSE loss generated a higher accuracy score for these models, while NML produced a higher accuracy score for models that outputted multiple trajectories (to use MSE loss on these models, we would take the most likely trajectory and pass it into the loss function). We trained our models using an Adam optimizer with a learning rate of between 1e-3 and 1e-4. We found that our models would consistently reach minimum loss after only 2 or 3 epochs.

6 Experimental Results

6.1 Models

Network	**NML Score**	**Qualitative Assessment**
SimpleNet (Baseline)	500	Trajectories oftentimes went off roads, or would result in collisions.
ConvNetV1	425	Trajectories oftentimes would result in collisions.
CSPV1	435	Trajectories would oftentimes go off the roads, or drift between lanes.
SuperNetV1	363	Trajectories would sometimes drift between lanes.
ConvNetV2	150	Trajectories oftentimes would result in collisions.
SuperNetV2	100	Trajectories closely match ground truth, avoid collisions and stay within lane markings.

Table 1: Summary of Model Output

We architected and implemented four deep learning algorithms for motion prediction, starting with a simple baseline model and later adding additional layers of complexity. The outputs of these models are summarized in the below table, and discussed in depth later. See Appendix for figures.

6.1.1 SimpleNet: Baseline Linear Model

First, as a baseline algorithm, we used a simple linear network that took in the BEV rasterized images, flattened them, and passed them through a linear layers with ReLU activations to predict a single trajectory. This model a relatively high NML score of 500, and outputted trajectories that often went off roads, predicted rapid and frequent lane changes, and would often result in collisions.

6.1.2 ConvNetV1: Convolutional Model

SimpleNet's linear architecture meant that it was not able to extract relational meaning from the input BEV rasterizations– for example, it was not able to recognize the distance between the target vehicle and other agents, and not able to extract information from the distance each vehicle was from the road edge. As such, for our second model, we turned to a convolutional architecture to help capture some of this relational meaning. In ConvNetV1, we passed BEV rasterizations through a pre-trained convolutional backbone, before pushing them through a fully connected layer to produce the ouptut predictions. The model architecture is included in appendix Figure 1; in the model, we used XCeptionNet41 pre-trained on ImageNet as our backbone. This model provided an improvement on SimpleNet, outputting a NML score of 425, and generating outputs that added did not as often go off of the road, but frequently would have resulted in collisions with other vehicles.

While ConvNet's output is better than SimpleNet's, we believed that its score was relatively high due to the fact that it was unable to accurately predict the trajectories of neighbors *around* the target, nor represent how the agent would interact with vehicles around it. This, we believe, prevents the network from understanding that vehicles around it are moving objects that have future trajectories that may intersect with the proposed trajectory outputted by the model.

6.1.3 ConvNetV2

While experimenting, we also implemented an improved version of ConvNetV1, with XCeption71 as a backbone, and adaptive average pooling to further improve ConvNet's ability to understand the connections between different portions of BEV raster. We also elected to output multiple trajectories (each with a confidence value), so that the model could express uncertainty in its output. This network generated a much better score than ConvNetV1 of 155, but qualitatively was observed to output multiple trajectories that would result in collisions (Image 1a in Appendix Figure 2). Once again, we believe that the model output could be improved by taking into account interaction prediction.

6.1.4 CSP: Convolutional Social Pooling

In our next model, we explored how well interaction prediction could help us predict the future trajectory of a vehicle. A modern approach to interaction prediction is convolutional social pooling, where neighbor states are encoded in an LSTM, and the LSTM hidden states are passed into a "social tensor".(5) The idea is that the social tensor gives spatial context to the LSTM encoding, and as such represents both the relative positions of each neighbor and their encoding (which may carry information about their driving style, speed, acceleration, etc). This helps represent the notion of local interaction, and helps the model understand how the target vehicle is interacting with the agents around it.

This social tensor is run through a convolutional network, and the output concatenated with the LSTM encoding of the target vehicle state. Finally, the concatenated encodings are sent through a decoder to predict the state of the agent. We implemented this idea in CSPV1– the network's architecture is shown in Appendix Figure 3. This network, however, quantitatively performed worse than ConvNetV1, with a NML score of 435. Qualitatively, however, we see that the areas where the predicted trajectories diverge from the ground truth oftentimes occur when the network appears to react to other vehicles: the predicted trajectories seem to *avoid* other vehicles on the map. These outputs are seen in Appendix Figure 4.

We believe that this may be because the network is only fed information about interactions between vehicles. In the dataset, many of the close interactions between agents involve agents *avoiding* each

other– as such, we believe that the network may have learned to output trajectories that similarly result in avoidance. To help mitigate this problem, we implemented a network that combined the outputs of the ConvNet (which is able to understand how agents will drive on a road as a function of road geometry), and CSP (which can understand interaction prediction). We will talk about this combined network in following section.

6.1.5 SuperNetV1 and V2: Convolutional Social Pooling+ ConvNetV1/V2

In our SuperNet networks, we combined the output of the CSP and a pre-trained ConvNet in the CSP decoder, by passing in BEV images to the ConvNet, scaling its output via a linear layer, and adding the resulting output to the predicted trajectory coming from the CSP. The architecture for this model is show in Figure 5.

Our SuperNets generated the best score of all of the networks we tested– SuperNetV1, which combined CSP with ConvNetV1, had an NML of 363, 100 below the scores of ConvNetV1 and CSP by themselves. SuperNetV2 had a score of 100, 50 below the score of ConvNetV2, and 300 below the score of CSP. The outputs of SuperNetV2 are shown in Figure 6. We believe that this is because these networks are able to combine an understanding of interaction prediction with road geometry, and thus produce trajectories that closely match the ground truth, stay within lane markings, and avoid collisions.

7 Discussion

We have evaluated a number of deep learning approaches for motion prediction, and find that combining Convolutional Social Pooling and a normal CNN of BEV images can yield better predictive ability than each method separately. This indicates that accurate motion prediction likely requires an understanding of *both* interaction prediction and road geometry.

Future work could look at incorporating traffic light information in our predictions. In some of our outputs from SuperNetV2, we predict cars entering an intersection even if the ground truth keeps them non-moving (presumably as the light is red). Incorporating traffic light data may help the model learn to take into account red and green lights in its predictions.

7.1 GraphNet: Graph Neural Network

Graph neural networks provide another, more general way of representing interactions between agents. We attempted to implement a graph neural network during this project– our architecture was modeled after (3). Future work could expand on this implementation and evaluate its predictive ability on the Lyft dataset.

Representation of Agents. Like with the convolutional networks described above, we need a way to represent the information associated with a particular scene. Graphs provide a relatively intuitive way of encoding the structural information for autonomous vehicles. We let the nodes $\{V\} = \{x_1, \ldots, x_n\}$ be the agents and assign features $\{f(\boldsymbol{x})\}_{i=1,\ldots,n}$ from the output of the RNN encoder in the world coordinate frame. Likewise, we construct a set of edge features by computing the pair-wise manhattan distances between all of the agents. (See Appendix Figure 13 for our network architecture. There are three two steps to any graph neural network that allow us to encode the relationships among the agents in the scene: (i) **Edge model** which combines the representations of each edge and its terminal nodes to output an updated edge representation; (ii) **Node model** which operates on each node to aggregate the representations of incident edges and outputs an updated node representation. (6)

While the theoretical underpinnings of graph networks are sound, we nonetheless ran into major difficulties in attempting to use them for trajectory prediction. As the data was passed through the Graph Convolutional layers, many of the node and edge feature embeddings became undefined. When passed to the RNN Decoder and the Multi-layer perceptron or fully connected layer, these values remained undefined. The results were therefore unusable.

References

[1] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[2] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," *CoRR*, vol. abs/1805.06771, 2018. [Online]. Available: http://arxiv.org/abs/1805.06771

[3] D. Lee, Y. Gu, J. Hoang, and M. Marchetti-Bowick, "Joint interaction and trajectory prediction for autonomous driving using graph neural networks," 2019.

[4] "Lyft 3d object detection for autonomous vehicles." [Online]. Available: https://www.kaggle.com/c/3d-object-detection-for-autonomous-vehicles

[5] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," *Computer Vision and Pattern Recognition 2018, 1468-1476*, 2018.

[6] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.01261.pdf

[7] Y. Yoon, T. Kim, H. Lee, and J. Park, "Road-aware trajectory prediction for autonomous driving on highways," *Sensors*, vol. 20, no. 17, p. 4703, Aug 2020. [Online]. Available: http://dx.doi.org/10.3390/s20174703

Appendix

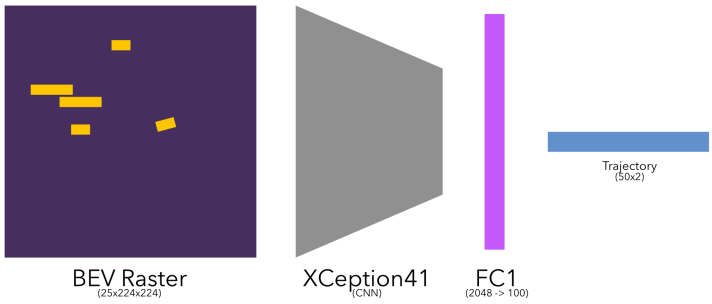
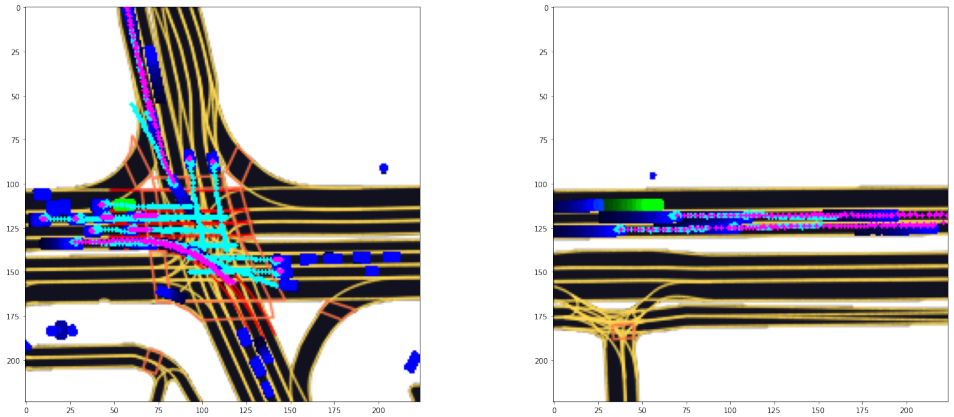


Figure 1: Model Architecture of ConvNetV1



(a) 1a (b) 1b

Figure 2: Output of ConvNet2, our convolutional network that implemented convolutional social pooling. Pink lines represent the ground truth trajectory; cyan represents the predicted one. While the network outputs multiple trajectories, only the most confident one has been displayed for clarity. As can be seen from Image 1a, some of the outputs of this network would result in collisions.

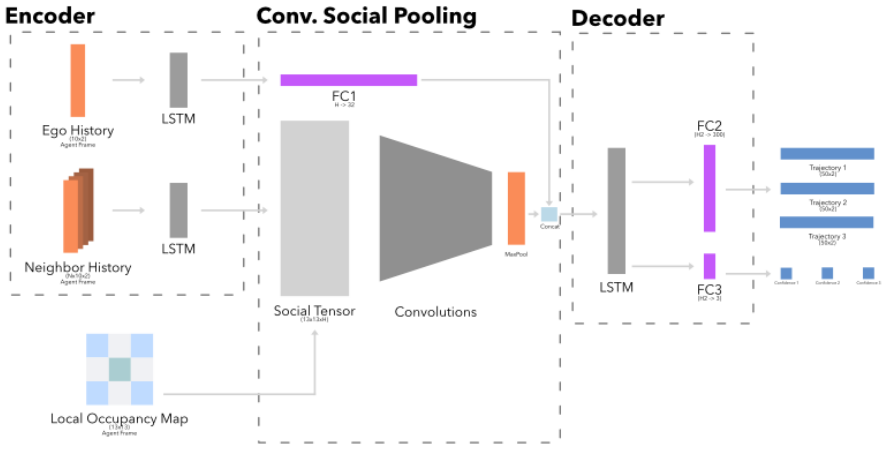
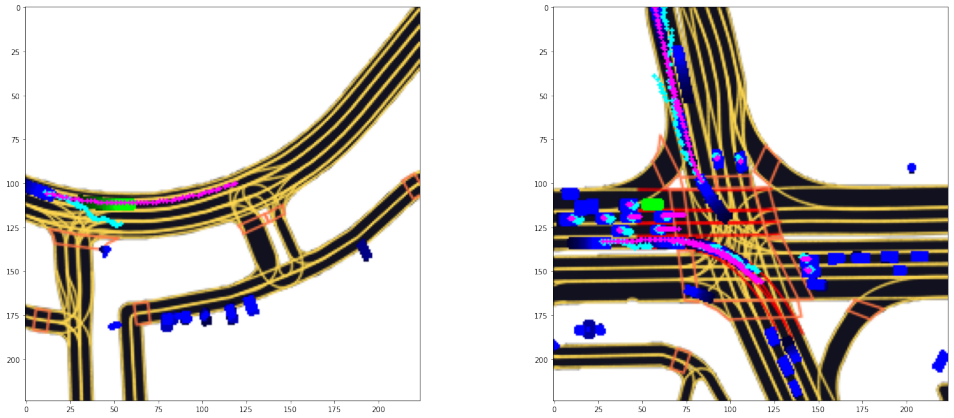


Figure 3: Model Architecture of CSPV1



(a) 1a (b) 1b

Figure 4: Output of CSP, our network that implemented convolutional social pooling. Pink lines represent the ground truth trajectory; cyan represents the predicted one. Many of the outputs of the network, like the one seen in Image 1a, appear to try to avoid other vehicles.

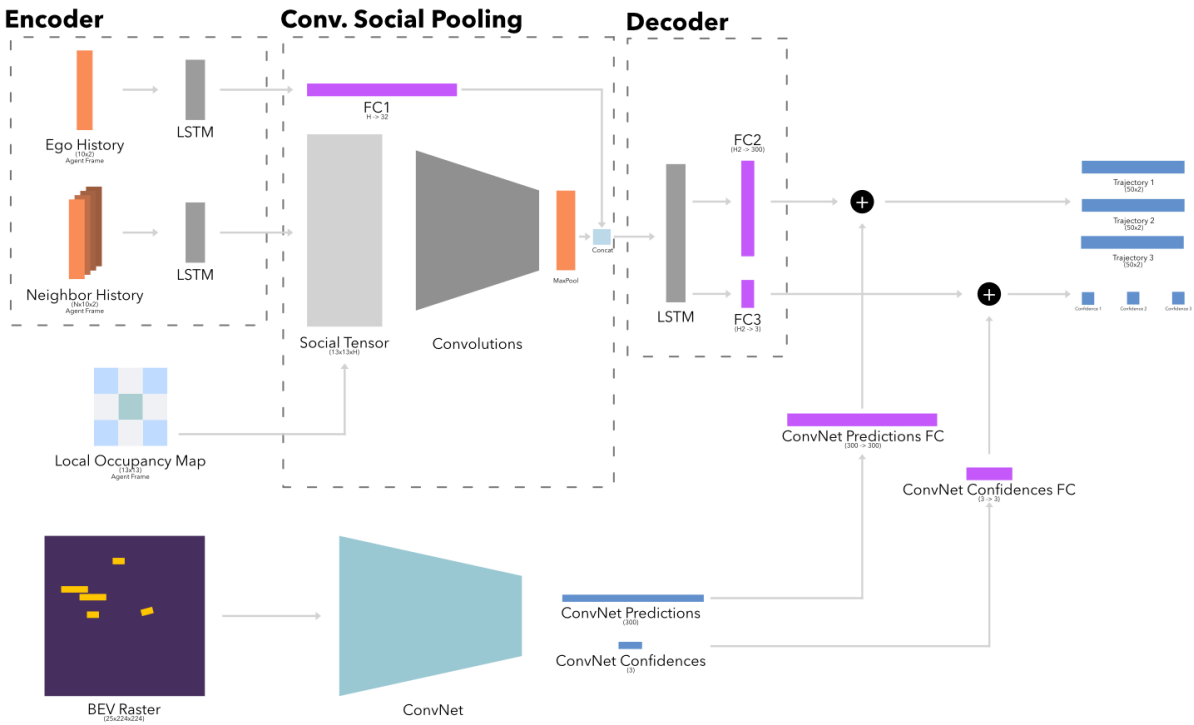
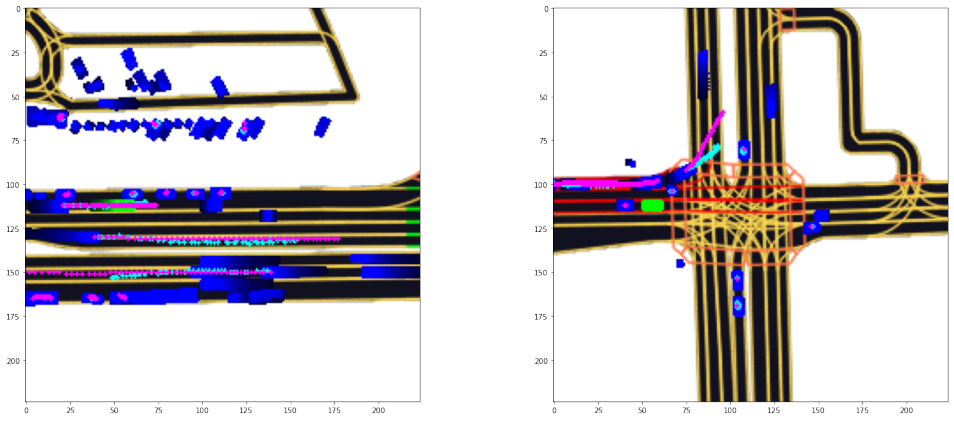


Figure 5: Model Architecture of SuperNet



(a) 1a (b) 1b

Figure 6: Output of SuperNetV2, our network that combined convolutional social pooling with ConvNetV2. Pink lines represent the ground truth trajectory; cyan represents the predicted one. As can be seen, the network predicts the ground truth trajectory with high accuracy, and rarely outputs trajectories that go off of the road or would result in collisions.

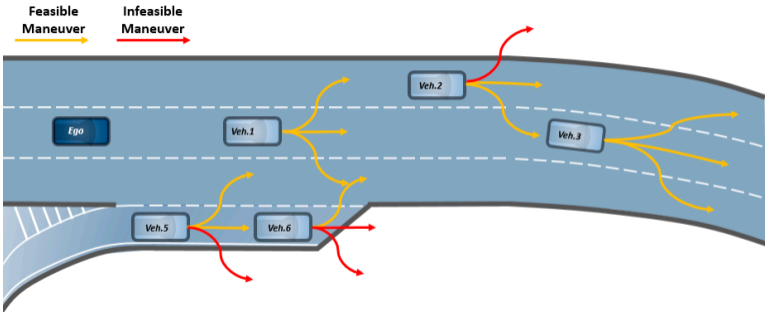


Figure 7: Importance of Road Geometry in Motion Prediction (7)

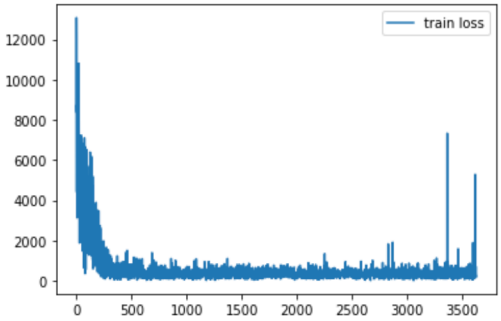


Figure 8: CSP Training Loss

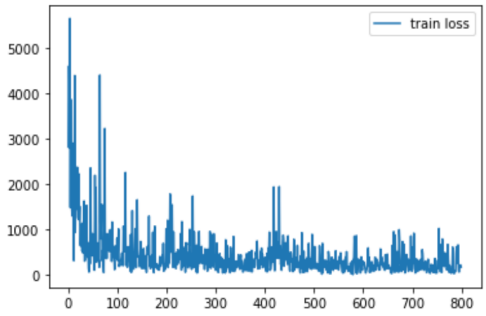


Figure 9: ConvNetV1 Training Loss

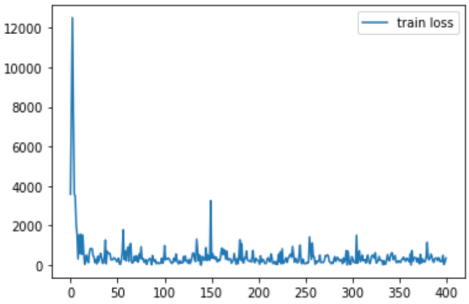


Figure 10: SuperNetV1 Training Loss

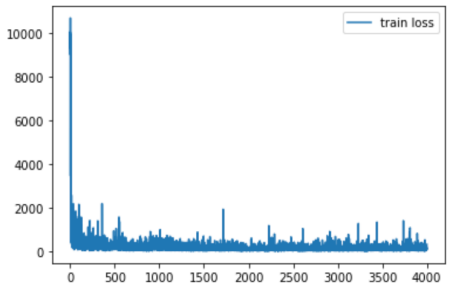


Figure 11: ConvNetV22 Training Loss

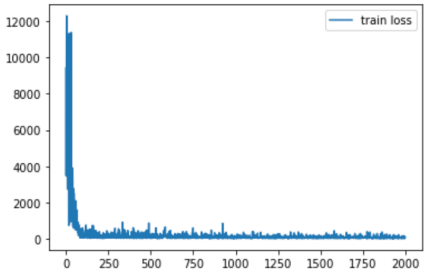


Figure 12: SuperNetV2 Training Loss

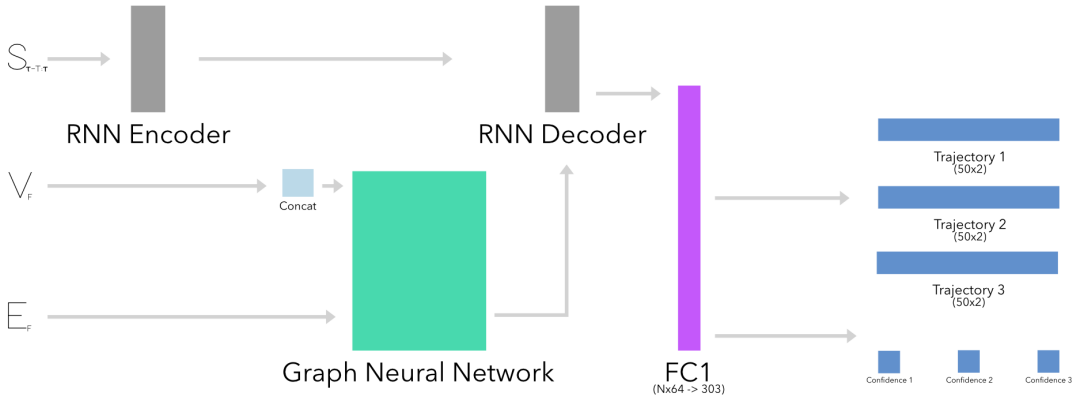


Figure 13: GraphNet, our graph neural network architecture diagram. We pass in the state trajectories of all the agents to the RNN Encoder. We then construct a graph with nodes and edge features using the RNN embeddings. The output of the graph network is passed to a RNN decoder and then a fully connected layer to predict 3 trajectories and their associated confidences.