INFERN: INtermediate representations for FuturE pRediction on Nuscenes dataset

Aravind Mahadevan
A12175508
armahade@ucsd.edu

Sanjeev Sinha
A13419612
sas115@eng.ucsd.edu

Abhishek Kandoi
A53315077
kandoi@eng.ucsd.edu

Eric Megrabov
A12177906
emegrabo@eng.ucsd.edu

Abstract

The ability to predict where different objects, specifically vehicles in road scenes, will be in the future is an extraordinarily useful task that can be applied to a plethora of real-world situations. This task has been possible in the past through the use of color information along with high-fidelity maps, but those strategies are unable to generalize properly when other datasets are used. Intermediate representations using onboard sensors claim to be able to solve this problem and allow us to generalize. In this paper, we use the NuScenes dataset to generate our own intermediate representations and use the architecture described in INFER to perform trajectory prediction.

1. Introduction

Trajectory prediction has been tackled repeatedly throughout the past several years with the intent of determining the best possible way to see where the driver of an ego vehicle as well as other vehicles may be traveling for the sake of safety. Knowing where a car or truck will likely be located has the potential to keep drivers and bystanders significantly safer and out of harm's way by anticipating issues that may arise before they happen. Previously renowned and accepted techniques typically all involve the use of very high quality texture and color information for given road scenes. As a result, any variation in provided datasets can cause issues for the prediction architectures since datasets that do not have extremely rich data are unable to properly capture important features in the scene. INFER, which stands for Intermediate representations for distant future prediction [10], seeks to solve such issues by using an intermediate representation that is almost entirely based on semantics of images. It uses inputs to an autoregressive neural network model, specifically using an LSTM. Furthermore, the authors of INFER also trained the model using high level semantic cues rather than using low level pixel values. Using semantic segmentation, instance segmentation, and disparity, they create Bird's-Eye-View (BEV) intermediate representations which become input to the model. The authors demonstrate that the intermediate representation and the model allows for zero shot generalization as the model is able to generalize to other data sets such as Cityscapes and Oxford RobotCar. We specifically want to apply this model to the Nuscenes dataset and see how well it performs in trajectory prediction. Furthermore, we also want to compare the performance of INFER when using intermediate representations generated from sensor data and annotations versus intermediate representations generated from HD maps.

2. Related Research

There have been many other approaches that have been used in the past to perform trajectory prediction. Original research tried to solve the problem by assuming different things about the scenes. For example, [1] assumes that if a set of points in a scene all move at the same speed, then they are part of the same object. In addition, they typically employ the use of non-deep learning techniques such as the extended Kalman filter along with occupancy grids. These strategies also use hand modelling to represent objects as geometric models with cuboids and surface delimiters [4]. Although these strategies work for trajectory prediction, they are much more prone to error due to manual sculpting of the problem as well as assumptions that do not always hold true. A development from such classical techniques is the use of inverse reinforcement learning to perform prediction of paths by estimating what a target does at a given time and using said estimate to apply an action in series. This is highly dependent on control theory as

well as static cameras recording over long periods of time in a given location since it cannot accurately account for both target and other object movement simultaneously in a moving scene. While this works well in a scene where the camera location does not change, it does not scale well to datasets with vehicle-mounted imaging devices. The evolution of trajectory prediction was largely due to the use of RNNs, or recurrent neural networks. One such popular model is called DESIRE [7] and it performed very well on datasets such as KITTI. It uses a conditional variational autoencoder, a RNN encoder-decoder and a control module. The use of neural networks vastly improves the performance over other models, but DESIRE is not presumed to work well for transfer learning. This is especially evident since they use different representations across datasets, unlike INFER, which demonstrates zero-shot transfer. Other techniques like [11] use LSTMs but only with motion data rather than semantic segmentation in order to make predictions, whereas INFER is largely dependent on semantics. There have been some attempts to use Generative Adversarial networks in order to determine where pedestrians will be more naturally in [6]. One other novel and interesting approach [5] employs the use of predicted maneuver classes, such as braking or accelerating, along with LSTMs in order to make estimates regarding where a target trajectory will lie. Lastly, CoverNet [9] uses ResNet-50 and ImageNet in order to perform trajectory prediction with the current and past states of trackable objects as well as a high-definition map obtained from NuScenes. A major advantage of our approach is rather than using high-definition maps, we instead use lidar point clouds mapped to semantic classes as representations of our inputs into the model. As a result, there is no need to obtain highly complex information in order to make the model work; with just cameras and a lidar sensor, it is possible to create simple yet powerful intermediate representations for critical portions of scenes that allow for accurate prediction. By using such an approach, it is easier to further adapt the INFER model to other datasets that do not contain high-definition maps of each scene.

3. Methodology

3.1. Semantic Segmentation

The first step of training the INFER model is obtaining the intermediate representations for each frame of each scene. Each image is preprocessed through the use of an off-the-shelf semantic segmentation network as seen in [2]. This is a version of WideResNet-38 which is highly optimized through the use of in-place activated batch normalization. It is trained on the Mapillary Vistas datasets, which provides 65 different semantic classes. By inspection, the semantic segmentation network transfers well to the NuScenes images with almost entirely accurate segmentations, thus demonstrating that it is appropriate for our task. Figure 1 below is an example output of doing semantic segmentation on the camera images. From these images we obtain the intermediate representations. The intermediate representations are split up into five different classes: Road, Lane, Obstacle, Other Vehicles, and Target Vehicle. The segmentation categories used in order to create said classes can be seen below.

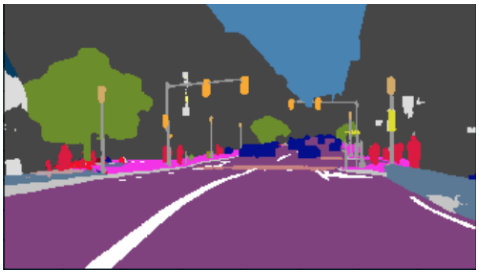


Figure 1. Semantic Segmentation done on Front Camera

- **Road**: Road
- **Lane**: Lane Marking - Crosswalk, Lane Marking - General
- **Obstacle**: Building, Curb, Vegetation

These representations are obtained on four different available ego-vehicle-mounted mono cameras, which are pointed to the front, front-left, front-right, and back. We decided on using these four cameras because it allows us to generate information-rich intermediate representations which can encode surrounding objects and vehicles and allow the model to make a better prediction on where the vehicle will be in the next frame.

3.2. Lidar Data Transformation

Once the semantic segmentation is performed on the image data retrieved from the four cameras on the ego-vehicle, the next step is to take the lidar data and transform it into the image plane in order to do the point painting. In the Nuscenes data set, the lidar data is represented as point cloud data in the lidar data frame. To do this transformation, we first transform the lidar data in the lidar data frame to the global coordinate frame. From the global coordinate frame, the lidar data is transformed into the ego vehicle frame and lastly is transformed into the camera frame. Once the lidar points are in the camera frame, we transform the lidar points by multiplying all points with the camera intrinsics matrix which gives us the lidar points on the image plane. The result of this operation can be seen in Figure 2. Once the lidar points are in the image plane, we filter points that are not in the camera's field of view and points that might be making contact with the ego vehicle itself. After filtering,

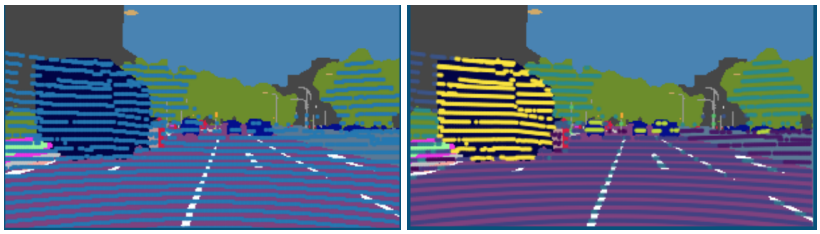


Figure 2. Transformed Lidar points in image plane before point painting and after point painting respectively

we use these lidar points in conjunction with the semantic segmentation to perform the point painting. After the lidar data is point painted, we project these points into the bird's eye view using an orthographic projection from the global coordinate frame. In other words, given $L_{global} \in \mathbf{R}^{3\times N}$, which represents the points in the global frame, and matrix P, we simply perform the following operation to get the points in the bird's eye view:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{1}$$

$$L_{BEV} = PL_{global} \tag{2}$$

Furthermore, the intermediate projections need to capture the temporal nature of the vehicle and the changing surrounding and road conditions. We specifically use the ego vehicle's pose at the time of prediction and transform the lidar points in the global frame in the previous time steps and future time steps with respect to the ego vehicle pose at time of prediction.

3.3. Point Painting

With the semantically segmented camera image and the lidar points in the image plane of the camera, we perform point painting. Our implementation of point painting differs from the point painting method introduced in PointPainting [12]. In PointPainting [12], after the points are projected to the image plane, the authors propose to take the scores generated by the semantic segmentation network and append it to the lidar point. In our case, once we have projected the lidar points into the image plane, we see the value of the pixel at the respective lidar point location in the image and see what class it corresponds to. Then, we simply assign the lidar point to the class it belongs to. In this method, we do not augment the lidar point vector but rather just map the lidar point to the corresponding class. Using [8], we check the pixel value of the lidar point in the image and then see what class it belongs to in order to assign the class to the lidar points. The end result of this process can be seen in Figure 2. This process is extremely crucial in generating the intermediate representations as it allows for the separation of the five different types of representations.

3.4. Intermediate Representation Using Painted Lidar Points

In our dataset, we have 85 scenes for which we need to generate intermediate representations. In each of these scenes, we are predicting the future trajectory of the ego vehicle 4 seconds into the future at any time with 2 seconds of history. Each frame is captured by the four cameras every 0.5 seconds, which leaves every scene to be around 40 frames since 20 seconds are recorded per scene. For each of these scenes, there exist multiple possible sequences we can obtain the intermediate representations of. At every point in time, there simply needs to exist 2 seconds of history (4 frames) and 4 seconds of possible future trajectory (8 frames). Given these sequences of frames generated per scene, we are able to generate the set of intermediate representations required per sequence to feed the LSTM model the correct information in order to obtain a viable future trajectory prediction.

The process described next happens for every possible sequence, for every scene in the nuScenes dataset we work with. We first obtain all of the camera intrinsic data from the dataset for each of the cameras, which are the three front cameras and the back camera. We also obtain the LiDAR data from the point cloud given to us. Once we obtain the ego vehicle pose as well, we transform the original LiDAR points to each of the camera frames using each of the camera intrinsic data and obtain four sets of LiDAR points in each camera frame. The next step is to obtain the time frame representations in each of the camera frames. We grab the (x,y) coordinate pairs from the LiDAR-to-camera transformed points, and create masks of the point clouds inside these camera frames in order to see which points are actually inside each respective image frame.

After this pipeline is complete, we simply keep recording the history of the agent which includes the point cloud around an ego vehicle, the global point cloud, the ego vehicle's pose, the masks generated for each camera frame, and the painted points. Once there is a frame which contains four frames of history, then a prediction can be made from that frame onward, until there are less than 8 frames in the future to predict the location of the ego vehicle in. We obtain the history that is relevant to the frame with the ego

vehicle pose and the rotation matrix with respect to the ego vehicle pose itself in order to put all of the representations in the same reference frame. We then obtain the relevant history required for that specific frame, for which we generate the following intermediate representations: ego vehicle, lanes, obstacle, road, other vehicles, and ego vehicle ground truth.

We acquire the ego vehicle representation by using the bounding box of the ego vehicle itself and filling in the box, showing that a car is there in the representation of the entire occupancy map. Similarly, we generate the other vehicles' channel representation in the same manner: by obtaining the bounding boxes of these vehicles from the bird's eye view and filling in the boxes to show that other vehicles take up this space in the occupancy grid. Acquiring the other intermediate representations was done via the following procedure: depending on the semantic class the rest of the points have in the frame, we map these to either being a lane, road, or obstacle. With this information, we generate occupancy maps for all three of these characteristics which serve as our representations. We then overlap all of these representations because they each exist for a given camera direction. If we combine these grids, we obtain an overall intermediate representation for all characteristics of the frame. To generate the ground truth for all frames in a sequence, we simply create the target ground truth representation of the current frame the next ego vehicle representation, with respect to the current ego vehicle pose. In order to accentuate the represented features in the intermediate representations, we apply some morphological transformations on them. Specifically, we use dilation on the obstacle, lane, and road representations and perform a Gaussian blur on the target vehicle representation. We dilate the obstacle, lane and road representation using a 5×5 kernel for one iteration. The reason we chose to dilate these representations is because there is a sense of connectedness in these representations. Dilation is able to connect bridges between pixels in images and can act as a form of interpolation, which creates a more accurate estimate of the scene. Lane markings in real-world conditions as well is in the camera images are typically connected and using lidar points by themselves does not capture the continuity since the lidar point clouds are generally sparse. This holds true for the road representation as well as the obstacle representation and motivated us to use dilation for these representations as well. For the target representation, we chose to do a Gaussian blur in order to give the model some leeway when it makes a prediction. Our reasoning was that as long as the model predicts close to the actual target then the model shouldn't be penalized as harshly. A Gaussian blur allows the target to be weighted slightly more heavily since it expands its dimensions along each axis. Once the morphological transformations are applied, we normalize all representations to have pixel values between $0 - 255$. See Figure 5 to see the intermediate representations for road, lane marking, and obstacles channels generated using the LiDAR point cloud data for the first frame of the scene named scene-0001 from the NuScenes dataset. Note that the representation is with reference to the ego pose of the vehicle at the time of prediction (i.e. the 4th frame, index starting from 0). See Figure 6 for the intermediate representations of the target and other vehicles channels for the same frame as above.

3.5. Intermediate Representation Using Map

Apart from using LiDAR points and bounding boxes to generate intermediate representations, we also decided to use the map extension provided with the NuScenes dataset for an alternate intermediate representation. For this, we used the layer $drivable\_area$ for the road representation, the $lane\_divider$ layer for the lane channel representation and a combination of the $walkway$ layer and objects like traffic cones and blockades to generate the obstacles channel representation. For the target channel we use the ego pose information of the vehicle and shift everything with respect to the ego pose at the time of prediction for each training sequence. Similarly, the other vehicles channel uses the sample's annotation information to draw the vehicles (as well as humans) and shifts everything with respect to the ego vehicle at the time of prediction for each training sequence.

See Figure 3 to see the intermediate representations generated using the map for the same scene as we did in section 3.4. We used the same representation for the target and other vehicles channel for both LiDAR based representations and map based representations. See Figure **??** for the intermediate representation of the target and other vehicle channel for the same frame as above.

To create these intermediate representations we wrote custom classes built on top of the classes $AgentRepresentation$, $StaticLayerRepresentation$ and $InputRepresentation$ provided with the nuscenes-devkit repository. The channels are all occupancy grid maps, with the color white (255,255,255) representing an occupied position and black representing an unoccupied position. All the channels were rotated to follow a single convention shared with the LiDAR point cloud based intermediate representations. For all channels we used a resolution of 0.5 meters per pixel and a 64 meter lookahead in all 4 directions of the ego vehicle at the time of prediction. This led to 5 channels, each of size 256x256 pixels.

3.6. Model Details

We use a convolutional LSTM model which is trained using our intermediate representations generated in the aforementioned sections. Each of these intermediate represen-

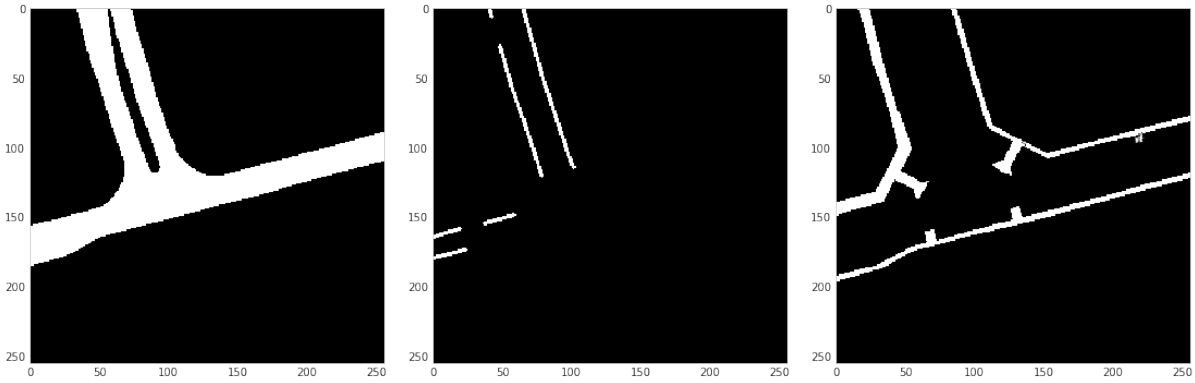


Figure 3. Intermediate Representations for the Road, Lane Marking and Obstacle channels using the Map data

tations are concatenated together with respect to a certain frame, yielding a 6x256x256 matrix for every frame taken in a sequence. For the input, we upsample the matrix to make it a 6x512x512 input as well. We use an LSTM to correctly learn the temporal dynamics of environment, as we are trying to evaluate the correctness of our future trajectory in relation to the ground truth. The model has an encoder-decoder architecture which essentially outputs a probability map with a 256x256 grid of where the vehicle can be in the occupancy grid which represents the environment. Skip connections are also added between encoder and decoder branches in order to preserve spatial and temporal information. This is because spatial and temporal information is lost due to downsampling in the encoder layers, and we need both sets of information to accurately make a proper future prediction about the location of the agent.

4. Experimental Evaluation

In this work, we specifically look at the Nuscenes data set. The Nuscenes data set contains 1000 scenes each being 20 seconds long, 1.4 million camera images coming from one of 6 onboard cameras on the ego vehicle, 390,000 Lidar sweeps, annotations for 23 object classes, and 1.4M bounding boxes annotated at 2Hz. [3] Since we observed that the Nuscenes data set is quite large and will not fit in memory, we decided to only use a subset of the entire data set. Specifically, we generate training and validation sequences from the first 85 scenes in the data set.

Initially, we trained the network on KITTI with a combination of a reconstruction loss and the safety loss. The loss function we used is:

$$\mathcal{L}_{total} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{safe} \tag{3}$$

Where $\mathcal{L}_{rec}$ is the MSE reconstruction loss that penalizes any deviation of the predicted trajectory from the ground truth. $\mathcal{L}_{safe}$ is the safety loss which discourages predicted states of vehicles from lying in an obstacle cell. This can

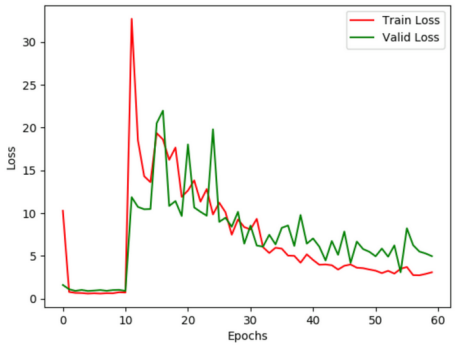


Figure 4. Training and Validation Curves using the Kitti Dataset

be thought of as discouraging the model to collide with any obstacle it sees in its path.

$$\mathcal{L}_{safe} = \left\| \mathcal{O} \odot \hat{\mathcal{F}} \right\| \tag{4}$$

During training, we decided to use $\lambda = 1$ and did not see good results on the KITTI dataset. The loss curves were too high when we used the safety loss, which led to us only using the MSE reconstruction loss.

Our main experiment was to do a comparative study between trajectory prediction using either map-based or LiDAR-based intermediate representations to feed the LSTM model. We generated the same intermediate representations and obtained an output map and trajectory plot for all the scenes we chose for testing. We saw that there was a difference in the type of trajectory and output maps each type of model generated, which will be discussed in the next section.

5. Results and Discussion

We first replicate the training and validation curve results the authors go over, which is shown in Figure 4. We implement the reconstruction loss they use for their model,

which uses a vanilla MSE loss. We decided not to use the safety loss mentioned in their paper because when we tried to generate the training and validation curves, the validation curve never seems to converge, which implies that the usage of this loss is not beneficial for our purposes. It is also important to note that the author's implementation trains the model using just a single frame of a sequence at a time instead of using batches, which led to a significantly larger training time.

Another observation we made when looking into the author's implementation is that when the model is past teacher forcing part of training and past the time of prediction, the authors use the future intermediate representations for obstacle, other vehicles, lane, and road channels while changing the target channel to be the output obtained at the previous time step. We decided to fix these channels in our implementation because in a real world scenario the model will not have access to the future representation.

If we look at Figure 5, we can see all of the input representations generated properly, as well as the inputs in Figure 8. However, we see that the output maps generated from both of these models are very different. We observe that there are a lot of periodic dots in the output generated from the map-based model shown in Figure 9, where as the output generated from the LiDAR-based model has a clear idea of where the the vehicle can potentially be (where the color is the brightest). We can see this reflect in the trajectory, where it seems like the trajectory from the map-based model seems to not go forward, but instead backwards. This can likely be a result of the incorrect output, where it seems like the car has an equal chance of being at each of those points where the dots in the output exist. In our LiDAR-based output as shown in Figure 7, we can see that that brightest dot is on the right side compared to the dot in the target vehicle and has a similar visualization to the ground truth representation as shown in Figure 6. This is a good sign, as we yield a pretty similar future trajectory to the ground truth trajectory.

However, if the ground truth representation does not match the LiDAR-based output very well (lots of noise in the output, brighter points away from the ground truth point), then the trajectory generated becomes hard to predict. It would take us more time to figure out what kind of abnormalities in our generated output yields certain type of trajectory miscalculations.

One of the other phenomena we observed is that the Nuscenes BEV rendering code did not capture the temporal nature of the ego vehicle and surrounding obstacles which led to the creation of our custom BEV rendering code. When taking a deeper inspection at the Nuscenes data set, the reason why the BEV rendering did not capture the overall temporal nature is because the lidar data was transformed from the lidar data frame to the ego vehicle frame first and then transformed into BEV. As a result, the ego vehicle was always in the center of the intermediate representations. By converting from global frame to BEV, we were able to see where the change in the ego pose vehicle along with a better estimate how the obstacles, lane, and road changed over time.

We also are skeptical of the systems performance when deployed in the real world mainly due to the long time it takes to segment the image. The segmentation roughly takes ten seconds per image and is a bottleneck in this entire system. Tens seconds in a real world scenario is an extremely long time to wait in order to do trajectory predictions as the world can change quickly in an instance. In order to reduce the overall time it takes to train the network, the segmentation was done beforehand on all the camera images used in the Nuscenes data set which roughly took around 12 hours to do.

Below, we show our results on both the lidar and the map representations. We train two different models using each type of representation, and test on them using the same splits on the Nuscenes dataset that we use. Below are the inputs, outputs, and trajectory predictions using the lidar-based intermediate representations.

In our studies, we determined that our ego vehicle is moving in a straight line (more or less), so the expected prediction trajectory in most cases should be a straight line. In this specific scene's frame, we can see that the the prediction and multimodal prediction both predict paths relatively well compared to the ground truth. However, our accuracy for future prediction was only 34.5 percent, which can imply there is either something wrong with the model or we did not train on enough scenes or epochs.

Also, in Figure 9 you can see the predicted output of the LSTM model that was trained using map-based intermediate representations. The predicted trajectory looks worse when compared to the predicted trajectory from the model trained using the LiDAR based data. This might be explained from the fact that the loss for the model trained on map-based representation was still high at the end of 30 epochs. See Figure 10 for the loss curve for the model trained using map-based representations. It might be fair to say that the model trained using map-based representations didn't learn much in just 30 epochs when compared to the other model.

5.1. Ablation Studies

While running these experiments, we also experimented with different hyperparameters when creating the data set and seeing its impact on training the model. One of the first hyperparameters that needed to be tuned were the kernel sizes for dilating the intermediate representation. We chose a 5×5 kernel for both the obstacle and road representations while we chose a smaller 2×2 kernel for the lane

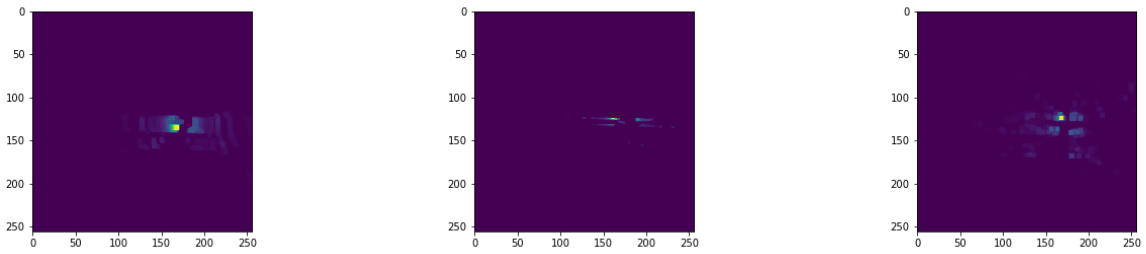


Figure 5. LiDAR Intermediate Representations for the Road, Lane Marking and Obstacle channels (respectively) for Scene 46

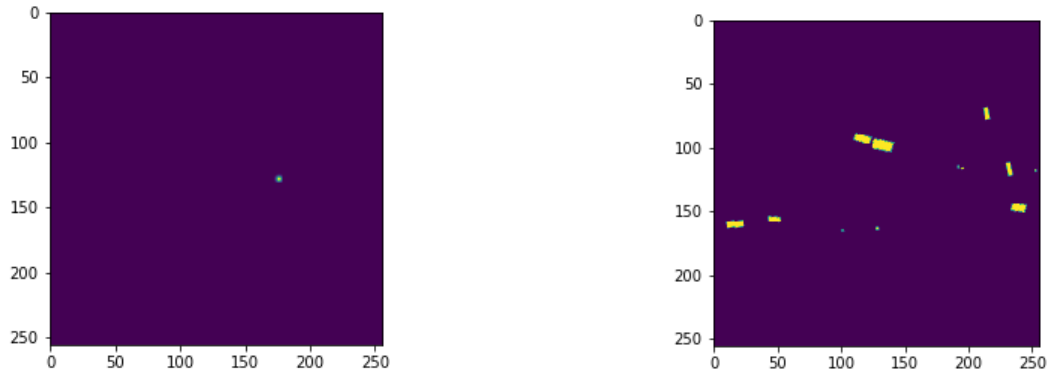


Figure 6. LiDAR Intermediate Representations for the Target and Other Vehicles channels (respectively) for Scene 46

representation. The reason why we chose larger kernels for the road and obstacle specifically was the fact that roads are inherently a connected block-like object. In Figure 2, the lidar points that are projected into the image plane that correspond to road are typically close together. When using smaller kernels, the representation didn't fully capture the block like essence of the roads. Similarly for the obstacle channel, since we include buildings and curbs, we need to capture the connectedness of these objects which we found were best achieved by a 5×5 kernel than any other kernels. We observed that when increasing the kernel size, the representation over-filled these regions and didn't capture the road and obstacles changing over time. For the lane representation, we decided to go with a smaller 2×2 kernel because we wanted to capture the connectedness and also capture the demarcation of the vehicle's current lane versus another lane. When experimenting with larger kernels, specifically a 5×5 kernel, we observed cases where the lanes were combining with one another, which defeats the purpose of having a lane representation because we want to encode the separation between the current lane that the vehicle is on and the other lanes which other vehicles are driving on.

The next important hyperparameter that was experimented on was the range of pixel values that were used for the intermediate representations. We tried training the model using intermediate representations that were either between $0-1$ and $0-255$. We initially decided to train the model using intermediate representations with pixels ranging from $0-1$ since it is a common preprocessing step used in many computer vision and deep learning techniques. When we tried training using these intermediate representations, we observed that the training loss and validation loss diverges and then stagnates after a few epochs for both training with map based and sensor based intermediate representations as seen in Figure 11 and Figure 12. When taking a deeper look at this issue, we discovered that the model suffered from gradient vanishing as the gradients at each layer of the model were either 0 or extremely close to 0. However, when we trained the model with intermediate representations with pixel values ranging from $0-255$, we observed the model did not suffer from gradient vanishing as seen in Figure 12.

6. Conclusion

In this study, we used the INFER model described in [10] to do trajectory prediction on the Nuscenes data set. We

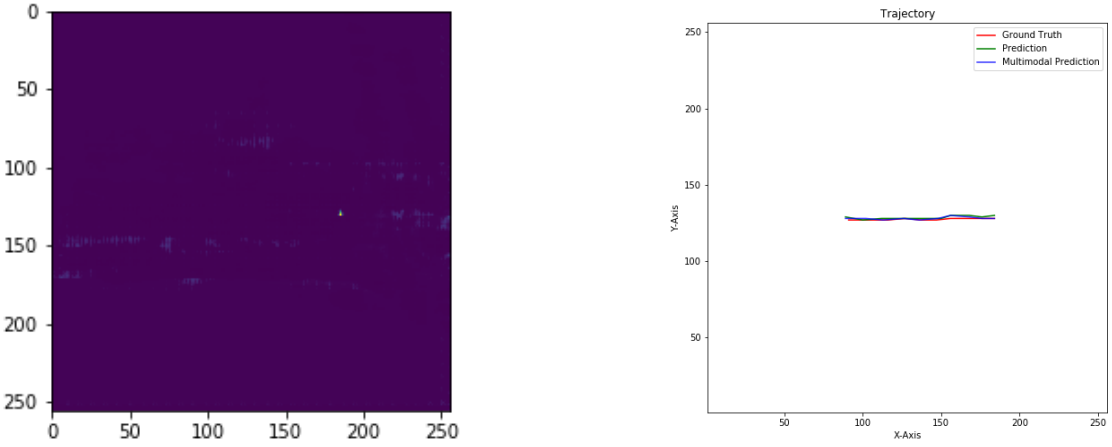


Figure 7. Output of LSTM and Trajectory Prediction (Regular and Multimodal) vs. Ground Truth for Scene 46 using LiDAR Intermediate Representations

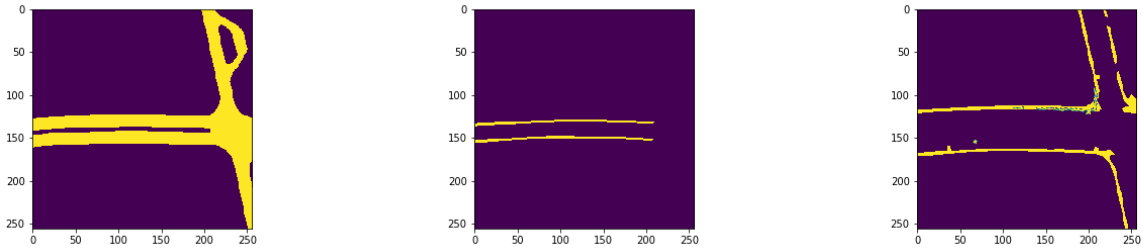


Figure 8. Intermediate Representations (map-based) for the Road, Lane Marking and Obstacle channels (respectively) for Scene 46

generated a map based intermediate representation and a sensor based intermediate representations for the nuScenes dataset from scratch. We also implemented custom BEV transformations in order to capture the temporal nature of trajectory prediction. We avoided doing instance segmentation and, instead, used the bounding boxes provided in the data set which allowed us to encompass the LiDAR points relating to these different instances of objects using bounding boxes. We discovered some dubious implementation choices by the authors, specifically training with the batch size being 1 and using future intermediate representations at time of prediction. While we see that there are some pitfalls in this method, we also see many opportunities to improve it. Specifically, we can experiment with different semantic segmentation networks that can significantly cut down the cost of segmentation using the current network. Furthermore, the training procedure and data loaders can be optimized to do batch-wise prediction rather than a serial frame-by-frame prediction. Lastly, instead of using a LSTM based approach, we can try experimenting with network architectures to use Transformers rather than LSTM which will allow for more parallelization and significant speedup.

References

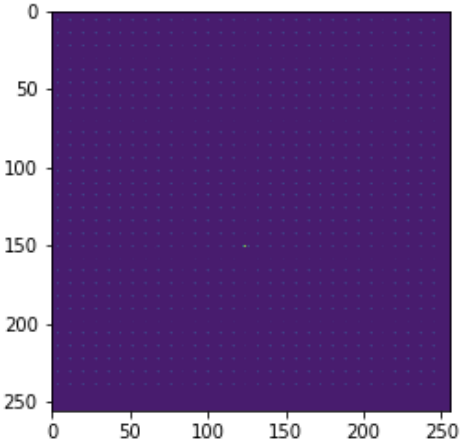
[1] A. Barth and U. Franke. Estimating the driving state of oncoming vehicles from a moving platform using stereo vision. *IEEE Transactions on Intelligent Transportation Systems*, 10(4):560–571, 2009.

[2] S. R. Bulò, L. Porzi, and P. Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. *CoRR*, abs/1712.02616, 2017.

[3] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CoRR*, abs/1903.11027, 2019.

[4] R. Danescu, F. Oniga, and S. Nedevschi. Modeling and tracking the driving environment with a particle-based occupancy grid. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1331–1342, 2011.

[5] N. Deo and M. M. Trivedi. Convolutional social pooling for vehicle trajectory prediction. In *Proceedings of the IEEE*



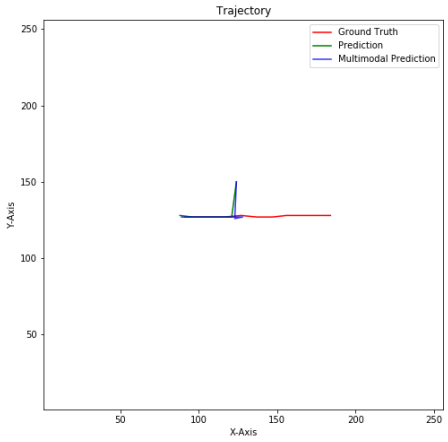


Figure 9. Output of LSTM and Trajectory Prediction (Regular and Multimodal) vs. Ground Truth for Scene 46 (using map-based representations)

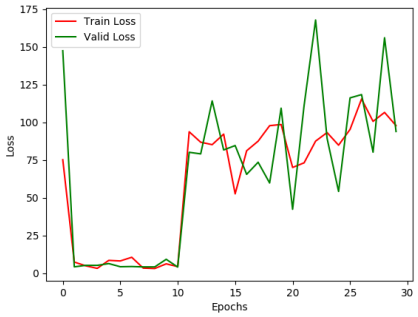


Figure 10. Loss Curve for the model trained using map-based intermediate representations with pixels in range 0-255.

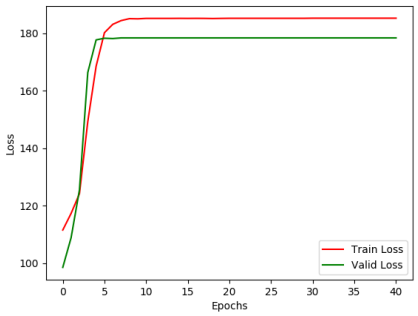


Figure 11. Training and validation loss curves after training with $0-1$ pixel valued map-based intermediate representations.

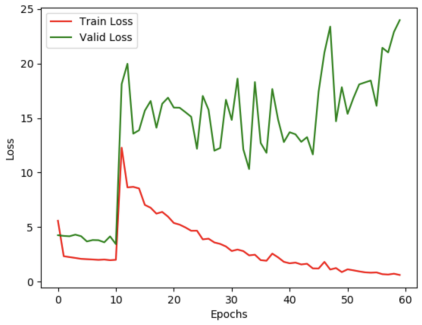


Figure 12. Training and validation loss curves after training with $0-1$ pixel valued LiDAR-based intermediate representations.

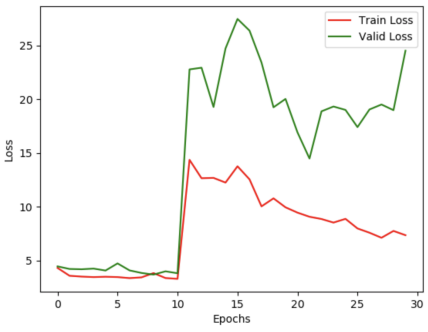


Figure 13. Training and validation loss curves after training with $0-255$ pixel valued LiDAR-based intermediate representations.

Conference on Computer Vision and Pattern Recognition Workshops, pages 1468–1476, 2018.

[6] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE Confer-*

ence on Computer Vision and Pattern Recognition, pages 2255–2264, 2018.

[7] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2017.

[8] G. Neuhold, T. Ollmann, S. R. Bulò, and P. Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5000–5009, 2017.

[9] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff. Covernet: Multimodal behavior prediction using trajectory sets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14074–14083, 2020.

[10] S. Srikanth, J. A. Ansari, S. Sharma, et al. Infer: Intermediate representations for future prediction. *arXiv preprint arXiv:1903.10641*, 2019.

[11] J. Virdi. *Using deep learning to predict obstacle trajectories for collision avoidance in autonomous vehicles*. PhD thesis, UC San Diego, 2017.

[12] S. Vora, A. H. Lang, B. Helou, and O. Beijbom. Pointpainting: Sequential fusion for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4604–4612, 2020.