**Resnet-GRU model**

Our first model was an attempt to create a baseline for the problem at hand. The model was based on a pre-trained Resnet\_34 network which was extended using a bi-directional GRU.

The input dataset consisted of input images, target positions of the agent vehicle as well as the ego vehicle. It also included target availability for the agent and ego vehicle. The input also provided us with historical positions of both sets of vehicles. For our model we used the input image of the agent as input and trained it on its given target positions. We optimized the model by calculating losses on target availability. The input number of channels was given by the number of historical positions/frames we wanted to consider.

For our scenario we considered 10 historical frames which gave us the number of input channels as 22 because we considered 10 for each agent and ego vehicle and also included the current frame for both of them.

The Resnet network was fed input images of the agent vehicle frame and the number of input channels were 22 as described above. The output from the resnet was of shape (bathc\_size, 512). For our experiments we considered batch\_size to be 16.

The output from the Resnet was further fed to the bi-directional GRU. The hidden size for the network was chosen to be 2048. The gru layer gave output with shape ((batch\_size, 1, 512), 2\*hidden\_size). The output features from this layer of GRU was then further fed to a sequential network consisting of a linear layer followed by a RElu and another linear layer and generated output with 1024 features. The hidden state from the previous GRU layer and the output features from the sequential network was fed to another bi-directional GRU layer and a similar cycle was followed.

The GRU layer returned an output with 1024 features which was fed to a linear network to finally get the desired target output features.

The total number of targets considered of the number of future positions we wanted to predict \* the number of trajectories we wished to predict. In our scenario we decided to predict 50 future positions for 3 trajectories as kaggle was evaluating the results with the same. We also chose to predict confidences for 3 trajectories given the high uncertainty of real time traffic. Each future position had X and Y coordinate values, hence total number of target became number\_of\_trajectories \* num\_of\_future\_positons \* 2 + confidences\_for\_3\_trajectories i.e (3\*50\*2 + 3 = 303).

Once we had the desired output, we split it into future positions (300) and confidences (3). The confidences were further transformed to probabilities using softmax. The final predicted outputs had shape (batch\_size,num\_of\_trajectories,50,2) and confidences had shape (batch\_size, num\_of\_trajectories). The model and output generation process described above is represented diagrammatically below.

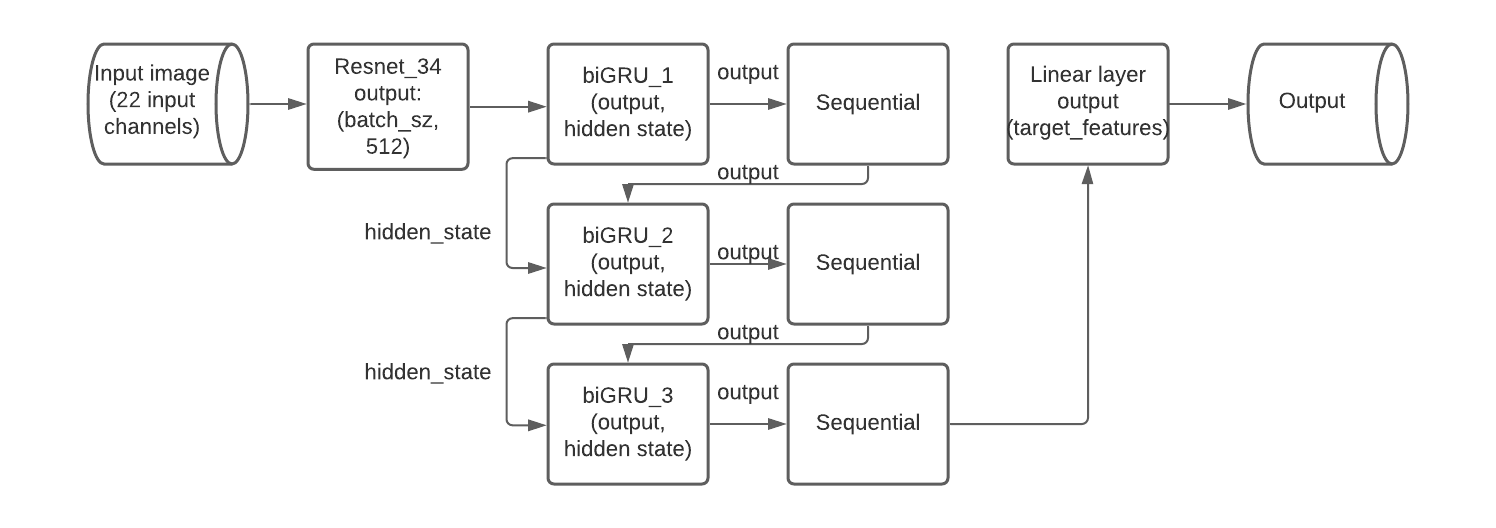


Fig 1. Model journey

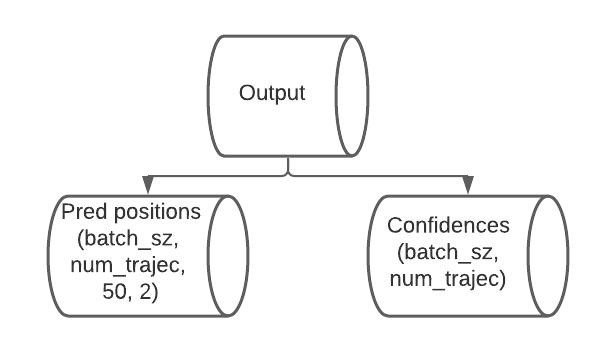


Fig 2. Output generation

For our experiments we executed our model multiple times on the training data to improve its performance. One of the critical things we ensured in this methodology was to use the trained model and latest optimizer state for the next cycle of training. Each cycle of training consisted of 12000 iterations. We generated our predictions from test data after each such cycle to observe the models performance. The predictions were evaluated by Kaggle using a negative log likelihood function. Below are the details of the three rounds of experiments conducted by us.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cycle no. | Experiment details | Loss function | Learning rate | Iterations | Average training Loss | Average test loss |
| 1 | Model is trained for the first 12000 iterations | Negative log Likelihood | 1e-4 | 12000 | 129.41 | 143.07 |
| 2 | Model state and optimizer state from cycle 1 is taken as the starting state | Negative log Likelihood | 1e-3 | 12000 | 59.62 | 76.86 |
| 3 | Model state and optimizer state from cycle 2 is taken as the starting state | Negative log Likelihood | 1e-4 | 12000 | 42.98 | 63.62 |

Table 1. Experiment details using negative log likelihood loss function

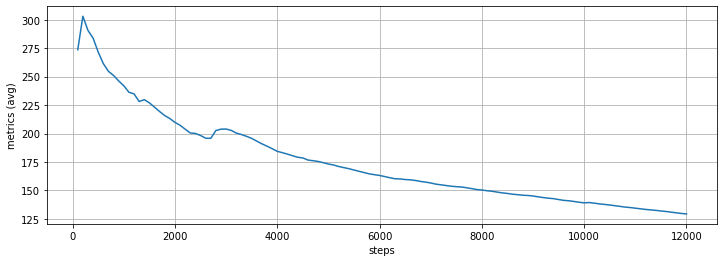


Fig 3. Cycle 1 average training loss curve

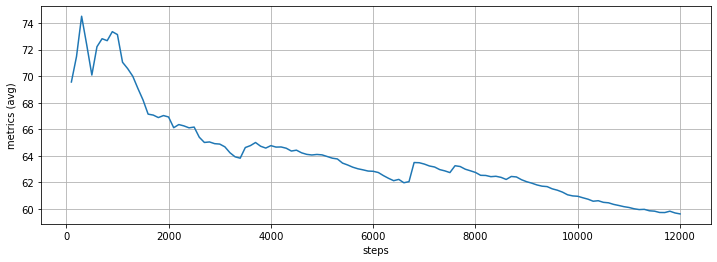


Fig 4. Cycle 2 average training loss curve

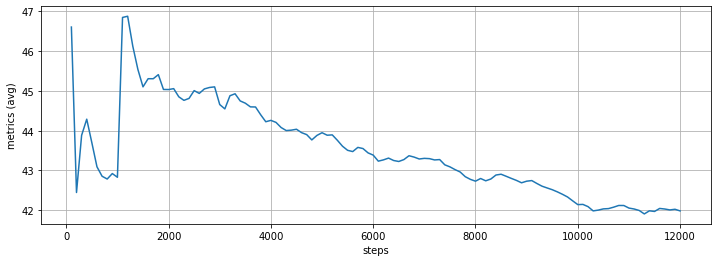


Fig 5. Cycle 3 average training loss curve

Prediction loss on test dataset evaluated by Kaggle:

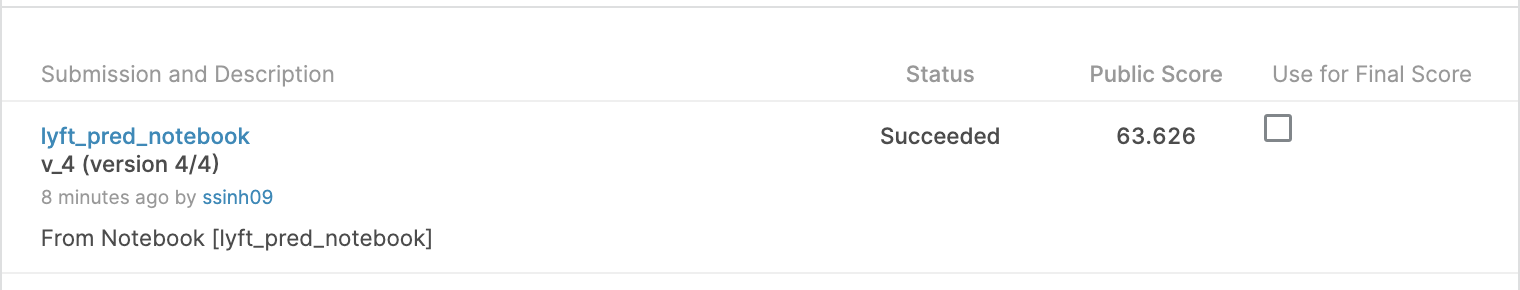


Fig 6. Best test result from kaggle submission

We also tried to train our model using a different loss function to compare the model’s performance

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cycle no. | Experiment details | Loss function | Learning rate | Iterations | Average training Loss | Average test loss  (neg log) |
| 1 | Model is trained with only 1 mode | MSE | 1e-4 | 12000 | 5.35  (5.4 val loss) | 7131.07 |
| 2 | Model is trained with 3 modes | rMSE | 1e-4 | 12000 | 1.53 | 193.90 |

Table 2. Experiment details using MSE and rMSE loss function

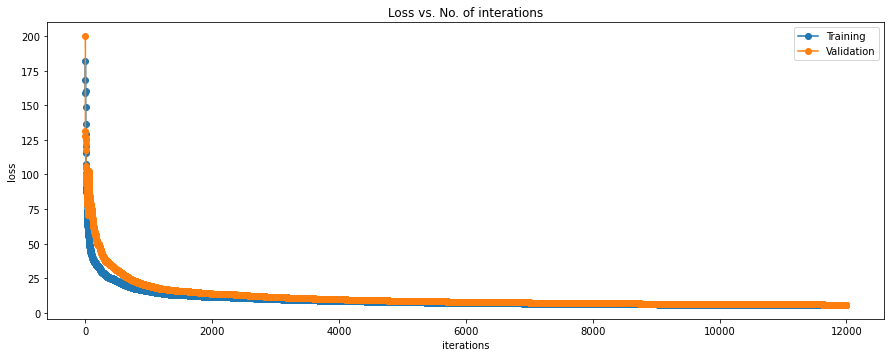


Fig 7. Training and Validation loss for MSE

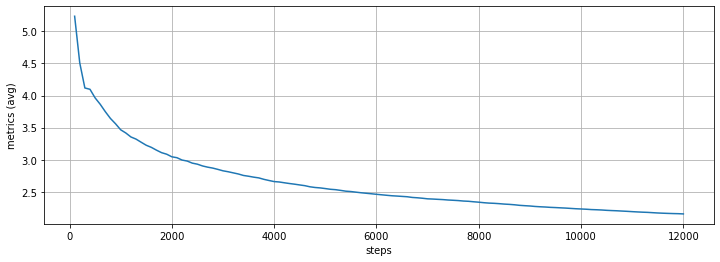
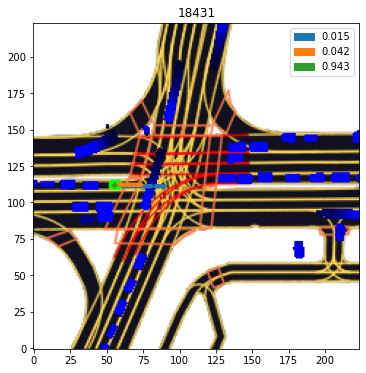
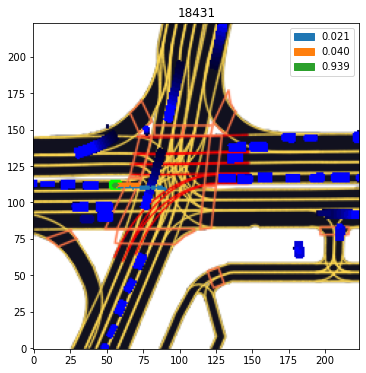
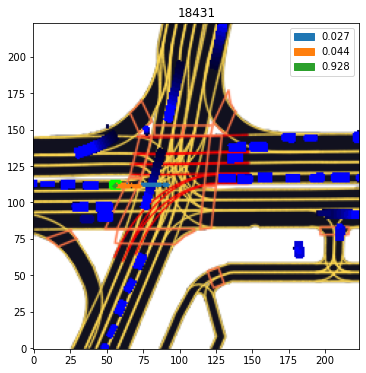


Fig 8. Training loss for rMSE

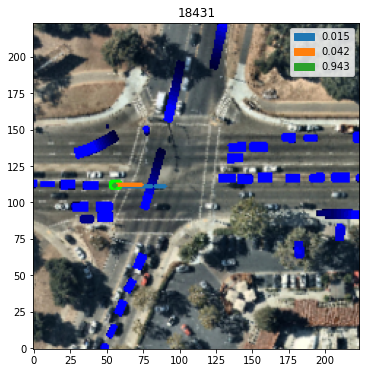
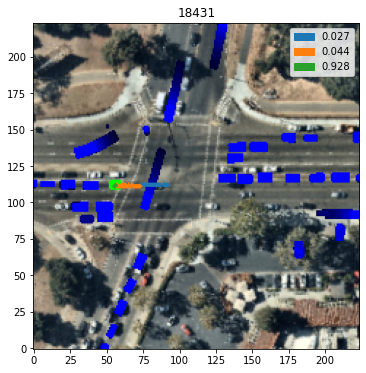
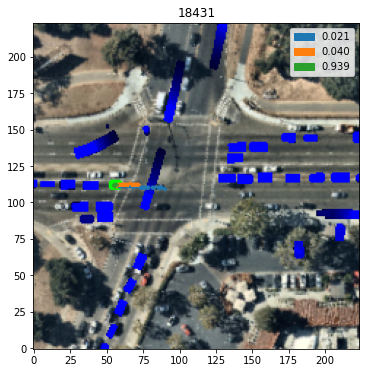
**Prediction trajectory visualization**

Every agent is identified by its track id and timestamp. In our submission file we have every record with timestamp and track id as index. For comparing the results generated by each of our models, we have compared trajectories predicted for the same agent. For my prediction I have used 3 modes or 3 prediction trajectories. Each trajectory holds 50 two dimensional (X,Y) predictions and each trajectory will have its own confidence. The sum of all three confidences will sum up to 1.

In the given visualization, we compared predicted trajectories for an agent with track id “18431”. We can observe that trajectory 3 represented by color green is dominant and as the model improves it reaffirms trajectory 3 with the highest confidence of 0.943. The visualizations are represented in both semantic and satellite view.



Model after cycle 1 Model after cycle 2 Model after cycle 3



Model after cycle 1 Model after cycle 2 Model after cycle 3