EE 599 Deep Learning - Surrounding Motion Predict Model for Autonomous Vehicle

Wenhao Cui, Guangrui Shen, Tieming Sun

Viterbi School of Engineering, University of Southern California

December 9, 2020

Abstract

An autonomous vehicle (AV) is able to percept and predict surrounding traffic agent's motion, such as cars, pedestrian and cyclist, as reliable as possible to accurately plan its future route. In this project, we train a CNN models to predict surrounding agent's motion over 5 seconds given a historical 1 second bird-eyes-view images with 10 frames and 25 channels. To further improve model's inferencing speed, and predict performance, we choose a novel architecture, pretrained Mixnet. Also, we apply 3 loss functions in different training stages, with the purpose of accelerating model's convergence. For multi trajectories evaluation metric, Negative Log Likelihood (NLL) is used, and the ensemble model performs best at 31.4 when we add a k-means algorithm layer in testing.

Keywords: Convolutional Neural Network, Mixnet, Motion Prediction

I. Introduction

1.1 Background

In modern life, many traffic accidents are caused by human negligence. Let vehicles drive themselves is a good way to reduce traffic accidents. Therefore, Autonomous vehicles is one interesting topic in machine learning and deep learning field. An autonomous vehicle is one that is able to operate itself and perform necessary functions without any human intervention, through ability to sense its surroundings. Autonomous vehicles are expected to dramatically redefine the future of transportation. We believe in a future where self-driving cars make transportation safer, environment-friendly and more accessible for everyone. However, there are still significant engineering challenges to be solved before one can fully realize the benefits of self-driving cars [1]. One such challenge is building models that reliably predict the movement of traffic agents around the autonomous vehicle, such as cars, cyclists, and pedestrians.

In this project, our task is to predict the motion of these traffic agents. We provide the circumstance around the autonomous vehicle in future 5 seconds in order to help it make the next decision.

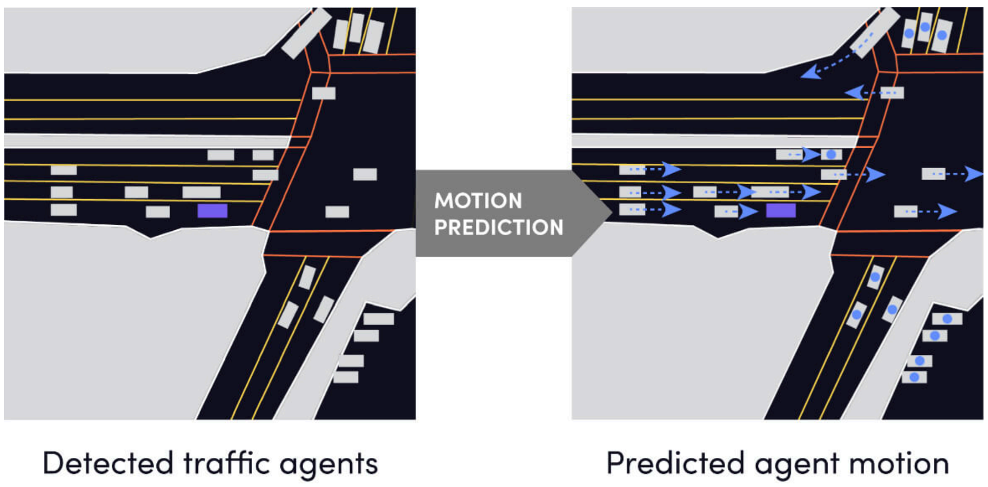


Figure 1. Prediction for the surrounding agents

1.2 Dataset

The dataset [2] is the largest collection of the traffic agent motion data. This dataset includes the logs of movements of cars, cyclists, pedestrians, and other traffic agents encountered by Lyft's autonomous fleet. These logs come from processing raw lidar, camera, and radar data through our team's perception systems and are ideal for training motion prediction models. The dataset includes 1000+ hours of traffic agent movement, 16k miles of data from 23 vehicle and15k semantic map annotations. The dataset consists of 170,000 scenes capturing the environment around the autonomous vehicle. Each scene encodes the state of the vehicle's surroundings at a given point in time.

II. Problem

In this section, we will talk about the problem we firstly encountered and tried to deal with. In the section 2.1, we firstly introduce the l5kit library, which is the core library that helps us to complete the project. Then we will introduce how we process the dataset using this library. In the section 2.2, we will list the models that we use. Then introduce the parameters that we set into the models.

2.1Data Pipeline

One of the problem is that we need to deal with this quite large dataset and learn to use l5kit library. L5kit is a python software library that can visualize and preprocess our dataset, train and evaluate our models [1]. To be more specific, with l5kit library, we can:

- Load driving scenes from .zarr files
- Read semantic maps
- Create birds-eye-view (BEV) images to represent a scene around an AV (Autonomous Vehicle) or another agent vehicle
- Sample data

- Train neural networks
- Visualize results

We extract the dataset from .zarr files, which has helped us split the dataset into training, validate and test set. In our project, we use training set to train our models and validate set to evaluate the models performance. The dataset contains 4 arrays, that is agents, frames, scenes, traffic light faces. L5kit contains several dataset package that already implements pytorch ready dataset. There are two kinds of dataset could be used: EgoDataset and AgentDataset. EgoDataset iterates over the AV annotations. AgentDataset iterates other agents annotations. Both of them can be iterated and return multi-channel images and future trajectories offsets from the rasterizer. In this project, our task is to predict surrounding agents motions, so we only used AgentDataset to train and evaluate our models. Other dataset package that we used is ChunkedDataset, which could help to make zarr dataset object.

Also, before we trained our models, we need to visualize the dataset. l5kit library provides two core packages for visualization. One is rasterization, another is visualization. Rasterization package contains classes that can get visual data as multi-channel tensors and turning them into interpretable RGB images. Visualization package contains utilities to draw additional information onto RGB images. For example, we can draw each agents or ego's trajectories using this package.

2.2 Model Overview

Another problem that we need to overcome is model selection. We use ResNet models and MobileNet models as our baseline models. ResNet is the model in which layers are in residual connections. As we can see in the figure shown below. The idea is used to reduce the vanishing gradient problem. If the gradient of F(x) which goes through two weight layers are near to zero, we still have gradient because of another output x that by-pass the weight layers.

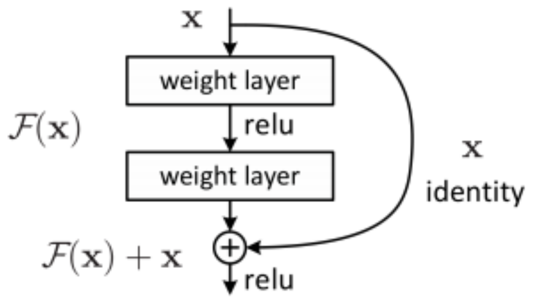


Figure 2. Residual connection

In this project, we tried the pretrained ResNet34. The architecture of ResNet34 is as follows.

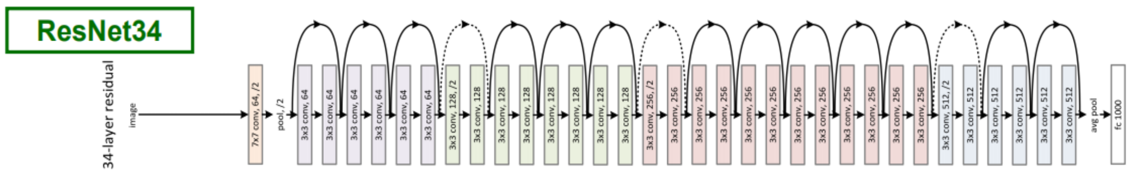


Figure 3. ResNet34 Architecture

Also, we used MobileNet model to see its performance. The idea of MobileNet model is to combine depth-wise convolution with many point-wise convolution, which can dramatically reduce the parameters utilization. So the MobileNet models' predicted speed is relatively faster than ResNet and meanwhile matches ResNet models prediction capability. The following figure shows the MobileNet-v2 structure:

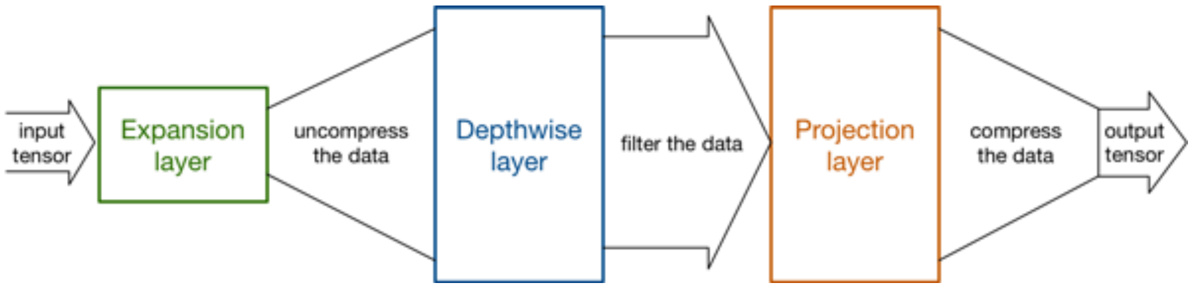


Figure 4. MobileNet-v2 Architecture

Considering our goal is to predict the surrounding agents motions of the autonomous vehicle over a 5-second-horizon given their 1 second historical and current positions. So we need 10 historical frames and 1 current frame, in which the distance of each two continuing frames are 0.1 second. Each frame contains ego car and agents that are on different channel. Also, we need to consider the R, G, B channels for semantic map into our input channels. In other words, given the current time t, the input channels comprise agent(t), agent(t - 1), agent(t - 2), ..., agent(t - 10), ego(t), ego(t - 1), ego(t - 2), ..., ego(t - 10), semantic map R, G, B. Thus, the total number of input channels we set is:

$$\textit{The number of input channels} = 2 \times (10+1) + 3 = 25$$

Because we want to output the next 5 second agents motions. We need 50 frames, which could be represented as 50 coordinates in two axes. Thus, the output size we set is:

$$\textit{output size} = 50 \times 2 = 100$$

We tried several models to compare their evaluate performance, finally we use MixNet(s) as our selected models. The detailed of MixNet(s) will be shown in the next section.

III. Mixnet

In this section, we will introduce the MixNet structure, which is in section 3.1. Also, we will discuss the advantages of MixNet in section 3.2.

3.1 Structure

The MixNet models are automatically built using Neural architecture search by replacing single convolutional kernels into a group of kernels ranging from 3x3 to 9x9.

The intuition is that the smaller kernels (3x3, 5x5) serve to capture lower resolution details while the larger kernels (7x7, 9x9) capture higher resolution patterns and thus ultimately build a more efficient network. Therefore, if we blend multiple kernels sizes into a single layer, then a single MixConv layer is built. The figure below shows the comparison between the vanilla depth-wise convolutional layer and MixConv layer.

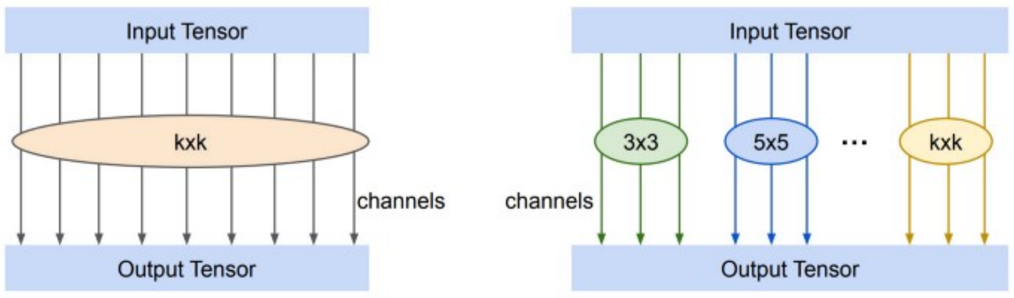


Figure 5. Comparison between vanilla depth-wise convolution and MixConv layer

MixNet is ultimately using a range of 1 to 5 kernels, which was considered to be the best standard. By leveraging Neural architecture search, the final MixNets models were built [3].

The following figure shows the MixNet-m's architecture:

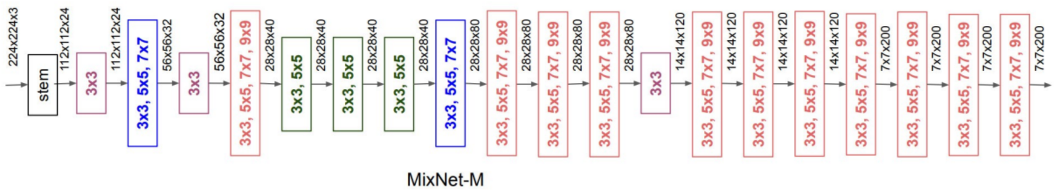


Figure 6. MixNet-m architecture

3.2 Advantages

When the MixNets models were built, it was mostly compared with the ResNet153's performance. Experiments show that MixNet-l outperforms ResNet153 with 8x fewer parameters. MixNet-m matches it exactly but with 12x fewer parameters and 31x fewer flops [4]. Thus, because we want to compromise the models' speed and the prediction capability and MixNets models perform very well, we finally focus on MixNets models as our selected models. In this project, each of our three member tried 3 different MixNets Models, which is MixNet-m, MixNet-l and MixNet-xl, where m, l, xl are the degrees of MixNets layers depth. For instance, MixNet-l is simply a 1.3 depth multiplier of MixNet-m.

IV. Evaluation

4.1 Implementation

In this section, we have trained our model on Lyft Level 5- Prediction Dataset with several Data entities. We use pretrained Resnet 34 as backbone model, then make a transition to a novel architecture Mixnet. Pretrained model is used and conducted by Timm package, then the input layer and output layer are modified to fit for the data.

During the implementation steps, pytorch (version 1.6) is used for the coding part, under Jupyter notebook provided by Kaggle. CUDA (GPU accelerator by NVidia) API is called for faster inference, backpropagation and testing. Also, Kaggle provides free access to

Nvidia P100 GPUs in kernels, enabling approximately 12.5X faster training than a deep learning model[5]. Every member in the team are responsible for training a CNN deep learning model, in order to ensembl in the later steps. So totally GPU quota time is over 210 hours in the project.

Python library provided by Lyft offers Agent Dataset, inherited from Pytorch Dataset. The most important entity we used from Agent Dataset is called “image”, a 25-channel enhanced image compared with the normal RGB one. The image can pack up 1 second’s history environment by 10 frames of a single agent, 10 frames of the ego (self-driving car), and background like the semantic map and traffic light. Both of Resnet and Mixnet conceive an RGB picture as the input, so we change the input convolutional layer to accept 25-channel image.

In terms of the output layer, we add a fully connected layer that maps the output vector into a size of 100 dimensions vector, representing the X and Y coordinate of the future 50 frames.

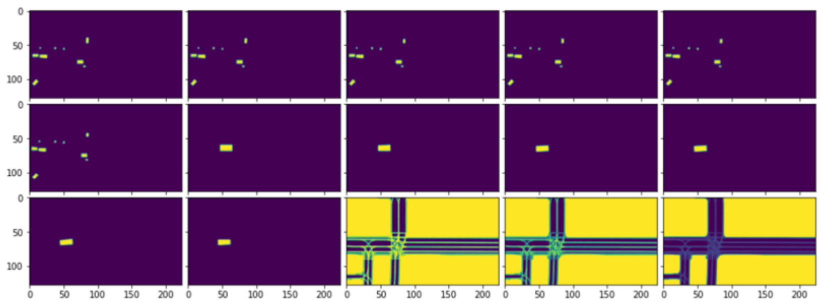


Figure 7. unpack the image from Agent Dataset

4.2 Parameter Tuning

During the training period, a lot of parameters should be taken into consideration in order to let the model achieve its best performance. A number of experiments are conducted, and we carefully select loss function, parameter from dataset and the hyperparameter for deep learning model.

4.2.1 Loss Function

In this project we use 3 loss functions: Smooth L1, Negative Log Likelihood (NLL) and customed NLL in different training stages.

Smooth L1 is often used for regression on some object detection problems [6] when models are not easily to converge and there exists relatively a big number of features. This loss function is a lot less sensitive to outliers than other regression loss, like L2 loss. The loss function is

$$loss(x,y) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} 0.5\times(y_i - f(x_i))^2, & if\ |y_i - f(x_i)| < 1 \\ |y_i - f(x_i)| - 0.5, & otherwise \end{cases}$$

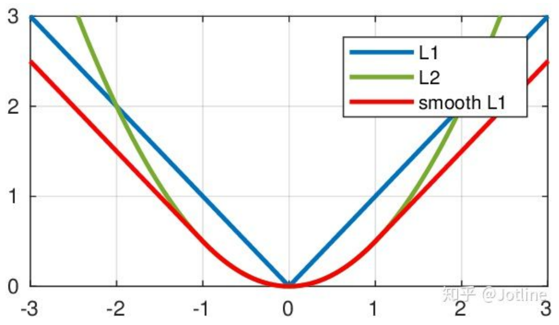


Figure 8. Comparison between L1, L2 and Smooth L1 loss function

Another advantage is that this loss function smooths the gradient when the difference between prediction and ground truth are quite close.

Negative Log likelihood loss function is often used in motion predicting problems, a special case of setting $c = [1,0,0]$ for multi-modal evaluate metric, will be fully introduced in section 4.3.1. This loss function is used after training in Smooth L1 for 10000 iterations. As seen in the following figure, shifting from Smooth L1 to NLL will give a sharp decrease in loss, shown in the figure.

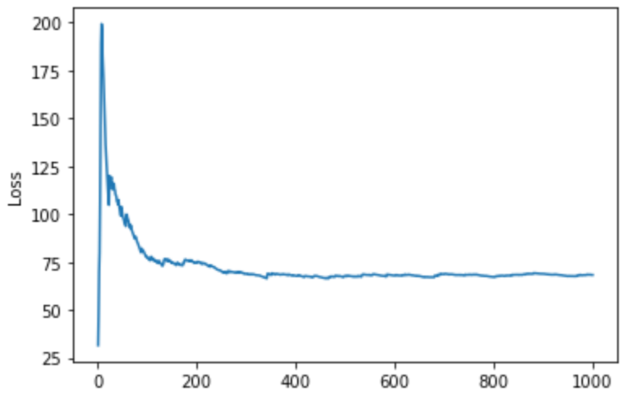


Figure 9. Loss performance when switching from Smooth L1 loss to NLL loss, run in 1000 iterations

Another loss function used in training is customed NLL, which is a derivation from NLL. Customed NLL is designed because we want to accelerate the model training since we have limited GPU quota. During validation stage, some corner cases where car are pulling over, making a turn or U-turn have bad performance. In order to let the model focusing more on these cases, we want to penalize more on them. Custom NLL will get a high loss when the deviation angle is large shown by the ground truth in the future 50 frames.

$$\begin{cases} NLL \times penalize \times arccos\left(\frac{|dx|}{\sqrt{dx^2 + dy^2}}\right) & arccos\left(\frac{|dx|}{\sqrt{dx^2 + dy^2}}\right) \geq 0.5 \\ NLL \times penalize \times 0.5 & arccos\left(\frac{|dx|}{\sqrt{dx^2 + dy^2}}\right) < 0.5 \end{cases}$$

where dx and dy means the X and Y coordinates difference between the first and the last frames and penalize by default is set as 1.25. If deviation angle is roughly smaller than 30 degree, we set the arc cosine function as 0.5.

However, during the training, we find custom NLL decreasing the loss quicker than the original NLL at the first 200 iterations. However, by experimenting 1000 iterations, NLL's loss is better than the custom NLL function's in validation. That is a main reason why we finally drop this idea. For longer iterations, we still believe NLL is safer and good enough for our project.

4.2.2 Dataset Parameter

Lyft L5 kit gives a lot freedom for using dataset, ranging from rasterization to detecting traffic lights. Here is a table for this project's configuration.

Table 1. Configurations for dataset parameters

Name	**Configuration**
Raster Size	[300,300]
Pixel Size	[0.5,0.5]
Ego Center	[0.25,0.5]
Filter Agent Threshold	0.5
Delta Time	0.1
Disable Traffic Light Faces	False

4.2.3 Hyperparameter

More than 15 models are trained for this project, aiming to find a best hyperparameter. Finally, we choose Mixnet-xl based model as our best one. The following table shows hyperparameters.

Table 2. Hyperparameters for best Model: Mixnet – xl

Name	**Configuration**
Batch Size	32
Iterations	43500
Learning Rate	1e-4
L2 Regularization	1e-6
Optimizer	Adam
Loss Function	Smooth L1 & NLL

4.3 Ensemble Model

4.3.1 Multi-modal Evaluate Metric

As a very commonly used evaluation metric in motion prediction and considering big ambiguity in real world road environment, a negative-log-likelihood of the ground truth data given these is used to evaluate one or multiple model's performance. [7]

Assume the ground truth positions and 3 predicted positions represented by their means are

$$x_1, x_2, \ldots, x_{50}, y_1, y_2, \ldots, y_{50}$$

$$\bar{x}_1^{3}, \bar{x}_2^{3}, \ldots, \bar{x}_{50}^{3}, \bar{y}_1^{3}, \bar{y}_2^{3}, \ldots, \bar{y}_{50}^{3}$$

Along with these positions, we predict confidences c with 3 hypothesizes.

Base on the assumption that the ground truth is modeled a mixture of multi-dimensional independent Gaussian distribution over time, the likelihood is yielded as

$$
\begin{aligned}
&P(x_{1,\ldots,50}, y_{1,\ldots,50} \mid c^{1,\ldots,50}, \bar{x}^{1,2,3}_{1,\ldots 50}, \bar{y}^{1,2,3}_{1,\ldots,50}) \\
&= \sum_{k=1}^{3} c^k N(x_{1,\ldots,50} \mid \bar{x}^{k}_{1,\ldots,50,} \Sigma = 1) N(y_{1,\ldots,50} \mid \bar{y}^{k}_{1,\ldots,50,} \Sigma = 1) \\
&= \sum_{k=1}^{3} c^k \prod_{t=1}^{50} N(x_t \mid \bar{x}^{k}_t, \sigma = 1) N(y_t \mid \bar{y}^{k}_t, \sigma = 1)
\end{aligned}
$$

Apply a log negative calculation on the possibility p

$$
\begin{aligned}
NLL &= -\log P(x_{1,\ldots,50}, y_{1,\ldots,50} \mid c^{1,2,3}, \bar{x}^{1,2,3}_{1,\ldots,50}, \bar{y}^{1,2,3}_{1,\ldots,50}) \\
&= -\log \sum_{k=1}^{3} e^{\log(c^k)} - \frac{1}{2} \sum_{t=1}^{50} ((\bar{x}^{k}_t - x_t)^2 + (\bar{y}^{k}_t - y_t)^2)
\end{aligned}
$$

4.3.2 K-Means

The other problem we should overcome is that we have trained a number of models, each can predict one trajectory, but to compare the performance of our model with the leaderboard from Kaggle[8], we should reduce to 3 trajectories. The intuition of K-Means algorithm is that the centroid of clusters will be more closed to the ground truth coordinates. Given a set of predictions $(a_1, a_2, \ldots, a_n)$, each of the prediction is a 100-dimensional vector, k-means clustering aims to partition them into 3 clusters $S = (S_1, S_2, S_3)$, then the objective is to find

$$
argmin \sum_{i=1}^{3} \sum_{x \epsilon S_i} ||x - \mu_i||^2
$$

Where μ_i is the mean of points in S_i.

Theoretically the more models for prediction is used, the better performance our model is. However, in reality, not so many well-performed models are trained, so 10 models in different architectures (4 from mixnet-l, 2 from mixnet-xl, 1 from mixnet-m and 3 from resnet 34) are used to get 3 clustered trajectories. The NLL metric for k-means algorithm is 31.4 while the vanilla ensembled model with confidence (1 from mixnet-l, 1 from mixnet-xl and 1 from mixnet-m) has 34.9 in validation metric.

4.4Results

The best single model we trained is in mixnet-xl architecture, with batch size 32, and 43500 iterations. Learning rate, with the purpose of fitting with batch size, is set as 10^{-4}. To avoid overfitting, L2 regularization is set as 10^{-6}. Adam optimizer is chosen because it has a better performance. Smooth L1 loss function is firstly picked because it has similar objectives to NLL function, and it is not so sensitive to outliers. Then we swift to negative-log-likelihood to get better performance. Table 3 shows other model we use and their performance.

Table 3. Performance comparison for some models

Model	**No. of trajectories**	**Iterations**	**Train Evaluation**	**Validate Evaluation**
Mixnet - m	1	35000	55.6	61.4
Mixnet - l	1	50000	52.9	53.0
Mixnet - xl	1	43500	48.3	51.8
Ensemble confidences	3			34.9
Ensemble Kmeans	3			31.4

V. Conclusion and future work

In this project we implement a model that will predict the motion of surrounding vehicles, using CNN based architecture, and ensemble multiple models for better prediction. We develop a data pipeline to create birds-eye-view (BEV) images. To better deal with Lyft's large prediction dataset containing more than 170,000 scenes, Mixnet is chosen because it performs great on both model speed and predict performance. We tried to run models on a large validation dataset to calculate the performance score. For each model, we apply a negative multi-log-likelihood evaluation metric to compare the prediction and ground truth. An effective K-Means algorithm is used on combining 10 models into 3 best models.

With regarding to the future work, different well-performed methods are proved to work in similar multi-modal trajectories predict tasks. A graph based generative adversarial network (GAN) is used to model the social interaction of pedestrians in a scene[9], and we will shift their work to predict the interaction of vehicle. Another possible way to improve our performance is to add recurrent layer[10], which will encode intermediate output in our current CNN layer with history coordinate provided by our dataset, then decoder will generate a sequence-to-sequence trajectories with simple confidences.

Reference:

[1] Mennie J. An Examination of Autonomous Vehicle Technology[J]. Muma Business Review, 2019, 3(17): 195-205.

[2] Kesten R., Usman M., Houston J., Pandya T., Nadhamuni K. Lyft Level 5 AV Dataset 2019: Dataset. [(accessed on 10 April 2020)]; Available online: https://level5.lyft.com/dataset/

[3] M Tan, QV Le. MixConv: Mixed Depthwise Convolutional Kernels. arXiv preprint arXiv:1907.09595, 2019.
https://arxiv.org/pdf/1907.09595.pdf

[4] Meet MixNet: Google Brain's new State of the Art Mobile AI architecture.
https://medium.com/@lessw/meet-mixnet-google-brains-new-state-of-the-art-mobile-ai-architecture-bd6c37abfa3a

[5] Running Kaggle Kernels with a GPU
https://www.kaggle.com/dansbecker/running-kaggle-kernels-with-a-gpu

[6] L7: Distances
https://www.cs.utah.edu/~jeffp/teaching/cs5955/L7-Distances.pdf

[7] L5kit scoring
https://github.com/lyft/l5kit/blob/master/competition.md

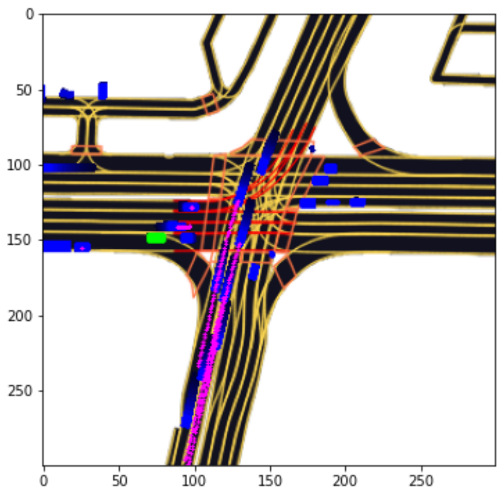
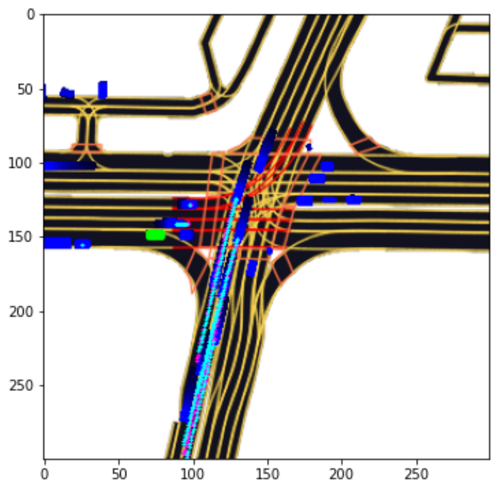
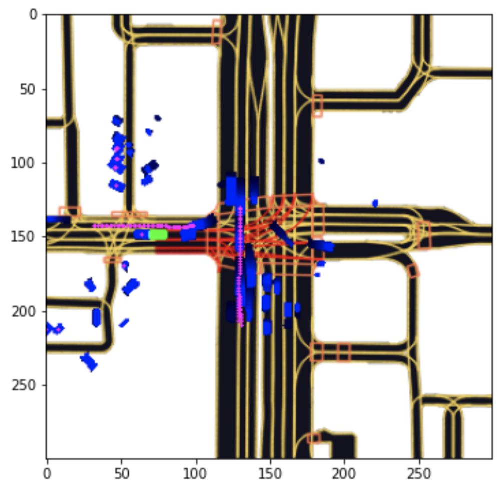
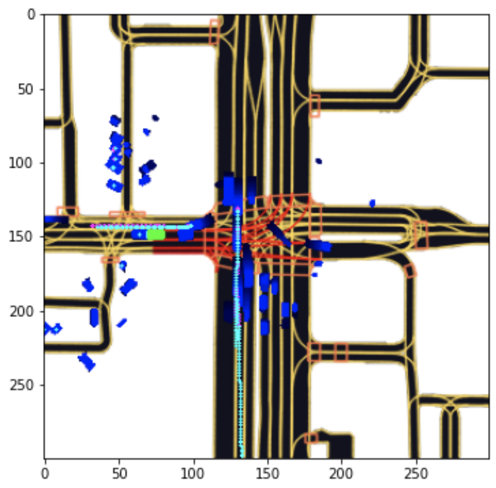
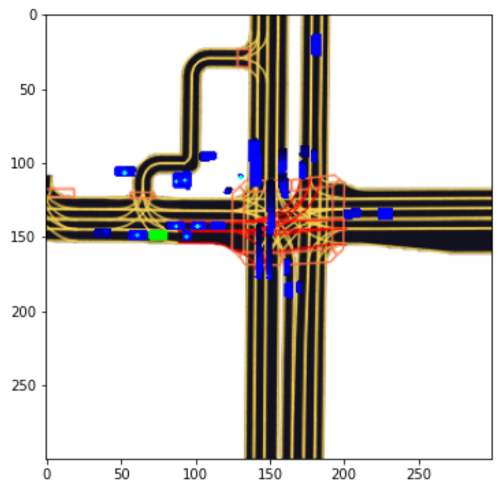
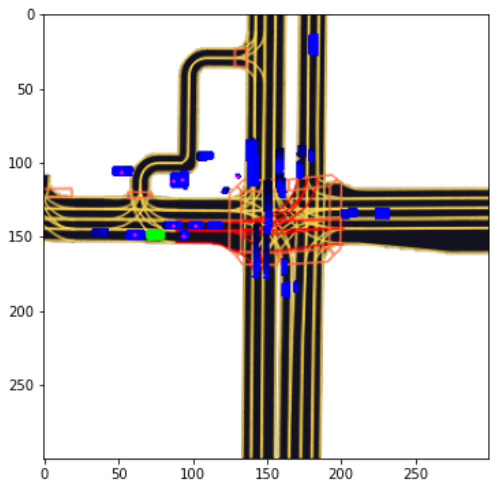
[8] Lyft Motion Prediction for Autonomous Vehicles - leaderboard
https://www.kaggle.com/c/lyft-motion-prediction-autonomous-vehicles/leaderboard

[9] Kosaraju, Sadeghian: Social-bigat: Multimodal Trajectory Forecasting Using Bicycle-gan and Graph Attention Networks
http://papers.neurips.cc/paper/8308-social-bigat-multimodal-trajectory-forecasting-using-bicycle-gan-and-graph-attention-networks.pdf

[10] Deo, Trivedi: Convolutional Social Pooling for Vehicle Trajectory Prediction
https://openaccess.thecvf.com/content_cvpr_2018_workshops/papers/w29/Deo_Convolutional_Social_Pooling_CVPR_2018_paper.pdf

Appendix

Examples for ground truth (pink line, left) and model prediction (light blue line, right)


0
50
100
150
200
250

0
50
100
150
200
250

0
50
100
150
200
250

0
50
100
150
200
250

0
50
100
150
200
250

0
50
100
150
200
250