Aalto University
School of Electrical Engineer
Master's Programme in ICT Innovation, Autonomous Systems

Riccardo Franceschini

Feasible and adaptive attention-based models for multimodal trajectory prediction in urban driving scenarios

Master's Thesis
Espoo, Sep 25, 2020

Supervisors:	Professor Themistoklis Charambolous, Aalto University
	Professor Elisa Ricci, University of Trento
	Professor Daniele Fontanelli, University of Trento
Advisor:	Tuan Anh Tran (Robert Bosch GmbH)
	Hendrik Berkemeyer (Robert Bosch GmbH)

Aalto University
School of Electrical Engineer
Master's Programme in ICT Innovation, Autonomous Systems

ABSTRACT OF MASTER'S THESIS

Author:	Riccardo Franceschini		
Title: Feasible and adaptive attention-based models for multimodal trajectory prediction in urban driving scenarios			
Date:	Sep 25, 2020	**Pages:**	70
Major:	Autonomous Systems	**Code:**	AUS
Supervisors:	Professor Themistoklis Charambolous Professor Elisa Ricci Professor Daniele Fontanelli		
Advisor:	Tuan Anh Tran (Robert Bosch GmbH) Hendrik Berkemeyer (Robert Bosch GmbH)		

A self-driving car which takes an autonomous decision needs three main building blocks, a perception module, a prediction module and a planning module. In this thesis, we consider the vehicle already capable of understanding the surrounding area; thus, we focus on the prediction module, which is responsible for predicting the future of the other agents in the scene. Thus, we examine, in particular, the prediction in urban driving scenarios in a multimodality setting where the model can learn to predict all the possible future scenarios in such complex environment. The predictions consist then of multiple sequences of coordinates plus the probability for each future. After having investigated the past and current methods, we have implemented different baselines, both deep learning methods and not. Hence, examining both the data that we used, and the network structures, we believe that some improvements are possible, and here we propose some methods to address those problems. First, we extend the previous loss with an additional term called offroad loss that penalise the model when the prediction lays outside of the road structure. Second, considering also that difficult scenes are rarer than simple scenes, we propose two different weighted sampling methods to overcome such imbalance, in this way, the model can adapt the prediction to more complicated and rare scenes. Finally, we try to extract more useful information from road structure, nearby agents and past information implementing different attention architectures inside the models. In this thesis, we also conduct a performance comparison between our methods and the baselines applying commonly used metrics. Moreover, to visually understand the impact of each method, we propose some anecdotal analysis showing the real differences in terms of prediction in some challenging situations.

Keywords:	Trajectory Forecasting, Autonomous Driving
Language:	English

Acknowledgements

I would like to thank Tuan Anh Tran and Hendrik Berkemeyer for the incredible opportunity and support they have given me.

Espoo, Sep 25, 2020

Riccardo Franceschini

Abbreviations and Acronyms

TV	Target Vehicle
SV	Surrounding Vehicle
EV	Ego Vehicle
BEV	Bird's Eye View
GAT	Graph Attention Network
ADE	Average Displacement Error
FDE	Final Displacement Error
MTP	Multimodal Trajectory Predictions

Contents

Chapter 1

Introduction

1.1 Background

According to the annual accident report [9] made by the European Road Safety Observatory in 2016, 25.600 people died from car accident and more than $1,4$ million of people got injured. In the past few years, autonomous driving has improved significantly and is shortly expecting to become a reality increasing safety and decreasing at the same time travelling time and traffic congestion.

Generally, in order to take an autonomous decision, a self-driving car needs three main components: a perception module, a prediction module and a planning module. The first one is the perception and consists on the localization and identification of the different parts of the road structure, for example being able of distinguishing the road structure from the vehicles that are on the road is part of the perception task. Once the perception task is completed, and the network can understand the surrounding area, we want to predict the immediate future of the agents in the area. Thus, the prediction task consists of predicting the immediate future of the nearby agents. After having completed both perception and prediction, we can plan the motion of the self-driving car and so we have the planning task.

Therefore, in this thesis, we aim to predict the future positions of a participating vehicle (agents) in an urban traffic scenario. Also in this thesis, we assume the availability of sensor data coming from the vehicles, as well as the contextual data coming from perception data, such as the surrounding map. In the following, we will give high level of our problem statement and approach.

1.2 Problem statement

One of the classic existing approach to address the behaviour prediction problem is to use traditional kinematic models, such as the bicycle model [15], relying only on the information coming from the target vehicle. However, due to the high complexity of the driving scenario, those methods are reliable only for short-time prediction because they are not able to capture all the scene details [17]. Thus, deep learning methods have started to gain more and more popularity due to their ability to handle more complicated situations.

Hence, to better define the problem that we are addressing, we can split the behaviour prediction into two macro-areas, the prediction in a highway scenario and the prediction in an urban driving scenario. In this thesis, we decided to focus on prediction on urban driving scenario for two reasons; the first one is that the urban scenario is much more complex and diverse than the highway. The second reason is that in the urban driving scenario, many interactions are happening between the agents in the scene, and so being able to correctly predict the behaviour is much more interesting than the highway from a practical application perspective.

Also, we have to split the prediction task into two categories, **unimodal prediction** and **multimodal prediction**. In the unimodal setting, the goal is to predict a single future that is close as possible to the real one, while with the multimodal setting, the goal is to capture all the possible driving scenarios generating multiple futures. However, in an urban driving scenario, every agent has multiple paths that can choose and so, a model that predicts only one possible future tends to predict a trajectory that is the average of all the possibilities, often resulting in an infeasible prediction. Hence, to overcome such problem the model should be capable of predicting multiple scenarios assigning to each future a probability and the multimodal setting try to address exactly this problem.

Therefore, the main goal of this thesis is to investigate the trajectory prediction, i.e., to predict the future positions of a given participating vehicle, in a multimodality setting. Given the input consisting both of agents dynamics and the contextual information, the output of the prediction is twofold: the set of future trajectories and the probabilities of each of them. Moreover, we are going to pay particular attention to the feasibility of the trajectories and to the adaptation of the model. We can define a trajectory feasible if it can be executed from both vehicle dynamic and road structure perspective. While, the adaptation is the ability of the model to adapt to the different road situations, e.g., road intersections or turning scenarios.

1.3 Approach overview

Considering that we decided to focus on urban driving scenarios, we chose as dataset nuScenes[3], where there are 1000 scenes of 20s each collected in Singapore and Boston. Then, we implemented three baselines, the Physical Oracle, MTP[5] and CoverNet[22]. The first one is based on kinematic models while the other two are deep learning methods. The Physical oracle receives as input only the agent dynamics to generate a unimodal prediction, while, MTP and CoverNet uses the agent dynamics and a top view representation of the area to produce a multimodal prediction. After having investigated both baselines and data structure of the dataset, we found some critical points, and in this thesis we propose some methods to solve those problems.

First, we noticed a significant data imbalance between complex and simple scenes, resulting in a trained model that is able to perform well on simple scenes but, it is unable to understand more complicated scenes. Therefore, we propose two different sampling methods based on lane density and lane curvature to show more often challenging scenes during the training process. This approach has shown to improve significantly the performance by allowing the model to better adapt to each scenario.

Next, we consider the level of understanding of the road structure. Examining the standard baselines, we realize that often the model is not capable of understanding the road structure, ending up with a non-feasible prediction. To avoid such behaviour, we introduce and additional loss term called **OffRoad** loss that penalizes trajectories that are outside of the drivable area.

At last, we believe that there are some parts of the baselines models structures where the features are not fully exploited, in particular, regarding the past agent state vector, the features from the top view image and from the nearby agents. For this reasons, we introduce multiple attention models to pay attention respectively to the information that is coming from the past using a Transformer layer [26], to the neighbor agents in the scene using a graph attention network[27] and to the surrounding image representation using a dual attention module [10].

Finally, we evaluate and compare the quality of the predictions of our improvements using multiple metrics commonly used in literature. Moreover, to show also the real differences between the predictions, we propose some anecdotal examples to better visualize difference and improvements.

1.4 Structure of the Thesis

This thesis is structured as follows, in Related Work (chapter 2) we analyze in detail the trajectory prediction problem and which are the typical approaches that are used to face this problem, classifying the previous attempts using the representation of input and output .

Moving to the Methodology (chapter 3), we discuss here which data we used during the experiments, how we represent it and some findings related to it. We also examine the three different baselines that we have implemented: Physical Oracle, MTP and CoverNet. Then starting from the baselines, we explain more in details the different improvement that we propose.

Afterwards in Implementation (chapter 4) we focus more on some technical aspects relative to the experiments such as the frame representation, the resampling methods and other techniques like batch averaging.

In the Evaluation (chapter 5) we then define the different metrics that are adopted to evaluate the models, and we analyze in detail each model and how every modification has affected the quality of the predictions. Additionally, it follows a section of anecdotal analysis in which we report some meaningful examples for understanding the different improvements. Consequently, it follows a Discussion (chapter 6) and then Conclusion (chapter 7).

Chapter 2

Related work

In order to understand the behaviour prediction problem, we introduce here some basic notation explaining the different approaches that researchers are commonly using to face this problem. Therefore, after a brief introduction about the terminology, we classify the most common approaches using their input and output representation. For each method, we consider positive and negative aspects giving some real examples.

2.1 Preliminary

Before starting it is useful to define some basic notation to represent the different agents that are involved in the problem:

- Target Vehicles (TVs), the vehicle interested in the prediction
- Ego Vehicle (EV), the autonomous vehicle from which we receive the data
- Surrounding Vehicles (SVs), the vehicles nearby the TVs, and they directly affect the TV's behaviour.
- Bird's Eye View (BEV), top view representation of the surrounding area.

2.2 Input representation

We start now from the input representation, examining in details the different methods. It is evident that to have a precise prediction; we should be able to represent the surrounding area in the highest accurate way. Thus, we should be capable of having a correct representation of the contextual information of the scene and at the same time, have a good representation of the agent dynamics. For representing the surrounding area, we commonly encode the contextual information with a top view representation. While, for the agent dynamics data, we rely more on the vehicle sensors to capture the movement of the vehicles.

Hence, we can divide the representation into four macro categories: only information form the TV, information from both TV and SVs, simplified bird's eye view and raw sensor data.

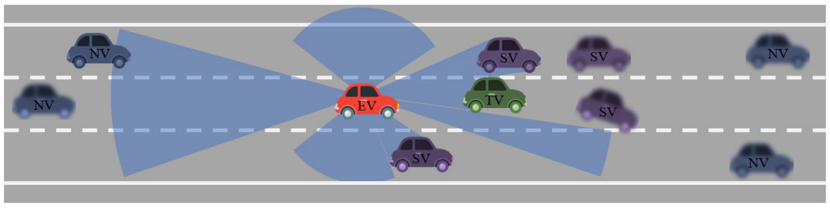


Figure 2.1: Example of driving scenario with the proposed terminology [20]

2.2.1 Target Vehicle track history

The fisrt approach is to use only the information from the target vehicle. Conventionally those features are position, velocity, acceleration and heading changing rate, here, we consider both the possibility of using only the current state or using both current and past state. Receiving information just from the TV the model is usually capable of learning well the dynamics of the vehicle, for example, if the vehicle is going straight, the network is able to predict the future correctly, same if it is turning. However, due to the lack of information from both nearby agents and road structure, the model does not have any information from the surrounding area. Thus, it is almost impossible for the network to predict a slow down if the vehicle in front is slowing down or a turning before the car starts to turn. As examples from the literature, we have [29, 30], where the track history is used to predict different road junctions, but they reduced the complexity of the scene removing other surrounding vehicles.

2.2.2 Target Vehicle and Surrounding Vehicles Information

A different input is to use track history from both the target vehicle and surrounding vehicles. As expected, this approach leads to a better understanding of the different vehicle's interactions. The number of SVs considered can be chosen statically as in [1] where they decided that 9 SVs was the correct number, or dynamically as in [18] where the authors have imposed a distance threshold for selecting if a surrounding vehicle is relevant or not for the TV. However, despite the number of SVs considered, those approaches do not consider either occlusion (which may hide connected vehicles) or the road structure which can profoundly influence the future trajectory.

2.2.3 Bird's Eye View

One more common approach is to use a simplified Bird's Eye View (BEV). Using this method is possible to combine dynamic elements such as surrounding vehicles, or pedestrian and static elements such as a roundabout, drivable lanes and pedestrian crossings on the same image. In fact, from Figure.2.2,

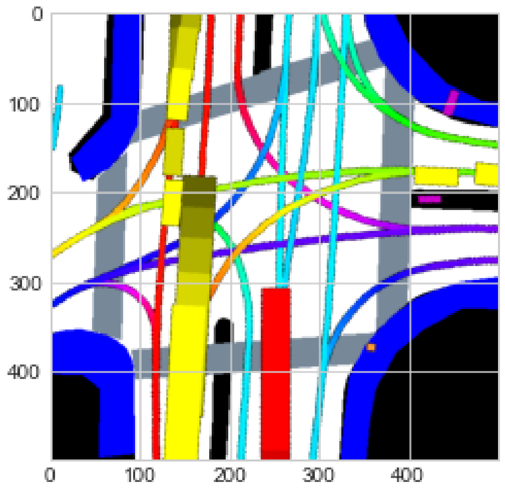


Figure 2.2: Example of BEV

the BEV is capable of representing both road structure and moving agents.

There are multiple examples of possible BEV representation. For example, from [16], they try to improve the safety of the ACC (Adaptive Cruise Control) stacking multiple two-channel BEV on top of each other building a

2n-channel image, encoding the temporal dimension into the input so that the network can sequentially learn better the road dynamics. Another example is [5] in which the network receives as input only one image. However, instead of being a two-channel image, every road element has his colour scheme that changes dynamically, resulting in a more informative representation like in Figure.2.2. Nevertheless, this approach is often results quite complicated for the network to understand, leading to the possibility of losing information only because the model is unable to extract it.

2.2.4 Raw sensor data

The last input method that we are covering is the usage of raw data. With this method, the network understand the data trying to extract meaningful information from it. However, this approach requires more computational power due to the high dimensionality of the data. Therefore, directly using the raw data can be computationally infeasible in a real implementation in an autonomous vehicle. An example where a model directly exploits the LiDAR data is [4] in which they use both LiDAR and BEV representation to predict the future coordinates and the intention such as keep lane, turn left, turn right, etc.

2.2.5 Combine inputs

Every input approach previously described is capable of emphasizing a particular aspect of a driving scenario. Thus, in the more refined models such as [5, 13, 19, 22] is common to combine information that is coming from the surrounding representation using the BEV and detailed dynamic information of the specific TV. With this approach, the network is then capable of having a general idea, but at the same time, can focus on the specific agent generating a more reliable prediction.

2.3 Output representation

We now move our focus to the possible output representations, examining the positive and negative aspect of each method.

2.3.1 Intention

As first and simpler method we mention the intetion prediction, which is the task of classifying future behaviour using a set of manoeuvres. For example, in [4, 7], the authors have trained the network to predict a set of future manoeuvres such as keep lane, turn lane, etc. It is immediate to see that this approach has its advantages and its disadvantages, in fact, reducing the forecasting task to a finite set of manoeuvres dramatically simplifies the complexity of the prediction. On the other hand, it is clear that despite the granularity of the manoeuvre set, for example, classifying sharp and regular lane, the set always remains finite. While directly predicting the coordinates allows the model to be much more precise at prediction time. Still, it increases at the same time, the complexity due to the infinite coordinate possibilities.

2.3.2 Unimodal Trajectory

In order to avoid the problems of precision introduced before with the intention approach, we present here the unimodal trajectory prediction. With this method, the task of the network is to predict the sequence of coordinates over a fixed time window. Thus, the problem of predicting the future trajectory can be defined as predicting:

$$X_{TVs} = \left\{(x_t^i, y_t^i), (x_{t+1}^i, y_{t+1}^i), ..., (x_{t+m}^i, y_{t+m}^i)\right\}_{i=1}^N, \tag{2.1}$$

where (x_t^i, y_t^i) are the Cartesian coordinates of the vehicle i at time t and N is the number of vehicles predicted and m is the length of the points predicted.

There are multiple attempts of unimodal trajectory, for example in [18] the authors used a graph neural network to predict the future location of the all nearby agents simultaneously or in [8], where they used as input a rasterized map, velocity, acceleration and heading change rate to predict the unimodal future.

Despite being much more refined than the intention base approach, the unimodal approach fails not considering the multimodality of the driving scenario. For example, if an agent is at a road junction, usually it has multiple possible future trajectories, like turning left or right. However, with a model capable of generating only one possible future we lose the possibility of understanding those road details, in fact, the network usually tends to average among all the possible futures, ending up with a non-feasible prediction.

2.3.3 Multimodal Trajectory

In order to solve the problem previously described, we introduce the multimodal trajectory prediction. With the multimodal approach, the network

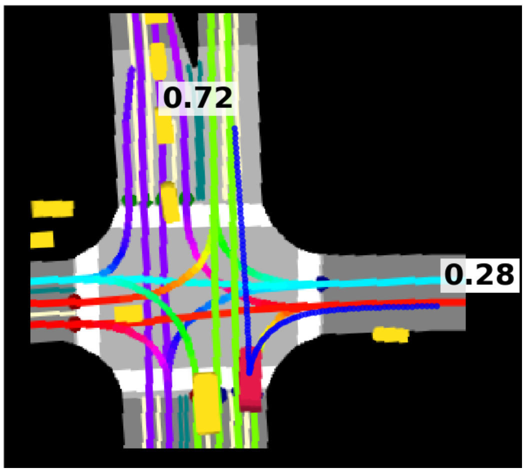


Figure 2.3: Multimodal Prediction [5]

predicts a fixed number of possible future trajectories, and for each trajectory output also the probability that every particular scenario is going to happen. Hence, the multimodality version of trajectory prediction can be defined as:

$$M_{TVs} = \left\{(X^{i}_{TVs}, p^{i}), (X^{i+1}_{TVs}, p^{i+1}), ..., (X^{K}_{TVs}, p^{K})\right\}^{K}_{i=1}, \tag{2.2}$$

where K represent the number of modes that the model predicts and p^i are the probability of each mode to be the real one. Therefore, predicting a variety of different futures, the model learns better to understand the dynamics of the road scenarios, realizing for example, that at an intersection usually, a vehicle has multiple potential possibilities of manoeuvres. To understand an example of how a multimodal prediction looks like we can see Figure.2.3, where the network predicts the possible driving scenarios, and for each future behaviour, it also predicts the probabilities.

Hence, the advantage of the multimodal approach became quite clear, especially in urban driving scenarios where the complexity of the road structure is much higher than the highway. As examples of multimodal we can cite [5] and [22] where the authors have used features coming from BEV and TVs to generate possible futures, those two models are also our baselines, and we will cover and analyze more in details their structure in the following chapters.

2.3.3.0.1 Why multimodal To realize why the multimodal prediction is the most complete among all the methods, consider that we want to use the behaviour prediction system into our autonomous vehicle. Hence, having a model that is capable of always generating the correct prediction out of his different modes can lead to a vehicle that is capable of understanding the future and then plan a safe trajectory as a consequence.

Moreover, with the multimodal approach, we can decide the number of modes that the model has to predict. So, we can create constraints and refine the level of understanding of the model. For example, if the network has 10 modes instead of 5, it will be able to generate a more refined prediction of a particular scenario. For instance, a turning manoeuvre can look very different depending on the driver, and so, the model with more modes will be able to create a better representation. At the same time, increasing too much the number of modes leads to more complexity that can deteriorate the model capabilities of choosing the correct predictions among the different modes. For all the reasons previously mentioned, our prediction approach will always be multimodal.

Chapter 3

Methodology

After having introduced the problem with all the different details between output and input representation, we now describe our methods. In the following sections, we are going to consider the data used, the different baselines implemented and how we have tried to improve the baselines performances.

3.1 Data Analysis

We start with the description of data used in this thesis and present some results of the analysis.

3.1.1 Dataset

Considering that our focus is on urban driving scenes, we decided to use as dataset nuScenes [3], developed by Motional. NuScenes is a public large scale dataset, where the data collected comes from real urban driving scenarios in Boston and Singapore, there are 1000 scenes of 20 seconds each with annotation for 23 object classes that are annotated every 2Hz. The data comes from a full suite of sensor: 1 LiDAR, 5 RADAR, 6 cameras, IMU and GPS.

Even though nuScenes was initially designed for perception tasks such as classification or segmentation, they expanded the dataset capabilities also for behaviour prediction. Therefore, Motional has created a devkit giving the possibility to extract relevant feature for the prediction task. For example, they have introduced the possibilities of extracting all the agents in a particular scene, or they allowed getting the past or the future agent state vector of each agent. Another useful implementation is the possibility to generate a BEV version of the map as described in the previous Figure 2.2.

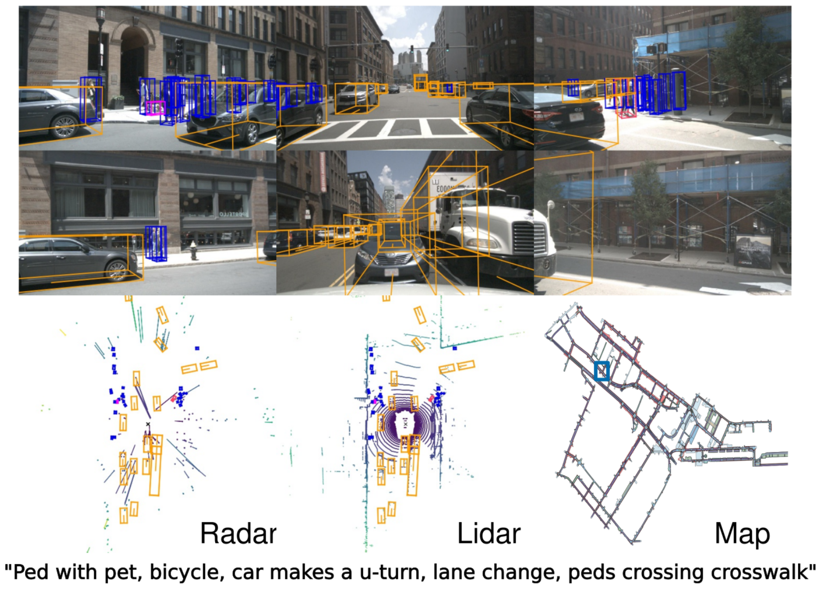

Radar
Lidar
Map
"Ped with pet, bicycle, car makes a u-turn, lane change, peds crossing crosswalk"

Figure 3.1: Example of data structure [3].

3.1.2 Input Representation

As input for our experiments, we always use a combination of the methods described in 2.2. In all the experiments, we changed the representation of the agent state vector multiple times, but the BEV image is always included to encode the general information. Therefore, having a frame in which the information are correctly represented, become crucial for good performance of the model. Thus, we construct the frame so that the target vehicle is always on centre pointing upwards. In this way, the TV is always at the centre of the reference system. Consequently, it is easier for the network to focus on the relevant data.

The frame has dimension $(500, 500)$ pixel with a resolution of 0.1 meters/pixel. There are $40m$ ahead, $10m$ behind, and $25m$ left and right to the TV at the centre. Moreover, every agent has the "shadow" in which the past is represented in a fading colour. The lanes centerline are represented with a specific colour schema in which red/orange lanes are in the opposite direction while blue/green lines are the ones in the correct driving direction of the TV.

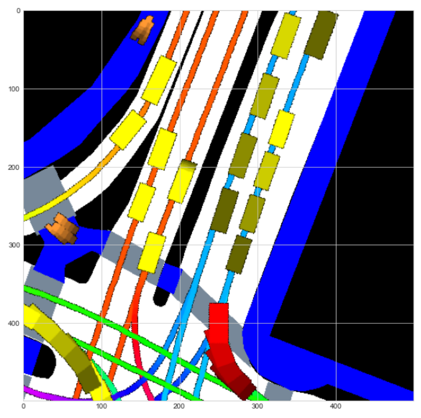


Figure 3.2: Frame example

3.1.2.1 Agent State and History

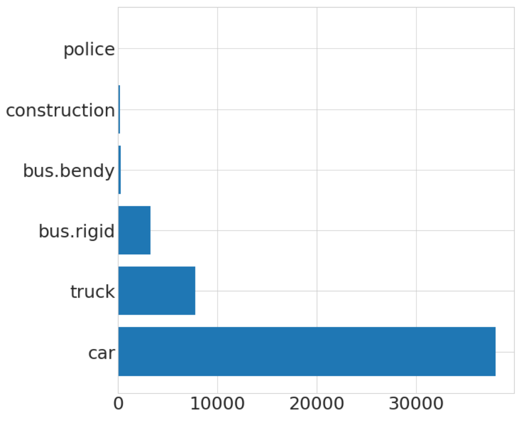
Another relevant part of the input is the agent state vector which is composed of speed, acceleration and heading changing rate of the target vehicle. However using only the current information might lead to a non-precise understanding of the physics of the target vehicle, for this reason, we also decided to include the previous two second of history in the agent state vector. Compared to HighD [14], which is a dataset based only highway, nuScenes is based on urban driving scenarios. Therefore, the interactions between the surrounding vehicles are very relevant for a complete understanding of the situation. For this reason, we also conducted experiments using state vectors from the nearby agents.

3.1.3 Data investigation

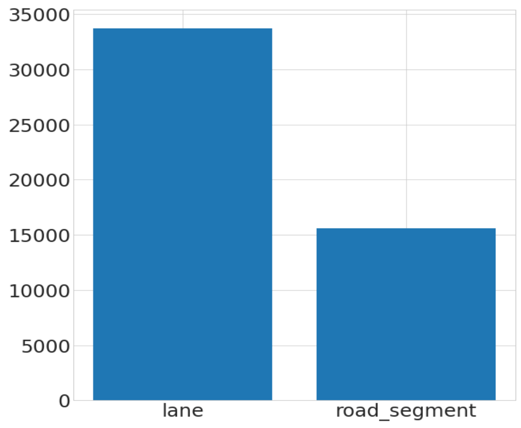
Evaluating more in detail the data structure, we find out that nuScenes identify the data using instance and sample tokens, where the instance defines the agent and the sample defines the time. Hence, using the sample token is possible to retrieve all the agent in a particular scene while with the instance token, we can retrieve information of a specific agent. In total there are 49787 pair of instance and sample tokens that we split in 32186 tokens for training and 9041 tokens for validation.

Considering the urban driving nature of nuScenes is interesting to investigate how is the distribution of the vehicles and the road elements. Using a token is possible to retrieve which vehicle we are analyzing and on which road elements the vehicle is riding. Where the road elements are classified into two macro-categories, the lanes and the road segments, the first ones are standard lanes while the second ones are intersections.

From Figure 3.3a we find that as we were expecting, the amount car is significantly higher than the other vehicles. However, it is important to notice how the bus, that is a typical city transportation method is relevant in the driving scenario, counting in total 3568 scenes considering both rigid and bendy busses. Regarding the road segments and lanes distributions, is interesting to point out the magnitude of road segments that despite being lower than the lanes are still in a considerable number and they are contributing to add more complexity in the driving scenario.



(a) Vehicle distribution



(b) Lane distribution

Figure 3.3: Vehicles and road elements distribution

3.1.3.1 Data imbalance

Even though nuScenes is based on urban driving scenarios, we have found a high level of imbalance between the complexity of each scene. To analyze such difficulty, we have extracted with a certain level of accuracy the manoeuvres, using the information relative to the motion of the vehicle. Thus, we classified three possible manoeuvres, lane following, turning, and lane changing. Then, analyzing the distribution of those manoeuvres, we notice a high imbalance between the manoeuvres, with the following lane covering the majority of the manoeuvres. We usually expect to have a certain level of imbalance because the other two manoeuvres tend to occur much more rarely. Nevertheless, we believe that a system to balance the different manoeuvre is needed. From Figure 3.4 is quite evident that the majority of samples are following lanes

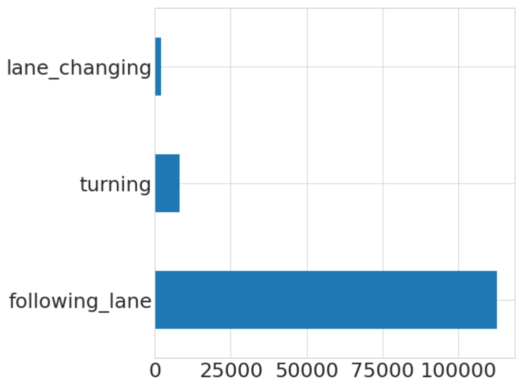


Figure 3.4: Maneuver occurrence in the dataset

which are usually easier to learn. In contrast, other manoeuvres such as turning or lane changing are rarer and consequentially more difficult for the network to learn. For these reasons, we propose two different methods of sampling that we will analyze in the following sections.

3.2 Models

We now examine the different models that we have implemented, considering their weak points and what we have done to improve those models.

3.2.1 Baselines

We have then implemented three different baselines.

3.2.1.1 Physics Oracle

The first model, that we consider, it is not a neural network but is based on different kinematic models. What the Physical oracle does is to execute multiple physical models and then select the most similar to the ground truth trajectory. We have then implemented the following models.

3.2.1.1.1 Constant velocity and heading Given x,y coordinates and v_x v_y velocity the model is:

$$\begin{bmatrix} x^{t+1} \\ y^{t+1} \end{bmatrix} = \begin{bmatrix} x^t + \Delta t v_x \\ y^t + \Delta t v_y \end{bmatrix}, \tag{3.1}$$

where Δt is the time passed between (x^t, y^t) and (x^{t+1}, y^{t+1}) .

3.2.1.1.2 Constant speed and yaw rate Considering that the speed and yaw(θ) rate are constant, we compute the future positions as follows:

$$\begin{bmatrix} x^{t+1} \\ y^{t+1} \\ \theta^{t+1} \end{bmatrix} = \begin{bmatrix} x^t + dist\_step * \cos\theta \\ y^t + dist\_step * \sin\theta \\ \tau^t + \theta\_step \end{bmatrix}, \tag{3.2}$$

where given the sampling frequency at $f = 2Hz$ the θ_step and $dist\_step$ are:

$$dist\_step = \frac{1}{f} * speed, \tag{3.3}$$

$$\theta\_step = \frac{1}{f} * yaw\_rate, \tag{3.4}$$

3.2.1.1.3 Constant acceleration and heading With the acceleration and the heading constant we obtain:

$$\begin{bmatrix} x^{t+1} \\ y^{t+1} \end{bmatrix} = \begin{bmatrix} x^t + \Delta t v_x + \frac{1}{2}\Delta t^2 acc_x \\ y^t + \Delta t v_y + \frac{1}{2}\Delta t^2 acc_y \end{bmatrix}, \tag{3.5}$$

3.2.1.1.4 Constant acceleration and yaw rate We examine now the acceleration constant and θ_rate constant as before:

$$\begin{bmatrix} x^{t+1} \\ y^{t+1} \\ \theta^{t+1} \end{bmatrix} = \begin{bmatrix} x^t + dist\_step * \cos\theta \\ y^t + dist\_step * \sin\theta \\ \theta^t + \theta\_step \end{bmatrix}, \tag{3.6}$$

considering always the frequency as $f = 2Hz$ and the θ_step as 3.4 but this time the $dist\_step$ is computed at every iteration as it follows:

$$dist\_step = \frac{1}{f} * speed, \tag{3.7}$$

where the speed is:

$$speed = speed + \frac{1}{f} * acc, \tag{3.8}$$

3.2.1.1.5 Prediction with kinematic models Thus, in order to generate the coordinate (x, y) given the state from the TV, the model compute the predictions using each kinematic model described above, then it select the prediction choosing the one with the least L2 error. This approach cannot be considered as a valid prediction approach because it is using the ground truth directly to generate future trajectories. However, it is useful to have it as a baseline because it allows us to study the difference between the deep learning approach and a more classic approach.

3.2.1.2 MTP

Moving to the deep learning models we start with MTP (Multimodal Trajectory Predictions for Autonomous Driving using Deep Convolutional Networks [5]), where the authors are combining rasterized image and agent state vector to predict the multimodality of the driving scenario.

3.2.1.2.1 Network structure From Figure 3.5 we see that the network receives two separate input, a rasterized image of the driving scene and the current actor state (velocity, acceleration, and heading changing rate). Then the BEV is processed trough a pre-trained CNN, in this case MobileNet-V2 [24]. After that, the rasterized features and the state input are concatenated

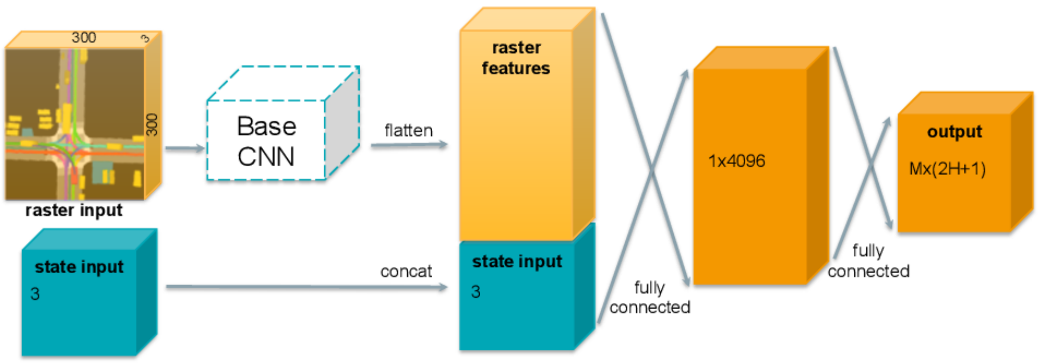


Figure 3.5: Network Structure from [5].

and processed trough some fully connected layers. The main contribution of MTP was to combine the prediction task and the mode classification task using only one architecture. Thus the output of the network is composed of the sequence of coordinates (x, y) multiplied for the number of modes M that the network predicts and the probability for each predicted mode.

3.2.1.2.2 Loss The final objective of MTP is to predict the future trajectories and then also to decide which one of that possible futures is the most likely. To do that the authors developed two different losses, one for the classification and another one for the regression and they call it Multiple-Trajectory Prediction loss. They created this loss because only using a mixture of experts (ME) eq.3.9, will cause the mode collapsing problem in which all the predictions point to the average, similar to what happens with the unimodal approach.

$$\mathcal{L}_{ij}^{ME} = \sum_{m=1}^{M} p_{im} L(\tau_{ij}, \widetilde{\tau}_{imj}), \tag{3.9}$$

So, they identify the mode $m*$ which is closest to the ground truth using a distance function $dist(\tau_{ij}, \widetilde{\tau}_{imj})$. Then, the best mode can be selected as:

$$m* = \operatorname*{argmin}_{m \in \{1..M\}} dist(\tau_{ij}, \widetilde{\tau}_{imj}), \tag{3.10}$$

as a classification loss, we can consider a cross-entropy loss defined as follows:

$$\mathcal{L}_{ij}^{class} = -\sum_{m=1}^{M} I_{m=m*} \log p_{im}, \tag{3.11}$$

where p_{im} is the probability of each mode. It is now possible to define the final MTP loss as:

$$\mathcal{L}_{ij}^{MTP} = \mathcal{L}_{ij}^{class} + \alpha \sum_{m=1}^{M} I_{m=m*} L(\tau_{ij}, \widetilde{\tau}_{imj}), \tag{3.12}$$

where α is a hyperparameter and L is the $\ell 2 - norm$. It is worth to mention how MTP select the best mode, in fact, it has proved that the euclidean distance it is not able to model well at the intersections and so they proposed to choose the best mode using the angle difference. From fig.3.6, we can see

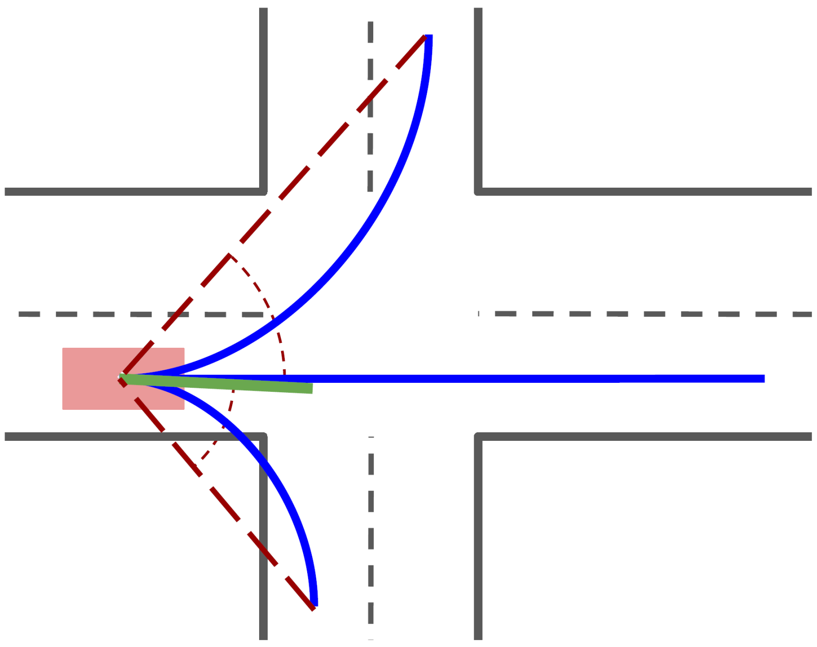


Figure 3.6: Difference between and angle distance [?]

the different predictions and the ground truth. The green line is the ground truth, and the blue lines are the predictions. If we use the distance approach to choose the best one, the right turn will result as the best; otherwise, if we select using the angle difference, the straight prediction will be the chosen one. This aspect is relevant because the regression loss will compute the difference using only the best trajectory. In this way, each output of the network will be specialized in a different trajectory (e.g. going straight, turning left or right).

3.2.1.3 CoverNet

Another interesting approach studied is CoverNet [22]; here, the authors have changed the task of the network from regression to classification. To do that, they have created a set of trajectories that are representative of the possible future scenarios. Therefore, what the network does instead of predicting the coordinates is to give a probability of each trajectory inside a defined set of trajectories that are generated sampling directly from the training data. The idea behind this approach is to use the trajectory set to avoid the mode

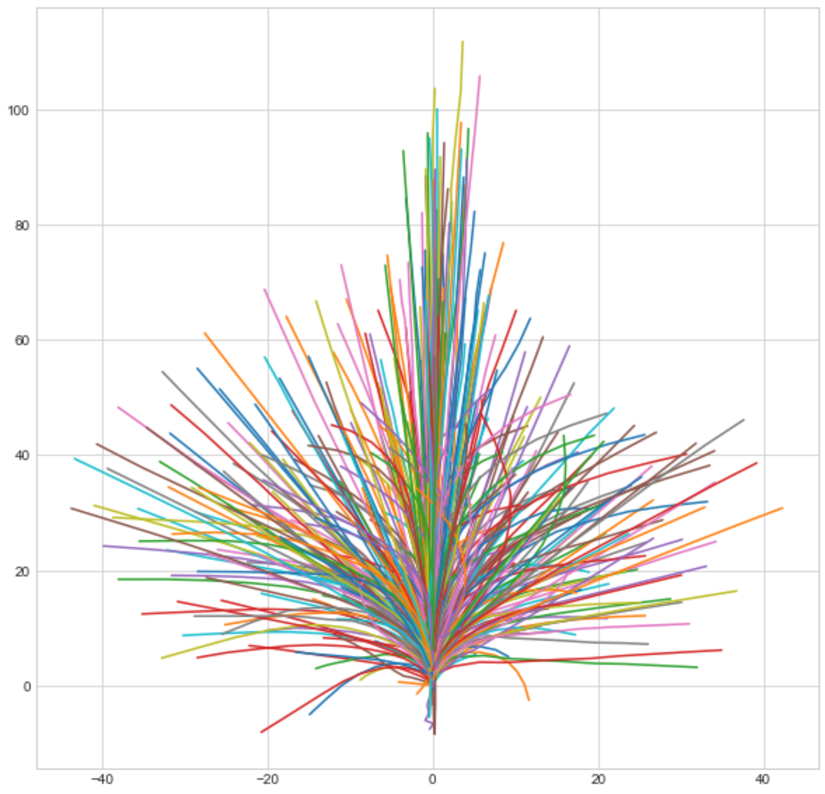


Figure 3.7: Trajecory set [22]

collapsing already discussed in 3.2.1.2. This method can solve the problem because the network does not have to predict future coordinates, but only has to predict the probability of each possible output scenario.

3.2.1.3.1 Network structure Regarding the network structure, we find a similar approach already encountered in MTP. The model receives as input a BEV image and the state input representation (speed acceleration and yaw rate). Then the network output is a vector containing the probabilities for

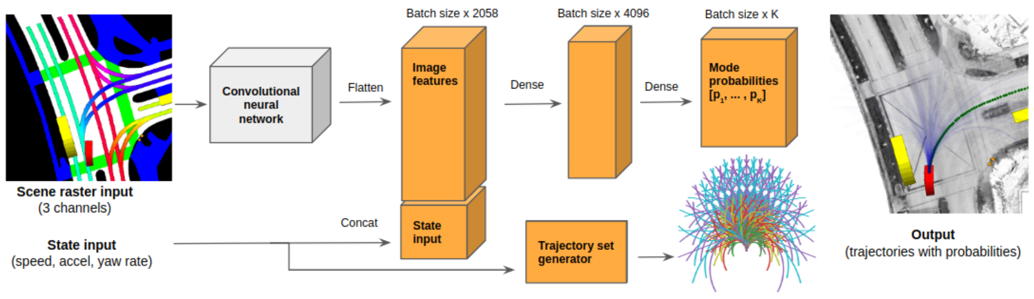


Figure 3.8: Network Structure [22].

each mode of the trajectory set. Differently from [5] the authors have used as backbone for the rasterized image a Restenet-50 [11] already pre-trained on ImageNet[23].

3.2.1.3.2 Loss Considering that the output dimension is equal to the number of different trajectories from the trajectory set, the loss of CoverNet is relatively simple. In fact, given the ground truth coordinates is possible to find which trajectory is the best for that particular situation using the mean pointwise L2 distance. So, given the coordinates:

$$t_m = \left\{(x_t^m, y_t^m), (x_{t+k}^m, y_{t+k}^m), (x_{t+n}^m, y_{t+n}^m)\right\}_{k=1}^{N}, \tag{3.13}$$

$$g = \{(x_t, y_t), (x_{t+k}, y_{t+k}), (x_{t+n}, y_n)\}_{k=1}^{N}, \tag{3.14}$$

respectively, the trajectory coordinates of a mode m and the ground truth coordinates, it possible to compute the $\ell 2$-norm as follows:

$$\ell 2\text{-norm} = \frac{\sum_{i=t}^{t+N} \parallel t_i^m - g_i \parallel_2}{N}, \tag{3.15}$$

At this point, to get the best mode we have to select the one with the lower distance using argmin. Therefore, knowing which mode is the best from the trajectory set, is possible to compute the cross-entropy loss among all the given output probabilities from the network:

$$\mathcal{L}^{covernet}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right), \tag{3.16}$$

3.2.2 Improvements

Examining the baselines described in the previous chapter, we have found some critical points of those architectures. Therefore, in the following sections, we are going to cover the methods implemented to overcome such problems.

3.2.2.1 Transformer

The first modification regard the lack of information coming from the past. As discussed in 3.1.2.1, most of the relevant information to predict the future is encoded in the past. However, those data are not considered in the baselines previously described. Therefore, we decide to learn the past using a Transformer [26] layer. We chose to use this approach because the Transformer is already well known for his capabilities of handling sequences of data. Thus, the past 2 seconds of the TV are processed by the Transformer layer to learn useful information.

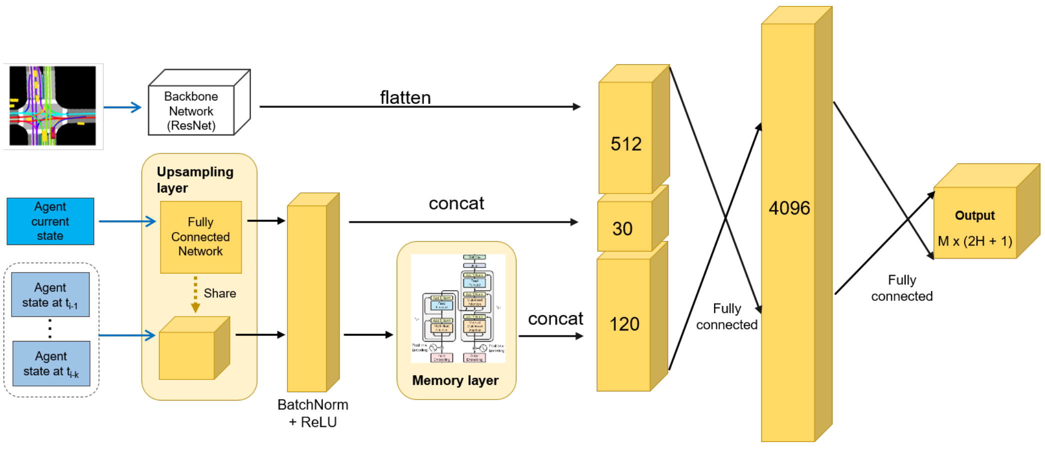


Figure 3.9: MTP architecture with Transformer [26].

Moreover, we have realized that after the concatenation between TV features and backbone feature, there might be a lack of understanding of the correlation between those representations. For this reason, we try to capture the correlation using the *self-attention* mechanism provided by the Transformer encoder.

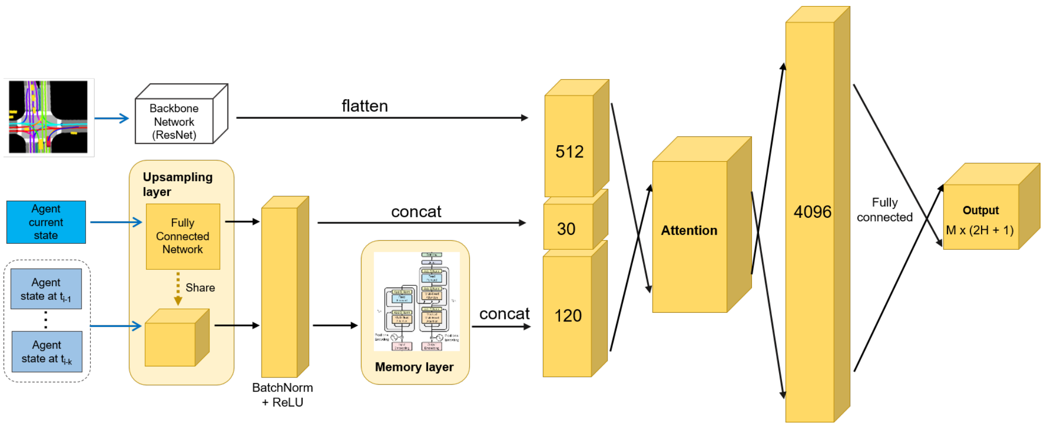


Figure 3.10: Dual attention architecture.

3.2.2.2 Graph attention network

Another critical piece of information that is missing from the previously described methods is the lack of information coming from the nearby agents. Especially in urban driving scenarios where interactions between agents occur much more frequently that information is fundamental for a reliable prediction. For this reason, we decided to model those interactions using a graph attention network [27]. In the past, there have been other attempts to model nearby interactions. For example, in [6], they used a convolutional social pooling layer to extract useful information from a social tensor populated using LSTM trough a spatial grid around the agent. However, graph networks are more suitable to learn interactions between different vehicles, and in particular, the graph attention network has shown to be efficient from a computation perspective without sacrificing state-of-the-art performance. For this reason, we decided to follow a similar approach of [13] in which they use Bicycle-GAN and graph attention network for pedestrian behaviour forecast.

What GAT does to understand the different interaction between nodes is to use a *self-attention* mechanism. Thus, the input is a set of node features, $h = \{\overrightarrow{h_1}, \overrightarrow{h_2}, .., \overrightarrow{h_n}\}$ with $\overrightarrow{h_i} \in R^F$, where N is the number of nodes, F are the features in each node and the output is a new set of node features $h = \{\overrightarrow{h'_1}, \overrightarrow{h'_2}, .., \overrightarrow{h'_n}\}$ with $\overrightarrow{h'_i} \in R^{F'}$ where $R^{F'}$ could have a different cardinality.

Therefore, after having transformed the input feature to an higher-level $R^{F'}$ with a fully connected layer:

$$z_i^{(l)} = W^{(l)} h_i^{(l)}, \tag{3.17}$$

a *self-attention* is performed on the nodes. Then, we obtain the un-normalize attention score between the nodes i, j as:

$$e_{ij}^{(l)} = LeakyReLU(\overrightarrow{a}^{(l)^T}(z_i^{(l)} | z_j^{(l)})), \tag{3.18}$$

where a is another fully connected layer and $|$ represent the concatenation. After that, the attention coeffiecents are normalized using the softmax function:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N} \exp(e_{ik})}, \tag{3.19}$$

It is important to notice in 3.19 that in the original paper the authors used a system of masked attention in order to pay attention only to the first-order neighbors of the node i, while in our implementation every node can attend all the other nodes. Next, we use the normalized attention score to compute the output of each node:

$$\overrightarrow{h_i'} = \sigma\left(\sum_{j \in N} \alpha_{ij} W \overrightarrow{h_j}\right), \tag{3.20}$$

Therefore, we create the graph as follows. Given N nearby agents, we create a fully connected graph, so that every node can attend every other node. Then we select the N nearby agents (NA) to the target vehicle and given the global coordinates (x_{NAG}, y_{NAG}) of the nearby agents and (x_{TVG}, y_{TVG}) of the TV is possible to compute the relative coordinates (x_{NAR}, y_{NAR}) with respect to the target vehicle. Then, we pass the past $2s$ of relative coordinates from both NA and TV trough the same LSTM layer. Thus, every output of the LSTM will be the feature of each node of the graph.

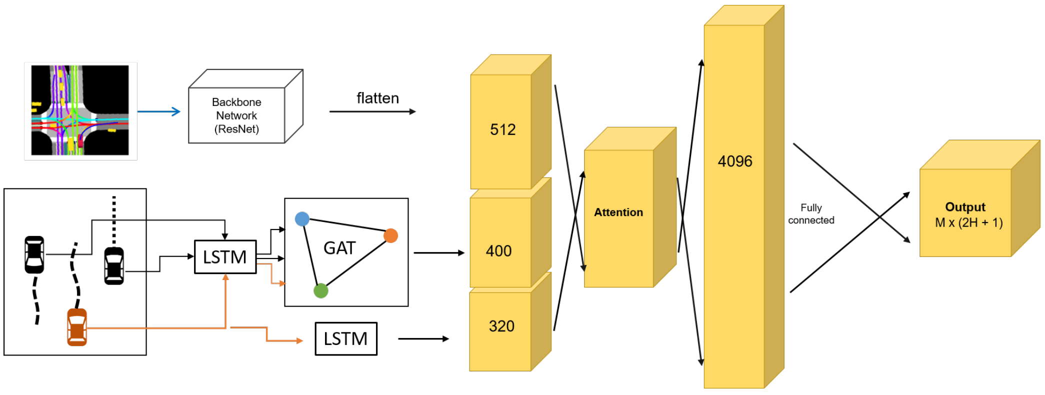


Figure 3.11: Architecture with GAT

We decide to use the relative coordinates as features from the graph instead of the agent state vector (speed, acceleration, heading changing rate) because we believe that knowing the position relative to the target vehicle is much more meaningful for the network when it has to learn dependencies between agents. However, the agent state vector is really important to fully understand vehicle dynamics. Thus, we process it using a separate LSTM encoder. Finally, we concatenate all the features, and we generate predictions.

3.2.2.3 Backbone Attention

We also believe that another weak point of the two baselines that can be improved are the features extracted from the BEV image. In fact, despite the backbone used to extract the features, we believe that there is a lack of attention on the relevant features that the network can extract from the BEV. For this reason, we are introducing here a way to attend the relevant features from the backbone. Therefore, we decided to implement a dual attention module from DaNet [10] in our previous architecture, we chose this approach for its ability to be implemented in a pre-existing backbone without adding a significant computational demand but significantly improving performance.

The idea behind DaNet is to have two different *self-attention* [28] modules that are executed in parallel after receiving features from a pre-trained backbone. Those two modules are the Position Attention Module (PAM) and the Channel Attention Module (CAM).

Therefore, the Position Attention Module (FIgure 3.12) can learn to ex-

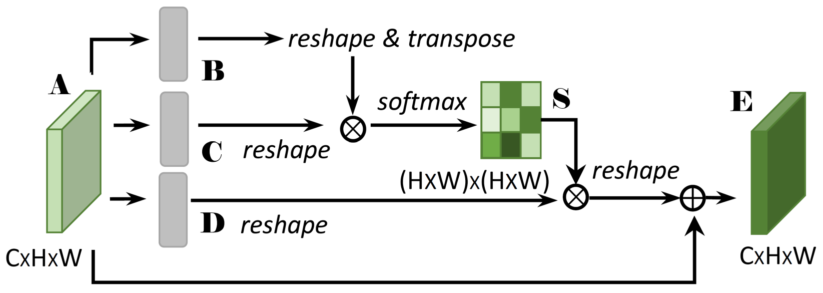


Figure 3.12: Position attention module [10].

tract useful information between local features while the Channel Attention Module can focus on the channel relationship improving the feature representation of a specific semantic.

Hence, to capture those relations the PAM module has to generate a spatial attention matrix S that model the relationship between any two pixels, thus, the S matrix is the softmax between the matrix multiplication of B and C:

$$s_{ij} = \frac{\exp(B_i \cdot C_j)}{\sum_{i=1}^{N} \exp(B_i \cdot C_j)}, \tag{3.21}$$

It is important to point out that on S, s_{ji} measures the i^{th} position impact on the j^{th} position. Then, a multiplication between the attention matrix and the original feature is done. Finally, we perform an element-wise sum between the matrix previously obtained and the original features, so that E is:

$$E_j = \alpha \sum_{i=1}^{N} (s_{ji} D_i + A_j), \tag{3.22}$$

and α is a parameter that is initialized as 0 and gradually learns to assign the right amount of weight. From eq.3.22 can be seen that E is a sum between the original features and weighted features across all the positions, this sum allows the PAM to understand the spatial relation between different elements. At the same time, the CAM (Figure 3.13) perform the same operations of the PAM except for the spatial attention matrix that is now the channel attention matrix and is calculated as:

$$x_{ij} = \frac{\exp(A_i \cdot A_j)}{\sum_{i=1}^{C} \exp(A_i \cdot A_j)}, \tag{3.23}$$

in this case x_{ji} measures the i^{th} channel impact in the j^{th} channel. Then the

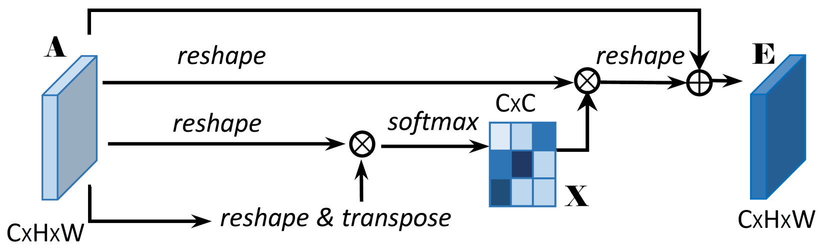


Figure 3.13: Channel attention module [10].

matrix E is:

$$E_j = \beta \sum_{i=1}^{C} (x_{ji} A_i + A_j), \tag{3.24}$$

also here, β is parameter that learns the weight. Like in 3.22 also in 3.24 we see that the final features of each channel is a sum between the original features and features from all the channels.

3.2.2.3.1 DaNet in our network We have implemented two different versions of our architecture with the DaNet module integrated. The first implementation has the DaNet module inserted after the backbone and before the concatenation (Figure 3.14). In this way, the features that before were coming from the backbone now are the sum of the two attention modules.

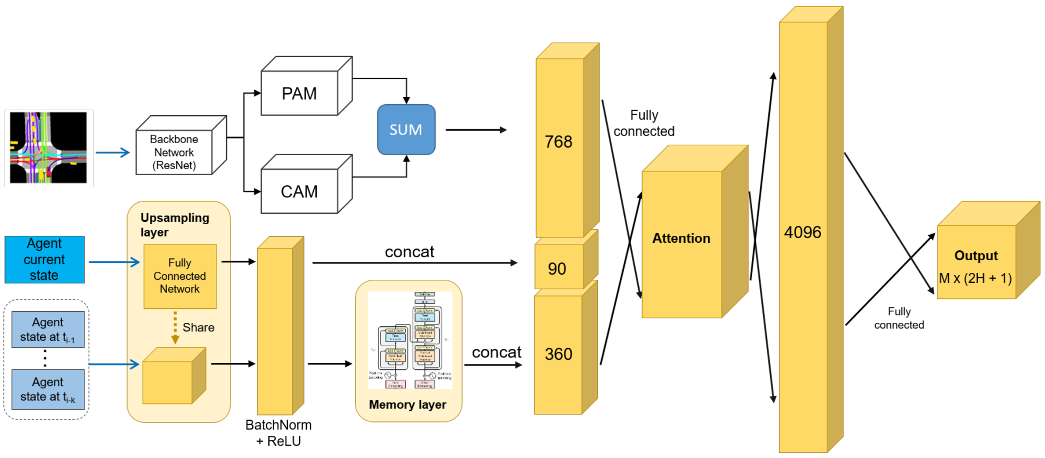


Figure 3.14: Single DaNet.

With the second approach, the structure is a bit different compared to the initial approach. In fact, we concatenate the features extracted by the PAM and CAM with the agent state vector, and we repeat for every frame of the past $2s$ (Figure 3.15). After that, we create a sequence of those features, and we encode the time dependencies using a sinusoidal positional encoder module like in [26]. In fact, we believe that learning the time dependencies from the features extracted by the frame helps the network to better understand the dynamics of the urban scenario.

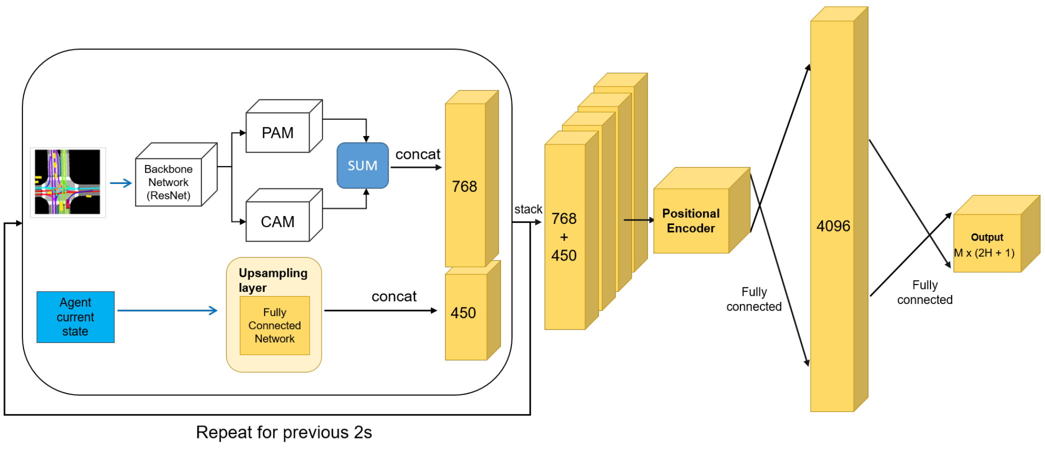


Figure 3.15: DaNet concatenated.

3.3 Loss

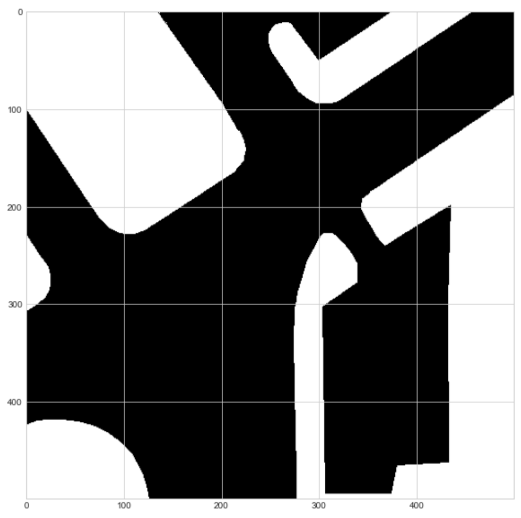
Another critical part of the learning process is the loss. We have already discussed in 3.2.1.2.2 how a good loss can avoid problems such as mode collapsing and how the eq.3.12 is already a well-structured loss to avoid such problem. However, the previous approaches do not consider the feasibility, and the safety of the trajectory predicted. For this reason, we decided to insert an additional loss term that is going to model if the trajectory predicted lays inside the drivable area.

3.3.1 OffRoad Loss

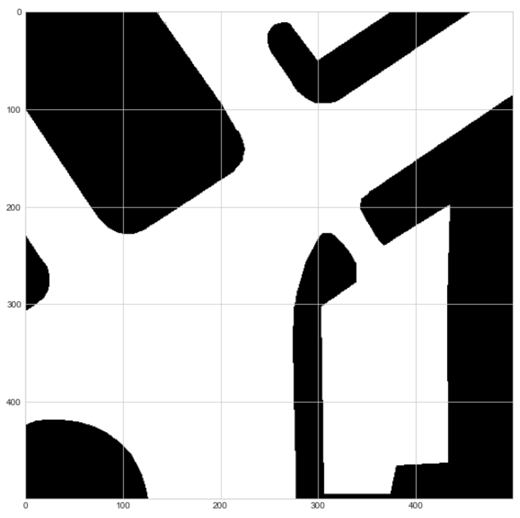
In order to guaranteed the feasibility of the trajectory predicted, we have to ensure that the predicted trajectory is inside of the drivable area. Therefore, inspired by [21] and [19] we implemented the offroad loss.

With the offroad loss we want to penalize the predictions that are far from the drivable area, and furthest is the prediction higher is the loss. For this reason, we created a heatmap of the drivable area as follows.

As first we retrieve the mask of the drivable area where the road as value 1 and the rest is 0, at this point, is possible to create the negative mask negating each pixel, so that, the drivable area has value 0 Figure 3.16.



(a) Drivable area mask



(b) Negative of Drivable area mask

Figure 3.16: Drivable area mask before and after negation

Afterwards, to compute the heatmap, every pixel is replaced with the distance to the closest point that lays inside the drivable area. Considering that the road structure can vary a lot and so also the value of each pixel, we normalize the heatmap (Figure 3.17). Once the network has generated the predictions, the offroad loss is computed as follows. For each prediction, given the sequence of coordinates $X = \{(x_t, y_t), (x_{t+1}, y_{t+1}), ..(x_{t+m}, y_{t+m})\}$ we sample from the heatmap (HM) using the trajectory predicted so that the offroad loss value for each prediction is:

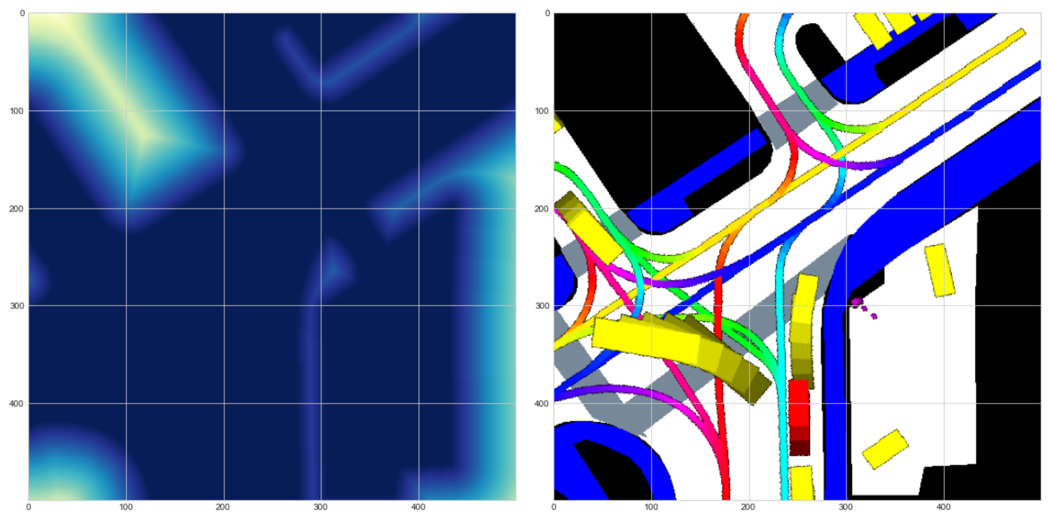
$$l^{offroad} = \sum_{i=0}^{m} HM(x_{t+i}, y_{t+i}), \tag{3.25}$$

Then, given M modes we average among the all prediction:

$$\mathcal{L}^{offroad} = \frac{\sum_{i=1}^{M} l_i^{offroad}}{M}, \tag{3.26}$$

Therefore the final loss will result as:

$$\mathcal{L}_{ij} = \mathcal{L}_{ij}^{class} + \alpha \sum_{m=1}^{M} I_{m=m*} L(\tau_{ij}, \widetilde{\tau}_{imj}) + \beta \mathcal{L}_{ij}^{offroad}, \tag{3.27}$$



(a) Drivable heatmap (b) Original frame

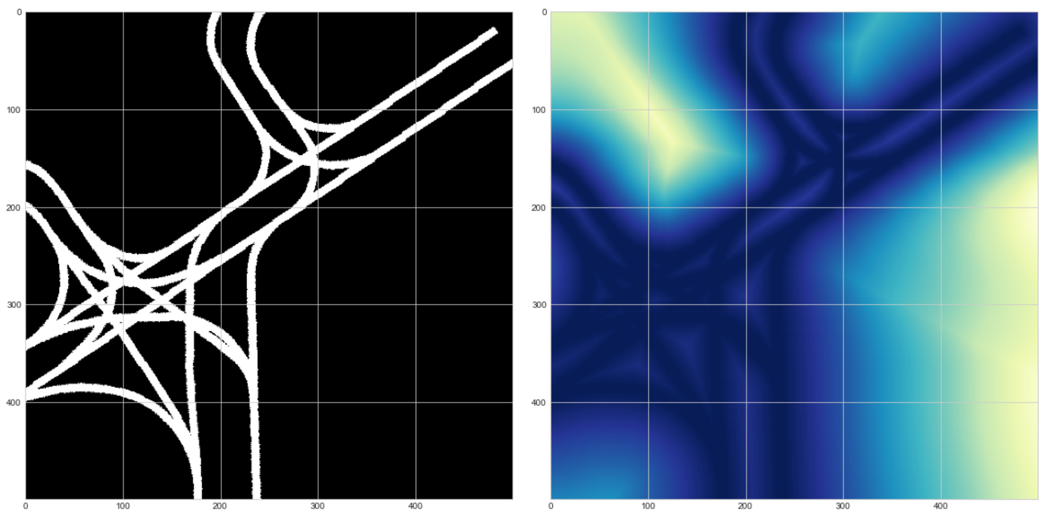
Figure 3.17: Off road heatmap

were β is an hyperparameter found empirycally.

3.3.1.1 Improvement: OffLane loss

The heatmap based on the drivable area already contains a lot of information for a safe prediction. However, we have noticed that sometimes, using just the mask from the drivable area could lead to a 0 OffRoad loss even if the action predicted was not feasible in real life. For this reason, we are introducing here a more fine-grained version generating the heatmap using the centre-lines. With this method the aim is to force the network to focus only on the possible maneuvers avoiding situations like the invasion of oncoming lanes that with the offroad standard the loss is not penalized.

The approach is very similar to the one previously described. As it is possible to see from Figure 3.18 we generate a mask, and then we calculate the heatmap. Comparing Figure 3.18b and Figure 3.17b is clear that using the lane heatmap we can restrict the area from actions that are "feasible", to the actions that are "feasible and safe" because the network is forced to put the focus on the lanes and learn the correct behaviour.



(a) Lane mask (b) Lane heatmap

Figure 3.18: Off Lane heatmap

Chapter 4

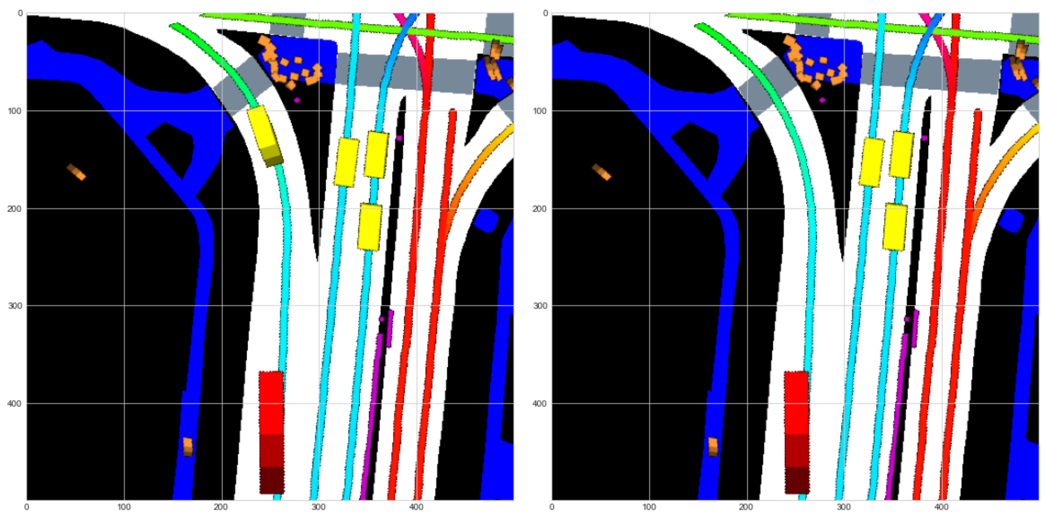
Implementation

We move now to the implementation chapter, where we cover here more the technical details regarding the implementation and some methods that we used to obtain the best from our models. Commonly among the all experiments, we used Python 3.7.7, as Deep Learning framework Pytorch 1.4.0 and as optimizer, we used Adam [12] with a learning rate of 0.0001 and weight decay of 0.01.

4.1 Frame Manipulation

As first, we have to ensure that the input of the model is correctly represented. Thus, analyzing the BEV with the Motional implementation, we noticed that the ego-vehicle was not represented in their frames. However, even if the task is to predict the surrounding agents and not the ego-vehicles, it is still an essential element of the traffic scenario that can influence the vehicles that we want to predict. For this reason, we decided to modify the frames inserting the EV in the representation. In fact, from the nuScenes documentation, we know that they used a Renault Zoe as a car and from the data that they have collected is possible to reconstruct the information relative to the ego-vehicle.

From Figure 4.1, we see that the yellow vehicle on the top left is not present in the original implementation, but with our implementation, the vehicle is correctly represented. Even if in this specific situation, the ego-vehicle does not directly impact the behaviour of the other agents, it is still information that can have an impact on the prediction and has to be encoded for completeness.



(a) Frame with ego vehicle (b) Frame without ego vehicle

Figure 4.1: Frame difference with and without egovehicle

4.1.1 Frame Pre-processing

Once the frame is correctly represented we have to extract the features from it. However, the feature extraction is a common part of the training process that we have to do in all the different architectures explained in 3.2 with the only exception of the DaNet in which we trim the *Resnet-18* one layer before to have more information to attend. Therefore, in all the cases, we always have a pre-trained backbone that extracts the features from the BEV image before the concatenation with the agent state vector. Thus, checking the inference time for *Resnet-18* we have noticed that the computational time for features extraction was taking a considerable amount of time. In fact, considering just a small set of 60 frames, the feature extraction from the backbone takes $6.66s$ while loading the same feature pre-processed takes just $0.015s$.

Hence, considering that we are not training the backbone, but we are only using it for extracting features, we decided to pre-compute those features to save time during the training process.

4.2 Early stopping

Selecting the best model while train can be not trivial, for this reason, we have created a mechanism of early stopping. At the end of every epoch, we evaluate the model on a subset of the training set using a heuristic that is so composed:

$$heuristic = 0.5 * MinFDE_1 + MinADE_5 \tag{4.1}$$

Where the metrics used in the heuristic are described in sec.5.1.1 and sec.5.1.3.

The idea behind those metrics is to extract a model that is capable of performing well on the single prediction $MinFDE_1$ and at the same time is also capable of understanding the multimodality of the traffic scenario $MinADE_5$. We propose this approach because the final goal is to predict the multimodality of the driving scenario and, focusing only on the final prediction could lead to a model that tends to average all prediction having the problem of the unimodal trajectory prediction (sec.2.3.2). The network is then tested on a sub-set of scenes selected using the adversarial validation technique explained in 4.2.1.

To end the training, we also used a system of patience; in this way, we can control how many epochs the model can continue training without improving his performance. In fact, at every epoch after the evaluation we control if the performance has improved, if not we increase the patience counter otherwise we set back to 0, once the counter reaches a specific limit N it means that the network has not learned anything useful on the past N epochs, for this reason, the training process is terminated.

4.2.1 Adversarial Validation

In order to extract the best model while training, we have to select carefully the scenes used for the evaluation at training time. For this reason, we use the adversarial validation technique to select the correct scenes.

Adversarial validation consists of training a classifier that can detect if the input data is coming from the training set or the validation set. Then, we use the trained model to select a subset of samples that are from the training set but are misclassified as part of the validation set or are likely to be within it, so this set of scenes is used during training time to evaluate the model. With this approach we are expecting to reduce overfitting on the training data because we can choose the model that will perform well on the train validation set and so, because of the adversarial validation is also likely to perform well on the validation set.

For our implementation, we trained a logistic regression model to learn the differences between the training scenes and the validation scenes. Then, we selected the scenes from the training set that were misclassified as part of the validation set, or they were likely to be in the validation set.

4.3 Batch Averaging

Another parameter that has deep impact on the training process is the dimension of the batch. Generally, a big batch lead to more a stable training and consequentially to better performance of the model. However, usually, the batch size is limited to the amount of space that is available on the GPU and so is not always possible to expand the batch dimension as our preference.

To overcome this problem, we use a technique called batch averaging that allows to treats batch separately and so expand the batch dimension without having an impact on the GPU memory.

```
accumulation_steps = 10
for i, (inputs, ground_truth) in enumerate(training_set):
    predictions = model(inputs)
    # Compute loss function
    loss = loss_function(predictions, ground_truth)
    # Normalize our loss
    loss = loss / accumulation_steps
    # Backward pass
    loss.backward()
    # Wait for several backward steps
    if (i+1) % accumulation_steps == 0:
        # Now we can do an optimizer step
        optimizer.step()
        # Reset gradients tensors
        optimizer.zero_grad()
```

Listing 1: Example of batch averaging [25].

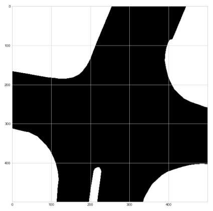
The basic idea is to process our standard batch as usual but, when it is time to perform the backward and the optimizer step operation we introduce a term called *accumulation_step* that as we can see from 1 normalize the loss at every backward step and allows the optimizer to perform a step only every *accumulation_steps* time. In this way, if we have the primary batch of 4 and the accumulate step of 10 it would be like training using a batch size of 40 but without having an impact on memory. In our implementation, we use a batch size of 32 and an accumulation step of 10.

4.4 Resampling

One more important aspect to consider is how the scenes are sampled at traning time. In 3.1.3.1 we have already discussed the data imbalance between the number of scenes which were relatively easy for the network to learn (folowing_lane) and more difficult scenarios (turning). Also from the first experiments conducted we noticed that our intuition was correct, in fact, the network was able to perform well on scenes in which the agents were simply going straight, but it is not able to understand more difficult scenes such as a turning scenario. For this reason, we decided to change the sampling methods using a weighting system; in this way, the more difficult scenes are more likely to be sampled than before. However deciding the complexity of a scene can be a non trivial task, so we implemented two different methods for weighting every scenes.

4.4.1 Lane Density

As the first criteria for sampling, we chose the lane density. We define lane density as the mean pixel value over the drivable area mask. So given drivable mask of (500,500) pixels, where the non-drivable are has value 0 and the drivable area as value 1, the lane density result in:



$$\Rightarrow density = \frac{\sum_{i=0}^{N} \sum_{j=0}^{N} p_{ij}}{N^2} \tag{4.2}$$

Then, analyzing the distribution over the all dataset at Figure 4.2 we see that the center of the distribution is at almost 0.3. Therefore, examining the scenes with that density (Figure 4.3), we notice that the complexity is relatively low and thus easy to learn for the network. Hence, we decide to set the threshold to divide simple and difficult scenes at 0.5.

4.4.2 Lane Curvature

The second approach is to use the lane curvature. In fact, we notice that the majority of the complex scenes are happening with an high lane curva-

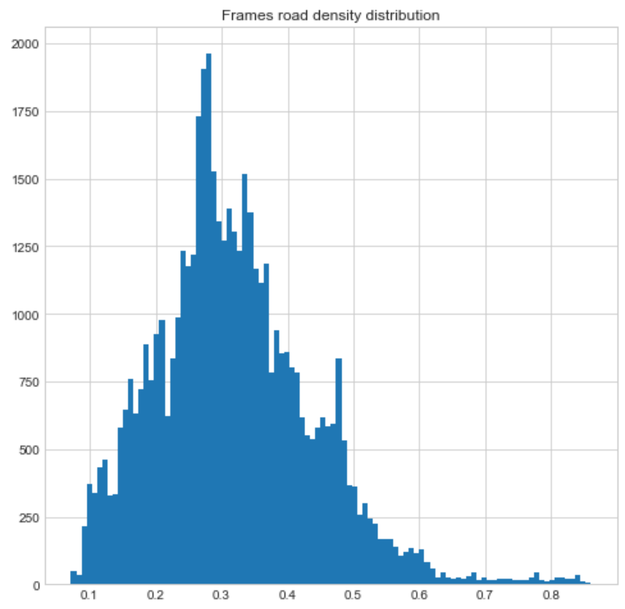
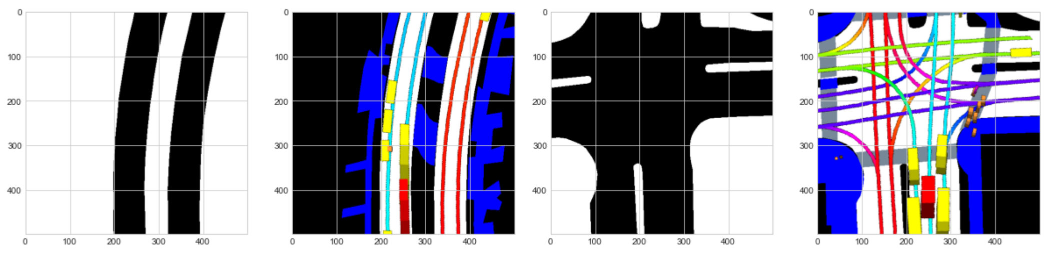


Figure 4.2: Lane density distribution over the full dataset



(a) Lane density 0.3 (b) Lane density 0.5

Figure 4.3: Scenes with different lane density

ture, for example turning in a crossing roads. Also from Figure 4.4 we can analyze the curvature distribution on the full dataset. Compared to the lane density approach, we notice that the distribution his much less uniformly distributed, this happens because the lane curvature is not able to capture as many details as the lane density about the scene complexity. However, we still believe that it is an interesting sampling method to analyze, since it directly gives us access to scenes where the vehicle is turning or doing more difficult manoeuvre than only going straight.

Thus, we decided to set the threshold at 0.012; this allows us to select scenes with a certain level of complexity as we can see from Figure 4.5:

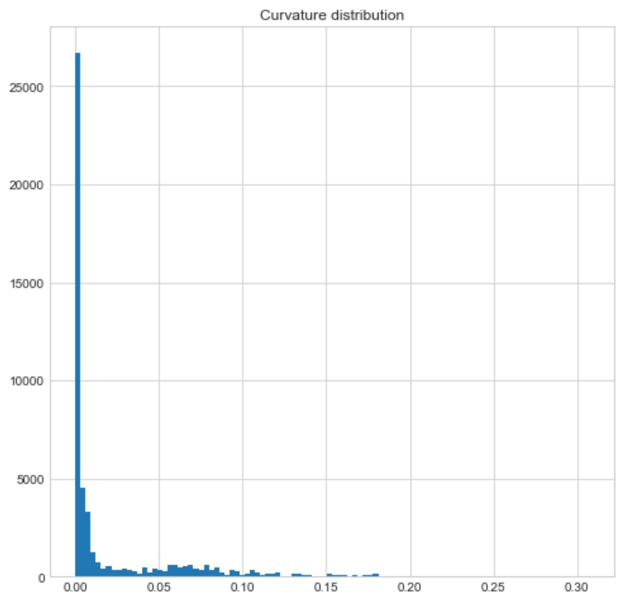
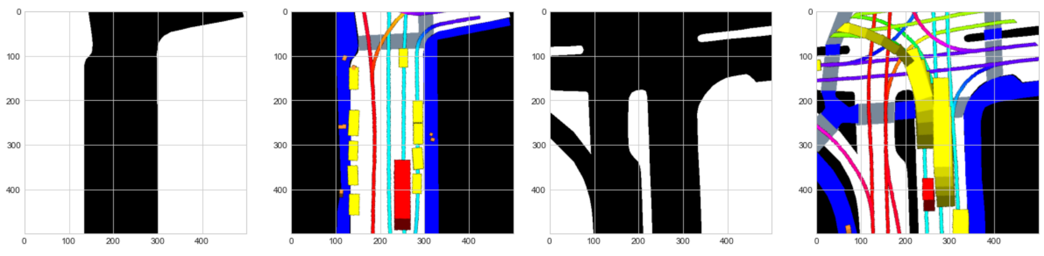


Figure 4.4: Lane distribution over the full dataset.



(a) Scene with curvature almost 0 (b) Scene with curvature at 0.012

Figure 4.5: Scenes with different lane curvature.

4.4.3 Weight Computation

Once the most relevant scenes are selected, we have to compute the weight in order to let the weight sampler know which scene extract more frequently.

Thus, knowing the threshold described in 4.4.2 and 4.4.1, we can count the number of scenes that are below or above the threshold. After that, knowing the total number of samples N and the number of sample for each class C_{above} and C_{below} we set the weight for the samples that are above the threshold simply as:

$$w_{above} = \frac{N}{C_{above}}, \tag{4.3}$$

and :

$$w_{below} = \frac{N}{C_{below}}, \tag{4.4}$$

for those one below the threshold. Finally, those weights are used by the weight sampler at training time to sample from the multinomial distribution.

4.5 Upsampling

The last technique that we used to improve the performance is data upsampling, which is commonly used to have more stable performances and to reduce overfitting. For this reasons we implemented upsampling methods in two different parts of the training process.

4.5.1 Upsample Data

As first, insipired by [2] were the authors used the perturbation of the input data to generate more samples obtaining consequentially a model capable of generalize in every situation. We used a similar approach, sampling the agent state vector from a Gaussian distribution. $\mathcal{N}(\mu, \sigma^2)$ with $\sigma = 0.5$. Thus, the new features are:

$$\begin{bmatrix} vel \\ acc \\ \theta\_rate \end{bmatrix} \Rightarrow \begin{bmatrix} \mathcal{N}(vel, \sigma^2) \\ \mathcal{N}(acc, \sigma^2) \\ \mathcal{N}(\theta\_rate, \sigma^2) \end{bmatrix}, \tag{4.5}$$

In this way, we noticed that the model generalize better and extract only the relevant features without overfitting.

4.5.2 Upsample Agent Features

The second upsample technique comes after analyzing the baselines network structures. From Figure 3.5 is clear that there is a significant imbalance between the features generate from the backbone and the agent state vector. In fact, the agent state vector features are only 3 while using, for example, a backbone as ResNet-18 the number of features extracted is 512. It is evident that with such huge imbalance, the network might be unable to pay attention to the features that are coming from the TV. For this reason, we decided to upsample those features using a fully connected layer as in Figure 3.10. In this way, the network is also able to learn a better representation of the agent features. After having conducted some experiments, we have found that the upsample factor of 10 was giving us good results.

Chapter 5

Evaluation

Model evaluation is a non-trivial task; in fact, choosing the metrics that can capture the performance is not always easy. Thus, in order to evaluate our models, we choose the following metrics that are already commonly used in the behaviour prediction task [3, 22].

5.1 Metrics

Considering that the task is to predict the multimodal behaviour, for all the metrics below described, we are going to differentiate each one depending on the number of future trajectories considered, in our case, 1,5 and 10 modes. Which means that for every metrics, we will have three value for considering the n most likely predictions.

5.1.1 MinADE

The minimum Average Displacement Error measures the average of pointwise L2 distance between the predicted trajectory and the ground truth. With this metric, we can understand if the prediction is correct along the entire trajectory,it is then computed as follows:

$$MinADE_k = \min_{y' \in P_k} (\frac{1}{t_f - t_{pred}}) \sum_{t=t_{pred}+1}^{t_f} ||\mathbf{y}^t - \mathbf{y}'^t||, \tag{5.1}$$

where P_k is the set of the most probable k trajectories.

5.1.2 MinFDE

Minimum Final Displacement Error is the L2 distance between the final points of the prediction and the ground truth. We use this value because sometimes it is useful just to know where the agent is going to be, and you are not interested in the full path. The metric is then computed as:

$$MinFDE_k = \min_{y' \in P_k} ||\mathbf{y}^{t_f} - \mathbf{y}'^{t_f}|| \tag{5.2}$$

5.1.3 Miss Rate

With the Miss Rate, we define a prediction a miss if the maximum pointwise L2 distance between the ground truth and the prediction is greater than 2 meters and can be computed as:

$$Miss_k = \begin{cases} 1 & min_{\mathbf{y'} \in P_k}(max_{t=t_{pred}}^{t=t_f}||\mathbf{y}^t - \mathbf{y'}^t||) \geq 2 \\ 0 & otherwise \end{cases} \tag{5.3}$$

5.1.4 OffRoadRate

Offroad Rate is defined as the fraction of trajectories that are not entirely contained in the drivable area of the map. In this case, we always consider all the trajectories predicted, thus is easier for a model with a lower number of modes to have a low OffRoadRate and vice-versa.

5.2 Evaluation

We now evaluate the performance of our models comparing the impact of each implementation described in 3.2.2 on the quality of the predictions. The models performances are measured on the validation set of nuScenes composed of 9041 samples where the network has to predict the possible futures over the next 6s. Then the predictions previously generated are processed computing the metrics up to the 10 most likely predictions. However, in this chapter, we reported only up to the 5 most likely predictions because there is not enough space on the text. For the complete tables, see Chapter .8.

5.2.1 Baselines

As first we evaluate the baselines described in the 3.2.1. In order to have a more fair comparison between the dynamic models and the Deep Learning models, we also report the metrics for the constant velocity dynamic model. Moreover, we decided to insert two different versions of MTP, one with 2 modes and another one with 10 modes.

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
oracle	**8.25**	**9.09**	9.09	**3.70**	3.70	**0.88**	0.88	**0.12**
cv	10.22	11.21	11.21	4.61	4.61	0.91	0.91	0.14
mtp (2)	8.97	11.49	7.41	4.87	3.23	0.93	0.88	0.15
mtp (10)	9.41	12.66	6.72	5.51	3.08	0.97	0.93	0.42
covernet	8.27	10.95	**5.55**	4.98	**2.80**	0.95	0.92	0.61

Table 5.1: Baselines model evalation

From 5.1 we notice that the oracle is capable of generating good predictions, however, because of the unimodal prediction, when we also consider multiple modes the model is not able to perform at the same level of the other models. Analyzing, the two MTPs we notice that the model with fewer modes is able to focus more on generating the best possible prediction and so the metrics such as MinFDEK1 and MinADEK1 are considerably lower than the counterpart with more modes. However, the model with 10 modes is able to better understand the all possibles scenarios, and so is capable to outperform the version with 2 modes when we consider multiple predictions.

Regarding CoverNet, because of his trajectory set is capable of performing well in the multimodal scenario. However, because of the trajectory set that does not consider the road structure, we can notice that is the model with the highest OffRoadRate.

5.2.2 Resampling

We consider now the impact of a custom resampling method on the baseline CoverNet and MTP(10). We examine both resampling methods described in 5.2.2. Thus, we have a version that uses the lane density 4.4.1 and another that uses the lane curvature 4.4.2 to set the weights of each scene.

Analyzing 5.2 and 5.3 we notice a clear improvement in performance after using the resampling methods previously described. Nevertheless, it is not possible to define a method that performs better than the other one. While MTP seems to be more affected by the curvature in every metrics considered in the table below.

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
mtp	9.41	12.66	6.72	5.51	3.08	0.97	0.93	0.42
mtp_curvature	**7.80**	**10.83**	**5.33**	**4.61**	**2.38**	**0.90**	**0.76**	**0.24**
mtp_density	8.11	11.10	5.68	4.74	2.57	0.92	0.86	0.43

Table 5.2: MTP resampling impact

Examining the CoverNet table, the situation is the opposite as the previously described. Density resampling seems to affect much more the performance while the curvature approach performs even slightly worse than the basic sampling method.

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
covernet	8.27	10.95	5.55	4.98	2.80	0.95	0.92	0.61
covernet_density	**7.90**	**10.77**	**4.95**	**4.79**	**2.51**	**0.94**	**0.92**	0.64
covernet_curvature	8.32	11.03	5.57	5.02	2.81	0.95	0.92	**0.57**

Table 5.3: CoverNet resampling impact

5.2.3 Memory

We start now modifying the input of the network using as additional information the past agent state vector. As already shown in 3.2.2.1, the network structures that we have implemented here can be considered as MTP-based structures in which the output is directly the coordinates plus their probabilities. From now on, we decided not to continue further development on CoverNet based models. Mainly because we believe that the MTP approach can learn better the different road dynamics, after all, the model is not bounded to a set of predefined trajectories that can limit the ability of the network to explore.

Therefore, from 5.6, we observe that the memory is playing a quite relevant role in the prediction. In fact, almost all the metrics involved they had a significant improvement over their memory-less counterpart.

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
mtp_curvature	7.80	10.83	5.33	4.61	2.38	**0.90**	**0.76**	**0.24**
mmtp_curvature	**7.78**	10.85	**5.24**	4.56	**2.35**	0.94	0.78	0.35
mtp_density	8.11	11.10	5.68	4.74	2.57	0.92	0.86	0.43
mmtp_density	7.91	**10.68**	5.62	**4.52**	2.57	0.92	0.85	0.44

Table 5.4: Memory performance impact

5.2.3.1 Dual Attention

Another modification that we propose is to attend the concatenated features, as explained in 3.10. In the table below, we notice that the attention layer helps to extract useful features from the concatenation between backbone features and agent state vector features, leading to a final better prediction.

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
mmtp_curvature	7.78	10.85	5.24	**4.56**	**2.35**	0.94	0.78	0.35
datt_mmtp	**7.77**	**10.77**	**5.12**	4.69	2.38	**0.91**	**0.75**	**0.20**

Table 5.5: Attention on feature concatenation impact

5.2.4 Loss

As described in 3.3 we decide to introduce an additional term in the loss in order to detect feasible trajectories. We are now analyzing the impact of the OffRoad loss.

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
datt_mmtp	7.77	10.77	5.12	4.69	2.38	0.91	0.75	0.20
datt_mmtp$_{offroad}$	**7.44**	**10.49**	**4.74**	**4.56**	**2.19**	**0.90**	**0.72**	**0.19**
datt_mmtp$_{lane}$	7.99	10.76	5.33	4.80	2.61	0.92	0.84	0.31

Table 5.6: Attention on feature concatenation impact

It is observable that the OffRoad loss has a profound impact on improving almost every aspect of the prediction, while the Lane loss has a negative impact on the performance. We believe that this behaviour might be caused due to the inability of the network to extract useful information from such restrictive loss. In fact, both GAT 5.7 and DaNet 5.8 approaches they have seen improvements using the Lane loss.

5.2.5 Graph Attention Network

We take now in consideration the Graph Attention Network for representing the nearby agents interaction. From the table 5.7, we can extract two useful information.

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
datt_mmtp$_{offroad}$	7.44	10.49	**4.74**	4.56	**2.19**	**0.90**	**0.72**	**0.19**
datt_mmtp$_{lane}$	7.99	10.76	5.33	4.80	2.61	0.92	0.84	0.31
gat_mtp(10)	8.34	12.03	5.05	5.07	2.33	0.91	0.74	0.28
gat_mtp$_{offroad}$(10)	8.02	10.95	5.64	4.78	2.55	**0.90**	**0.72**	0.20
gat_mtp$_{offroad}$(20)	9.64	12.89	6.20	6.08	3.19	1.00	0.97	0.43
gat_mtp$_{lane}$(10)	7.89	11.06	5.15	4.72	2.36	0.92	0.78	0.22
gat_mtp$_{lane}$(4)	**7.17**	**9.80**	5.05	**4.23**	2.27	0.91	**0.72**	0.24

Table 5.7: Graph attention performance

The first one is that the graph representation can interpret better a loss, such as a lane loss, in fact, there is a clear improvement in performance between Lane and OffRoad loss. The second one is that the number of agents considered has a deep impact on the performances (the number of agents considered is represented in brackets). We immediately see that if we ask the model to pay attention to too many agents, the model is not able to extract any more useful information. Therefore, we have found a good balance considering 4 agents.

5.2.6 Backbone Attention

We assess here the attention modules inserted after the backbone feature extractions described in 3.2.2.3. Where *m_danetmtp* is the DaNet version that loop through the past frames while *danetmpt* is the model in which we pass only one frame through the attention model.

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
danetmtp	10.35	12.25	9.54	5.48	4.23	0.97	0.92	0.29
m_danetmtp$_{offroad}$	9.94	12.12	8.31	5.71	3.88	0.97	0.91	0.20
m_danetmtp$_{lane}$	**8.91**	**11.46**	**6.80**	**5.30**	**3.18**	**0.95**	**0.86**	**0.17**

Table 5.8: Danet performance

From 5.8 is clear that inserting the time dependencies trough multiple frame processing, has helped to generate more reliable predictions. However, none of those two models has overall good performances but analyzing the predictions more in details we notice some useful findings that we will discuss in the next section and are worth to report.

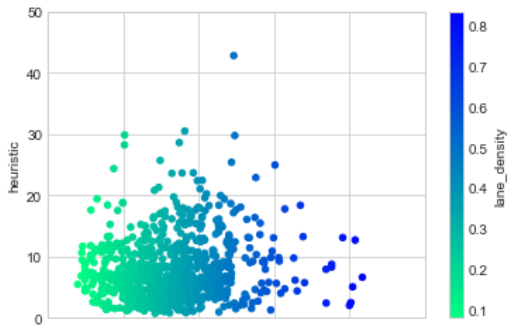
5.2.7 Performance on scenario

In order to understand more deeply how the different improvements have impacted the performance, it might be useful to try to visualize the performance among all dataset.

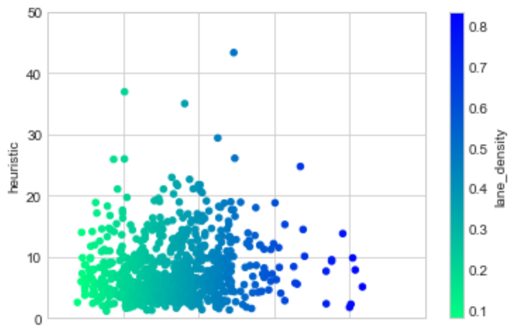
name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
dattmtp	6.80	9.63	**4.21**	4.22	**1.99**	**0.89**	**0.68**	0.21
gat_mtp	**6.59**	**8.88**	4.75	**3.86**	2.15	0.90	0.70	0.27
mmtp	7.15	10.18	4.42	4.56	2.06	0.93	0.76	0.22
m_danetmtp	8.45	11.13	6.16	5.20	2.88	0.95	0.84	**0.20**

Table 5.9: Low complexity scenes

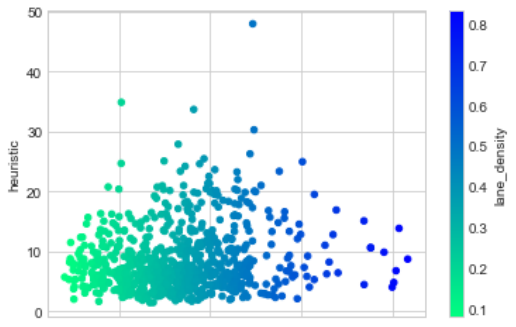
In Figure 5.1 we report the heuristic value for each scene, the points are colored using the lane density where the blue dots they represent the more



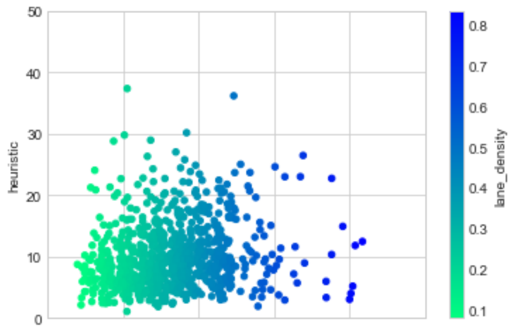
(a) Dual Attention



(b) Graph Attention Network



(c) Memory MTP



(d) Danet

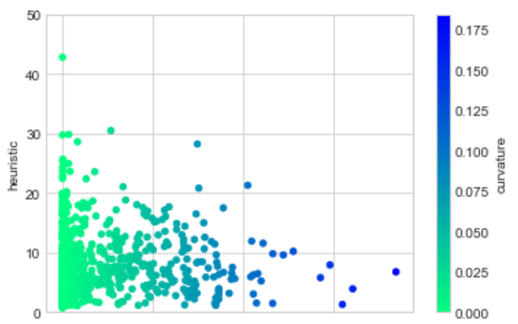
Figure 5.1: Heuristic metrics over lane density

complex scenario while the green dots are the simple one.

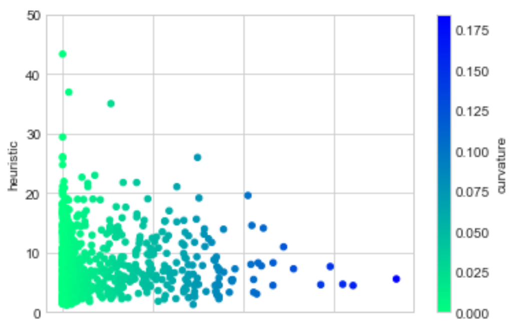
While in Figure 5.2 on the x-axis, we have the curvature to represent the scenes difficulties. In both the representations we notice a trend in which the DualAttention network tends to perform better in scenes with a low complexity while the model with the Graph attention network can perform better in scenes with high complexity.

Extracting the metrics separately for high complexity scene ($lanedensity > 0.5$) and low complexity scenes ($lanedensity < 0.3$) we observe a clear difference. From 5.9 and 5.10, we notice the ability of the Dual Attention and memory MTP to have good performances on low complexity scenes while the graph model outperforms the other approaches on complex scenes where the interaction between agents are fundamental.

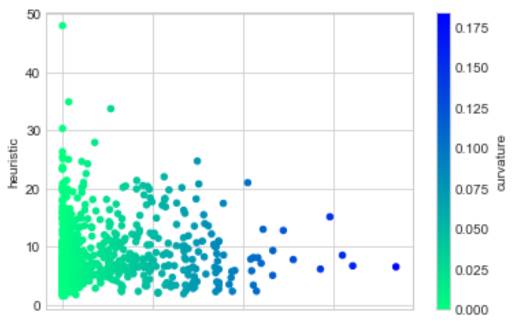
Those differences are caused because, with the graph, the model put more emphasis on the relationship between agents while the dual attention model we focus more on the agent state vector of the target vehicle. For this reasons



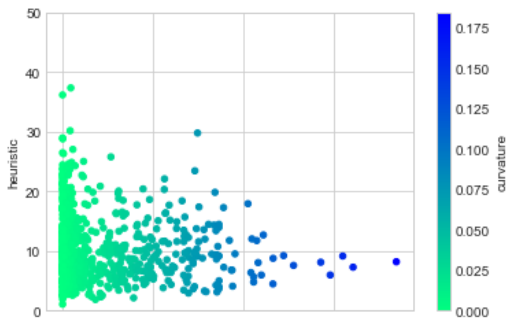
(a) Dual Attention



(b) Graph Attention Network



(c) Memory MTP



(d) Danet

Figure 5.2: Heuristic metrics over lane curvature

name	Heuristic	MinFDEK1	MinFDEK5	MinADEK1	MinADEK5	MissRateTopK_21	MissRateTopK_25	OffRoadRate
dattmtp	9.30	12.41	7.19	5.26	3.09	0.94	0.81	0.20
gat_mtp	**8.92**	**11.88**	**6.97**	**5.02**	**2.98**	**0.93**	**0.78**	0.24
mmtp	10.24	13.95	7.49	6.01	3.26	0.96	0.84	0.24
m_danetmtp	11.505	13.89	10.39	6.09	4.56	0.95	0.88	**0.19**

Table 5.10: High complexity scenes

we have such difference in performance depending on the scene structure.

Differently, the Danet approach is never really able to capture the information that is relevant for the prediction. In fact, from Figure 5.1d and Figure 5.2d we can observe the overall worst performances with the points that are more spread around.

However, from 5.10 and 5.9 we noticed that the Danet approach is always able to perform better in terms of OffRoadRate which means that despite the lower overall performances the model is able to understand better than the others the road structure.

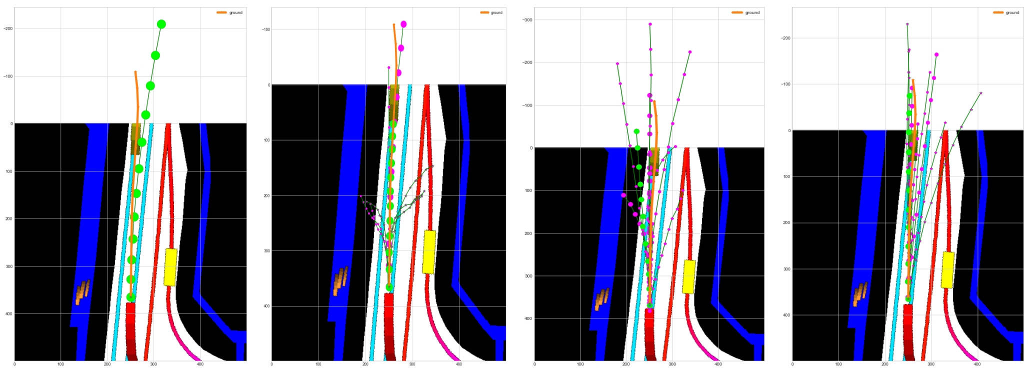
5.3 Anecdotal analysis

It makes sense now to analyze some particular scenes that are representative of the different improvements made with our modifications. In the following frames the predictions are represented with the purple dots of different dimension, where the dimension is directly proportional to their probability, also the most likely prediction is represented with a green color while the ground truth is the orange line.

5.3.1 Baselines

We can start briefly analyzing how the baselines prediction looks like compared to our modified models. Therefore, we report here the 10 most likely prediction for the baselines implemented.

From Figure 5.3 we notice that despite the lack of information coming from the road structure, the Oracle is capable of generating a plausible prediction. However, even though the model has access to the ground truth to choose the best prediction, none of the dynamic models is capable of capturing a correct predictions.



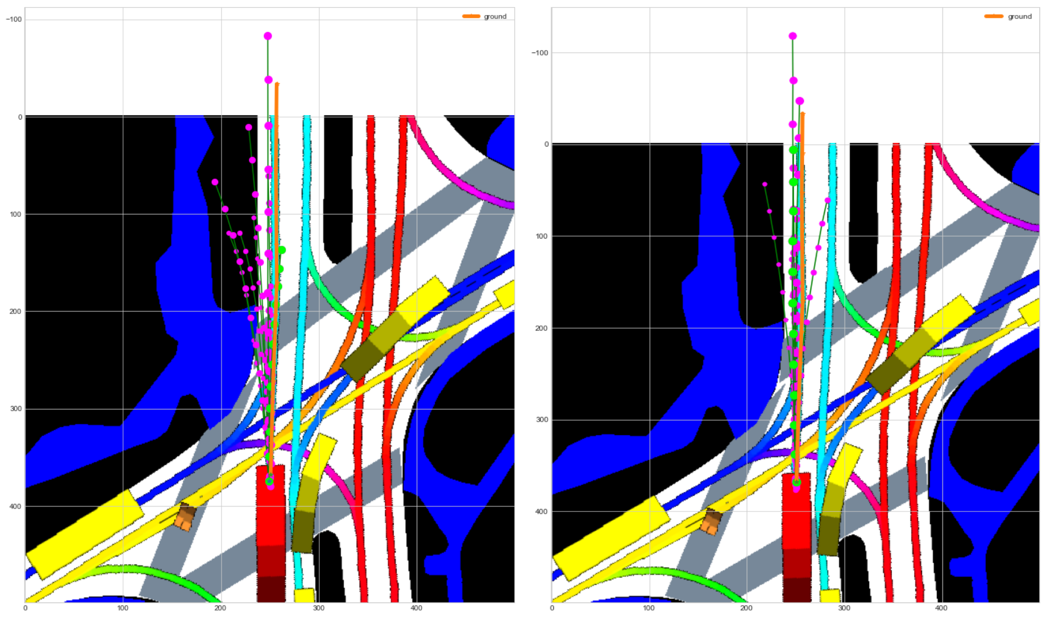
(a) Oracle (b) MTP (c) CoverNet (d) Gat MTP

Figure 5.3: Baselines comparison

For MTP, the model tries to generate some possible futures, but except for few modes that are relatively close to the ground truth, the others are almost unfeasible predictions both from a dynamic perspective and road structure perspective. Instead, CoverNet is almost capable of choosing the right trajectories, and also the others are relatively close to the real one, however, because of the trajectory set that limits the possible predictions the network is limited to choose between a predefined set that is general and not made for that specific situation. At the same time, Gat MTP (Figure 5.3d) predicts trajectories that are almost all possible futures and among those predictions is capable of selecting a mode that is representative of the ground truth.

5.3.2 Loss

Moving now to the loss impact, here we have two dual attention models that are the same except for the loss that they have used while training; one is trained with the OffRoad loss while the other one not. From Figure 5.4, we notice that the model trained with the additional loss term is now able to have a better understanding of the road structure predicting almost only feasible trajectories.

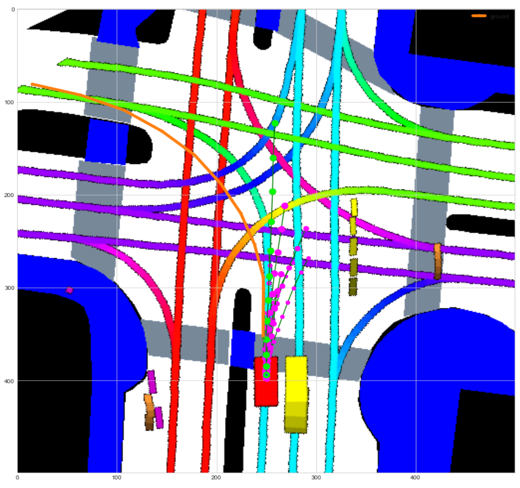


(a) Non Offroad (b) Offroad

Figure 5.4: Offroad Performance Difference

5.3.3 Gat agents

Another relevant difference to check visually is how the number of agents considered for Gat MTP impact the predictions. From Figure 5.5 is interesting to notice how the model trained with 10 agents wrongly focus on the


(a) 10 agents considered

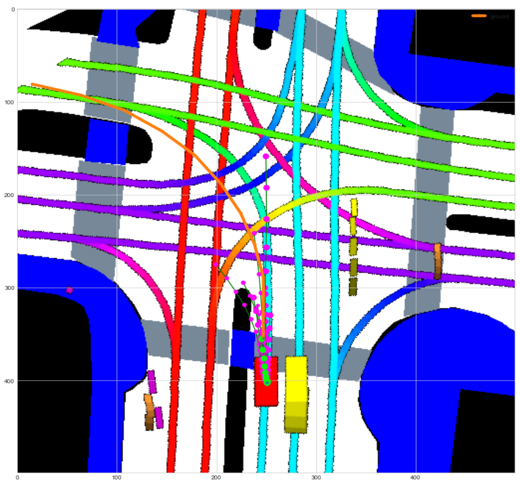
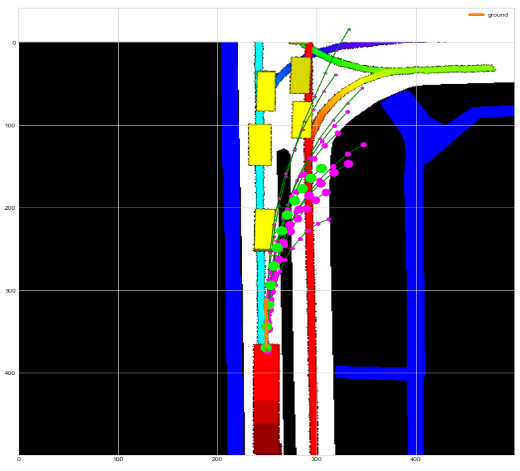

(b) 4 agents considered

Figure 5.5: Agent impact

vehicle that is coming from the right while the other one is able to ignore it and produce a prediction more realistic even if not optimal.

5.3.4 Backbone attention

It is now interesting to evaluate the models with the attention on the backbone. In fact, despite their poor performances, the model has interesting behaviours.


(a) Single DaNet

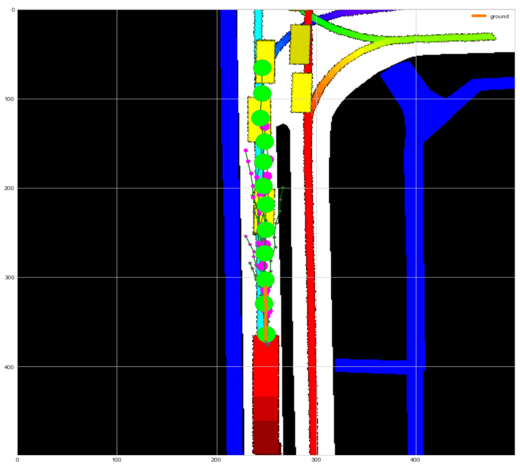
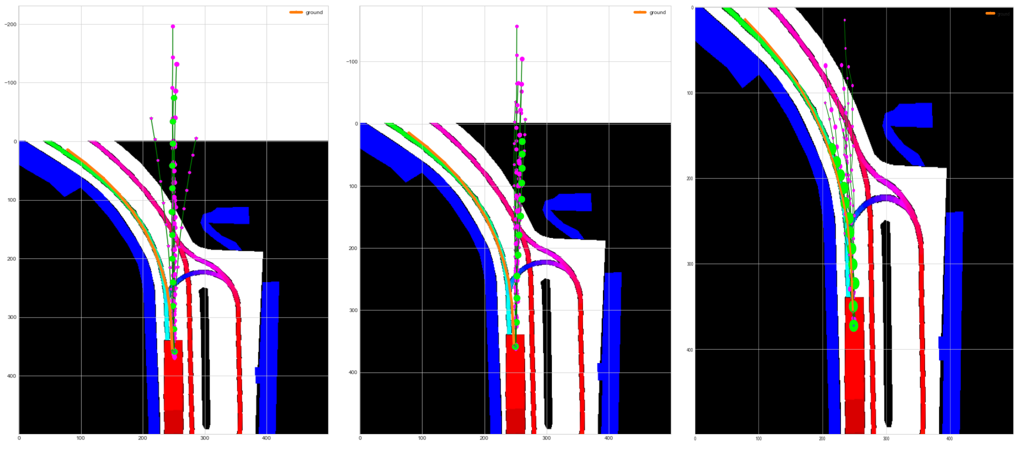

(b) Memory DaNet

Figure 5.6: Memory impact on DaNet

Especially in the non-iterative attention, the network has learned to always try to explore new paths on the map even if those ones are not feasible from a road structure point of view. From Figure 5.6 we can see that the single DaNet tries to explore and go to the new road segment while the memory counterpart has learned the different time dependencies and follows the most likely path.

5.3.5 Map relevant Scenes

Another aspect to examine is the level of understanding that the model has of the road structure. Analyzing multiple map relevant scenes, we have noticed that both the Graph and the Dual attention models tend to focus a lot on the agent state vector and not enough on the features that are coming from the backbone.



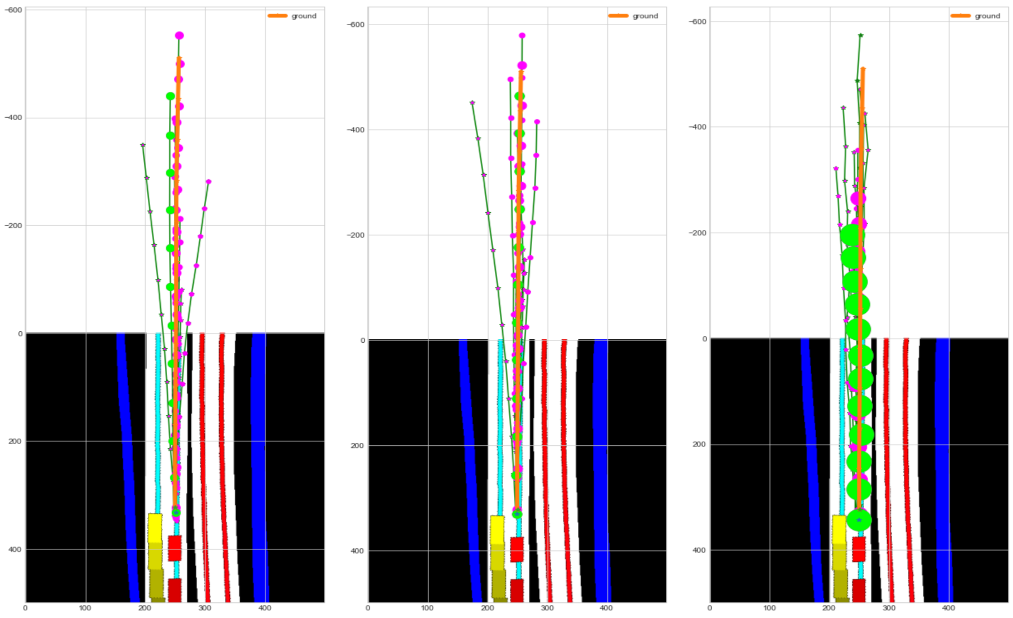
(a) Dual attention MTP (b) GAT MTP (c) Memory DaNet MTP

Figure 5.7: BEV relevant scene

For example, in Figure 5.7 we are in a quite difficult situation for the network. In fact, there is only one road to follow, the agent is going quite fast, and it has not started to turn yet. Therefore, to correctly predict the future, the network has to rely only on the information that is coming from the BEV analysis. Thus, it is quite clear that the only model that has understood the road structure is the one with the attention on the backbone.

5.3.6 Agent relevant Scenes

On the other hand, having the attention on the backbone, the model tends to not focus enough on the relevant features that are coming from the agent state vector. An example can be seen in Figure 5.8, where the situation is relatively simple; in fact, all the other models manage to predict the correct behaviour, while, the memory DaNet MTP model put too much attention on the BEV. For this reason, his prediction is accurate from a direction perspective. Still, it has failed to understand velocity and acceleration, and so the most likely trajectory is far from the ground truth.



(a) Dual attention MTP (b) GAT MTP (c) DaNet MTP

Figure 5.8: Agent state relevant scene

Chapter 6

Discussion

From the previous chapter, we are now able to understand the difficulties of the behaviour predictions. In facts, there are an infinite amount of possible scenarios that are changing rapidly and in an unpredictable way due to the human factor.

6.1 Scene based prediction

So, being able to understand which model suits in a specific scenario and why it performs better in certain situations became a critical task to obtain a reliable prediction.

Considering the previous chapter, we are able the understand a few things. From 5.2.2 it is clear that one of the aspects that affect more the performances is the scene that the network has to predict. In fact, showing more often the difficult scenes has improved in a significative way the performances. Thus, we understand that there are scenes that require a certain approach to improve and scenes that require another one.

Therefore, we propose some structural methods to improve the performances. In sec.5.2.3 we have injected information coming from the past; this allows the network to understand what was the state vector in the previous seconds and react as a consequence. With this approach, we notice a clear improvement in basically every scenario, but in particular, this improvement is more significant in scenes where the information that is coming from the vehicle are more relevant than the understanding of the BEV.

However, not every scene is simple enough such that the agent state vector is sufficient for a reasonable interpretation of the situation. For this reason, we decided to include also the nearby agents as input to the network, and we let a graph attention network extract useful information out of it. This approach is particularly useful because, with nuScenes, we are working on an urban driving scenarios which are very complex, and the interaction between agents is a crucial component. In fact, from 5.2.5 is clear that the network can extract useful information from the other agents allowing the model to generate a more reliable predictions.

Although, an interesting analysis comes from 5.2.7 where is evident the different performances of each model depending on the scenario that we are considering. From, the two tables 5.9 and 5.10 we notice how the graph model is able to perform better on complex scenarios while both the memory MTP and the dual attention models are able to outperform on less complex scenes, this difference can also be seen in Figure 5.1 and Figure 5.2. This discrepancy in performance proves what we explained before about the different representation between Dual Attention MTP and Graph Attention MTP, where the first one is capable of a better understanding of the vehicle kinematic model and the second one is capable of capture relevant interactions between different agents.

6.2 Road Structure

Another relevant component of a prediction model is a the ability to extract useful information from the BEV. Thus, we propose two approaches to focus more on the BEV. As first we introduce the OffRoad/Lane loss; that penalize prediction that are not inside the drivable area forcing in this way the network to predict feasible futures as we have seen in Figure 5.4.

The second approach that we propose is the use of an attention model on the backbone using the DaNet [10] network in our model. From fig.5.7 it is evident that the attention on the backbone has helped the model to have a better understanding of the road. In fact, in a challenging situation such as the one the previously mentioned, the model has to rely on the information that is coming from the BEV, and it is clear that both the dual attention model and the graph model are not able to extract enough useful information from it.

6.3 Future paths

As previously described, every approach that we have implemented has positive and negative aspects. But, we have learned that scenes are very different among themselves and for this reason is very challenging to build a model that is able to perform equally well in every possible scenario. In fact, we have already seen some models that are able to outperform other models in a certain type of scene and at the same time being outperformed in another type of scene.

Thus, an interesting approach to follow could be a multi-task approach, in this way the network will be able to react differently depending on the situation that is facing because it will have a different parts of the network that are specialized for each particular task.

Chapter 7

Conclusions

With the adoption of self-driving cars, we expect to increase safety and at the same time to reduce travel times avoiding situations such as traffic jams. Therefore, to ensure an adequate level of safety, the vehicle should not be able to see only the other vehicles. However, it should also be able to predict the future movement of other agents.

Hence, also considering that our focus is on urban driving scenarios, the predictions follows a multimodality setting in which multiple possible futures are predicted along with their probabilityies. As discussed in Chapter 2, we evaluate also other different prediction representation. However, as we have seen, those methods fails to understand the scenes complexity of an urban driving scenario.

Then using as a starting point the two baselines MTP [5] and CoverNet [22], we can summarize our contribution into four main parts.

At first, we notice analyzing the data distribution, that despite nuScenes being based in an urban driving scenario, there is a relevant imbalance between scenes that are easy to learn for the model and scene that are not. So, using custom resampling methods, we have been able to balance the situation allowing the model to adapt better to each situation.

From the baselines, there were also problems of non used information. For example, in none of the model implemented, neither the data from the past nor the data form nearby agents is used. Therefore, as second contribution, we propose a Transformer attention layer to attend past agent state vector. This approach has shown to improve significantly the performance raising the level of understanding of the network regarding the kinematic of the vehicle.

As third improvement, we included the surrounding agents in the model structure, to do that, we have implemented a graph attention layer that has proven to be able to capture the interactions among the vehicles, increasing the accuracy of the prediction, especially in those scenes where the interactions are a determining component.

The last contribution concerns the level of understanding that the model has of the road structure. First we have implemented an additional loss term called OffRoad loss which penalizes forecasts that are outside the road structure. While, the second approach introduces an attention module on the backbone features. Both methods have been shown to improve performance in challenging situations where the agent state vector was not relevant for prediction and the road structure played a central role.

In summary, we showed the heterogeneous nature of the behavior prediction problem and demonstrated that by using attention methods, off-road losses and customized sampling methods, the network learned to better adapt to each scenario by generating more feasible predictions.

Bibliography

[1] Altché, F., and de La Fortelle, A. An LSTM network for highway trajectory prediction. *CoRR abs/1801.07962* (2018).

[2] Bansal, M., Krizhevsky, A., and Ogale, A. S. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *CoRR abs/1812.03079* (2018).

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

[4] Casas, S., Luo, W., and Urtasun, R. Intentnet: Learning to predict intention from raw sensor data. In *Proceedings of The 2nd Conference on Robot Learning* (29–31 Oct 2018), A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87 of *Proceedings of Machine Learning Research*, PMLR, pp. 947–956.

[5] Cui, H., Radosavljevic, V., Chou, F., Lin, T., Nguyen, T., Huang, T., Schneider, J., and Djuric, N. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. *CoRR abs/1809.10732* (2018).

[6] Deo, N., and Trivedi, M. M. Convolutional social pooling for vehicle trajectory prediction. *CoRR abs/1805.06771* (2018).

[7] Ding, W., Chen, J., and Shen, S. Predicting vehicle behaviors over an extended horizon using behavior interaction network. *CoRR abs/1903.00848* (2019).

[8] Djuric, N., Radosavljevic, V., Cui, H., Nguyán, T. T. V., Chou, F.-C., Lin, T. H., and Schneider, J. G. Short-term motion prediction of traffic actors for autonomous driving using deep convolutional networks. *arXiv: Learning* (2018).

[9] European Road Safety Observatory. Annual accident report 2018. `https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/pdf/statistics/dacota/asr2018.pdf`.

[10] Fu, J., Liu, J., Tian, H., Li, Y., Bao, Y., Fang, Z., and Lu, H. Dual attention network for scene segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 3146–3154.

[11] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR abs/1512.03385* (2015).

[12] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization, 2014.

[13] Kosaraju, V., Sadeghian, A., MartÃn-MartÃn, R., Reid, I., Rezatofighi, S. H., and Savarese, S. Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks, 2019.

[14] Krajewski, R., Bock, J., Kloeker, L., and Eckstein, L. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)* (2018), pp. 2118–2125.

[15] LaValle, S. M. *Planning Algorithms.* Cambridge University Press, USA, 2006.

[16] Lee, D., Kwon, Y. P., McMains, S., and Hedrick, J. K. Convolution neural network-based lane change intention prediction of surrounding vehicles for acc. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)* (2017), pp. 1–6.

[17] Lefevre, S., Vasquez, D., and Laugier, C. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal 1* (07 2014).

[18] Li, X., Ying, X., and Chuah, M. C. Grip: Graph-based interaction-aware trajectory prediction. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* (2019), pp. 3960–3966.

[19] Messaoud, K., Deo, N., Trivedi, M. M., and Nashashibi, F. Trajectory prediction for autonomous driving based on multi-head attention with joint agent-map representation, 2020.

[20] Mozaffari, S., Al-Jarrah, O. Y., Dianati, M., Jennings, P., and Mouzakitis, A. Deep learning-based vehicle behaviour prediction for autonomous driving applications: A review, 2019.

[21] Park, S. H., Lee, G., Bhat, M., Seo, J., Kang, M., Francis, J., Jadhav, A. R., Liang, P. P., and Morency, L.-P. Diverse and admissible trajectory forecasting through multimodal context understanding, 2020.

[22] Phan-Minh, T., Grigore, E. C., Boulton, F. A., Beijbom, O., and Wolff, E. M. Covernet: Multimodal behavior prediction using trajectory sets, 2019.

[23] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. Imagenet large scale visual recognition challenge. *CoRR abs/1409.0575* (2014).

[24] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR abs/1801.04381* (2018).

[25] Thomas Wolf. Training neural nets on larger batches: Practical tips for 1-gpu, multi-gpu distributed setups. `https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255`.

[26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2017.

[27] VeliÄkoviÄ, P., Cucurull, G., Casanova, A., Romero, A., LiÃ^2, *P., andBengio, Y. Graphattentionnetworks*, 2017.

[28] Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. Self-attention generative adversarial networks, 2018.

[29] Zyner, A., Worrall, S., and Nebot, E. A recurrent neural network solution for predicting driver intention at unsignalized intersections. *IEEE Robotics and Automation Letters 3*, 3 (2018), 1759–1764.

[30] Zyner, A., Worrall, S., and Nebot, E. M. Naturalistic driver intention and path prediction using recurrent neural networks. *CoRR abs/1807.09995* (2018).

Chapter 8

Appendix

In the next page are reported all the metrics up to the 10th prediction. The table is rotate for make it more readable.

name	Heuristic	MinFDEK1	MinFDEK5	MinFDEK10	MinADEK1	MinADEK5	MinADEK10	MissRateTopK_21	MissRateTopK_25	MissRateTopK_210	OffRoadRate
oracle	8.25	**9.09**	9.09	9.09	**3.70**	3.70	3.70	**0.88**	0.88	0.88	**0.12**
cv	10.22	11.21	11.21	11.21	4.61	4.61	4.61	0.91	0.91	0.91	0.14
covernet	8.27	10.95	5.55	4.21	4.98	2.80	2.39	0.95	0.92	0.92	0.61
mtp (2)	8.97	11.49	7.41	7.41	4.87	3.23	3.23	0.93	0.88	0.88	0.15
mtp (10)	9.41	12.66	6.72	6.31	5.51	3.08	2.91	0.97	0.93	0.93	0.42
mtp_curvature	7.80	10.83	5.33	4.44	4.61	2.38	2.06	0.90	0.76	0.72	0.24
mtp_density	8.11	11.10	5.68	4.56	4.74	2.57	2.21	0.92	0.86	0.85	0.43
covernet_curvature	8.32	11.03	5.57	4.63	5.02	2.81	2.57	0.95	0.92	0.92	0.57
covernet_density	7.90	10.77	4.95	3.84	4.79	2.51	2.22	0.94	0.92	0.92	0.64
mmtp_curvature	7.77	10.74	5.32	4.20	4.51	2.40	2.00	0.90	0.76	0.70	0.43
mmtp_density	7.91	10.68	5.62	5.26	4.52	2.57	2.44	0.92	0.85	0.84	0.44
datt_mmtp	7.77	10.77	5.12	3.90	4.69	2.38	1.90	0.91	0.75	0.66	0.20
datt_mmtp$_{offroad}$	7.44	10.49	**4.74**	3.66	4.56	**2.19**	1.81	0.90	**0.72**	0.65	0.19
datt_mmtp$_{lane}$	7.99	10.76	5.33	4.53	4.80	2.61	2.33	0.92	0.84	0.81	0.31
gat_mtp$_{lane}$(4)	**7.17**	9.80	5.05	**3.62**	4.23	2.27	**1.76**	0.91	**0.72**	0.63	0.24
gat_mtp$_{offroad}$(10)	8.02	10.95	5.64	4.12	4.78	2.55	1.95	0.90	0.72	**0.61**	0.20
gat_mtp (10)	8.34	12.03	5.05	4.18	5.07	2.33	2.00	0.91	0.74	0.68	0.28
gat_mtp$_{lane}$(10)	7.89	11.06	5.15	3.89	4.72	2.36	1.88	0.92	0.78	0.65	0.22
gat_mtp$_{offroad}$(20)	9.64	12.89	6.20	6.07	6.08	3.19	3.13	1.00	0.97	0.97	0.43
danetmtp	10.35	12.25	9.54	7.20	5.48	4.23	3.30	0.97	0.92	0.88	0.29
m_danetmtp$_{offroad}$	9.94	12.12	8.31	6.38	5.71	3.88	3.08	0.97	0.91	0.87	0.20
m_danetmtp$_{lane}$	8.91	11.46	6.80	5.24	5.30	3.18	2.58	0.95	0.86	0.80	0.17