Deep Learning Based Motion Forecasting for Autonomous Driving

A Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Rodney Gracian Dsouza, B.Tech

Graduate Program in Department of Electrical and Computer Engineering

The Ohio State University

2021

Master's Examination Committee:

Wei-Lun Chao, Advisor

Irem Eryilmaz

Abstract

Autonomous driving (AD) is a promising technology that has grown into one of the primary areas of interest for the controls and machine learning research communities and commercial companies in the current decade. Solving this challenging problem can have significant impact on society and in the way we commute and interact with our surrounding environment. Studies have shown that self-driving vehicles could help significantly reduce loss of life due to road accidents and also reduce carbon emissions. Simply put, AD can make travelling from one point to another more efficient and eco-friendly. There's been a recent boom in number of research labs and companies working on this problem which can be accredited to the success of deep neural networks in solving challenging tasks involving learning such as image recognition and language translation. This has driven the applied machine learning community to implement the state of the art techniques in the field to the problem of self-driving. Although, this problem turns out to be much more complicated than the former due to the amount of uncertainty in the dynamic environment and the decision making involved. Nevertheless, there has been a significant progress in the field in the area of object tracking, motion forecasting and path planning and the self-driving pipeline is rapidly evolving.

In this thesis, one such module of the self-driving pipeline known as motion forecasting is tackled. Motion forecasting is tasked with predicting the future states of all

the agents in a given scene that affect the future trajectory of the self-driving agent. This component takes input from the object detection and tracking module and predicts the future trajectories of all agents that affect the behaviour of the self-driving ego agent. The output from this module helps the ego agent perform path planning efficiently to drive itself from origin to destination. The architecture for the model proposed in the thesis is built on deep learning based neural networks as they are appropriate for this data-driven learning task.

After discussing different approaches to solving the problem in hand from the literature survey and highlighting some related work, the thesis focuses mainly on demonstrating two novel proposed architectures based on the encoder-decoder design, discussing the methodology behind the design along with the merits and demerits. The proposed design includes elements such as spatio-temporal encoding to efficiently learn the static map and temporal agent states, agent interaction modeling with different attention mechanisms to capture interdependence between various agent behaviors and specialized basis based trajectory decoders to precisely estimate the future trajectories of the agents. This is followed by a series of experiments and results that demonstrate the incremental improvement made in the model design to achieve state of the art results and the performance of the models on a popular motion prediction dataset. Additionally, a meta-learning based test time adaptation technique is proposed to perform test-time single shot correction of the predictions for newly encountered driving scenarios.

Solving the problem of motion forecasting is a major step towards achieving Level 5 self-driving, and the proposed models in this thesis are a small contribution to the progress to be made in solving this challenging tasks.

This is dedicated to my parents, siblings and grandparents for their constant support

Acknowledgments

This masters thesis was completed as a part of the Masters of Science degree requirement in Electrical and Computer Engineering with specialization in Computer Vision and Machine Learning. The research pertaining to the thesis was conducted as a part of the Machine Learning as Basis (MLB) lab under Dr. Wei Lun Chao, an assistant professor in the Department of Computer Science and Engineering (CSE) at The Ohio State University. Several people had a strong influence on the successful completion of this thesis project and I would like to extend my heartful gratitude.

Firstly, I would like to thank my advisor Dr. Wei Lun Chao for accepting me into his lab although I was from a different department. His constant guidance and innovative ideas during our meetups shaped the direction of the research and helped me learn the right approaches and mindset to doing impactful research. I would also like to thank all my lab colleagues for their contribution and for our informative discussions during reading group sessions which had me updated on current research directions in Machine Learning. I wish them the best in their doctorate program. Additionally, I extend my gratitude to Qianli Feng, a doctorate student under Dr. Aleix Martinez from my department with whom I had worked on an interesting project on Human Activity Recognition during my first year which influenced me to dive deep into this promising field. I also thank my department advisor Dr. Irem Eryilmaz for

her support throughout the project and for allowing me to conduct cross-department research.

My family has always been my support pillar and they inspire me to explore new possibilities and work towards becoming a better individual. Having had a rural upbringing, I'm indebted to my parents for their sacrifice during our tough times which had a significant impact on my career path. I thank the almighty for my two lovely siblings, with their quirks always making me smile. I would also like to thank my grandparents and other family members for their support. My sincere gratitude to all my friends and roommates from university and prior workplaces for helping me grow as an individual and for our riveting conversations.

Rodney Gracian Dsouza

Apr 11, 2021

Vita

October 11, 1993	Born - Udupi, India
2015	B.Tech. Electronics and Communication Engineering
2015-2019	Member of Technical Staff, Tonbo Imaging
2021-present	Graduate Teaching Associate, CSE Department, The Ohio State University.

Fields of Study

Major Field: Electrical and Computer Engineering

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Road safety has been a major concern since the commercialization of automobiles in the early years of the twentieth century. With the progress made in road infrastructure and vehicle affordability, the number of privately owned vehicles has increased exponentially especially in densely populated countries like China and India as seen in Figure 1.1. Although, there has been significant innovation in improving road safety in the recent decades with safe driving regulations and in-vehicle improvements such as air bags, the infrastructure and safety standards have just not been able to keep up with this exponential increase and the statistics on road accidents based fatalities speaks to this imbalance.

In recent decades, Autonomous Driving (AD) has emerged as one such technology-driven solution that might help cater to the road safety issue along with other benefits such as congestion control and environmental gains. Although this solution is promising, the challenges involved in realising this technology are manifold due to its complexity. Recent advances in machine learning with neural networks have reestablished the possibility of fully realizing this technology in the coming years. There are numerous companies like Argo, Tesla, Waymo and Cruise that have already made significant progress in making autonomous driving a reality. Although a fully self-driving car referred to as Level 5 in AD terminology is probably a decade away, there

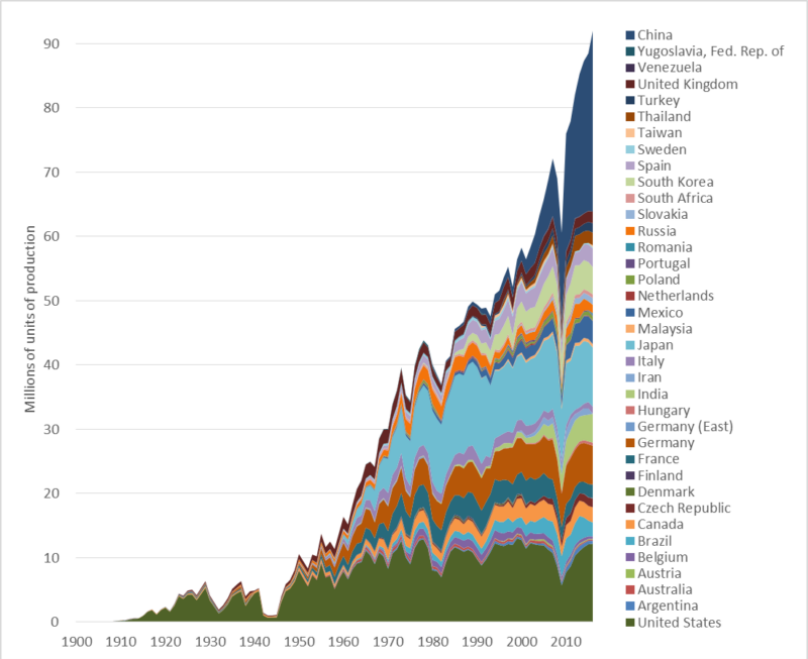

Figure 1.1: Global automobile production (cars, buses and trucks) per year [25]

has been significant progress made in collision avoidance systems and driver assistance/monitoring systems to improve road safety. The incremental progress made in different areas related to AD are stepping stones to achieving a fully autonomous vehicle and its future sure is promising.

This thesis primarily discusses the machine learning based techniques to achieve AD. The machine learning based self-driving pipeline can be broadly divided into the following five sequential areas as seen in Figure 1.2:

1. Sensor Fusion
2. Perception (Detection and Tracking)
3. Motion Forecasting

4. Trajectory Planning

5. Actuation and control

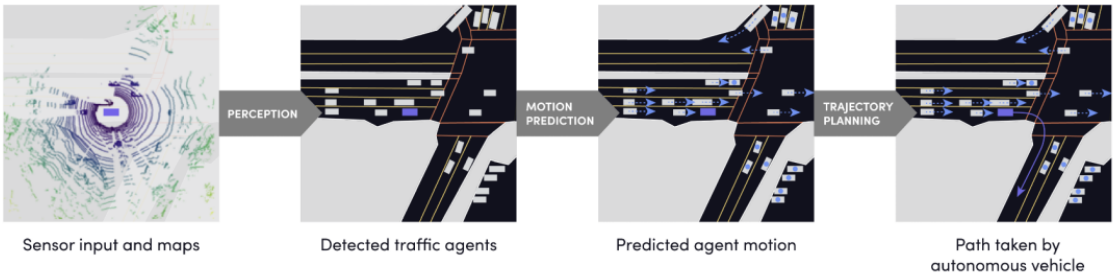

Figure 1.2: Sequential self-driving pipeline [21]

This thesis focuses on the area of Motion Forecasting, discussing the various approaches and related work followed by the proposal of a novel forecasting architecture based on deep learning, achieving state of the art results on a popular AD dataset. Before delving into this core topic of discussion in the following chapters, this chapter first provides a background on Autonomous Driving as a whole, discussing the history and terminologies involved, followed by Motion Forecasting problem formulation and its impact on the self-driving pipeline.

1.1 Background on Autonomous Driving

The first demonstration of an autonomous vehicle dates back to 1925, with the Houdina Radio Control demonstrating the 'American Wonder', which was a remotely controlled vehicle that traveled along Broadway in New York City trailed by an operator in another vehicle [23]. This was followed by more prototypes by General

Motors and RCA Labs although all these vehicles relied on dedicated infrastructure. The first self-driving prototypes to not rely on any infrastructure was demonstrated in 1986 by the Navlab team from CMU in the US as well as Ernst Dickmanns's team at the Bundeswehr University Munich in Germany[23]. Although the approaches used were contrary with the former using a imitation learning approach and the latter using a more modular approach. The success of these two projects motivated more research in this area, with DARPA announcing the Darpa Grand Challenge in 2004. Companies like Google and Audi jumped in on this task in the same decade. In 2012, the KITTI Vision Benchmark 3 was released [17]. For the first time, researchers around the globe were able to evaluate their progress on various self-driving perception tasks, which included reconstruction, motion estimation, and object recognition in a fair and objective manner. At the same time, deep learning started to revolutionize many fields, including computer vision and robotics, which laid the foundations for significant improvements in accuracy, robustness, and run-time of the perception components of self-driving vehicles. In 2014, the Society of Automotive Engineers (SAE) released their classification of autonomous driving systems into 6 SAE levels of autonomy, ranging from Level 0 (no autonomy) to Level 5 (full autonomy)[23]. Figure 1.3 shows the updated version of these levels followed by Table 1.1 which highlights the features available under each level.

In the following section, the various subsystems involved in the functioning of a self-driving vehicle as seen in Figure 1.4 are discussed. The critical technological subsystems to enable full self-driving can be defined as follows:

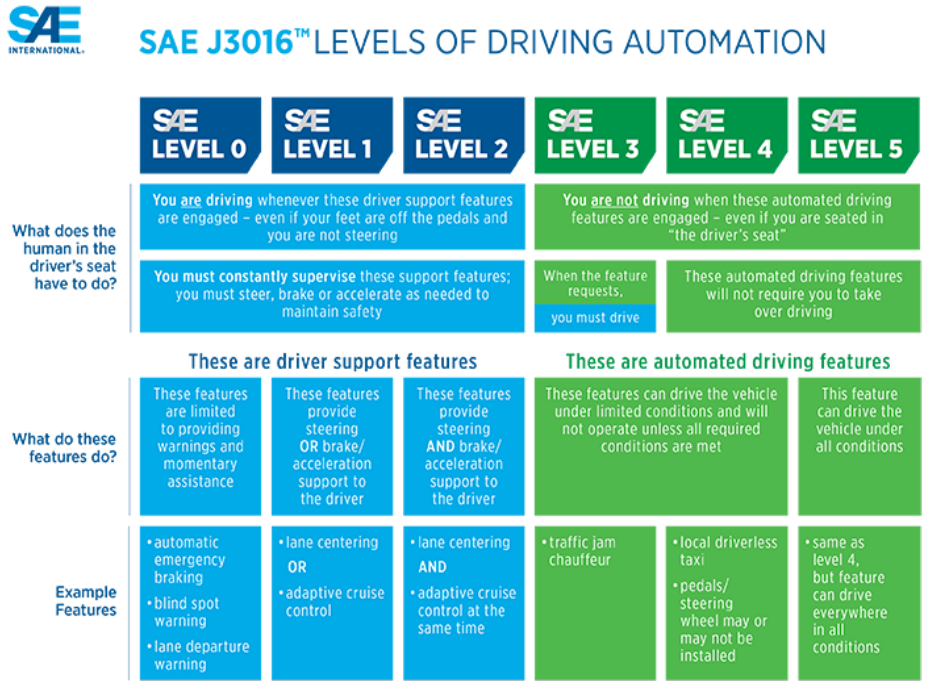

Figure 1.3: Levels of driving automation as defined by SAE [22]

Levels	**Example Features**
Level 0	Automatic Emergency Braking, Lane-depature warning
Level 1	Lane centering, Adaptive cruise control
Level 2	ADAS
Level 3	Traffic Jam Pilot
Level 4	Ride sharing under geofencing
Level 5	Full self-driving

Table 1.1: Autonomy levels and features

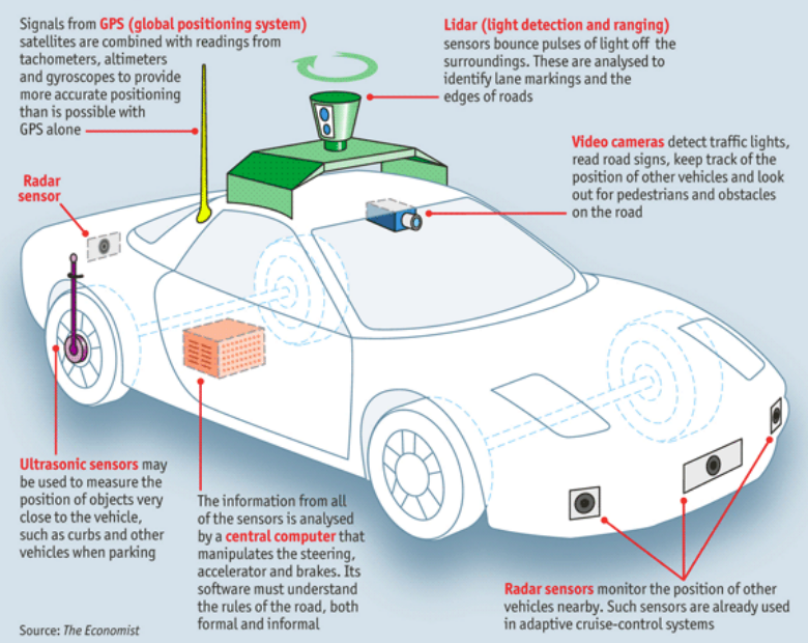

Figure 1.4: Various subsystems of an autonomous vehicle [5]

Self-positioning

Self-positioning involves obtaining the precise real-time location of the ego-vehicle with respect to a coordinate frame, generally being the global coordinate system represented by latitude, longitude and elevation. This is a critical component as it localizes the vehicle in the environment allowing fusion with other subsystems such as high-definition maps.

A few key technologies that are used for this task are

- `Differential GPS (DGPS):`

  Uses ground based reference stations to improve GPS positioning accuracy to a few centimeters.

- `Inertial Measurement Units and Landmark orientation:`

  A combination of accelerometer, gyroscope, magnetometer and other sensors to keep track of the vehicle's state such as orientation, velocity, and position when DGPS signals are weak.

  Landmark orientation techniques use cameras to detect certain landmarks and then calculate the vehicle's state such as orientation based on the stored meta-data about the landmark.

- `Radio Frequency Identification:`

  Small chips or other electronic orientation aids are applied in or on the road or in the road environment, which the corresponding transmitter-receiver systems can detect electromagnetically to use for self-positioning [18].

High-definition static maps and dynamic meta-data:

High-definition static maps contain the static geodata of a given scene/location, such as drivable area, number of lanes, lane coordinates, traffic signs, construction zones and dynamic metadata such as traffic congestion zones, traffic light states, accident prone zones and so on. This can be seen in Figure 1.5 generated from a popular AD dataset called NuScenes, which highlights the different color coded layers. This information is fused with ego self-positioning to provide context of the surrounding scene to the decision making engine of the vehicle. Techniques like Simultaneous Localization and Mapping (SLAM) and spatial geometric modeling are used to generate and update these high-definition maps.

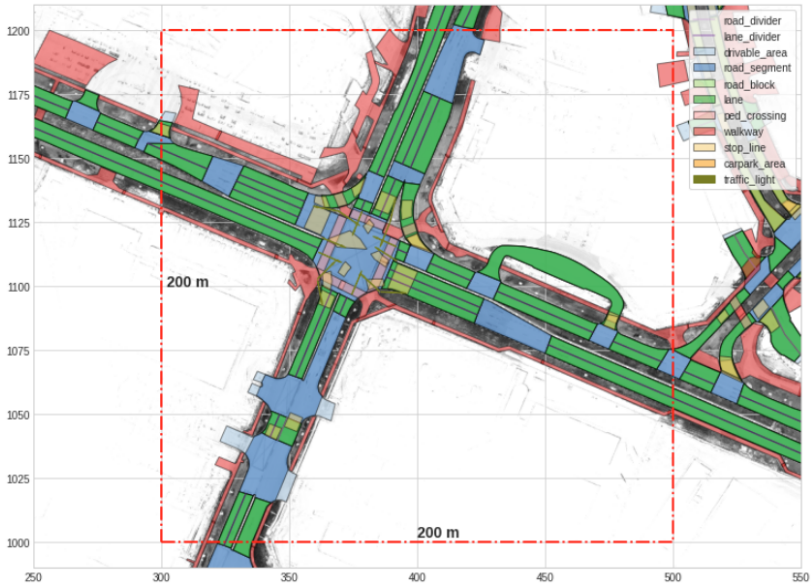

Figure 1.5: Example of a high-definition static map [11]

Agents detection/tracking and environment mapping sensors:

In order to plan a trajectory, the ego vehicle first requires the exact position of static objects like buildings, road dividers, traffic signs in the scene as well as the dynamic states such as precise position, outline, speed and heading of all moving agents like cars, bicycles and pedestrians in the scene. This is done with real-time sensors such as mono/stereo cameras, laser scanner/LIDARs, RADAR and ultrasound. The data from these sensors are noise filtered and passed to a sensor fusion module which extracts all required information for detection and tracking module.

Connected vehicle infrastructure network:

As the decision making engine is generally data driven, the ego-vehicle can connect to a wireless local area network consisting of other vehicles and infrastructure

nodes in the scene, and exchange information that can be extremely beneficial, especially in emergency situations. The currently much-discussed 5G mobile standard is particularly suitable for V2X communication given its high data transmission rates, real-time transmission, and low latency. WiFi access points on street lights have been another contender for this task too. A few examples of the data shared can be neighbouring agent vehicle states, traffic congestion scores along a lane, traffic light states, and emergency collision avoidance beacons that can be life-saving.

Actuation/Control and Vehicle-Driver interface:

Actuation/Control deals with taking the inference output from the decision making engine such as estimated future velocity, acceleration, heading rate and producing the proportional actuation such as accelerator pedal, braking and wheel rotation such that the given maneuver is safe under the vehicle's maneuverability limits. These operations can be controlled by the existing Engine Control Units (ECU) in the vehicle. A Vehicle-Driver interface provides a smooth medium of interaction between the human passenger/driver and the vehicle's autonomous unit. This is crucial when a switch-over to manual is required in sub Level 5 systems when the self-driving unit encounters an anomaly and is incapable of making a decision.

Prediction and Decision making engine:

This component is at the core of the whole self-driving pipeline. It consists of specialized hardware such as custom Graphics Processing Units (GPU), Tensor Processing Units (TPU), or other Application Specific Integrated Circuits (ASIC) that run the neural network based self-driving pipeline. Existing approaches to a self-driving pipeline can be roughly categorized into two types.

- `Modular based:`

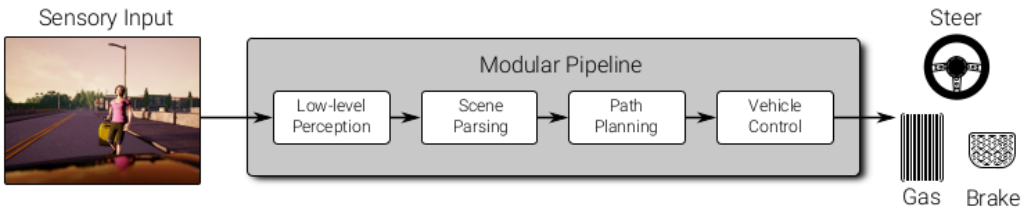

Figure 1.6: Modular based pipeline [23]

The modular pipeline is the standard approach to autonomous driving, mostly followed in the industry. The key idea is to break down the complex mapping function from high-dimensional inputs to low-dimensional control variables into modules which can be independently developed, trained, and tested. These modules comprise of sensor fusion, detection and tracking, motion forecasting, trajectory planning, and vehicle control as illustrated in Figure 1.6. However, this is just one example of modularizing a self-driving stack and other or more fine-grained modularizations also exist. Existing approaches typically leverage deep neural networks to extract low-level features or to parse the scene into individual components. In contrast, path planning and vehicle control are dominated by classical state machines, search algorithms, and control models.

- `End-to-end learning based:`

As most of the modules in the modular approach are trained and validated independently from each other, making use of auxiliary loss functions, the overall solution might not be optimal. Hence, an alternative to modular pipelines is

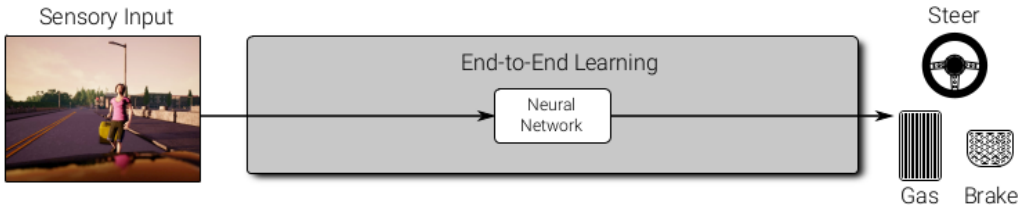

Figure 1.7: End-to-end based pipeline [23]

end-to-end learning-based models, which try to learn a policy, from observation to actions using a generic model such as a deep neural network. The network parameters can be learned either via imitation learning by replicating the behavior of a teacher or using reinforcement learning by exploring the world and taking actions that are likely to yield a high user-specified reward. However, reinforcement learning approaches suffer from the credit assignment and reward shaping problems, which are typically slow and can only be applied in non-safety-critical simulation environments. Imitation learning, on the other hand, suffers from overfitting and does not easily generalize to novel scenarios. Furthermore, holistic neural network-based approaches are often hard to interpret as they present themselves as "black boxes" to the user which do not reveal why a certain error has occurred [23].

This thesis is primarily concerned with the modular approach due to the demerits discussed with end-to-end learning and below is a brief description on the elements of the self-driving pipeline with this approach:

1. `Sensor Fusion:`

   This is primarily a feature extraction stage that selectively fuses data from

different sensors such as cameras, LIDAR and RADAR as a feature input to the next stage.

2. `Detection, Semantic Segmentation and Tracking:`

   This module performs 3D localization on the fused data to detect different static/dynamic objects in the scene and tracks motion of moving agents such as cars and pedestrians. YOLOv3 [26] and Faster-RCNN [27] are some of the popular object detectors. Tracking can be done using spatio-temporal graphs or Recurrent Neural Networks (RNN).

3. `Motion Forecasting:`

   This module predicts one or more possible future trajectories of the target agents based on the current and past observations of the surrounding environment. Predicting the future states of all agents in the scene will help the ego-agent perform trajectory planning in a safe and efficient way. This module uses the detection and tracking information from the previous module along with high definition maps to minimize prediction uncertainty.

4. `Trajectory planning:`

   Based on the trajectory prediction of all agents in the given scene, this module plans a safe trajectory for the ego vehicle. Here, generally the classical grid search algorithms and dynamic programming are used to find an optimal minimum cost trajectory with the constraints imposed by the state predictions of other agents.

With the large-scale commercialization of Graphics Processing Units (GPU) and the design of specialized hardware like Tensor Processing Units (TPU) for machine

learning, plenty of automotive companies and new startups have invested resources in climbing up the ladder of autonomy levels. There has been a lot of success stories in the Level 0 to Level 2 category with quite a few production cars rolling out with these capabilities. Deployment success has been limited for Level 3 and above as the technological leap is significant with only a few names such as the Tesla FSD beta. Level 5 autonomy demands sophisticated algorithms and enormous compute power to correctly handle the variability of driving environment scenarios and is the current area of interest for the autonomous research community. In this thesis, one such critical task of Motion Forecasting is being tackled and solving this problem will contribute to realizing the Level 5 autonomy.

1.2 Addressing the Motion Forecasting problem

Vehicles generally have well-defined motions governed by traffic rules and drivable areas. And as vehicles are non-holonomic, they cannot change their trajectories suddenly to the desired ones. Although, the problem of motion forecasting in itself is not a trivial task as the following factors variably influence the future states:

- Interdependence in behaviour among vehicles in the vicinity
- Constraints imposed by road geometry and traffic rules
- Inherent uncertainty in prediction due to the multimodal nature of future states
- Pedestrians, cyclists have a higher degree of freedom causing more uncertainty
- Limited view of the surrounding agents due to sensor pack limitations and occlusions

As listed in the last point above, the model can suffer if the forecasting is done locally by the ego-agent due to limited observability of the onboard sensors and occlusions by large agents such as trucks. This can be resolved by having the infrastructure run the scene localization/tracking instead and create a state encoded birds-eye view of the scene at a given time, illustrated in Figure 1.8, which can be downloaded by the agents present in the scene and motion prediction can be run on this representation. In this thesis, to simplify the problem, this birds-eye representation is presumed to be available for a given scene at a given instance of time. Before formulating the problem statement, the below paragraph defines the necessary terminology for the following sections.

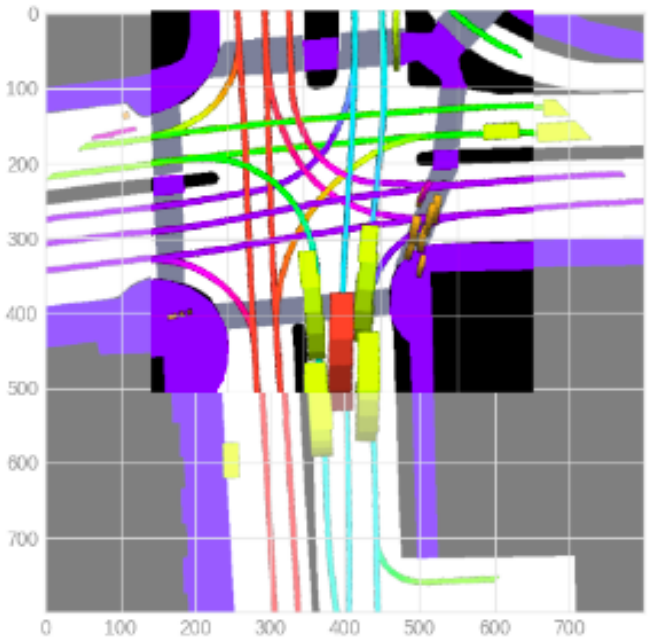

Figure 1.8: Birds-eye-view representation of a scene

Given a birds-eye view of a scene like the one shown in Figure 1.8, agents can be classified under the following categories:

- `Ego Agent (EG):` The autonomous vehicle that runs the self-driving pipeline by observing target agents and decides a safe path to take towards destination.
- `Target Agent (TA):` The vehicle of interest whose future trajectory is being predicted (shown in red in the figure).
- `Interaction Space (IS):` Region around the Target Agent containing the agents that can affect the future trajectory states of the Target Agent. This is decided based on a vicinity criterion such as a circle around the target vehicle or a cropped region (shown as non-faded region in the figure).
- `Surrounding Agents (SA):` All agents in the Interaction Space. These agents can affect the TA's future trajectory states. These agents can be pedestrians or cyclists too. (shown in yellow and brown in the figure).
- `Ineffectual Agents (IA):` Vehicles outside the Interaction Space defined by the criterion that do not affect the future trajectory states of the Target Agent (faded region in the figure).

With the above terminology, the Motion Forecasting problem can be formulated in a basic form as below:

Let I_i^t be the Interaction Space (IS) defined for a given Target Agent i with its state tensor S_i^t at time t defined as below:

$$S_i^t = (x_i^t, y_i^t, v_i^t, a_i^t, \theta_i^t) \in \mathbb{R}^5,$$

where x_i^t and y_i^t are the TA's relative spatial co-ordinates, v_i^t and a_i^t are velocity and acceleration respectively and θ_i^t is the yaw rate.

The co-ordinates are expressed in a stationary frame of reference where the origin is the position of the TA at a given time t. State history of any given agent j at the time of prediction t_0 for t_h past time steps can be defined as

$$S_j = [S_j^{t_0-t_h}, ..., S_j^{t_0}] \in \mathbb{R}^{t_h \times 5}.$$

Let $S_{SA}^{t_0}$ denote the history tensors of N surrounding agents set SA in the interaction space $I_i^{t_0}$ upto t_h past time steps for the target agent i.

$$S_{SA} = [S_j] \in \mathbb{R}^{N \times t_h \times 5}, \forall\, j \in SA$$

For a given target agent i and time of prediction t_0, we wish to find its future states Y_i for t_f future time steps defined as

$$Y_i = [Y_i^{t_0+1}, ..., Y_i^{t_0+t_f}] \in \mathbb{R}^{t_h \times 5}.$$

This can be expressed as estimating a probability distribution over the future states defined as $P(Y_i|S_i, S_{SA}, M_i)$, where $M_i \in \mathbb{R}^{A \times A \times C}$ is the multilayered high-definition map with the target agent spatial position as the origin, with resolution $A \times A$ pixels and C feature channels consisting of lane information, traffic signs and other dynamic meta-data.

The above formulation can be easily extended to multimodal estimation wherein instead of estimating only one trajectory, we estimate K possible future trajectories by learning K different probability distributions over the future states.

$$Y_i^{(MM)} = [Y_{i^k}],\ k\ =\ 1, ..., K$$

This estimation is run for all required target agents (TA) in a given scene. In the next chapter, the various approaches to network parameterization and optimization objectives in learning these probability distributions are discussed.

1.3 Problem significance and solution impact

Having a highly accurate motion forecasting model can drastically improve the trajectory planning of the ego agent especially in a complex environment such as city traffic and intersections. Executing a low confidence trajectory without considering the future states of other agents can have dire consequences. Compared to vehicles, pedestrians have a much higher degree of freedom while crossing the road. Hence, it becomes essential to predict the motion of pedestrians as accurately as possible before planning a trajectory for the ego vehicle. Even a small error in the prediction can result in injuries and sometimes loss of life. Hence, the performance of the forecasting algorithm is of utmost significance.

Improvement in the forecasting algorithm will in turn improve the performance of the self-driving vehicle hence pushing the vehicle a few steps closer towards Level 5 autonomy. The primary goal of Level 5 self-driving is to improve road safety and improve transport efficiency. The somber statistics on road accidents every year remind us of the dire need to have better road safety. Below are few of the statistics on the amount of road accidents that occurred around the world in recent years.

- There were 33,244 fatal motor vehicle crashes in the United States in 2019, in which 36,096 deaths occurred. This resulted in 11.0 deaths per 100,000 people

and 1.11 deaths per 100 million miles traveled [2]. Figure 1.9 compares the statistics in the US over the years.

Year	Fatalities	Annual percent change	Fatality rate per 100 million vehicle miles traveled	Fatality rate per 100,000 registered vehicles
2010	32,999	-2.6%	1.11	12.82
2011	32,479	-1.6	1.10	12.25
2012	33,782	4.0	1.14	12.72
2013	32,893	-2.6	1.10	12.21
2014	32,744	-0.5	1.08	11.92
2015	35,484	8.4	1.15	12.61
2016	37,806	6.5	1.19	13.13
2017	37,473	-0.9	1.17	12.90
2018	36,835	-1.7	1.14	12.31
2019	36,096	-2.0	1.10	NA

NA=Data not available.

Source: U.S. Department of Transportation, National Highway Traffic Safety Administration.

Figure 1.9: Traffic related deaths in the US over the years [4]

- On a global scale, approximately 1.35 million people die in road crashes each year; on average 3,700 people lose their lives every day on the roads. Road traffic injuries are the leading cause of death among young people aged 5-29. Young adults aged 15-44 account for more than half of all road deaths. More than 90 percent of all road fatalities occur in low and middle income countries, even though these countries have approximately 60 percent of the world's vehicles [1]. Figure 1.10 compares the global statistics over the years.

Causes for road accidents can be attributed to a handful of human-related factors such as alcohol influenced driving, failure to use restraint, distraction from media

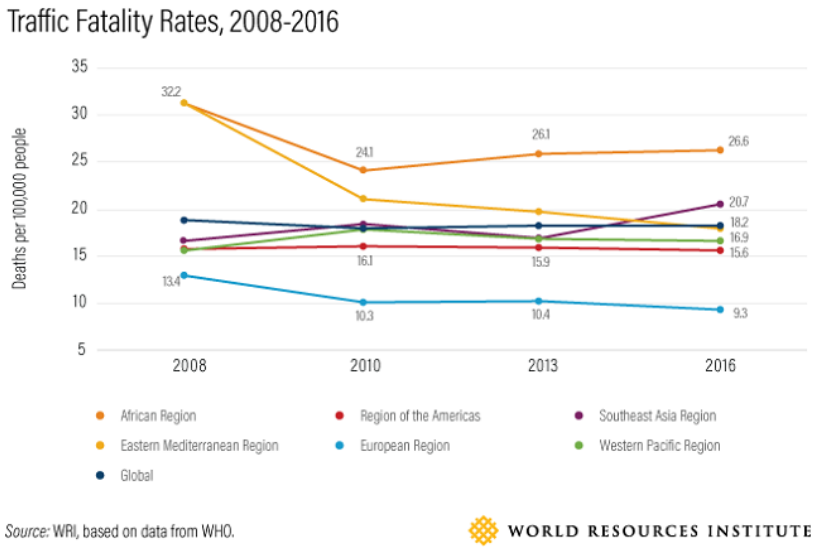

Figure 1.10: Traffic related deaths around the globe over the years [3]

gadgets, and so on. All these accident-inducing factors can be resolved by minimizing the dependence of the car on the human to drive. Having the manual driving cars replaced by the Level 5 self-driving cars could significantly impact road transportation and could reduce the accident-related casualties by a significant margin. Thus, solving the problem of motion forecasting provides a significant performance boost for the self-driving vehicle which in turn can positively impact road safety concerns.

1.4 Organization of this Thesis

The rest of this thesis is organized as follows.

Chapter 2 will start off introducing all popular approaches to deep learning based motion forecasting mentioned in the literature. It will showcase the concepts involved accompanied by a few illustrations/use-cases. This will be followed by a review of

state of the art models built on one of these approaches, which served as an inspiration to the model being proposed.

Chapter 3 gives a comprehensive background on the methodology used in the implementation and experiments involving the proposed model. This includes the overall architecture design, input representation, state encoding, agent interaction, trajectory decoding, and optimization objectives.

Chapter 4 lists the details about the dataset, introduces the metrics used for evaluation, and discusses the baseline designs. This is followed by a description on implementation and result analysis of six experiments related to the proposed model and a discussion on additional experiments and special case evaluations.

Chapter 5 discusses the future scope of the project proposing a few promising extensions and improvements to the model and then finishes off with concluding remarks.

Chapter 2: Existing approaches and related work

The first section of this chapter delves into the various approaches to solving the motion forecasting problem for autonomous driving. It first discusses the classical approach to the problem followed by the learning based approach. The second section highlights some of the related work based on these approaches that served as an inspiration for the proposed model.

2.1 Existing approaches

These can be primarily subdivided into classical based models such as classical state machines that are physics driven and into learning based models that are data driven.

2.1.1 Classical approach

The core idea under classical approach is that the mapping function between the input representation (target vehicle state history/map information) and the output trajectories is manually defined by physics models, for instance built on forward/inverse kinematics and dynamics equations. Future states are predicted using dynamic and kinematic models linking control inputs like accelerator and steering

and car properties like weight and external conditions like friction coefficient of the road surface to the state outputs of the vehicle like position, speed and heading.

Constant velocity model with Kalman filtering:

The simplest state propagation can be done using the constant velocity model which uses a Kalman filter to generate robust predictions with a constant velocity assumption at every time step. The velocity sensor measuring vehicle speed v in a heading direction Φ is used to calculate the x and y components of the velocity using the revolution of wheels and heading rate sensor. Figure 2.1 shows the prediction and correction operations applied at every step.

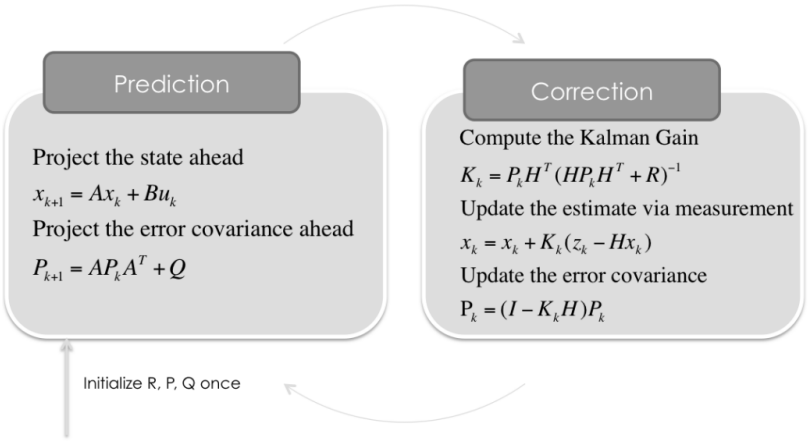

Figure 2.1: Each estimation step involves a prediction and correction stage [6]

Here, we assume the control input u_k to the model as 0. With this assumption, the state vector at a time step k can be defined as:

$$X_k = [x_k\, y_k\, \dot{x}_k\, \dot{y}_k]^T$$

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$X_{k+1} = A\, X_k$$

where X_{k+1} is the next state and A is the state transition matrix. Now, the observation model can be defined as:

$$Y_k = H\, X_k$$

$$H = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where H is the measurement matrix and Y_k is the output state at time step k.

Below, we define the terms initial uncertainty (P_0), measurement noise covariance (R) which informs the Kalman filter about the sensor reading accuracy and process noise covariance (Q) which models the disturbance in the process caused by an external factor.

$$P_0 = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 \\ 0 & \sigma_y^2 & 0 & 0 \\ 0 & 0 & \sigma_{\dot{x}}^2 & 0 \\ 0 & 0 & 0 & \sigma_{\dot{y}}^2 \end{bmatrix}$$

$$R = \begin{bmatrix} \sigma_{\dot{x}}^2 & 0 \\ 0 & \sigma_{\dot{y}}^2 \end{bmatrix}$$

$$Q = \begin{bmatrix} \sigma_x^2 & 0 & \sigma_{x\dot{x}} & 0 \\ 0 & \sigma_y^2 & 0 & \sigma_{y\dot{y}} \\ \sigma_{x\dot{x}} & 0 & \sigma_{\dot{x}}^2 & 0 \\ 0 & \sigma_{\dot{y}y} & 0 & \sigma_{\dot{y}}^2 \end{bmatrix}$$

With these matrices formulated, we can define the two steps as:

1. `Prediction:` Propagate the state X_k to next time step using state transition matrix and project the error covariance matrix P_k ahead.

2. `Correction:` Compute the Kalman filter gain using H, P_k and R. Once this gain is known, update the process estimate via sensor measurements z_k and update the error covariance matrix P_k. These updated values are used for next time step. This is illustrated in Figure 2.1.

Physics Oracle:

This is an ensemble model that selects the best performing physics based model based on a predefined metric. Below are the physics based models that are contenders for the oracle.

- Kalman Filter with Constant Velocity
- Kalman Filter with Constant Acceleration
- Adaptive Kalman Filter with Constant Velocity
- Extended Kalman Filter with Constant Turn Rate and Velocity (CTRV)
- Extended Kalman Filter with Constant Turn Rate and Acceleration (CTRA)

The first model design was examined in the previous paragraph. For this thesis, the best performing out of all these models is selected as the physics oracle based on the evaluation metrics discussed in Chapter 4.

2.1.2 Learning based approach

This approach is data driven wherein typically a machine learning model learns the mapping between the input representation (target agent history and high-definition

map) by minimizing an objective function over the data samples. This can be further divided into maneuver intention based approaches which treat the task in hand as a classification problem and multimodal trajectories based approaches which are more sophisticated. In terms of the model architecture, we will concentrate on the deep neural networks (DNN) to solve the task in hand due to their better generalization capabilities. The multimodal trajectories approach will be the center of focus for the following sections.

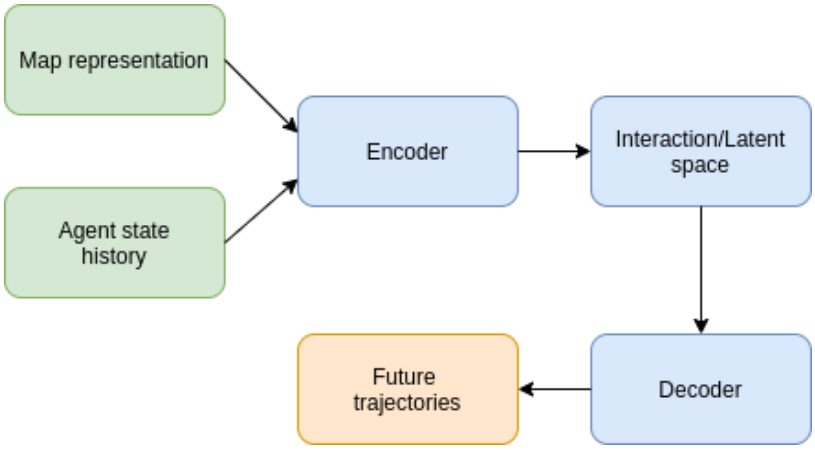

Figure 2.2: Basic structure of the encoder-decoder DNN architecture

The general architecture for trajectory state prediction can be classified as an encoder-decoder based DNN design as seen in Figure 2.2. The encoder takes the input representation and outputs low-dimensional dense feature vectors which can then be passed to the decoder to generate the future trajectory positions. Now, as there are multiple components involved in the architecture, there are quite a few design choices for the same. Table 2.1 describes the different design choices available under the four main components of the architecture.

Map Representation	Trajectory Encoder	Social Interaction	Trajectory Decoder
Lane Polylines	LSTM/GRU	Linear attention	LSTM + GMM
Rasterized CNN	1D Convolution + LSTM	Multihead attention	Polyfit
Lane Graph	Social LSTM	Transformers	Occupancy map
Spatio-temporal graph	Rasterized fading	Graph attention	Predefined anchors with offset

Table 2.1: Design approaches for modularization

Map Representation and Encoding

There are plenty of ways to represent and encode high-definition map information. These maps include static information such as road lanes, crosswalks, road signs, and dynamic information such as traffic light states, congestion areas, and so on. One way to represent lane information is using lane polylines. A better way to represent spatial information is to use the birds-eye view representation wherein you rasterize the static map in a top-view fashion with the origin centered on the target agent's location and color code different classes. This is a hierarchical way of representing map information starting from the base containing the drivable area with lanes and going all the way to the top with the traffic signs, traffic light states and so on.

Here are a few key features of this representation:

- `Area masks:` Indicate area properties such as drivable area, bicycle lane, crosswalk
- `Lane polylines:` Center line or left-right borders of all drivable lanes
- `Colorized Rasterizer:` Colors the lane pixels based on the angle to a reference axis using HSV color wheel. The angle is assigned to the hue value and the other two components are set to 1.

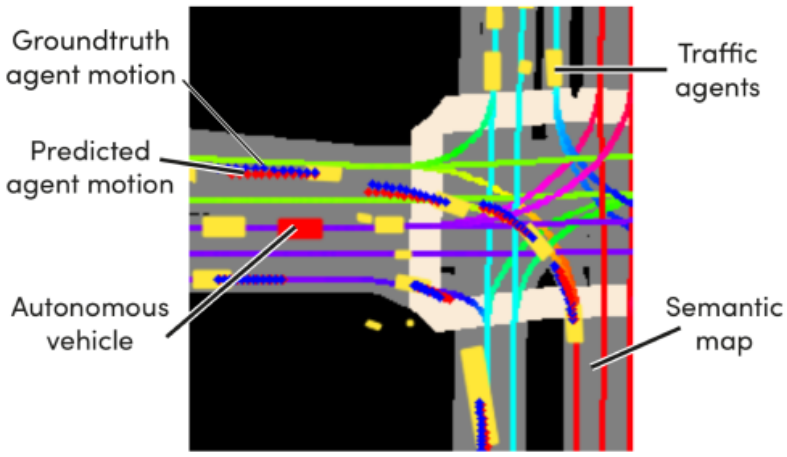

Figure 2.3: Semantic representation from Lyft Level 5 dataset [21]

- `Agent localization:` The target agent and the surrounding agents can be localized at the prediction timestep on the rasterized map with different colored extent boxes.

- `Local Dynamic Map:` Contains varying number of channeled rasters containing static maps in the base channels and meta data such as traffic signs, landmarks, accident zone warnings, and so on in higher channels.

- `History fading:` The history of the target and surrounding agents can also be encoded in the map by fading the color of the agent boxes for older history points.

Rasterized representations are generally encoded using pretrained Convolutional Neural Networks (CNN) like ResNet50 [19] to obtain dense feature vectors. These residual networks use skip connections to enable large enough gradients to back-propagate through a large number of layers.

An alternative way to represent the information is to use spatio-temporal graphs [29] wherein each node can be represented as an intersection or a split point in lanes and the edges represent the lanes. All meta-data can be encoded as node and edge information. These graphs are generally encoded using Graph Convolutional Networks (GCN).

Trajectory Encoder

The simplest way to encode history states without introducing the RNNs is to fade the color of the agents in the rasterizer map at which point, history encoding becomes a part of input representation. However, as this does not explicitly model the temporal relation between the states, the RNN approach is explored.

Recurrent Neural Networks are popular to model the temporal relationship between states. These networks are basically fully connected linear layers (FC) rolled out in time as seen in Figure 2.4.

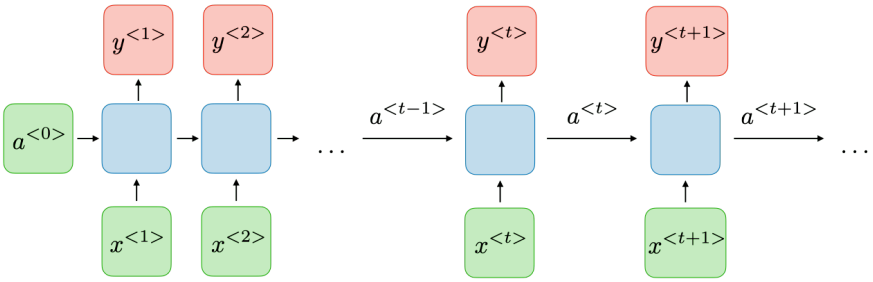

Figure 2.4: Recurrent neural network architecture [9]

RNNs suffer from vanishing gradients when rolled out for longer time periods. To fix this issue, gated control mechanisms are introduced which learn what information is necessary and needs to be maintained over time and this results in improved memory

modules such as Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU).

`Long short term memory (LSTM):` LSTM was proposed in 1997 to counter the drawbacks of recurrent neural networks [20]. A basic LSTM cell architecture is illustrated in Figure 2.5. Below are a few key points about LSTM cells shown in the Figure 2.5.

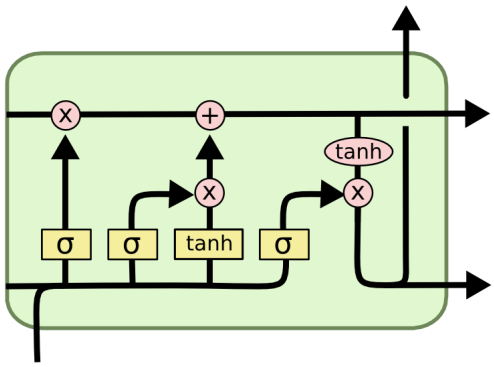

Figure 2.5: LSTM cell with forget, input and output gates [10]

- It maintains a cell state that connects cells all the way through and acts as a data transport lane that transfers sequential information down the given chain acting as a memory state. The information to be updated or removed in the cell state is controlled by gates whose controls are learnt.
- Gates use sigmoid activation to update or forget information
- Forget gate decides what kind of information needs to be kept or thrown away. Information from the previous hidden state and the current input state are

passed through this gate. An output value of 0 indicates forget and a value closer to 1 indicates keep.

- Input gate updates the cell state. The hidden state and current input go through the tanh activation and sigmoid activation, with the sigmoid deciding which part of the tanh output is to be kept.

- Output gate decides what the next hidden state needs to be. The previous hidden state and current input are passed through similar activations to decide which part of the hidden state information needs to be forward propagated.

`Gated recurrent units (GRU):` The GRUs can be considered an alternative to LSTMs with no explicit cell states [13]. Instead, GRUs use the hidden state itself to propagate information and they have only reset and update gates. The update gate is analogous to the forget and input gates of an LSTM. The reset gate allows to erase all information coming from the previous cell. The architecture of a GRU cell is illustrated in Figure 2.6.

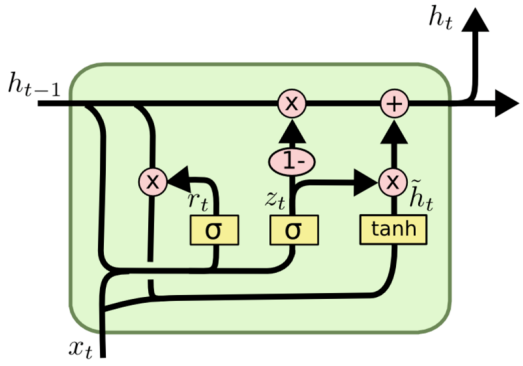

Figure 2.6: GRU cell with update and reset gates [8]

`Social LSTM:` An alternative way to encode agent trajectories is to preserve the relative spatial relationship between the agents by stacking the LSTM encodings on a spatial grid. This representation is known as Social LSTM. More information on this representation is presented in the next section of the chapter.

Social Interaction with Attention mechanisms

Social interaction aims at capturing how the behaviour of surrounding agents affects the prediction of the target agent. This type of interaction can be complex and involve uncertainty at multiple levels. In the recent developments, this has generally been implemented with attention mechanisms. We will particularly concentrate on the visual attention aspect of the networks. This module generally outputs a context vector that is then concatenated with the target agent state encoding as an input to the decoder.

Although attention mechanisms were initially applied to natural language processing (NLP) to help memorize long sentences, they have been making their way into vision tasks as well [31]. Attention layers learn a set of weights to linearly combine a set of features such that the weights reflect the corresponding feature's importance in a particular task. This was primarily used in NLP for instance, in language translation to learn the importance of each of the encoder words in predicting a decoder word.

A few types of visual attention mechanisms popular in computer vision are

- `Additive attention:` Uses one-hidden layered feed-forward networks to determine the weights between a given query and a value by learning a linear combination of the two.

- `Multiplicative attention:` Uses cosine similarity between queries and keys to determine the weights for a given set of values.

- `Self-attention:` This is a case when a specific query does not exist. Rather, the query, keys and values are all same. In essence, the features attend to themselves. Transformers are a great example of an architecture built on multihead self-attention blocks. In multihead mechanisms, the layer learns multiple classes of attention weights instead of just one to cover different types of interactions. One such block is seen in Figure 2.7

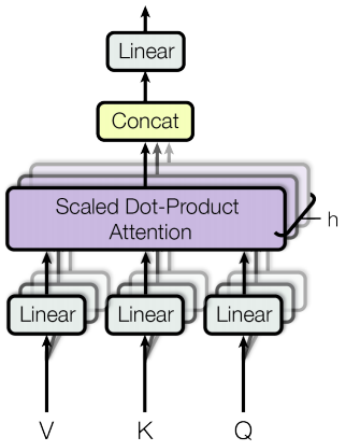

Figure 2.7: Multi-head attention mechanism with queries, keys and values [30]

Another class of differentiation for attention mechanisms is

1. `Soft-attention:` The alignment weights are learnt and placed softly over all patches in the source image. That is, a softmax function is applied over the weights to make them smooth. Generally, the network here can focus on multiple interest points in the image.

2. `Hard-attention:` Only one part of the image is attended to at a time. That is, the network focuses only on a single part of the image.

Trajectory Decoder

The trajectory decoder takes the input from the interaction module in the form of a context vector and the target agent encoding and decodes one or more valid future trajectory predictions for a given scenario. Various approaches have been implemented in the literature to efficiently decode the future trajectories and here, a few of the popular ones are listed.

- LSTMs: Unlike the many-to-one LSTMs used in the encoder, the decoder design includes a one-to-many prediction pipeline. It takes the context features from the interaction module and decodes the future trajectories at every time step.

- GMM: Gaussian mixture models are great to model a probability distribution over a set of trajectories. A GMM assumes that all data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. These multi-gaussian models learn to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians. The Gaussian density function can be given as

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp(-0.5 * (x-\mu)^T \Sigma^{-1}(x-\mu)),$$

  where the parameters μ and Σ are generally estimated by a neural network decoder and D is the dimension of the data points. The likelihood of the data distribution $p(X)$ can be expressed as the joint probability of all the N data points.

$$p(X) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{m=1}^{M} \pi_m \mathcal{N}(x_n | \mu_m, \Sigma_m),$$

where π represents the probability distribution over M Gaussians represented by a latent variable z such that $\pi_m = p(z_m = 1)$.

2.2 Related work

This section briefly goes through some of the popular models shortlisted based on the design principles mentioned above.

2.2.1 Colorized rasterization and CNN encoder

One of the models built on these components is referred to as Multimodal Trajectory Predictions (MTP) in the literature [14] which uses colorized rasterization to represent the map information as well as the agent information using extent boxes and faded color representation for history. This model proposes a novel multimodal regression loss to generate K different trajectories.

Figure 2.8 displays the architecture from the paper.

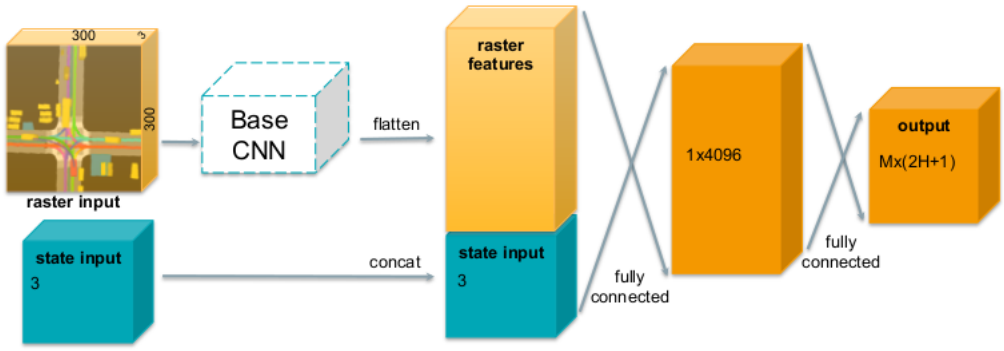

Figure 2.8: MTP model architecture [14]

Encoder design:

- The rasterized input of shape $300 \times 300 \times 3$ is passed through a pretrained base CNN model such as ResNet50 that acts as a feature extractor.
- The dense features from the CNN are flattened and concatenated with the target agent state consisting of velocity, acceleration and heading rate.

Decoder design:

The decoder is made up of two fully connected layers of size 4096 and $K \times (2t_f + 1)$ respectively, where K is the number of predicted trajectories and t_f is the total number of prediction timesteps per trajectory given by $t_f = T_f * f_s$ where T_f is the number of seconds of prediction and f_s is the sampling frequency of the trajectory points. The final layer of the decoder also predicts the probability of each trajectory prediction.

Loss function:

This paper proposes a novel loss function to perform multimodal regression on future agent trajectories. Firstly, a simple unimodal loss can be expressed in terms of the L2 regression loss as

$$L(Y_i^{t_0}, \tilde{Y}_{ik}^{t_0}) = \frac{1}{t_f} \sum_{t=t_0+1}^{t_0+t_f} ||Y_i^t - \tilde{Y}_{ik}^t||_2,$$

where $Y_i^{t_0}$ is the ground truth trajectory for target agent i at prediction time t_0 and $\tilde{Y}_{ik}^{t_0}$ is the k prediction mode.

Now, a simple way to extend this to multimodal loss is to use Mixture-of-Experts (ME) loss given by

$$L_i^{t_0^{(ME)}} = \sum_{k=1}^{K} p_{ik} * L(Y_i^{t_0}, \tilde{Y}_{ik}^{t_0}),$$

which is basically the expectation of the loss function over the K trajectories, where p_{ik} is the estimated probability of each mode. As claimed by the authors, this loss function is prone to mode collapse where all trajectories end up following the ground truth.

To prevent this, the authors propose the MTP loss wherein instead of averaging over all modes, they propose a metric to select the closest mode to ground truth and optimize the regressor only over that mode.

$$k^* = \underset{k \in (1,\ldots,K)}{\arg\min}\, d(Y_i^{t_0}, \tilde{Y}_{ik}^{t_0})$$

The $d(\cdot, \cdot)$ metric here can either be Euclidean distance or angle between the last points of the two trajectories. Once k^* is found, the loss function is defined as

$$L_i^{t_0^{(MTP)}} = L_i^{t_0^{(class)}} + \alpha \sum_{k=1}^{K} I_{k=k^*} L(Y_i^{t_0}, \tilde{Y}_{ik}^{t_0}).$$

This is a regression loss combined with classification loss. I is a binary indicator function equal to 1 when $k = k^*$, α is a hyper-parameter deciding the trade-off between the two losses and $L_i^{t_0^{(class)}}$ is the cross-entropy classification loss given by

$$L_i^{t_0^{(class)}} = -\sum_{k=1}^{K} I_{k=k^*} \log p_{ik}$$

It is interesting to note that the regression loss update is done only for the best mode, but the classification loss update is done for all modes. This enforces the model

to find unique trajectories and push the probability of the best modes evaluated by the distance metric towards one.

2.2.2 Predefined trajectory anchors with offsets and GMM distribution

The model being discussed here is referred to as MultiPath in the literature [12]. This model takes a different approach than regression. It samples likely modes called anchors from a predefined trajectory set and learns an offset per mode based on the anchor states and uncertainty distributions for each future time step. The model aims at the following:

- Estimate a weighted set of discrete trajectories formed with unsupervised learning from training data via K-means clustering or uniform sampling
- Closed-form evaluation of the likelihood of any trajectory

The goal is to estimate $p(Y|X, M)$ where Y represents the future anchor states, X represents the history and current states of the agents, and M represents the contextual information.This model formulation can be split into below two uncertainty estimations.

- `Intent uncertainty:` This treats the distribution over the anchor sets as a classification problem.

$$\pi(a_k|X) = \frac{\exp f_k(X)}{\sum_{k=1}^{K} \exp f_k(X)},$$

where $A = [a^k]_{k=1}^{K}$ is a set of K anchor trajectories and $f_k(X) : \mathbb{R}^{d(X)} \rightarrow \mathbb{R}$ is the mapping learnt by a deep neural network.

- `Control uncertainty`: Learn a normal distribution with mean offset and covariance measuring aleatoric uncertainty with the mean offset acting as an anchor residual. It is a Gaussian distribution dependent on each waypoint state of an anchor trajectory defined as

$$\phi(Y_k^t|a_k, X) = N(Y_k^t|a_k^t + \mu_k^t(X), \Sigma_k^t(X)),$$

where the Gaussian parameters $\mu_k^t(X)$ and $\Sigma_k^t(X))$ are learnt by the model.

The distribution over the whole state space is obtained by marginalizing over agent intent given by

$$p(Y|X) = \sum_{k=1}^{K} \pi(a_k|X) * \Pi_{t=t_0}^{t_0+t_f} \phi(Y^t|a_k, X).$$

Learning is done using the time-sequence extension of standard Gaussian Mixture Model (GMM) likelihood fitting defined as

$$l(\theta) = -\sum_{m=1}^{M}\sum_{k=1}^{K} I(k = \hat{k}^m) \log(p(Y|x^m; \theta)),$$

where $m = 1, ..., M$ are the M data points and I is a binary function similar to the one seen in previous model that returns the k^m that is closest to ground truth $\hat{Y}^m$.

Figure 2.9 illustrates a scenario where there are three likely intents with anchor offsets respecting the road constraints and control uncertainty growing over time as expected.

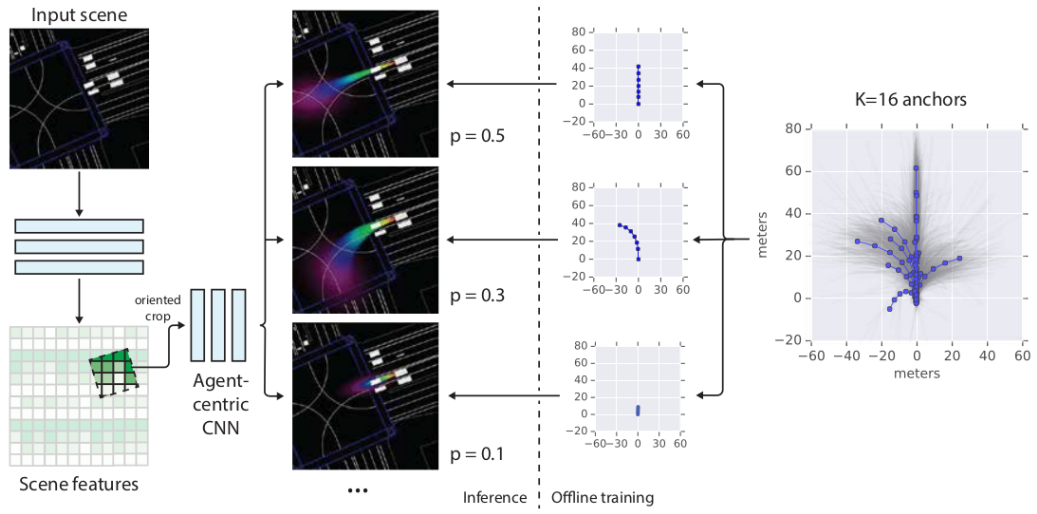

Figure 2.9: MultiPath model architecture [12]

Input Representation

Here, we discuss the input representation implementation details for the model.

- Input is a 3D tensor with depth layers adding different levels of contextual information (15 depth channels).
- Input tensor resolution is $400 \times 400 \times 15$ with channels 0-2 containing color-coded road semantics.
- 5 of the channels contain top-down orthographic projection of the past 5 time steps.
- Remaining channels contain meta-data such as distance-to-road-edge map, speed limit, and traffic light states.

Network Architecture:

Here, we discuss the network architecture implementation details for the model.

- The encoder used for feature extraction is of ResNet family
- The decoder stage is an agent-specific trajectory estimator with heading normalization. It outputs $K \times t_f \times 5$ parameters representing the GMM and K softmax logits.

2.2.3 Graph representation, CVAE and Dynamic Integration

This section discusses a model that uses graph representation as input and has a Conditioned Variational Auto-Encoder (CVAE) based architecture. The decoder also involves a dynamics integrator to minimize trajectories physically not possible to execute for the vehicle. The architecture diagram of this model called Trajectron++ in the literature [29] can been seen in Figure 2.10.

Input representation:

The scene is represented using a spatio-temporal graph $G = (V, E)$ with nodes representing the agents and the edges representing their interactions. Edges are directed and exist to indicate the affect of a surrounding agent's behavior on the target agent. The road lane information is represented by a high-definition map along with dynamic meta-data layers. This map is processed by a CNN such as ResNet50.

Trajectory Encoder and Agent Interaction:

Past and current trajectories of the target agent are encoded by feeding their state vectors containing velocity, acceleration, and heading rate to LSTMs. The states of surrounding agents of the same class are first aggregated using a numeric

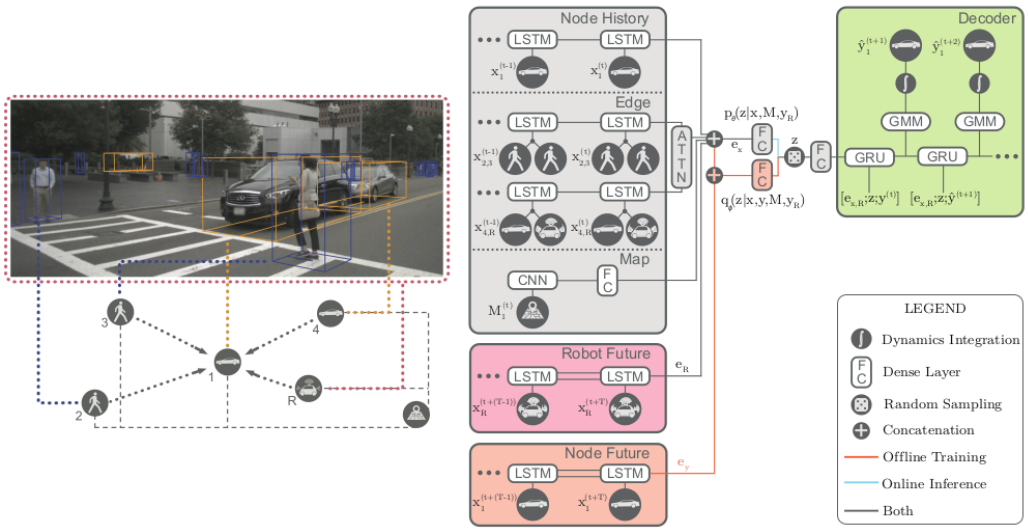

Figure 2.10: Trajectron++ model architecture [29]

sum and then fed to the LSTMs to avoid handling variable number of nodes via a fixed architecture. The LSTM weights are shared across all edge instances of the same type. For instance, all Car-Pedestrian edge LSTMs share the same weights. The encodings from all these edges that connect to the target agent are then aggregated using additive attention. The output of this called edge influence encoding is then concatenated with the target agent's encoding to form a single node representation e_X [24].

CVAE:

This model explicitly handles the multimodal nature of trajectories by using the CVAE latent variable framework. It estimates $p(Y|X)$ distribution by learning a discrete categorical latent distribution with random variable $z \in Z$ which encodes high-level latent behavior. This can be expressed as

$$p(Y|X) = \sum_{z \in Z} p_\Phi(Y|X, z) * p_\theta(z|X),$$

where $|Z| = 25$ and Φ and θ are weights parameterizing their respective distributions. During training, a bidirectional LSTM is used to encode the target node ground truth future trajectory, producing $q_\phi(z|X, Y)$.

Trajectory Decoder:

The latent variable z and the single node representation vector e_X are fed to a Gated Recurrent Unit (GRU). Each of the GRU cells output parameters of a bivariate Gaussian distribution over control actions such as acceleration and steering rate. These control actions are then integrated by the dynamics integration module to estimate the future x and y coordinates of the target agent.

Objective function:

The objective function is defined as:

$$\max_{\theta,\phi,\Phi} \sum_{i=1}^{N} E_{z \sim q_\phi(.|X_i,Y_i)}[log p_\Phi(Y_i|X_i, z)] - \beta D_{KL}(q_\phi(z|X_i, Y_i) \,||\, p_\theta(z|X_i)) + \alpha I_q(X; z),$$

where I_q is the mutual information between X and z under $q_\phi(X, z)$

2.2.4 Joint Agent-Map Representation and Multi-head Attention

These methods take the approach of spatially preserving the relationship between agents by having a dense feature grid instead of a flattened vector and by placing the agent state encodings on top of this dense map encoded grid resulting in a social tensor. It then applies multi-head attention between the target agent and the social

tensor to generate a context vector. Figure 2.11 shows the architecture diagram of one such models known as MHA-JAM in the literature [24].

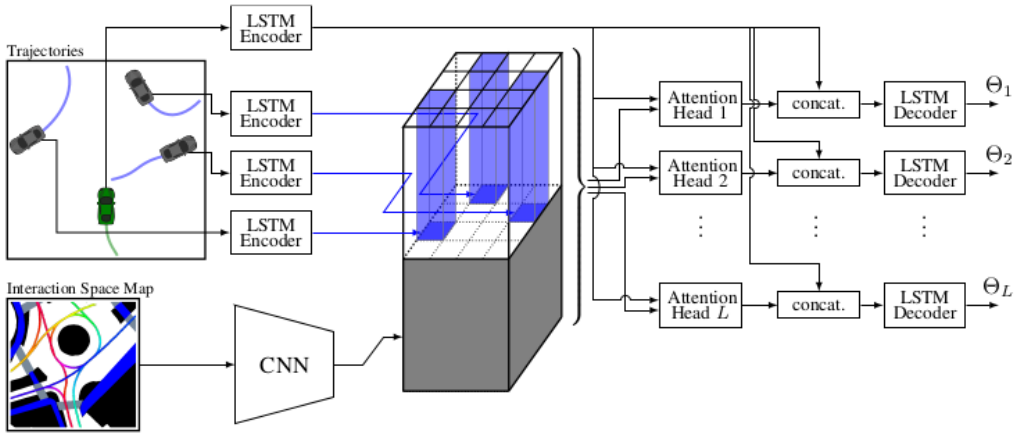

Figure 2.11: MHA-JAM model architecture [24]

Input representation:

The authors here propose a joint representation of the agents and the static map as the feature encoding. The static map is rasterized in a similar way as previous approaches. Again, a CNN based encoder is used to get the dense features of the map. However, unlike previous approaches, the final feature is not a vector but a 2D dense spatial feature grid. This way the spatial relationship between different agents is preserved. The model uses a pretrained ResNet50 to obtain the dense spatial grid of resolution $28 \times 28 \times 512$.

Trajectory encoding and multihead attention:

Trajectory encoding for all agents is done using LSTMs similar to previous approaches. For all surrounding agents, their corresponding position in the dense spatial

grid is found and their trajectory encodings are stacked on top of the map feature in that position. This joint representation is called a social tensor.

To capture the interaction between the target agent and the surrounding agents, multihead attention is applied between the target agent encoding and the social tensor. This can be formulated as shown.

Let F_s and F_m denote the social tensor and the map features, respectively. The combined tensor is obtained by concatenating the two tensors along the channel dimension.

$$F = F_s \oplus F_m$$

Attention mechanisms can be generally defined with queries, keys, and values. Keys and values are the column entries of a look up dictionary mapping keys to values. The task is to find the value for a given query. This is done by computing an inner product between the query and all the keys that serves as the weights to the linear combination of values. This can be formulated as

$$Q_l = \theta_l(h_T^{t_0}; W_{\theta_l}),$$

$$K_l = \phi_l(F; W_{\phi_l}),$$

$$V_l = \rho_l(F; W_{\rho_l}),$$

where $h_T^{t_0}$ is the target agent encoding, θ_l is fully connected layer parameter, ϕ_l and ρ_l are 1×1 convolution layers.

The output of each attention head is then calculated as a weighted sum of the values.

$$A_l = \sum_{m=1}^{M} \sum_{n=1}^{N} \alpha_l(m,n) * V_l(m,n,:),$$

where (M, N) is the size of the spatial grid and the weights are given by

$$\alpha_l(m,n) = Softmax(\frac{Q_l * K_l^T(m,n,:)}{\sqrt{d}}),$$

where d is the dimension of the feature vector

This context vector is then concatenated with the target agent encoding and fed to the decoder.

Trajectory Decoding:

Each context vector from L attention heads is concatenated with the target agent encoding and passed to L LSTM decoders. The decoders then generate the predicted parameters of the distribution over the target agent future positions. All these LSTMs share the same weights. Along with these parameters, the decoder also outputs the probability of each of the L mixture components.

Optimization objective:

The loss design is similar to the MTP model. As the model outputs parameters of a bivariate Gaussian distribution at each time step for L modes, Negetive Log-Likelihood (NLL) loss is used as the regression loss, computed between the ground truth and each of the modes. The mode with the minimum NLL loss is selected for regression update.

$$L_{reg} = \min_{l} \sum_{t=t_0+1}^{t=t_0+t_f} -log(P_{\Theta_l^t}(Y_l^t|X, M),$$

where S is the past and current states of all agents and M is the input map.

Classification loss is given by:

$$L_{cls} = -\sum_{l=1}^{L} I(l = l^*) log(P_l),$$

where I is the binary function equal to 1 when l is the mode with minimum NLL loss.

Overall loss is given by:

$$L = L_{reg} + \alpha Lcl + \beta L_{or}$$

where L_{or} is the off-road loss to force predictions to be in the drivable area.

In the next chapter, the methodology used in the proposed model is introduced.

Chapter 3: Proposed methodology

This chapter discusses the various elements that were explored in order to design two novel architectures of the proposed model for motion forecasting. The chapter goes through a similar order of modules as Chapter 2 to highlight the similarities and differences. Figure 3.1 and 3.2 show the high-level architectures of the two versions of the motion forecasting model being proposed. The iterative improvements that were made over these architectures are discussed thoroughly in a series of experiments listed in the Chapter 4. The two architectures will hereby be referred to as Arch-I seen in Figure 3.1 and Arch-II seen in Figure 3.2. This chapter sequentially goes through each of the design elements in the two proposed architectures and the reader is requested to refer to these figures while going through the design elements that follow.

3.1 Input representation

The input representation used for the Arch-I is similar to the ones used by the models in Chapter 2. The scene map is cropped in such as way that the target agent's location is at the origin. The steps followed in creating this representation are

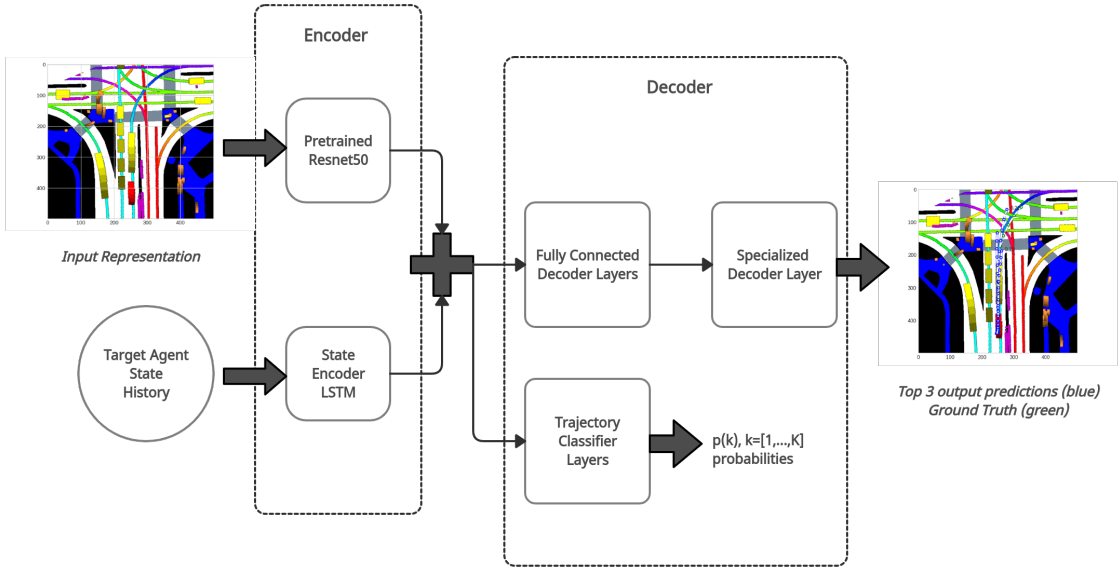

Figure 3.1: Architecture-I: Multimodal regression with specialized decoders

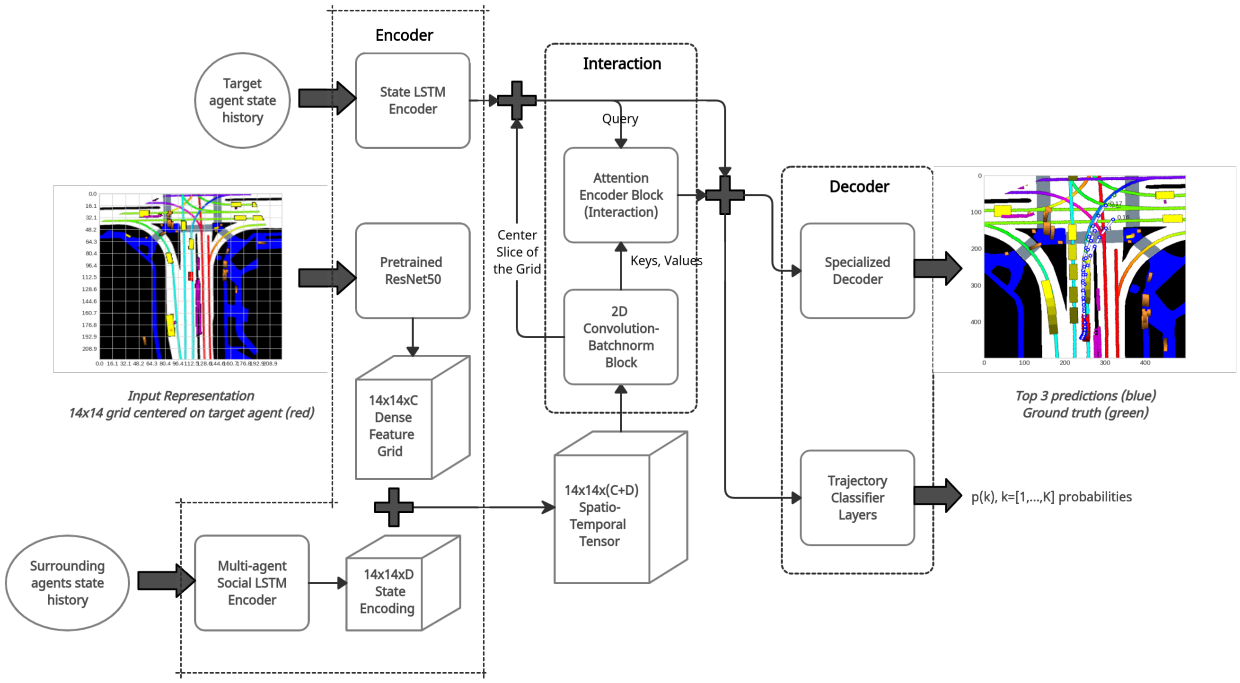

Figure 3.2: Architecture-II: Spatio-temporal Social Encoder(STSE) with Interaction

1. Given a target agent, extract the scene map such that the agent's location is at the center and the map extends to 50 meters in the four directions from the center.

2. Draw color coded boxes for all the agents in the scene with the dimension of the box representing their extent.

3. Fade the colors of the boxes based on the agent state history.

4. Apply an affine transformation to align the map in such a way that the target agent heading aligns with the y-axis.

5. Crop the map such that interaction space is narrowed down to 50 meters to the front, 25 meters to the left and right and 10 meters to the back.

Figure 3.3 illustrates this representation. Various elements in the scene are color coded based on their class and the history of the agents is faded.

Table 3.1 shows the color coding used for the classes throughout this project.

Class	**Color coding**
Target agent	Red
Surrounding agents	Yellow
Pedestrians	Brown
Lanes	Orientation colorized
Drivable area	White
Cross-walk	Grey
Side-walk	Purple
Off-road	Black

Table 3.1: Object classes and color coding

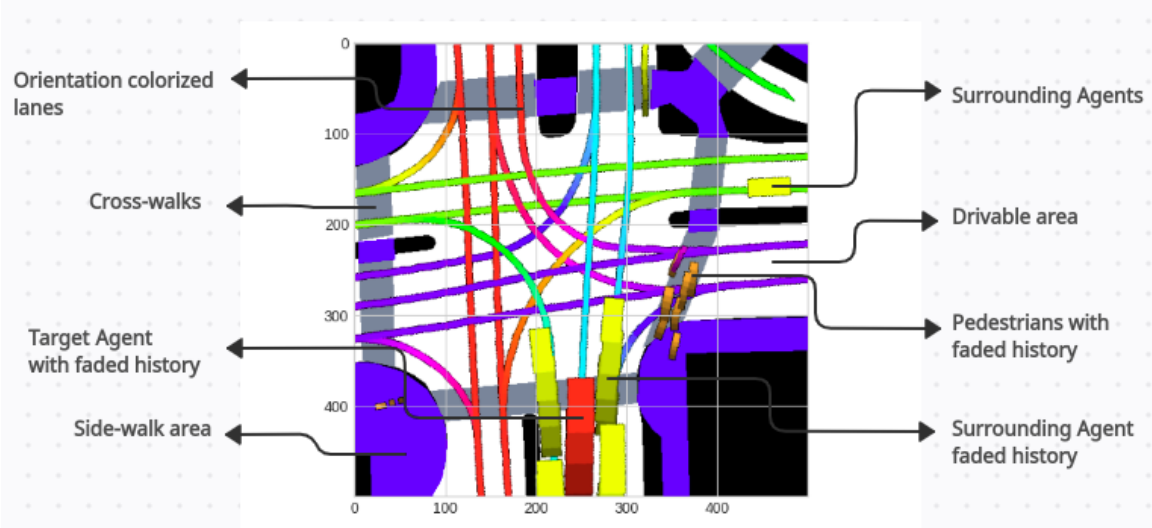

Figure 3.3: Input representation for Arch-I

The actual resolution and other numerical details will be pointed out in the experiments section of the next chapter.

Coming to the input representation for Arch-II, there are a few modifications done over the previous representation due to the change in architecture. The color coding is still the same. One major difference is that the history fading is disabled for a selective class of agents called primary agents, as the history encoding is handled by the social LSTM module for these classes in this architecture. These primary agents include cars, buses, trucks, motorcycles and in general engine-powered vehicles. For these classes, only their current position is rasterized in the map. All other classes such as pedestrians and other moving agents have their history faded in the input representation instead as before, along with their current position encoding. Another major difference is that the map is split onto a 14×14 spatial grid such that each

of the primary agents locations falls under one of the grid locations. This input representation is illustrated in Figure 3.4

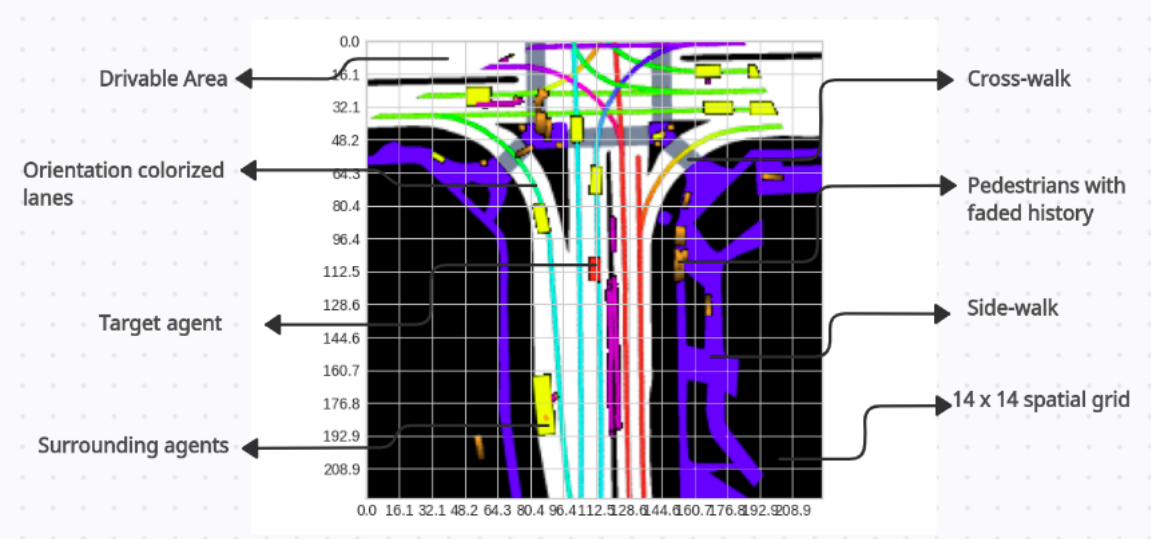

Figure 3.4: Input representation for Arch-II

The steps followed to generate this input representation are

1. Given a target agent, extract the scene map such that the agent's location is at the center of the map and the map extends to 50 meters in the four directions from the center.

2. Draw color coded boxes for all the agents in the scene with the dimension of the box representing their extent.

3. Fade the colors of the boxes only for non-primary agents based on their state history.

4. Apply an affine transformation to align the map in such a way that the target agent heading aligns with the y-axis.

3.2 Encoder design

For the Arch-I model, the encoder mainly comprises of a CNN based feature extractor ResNet50 for map encoding and a variable length state LSTM for target agent encoding as seen in Figure 3.5. The ResNet50 encoder is pretrained on ImageNet [28] and the last fully connected layer is modified to give the expected length of the feature tensor.

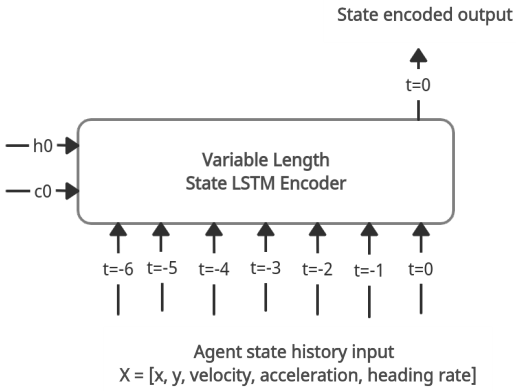

Figure 3.5: Target input state encoding LSTM

Regarding state encoding, only the state history of the target agent is separately encoded via the variable length state many-to-one LSTM encoder seen in Figure 3.5 and the history information of surrounding agents is captured by the CNN feature extractor. The states consist of x, y coordinates, velocity, acceleration and heading rate of the target agent for a given history period. As the states are not guaranteed

to be available throughout the interval, the state history input is zero padded on the right side of the sequence accordingly and is fed to a variable length LSTM, which is designed to ignore the zero-padded inputs. The hidden and cell states are initialized with Xavier initialization. The hidden state of the last LSTM cell ($t = 0$) in the sequence is extracted as the encoded state. We can formulate the encoding for Arch-I model as below.

Let T be the target agent, X_T^t be its state input, M be the map representation and X be the state history information of all agents, then

$$F_T^{raster} = f_\theta(X, M) \in \mathbb{R}^C,$$

$$S_T = LSTM(X_T^t, h_0, c_0) \in \mathbb{R}^D,\ t = [t_0 - t_f, ..., t_0],\ X_T \in \mathbb{R}^{t_f \times 5},$$

$$F_T^{encoder} = (F_T^{raster} \oplus S_T) \in \mathbb{R}^{C+D},$$

where f_θ is the feature extractor (ResNet50) parameterized by θ, F_T^{raster} is the map encoding with feature length C and S_T is the target state encoding with feature length D.

Finally, the dense features from the CNN backbone and the state encoding from LSTM are concatenated to form encoder output features $F_T^{encoder}$.

For Arch-II, the encoder consists of a similar rasterization feature extractor as before, but this time the final fully connected layer is removed to obtain a 14×14 grid encoding tensor. This is done to preserve the spatial relationship between agents. Additionally this time, the state history encoding is done for target agent as well as primary surrounding agents. Motorized agents like cars, buses, and trucks classify as primary agents. All other agents have their history represented in the input representation itself. The state encoding of primary agents is arranged in a similar

grid fashion as the map encoding with the location on the grid corresponding to the location of the agent in the grid representation of the input.

The state encoding is done using a multiagent variable length LSTM seen before, and all the agents share the same weights. This social LSTM output is then concatenated with the dense feature grid tensor along the channels dimension to obtain a spatio-temporal social tensor. This can be formulated as shown.

$$F_T^{grid} = f_\theta(X, M) \in \mathbb{R}^{14\times14\times C},$$

$$S_T = LSTM(X_T^t, h_0, c_0) \in \mathbb{R}^D, X_T \in \mathbb{R}^{t_f\times5},$$

$$S_{ab} = LSTM(X_i^t, h_0, c_0) \in \mathbb{R}^D,\ S \in \mathbb{R}^{14\times14\times D},\ X_i \in \mathbb{R}^{t_f\times5},\ T \notin i,$$

$$F_T^{encoder} = (F_T^{grid} \oplus S) \in \mathbb{R}^{14\times14\times(C+D)},$$

where $i = [1, ..., N]$ are the N surrounding agents, $(a, b) = [1, ..., 14]$ is the 2D grid location of the surrounding agent and $t = [t_0 - t_f, ..., t_0]$ is the history time interval. It is important to note that the target agent T is not in i as it does not belong to any grid given its placed at the center of the grid. S is the social LSTM tensor which is initialized as a tensor of zeros. Encodings for agents that fall in the same location of the grid are aggregated by sum in the social LSTM tensor. $F_T^{encoder}$ is the spatio-temporal social tensor formed by the concatenation of the dense feature grid F_T^{grid} and social LSTM S.

The spatio-temporal tensor $F_T^{encoder}$ and the target state encoding S_T form the output of the encoder, which are passed to an interaction module discussed in the next section.

3.3 Cross-agent and map interaction

Interaction captures the interplay between the target agent and the surrounding agents. It needs to estimate the factor of dependence of a surrounding agent's behavior on the target agent's future trajectory and output a context vector that represents the dependence. Arch-I does not include any interaction component to keep the architecture simple.

Arch-II interaction module mainly has two components as discussed.

Convolution-BatchNorm block:

Similar to the ResNet convolution blocks, this block has two sequential Conv2d-BatchNorm-Activation batch operations as seen in Figure 3.6. The goal of this block is to extract useful features from the spatio-temporal tensor. As this tensor contains information from two different modalities (spatial from CNN and temporal from LSTM), this block learns multiple filters that can extract this multimodal information that will provide the next interaction block with a feature-rich input. We can also use dilated convolution instead of regular convolution here, which could result in better performance.

Attention block:

The second component is the attention block which takes the input from the above block and finally outputs the context vector. There were three different attention mechanisms tried out for this task. Estimating attention boils down to find the weighted combination of the values for a given query. In our case, query is the target agent features and the values are each of the slices of the 7×7 grid obtained from the convolution block. To obtain the query, we concatenate the target state encoding

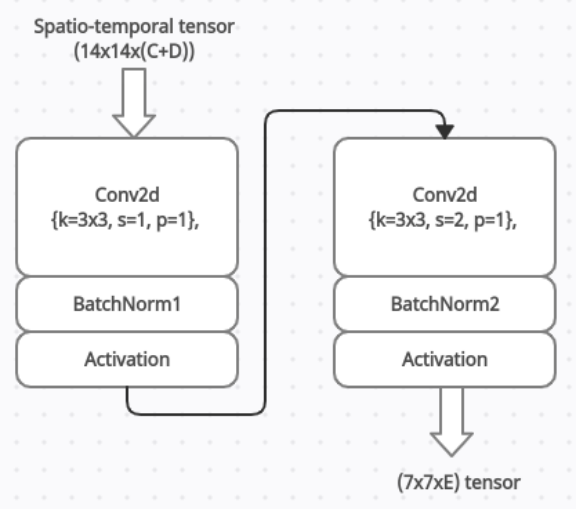

Figure 3.6: Conv2d-BatchNorm block

with the central slice of the grid encoding as it contains information about the target agent.

$$Q_T = S_T \oplus F_{33}^{cb}$$

$$V = F^{cb}$$

$$c_T = \sum_j \alpha_j * V_j, j = [1, ..., 49]$$

where F^{cb} is the output from the convolution-batchnorm block which are the values V in the attention terminology, α_j are the softmax applied attention weights and c_T is the output context vector. The three different mechanisms to calculate the attention weights and thereby the context vector are

1. `Additive attention:` This mechanism uses a single-layer MLP with hyperbolic tangent activation to compute the attention weights as formulated by the attention function.

$$f_{att}(Q_T, V_j) = v_a^T tanh(W_1 Q + W_2 V_j),$$

where W_1 and W_2 are the linear layer learnable weights and v_a is a scaling factor.

2. `Multihead attention:` This mechanism uses a scaled dot-product attention using queries, keys and values. This mechanism has its similarities to the multiplicative attention. It is called multihead because it learns H different attention weights for a given query. This is especially useful in prediction tasks to cover different types of interaction possibilities. This is formulated as

$$Q_l = \theta_l(Q_T^{t_0}; W_{\theta_l}),$$

$$K_l = \phi_l(F^{cb}; W_{\phi_l}),$$

$$V_l = \rho_l(F^{cb}; W_{\rho_l}),$$

where θ_l is fully connected layer parameter applied on the target feature vector, ϕ_l and ρ_l are 1×1 convolution layers applied on the grid tensor.

The output of each attention head is then calculated as a weighted sum of values.

$$c_l = \sum_{a=1}^{7} \sum_{b=1}^{7} \alpha_l(a, b) * V_l(a, b, :),$$

where attention weights α are given by

$$\alpha_l(a, b) = Softmax(\frac{Q_l * K_l^T(a, b, :)}{\sqrt{d}}),$$

where d is the dimension of the heads. Figure 3.7 illustrates this mechanism.

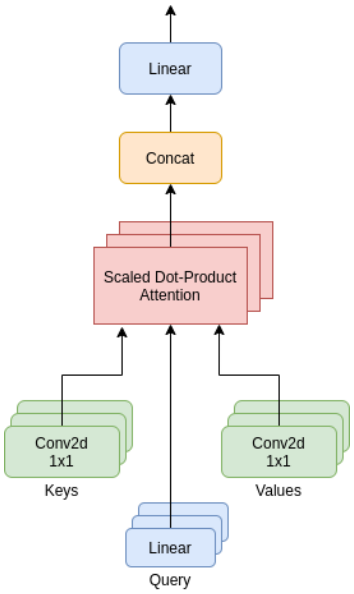

Figure 3.7: Multihead attention block

3. `Transformer Encoder block:` Transformer is primarily built on multihead self-attention. It allows for stacking of multiple such blocks for deep attention. We use the visual transformer (ViT) formulation to implement the attention mechanism [15]. Figure 3.8 is the block diagram of one such block.

   Instead of the keys and values going through convolution layers as done in previous mechanism, here linear layers are used instead with the query, keys and values all being the same. This is also commonly known as self-attention. The target state encoding is merged with the center of the grid encoding using a linear layer concatenation. Self-attention is then applied on these 49 encoded

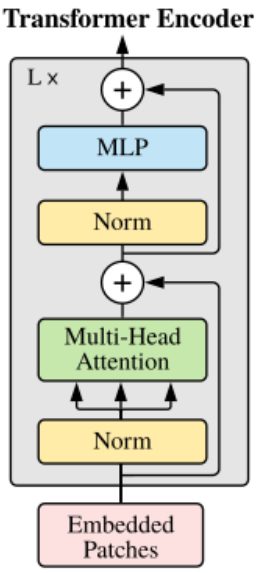

Figure 3.8: Transformer Encoder block [15]

vectors. Similar to the formulation in [15], we add a positional learnable embedding to the input vectors inorder to specify spatial location of the vectors on the grid. These blocks shown in Figure 3.8 are stacked L times before extracting the context vector from the last block. The context vector is extracted from the center location index of the final output tensor.

All these attention mechanisms output a context vector c_T which is then concatenated with the target agent feature Q_T and passed to the decoder as shown.

$$F_T^{(att)} = Q_T \oplus c_T$$

3.4 Decoder design

The decoder takes the input from the interaction module which mainly contains the context vector and the target state features and estimates K different multimodal trajectories and their respective probabilities. In the proposed model, we provide an alternative to the LSTM based decoders that are popular for these tasks by proposing linear basis based decoders that learn coefficients as curve fitting parameters to the future trajectory. This is especially advantageous as during training, we can fit the history and future trajectories with the same parameters which opens opportunities for test-time adaptation as history is always known. Additionally, the number of output parameters are independent of the length of the prediction horizon. This is the core idea behind the decoder design.

Most of the decoder architecture elements are the same for Arch-I and Arch-II except for a few improvements. Hence, it will be discussed in unison. The decoder is mainly made up of two fully connected layers with an activation layer in between and a specialized basis matrix inner product operation. The final fully connected layer basically predicts the coefficients of the specialized basis vector to estimate the future x and y positions of the trajectory. This can be thought of as a curve fitting task on the history and future trajectories. Let $Linear_i$ be the i-th fully connected layer of the decoder. We use the Rectified Linear Unit denoted as $ReLU$ as the nonlinear activation function and a softmax operation denoted as $Softmax$ to normalize the probability estimates. The decoder is formulated as

$$h = Linear_1(F_T^{(att)}),$$

$$h = ReLU(h),$$

$$h_{coeff} = Linear_2(h) \in \mathbb{R}^{K \times d \times 2},$$

$$Y_t^{xy} = h_{coeff} * \Theta_t \in \mathbb{R}^{K \times 2}, \Theta \in \mathbb{R}^{t_f \times d \times 2},$$

$$p_k = Softmax(Linear_p(F_T^{(att)})) \in \mathbb{R}^K,$$

where h is the first hidden layer output and h_{coeff} are the predicted coefficients to the basis matrix Θ, K is the number of unique trajectories, d is the degree of the basis matrix and Y_t^{xy} are the estimated x and y coordinates of K trajectories for a given timestep t, p_k are the respective probabilities of each trajectory and t_f is the future prediction horizon.

The different basis decoders that were explored starting with the basic ones are

- **Polyfit:** Polynomial fitting is a popular and straight forward approach to curve fitting. It derives from the Taylor expansion and can be formulated as

  $$f(t) = \sum_{n=0}^{N} a_n * t^n,$$

  where N is the degree of the polynomial and a_n are the coefficients that are estimated by the final decoder layer. $N = 2$ which is quadratic, is generally not precise enough for complex curves. We found $N > 4$ to give optimal estimates through the curve fitting experiments.

- **Taylor/MacLaurin series based fitting:** This uses the MacLaurin series to fit the trajectory wherein the coefficients learnt are the derivatives and the evaluation point.

$$f(t) = \sum_{n=0}^{N} \frac{f^{(n)}(a)}{n!}(t-a)^n,$$

where a is known as the evaluation point. The terms $\frac{f^{(n)}(a)}{n!}$ and a are learnt by the decoder in order to predict the future co-ordinates. One key observation from our experiments was that this learning of the evaluation point a helps improve the performance of polynomial fitting.

- **Orthogonal Polyfit:** This uses a class of orthogonal polynomials for curve fitting. Any two different polynomials generated by this class have their inner product equal to 0.

$$\int P_a(x)\, P_b(x)\, dx = 0,\ a \neq b$$

In other words, a column or row basis of the basic matrix is orthogonal to every other column or row basis. This provides an improvement over polynomial fitting as regular polynomials tends to suffer from covariance between different degree terms while performing fitting.

In this project, the Orthonormal Legendre class of orthogonal polynomials were chosen due to their generalization as plotted in Figure 3.9. These are orthonormal basis vectors as their Eigen values are one. Below is the formulation where the basis vectors can be found using the Rodrigues' formula although many other methods exist for calculating the same

$$f(t) = \sum_{n=0}^{N} a_n * P_n(t),$$

$$P_n(t) = \frac{1}{2^n\, n!} \frac{d^n}{dt^n}(t^2 - 1)^n, n = [0, ..., N],$$

where P_n are the basis vectors of degree $0, ..., N$ and a_n are the estimated coefficients by the decoder.

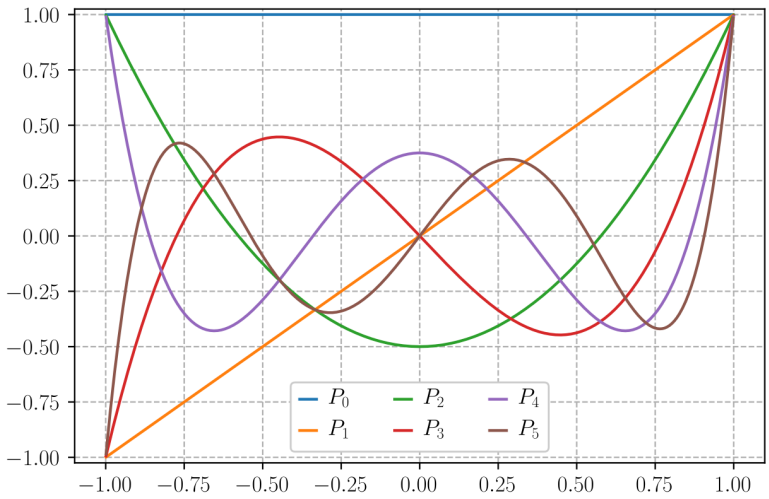

Figure 3.9: First six Legendre basis functions used in the model [7]

- **Discrete Fourier Transform (DFT) based fitting:** These are another class of orthogonal basis functions that are generated in the frequency domain. Fourier transform is especially interesting for this use-case due to its complex domain formulation. We can thereby use a complex representation for the x and y coordinates and operate in the complex domain for decoding. That is, we use a 2 dimensional complex coordinate space for prediction defined as

$$C^2 = [(x, y)|x, y \in C, z = x + iy].$$

Fourier based fitting can be well explained with the Discrete Fourier Transform (DFT) formulation. The DFT operation can be defined as

$$F_k = \sum_{n=0}^{N-1} f_n * e^{-i\omega_0 n},$$

$$f_n = \frac{1}{N} \sum_{n=0}^{N-1} F_k * e^{i\omega_0 n},$$

where $w_0 = \frac{2\pi}{N}$. F_k are the Fourier coefficients and the exponential terms act as orthogonal basis functions. f_n represents the reconstructed signals in the time domain with N being the periodic length of the function.

The Fourier approximation of a given periodic function with period $p = 2L$ can be defined in terms of sinusoids as

$$f(t) = A_0 + \sum_{n=1}^{\infty} (A_n \cos(\frac{n\pi}{L}t) + B_n \sin(\frac{n\pi}{L}t)),$$

where the Fourier coefficients can be found by Euler formulae as follows

$$A_0 = \frac{1}{2L} \int_{-L}^{L} f(t)dt,$$
$$A_n = \frac{1}{L} \int_{-L}^{L} f(t) \cos(\frac{n\pi}{L})dt,$$
$$B_n = \frac{1}{L} \int_{-L}^{L} f(t) \sin(\frac{n\pi}{L})dt.$$

By expressing this in the complex form we get

$$f(t) = \sum_{-\infty}^{\infty} c_n * e^{int},$$

where

$$c_n = A_n - iB_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t) * e^{-int} dt.$$

Here, c_n which are the complex coefficients are estimated by the final decoder layer and an inner product with the complex Fourier basis gives the required x and y coordinates in the complex space. The fast Fourier implementation of DFT known as Fast Fourier Transform (FFT) was used in the model to speed up computation.

For Arch-II, instead of learning the coefficients for K trajectories in the same decoder layers, K different decoder layers with shared weights can be setup and the input to the linear layers is fed from K attention head outputs obtained from the multihead attention modules discussed in the previous section. This can help the decoder address different driving scenarios encountered in a given scene.

Decoder curve fitting analysis:

The curve fitting capability of the decoder variants is analyzed quantitatively with the mean squared error and qualitatively by plotting the x and y coordinates and the parameterized curve. The mean squared error for polynomial and orthogonal curve fitting is around 0.11 and 10^{-4} for FFT and around 0.23 for Discrete Cosine Transform (DCT) which is an optimized real version of DFT. Figure 3.10 displays the qualitative comparison of the different encoders. From the analysis, FFT turns out to be the best performing curve fitting basis.

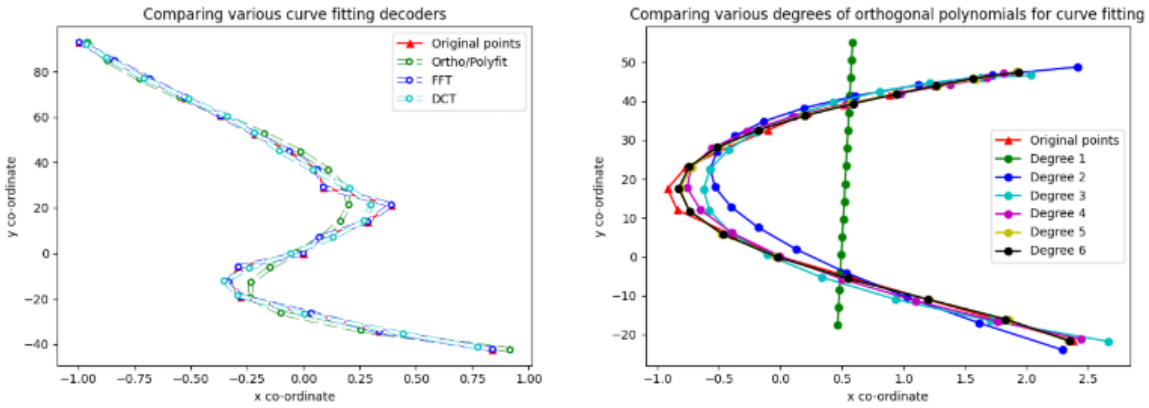

Figure 3.10: Qualitative evaluation of curve fitting decoders

3.5 Output trajectory representation

The final output from the decoder is a set of K unique trajectories with x and y co-ordinates for t_f time-steps and K probability values for the trajectories. Figure 3.11 is a render of 10 multimodal trajectory predictions and their respective probabilities compared against the ground truth (GT) for a randomly sampled scene.

3.6 Optimization objectives

Given the task in hand is an optimization problem, designing a good objective function is crucial to the model performance. Both Arch-I and Arch-II use similar loss functions for the network. We use a multimodal regression based loss function first proposed in the MTP paper [14].

We need to estimate $p(Y_k|X, M)$ distribution with k being one of the K trajectories and X and M being the agent history states and map representation respectively. As

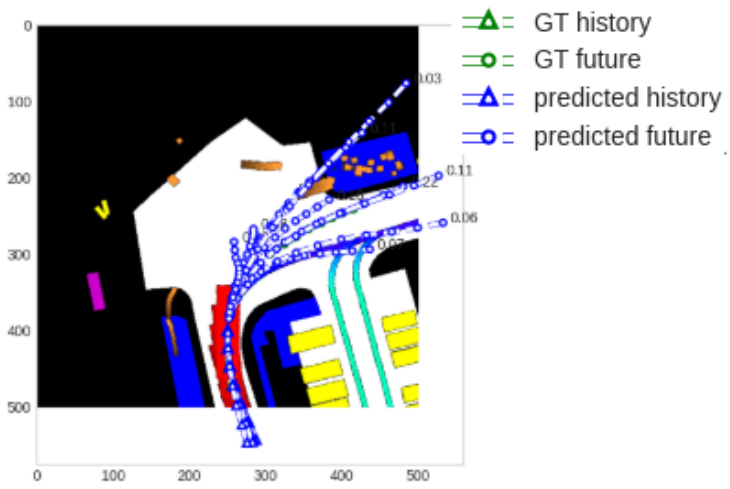

Figure 3.11: 10 trajectory predictions from the model with associated probabilities

the network outputs curve coefficients that fit the history and future, we use two different losses for the same.

For history loss, mean square error (MSE) over all the K history predictions is used as shown.

$$loss^{history} = \frac{1}{K}\frac{1}{t_h}\sum_{k=1}^{K}\sum_{t=t_0-t_h}^{t_0}(Y_t - \tilde{Y}_{kt})^2 * \delta_t,$$

where t_h is the number of history timesteps, $\tilde{Y}_{kt}$ is prediction of trajectory k for timestep t, Y_t is the ground truth (GT) history at t and δ_t is a binary mask indicating history availability at point t. As the history is already known, this loss forces all the K trajectory history predictions to follow the ground truth history.

Loss for future predictions is the combination of regression and classification. For regression, a similar MSE loss is used but only the prediction closest to ground truth (GT) future is included in the loss. Minimum Euclidean distance is used as a metric

for determining the closest trajectory to the ground truth. For classification, cross-entropy loss is used between the closest to GT prediction label and the K probability estimates. This is formulated as

$$k^* = \arg\min_k \frac{1}{t_f} \sum_{t=t_0}^{t_f} (Y_t - \tilde{Y}_{kt})^2 * \delta_t,$$

$$loss_{reg}^{future} = \frac{1}{t_f} \sum_{k=1}^{K} \sum_{t=t_0}^{t_f} I(k = k^*) * (Y_{kt} - \tilde{Y}_t)^2 * \delta_t,$$

where k^* represents the closest trajectory to ground truth and I is a delta function equal to 1 when $k = k^*$. This loss forces the best matching trajectory to follow ground truth without affecting other trajectory predictions. The classification loss is defined as

$$loss_{cls}^{future} = -\sum_{k=1}^{K} I(k = k^*) * \log p_k,$$

where p_k is the probability estimate of trajectory k.

In addition to these losses, an off-road loss is also proposed that enforces the predictions to be in the drivable area and to follow the outgoing lanes.

1. `Distance to drivable area:` If a prediction falls in the drivable area, then this term is 0. If it does not, the Euclidean distance between the prediction and the closest drivable point is added to this loss.

   $$loss_{or}^{da} = \frac{1}{K} \frac{1}{t_f} \sum_{k=1}^{K} \sum_{t=t_0}^{t_f} d(Y_{tk}^{cd}, \tilde{Y_{tk}}),$$

   where Y_{tk}^{cd} is a location on drivable area closest to the prediction point.

2. `Distance to closest outgoing lane:` This term is applied to all trajectories except the trajectory closest to ground truth. Given a predicted trajectory,

the closest outgoing lane to the last few points of that trajectory prediction is found and then the distance between each trajectory point and its corresponding closest point on the lane is calculated and averaged to get this loss term.

$$L_k^{l^*} = \arg\min_l \, d(\tilde{Y}_k, L_k^l),$$

$$loss_{or}^{cl} = \frac{1}{K}\frac{1}{t_f}\sum_{k=1}^{K}\sum_{t=t_0}^{t_f} d(\tilde{Y}_{tk}, L_{tk}^{l^*}),$$

where L_k^l is the set of outgoing lanes for $\tilde{Y}_k$ and $L_k^{l^*}$ is the closest lane to the last few points of the trajectory.

Hence, the total loss can be expressed as

$$loss = loss_{reg}^{future} + \alpha\, loss_{cls}^{future} + \beta\, loss^{history} + \gamma(loss_{or}^{da} + loss_{or}^{cl}),$$

where α, $\beta\, and\, \gamma$ are hyper-parameters. The model is trained till convergence using batch gradient descent with gradients calculated from the back-propagation algorithm. More implementation details are given in the Chapter 4.

3.7 Test-time adaptation with meta-learning

Due to the fact that during training, the decoder uses a curve fitting approach to fit history and future trajectories with the same parameters, during testing with only history known, we can apply test time adaptation on the history predictions to correct the history loss. This is done by fine-tuning the decoder layers. This fine-tuning helps the model adapt its future trajectory predictions to a given scene environment. This method holds the assumption that correcting the history predictions should improve

the future predictions too, as the history and future trajectories are parameterized by the same coefficients of the basis. Figure 3.12 is an illustration of a basic test-time adaptation setup.

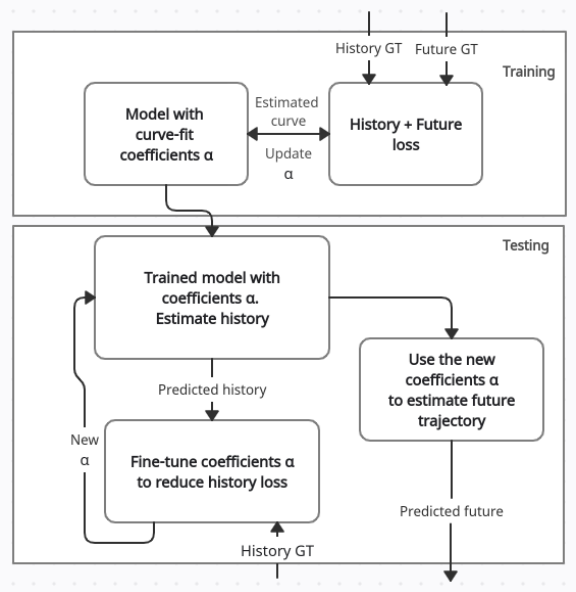

Figure 3.12: Basic test-time adaptation

Through the experiments, it was observed that with the above approach, the gradients of the coefficients are too small and it takes many steps to reduce the history loss. A solution to this is test time adaptation with meta-learning. Meta-learning is a specialized training procedure that allows the model to generalize across multiple tasks. Hence, it performs very well in few shot learning when few examples per class are available. It has been applied to plenty of multitask few shot learning tasks with improvement in performance. The algorithm proposed was inspired by the work done in [16].

The reason to apply meta-learning for the current task is to improve the test-time update of model parameters by replicating this update during training. In the meta-learning terminology, we use the trajectory history prediction as the support set and the full trajectory (history + future) prediction as the query set. The meta-learning algorithm proposed for this task is shown.

Algorithm 1 Training loop for meta-learning on history and future trajectory predictions

1. Initialize data-loader with batch size N
2. Load a pretrained model trained in the traditional way with decoder params θ
3. **while** not converged:
4. Sample a batch at random
5. **for** each sample X_i in the batch, do
6. **for** 1 to H epochs, do
7. Evaluate history loss $L_{X_i}^{history}$ on support set with θ
8. Perform decoder parameter θ update via gradient descent:

$$\theta_i^{'} = \theta - \alpha \nabla_\theta L_{X_i}^{history}(f_\theta)$$

9. Evaluate the complete loss L_{X_i} with the updated $\theta_i^{'}$ on query set
10. **end for**
11. **end for**
12. Perform an outer loop update of original parameters θ

$$\theta = \theta - \beta \nabla_\theta \sum_{X_i} L_{X_i}(f_{\theta_i^{'}})$$

13. **end while**

where line 6 - 9 is the inner-loop update which uses the history loss to force all history predictions to follow the ground truth and Line 12 is the outer loop update of the meta-parameters which uses the full-fledged loss to adapt the model parameters to the update caused by the inner loop optimization.

Note that as this algorithm has an inner loop gradient update along with an outer loop update, it involves the calculation of second order gradients. Hence, it can be slow to train this for larger models. There is also a first-order approximation to the algorithm which, although is faster, compromises on performance for this task. For this task, only the decoder parameters undergo the meta-update in order to speed up the training. More details and results are discussed in the next chapter.

Chapter 4: Experiments and Evaluation

This chapter first introduces the datasets used to train and evaluate the motion forecasting task. This is followed by formulation of evaluation metrics used to compare different models. We then discuss the baseline models to compare against and then move onto the core section containing details about the six primary experiments, setup and evaluation results.

4.1 Datasets

There are quite a few open-source datasets available for autonomous driving research. However, very few options exist specifically for the task of motion forecasting as it involves a lot of human annotated data. Below are the two datasets that were utilized to train the proposed model on.

NuScenes by Motional

The nuScenes dataset is a public large-scale dataset for autonomous driving developed by the team at Motional (formerly nuTonomy) [11]. A few key features of the dataset are

- It consists of 1000 scenes of 20 seconds long, from Boston and Singapore. The area map covered can be seen in Figure 4.1.

- It provides data from the entire sensor suite of an autonomous vehicle (6 cameras, 1 LIDAR, 5 RADAR, GPS, IMU). Multimodal sensor outputs are seen in Figure 4.2 and the sensor placement on the ego-car can be seen in Figure 4.3.

- The full dataset includes approximately 1.4 million camera images, 390 thousand LIDAR sweeps, 1.4 million RADAR sweeps and 1.4 million object bounding boxes in 40 thousand keyframes.

- It contains 23 categories including different vehicles, types of pedestrians, mobility devices, and other objects. Per keyframe, there are 7 pedestrians and 20 vehicles on average. Figure 4.4 shows one of the maps with the colorized road lanes overlayed on top.

- It comes with the nuscenes-devkit containing useful apis to get started with motion forecasting tasks.

Level5 by Lyft:

Lyft Level5 dataset is another large-scale dataset for motion forecasting provided by Lyft [21]. A few key features of this dataset are

- It contains 1,000 hours of traffic scenes capturing the motions of traffic participants around 20 self-driving vehicles, driving over 26,000 kms along a suburban route.

- It also provides a high-resolution aerial image of the area, spanning $74\,km^2$ at a resolution of $6\,cms/pixel$, providing further spatial context about the environment. This is shown in Figure 4.5. The sensor suite placement on the ego car is seen in Figure 4.6.

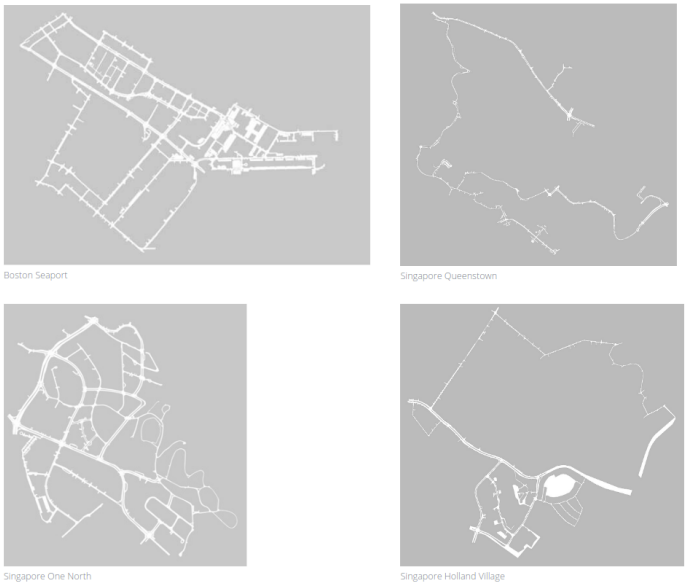

Figure 4.1: Top view of the 4 areas mapped by the ego agents [11]

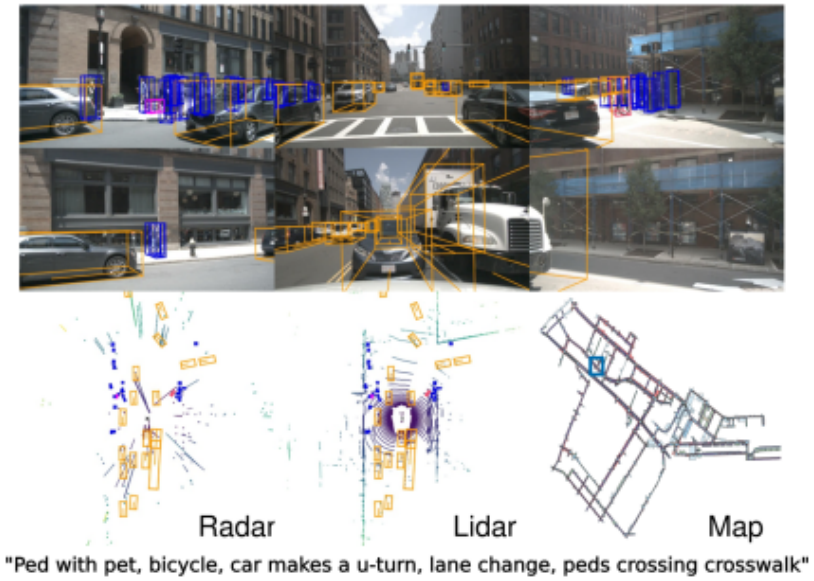

Figure 4.2: Sensor suite: 6 different camera views, lidar, radar data, human annotated semantic map [11]

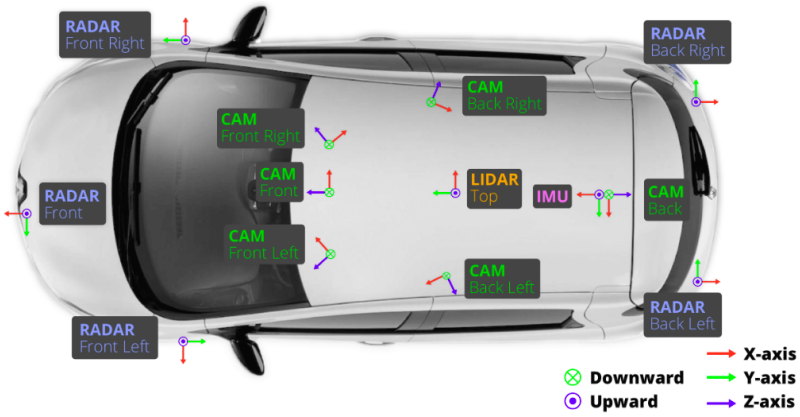

Figure 4.3: Sensor placement layout on the ego agent [11]

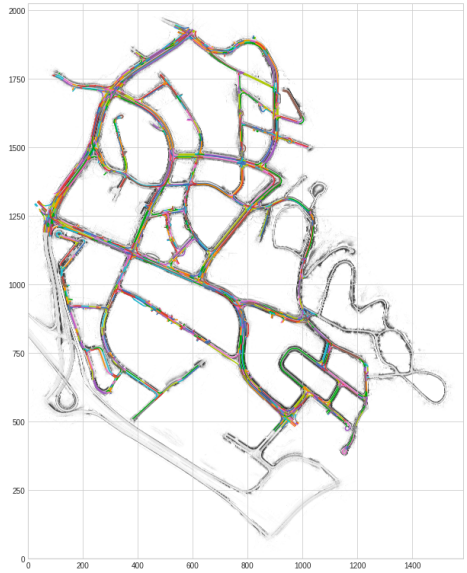

Figure 4.4: Colorized lanes annotated for one of the map locations [11]

- It includes L5Kit - a Python software library for accessing the dataset, together with a baseline machine learning solution for the motion prediction task.

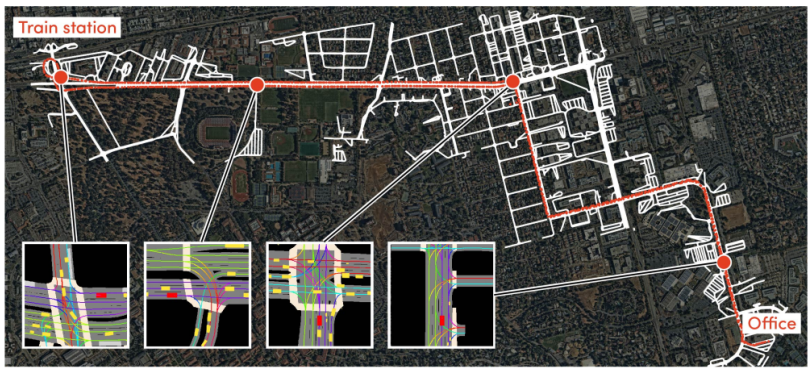

Figure 4.5: Top view of the area mapped by 20 ego agents for Lyft dataset [21]

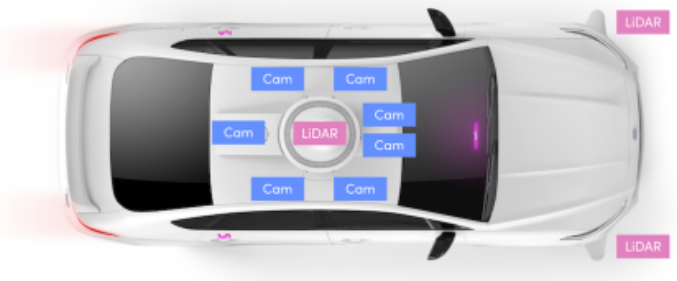

Figure 4.6: Sensor placement layout on the ego-agent for Lyft dataset [21]

NuScenes was selected as the primary dataset for this task as it had an easier to use development kit in the form of nuscenes-devkit and had easy access to metadata such as drivable area masks, outgoing lanes from a moving point and so on. All the

evaluation results to follow were obtained as a result of training and cross-validating the proposed model on this NuScenes dataset using the standard train-val split.

4.2 Evaluation metrics

The evaluation metrics used to compare the performance of the model prediction with other state of the art models and baselines are formulated here. These metrics take as input the ground truth future trajectory and K predicted future trajectories and report an average value over all the N test dataset samples.

- $minADE_k$**:** Computes the average displacement error over the prediction horizon t_f for all the K predictions and then returns the minimum from the top k predictions averaged over all the data samples.

$$minADE_k = \frac{1}{N}\sum_{n=1}^{N} min(top_k(\frac{1}{t_f}\sum_{t=t_0}^{t_0+t_f} ||Y_t - \tilde{Y}_{kt}||_2, p(k)))$$

  where $p(k)$ is the probability estimate of trajectory k output by the decoder.

- $minFDE_k$**:** Computes the final displacement error at the horizon timestep t_f for all the K predictions and then returns the minimum from the top k predictions averaged over all the data samples.

$$minFDE_k = \frac{1}{N}\sum_{n=1}^{N} min(top_m||Y_{t_0+t_f} - \tilde{Y}_{k(t_0+t_f)}||_2, p(k))$$

- *Negative Log Likelihood (NLL):* NLL assumes a multidimensional independent normal distribution over time for the ground truth positions, yielding the likelihood estimate shown.

$$p(Y|C,\tilde{Y}) = \sum_k C_k \prod_t \mathcal{N}(Y|\tilde{Y}_{tk}, \sigma = 1),$$
$$NLL = -\log(p(Y|C,\tilde{Y})),$$

where, C contains the probability distribution of the k trajectories earlier denoted as $p(k)$ output by the decoder.

- $missRate_{k,d}$**:** For a given distance d, and the top k probable predictions generated by the model, the set of k predictions is considered a miss based on

$$missRate_{k,d} = \begin{cases} 1, & if\ min_k(max_t\ ||Y_t - Y_{tk}||_2) \geq d \\ 0, & \text{otherwise} \end{cases}$$

- *Off-road rate:* Measures the fraction of predicted trajectories that fall outside the drivable area of map.

4.3 Baselines, state of the art models and evaluation

This section and the next section reports the above evaluation metrics of different models trained and evaluated on the NuScenes prediction dataset.

In this section, the baseline models and their performance on the NuScenes dataset are listed, followed by the state of the art models and their performance on the dataset.

Baselines on NuScenes:

Following three models were selected as baselines for the evaluation.

1. `Constant velocity and yaw with Kalman filtering`
2. `Physics oracle`: Picks the best out of different physics based models

3. `Unimodal version of Arch-I with specialized decoder disabled (Uni-sndec):` This is a simplified Arch-I model which estimates only one trajectory instead of K different trajectories and has the specialized decoder disabled.

State of the art models on NuScenes:

Following five models were selected from the literature as the state of the arts for evaluation based on their performance on the described metrics.

1. `MTP:` Proposes the MTP loss based on multimodal regression
2. `Multipath:` Proposes predefined trajectory anchors with learnt offsets and GMM decoder
3. `CoverNet:` Similar to Multipath but with dynamic trajectory generator
4. `Trajectron++:` CVAE based design with graph representation and dynamic integration
5. `MHA-JAM:` Joint agent-map representation and multi-head attention

Evaluation:

The evaluation task is split into quantitative analysis, qualitative analysis, and observation. The quantitative evaluation is done using the evaluation metrics discussed in the metrics section. Models with lower values for the metrics are better performing models. Qualitative evaluation is done by considering scenes from the dataset with different kinds of interaction scenarios. Below are the four different scenarios that were shortlisted to qualitatively compare all the models.

1. `Scene-1:` High speed sharp turn for the target agent

2. `Scene-2:` Target agent in a roundabout

3. `Scene-3:` Target agent in a congested scene with slower speeds

4. `Scene-4:` Target agent at an intersection

The observation section discusses the inference results from the quantitative and qualitative evaluation.

4.3.1 Baseline model performance

Here, we run the quantitative and qualitative evaluation schemes discussed above to compare the different baseline models that were shortlisted earlier.

Quantitative evaluation:

Model	$minADE_1$	$minADE_5$	$minADE_{10}$	$minFDE_1$	$minFDE_5$	$minADE_{10}$
Constant vel-yaw	4.61	4.61	4.61	11.21	11.21	11.21
Physics oracle	3.71	3.71	3.71	9.09	9.09	9.09
Uni-sndec	3.75	3.75	3.75	8.65	8.65	8.65

Table 4.1: Baseline models evaluation metrics-I

Model	$MissRate_{5,2}$	$MissRate_{10,2}$	Off-road Rate
Constant vel-yaw	0.91	0.91	0.14
Physics oracle	0.88	0.88	0.12
Uni-sndec	0.91	0.91	0.08

Table 4.2: Baseline models evaluation metrics-II

Qualitative evaluation:

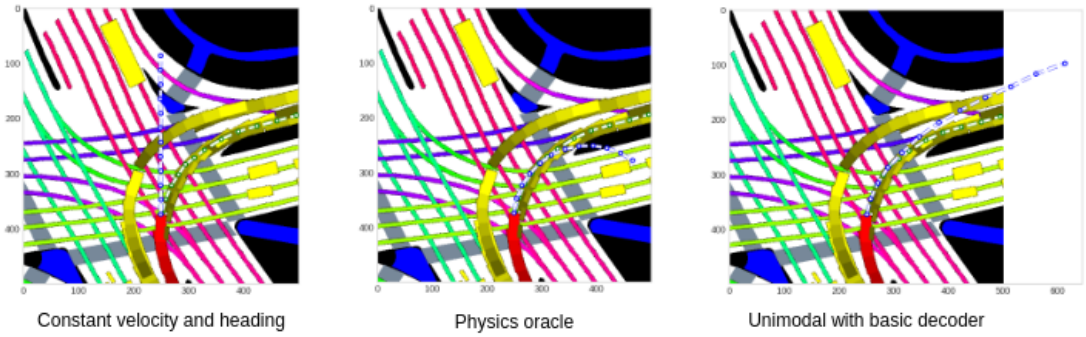

Figure 4.7: Scene-1 evaluation between the baselines

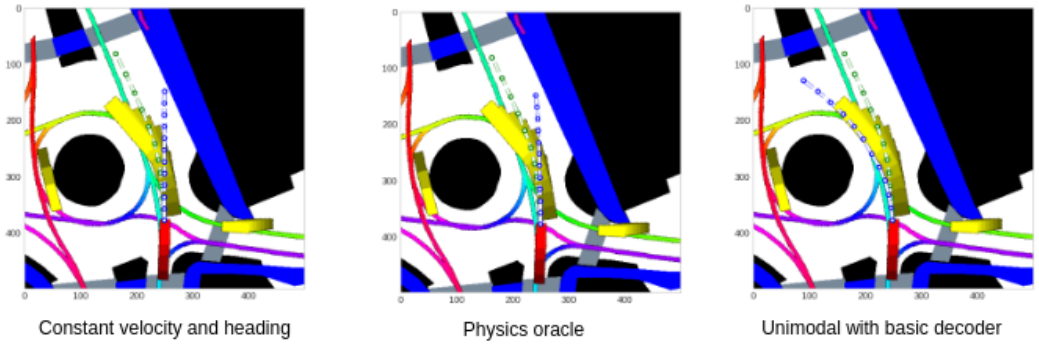

Figure 4.8: Scene-2 evaluation between the baselines

Observations:

- From the quantitative metrics, it is clear that the constant velocity and yaw rate model does not perform well compared to the other two baselines. The physics oracle and machine learning based `Uni-sndec` single mode predictor

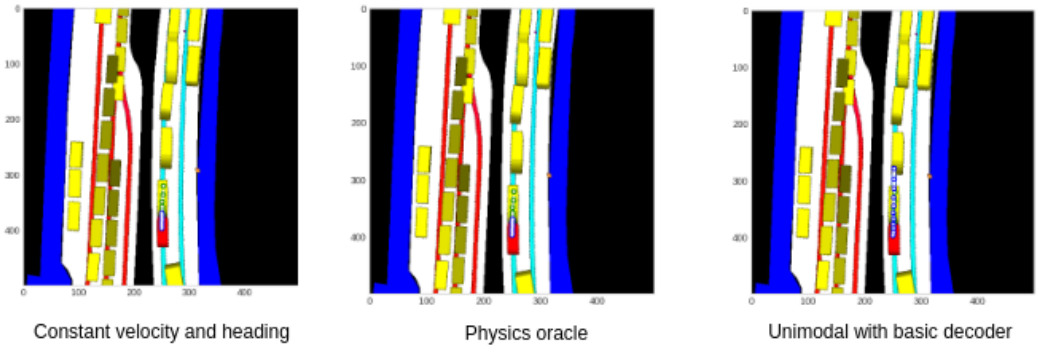

Figure 4.9: Scene-3 evaluation between the baselines

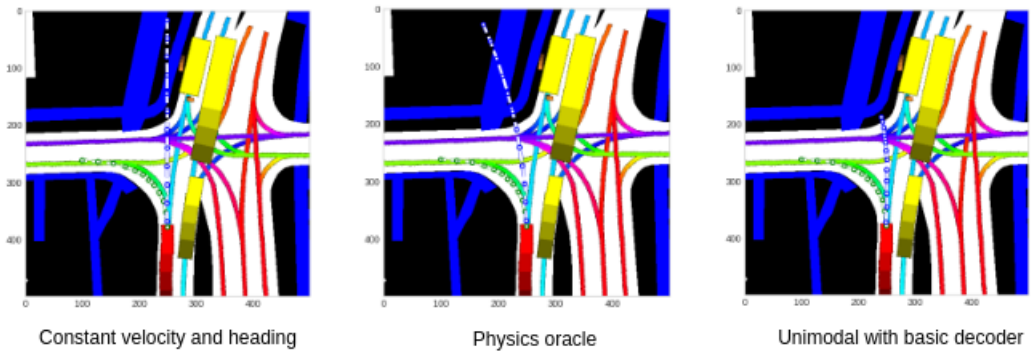

Figure 4.10: Scene-4 evaluation between the baselines

have similar $minADE$ metrics but `Uni-sndec` does better on $minFDE$ metric which is expected as it is a data-driven model. The classical based methods are prone to exponentially rising errors with longer horizon timesteps. With neural network based models such as `Uni-sndec`, the rise in error is not as significant.

- `Uni-sndec` does better on off-road rate which is again expected given the map encoding.

- On the qualitative comparison front, `constant vel-yaw` does not do very well. `Physics-oracle` does fairly well in Scene-1 but fails in the rest of the scenes. `Uni-sndec` does well in all except Scene-4, which requires a multimodal approach.

4.3.2 State of the art models performance

Tables 4.3 and 4.4 list the performance of the state of the art models evaluated on the same metrics discussed above. The goal of the proposed models is to perform at par or even beat the performance of these state of the arts (SOTA). From the tables, it is clear that `MHA-JAM` is the best performing SOTA model.

Model	$minADE_1$	$minADE_5$	$minADE_{10}$	$minFDE_1$	$minFDE_5$	$minADE_{10}$
MTP	4.42	2.22	1.74	10.36	4.83	3.54
MultiPath	4.43	1.78	1.55	10.16	3.62	2.93
CoverNet	-	2.62	1.92	11.36	-	-
Trajectron++	-	1.88	1.51	9.52	-	-
MHA-JAM	3.69	1.81	1.24	8.57	3.72	2.21

Table 4.3: State of the art models evaluation metrics-I [24]

Model	$MissRate_{5,2}$	$MissRate_{10,2}$	Off-road Rate
MTP	0.74	0.67	0.25
MultiPath	0.78	0.76	0.36
CoverNet	0.76	0.64	0.13
Trajectron++	0.70	0.57	0.25
MHA-JAM	0.59	0.45	0.07

Table 4.4: State of the art models evaluation metrics-II [24]

4.4 Experiments, setup and evaluation

The experimental setup details for the experiments are reported here:

- All models were trained on the NuScenes prediction dataset by using the standard train-val split for training and validation respectively.
- Input raster size used for Arch-I was $500 \times 500 \times 3$ and for Arch-II was $224 \times 224 \times 3$.
- 3 seconds of agent state history was used as input to predict 6 seconds of future trajectories with a sampling frequency of 2 Hz.
- The models were trained for 15 epochs with a batch size of 16 on a Nvidia RTX 2080Ti GPU and the training took around 4 hours.
- Adam optimizer with a small weight decay of 10^{-4} was used.
- LR cyclic scheduler was used to set the learning rate starting from 10^{-4} to 10^{-3} and back. We also experimented with gradient clipping for greater stability in training.

- A small subset of the training split was used for hyper-parameter tuning of learning rate and the loss weighting parameters.

We now discuss the six experiments that were conducted to evaluate and validate the two proposed architectures Arch-I and Arch-II. The experiments are conducted in a sequential manner starting from the basic design of Arch-I all the way to the full fledged Arch-II model. Each experiment section consists of a model design block where we discuss the model setup being used, followed by the quantitative and qualitative evaluation and concluded with observation remarks where the inference from the evaluation is discussed.

4.4.1 Experiment 1

The first experiment implements the basic form of Arch-I model having raster and target state encoding, a basic decoder with specialized decoder layers disabled and a multimodal loss formulation.

Model design:

- This experiment uses the Arch-I model with a basic decoder as shown in Figure 4.11. The gray blocks in the figure indicate elements that are disabled.
- It uses the multimodal loss formulation with only future regression and classification loss terms and no history loss or off-road loss terms.
- The specialized decoder layer is disabled resulting in a basic version of the decoder learning x and y coordinates directly.
- It uses the target agent variable length LSTM encoder for agent history encoding and ResNet50 for map encoding.

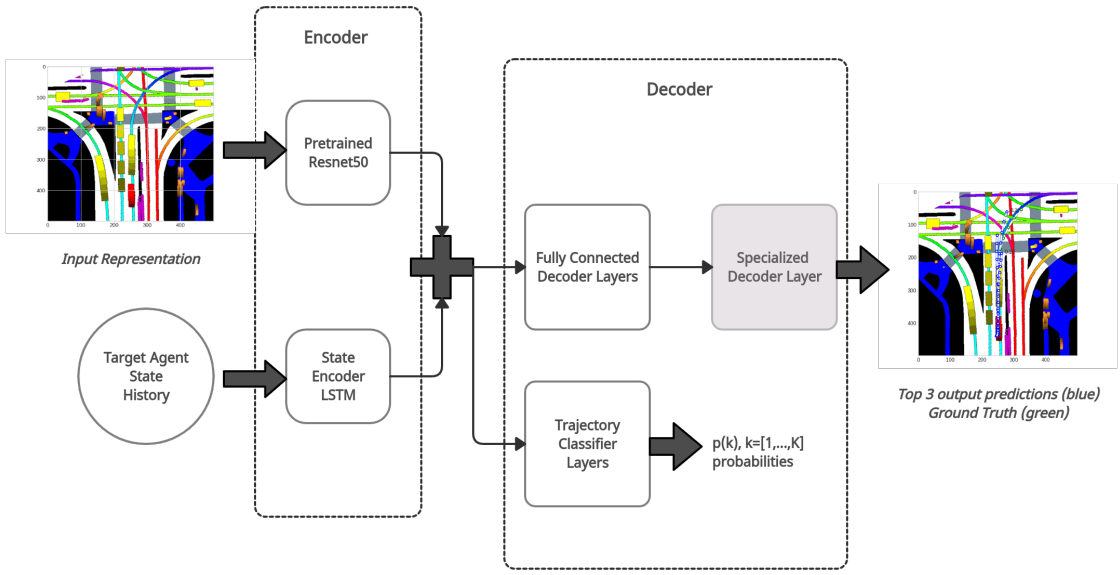

Figure 4.11: Exp-1 model setup

Quantitative evaluation:

Table 4.5 and 4.6 compare the quantitative performance of this model (`Exp-1-basic`) against the unimodal baseline `Uni-sndec` evaluated earlier.

Model	$minADE_1$	$minADE_5$	$minADE_{10}$	$minFDE_1$	$minFDE_5$	$minADE_{10}$
Uni-sndec	3.75	3.75	3.75	8.65	8.65	8.65
Exp-1-basic	4.35	1.74	1.34	9.73	3.52	2.52

Table 4.5: Exp-1-basic evaluation metrics-I

Qualitative evaluation:

In Figure 4.12, we look at the selected four scene evaluations visually.

Model	$MissRate_{5,2}$	$MissRate_{10,2}$	Off-road Rate
Uni-sndec	0.91	0.91	0.08
Exp-1-basic	0.64	0.62	0.24

Table 4.6: Exp-1-basic evaluation metrics-II

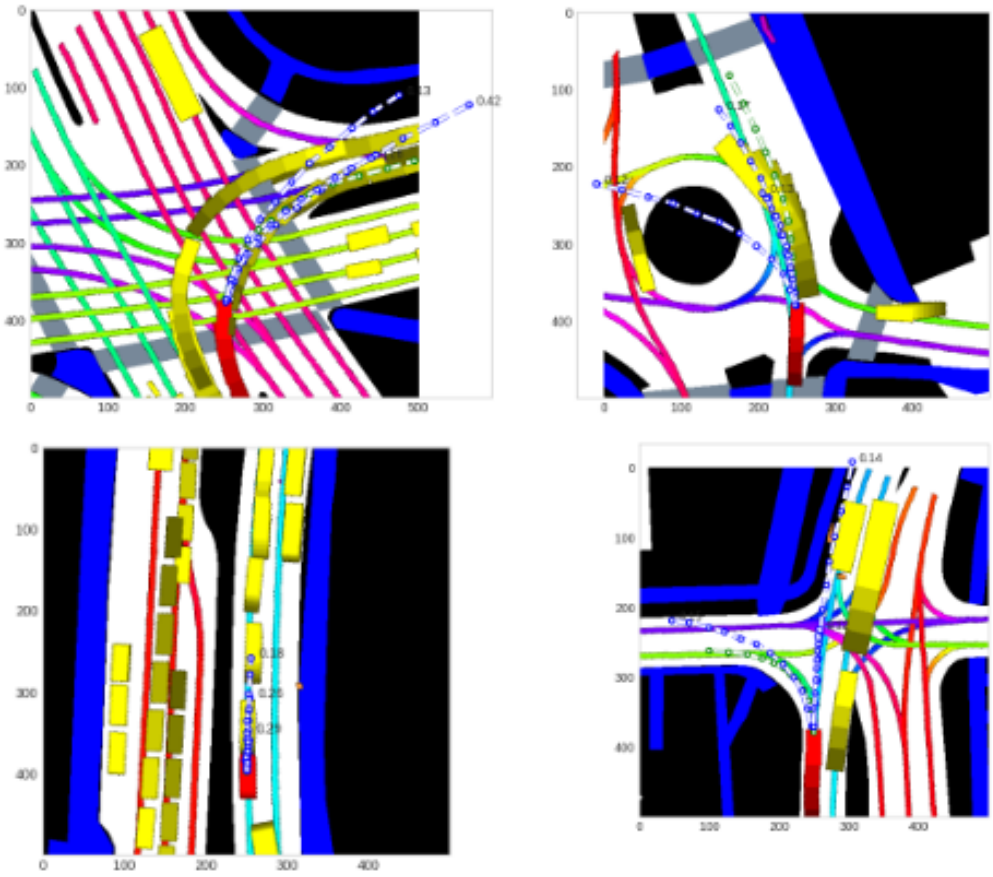

Figure 4.12: All 4 scene evaluations on Exp-1 model

Observations:

- Quantitatively, although $minADE_1$ and $minFDE_1$ are not as good as the unimodal baseline `Uni-sndec`, the multimodal output allows the model to perform extremely well under the other columns of the same metrics.
- MissRate has significantly improved now that the trajectories are multimodal and cover various driving scenarios
- Off-road rate has got worse due to some of the multimodal predictions falling off-road to remain diverse.
- Qualitatively, the model does well on all 4 scenes by covering the different possible maneuvers with multimodal predictions. In Scene-3, it predicts the slow down of the vehicle due to congestion and in Scene-4, although it predicts a left turn, the lane chosen is incorrect.

4.4.2 Experiment 2

In the second experiment, we enable the specialized decoder layer and set it to polyfit based decoding as shown in Figure 4.13.

Model design:

- The model design is illustrated in Figure 4.13.
- This experiment enables the specialized decoder layer and sets it to the polynomial fitting decoder with degree 5. All other sections of the model are the same.

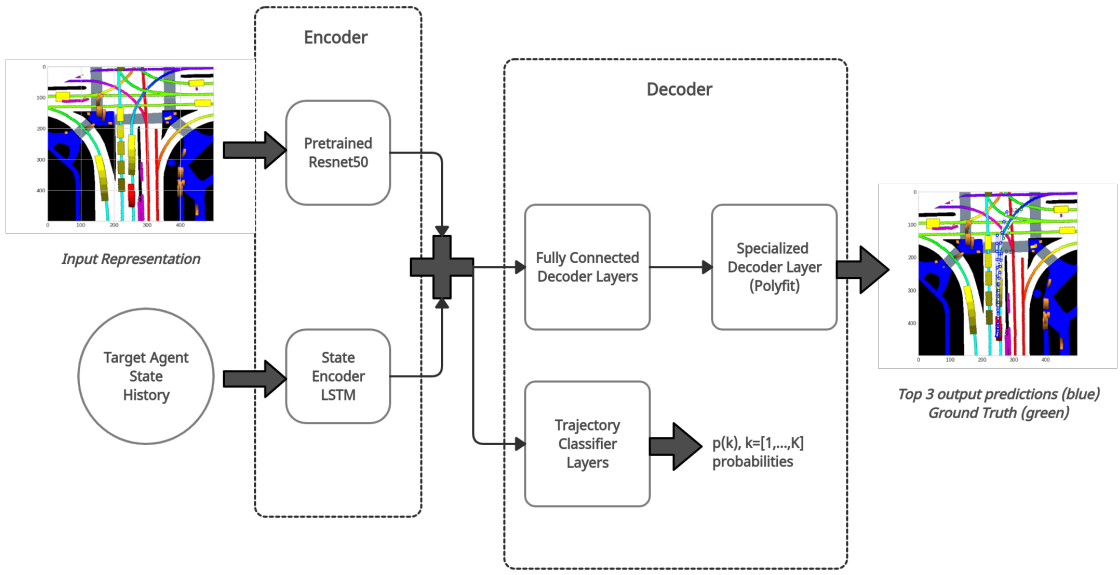

Figure 4.13: Exp-2 model setup

- With the specialized decoder enabled, this model also learns to predict history and learns a curve fitting parameterization over both history and future trajectories, which helps in improving performance.

Quantitative evaluation

Table 4.7 and 4.8 compare the quantitative performance of this model (`Exp-2-poly`) with the model from Experiment 1 having the basic decoder.

Model	$minADE_1$	$minADE_5$	$minADE_{10}$	$minFDE_1$	$minFDE_5$	$minADE_{10}$
Exp-1-basic	4.35	1.74	1.34	9.73	3.52	2.52
Exp-2-poly	4.49	1.77	1.44	9.93	3.54	2.69

Table 4.7: Exp-2-poly evaluation metrics-I

Model	$MissRate_{5,2}$	$MissRate_{10,2}$	Off-road Rate
Exp-1-basic	0.64	0.62	0.24
Exp-2-poly	0.69	0.68	0.30

Table 4.8: Exp-2-poly evaluation metrics-II

Qualitative evaluation:

In Figure 4.14, we look at the 4 scene evaluations for Exp-2.

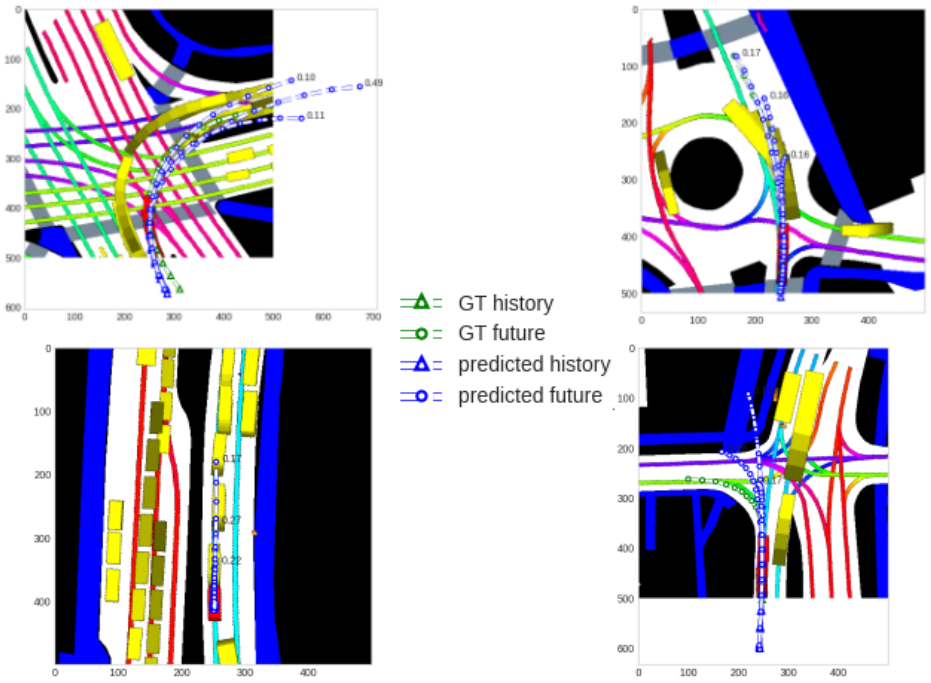

Figure 4.14: Qualitative evaluation of Exp-2

Observations:

- Regarding quantitative evaluation, the polynomial fitting decoder does not do better than the basic decoder, especially at the lower number of modes for metrics-I. This can be attributed to the curve constraints introduced by the fit and the covariance issue between the degree terms that was discussed in the methodology chapter.

- Polyfit decoder does not do well on metrics-II as well. This could be attributed again to the constraints in the curve fit that might force the trajectory to go off-road at longer horizon steps.

- Regarding qualitative evaluation, polyfit does better than basic decoder in Scene-1 and Scene-2. Like the basic decoder, it also detects the slow down in speed due to congestion in Scene-3. But it fails in Scene-4 as although it predicts a left and a straight maneuver, they are both off-road at longer horizons

4.4.3 Experiment 3

In the third experiment, we explore more specialized decoder layers such as Orthogonal Polyfit and Complex FFT as shown in Figure 4.15.

Model design:

- The model design is illustrated in Figure 4.15.

- In this experiment, more specialized decoders are tried out, such as the orthonormal Legendre decoder and the FFT complex domain decoder.

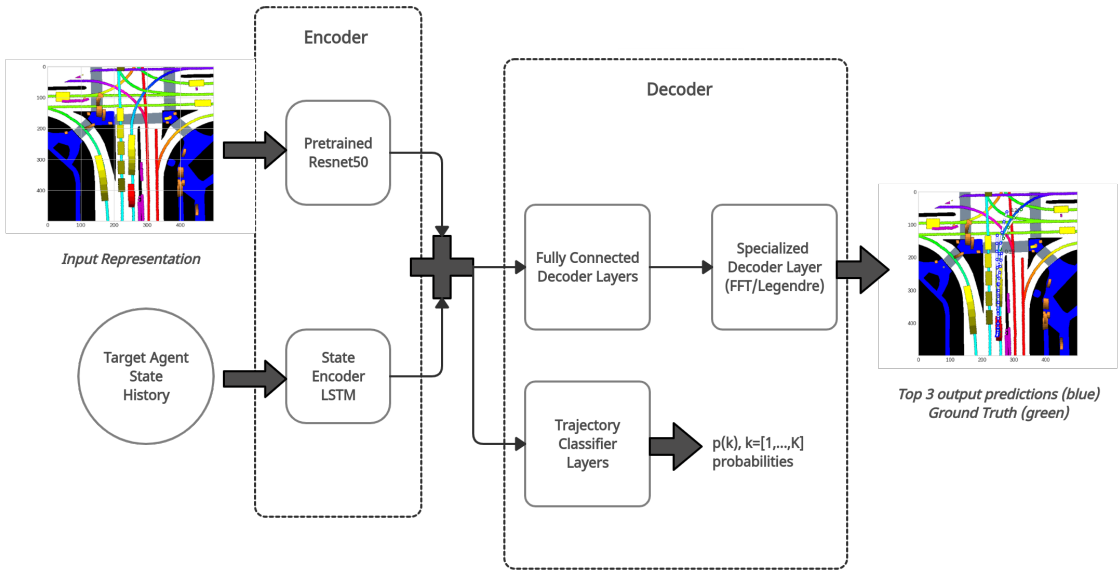

Figure 4.15: Exp-3 model setup

- The off-road loss term had an implementation constraint due to which it has not been included in the experiments and is a future scope of work. All other sections remain the same

Quantitative evaluation:

Table 4.9 and 4.10 compare the quantitative performance of this model with the two decoder designs that are Orthogonal polyfit (`Exp-3-ortho`) and Complex FFT (`Exp-3-fftc`) with the two models from Experiment 1 and Experiment 2.

Qualitative evaluation:

Figure 4.16 looks into the qualitative evaluation of the orthogonal decoder and Figure 4.17 displays the qualitative evaluation of the FFT decoder.

Model	$minADE_1$	$minADE_5$	$minADE_{10}$	$minFDE_1$	$minFDE_5$	$minADE_{10}$
Exp-1-basic	4.35	1.74	1.34	9.73	3.52	2.52
Exp-2-poly	4.49	1.77	1.44	9.93	3.54	2.69
Exp-3-ortho	4.21	1.70	1.33	9.35	3.41	2.48
Exp-3-fftc	4.33	1.76	1.39	9.79	3.60	2.61

Table 4.9: Exp-3 orthogonal and FFT-complex decoder evaluation metrics-I

Model	$MissRate_{5,2}$	$MissRate_{10,2}$	Off-road Rate
Exp-1-basic	0.64	0.62	0.24
Exp-2-poly	0.69	0.68	0.30
Exp-3-ortho	0.65	0.62	0.21
Exp-3-fftc	0.66	0.63	0.23

Table 4.10: Exp-3 orthogonal and FFT-complex decoder evaluation metrics-II

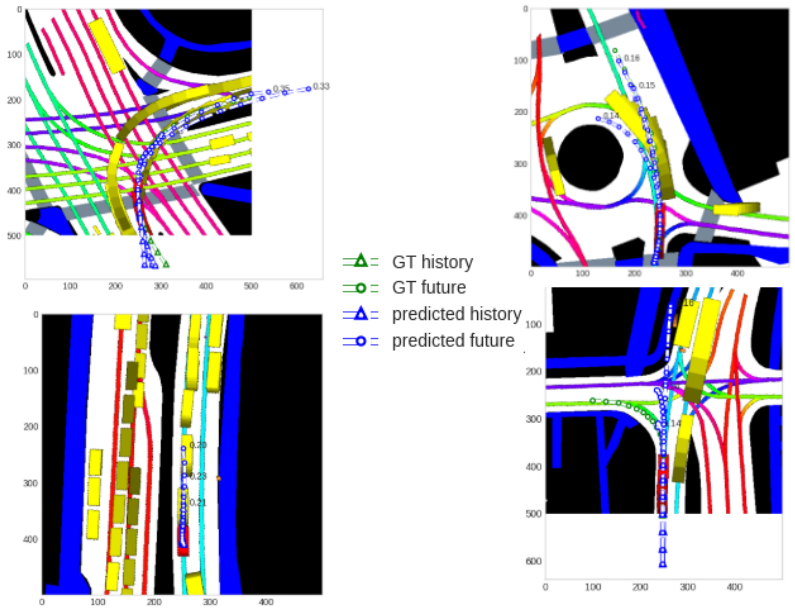

Figure 4.16: Qualitative evaluation of Exp-3 orthogonal decoder

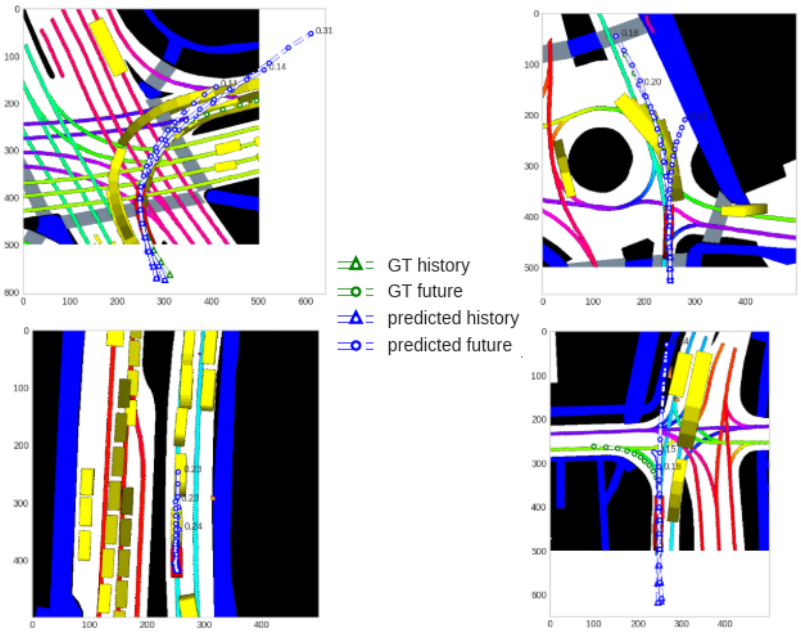

Figure 4.17: Qualitative evaluation of Exp-3 FFT decoder

Observations:

- Regarding quantitative, orthogonal decoder performs the best out of all earlier decoders in almost all metrics. FFT complex decoder lags behind orthogonal in metrics-I and metrics-II but beats the regular polynomial decoder. Although having a complex representation of the output frame should help the FFT decoder, the reasons for poorer performance on metrics-I could be over-fitting to the training data as it has high variance capabilities in curve fitting and better fine-tuning might give promising results. Although, FFT does fine on metrics-II.

- Regarding qualitative evaluation, the orthogonal decoder does very well on Scene-1,2,3 but partially well in Scene-4 as it does predict a left turn, but the predicted locations are well short of the ground truth.

- FFT decoder does very well on Scene-2,3 and fairly well on Scene-1. But it fails in Scene-4 with off-road prediction and failure to detect the left turn on time.

4.4.4 Experiment 4

From this experiment onwards, we use the Arch-II model which has improvements in the encoder block. Arch-II model uses the Spatio-temporal Social Encoder (STSE) to encode the map and agent history states. For this experiment, we only use the first stage of the interaction block. That is, we enable only the Conv2d-BatchNorm block and disable the attention block. Instead, to aggregate the grid features, we use average pooling and obtain the context vector. This is illustrated in Figure 4.18.

Model design:

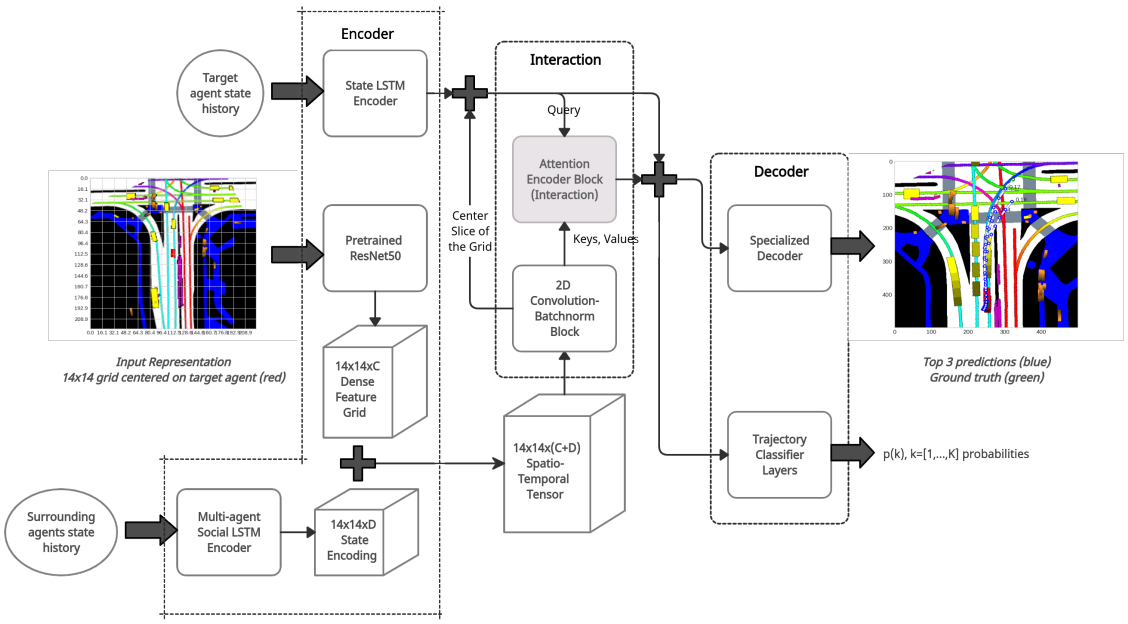

Figure 4.18: Exp-4 model setup

- The model design is illustrated in Figure 4.18 with the attention block disabled.
- This is a major upgrade in the model design as it uses the Arch-II architecture which is built on spatio-temporal encoding and interaction.
- It uses the social LSTM to encode all agent features and concatenates it to the spatial grid encoding of the map information.
- This model only has the primitive block of interaction which is the convolution-batchnorm block. It does not have any attention mechanisms. Instead, to aggregate the grid features, an average pooling operation is applied and concatenated with the target state encoding.
- The specialized decoders stay the same

Quantitative evaluation:

In the Tables 4.11 and 4.12, we compare the performance of this model having four different decoders which are basic (`Exp-4-basic`), polyfit (`Exp-4-poly`), orthogonal polyfit (`Exp-4-ortho`), and Complex FFT (`Exp-4-fftc`) against the better performing model from Experiment 3 (`Exp-3-ortho`) and the best performing state of the art model (`MHA-JAM`).

Model	$minADE_1$	$minADE_5$	$minADE_{10}$	$minFDE_1$	$minFDE_5$	$minADE_{10}$
Exp-3-ortho	4.21	1.70	1.33	9.35	3.41	2.48
MHA-JAM (sota)	3.69	1.81	1.24	8.57	3.72	2.21
Exp-4-basic	3.67	1.56	1.23	8.53	3.17	2.30
Exp-4-poly	3.45	1.52	1.25	7.98	3.10	2.29
Exp-4-ortho	3.48	1.50	1.16	8.02	3.05	2.11
Exp-4-fftc	3.44	1.53	1.16	7.93	3.17	2.16

Table 4.11: Exp-4 model with varying decoders evaluation metrics-I

Model	$MissRate_{5,2}$	$MissRate_{10,2}$	Off-road Rate
Exp-3-ortho	0.65	0.62	0.21
MHA-JAM (sota)	0.59	0.45	0.07
Exp-4-basic	0.63	0.59	0.18
Exp-4-poly	0.62	0.59	0.21
Exp-4-ortho	0.58	0.55	0.16
Exp-4-fftc	0.58	0.54	0.19

Table 4.12: Exp-4 model with varying decoders evaluation metrics-II

Qualitative evaluation:

In Figure 4.19, we look into the qualitative evaluation of the spatio-temporal social encoder with specialized decoders.

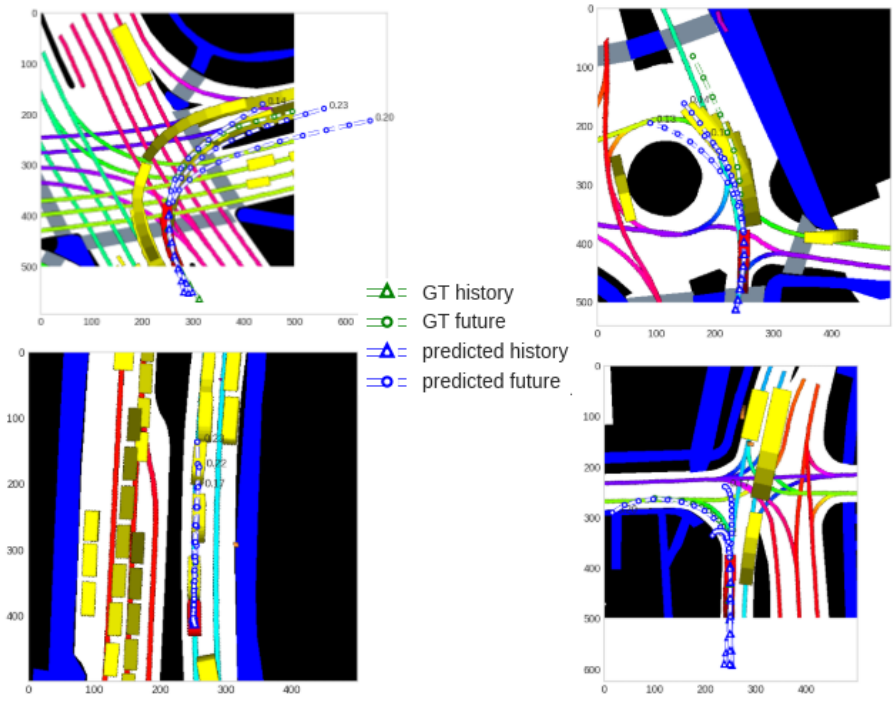

Figure 4.19: Qualitative evaluation of Exp-4 with orthogonal decoder

Observations:

- From the quantitative evaluation, it is clear that there is a significant improvement in performance across all metrics in metrics-I and a fair improvement in metrics-II with this new representation and encoder. Given these metrics, this model in fact performs better or at par with the state of the art models reported in Tables 4.3 and 4.4.

- All 4 decoders compliment the encoder and perform well. It is interesting to note that the basic decoder now is the worst performing out of the 4 decoders by a significant margin. This can be attributed to the feature rich input to the decoder which helps the specialized decoders perform curve fitting better. The FFT decoder, which performed poorly in the previous architecture, now performs at par with the orthogonal decoder.

- From the qualitative analysis, it can be seen that the model does very well in Scene-1 and Scene-4, which is a difficult scenario. It predicts the left turn and its position fairly accurately.

- Performance is less in Scene-2 where although it predicts a left and a straight possibility, it is lagging behind the ground truth and in Scene-3 where it overestimates the prediction as the attention module is disabled.

4.4.5 Experiment 5

In this experiment, we enable the attention block in the interaction module of Arch-II model. This allows for a weighted aggregation of the grid encoded features

where the weights are estimated by the attention mechanisms. The model is illustrated in Figure 4.20.

Model design:

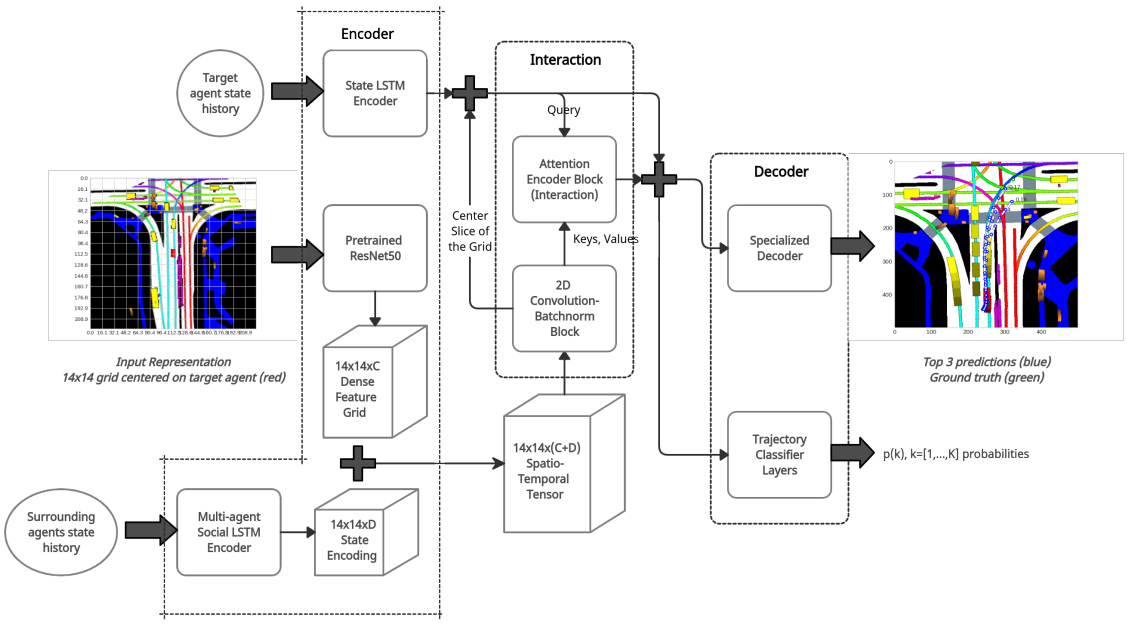

Figure 4.20: Exp-5 model setup

- This model is an extension to the previous one and adds an attention block in the interaction module as shown in Figure 4.20.
- It implements two separate attention mechanisms. One is the additive attention denoted as `Exp-5-aa` in the table and multi-head attention denoted as `Exp-5-mha`. These blocks learn the dependence of the surrounding agent behaviour on the target agent. The aggregation of the grid features is here done by

the attention block learning different attention weights for the spatio-temporal grid.

- The 10 context vectors are then concatenated with the target agent encoding and passed to decoder blocks which learn to decode the trajectories.

Quantitative evaluation:

In the Tables 4.13 and 4.14, we compare the performance of this model having two different attention mechanisms which are additive and multihead and two different decoders which are Orthogonal Polyfit and Complex FFT against the better performing models from Experiment 4 (`Exp-3-ortho`) and the best performing state of the art model (`MHA-JAM`).

Model	$minADE_1$	$minADE_5$	$minADE_{10}$	$minFDE_1$	$minFDE_5$	$minADE_{10}$
MHA-JAM (sota)	3.69	1.81	1.24	8.57	3.72	2.21
Exp-4-ortho	3.48	1.50	1.16	8.02	3.05	2.11
Exp-4-fftc	3.44	1.53	1.16	7.93	3.17	2.16
Exp-5-aa-ortho	3.56	1.56	1.15	8.30	3.19	2.09
Exp-5-mha-ortho	3.56	1.56	1.16	8.29	3.16	2.09
Exp-5-mha-fft	3.54	1.53	1.14	8.24	3.14	2.08

Table 4.13: Exp-5 attention model evaluation metrics-I

Model	$MissRate_{5,2}$	$MissRate_{10,2}$	Off-road Rate
MHA-JAM (sota)	0.59	0.45	0.07
Exp-4-ortho	0.58	0.55	0.16
Exp-4-fftc	0.58	0.54	0.19
Exp-5-aa-ortho	0.59	0.54	0.14
Exp-5-mha-ortho	0.60	0.55	0.14
Exp-5-mha-fft	0.57	0.53	0.16

Table 4.14: Exp-5 attention model evaluation metrics-II

Qualitative evaluation:

In Figure 4.21, we look into the qualitative evaluation on the 4 scenes and visualization of a couple of attention maps learnt during the process.

Observation:

- On quantitative evaluation, the performance is on par with the previous models on metrics-I except for the $k = 1$ metrics where the error is higher.
- There is a significant improvement in off-road loss with this model.
- On qualitative evaluation, the model does very well on Scene-2,3 and fairly well on Scene-1,4. Its Scene-1 predictions are a bit off-road and in Scene-4, although it has detected a left turn, it lags behind the ground truth.
- Looking at the visualization of the attention maps, we can see from the top attention map that the grids containing the lanes with agents have higher weights (brighter) than grids without any interesting agents. The bottom map is learning some other interaction behavior different from the previous one.

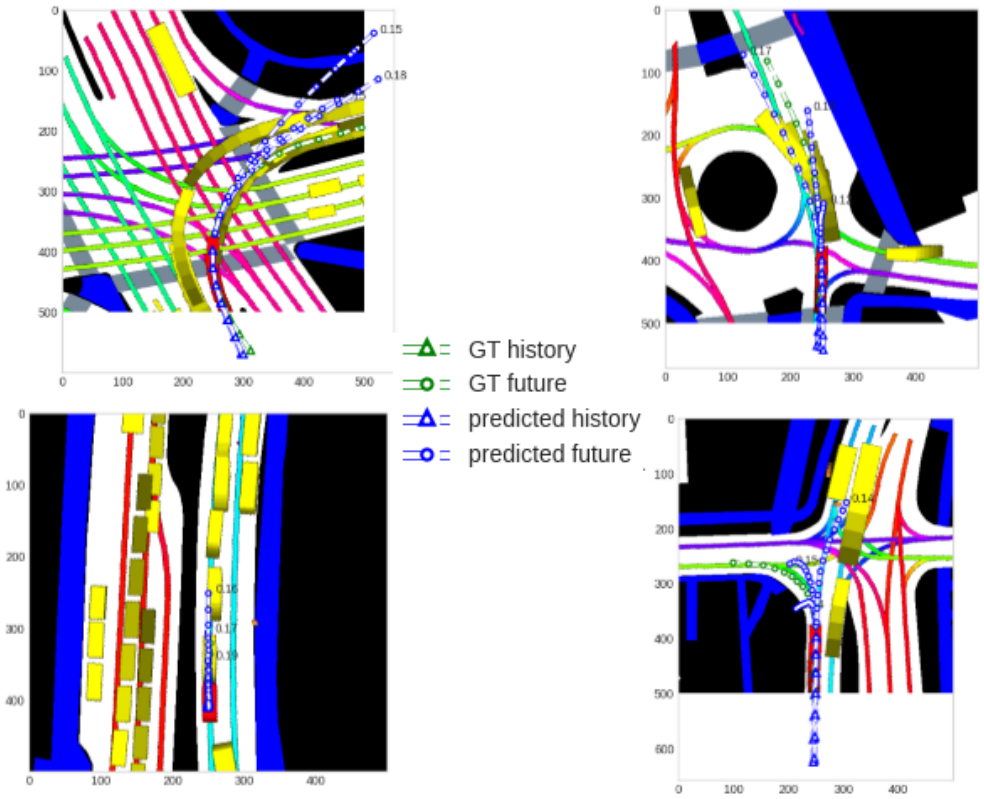

Figure 4.21: Qualitative evaluation of Exp-5 attention with orthogonal decoder

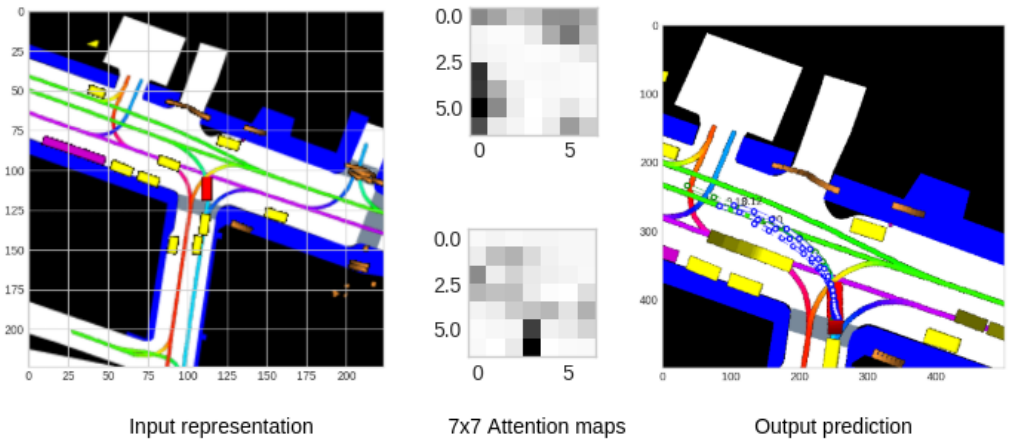

Figure 4.22: Visualization of two attention maps learnt by the interaction module

Figure 4.23 shows a consolidated result of $minADE$ and $MissRate_2$ metrics from all the above experiments plotted against the number of trajectory predictions. As seen from the graphs, multimodal predictions address the issue of uncertainty in estimation. And as seen, there are incremental improvements made through the experiment sequence.

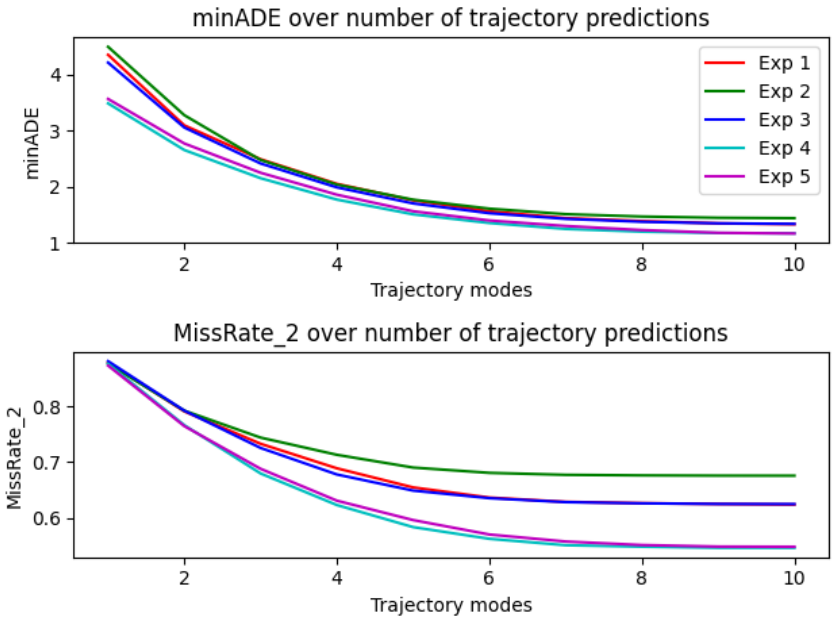

Figure 4.23: $minADE$ and $MissRate_2$ metrics over number of trajectories

4.4.6 Experiment 6

In the final experiment, we evaluate the test-time adaptation capabilities of Arch-I model with meta-learning. As shown in Figure 4.24, we start with a pretrained model from the previous experiments and add a residual decoder layer that needs to learn the corrections to be applied to the history and future trajectories. All original parameters of the pretrained model are frozen except for the classifier parameters in addition to the residual layers.

Model Design:

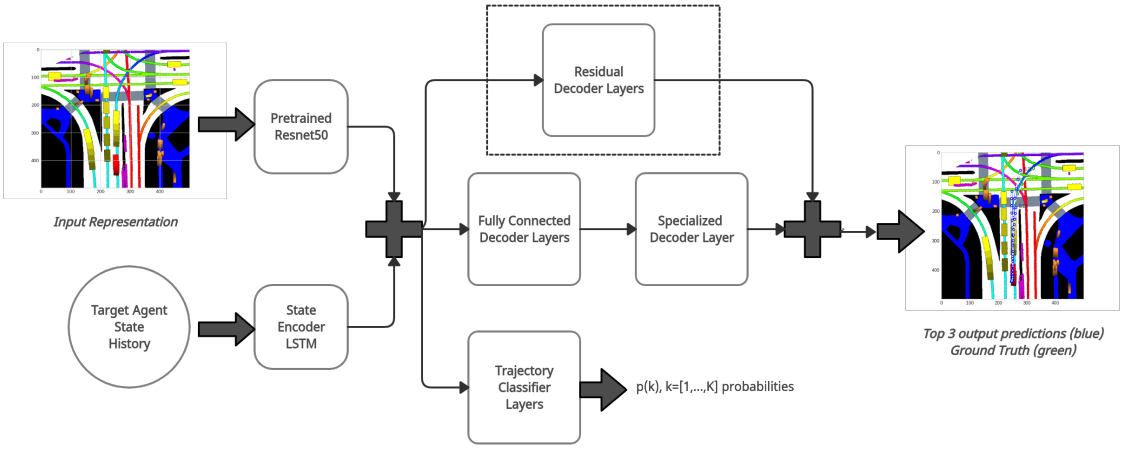

Figure 4.24: Exp-6 model setup

- This experiment starts with a pretrained model from previous experiments as shown in 4.24.
- In the meta-train stage, only the weights of the residual decoder layer and the classifier are learnt as meta-parameters using the proposed meta-learning

algorithm listed in the previous chapter. The model architecture can be seen in Figure 4.24.

- During testing, only the inner loop update is performed, which applies history correction and in turn improves the overall prediction.

Quantitative Evaluation:

The performance of the model after test time update worsens compared to the pretrained model performance. Although the test-time performance improves with the increase in the number of inner loop updates roughly in between five and ten, the values across the metrics do not reflect improvement over the pretrained model. Hence, this is currently an area of work to find out which aspect of the model causes a drop in performance. However, it is clear from the experiments that correcting the history predictions improves the future predictions suggested by the improvement in metrics with the increase in the number of inner-loop updates.

Qualitative Evaluation:

Figure 4.25 shows the qualitative evaluation for this meta-learning experiment.

Observations:

- As seen from the qualitative evaluation, when no test-time update is done (i.e 0 epochs), the performance is poor compared to the pretrained model the training started from. This is the primary area of concern to fix the meta-learn stage. Ideally, the model is expected to perform comparable to the pretrained model if not match the performance.

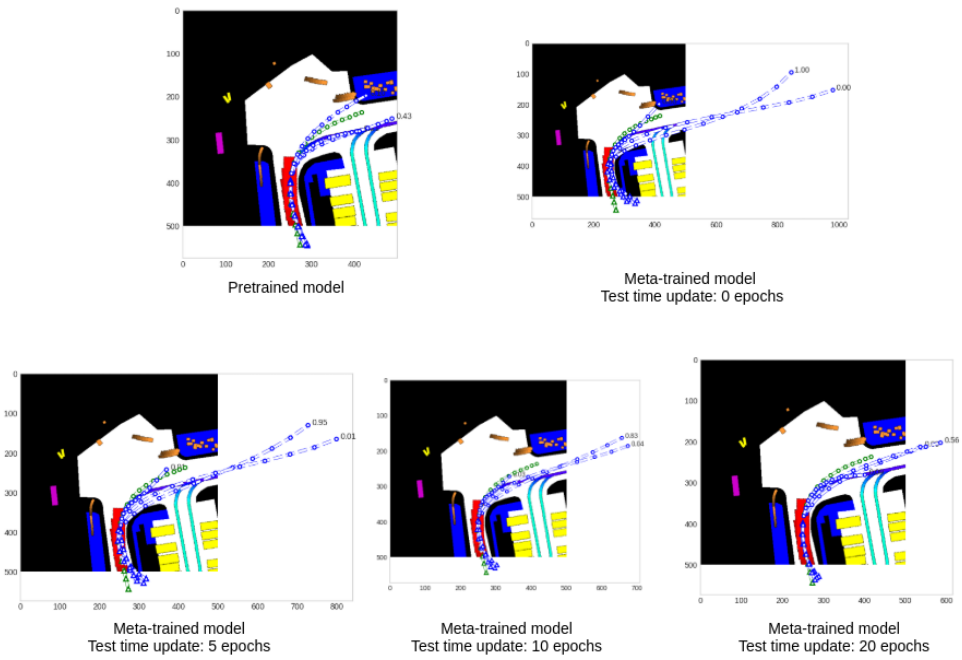

Figure 4.25: Qualitative evaluation of Arch-I meta-learnt model with various test-time correction epochs

- With the increasing number of test-time epochs, all history predictions are moving towards the ground-truth history as expected which in turn is improving the future trajectories as well as seen in the figure.

- With the increase in the number of test-time updates, the performance keeps improving and getting closer to the pretrained output at 20 epochs. As increasing test-time epochs increases the computation and execution time, test-time updates under 10 epochs are ideal. Hence, this issue needs to be addressed in future work.

4.5 Additional experiments

4.5.1 Ablation experiment: Disabling the target agent LSTM encoding

This ablation experiment aims to quantify the impact of the target state encoder LSTM in the Arch-I model. Here, Arch-I model with a basic decoder is used for the experiment. The same metrics used before are applied to quantify the performance difference.

Quantitative evaluation:

Model	$minADE_1$	$minADE_5$	$minADE_{10}$	$minFDE_1$	$minFDE_5$	$minADE_{10}$
Without EncLSTM	6.19	2.33	1.66	12.68	4.37	2.81
With EncLSTM	4.35	1.74	1.34	9.73	3.52	2.52

Table 4.15: Impact of encLSTM on metrics-I

Model	$MissRate_{5,2}$	$MissRate_{10,2}$	Off-road Rate
Without EncLSTM	0.80	0.77	0.19
With EncLSTM	0.64	0.62	0.24

Table 4.16: Impact of encLSTM on metrics-II

Qualitative evaluation:

Figure 4.26 shows the qualitative evaluation comparing the model performance with and without encoder LSTM.

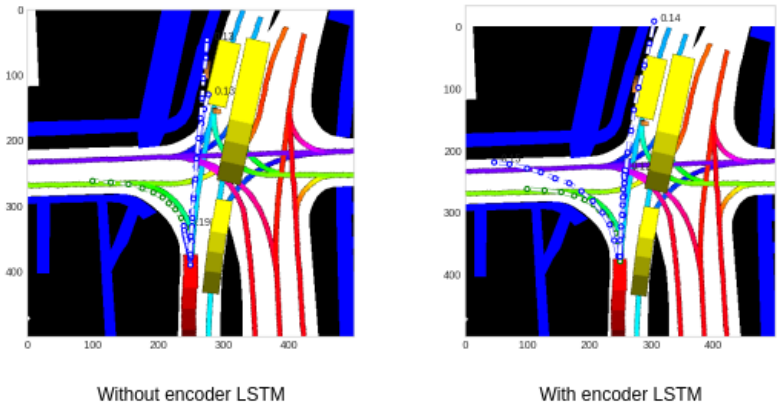

Figure 4.26: Qualitative evaluation for comparing model performance with and without encoder LSTM

Observations:

- The quantitative evaluation suggests that the encoder LSTM is impactful by a significant margin in predicting the future states of the target agent. This evaluation falls in line with the model performance expectations as capturing the temporal history information of the target agent gives the decoder enough prior information to make more accurate predictions.

- Figure 4.26 visually confirms the performance difference at an intersection scene. Without the encoder LSTM, although the model predicts a left turn, it heavily underestimates the states (velocity, acceleration, yaw rate) of the target agent. Whereas in the case with the encoder LSTM, it does better in predicting the left turn and the states as it has the prior temporal context of the target agent.

4.5.2 Special case evaluations

In Figure 4.27, one of the best performing models `Exp-5-mha-ortho` from the experiment evaluation, is used to evaluate four selected special case scenarios to test the robustness. As seen in the figure, the model estimates the future predictions well.

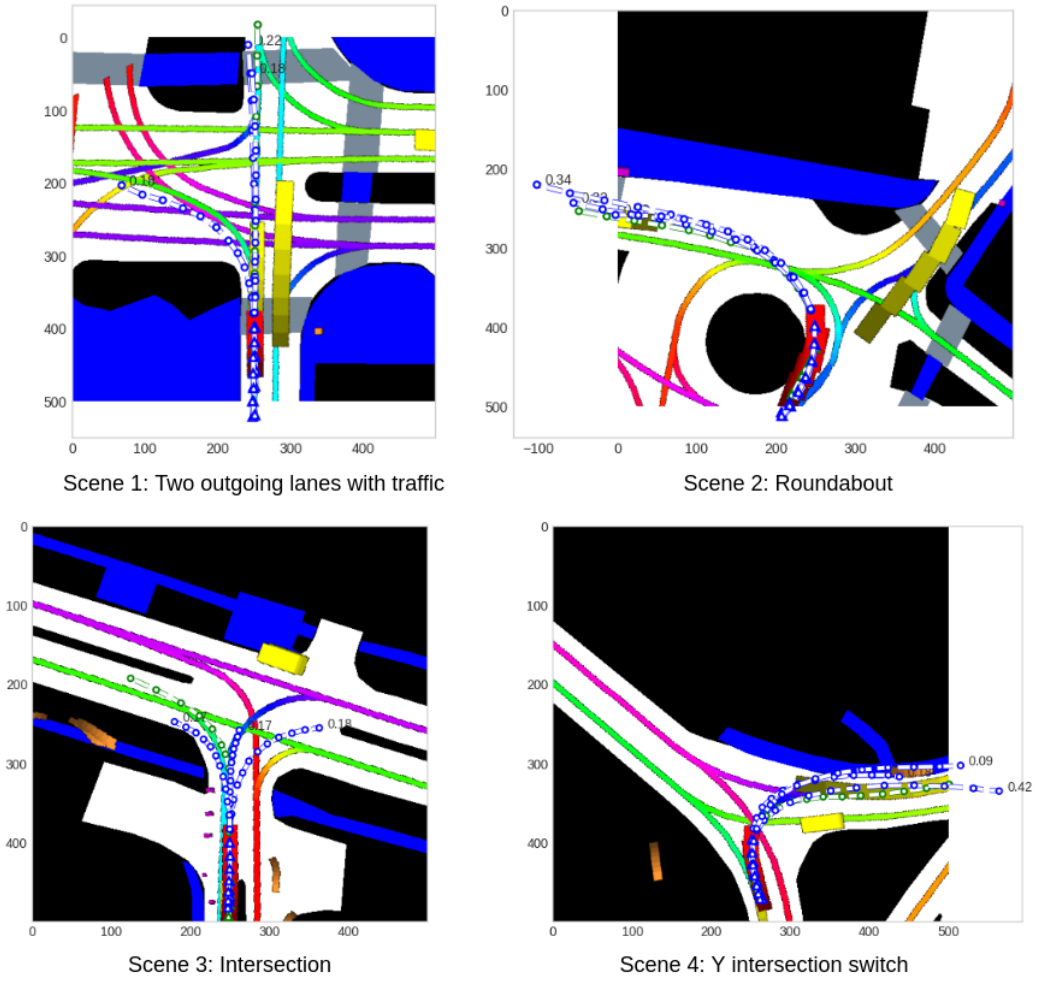

Figure 4.27: Qualitative evaluation of `Exp-5-mha-ortho` for 4 selected scenarios

Chapter 5: Conclusion

In this final chapter, we first discuss the future scope of work for the proposed models highlighting a few improvement areas and then provide concluding remarks on the status of the self-driving pipeline, the Motion Forecasting problem-solution and the role of this thesis in addressing the problem in hand.

5.1 Future Scope

In the previous chapter, all the experiments conducted to verify the methodology proposed for motion prediction were discussed with respective quantitative and qualitative evaluations. In this section, we list some of the future improvements over the methodology and additional experiments that have the scope to further solidify the model and improve benchmarks. Below is a list of such future scope of work:

- Addressing the poor performance of the meta-learning model during smaller test-time updates (i.e $\leq$ 10 epochs).
- Implementing and improving the off-road loss by adding a lane-attention module that learns to follow the current lane and also conduct lane changes respecting the restrictions imposed by the interaction space.

- Using visual-transformer based stacked blocks to further improve the interaction block performance.

- Evaluating the classifier performance to check for any imbalance in label predictions. This can give an insight to improve the classifier to include all possible maneuvers for a given scene.

- Learning a latent space from the encoder output to generate different possible trajectory states inspired by the variational autoencoder design. This is particularly helpful to handle any new scenarios encountered during test time.

- Run the benchmark models on more number of motion prediction datasets.

5.2 Concluding Remarks

Autonomous driving is a promising technology at the brink of revolutionizing transport, but the self-driving stack is not yet evolved enough to achieve Level 5 driving on our streets. Primarily, the problem of vision, environment interaction, and decision making needs to be solved for real-world scenarios. This thesis tackled one such crucial component in the pipeline which is motion forecasting. Given that future forecasting is a complex task with uncertainty, having an accurate and adaptive forecasting model can significantly improve the self-driving pipeline especially in efficiently and safely planning the trajectories to take towards the destination.

The thesis started off with the introduction of the self-driving pipeline, followed by a literature review for popular approaches and then delving into the methodology of two proposed architectures and finishing off by discussing the metric evaluation and visualization of the model performance on the nuScenes motion prediction dataset.

From the conducted experiments, it is clear that end-to-end deep neural networks outperform the traditional motion prediction models. Although the Arch-I model follows a simple design approach, it seems to demonstrate the previous claim. Additionally, given the multimodal nature of the future predictions especially at intersections, it is beneficial to have a multimodal trajectory decoder which can include all possible maneuvers for a given scene. And, it is also important to have a strong and robust encoder as feature rich encoder representations help improve the performance of the following blocks like interaction and decoder as seen from the performance difference between Arch-I and Arch-II architectures. Designing a spatio-temporal encoder in the proposed Arch-II models allowed to capture the spatial and temporal information from the scene and the agents and better equipped the attention blocks to learn intricate interactions between the agents and the scene. We also demonstrated a simpler decoder design with the linear basis representation which significantly reduces the number of parameters to be learnt. In fact, the number of parameters are independent of the length of the prediction horizon. Given that vehicles are non-holonomic agents, having this kind of decoder representation also allows the model to learn smoother trajectories due to the nature of curve fitting compared to directly predicting the trajectory locations. Lastly, we proposed an algorithm to perform test-time adaptation for new road scenarios using meta-learning. This provides the model the ability to be more robust in predictions to the changing environment and agents.

The self-driving pipeline is improving at a stellar pace and this thesis was an attempt to contribute to solving a challenging problem in the pipeline of predicting the future states of agents in a given scene. The experimental results provide a promising path forward to extend the proposed architectures to more complex scenarios with

a few ideas listed in the future scope section and to implement the architectures on a real-time embedded system. With this, the concluding remark of the thesis is that in order to accelerate the progress towards Level 5 driving, there needs to be significant research interests in developing more robust AD datasets and focusing on challenging and not completely solved tasks such as motion prediction and test-time domain adaptation.

Bibliography

[1] Asirt: Annual global road crash statistics. https://www.asirt.org/safe-travel/road-safety-facts.

[2] Fatality facts 2019: State by state. https://www.iihs.org/topics/fatality-statistics/detail/state-by-state.

[3] Global traffic fatality rates by thecityfix. https://thecityfix.com.

[4] Iii: Facts + statistics: Highway safety. https://www.iii.org/fact-statistic/facts-statistics-highway-safety.

[5] Inside story-look, no hands. https://www.economist.com/technology-quarterly/2012/08/30/look-no-hands.

[6] Kalman filter with constant velocity model. https://balzer82.github.io/Kalman/.

[7] Legendre polynomials. https://en.wikipedia.org/wiki/Legendre_polynomials#/media/File:Legendrepolynomials6.svg.

[8] Rnn, talking about gated recurrent unit. https://technopremium.com/blog/rnn-talking-about-gated-recurrent-unit/.

[9] Stanford cs230 slides. https://stanford.edu/~shervine/teaching/cs-230/. Accessed: 2021-05-02.

[10] Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[11] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving, 2020.

[12] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction, 2019.

[13] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[14] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks, 2019.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

[17] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[18] Matthias Hartwig. "Self-driving and cooperative cars".

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Long Chen, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset, 2020.

[22] SAE International. "SAE International Releases Updated Visual Chart for Its "Levels of Driving Automation" Standard for Self-Driving Vehicles". December 11. 2018.

[23] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *CoRR*, abs/1704.05519, 2017.

[24] Kaouther Messaoud, Nachiket Deo, Mohan M. Trivedi, and Fawzi Nashashibi. Trajectory prediction for autonomous driving based on multi-head attention with joint agent-map representation, 2020.

[25] Darrin Qualman. "Happy motoring: Global automobile production 1900 to 2016". June 13. 2017.

[26] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[27] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[29] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data, 2021.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[31] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention, 2016.