The University of Hull
Faculty of Science and Engineering
School of Computer Science

Short-term motion prediction of autonomous vehicles in complex environments: A Deep Learning approach

Albert Dulian

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science of the University of Hull,
April 2, 2024

To my beloved great-grandmother
Waleria Bartczak

List of Publications

Conference Papers

- **Dulian, A.** and Murray, J.C., 2021, May. Exploiting latent representation of sparse semantic layers for improved short-term motion prediction with Capsule Networks. In 2021 IEEE International Conference on Robotics and Automation (ICRA) (pp. 8537-8543). IEEE.

Preprints

- **Dulian, A.** and Murray, J.C., 2021. Multi-modal anticipation of stochastic trajectories in a dynamic environment with Conditional Variational Autoencoders. arXiv preprint arXiv:2103.03912.

Acknowledgments

This ride has been a real rollercoaster. I consider the last couple of years to be extremely transformative and I truly believe that going through this period in my life has been critical, albeit extremely challenging. Prior to embarking on this PhD journey I felt unsure about the future. Now however, I have no doubt that I ended up on the right track. Research, curiosity, novelty and desire to understand complex concepts gives me an enormous joy and fulfilment. I will be forever grateful that this opportunity was presented to me and I hope I will be able to successfully continue on this path.

That being said, I would like to first of all, express my sincere gratitude to my supervisor, Professor John C. Murray for his patience, encouragement and firm guidance throughout the last couple of years. Without John's invaluable support completing this thesis would not be possible. Moreover, I would like to thank Dr. Yongqiang Cheng who provided an immense amount of support in my final months of PhD. Furthermore, I would also like to extend my deepest appreciation and gratitude to Dr. Nina Dethlefs for her meticulous and insightful feedback, which has greatly contributed not only to the refinement and improvement of my work but also to my growth as a researcher. Next, I would like to thank my lab friends Dr. Annika Schoene and Ashley Williamson for all the laughter and stimulating research conversations, which have been a constant source of inspiration.

I am particularly thankful to be constantly surrounded by amazing people. To my loving parents, Justyna and Sylwester Dulian, thank you for believing in me and my life choices. You allowed me to pursue my dreams whilst offering a continuous support and an unwavering faith in my abilities, without it, this significant milestone would not have been possible. To my dear sister Zuzanna Dulian and her amazing fiancé Wiktor Marszalek who, throughout those stressful years, were always there for me whenever I needed a reminder that there was a world beyond academia. Thank you for being there for me, always offering solace and a great sense of normalcy that kept me well grounded. To my wonderful brother Sebastian Sodolski and his fiancée

Oliwia Gasior. It is difficult to express in words what it means to have people like you in my life. There is no one else in this world with whom I have shared such heartfelt laughter, and I sincerely hope that in the future, we will have the opportunity to spend even more time together. Next, I would like to extend my deepest appreciation to my best friends. To Filip Kiedrowski, Robert Popowski, Sebastian Dabrowski, Szymon Bukowski and Adam Gasowki. I will be forever grateful that our paths crossed, as your presence has had a profound influence on my life. I sincerely wish for everyone to be surrounded by extraordinary individuals like you. To Kuba Maruszczyk and his fiancée Dr. Sylwia Macinska, I wish that one day both of you will realize the magnitude of positive impact you have had on my life. It is evident to me that without knowing you, I would never have achieved the remarkable milestones I have, nor would I have dared to envision the extraordinary path that unfolded before me. Lastly, to Przemyslaw Doruch, my life long best friend, thank you for it is you who instilled in me a passion for technology.

Finally, I want to express my dearest gratitude to my beloved fiancée Agata Jakubowska. You have been by my side since the very beginning of this endeavor, accompanying me every step of the way. During the last couple of years you have been the most profound source of support. It is you who have given me the greatest sense of tranquility, composure and immense happiness. You have endured me during the most demanding and stressful years of my life, I hope I will be able to repay you. Today, I truly believe that with you by my side I can accomplish anything because "No man succeeds without a good woman behind him". I love you my dear.

Abstract

Complex environments manifest a high level of complexity and it is of critical importance that the safety systems embedded within autonomous vehicles (AVs) are able to accurately anticipate short-term future motion of agents in close proximity. This problem can be further understood as generating a sequence of coordinates describing the plausible future motion of the tracked agent. Number of recently proposed techniques that present satisfactory performance exploit the learning capabilities of novel deep learning (DL) architectures to tackle the discussed task. Nonetheless, there still exists a vast number of challenging issues that must be resolved to further advance capabilities of motion prediction models.

This thesis explores novel deep learning techniques within the area of short-term motion prediction of on-road participants, specifically other vehicles from a points of autonomous vehicles. First and foremost, various approaches in the literature demonstrate significant benefits of using a rasterised top-down image of the road to encode the context of tracked vehicle's surroundings which generally encapsulates a large, global portion of the environment. This work on the other hand explores a use of local regions of the rasterised map to more explicitly focus on the encoding of the tracked vehicle's state. The proposed technique demonstrates plausible results against several baseline models and in addition outperforms the same model that instead uses global maps. Next, the typical method for extracting features from rasterised maps involves employing one of the popular vision models (e.g. ResNet-50) that has been previously pre-trained on a distinct task such as image classification. Recently however, it has been demonstrated that this approach can be sub-optimal for tasks that strongly rely on precise localisation of features and it can be more advantageous to train the model from scratch directly on the task at hand. In contrast, the subsequent part of this thesis investigates an alternative method for processing and encoding of spatial data based on the capsule networks in order to eradicate several issues that standard vision models exhibit. Through several experiments it is established that the novel capsule based motion predictor that is trained from scratch is able to achieve competitive results against numerous popular vision

models. Finally, the proposed model is further extended with the use of generative framework to account for the fact that the space of possible movements of the tracked vehicle is not strictly limited to single trajectory. More specifically, to account for the multi-modality of the problem a conditional variational auto-encoder (CVAE) is employed which enables to sample an arbitrary amount of diverse trajectories. The final model is examined against methods from literature on a publicly available dataset and as presented it significantly outperforms other models whilst drastically reducing the number of trainable parameters.

Contents

List of Tables

List of Figures

Acronyms

ABS Anti-lock brake system

ACC Adaptive cruise control

AHC Agglomerative hierarchical clustering

ADAS Advanced driver assistance systems

AE Auto-encoder

AV Autonomous vehicle

ADE Average displacement error

BI Bayesian inference

BF Bayesian filter

BN Bayesian network

CA Constant acceleration

CAN Controller area network

CCA Constant curvature and acceleration

CL-RRT Closed loop rapidly-exploring random tree

CNN Convolutional neural network

CSV Constant steering angle and velocity

CTRA Constant turn rate and acceleration

CTRV Constant turn rate and velocity

CV Constant velocity

CVAE Conditional variational auto-encoder

CYRA Constant yaw rate and acceleration

DAE Denoising auto-encoder

DARPA Defense Advanced Research Projects Agency

DL Deep learning

DGPS Differential global positioning system

DP Dirichlet process

ELU - Exponential linear unit

EKF Extended Kalman filter

FDE Final displacement error

GAN Generative adversarial network

GD Gradient descent

GHMM Growing Hidden Markov model

GMM Gaussian mixture model

GNG Growing neural gas

GNN Graph neural network

GP Gaussian process

GPS Global positioning system

GRU Gated recurrent unit

HMM Hidden Markov model

ICS Inevitable collision state

IOC Inverse optimal control

KF Kalman filter

LSTM Long short-term memory

MAE Mean absolute error

MBGD Mini-batch gradient descent

MCP McCulloch-Pitts neuron

ML Machine learning

MOG Mixture of Gaussians

MLP Multilayer perceptron

MRM Manoeuvre recognition module

MSE Mean squared error

PCA Principal component analysis

PFSM Probabilistic finite-state machine

PPM Partial motion planning

ReLU Rectified linear unit

RNN Recurrent neural network

RRT Rapidly-exploring random tree

RVM Relevance vector machine

SBL Sparse Bayesian learning

SGD Stochastic gradient descent

SVM Support vector machine

UKF Unscented Kalman filter

VAE Variational auto-encoder

VGMM Variational Gaussian mixture model

Chapter 1

Introduction

1.1 On a Road to Vehicle Automation

Throughout the last couple of decades the domain of transportation has matured to the point where it became an inseparable part of today's society. Automobiles became a dominant form of transportation around the world and it has been estimated that in merely 20 years, i.e. between 1996 and 2016 the number of motor vehicles in operation has almost doubled in quantity from 670 million to over 1.3 billion (Petit 2017).

Furthermore, as safety became a crucial factor over the past years, automotive companies introduced various technological advancements to increase traffic safety and assist motorists with the task of driving. A great portion of those advancements has been adopted in numerous real-time embedded systems known as ADASs (Advanced Driver Assistance Systems) that offer a range of support, from advanced visual warnings to taking over lateral and longitudinal control of the vehicle. ABS (Anti-lock Brake System) and ACC (Adaptive Cruise Control) are just two examples of well known commercial ADASs. ABS is designed to enable the driver to

retain control of the vehicle by preventing the wheels from locking during braking which can cause the vehicle to skid, thereby causing loss of control (Broughton & Baughan 2002). On the other hand, ACC is responsible for sustaining a safe distance with respect to the vehicle in front, whilst maintaining a pre-set or optimal velocity, therefore the system can be further classified as longitudinal control and avoidance system. For instance, if sensors detect that the velocity of vehicle in front has decreased and therefore the minimum safety distance has been exceeded, ACC will automatically adjust the velocity of its vehicle to ensure that enough safe space is maintained (Winner et al. 1996).

Nonetheless, despite all amenities offered by the advance technology of today's automobiles, there still exist a vast number of drawbacks as road transports are considered to be one of the major causes of fatalities. According to global road crash statistics published by ASIRT ('Association for Safe International Road Travel') over 3,000 people die daily in road accidents, with the annual figure being as high as 1.25million (Organization et al. 2023), leaving further 20-50million people injured or disabled. Likewise, data from 2015 presents that in United States alone traffic accidents were classified as number four cause of death accounting for more than 5% of total casualties (for Health Statistics 2017). Moreover, the primary factor that contributed and caused an estimate of 94% of those accidents was in fact a human error (Singh 2015).

Additionally, although ADAS systems are designed with the intention of improving traffic safety, reducing driver errors, and increasing efficiency, it is often a case that those system present a host of potential hazards and problems. For example, according to Lee et al. (2004), creating a collision warning system is not a trivial task as its effectiveness is highly dependable on how well the system is designed with respect to a driver's capabilities and preferences. As such, an inappropriately designed warning system might lead to a situation where the driver will ignore warning signals,

and therefore fail to act when necessary (Bliss & Acton 2003). Furthermore, Bishop (2005) points out the importance of designing a system where the trust between the driver and technology is well-balanced to avoid (in case of "over-trust") a creation of dependence which could potentially result in significantly reduced vigilance. Lastly, Rudin-Brown & Parker (2004) presented the negative effect of ACC on driver's behavioral adaptation where ACC-participants' response time to hazard detection as well as lane position variability has increased considerably.

Based on results of past experiments discussed in previous paragraphs it is reasonable to speculate that achieving significant improvements in terms of on-road safety cannot be accomplished without restricting or even completely eliminating the need for a human driver. The challenge of eradicating human error has caused a significant research interest increase in the field of autonomous transportation not only from academic research institutions but also from several industry developers such as Tesla, Waymo, Volvo and many more. In this dissertation the term 'autonomous vehicle' (AV) will be further used to refer to vehicles that exhibit intelligent capabilities such as navigating and sensing environments with little to no human input.

The development of AV technology has seen its initial advancements only in the last decade, nevertheless, the work in this area has been ongoing for over three decades (Pomerleau 1989, Dickmanns et al. 1994, Pomerleau & Jochem 1996, Broggi et al. 1999). A pioneering piece of work can be attributed to the research project carried out in the 1980's at Carnegie Mellon University where Dean Pomerleau trained ALVINN (Autonomous Land Vehicle in Neural Network), a single hidden-layered neural network (Goodfellow et al. 2016) with data gathered from a front facing camera and laser range to output direction units which represent road curvature that the test vehicle should track in order to effectively follow the road (Pomerleau 1989).

Despite the fact that ALVINN unveiled various potential possibilities for the future

of autonomous transportation, the field was greatly restricted in its capabilities for further advancements primarily due to the hardware and software limitations of the time. The next big milestone came through a series of competitions (Grand Challenge) sponsored by DARPA (Defense Advanced Research Project Agency) between 2004-2007 that aimed to accelerate research and development of AVs. The first competition organised in 2004 has not seen a single vehicle finishing the 150 mile route, yet, it can still be considered as a significant success due to the impelled interested which resulted with numerous teams returning in the following year. The second edition of the Grand Challenge, held in 2005 involved a 130 mile route where vehicles were challenged to pass through a dessert terrain of various complexity, five out of 23 teams successfully finished the race with the winning entry 'Stanley' completing the track in 6h and 54min (Thrun et al. 2006). The success of the 2005 competition revealed a great number of problems, one of the most important being static race environment and inability of vehicles to perceive and interact with moving traffic. The third competition commonly known as 'DARPA Urban Challenge' involved 11 selected teams and focused mainly on addressing issues from previous years. The track required vehicles to navigate through roughly 60 miles of abandoned urban environment, which included single and multi lane roads, traffic circles, intersections, open areas, parking lots etc, whilst interacting with other moving vehicles and obeying California's traffic laws (Urmson et al. 2008). Yet again, the 2007 competition exposed numerous issues within AV technology, nonetheless, it is crucial to emphasize its various positive outcomes such as breakthroughs in software (Ferguson et al. 2008) and laser sensors (*Velodyne LiDAR* n.d.) without which the field of autonomous transportation would not evolve as rapidly as it has.

It's been over a decade since the DARPA Urban challenge and the technology deployed within today's AVs has developed dramatically. SAE International has categorized autonomy of vehicles into six levels (Committee et al. 2018) ranging from level 0 i.e. no driving automation to level 5 which defines full automation where no

human attention is required and the vehicle is capable of performing driving task in all conditions (see Figure: 1.1). As previously mentioned various car manufacturing companies are involved in the creation and development of AVs with an ultimate goal of providing technology capable of level 5 automation. At the moment, Tesla is considered as the most publicly used AV with their autopilot offering level 3 automation and providing functionality such as: navigating in complex environments (Autosteer+), summoning the vehicle in parking lots (Smart Summon), front and side collision warning, automatic adjustment of high/low beams and many more. In contrast, Waymo is already offering a level 4 automation in their vehicles, however, these are only available in Phoenix, AZ and are driven by a limited number of selected test riders (*Waymo Technology* n.d.). As of present, there are no companies offering full autonomous capabilities (level 5), and many experts believe that it may still take decades to create an intelligent autonomy capable of covering 99% of conditions (Litman 2017).

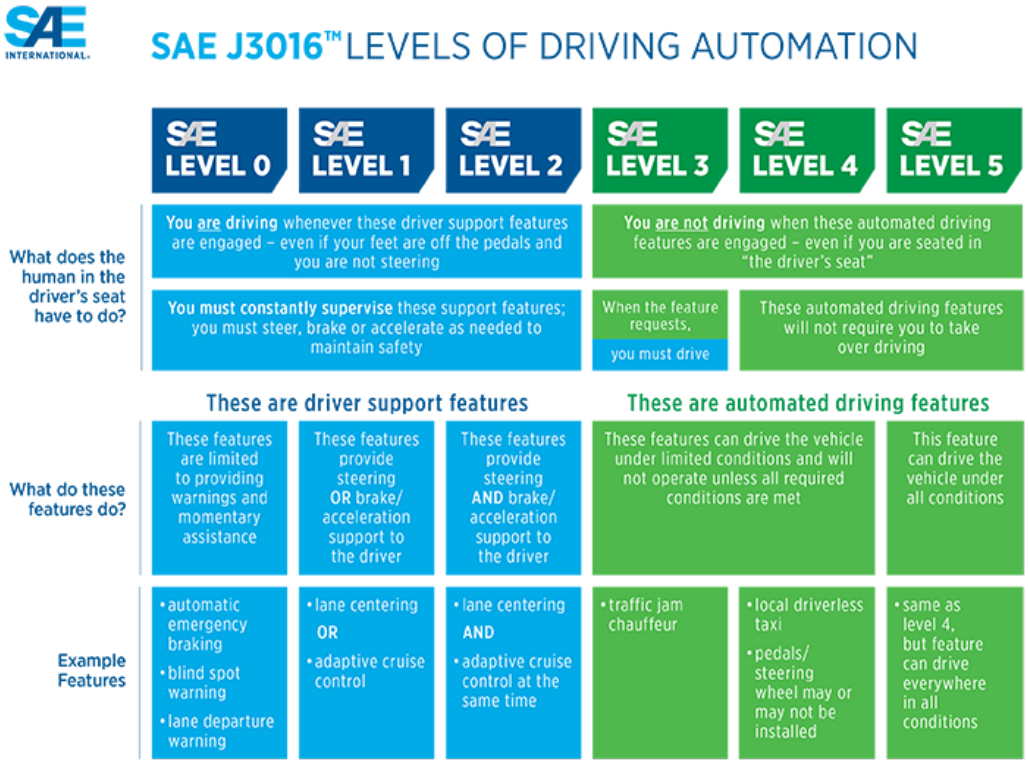

Figure 1.1: Summary of the driving automation levels as defined by SAE International (Committee et al. 2018).

1.2 Intelligently Driven Vehicles

With the need of creating intelligent autonomy comes the requirement of developing and equipping vehicles with artificially intelligent systems capable of performing various driving tasks such as path planning and environmental perception in a rational and reasonable manner. The domain of artificial intelligence (AI), more precisely its sub-domain deep learning (Goodfellow et al. 2016) has advanced tremendously in the last decade yielding a variety of novel techniques that have been effectively applied in numerous fields. In computer vision for instance, DL has been widely used for example to classify a given image into a particular category (Krizhevsky et al. 2012), reconstruct corrupted data (Liu et al. 2018), apply one type of image style to another (Gatys et al. 2016), and generate completely new images from either noise (Goodfellow et al. 2014) or existing data (Zhu et al. 2017) to name few. Undoubtedly, a considerable achievement accomplished with the use of DL, specifically deep reinforcement learning (DRL) (Sutton et al. 1998, Mnih et al. 2015) can be attributed to AlphaGO (Silver et al. 2016), a system that in 2016 beat former GO champion Lee Sedol in five-game match, a game that many considered a grand challenge for AI (McCarthy 1997, Mechner 1998).

A great number of deep learning based approaches have been deployed and used within AVs for many years, which is most notable in vehicles produced by Tesla and Waymo (Tesla Autonomy Day). For example, as previously indicated a key component of any AV is to meaningfully perceive its surrounding in order to detect and recognize objects. This is often achieved with use of object detection algorithms. A number of DL methods have been proposed for the mentioned task over the past decade (Sermanet et al. 2013, Ren et al. 2015, Redmon et al. 2016) allowing these techniques to not only categorise multiple objects into a specific class but also obtain their spatial location within an input image (see Figure: 1.2). An interesting use of DL have been demonstrated in the work developed by Bojarski et al. (2016)

where a simple idea of mapping image pixels from a front-facing camera into a steering command allowed to approximate and predict steering angle for the current state. Unexpectedly, the proposed model managed to not only anticipate steering angle, but also learned to recognise lanes and follow the road without the aid of any additional modules e.g. object detection.

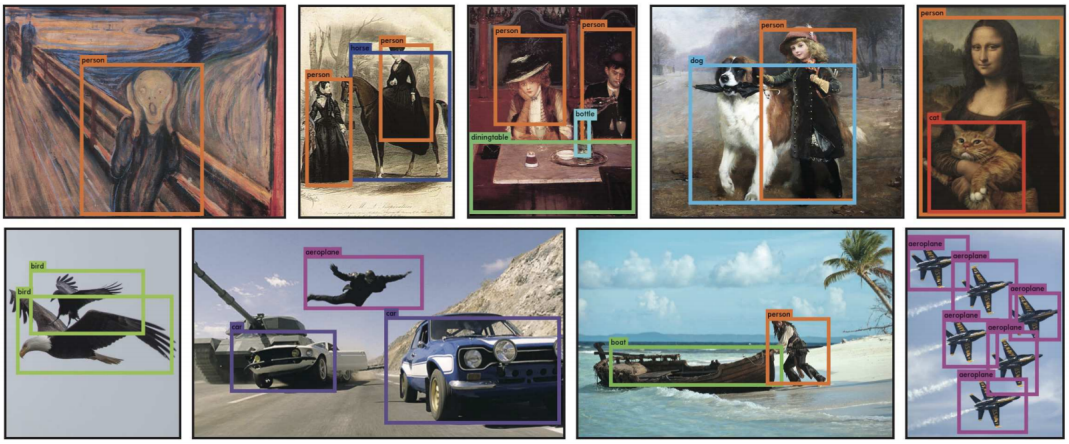

Figure 1.2: Sample image from the work by Redmon et al. (2016) with various objects detected and located in both artwork and natural images. Interestingly, the model mistakenly classifies the object as an airplane instead of a person in the second image of the second row.

These few examples merely illustrate the true capacity of modern AI, and it is therefore only reasonable to optimistically speculate that deep learning based systems are the key to achieving full, desired automation within the transportation industry and to acknowledge that higher levels of autonomy have a great potential to offer robust and effective solutions that will have significant impact in a reduction of on-road risk.

Nevertheless, despite the great advancements within the intelligent technologies that AVs are being equipped with there still exists a number of tasks that pose a variety of complicated challenges. For instance, it is critical that an autonomous vehicle, just like a human driver, has the capability accurately analyse and track surrounding entities such as vehicles and pedestrians within its immediate environment, with an awareness of their state, i.e. position and relative speed. The ability to track

and analyse states of other entities can further allow for a more accurate short-term prediction about their future actions which can in turn enable the AV system to take a set of actions that will drastically decrease its chances of ending up in potentially dangerous on-road situations such as vehicle to vehicle collision.

1.3 Research Aim

The central objective of this thesis is to investigate and develop novel deep learning methodologies to enable accurate prediction of future paths for on-road vehicles. Specifically, the research focuses on the perspective of autonomous vehicles navigating through complex and uncertain environments. This problem can be further formulated as predicting or generating a sequence of future coordinates that describe the possible motion of a tracked agent over a specified time horizon. Forecasting the trajectory of surrounding vehicles is imperative for autonomous vehicles to proactively ensure safety and efficiency in navigating intricate traffic environments, anticipate potential risks, and sustain seamless trajectories; while achieving even brief yet precise predictions spanning 2 seconds is commendable, extending the predictive horizon to 4-6 seconds can yield significant and transformative outcomes. The current approaches towards the discussed task pose a number of challenges (further described in more detail in subsequent chapters) that mainly include poor abilities to perform accurately in complex environments. This in large part is caused by the difficult nature of the task itself but also by the lack of sufficient amounts of quality training data that can be used to train a model that can generalize well to previously unseen data. To improve current motion prediction models the following questions were devised:

Can the accuracy of a motion detection model be improved by incorporating data that more directly models the internal state of the tracked agent?

In general, the training of the motion prediction models is often based on using an internal representation of the global surroundings with use of e.g. high definition maps. This greatly ignores the fact that the local surroundings of the agent, devised from such types of data, could potentially further improve the performance of the model. To answer this question, a novel usage of high definition maps in a form of local maps will be explored and examined.

Can the computational complexity (number of model's parameters) of motion prediction model be reduced without scarifying its performance?

A general trend within the deep learning domain revolves around usage of large pre-trained models that are not necessarily trained on the target domain, this poses two important difficulties. First of all, autonomous vehicles should be equipped with models that are relatively low in complexity considering the need to perform a number of tasks in real-time, and secondly using models that are pre-trained on a distinct domain can potentially reduce the performance of the model as opposed to improving it. The above question will be tackled by examining and introducing a model based on novel capsule networks (Hinton et al. 2011) that alleviates a number of drawbacks introduced by currently used vision models and is better suited towards learning from smaller quantities of available data. In addition, due to the nature of the task the model will be further extended with use of generative framework to allow predicting a large number of future trajectories of the tracked agent.

Based on devised research questions the main hypothesis that will be investigated within this thesis in order to overcome the discussed challenges are:

1. *Incorporating local information about context of the tracked agent's surrounding can be exploited to increase the general performance of motion prediction model.*

2. *The computational complexity of the motion predictor can be significantly reduced by training a model based on a robust capsule network from scratch directly on the target domain.*

1.4 Contributions

Throughout the thesis there are number of contributions that are being made which aim to alleviate identified issues. The following summarises the main contributions made throughout the thesis:

1. An introduction and examination of disentangled local semantic maps. An alternative way of using rasterised maps to encode the context of the environment and focus more specifically on encoding the information about the state of the tracked agent.

2. The first use of capsule networks with respect to the task of predicting short-term future motion of vehicles in complex environments. The introduced capsule based motion predictor demonstrates a promising avenue for the training of smaller models on a limited quantity of data.

3. Extension of the proposed capsule based motion predictor with a generative model, more specifically a conditional variational auto-encoder (Sohn et al. 2015) which enabled the model to account for the multi-modal (more than one output prediction i.e. multiple trajectories in this case) nature of the task and predict a diverse set of plausible paths with regards to the tracked agent.

4. Examination of the proposed generative motion predictor with use of an alternative loss function, the Minimum over N (MoN) which leads the model to learn a more diverse set of predictions and significantly improved its accuracy.

5. Comparison of the final model's performance against recent state-of-the-art methods from the literature on publicly available dataset.

1.5 Thesis Outline

The chapters within this thesis are organised into two background chapters, three technical chapters and finally a conclusion chapter.

More specifically, chapter 2 provides a necessary technical background about the deep learning methodology, starting off with some history of neural networks, then moving onto most common types of layers that are used for different types of tasks (e.g. vision or language tasks) and finally finishing off with the explanation of back-propagation and optimization methods used to train deep networks. Next, chapter 3 provides an extensive overview of the literature with a focus on the task of motion prediction. First, the classic set of methods based on physical models are being discussed. Then a set of sampling based algorithms such as Monte Carlo methods are further introduced. Next, motion prediction algorithms based on Bayesian framework models are reviewed in order to give an overview of probabilistic models. Finally the most recent machine learning (and deep learning) based methodologies are thoroughly addressed to identify their strengths and weaknesses.

Chapter 4 introduces and examines the first usage of local geometrical semantic maps and establishes a set of baseline models that are used in a subsequent chapter of the thesis. Furthermore, chapter 5 focuses on developing a novel method for predicting short-term motion of tracked agents with use of capsule based networks, stating its advantages over a classical convolutional neural network and examining its performance against some of the most popular CNN based feature extractors. Next, chapter 6 extends the model presented in chapter 5 with an introduction of conditional variational auto-encoder and further examines the usage of an alternative

cost function known as Minimum over N (MoN). Lastly, the performance of the final generative model is compared against recent state-of-the-art methods to demonstrate its capabilities on a public dataset that includes a diverse set of scenarios from complex, urban environments.

Chapter 2

Deep Learning - Technical Background

2.1 History

2.1.1 McCulloch-Pitts Model

In 1943 Warren McCulloch and Walter Pitts presented the first computational model of an artificial neuron known as McCulloch-Pitts neuron (MCP) or linear threshold gate that was inspired by the behavior of a biological neuron (McCulloch & Pitts 1943). Figure 2.1 demonstrates the structure and the data flow that occurs within a biological neuron, as observed the neuron is primarily defined by four major parts: *dendrites*, *axon*, *soma* also known as cell body and *synapses* (Zurada 1992).

In comparison to the biological neuron, the McCulloch-Pitts neuron simulates its behavior in a similar, yet simplified fashion. First, the artificial neuron receives a binary vector $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]$ where $x_i \in \{0, 1\}$ which can be compared to electrical signals collected by dendrites. Next, a vector of fixed connections values

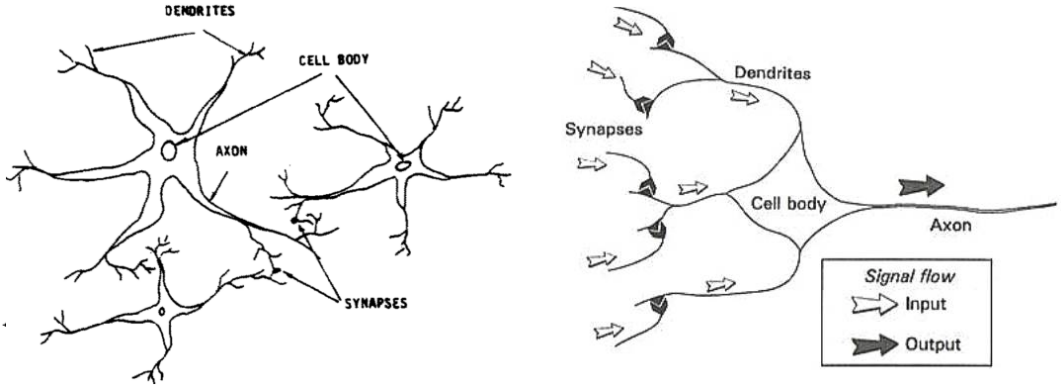

Figure 2.1: Simplified representation of the biological neuron (left) (Hajela & Berke 1991) as well as a flow of input-output information within neuron's body (right) (Gurney 1997).

(weights) $\boldsymbol{w} \in \mathbb{R}^n$ is defined in order to increase or decrease input values by weighting each x_i by its corresponding w_i. The weighted sum of $\boldsymbol{x}$ and $\boldsymbol{w}$ is further computed to produce the neuron's output value, thus imitating the process of the cell body:

$$z = x_1 w_1 + x_2 w_2 + ... + x_n w_n = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

where:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \tag{2.2}$$

Lastly, the weighted sum is passed through an activation function $\varphi(z)$ in this case the linear threshold function or step function to compute the final output. The output of the step function is further dependant on the empirically defined threshold value θ that determines whether the artificial neuron should "fire" (pass the signal to the subsequent neuron) or not:

$$y = \varphi(\boldsymbol{z}) = \begin{cases} 1 \text{ if } \boldsymbol{z} \geq \theta \\ 0 \text{ if } \boldsymbol{z} < \theta \end{cases} \tag{2.3}$$

The following figure presents the flow of data within McCulloch-Pitts model. One can notice that the artificial neuron strongly resembles procedures that take places with its biological counterpart (Fig. 2.1).

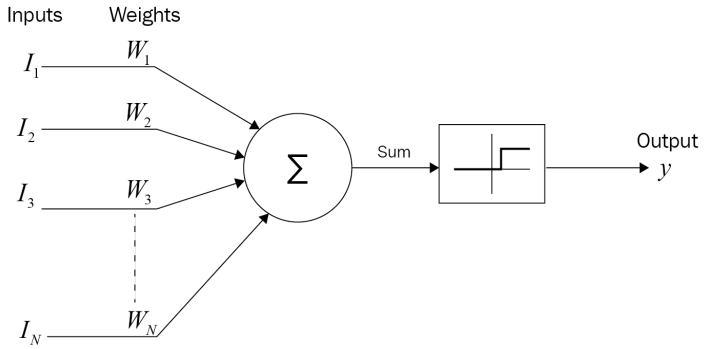

Figure 2.2: McCulloch-Pitts model of an artificial neuron that takes the weighted sum of inputs and corresponding weights and passes it through an activation function to compute the final output value in a binary form (Rothman 2018).

Unfortunately the model presents several limitations as it can be only used for linearly separable tasks. In addition its parameters (weights and threshold) are pre-defined and fixed, and the input-output values can only be binary, thus greatly restricting the model.

2.1.2 Perceptron

The next major advancement happened in 1958 when Frank Rosenblatt developed what is known as perceptron, a supervised learning algorithm for binary classification task (Rosenblatt 1957). The idea of a perceptron was based on the classical MCP model, with one crucial difference. Rosenblatt's main contribution came from the introduction of a perceptron's learning rule which enabled the algorithm to learn or

adjust its weights to a set of values that would allow it to draw a linear decision boundary between two linearly separable classes.

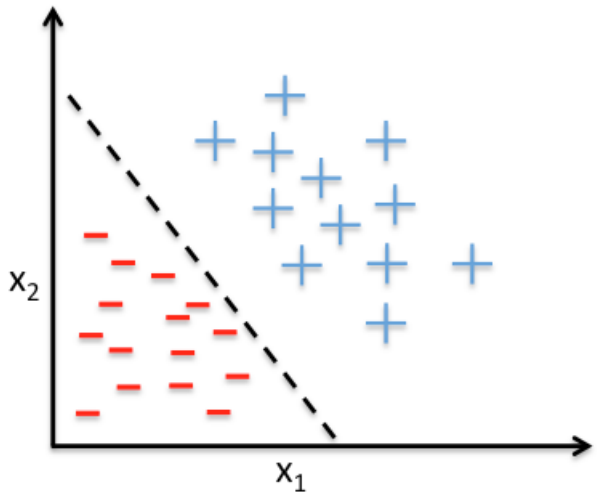

Figure 2.3: An example of two classes being linearly separated by decision boundary drawn with use of perceptron.

During the learning phase the model is exposed to a data set that contains a set of training examples $\boldsymbol{X}$ and a set of corresponding output labels $\boldsymbol{Y}$. Firstly, the vector of weights is initialized either with small random numbers or with zeros. Next, for each training sample $\boldsymbol{x_i} \in \boldsymbol{X}$ the output class label $\hat{\boldsymbol{y}}$ is computed by passing the weighted sum of weights $\boldsymbol{w}$ and input features from the current training sample $\boldsymbol{x_i}$ to the activation function $\varphi(\boldsymbol{z})$ (Eq. 2.1 and 2.3). The computed or predicted $\hat{\boldsymbol{y}}$ is compared with the corresponding ground truth label $\boldsymbol{y_i} \in \boldsymbol{Y}$ to calculate the error value and update the weights accordingly. The perceptron's learning rule states that each weight $w^j \in \boldsymbol{w}$ is formally updated as:

$$w^j := w^j + \Delta w^j \tag{2.4}$$

and the Δw^j value for updating weight w^j is calculated as:

$$\Delta w^j = \eta(y_i - \hat{y}_i)x_i^j \tag{2.5}$$

where w^j is the weight connection for the j input feature in x_i training sample, y_i is the ground truth class label, $\hat{y}_i$ is the predicted label and η is a small constant value between 0.0 and 1.0 typically known as learning rate. The learning rate is a crucial hyperparameter in training perceptron models, as it significantly influences the convergence and stability of the learning process.

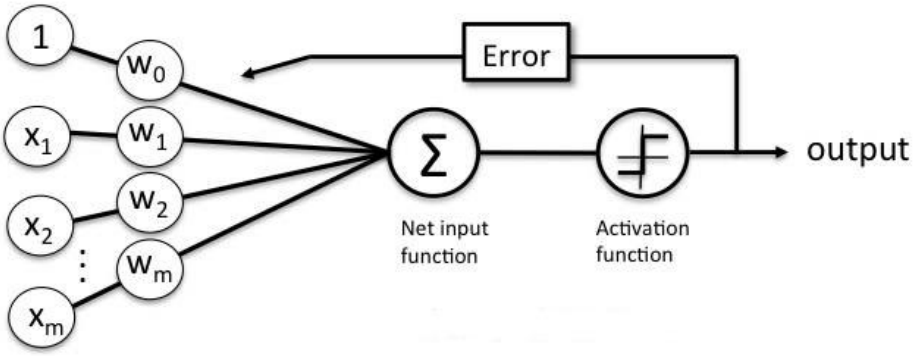

Figure 2.4: Perceptron learning rule.

Figure 2.4 demonstrates how similar the perceptron is in comparison to the MCP model (see Figure: 2.2) with the difference of computing the error in order to update the weights.

In contrast to the MCP, the perceptron does not only accept binary values as its input but rather any real values, however, it's main advantage comes from the capability of learning weights with respect to the input data, giving it more flexibility and an ability for greater generalization. The model does certainly demonstrate several improvements over classical MCP, nonetheless, it is still largely restricted only to binary classification tasks where the data must be linearly separable. Despite the limitations of both MCP and perceptron, both models made an enormous impact within the field and laid a foundation ground for the future development of artificial neural networks that will be discussed further in the following subsections.

2.2 Multilayer Perceptron

A model of a perceptron can be further understood as a two-layered neural network with an input and an output layer. A multilayer perceptron (MLP) also referred to as feed-froward neural network builds upon the idea of a conventional model of a perceptron by introducing at least one or more hidden layers between an input and an output layers as well as a non-linear activation functions e.g *sigmoid*:

$$\varphi(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

Figure 2.5 demonstrates an example of an MLP with four layers where the data traverses in a forward way from input to output layer through intermediate hidden layers. One can notice that every neuron (node) in one layer is connected to every other neuron in the following layer through the weighted connections, thus enabling the model to create more complex internal representation of the data within each consecutive layer.

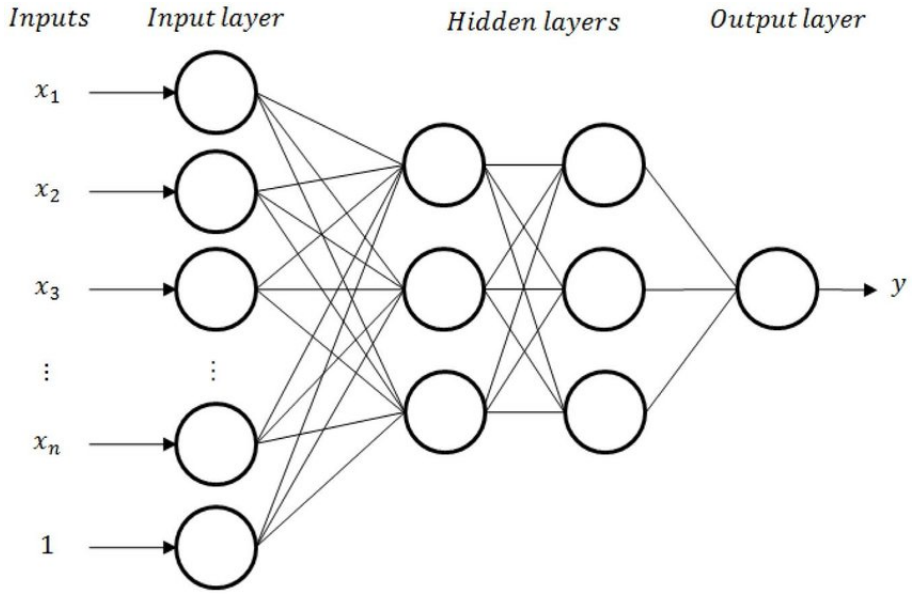

Figure 2.5: Multilayer perceptron with a single input layer, two hidden layers and an output layer with a single node (Guérillot et al. 2017).

The aim of such an MLP is to approximate an arbitrary function f^* by learning values of connection weights $\boldsymbol{W}$ between the neurons so that $f^* : \boldsymbol{x} \rightarrow \boldsymbol{y}$ where $\boldsymbol{x}$ is a vector of features corresponding to a single data sample and $\boldsymbol{y}$ is a desired output (LeCun et al. 2015).

To further understand the computational process that occurs within an MLP let's consider an example of a regression task where a three-layered network with input layer l^{in}, one hidden layer l^h and output layer l^{out} is trained to predict an output vector $\hat{\boldsymbol{y}}$ so that $\hat{\boldsymbol{y}} \approx \boldsymbol{y}$. First, let $\mathcal{D}$ denote the dataset with $\boldsymbol{X}$-$\boldsymbol{Y}$ input-output pairs where $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, $\boldsymbol{Y} \in \mathbb{R}^{m \times n}$ with m denoting a number of samples within $\mathcal{D}$, and n defining the size of input-output vectors. Next let $\boldsymbol{x} \in \boldsymbol{X}$ represent a single data sample with $\boldsymbol{y} \in \boldsymbol{Y}$ being a corresponding ground truth label, $\boldsymbol{W}$ randomly initialized connection weights between neurons and $\boldsymbol{b}$ a bias vector.

The l^{in} is initialized with n neurons where n is equal to the size of input data $\boldsymbol{x}$, thus an i^{th} neuron of l^{in} corresponds to $x_i \in \boldsymbol{x}$. As mentioned before each neuron in layer l is connected to all neurons in layer $l-1$, and therefore an input $\boldsymbol{z}$ to the first neuron in hidden layer l_1^h i.e. weighted sum of connection weights and input values can be computed as:

$$\boldsymbol{z} = l_1^{in}\boldsymbol{W}_{1,1}^h + l_2^{in}\boldsymbol{W}_{1,2}^h + \ldots + l_n^{in}\boldsymbol{W}_{1,n}^h + \boldsymbol{b} \tag{2.7}$$

where l_n^{in} is the n^{th} neuron in layer l^{in} and $\boldsymbol{W}_{1,n}^h$ is the weight connecting neuron l_1^h with neuron l_n^{in}. The final output value of neuron l_1^h is further calculated by passing $\boldsymbol{z}$ through a non-linear activation function e.g. Sigmoid function (see Equation:2.6):

$$l_1^h = \varphi(\boldsymbol{z}) \tag{2.8}$$

All neurons in each layer are computed with 2.7 and 2.8 although the choice of

activation function may vary throughout the layers. The choice of an appropriate activation function is often determined by numerous factors, for example, when predicting values within range 0-1 such as probabilities, it is desirable to use an activation function whose output offers this range. Other reasons might include computational speed or ease of optimization. Next, the predicted value $\hat{\boldsymbol{y}}$ i.e. the output value of a neuron in l^{out} is compared with $\boldsymbol{y}$ to measure the prediction error for the particular sample $\boldsymbol{x}$. This is further repeated for all data points $x_1, x_2, \ldots, x_n$ with use of a loss function $J(\boldsymbol{W})$ e.g. *Mean Squared Error* (MSE) which computes an averaged squared distance between all predictions and the corresponding ground truth such that:

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y}^i - \hat{\boldsymbol{y}}^i)^2 \tag{2.9}$$

The result of the $J(\boldsymbol{W})$ is used by *backpropagation* (Rumelhart et al. 1988) to compute its gradient $\Delta J(\boldsymbol{W})$ with respect to parameters $\boldsymbol{W}$. The computed gradient is further used by an optimisation algorithm e.g. *gradient descent* (Ruder 2016) to slightly adjust the weights in a way that minimizes $J(\boldsymbol{W})$, hence allowing the network to predict $\hat{\boldsymbol{y}}$ that is closer to the ground truth $\boldsymbol{y}$. This briefly explains a single epoch (iteration) of the MLP training process which is repeated for n number of epochs or until the network converges i.e. the global minimum of $J(\boldsymbol{W})$ has been reached.

2.3 Convolutional Neural Network

The development and design of neural networks for processing data with grid-like topology (e.g. images) known as *Convolutional Neural Network* (CNN) goes back to the 80's (Fukushima 1980, Waibel et al. 1989) and have been primarily inspired by the prior research on the visual system and processing within animal's brains (Hubel & Wiesel 1968). And although in 1989 LeCun et al. (1990) successfully

demonstrated capabilities of CNN's on a real-world task by training such a network with backpropagation to recognise handwritten digits, it wasn't until 2012 when Alex Krizhevesky won the annual ImageNet challenge with *AlexNet* (Deep CNN) (Krizhevsky et al. 2012) that these networks have been truly recognized.

Similar to an MLP, a CNN architecture and model is composed of number of layers where layer l is connected to layer $l+1$, however, the main difference comes from the use of a *convolution operation* or a *convolution layer* instead of a general matrix multiplication in at least one of layers (Goodfellow et al. 2016). First, an input image $\boldsymbol{I} \in \mathbb{R}^{h_1 \times w_1}$ is fed to the initial convolution layer which is composed of number of learnable 2D *filters* or *kernels* that can be further considered as CNN's weights $\boldsymbol{W}$. Each filter is defined as a $\boldsymbol{K} \in \mathbb{R}^{h_2 \times w_2}$ where $h_2 \leq h_1$ and $w_2 \leq w_1$. The main idea behind convolution is to "scan" a small portion of an input image $\boldsymbol{I}$ with a *local receptive field* whose size is defined by the size of $\boldsymbol{K}$, and compute the dot product between "scanned" patch and $\boldsymbol{K}$, hence a result of discrete 2D convolution between matrices $\boldsymbol{I}$ and $\boldsymbol{K}$ at point (i, j) is computed with:

$$(\boldsymbol{I} \cdot \boldsymbol{K})_{i,j} = \sum_m \sum_n \boldsymbol{I}_{i-m,j-n} \boldsymbol{K}_{m,n} \tag{2.10}$$

which can be further visualized with the following figure: The receptive field, thus

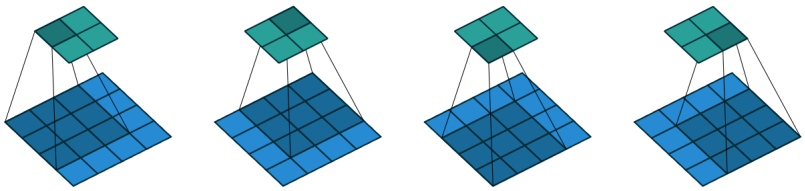

Figure 2.6: Convolution operation between input matrix (lower blue grid) and kernel matrix. The darker blue grid indicates patches of the input data that are currently scanned by the receptive field. Each patch is multiplied by the respective kernel values and summed up to yield the state of the neuron (dark green node) in the feature map (upper green grid) (Dumoulin & Visin 2016).

$\boldsymbol{K}$ is then "shifted" by the stride $\boldsymbol{s} = (h, w)$ for both height and width of $\boldsymbol{I}$, repeating the convolution until the whole of $\boldsymbol{I}$ has been covered by $\boldsymbol{K}$ (Fig. 2.6).

Such convolution yields a matrix $\boldsymbol{F}$, often called *feature map* that stores states of computed neurons. Interestingly, what can be observed from figure 2.6 is that the spatial size $\boldsymbol{o}^f$ of $\boldsymbol{F}$ is smaller than $\boldsymbol{I}$, more precisely $\boldsymbol{o}^f = \left\lfloor \frac{\boldsymbol{o}^i - \boldsymbol{o}^k}{\boldsymbol{s}} \right\rfloor + 1$ where $\boldsymbol{o}^i$ and $\boldsymbol{o}^k$ is the size of $\boldsymbol{I}$ and $\boldsymbol{K}$ respectively. In numerous situations however, it is desirable to e.g. retain the spatial size of an input or convolve over it such that the size of an output feature maps is a fraction of the original size of an input. To overcome this, a convolution layer introduces another hyperparameter $\boldsymbol{p}$ known as *padding* which pads the input volume with zeros around its borders. The padding amount can be further determined by its mode, and thus the size of $\boldsymbol{o}^f$ is now defined as $\boldsymbol{o}^f = \left\lfloor \frac{\boldsymbol{o}^i - \boldsymbol{o}^k + 2\boldsymbol{p}}{\boldsymbol{s}} \right\rfloor + 1$. Figure below demonstrates the use of **same** padding mode, one can clearly notice that such mode allows to preserve the original size of $\boldsymbol{I}$ at its output.

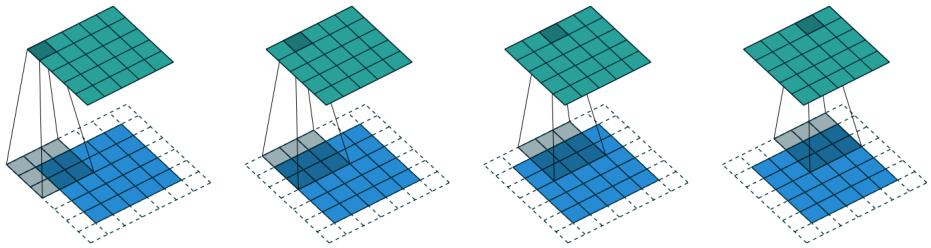

Figure 2.7: Convolution between $\boldsymbol{I} \in \mathbb{R}^{5\times5}$ and $\boldsymbol{K} \in \mathbb{R}^{3\times3}$ where $\boldsymbol{s} = (1, 1)$ and $\boldsymbol{p} = same$ which yields $\boldsymbol{F} \in \mathbb{R}^{5\times5}$ (Dumoulin & Visin 2016).

It is worth noting that the convolution layer produces several feature maps using learnable filters, where each filter aims to detect or extract certain feature from the input image (e.g. straight line). Therefore, the output of convolution layer is a 3D tensor $\mathbf{F} \in \mathbb{R}^{h\times w\times c}$ where h and w denote the height and width of each feature map respectively, and c denotes the total number of feature maps. In addition, each $\boldsymbol{F} \in \mathbf{F}$ is passed through a non-linear activation function e.g. 2.6 before being passed further to a network's consecutive layers e.g. pooling layer.

In a typical CNN architecture the convolution layer is generally followed by a *pooling layer* which aims to summarise a $m \times n$ area of the input feature map. The pooling

mechanism can be further understood by comparing it to the convolution where a $h \times w$ kernel "shifts" through the input by stride $\boldsymbol{s}$. Pooling employs similar structure, however, instead of using kernels and computing dot product between $\boldsymbol{I}$ and $\boldsymbol{K}$, it extracts e.g. a **maximum** or **average** value from the current $m \times n$ patch. The result is a spatially reduced feature map that preserves most valuable information. In addition to downsampling, pooling also helps to attain a certain level of invariance with respect to various linear transformations e.g. translation, rotation.

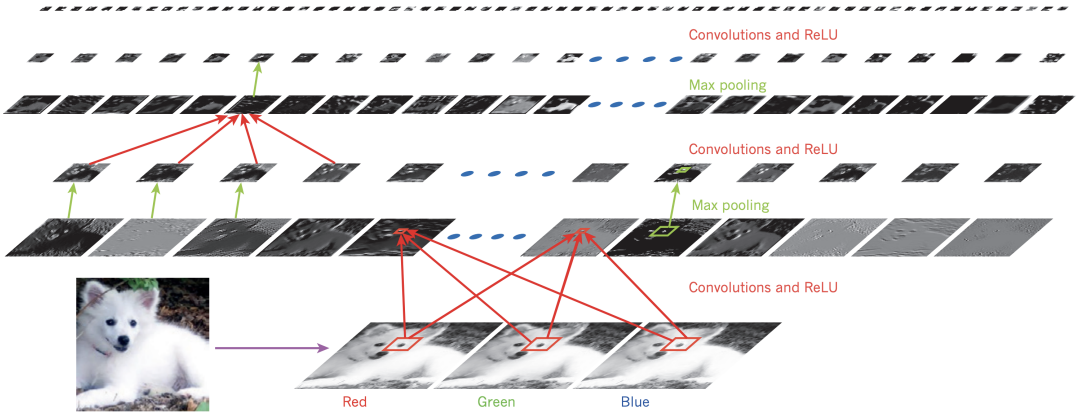

Figure 2.8: An example of a CNN architecture. The input image of the Samoyed dog (bottom left corner) runs through number of convolution layers where learned filters extract distinctive features, creating several features maps. Each convolution layer is also followed by pooling layer where spatial size of its output is significantly reduced (LeCun et al. 2015).

As can be concluded, a CNN introduces within its architecture several techniques that makes it more suitable for processing for instance an image data where modelling as well as maintaining a spatial relationship between features is critical. First and foremost, the idea of kernels and receptive field allows the network to greatly reduce the number of learnable parameters, this is formally known as *weight sharing* where each small kernel (weights) is reused over the entire input data. In comparison, a feedforward network with an input image of size 100×100 would have 10,000 nodes in its input layer, resulting in 10,000s connection weights with proceeding

layer. Furthermore, image data have a strong 2D local structure which needs to be maintained at least to a certain degree to extract local features and preserve correlation between them. MLP ignores that as the input to the network in case of 2D image would be a flatten matrix i.e. vector of features, and therefore the result of training would be constant regardless of the order of nodes (pixel values) in input layer (LeCun et al. 1995).

Figure 2.9: Visualisation of features learned by a CNN. Each column (from left to right) corresponds to a deeper layer within the network, thus the visualisation becomes more abstract within each consecutive layer. Features range from simple features such as edges, textures (first two columns) to complex features such as parts and whole objects (last two columns) (Olah et al. 2018).

2.4 Recurrent Neural Network

Previous subsections gave a brief overview of two types of networks, MLP where the information flows unidirectionally from an input to an output layer, and CNN that can effectively capture spatial dependencies and extract salient features from

e.g. images. Yet, it is often the case that the network's predicted output depends on the information from the past, e.g. anticipating vehicle's future position greatly relies on its past trajectory. *Recurrent Neural Network* (RNN) (Rumelhart et al. 1988) have been designed specifically to model temporal dependencies and process sequential data. These types of network have been extensively applied to a variety of tasks, mainly in the field of natural language processing (NLP), for instance speech recognition (Graves et al. 2013), machine translation (Kalchbrenner & Blunsom 2013) and language modelling (Mikolov 2012).

Figure 2.10 demonstrates a comparison of a MLP and RNN architecture with a single hidden layer. As previously discussed, in MLP the data flows in one direction i.e. from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ through hidden state $\boldsymbol{h}$, in contrast the input of RNN's hidden state $\boldsymbol{h}^t$ where t refers to a particular point in time (or position in the sequence) is composed of the input vector $\boldsymbol{x}^t$ at time t and the output of the hidden layer from previous timesteps. Feeding the hidden layers of RNN with the data from previous timesteps enables the model to retain information about the past samples, hence $\boldsymbol{h}$ can be further interpreted as a "naive memory" of the network that contains a summary of past sequence of inputs up to t.

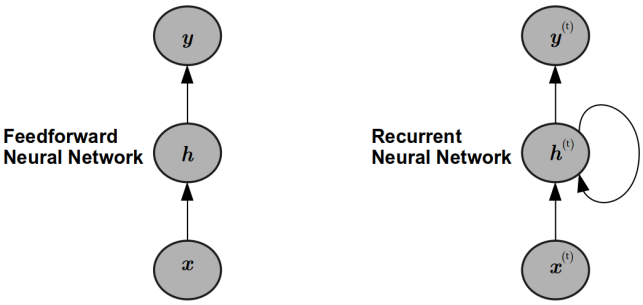

Figure 2.10: A comparison of MLP and RNN architecture with a single hidden layer.

As stated before RNN's hidden layer (note that both state and layer will be used interchangeably throughout this section) receives its incoming signal from the input layer as well as from past hidden layers, therefore the computation of a forward pass is almost identical to that of MLP. To understand the process further let's consider

an example of a RNN with just a single hidden layer. Figure below illustrates an unfolded computational graph of the RNN presented earlier in figure 2.10

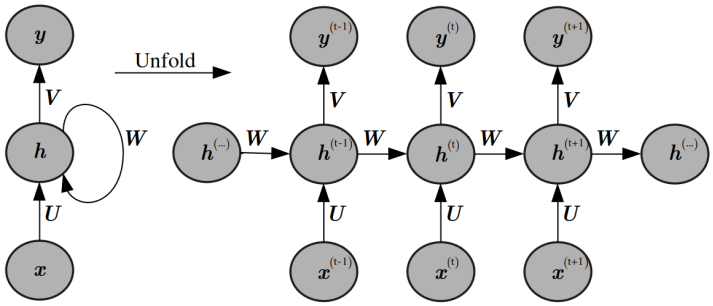

Figure 2.11: Unfolding of the computational graph of a single layered RNN. Note, for simplicity the bias term as well as non-linear activation function have been omitted.

Let $\boldsymbol{x}^t$, $\boldsymbol{y}^t$ and $\boldsymbol{h}^t$ denote the input, output and hidden state of the network at time t respectively. The connection between $\boldsymbol{x}^t$ and $\boldsymbol{h}^t$ is parameterized by weight matrix $\boldsymbol{U}$, whereas the recurrent connection from $\boldsymbol{h}^{t-1}$ to $\boldsymbol{h}^t$ by weight matrix $\boldsymbol{W}$, hence, the output of $\boldsymbol{h}^t$ can be computed with the following equation:

$$\boldsymbol{a}^t = \boldsymbol{U}\boldsymbol{x}^t + \boldsymbol{W}\boldsymbol{h}^{t-1} + b \tag{2.11}$$

$$\boldsymbol{h}^t = \varphi(\boldsymbol{a}^t) \tag{2.12}$$

where b corresponds to the additional bias term and φ represents a non-linear activation function. The result of $\boldsymbol{h}^t$ is further used to obtain the output of the network at time t (i.e. $\boldsymbol{y}^t$) by applying the following:

$$\boldsymbol{o}^t = \boldsymbol{V}\boldsymbol{h}^t + b \tag{2.13}$$

$$\boldsymbol{y}^t = \varphi(\boldsymbol{o}^t) \tag{2.14}$$

where $\boldsymbol{V}$ denotes the connection weights between $\boldsymbol{h}^t$ and $\boldsymbol{y}^t$. One can notice that the unfolded graph illustrates a crucial idea that allows RNNs to effectively process

sequential data i.e. parameter sharing. Weight matrices $\boldsymbol{U}$, $\boldsymbol{W}$ and $\boldsymbol{V}$ are shared across timesteps (or sequence lengths), therefore the network can exploit that property to generalize to the data of various sequence length. Let's consider an example of the following two sentences "Yesterday, I went to the cinema" and "I went to the cinema yesterday", standard MLP would have a separate parameters for each input feature, thus, it would learn rules for a sentence where words are placed in specific order, whereas RNN model would present generalization capabilities due to the use parameter sharing (Goodfellow et al. 2016).

Unfortunately, training a conventional RNN can be difficult if the input sequence is long (e.g. over 10 timesteps) as the gradient that propagates back through time tends to vanish or explode exponentially, thus preventing the network from updating weights and learning the appropriate mapping between input-output. In the literature this issue is commonly referred to as *vanishing gradient* (Bengio et al. 1994) and over the years numerous solutions were proposed to address it with a common RNN choice being a *LSTM* (Hochreiter & Schmidhuber 1997) or *GRU* (Chung et al. 2014) architectures.

2.5 Backpropagation and Optimization

2.5.1 Backpropagation

The process of finding an optimal set of weights during the training of a neural network is at its heart guided by the previously mentioned algorithm called *Backpropagation* (Rumelhart et al. 1988). The backpropagation algorithm is used to compute the gradient of a function where in a case of neural networks the function is an arbitrary loss function that measures e.g. the difference between the network's predictions and the ground truth data. The gradient vector is further computed with

respect to all parameters of the network using chain rule which describes the impact of each parameter on the objective function and defines the direction in which the weights should be adjusted as to minimize (negative gradient) network's overall loss.

As an example, let's first start with a simple case of performing a backpropagation on a single neuron (a perceptron) with two inputs x_1, x_2 and a single output y. Let $f : (x_1, x_2) \rightarrow y$ be the mapping function that describes the computation of the neuron which is further parameterized by corresponding weights and bias term such that $f = f(x_1, x_2; w_1, w_2, b)$. Recall that the activation of a neuron is computed by first multiplying inputs with corresponding weights, let $h(x_i, w_i) = x_i w_i$ define the function that multiplies a single input x_i with a corresponding weight w_i. Next, each output of $h(x_i, w_i)$ is summed with the bias term to obtain the raw score of a neuron which can be defined by the following function given two inputs $z = g(h(x_1, w_1), h(x_2, w_2), b) = h(x_1, w_1) + h(x_2, w_2) + b$. The computed raw score is then passed through some activation function, for this example let the activation function be a ReLU non-linearity:

$$\varphi(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.15}$$

thus, the final activation of a neuron is $a = \varphi(z)$. Since this is a case of a single neuron the activation a is the actual target output $\hat{y}$ such that $a = \hat{y}$. Lastly, in order to measure the difference between a neuron's output $\hat{y}$ and the target ground truth y a loss function is employed, in this case a simple MSE such that $J(\hat{y}, y) = (\hat{y} - y)^2$. As can be seen from the example the computation required to obtain the measure of how close the prediction is to the ground truth follows the following chain of functions $J(\varphi(g(h(x_1, w_1), h(x_2, w_2), b)), y)$. Recall that the aim of backpropagation is to compute the gradient vector of a loss function which contains a set of partial derivatives with respect to the network's parameters, in this case the parameters of

a single neuron are w_1, w_2 and b therefore the aim is to obtain the following:

$$\nabla J(\hat{y}, y) = \begin{bmatrix} \frac{\partial}{\partial w_1} J(\hat{y}, y) \\ \frac{\partial}{\partial w_2} J(\hat{y}, y) \\ \frac{\partial}{\partial b} J(\hat{y}, y) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial w_1} \\ \frac{\partial}{\partial w_2} \\ \frac{\partial}{\partial b} \end{bmatrix} J(\hat{y}, y) \tag{2.16}$$

and as mentioned before, the chain rule which states that the derivative of a function chain is a product of derivatives of all functions in the chain is being used to compute each partial derivative within the gradient vector. For example, the chain rule states that the derivative with respect to x given the following chain of functions $f(g(x))$ is:

$$\frac{d}{dx} f(g(x)) = \frac{df(g(x))}{dg(x)} \frac{dg(x)}{dx} \tag{2.17}$$

and therefore the partial derivative of the loss with respect to e.g. w_1 can be further computed using the chain rule as:

$$\frac{\partial J(\hat{y}, y)}{\partial w_1} = \frac{\partial J(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial x_1 w_1} \frac{\partial x_1 w_1}{\partial w_1} \tag{2.18}$$

where the partial derivative of the loss function (MSE) with respect to the neuron's prediction is calculated as follows:

$$\frac{\partial J(\hat{y}, y)}{\partial \hat{y}} = 2(\hat{y} - y) \tag{2.19}$$

and the partial derivative of the ReLU non-linearity with respect to the neuron's raw output is computed as:

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial ReLU(z)}{\partial z} = 1(z > 0) = \begin{cases} 1 \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.20}$$

and furthermore, the partial derivative of a neuron's output z which is a result of $g(h(x_1, w_1), h(x_2, w_2), b) = h(x_1, w_1) + h(x_2, w_2) + b$ with respect to $h(x_1, w_1)$ is

simply 1. Lastly, the partial derivative of $h(x_1, w_1)$ with respect to the weight w_1 is:

$$\frac{\partial h(x_1, w_1)}{\partial w_1} = \frac{\partial x_1 w_1}{\partial w_1} = x_1 \tag{2.21}$$

and therefore computing the partial derivative of a loss with respect to w_1 can be formulated as:

$$\frac{\partial J(\hat{y}, y)}{\partial w_1} = 2(\hat{y} - y)1(z > 0)x_1 \tag{2.22}$$

This process can be then repeated to find the partial derivatives of w_2 and b and compute the gradient vector. Generalizing the above process where a backpropagation, or a backward pass is performed on an arbitrarily deep neural network is also trivial. For instance, considering the current example of a single neuron, let that particular neuron connect to multiple neurons in the proceeding layer within the network, thus each of those neurons receives an input (an activation) from that one particular neuron. In such a case, when backpropagation is performed, each neuron from the proceeding layer will return its partial derivative with respect to an input received from a neuron from a previous layer. Therefore, that single neuron in the current layer will receive, instead of a single value, a set of values (vector) that encapsulates partial derivatives of all neurons from the proceeding layer to which that particular neuron is connected to. The vector that contains all those partial derivatives is then summed up so that the neuron at the current layer ultimately receives a single value that defines it's impact on the proceeding layers and ultimately the loss of the whole network.

2.5.2 Optimization

Once the backpropagation is finished computing partial derivatives of loss with respect to each weight within the network the last step in the learning process of the network can be executed. The optimization of neural network is relatively straight

forward at its core and follows a slightly similar process to the one used within perceptron's learning rule defined in equations 2.4 and 2.5. In essence, given a weight w (or a bias b) and it's computed derivative Δw the simplest optimization step can be performed as follows:

$$w_{i+1} = w_i - \eta \Delta w_i \tag{2.23}$$

where η defines a learning rate. The above equation defines the simplest optimization algorithm known as Gradient descent (GD) (Ruder 2016) which aims to optimize weights of neural network such that the network converges to some global minimum. In theory, given a convex loss function, the GD algorithm is guaranteed to reach the global minimum, however, in practice when training deep neural networks which are typically used to learn a complex mapping between input and output data, it is non-trivial to reach the global minimum and the network generally ends up in some local minimum. For example, simply choosing an appropriate η value for learning might often be a challenge. A learning rate that is too low might cause the network to stagnate causing the model to be stuck in some non-optimal local minimum. On the other hand, high updates to network's parameters might cause the model to drastically jump over the loss landscape, never reaching some optimal point.

Generally speaking, there exists a number of different variations of the vanilla GD algorithm. Amongst them the most well known are Stochastic gradient descent (SGD) and Mini-batch gradient descent (MBGD). The vanilla GD fits the whole dataset at once whereas the SGD fits a single sample at a time and a MBGD fits a small batch of data which is usually sampled at random. Usually, the vanilla GD is not the optimization algorithm of choice as it's extremely slow and can be intractable. On the other hand, SGD fits only a single sample at a time meaning that it is significantly faster than GD but the training process can be very unstable. Finally, the MBGD, which is actually often refereed to as SGD in the literature or other deep learning resources, combines best of both vanilla GD and SGD. Since

the MBGD performs weights update for every sampled mini batch it means that the training is still considerably fast and most importantly it is significantly more stable since the variance of parameters updates is noticeable reduced as compared to SGD.

2.6 Conclusion

This chapter provided a thorough explanation of the basics of essential deep learning concepts. First of all, a brief history of neural networks was provided, starting with an explanation of a simple McCulloch-Pitts model who's development was inspired by the biological neuron. Next, a perceptron model was introduced to demonstrate the subsequent step in development of an artificial neuron. The perceptron takes the idea that was previously presented with the MCP model, however, in addition it introduces its learning rule, a crucial part of the algorithm which allows the model to adjust its weights so that its output is more precise with respect to the target data. Next, an explanation of Multilayer perceptron was provided to demonstrate further extension of a simple perceptron into an artificial neural network which introduces the idea of using a layer of neurons that propagate data in a forward manner from an input layer, through a number of hidden layers to finally reach the output layer. Furthermore, two popular variants of an artificial neural network were demonstrated, namely the convolutional and recurrent neural networks which aim to deal with spatial and temporal data respectively. Finally, a vital algorithm, backpropagation was clearly explained with a simple example of how the network's error is propagated backward to compute its gradients which are then used by an optimization algorithm to train the network. The next chapter will provide the reader with an extensive review of the literature within the area of vehicle motion prediction.

Chapter 3

Literature Review

This chapter will provide a comprehensive review of the relevant literature within the area of short-term motion prediction. The literature review will primary focus on a trajectory forecasting from the perspective of autonomous vehicle, however, some work from the area of pedestrian motion prediction will also be discussed as numerous methods can be transferable and applicable for vehicle motion forecasting.

3.1 Kinematic and Dynamic Models

Vehicle's motion forecasting has been generally approached with an assumption that the evolution of object's state through time remains primarily governed by laws of physics and can be therefore modeled with some known mathematical model. These methods commonly use a variation of the well known bicycle model (Gillespie 1992) that portrays a vehicle as a two-wheeled entity on a 2D plane (see Figure: 3.1). The bicycle model is then combined with a kinematic motion model that computes an estimate of the future state vector with the following differential equation (Li & Jilkov 2003):

$$\dot{x}(t) = v(t)\cos\theta(t) \tag{3.1}$$

$$\dot{y}(t) = v(t)\sin\theta(t) \tag{3.2}$$

$$\dot{v}(t) = a_t(t) \tag{3.3}$$

$$\dot{\theta}(t) = a_n(t)/v(t) \tag{3.4}$$

where a_n denotes normal acceleration, a_t the tangential acceleration, $\dot{x}, \dot{y}, \dot{v}, \dot{\theta}$ represents target x, y coordinates as well as the velocity and heading angle of a vehicle respectively. Nonetheless, kinematic models are often restricted due to the lack of consideration of numerous forces e.g. tire or aerodynamic drag forces that affect vehicles' motion. Thus, dynamic motion models are often employed to account for those limitations (Rajamani 2011). Some of the popular motion models frequently found in the literature include; constant velocity (CV), constant acceleration (CA), constant turn rate and velocity (CTRV), constant turn rate and acceleration (CTRA), constant steering angle and velocity (CSAV) and constant curvature and acceleration (CCA). A more in-depth review of those models can be found in Schubert et al. (2008). Additionally, filtering algorithms such as Kalman filter (KF) (Kalman 1960) and its variations namely extended KF (EKF) (Jazwinski 2007) and unscented KF (UKF) (Julier & Uhlmann 2004) are often combined with motion models to improve prediction accuracy and account for the noise introduced through numerous sensor measurements. For instance, Lin et al. (2000) proposed a modified dynamic bicycle model in order to predict a vehicle's future path and prevent drivers from drifting off the roadway in the presence of various disturbances e.g. wind effects, road crown. The path prediction however, could only be executed under an assumption that the certain road conditions are met, and the vehicle's speed remains constant for the duration of prediction.

Moreover, Huang & Tan (2006) investigated if the differential GPS (DGPS) based system can be used to obtain more accurate vehicle data such as its position in global coordinate systems against previously proposed methods that focused on data col-

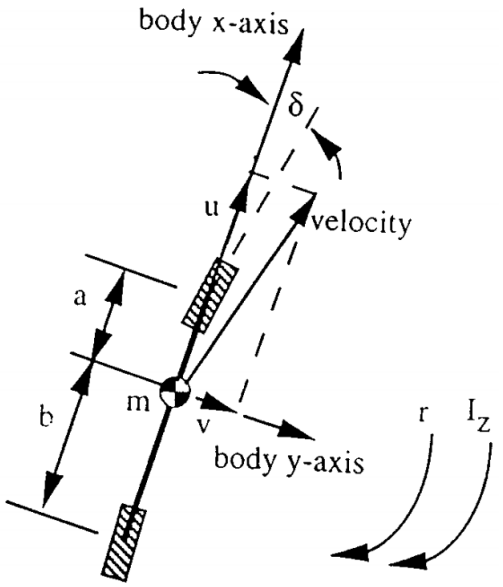

Figure 3.1: An example of two degree of freedom bicycle model by Lin et al. (2000) where δ denotes the front wheel steering angle, u the forward vehicle speed, m the lumped vehicle mass, v the translational velocity along the body-fixed lateral axis, r the yaw rate, I_z the yaw moment of interia and lastly, a and b the distance from vehicle's centre of gravity to the axels.

lection from vision sensors. The captured data was then combined with information describing vehicle dynamics to estimate its future states using proposed models. Nonetheless, presented methods were based on a simple kinematic model that assumed no drastic change of a driver's intention as well as that both longitudinal and lateral motion will remain constant during the short prediction window.

On the other hand, Hillenbrand et al. (2006) incorporated a simple kinematic model to reason about future states of two agents of interest and use it to measure the time until potential collision. The accurate estimation of vehicles future motion with use of proposed model however, was again highly dependant on several assumptions such that the acceleration remains constant during prediction time, and that past states of tracked vehicles are perfectly known.

In addition, Ammoun & Nashashibi (2009) also addressed the issue of risk estimation from the perspective of predicting vehicle's trajectory using dynamic model and

geometrical data that represents road topology. The proposed method applied KF to filter the noisy input data and recursively estimate a future position of the agent. Nevertheless, an accurate approximation could only be achieved if several parameters remained constant during computation time.

Next, Barth & Franke (2008) introduced an image based solution for vehicle tracking and motion estimation. The method proposed the use of a 3D point cloud to detect and represent an object of interest in a spatial domain through time. Additionally, the further use of a 3D point cloud representation allowed to estimate its motion parameters that could be then filtered with the EKF. Unfortunately, the approach could not be considered as robust and reliable as the uncertainty of trajectory prediction grows quadratically as the distance increases.

As previously mentioned, variety of methods in the literature focus on using trajectory prediction as a intermediate step towards detection of dangerous situations that might potentially lead to a collision. This idea was further explored by Batz et al. (2009) where a dynamic model and the KF were adopted for future state estimation. Interestingly, the methodology was presented from a perspective of agents working in a cooperative fashion where the data between vehicles was exchanged for anticipation of both motion and collisions. Yet, in order for the proposed approach to be successful it required no changes of driver's actions during prediction interval, and an access to an advanced digital map that defines the surrounding.

Another body of work that approached the issue of path prediction and collision avoidance from the perspective of modelling an environment with cooperative agents was presented by Lytrivis et al. (2008). The proposed method exploited the dynamic vehicle model and fused the motion data processed by the KF as well as road geometry information using (Jang et al. 1997) to compute more realistic path for up to 3 seconds into the future. In addition, each agent could communicate with other agents within 400 meter radius to exchange KF filtered motion data, which further

allowed for more advanced path anticipation by considering intentions of all agents within the defined reach.

Benefits of including road geometry for the more reliable path prediction were further demonstrated by (Polychronopoulos et al. 2007). The presented approach combined several algorithms as well as data defining road structure to create a hierarchically-structured method for the vehicle's state estimation that was then used to anticipate the driver's intention and detect potentially dangerous situations.

Furthermore, the following work by Houenou et al. (2013) examined and discussed numerous advantages that come from having a manoeuvre recognition module (MRM) integrated into motion forecasting system as its intermediate step. The classification of tracked vehicle's current manoeuvre considered the computed distance between target's current path and road lane's center lines as a classification metric. For instance, if the measured distance between vehicle's path and lane that was being followed was lower than empirically defined threshold, then the vehicle was performing a keeping lane manoeuvre. Next, with regards to the motion prediction, authors combined the following; a Constant Yaw Rate and Acceleration (CYRA) motion model as well as path prediction based on manoeuvre recognition, in order to compute the final, weighted prediction of the future position. Experimental results demonstrated the effectiveness of the final model whereby the prediction error of future path decreased dramatically when comparing separate predictions of both CYRA and MRM, particularly on non-linear paths. For example, the average prediction error of the model based solely on CYRA and for the time horizon of 2-3 seconds was reported to be 2.3 metres whereas the model that combined both CYRA and MRM managed to reduce this error to just 0.28 metres.

3.1.1 Summary

This section extensively explored the application of kinematic and dynamic models, often rooted in the bicycle model. Kinematic models, though foundational, are limited by their oversight of critical forces like tire or aerodynamic drag, prompting the adoption of dynamic models, including constant velocity (CV), constant acceleration (CA), and others. The integration of filtering algorithms, such as Kalman filters, enhances prediction accuracy by addressing sensor measurement noise. While these models offer a diverse range of options for motion prediction, drawbacks include their reliance on assumptions of constant conditions, potential inaccuracies in unpredictable scenarios, and dependencies on idealized circumstances or advanced infrastructure. Robustness issues, particularly in image-based solutions, and the complexity of certain algorithms further highlight the challenges in achieving accurate short-term motion predictions for autonomous vehicles in dynamic real-world environments.

3.2 Monte Carlo Methods

In numerous situations some desired quantity cannot be precisely determined due to e.g. required computational complexity. Instead, its approximation can be obtained using random sampling based algorithms such as Monte Carlo methods (Hammersley 2013) at reduced cost. As the sampling from a given distribution allows to compute integrals in the form of:

$$c = \mathbb{E}_{x \sim P}[f(x)] = \int P(x) f(x) dx \tag{3.5}$$

which can be considered as an expectation over random variable x. Hence, the target expectation c can be further estimated by drawing n samples from P to compute

the empirical average such that:

$$\hat{c}_n = \frac{1}{n}\sum_{i=1}^{n} f(x^i) \approx c = \mathbb{E}_{x\sim P}[f(x)] \tag{3.6}$$

and as **the law of large numbers** states that as the sample size n grows and if all samples $x_i, \ldots, x_n$ are *i.i.d*, then the average converges to the expected value:

$$\lim_{n\to\infty} \hat{c}_n = c \tag{3.7}$$

The following work, described by Broadhurst et al. (2005) defines a framework based on Monte Carlo sampling to approximate a probability distribution of potential future paths for every object in the scene. Next, the agent's trajectories sampled from the estimated distribution are evaluated against approximated paths of all other agents within the scene to determine the potential likelihood of future collision. Moreover, Eidehall & Petersson (2008) extend the threat assessment framework presented by Broadhurst et al. (2005) with an introduction of iterative sampling which removes and replaces sampled trajectories that lead to collision earlier in the processing pipeline.

Furthermore, work presented by Pepy et al. (2006) proposed a combination of the dynamic bicycle model examined by Taheri (1992) with rapidly-exploring random tree (RRT) (LaValle 1998) to reduce prediction error of the path planning module. Authors demonstrated that their method was capable of producing realistic and smooth paths whilst accounting for wheel slipping. Yet, high vehicle speed during computation time restricted the method to successfully find collision free paths.

Next, Petti & Fraichard (2005) introduced the so called partial motion planning (PMP) technique to address the problem of motion planning within dynamic environments composed of variety of stationary and dynamic obstacles. In general, the PMP was used to compute and construct a set of partial, possible trajectories

over small period of time. Then, a small subset of trajectories was explored and examined with RRT as well as inevitable collision state (ICS) (Fraichard & Asama 2004) framework to determine and chose collision-free paths exist within the subset.

An alternative variation of RRT, namely closed-loop RRT (CL-RRT) for motion planning was further described by Kuwata et al. (2009). It is worth mentioning that the outlined algorithm was used during 2007 DARPA Urban Challenge by the MIT team and their vehicle Talos. The CL-RRT computed and then simulated possible future movement of the agent based on the defined vehicle motion model and the control unit. The forward pass of the algorithm generated a set of potential future states which were then simulated and examined against what was defined as the drivability map (2D grid) that encoded the data of the surrounding such as lane boundaries, obstacles etc, to find an optimal, collision-free set of nodes (future states) that could be selected as a candidate trajectory.

3.2.1 Summary

While traditional deterministic methods for predicting agent motion in autonomous vehicles can be computationally expensive, several Monte Carlo and sampling-based approaches offer efficient alternatives. Techniques like Monte Carlo sampling and iterative sampling estimate likely future trajectories by drawing samples from probabilistic models, while RRT-based methods like PMP and CL-RRT explore and refine feasible paths within dynamic environments. These methods excel in reducing computational load and handling complex situations with multiple agents, even achieving success in real-world challenges like the DARPA Urban Challenge. However, limitations exist in their accuracy, computational demands at high speeds, and ability to model long-term interactions. Further research is necessary to refine these approaches and address these limitations for robust and accurate short-term motion prediction in autonomous vehicles.

3.3 Bayesian Framework

3.3.1 Bayesian Networks

Probabilistic models such as Bayesian networks (BNs) (Jensen et al. 1996) allow to capture and define conditionally dependent and independent relationships between a set of random variables. Those are further modelled with a directed acyclic graph where nodes correspond to random variables, and edges represent their relationship as a probability distribution. As an example, let A, B and C denote a set of three random variables where $P(B \mid A)$, $P(C \mid A)$ and both B and C are independent of each other such that $P(B \mid A, C)$, $P(C \mid A, B)$. Hence, the joint probability of B and C given A can be computed as a product of $P(B \mid A)$ and $P(C \mid A)$:

$$P(B, C \mid A) = P(B \mid A)P(C \mid A) \tag{3.8}$$

and so the joint probability of $P(A, B, C)$ can be further defined as:

$$P(A, B, C) = P(B \mid A)P(C \mid A)P(A) \tag{3.9}$$

Alternatively, Hidden Markov models (HMMs) (Rabiner 1989) which can be further perceived as dynamic Bayesian networks where edges of the graph do not have to be direct and can therefore contain cycles are also often employed to model the probability distribution over a sequence of states that are not directly observable (hidden states). The system that is being modeled with the HMM satisfies the following assumptions:

1. Probability of transitioning to the future hidden state depends solely on the current hidden state i.e. $P(s_{t+1} \mid s_t) = P(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_0)$, and is therefore independent of the past which further satisfies the **Markov property**.

2. After each transition, the model yields an observable state that is dependent only on the current hidden state i.e. $P(o_t \mid s_t) = P(o_t \mid s_t, o_{t-1}, s_{t-1} \ldots, o_0, s_0)$.

The work described by Schreier et al. (2014) presents a framework for safety assessment of on-road situations in a multi agent scenarios with use of Bayesian networks for manoeuvre identification and motion forecasting. At each measured time step a modeled BN yields a probability mass function over a discrete set of manoeuvres for all agents within the environment, which is then used to associate appropriate trajectory prediction model for each of the detected manoeuvres. Thus, enabling the overall framework to use trajectory prediction model that can adapt to the specific situation.

An example where HMMs were used to model probability distribution over a sequence of observations can be found in work by Vasquez et al. (2008). Authors of this work discussed the importance of learning motion patterns that could be associated with agents' destinations in a given environment, thus, allowing HMMs to learn and exploit them for an accurate approximation of future movements. In addition, an unsupervised learning algorithm namely the Growing neural gas (GNG) (Fritzke 1995) was implemented to identify and learn HMMs' states as well as a prior state transition probabilities. This work was further extended by Vasquez et al. (2009) where a variation of HMM specifically growing HMM (GHMM) was proposed. The suggested improvement allowed GHMMs to perform online learning, thereby enabling constant modification and update of states and their transitions with new observations.

Another body of work that examined use of HMM with respect to the task of motion anticipation in urban environments was discussed by Akabane et al. (2017). HMMs were trained to output a sequence of possible future road segments based on the vehicle's current location where a single segment denoted the closest reachable position within 20 meter radius. In addition, the Viterbi algorithm (Forney 1973)

was applied to find the most likely sequence corresponding to the agent's future motion. Despite promising findings, this work requires further advancements as several complex and stochastic factors that occur within urban context were ignored.

Next, Berndt et al. (2008) implemented HMMs as the proposed solution with respect to the issue of ego vehicle's intention inference and recognition. The suggested approach consisted of numerous separate HMMs that were trained to distinguish between lane changing, turning and lane following manoeuvres at an early stage. Classification results presented in the paper revealed promising results with relatively low error rate indicating a sensitivity of 71% and 74% for detecting left and right lane change maneuvers, respectively. Nevertheless, in order for the system to be successfully applied in a broader context e.g. for traffic risk assessment or motion forecasting, further experiments with manoeuvre recognition of not just ego vehicle's intention but rather surrounding agents should have been conducted.

Hülnhagen et al. (2010) investigated and discussed a manoeuvre recognition approach based on probabilistic finite-state machine (PFSM) (Vidal et al. 2005) and Fuzzy logic (Zadeh 1965). More specifically, each of the considered manoeuvres was decomposed into a temporal sequence of basic elements that described it (e.g. braking, driving slowly). Then, those elements were identified using the Fuzzy logic based module, and further modeled with PFSM to represent transition probabilities between them. Lastly, a Bayes filter presented by Thrun (2002) was applied to classify the manoeuvre.

3.3.2 Gaussian Process

In numerous statistical approaches parameters θ of some unknown function f_* are being inferred from the data $\boldsymbol{x}$ such that $P(\theta \mid \boldsymbol{x})$. The Gaussian process (GP) (Rasmussen 2003) on the other hand aims to define a prior distribution directly

over functions $P(f \mid \boldsymbol{x})$ that fit the data and then compute and update its posterior with e.g. Bayesian inference (BI). For a regression task, a GP's prior can be fully define as:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')) \tag{3.10}$$

which further requires a finite, arbitrary set of function's outputs $f(\boldsymbol{x}_i)$ such that $p(f(\boldsymbol{x}_i), \ldots, f(\boldsymbol{x}_n))$ forms a multivariate Gaussian characterised by some mean vector $\boldsymbol{\mu} = m(\boldsymbol{x})$ and covariance matrix $\boldsymbol{\Sigma} = k(\boldsymbol{x}, \boldsymbol{x}')$ which models covariance between each pair in $\boldsymbol{x}$. Thus, $m(\boldsymbol{x})$ and $k(\boldsymbol{x}, \boldsymbol{x}')$ define GP's mean and covariance (kernel) function respectively:

$$m(\boldsymbol{x}) = \mathbb{E}_{x \sim P}[f(\boldsymbol{x})] \tag{3.11}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_{x \sim P}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))^T] \tag{3.12}$$

where $k(\boldsymbol{x}, \boldsymbol{x}')$ must be a positive definite kernel. Since the kernel of GP controls its measure of uncertainty (variance) its choice might vary, one of the often used kernel function is squared exponential kernel (RBF kernel):

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2(-\frac{1}{2\ell^2}(\boldsymbol{x} - \boldsymbol{x}')^2) \tag{3.13}$$

As an example, let $\boldsymbol{f}_*$ denote function outputs of interest given elements of the test set $\boldsymbol{x}_* \in \boldsymbol{X}_*$. Given GP's definition, the joint distribution of the training and the test data is formulated as:

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{K}_{**} \end{bmatrix} \right) \tag{3.14}$$

where $\boldsymbol{\mu}$ represents the mean of the training data and $\boldsymbol{\mu}_*$ the test data. $\boldsymbol{K}$, $\boldsymbol{K}_*$ and $\boldsymbol{K}_{**}$ further denote the covariance matrices of training set, training-test set and test

set respectively. Considering that GP's prior defines joint Gaussian:

$$P(\boldsymbol{f} \mid \boldsymbol{X}) = \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{\mu}, \boldsymbol{K}) \tag{3.15}$$

its posterior distribution for test set, given training set can be further derived with:

$$P(\boldsymbol{f}_* \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{f}) = \mathcal{N}(\boldsymbol{f}_* \mid \boldsymbol{\mu}(\boldsymbol{X}_*) + \boldsymbol{K}_*^T \boldsymbol{K}^{-1}(\boldsymbol{f} - \boldsymbol{\mu}(\boldsymbol{X})), \boldsymbol{K}_{**} - \boldsymbol{K}_*^T \boldsymbol{K}^{-1} \boldsymbol{K}_*) \tag{3.16}$$

The GP based approach was further explained by Joseph et al. (2011) in the context of motion modelling for the future position anticipation. A mixture of Gaussian processes were applied to provide an adaptive representation of motion patterns derived from trajectory data where each pattern was quantified by a combination of location derivatives (i.e. velocities) which further allowed to group similar patterns that were close to one another. In addition, the authors pointed out an important issue in regards to GP where a finite number of patterns needs to be known in advance for the correct modelling. To remedy this issue a Dirichlet process (DP) (Teh 2010) was included to allow adding new behaviours as the data was observed. Hence, forming a distribution over discrete set of distributions of potentially unbounded number of motion patterns with an assumption that a finite number of these patterns is generally followed.

Furthermore, a combination of GP and UKF was used within the work by Tran & Firl (2013) to tackle the issue of modelling and and predicting drivers' intentions as well as their future motion in a complex surrounding, specifically urban intersections. Authors trained a single GP regression model per each of the defined manoeuvres which included driving straight and turning left or right. Then, using the observed data that defines subject's vehicle state over time a manoeuvre model that best fits that state was adopted for further movement prediction. Lastly, the selected model was combined with UKF to account for non-linear prediction of future position in

a multi-step manner. Moreover, improvements of this work were later presented by Tran & Firl (2014) where drawbacks of using UKF were addressed. Firstly, to improve the long term prediction a particle filter algorithm (Thrun 2002) based on given GP regression model was employed with combination of the Monte Carlo sampling for multimodal predictions.

Next, McCall et al. (2007) addressed the recognition of driver's intention to lane change with sparse Bayesian learning methodology (SBL) (Tipping 2001) to learn the mapping between the input vector of features and the set of discrete manoeuvres in a probabilistic fashion. Results indicated an increase in performance when the data describing head motion was incorporated within the prediction model as opposed to sole use of lane tracking and CAN bus data.

Moreover, Morris et al. (2011) developed a framework based on Bayesian variation of support vector machine (SVM) (Cortes & Vapnik 1995) namely relevance vector machine (RVM) (Tipping 2000) to capture and predict driver's future intent. The predictive model was trained specifically to distinguish between an intention of lane changing and lane keeping using features collected from numerous sensors, including head tracking camera to observe driver's head position and orientation. As reported, the RVM based classifier achieved reliable results in a highway scenario where lane change manoeuvre was anticipated for up to three seconds before it was executed.

3.3.3 Summary

The Bayesian framework offers several promising approaches for short-term motion prediction in autonomous vehicles. Bayesian networks model agent maneuvers and adapt predictions accordingly, while hidden Markov models learn motion patterns and destinations, even adapting online. Gaussian processes provide adaptive motion representations and handle multimodal predictions, and combined methods

address specific challenges like long-term prediction and driver intention. However, limitations remain in accuracy, computational demands, and handling complex environments.

3.4 Machine Learning

3.4.1 Classification and Regression

Predictive models provide a way to approximate some arbitrary function $f*$ which aims to map a set of inputs features $\boldsymbol{x}$ to an output variable $y \in \boldsymbol{y}$. In a supervised setting where ground truth labels are present the training of predictive models can be loosely categorised into tasks of classification or regression where the former aims to map $\boldsymbol{x}$ to a discrete output variable y corresponding to a class label e.g. cat or dog (binary classification), and the latter predicts y as a continuous quantity, for instance a price of a property.

Support vector machine (SVM) (Cortes & Vapnik 1995) being a widely used supervised model solves the classification task by finding an optimal decision boundary (hyperplane) in an affine space by maximising margin between closest data points (support vectors) of both classes $y \in \{+1, -1\}$ such that:

$$f(\boldsymbol{x}) = \begin{cases} +1 \text{ if } \boldsymbol{w}^T\boldsymbol{x} + b \geq 0 \\ -1 \text{ if } \boldsymbol{w}^T\boldsymbol{x} + b < 0 \end{cases} \tag{3.17}$$

where $\boldsymbol{w}$ is the weight vector and b is the bias term. Thus, if the given data point $\boldsymbol{x}_i$ lies below or above the hyperplane then its class label is $y_i = -1$ or $y_i = +1$ respectively. Although SVM is generally used for classification, it can also be applied for regression task where the decision boundary is used to find a some continuous value y rather than to separate the data point $\boldsymbol{x}_i$ to class y_i.

In a work by Mandalia & Salvucci (2005) the use of SVM was explored for the task of detecting and classifying lane change behaviour. First of all, the authors examined a set of features that predominantly contribute to a differentiation of the defined classes i.e. lane keeping or lane changing. Then, a SVM was trained with the data collected from highway scenarios in a temporal manner where an input to the model was constructed from a set of features collected over a constant time window.

Another example of SVM classifier was presented by Aoude & How (2009) where the predictive model was trained to distinguish between harmless and dangerous behavior. Additionally, the output of the SVM was further processed with Bayesian filter (BF) (Bishop 2006) to account for the temporal manner of the problem by estimating the probability of the SVM's output in the future. Lastly, the approximated classification score was fed through the empirically defined threshold unit to compute final behavior category. Further advancements of this work were presented by Aoude et al. (2011) where the performance of the SVM-BF algorithm was examined against the HMM to determine their suitability for discussed scenarios.

3.4.2 Clustering

Clustering algorithms allow to analyse the data in an unsupervised fashion where no ground truth labels are present. Furthermore, these methods focus on identifying and grouping together data points that manifest similar features and are therefore close together in a given space. The degree of similarity between data points is determined by a dissimilarity function e.g. Euclidean distance:

$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{\sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{x}'_i)^2} \tag{3.18}$$

In the context of trajectory or manoeuvre prediction clustering can for instance refer to determining motion patterns within the data for further inference. K-Means

(Hartigan & Wong 1979) is an example of a popular clustering algorithm which aims to iteratively group n data points into fixed number of k clusters with nearest mean by minimising the objective function J:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \|x_i^{(j)} - c_j\| \tag{3.19}$$

where $\|x_i^{(j)} - c_j\|$ is the distance between data point $x_i^{(j)}$ and cluster centre c_j. The step-by-step approach to the K-Means works as follows:

1. k random points are randomly selected as cluster centers or centroids $c_1, \ldots, c_k$.
2. Each data point $x_1, \ldots, x_n$ is assigned to the closest centroid c_* based on the distance function.
3. Each centroid $c_1, \ldots, c_k$ is updated by re-computing the average of all data points within given cluster.
4. Step 2 and 3 is repeated until convergence.

K-Means offers a straightforward, yet naive and limited solution for cluster analysis. Its main drawbacks arise from hard clustering in which each data point is assigned to a single group, and from its distance function which is measured from centroids, effectively creating restricted round shaped clusters. Those issues can be addressed through soft probabilistic clustering such as Gaussian mixture model (GMM) (Reynolds 2009), also called mixture of Gaussians (MOG) where each cluster is assumed to form a single multivariate Gaussian distribution, thereby allowing each data point to be assigned to more than one cluster in a probabilistic manner. Considering that GMM is just a linear combination of k Gaussians with some mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ weighted by π it can be further formulated as:

$$P(\boldsymbol{x}_i \mid \theta) = \sum_{j=1}^{k} \pi_j \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \tag{3.20}$$

Vasquez & Fraichard (2004) suggested a statistical approach towards motion prediction based on the clustering technique from earlier work of Hofmann & Buhmann (1997). The primary hypothesis established in this work was that objects' movement within a given environment can be characterised with a certain number of general patterns which further enabled the model to learn those from training data. Then, during estimation stage, a partially observed trajectory was associated with one of k clusters for further estimation of agent's possible future motion.

The work by Veeraraghavan et al. (2006) further investigated how the combination of GMM and UKF can be utilised to predict motion of vehicles at intersection by sampling from the state distribution. Moreover, the data defining motion of agents (discrete position and velocity) was collected using surveillance camera rather than on-board sensors, and further processed with a computer vision tracking algorithm (Veeraraghavanm & Papanikolopoulos 2004). Results presented in numerous figures demonstrate that the proposed methodology, based on Gaussian Mixture Model (GMM) and deterministic sampling, was capable of predicting future motion with significantly lower error compared to the standard Kalman filter.

Next, Wiest et al. (2012) suggested to use GMM to approximate the probability density function that describes possible future motion. The GMM based approach enabled the model to extract and learn motion patterns from agents' trajectories and then based on observed historical data infer a probability distribution defining subsequent movement for up to 2 seconds. In addition, a variant of GMM was implemented i.e. Variational GMM (VGMM) (Nasios & Bors 2006) to overcome the disadvantages imposed by GMM such as overfitting and sensitivity to outliers.

A more recent body of work with regards to clustering of a lane changing and lane keeping behaviour was demonstrated by Augustin et al. (2018). More specifically, the K-means as well as agglomerative hierarchical clustering (AHC) (Tan et al. 2016) were trained to form a representation (prototype) of trajectories with respect

to the defined manoeuvres classes. Results revealed predominance of using AHC as opposed to K-means in a highway environment. Then, with the generative classifier from (Hubert & Van Driessen 2004) a manoeuvre estimation was executed using an input in a form of partially observed motion of an agent as well as clusters' prototypes defined as the mean value of all trajectories within particular cluster. In addition, authors proposed to utilise classification results for the approximation of the vehicle's future motion with the discrete Wiener process acceleration model (Bar-Shalom et al. 2004).

3.4.3 Deep Learning

Lee et al. (2017) introduced a framework for multi-modal vehicle trajectory prediction and presented numerous benefits of integrating an interaction aware module within it. The proposed approached named *DESIRE* was primarily constructed through a combination of a CNN to extract semantic scene context, the conditional variational auto-encoder for multiple plausible future trajectories conditioned on past trajectories, and the RNN to account for encoding and decoding of states through time. In addition, to further assess the quality of predicted hypothetical trajectories, a framework based on inverse optimal control (IOC) (Abbeel & Ng 2004) was employed to learn an unknown reward function for ranking and refining of the decoded future motion.

Furthermore, Choi et al. (2019) also proposed a multi modal motion prediction framework *DROGON* based on causal reasoning of agents' intentions and interactions. Yet again, the CVAE conditioned on agents' possible actions was employed to approximate the posterior distribution for multiple possible trajectory anticipation. In addition, numerous penalty terms were introduced to encourage the model to predict more natural future locations within drivable road boundaries. Results clearly demonstrate a great reduction with respect to prediction error which further

indicate advantages of incorporating intention and interaction aware modules into the network. For example, DROGON managed to reduce the average and final displacement error (ADE/FDE) (Eq. 4.6, 4.7) over 40% for trajectory prediction of up to 4 seconds as compared to the state-of-the-art GatedRN-20 Choi & Dariush (2019).

Luo et al. (2018) approached the discussed task from a perspective of training their proposed model named *Fast and Furious* (FaF) for numerous related tasks such as detection, tracking and motion prediction, in an end-to-end fashion. As typically employed, these task are learned independently, sending their output forward within the pipeline which fundamentally, as authors argue, can greatly limit the propagation of uncertainty produced by each module. FaF on the other hand is trained end-to-end allowing it to learn dependencies of each task and jointly reason about future predictions. The model uses a 3D point cloud data to construct a sparse bird-eye-view of the surrounding over observed number of timesteps which is further processed by numerous convolutional layers that output future bounding boxes of tracked agent. In terms of motion prediction, results demonstrate a satisfactory performance being able to predict 10 frames of motion into the future with L2 being less than 0.33 meter. However, the model was not tested on any publicly available dataset but rather on an in-house dataset, and in addition no comparison with any of the recent approaches (at the time of writing) was made.

Chandra, Bhattacharya, Bera & Manocha (2019) presented a deep learning based model (hybrid LSTM-CNN) named *TraPHic* that addresses the issue of motion forecasting in dense traffic whilst simultaneously accounting for interactions between numerous heterogeneous on-road participants (e.g. bikes, buses). TraPHic captures relationships between traffic agents by defining a fixed horizon of interest (semi-elliptical region) using an empirically determined radius as opposed to the conventionally employed grid-based methodology (Deo & Trivedi 2018*a*). The

states of observed agents are then processed through a LSTM layer to account for the temporal dependencies, and further encoded through a CNN layer to learn local on-road interactions. Further extension of this work was presented by Chandra, Bhattacharya, Roncal, Bera & Manocha (2019) where instead of using manually annotated data to train the proposed model, all trajectories were obtained from RGB cameras and tracking algorithm.

The following work presented by Deo & Trivedi (2018*b*) proposes a LSTM network based on encoder-decoder architecture to tackle the issue of motion prediction on a freeway. The main contribution of the work comes from training the model in a fashion that can present interaction aware capabilities between multiple agents, and by conditioning the future trajectory prediction of the vehicle of interest on six different manoeuvre classes. The network is constructed such that it takes the past motion of all agents within the scene as its input and then predicts manoeuvre class of the vehicle of interest which allows to further anticipate the distribution of possible future motions with respect to the anticipated class. Despite plausible results it is quite evident that the network computes its predictions solely on naive past states of observed agents (past motion coordinates) which could be further improved by including numerous other relevant data points such as their velocities, yaw angles etc, potentially leading to an increase of performance as well as an ability to being examined in more complex scenarios rather than just on freeway.

In their work Djuric et al. (2020) argued that in order to learn essential features of the environment, one must transform its context into a meaningful representation that will contain information with respect to road layout as well as all agents within the scene. Their work proposes to model the static scene context by rasterising it into a top-down view HD map data that describes the road's drivable surface, lanes, crosswalks, agents' movement through time etc. The rasterised static context of the scene is then used as an input to the CNN layer which extracts and learns salient

features of the road. The encoded road features are further combined with the agent's state and passed through a subsequent MLP in order to predict a plausible future motion along with the standard deviation that accounts for the uncertainty of predictions per trajectory point. Further extension of this work was demonstrated by Cui et al. (2019) where yet again a rasterised image of the surrounding was utilised for the prediction of agents' future motion. However, in comparison to the previous study where a single trajectory for the agent of interest was anticipated, this work extended the prior model and tackled the issue of reasoning about multi-modal nature of the task by forecasting number of candidate future trajectories along with the estimate of the probabilities. In order to assign the best matching probability to each of the mode (trajectory), authors suggested to first predict an arbitrary number of future motions which can then be used to select the closest matching mode with respect to the ground truth motion, and then to employ a novel multiple-trajectory prediction (MTP) loss (Rupprecht et al. 2017) to force the probability of the best selected mode to be as close to 1 as possible.

Next, Marchetti et al. (2020) demonstrated an interesting approach by utilising the memory augmented network (Weston et al. 2014) to learn the association between past and future motion and memorise most meaningful samples. The proposed model was constructed in a encoder-decoder manner with two encoding functions for learning past and future trajectories, and a decoder for reconstruction of future motion from latent representation of encoders' outputs. Moreover, the network presented ability of updating its internal representation of motion samples in an online fashion, thus allowing for continuous improvement as new samples are collected. Despite the fact that various aspects such as interactions between agents were not modelled, the network managed to outperform several previously proposed approaches e.g. DESIRE by over 25% on a 4 seconds prediction horizon.

One of the main issues with regards to motion prediction that the current approaches

manifest is poor generalisation capabilities with respect to e.g. unknown or never previously seen datasets. The work by Srikanth et al. (2019) tackles this very issue by constructing an intermediate representation of feature space from numerous bird's eye view maps. Each map is generated from stereo and LiDAR data to form various semantic representations of the scene such as lanes, roads, obstacles etc. Next, the generated intermediate representation is fed through a CNN-LSTM based network to predict location of the vehicle of interest on an occupancy grid map.

On the other hand Gao et al. (2020) investigated a novel approach of transforming input features such as lanes, agents' trajectories, crosswalks into a vectorised form and incorporating those into a model based on graph neural network (GNN) (Battaglia et al. 2018). The structure of GNN was divided into local (MLP) and global (self-attention (Vaswani et al. 2017)) graphs in order to allow the model to capture data from both individual polylines as well as from its aggregated global representation. The performance of proposed GNN was further compared to a CNN based on ResNet-18 architecture. Experimental results indicate that the proposed GNN required over 70% less parameters as compared to ResNet-18, which ultimately lead to significant boost in performance (18%) and reduced computational requirements whilst achieving state-of-the-art performance for predictions of up to 3 seconds on the publicly available Argoverse dataset (Chang et al. 2019).

Another use of the GNN for the task of the trajectory prediction was introduced by Lee et al. (2019) where the main task was divided into two sub-tasks i.e. modelling joint interactions between a pair of agents, and combining predictions from subsequent task with agents' past states to reason about their future motion. The authors assumed that there exists a set of discrete interactions between a pair of agents e.g. yielding to another driver, that can be learned from labeled data, thus allowing the GNN based model whose nodes and edges represented agents of interest as well as their long-term intents respectively to learn those in a supervised fashion.

Next, Messaoud et al. (2020) emphasized the importance of inferring trajectory cues from both scene structure and past motion data of interacting agents, and proposed a model based on multi-head attention (Vaswani et al. 2017) for the future motion anticipation. The discussed work suggests to encode and concatenate a joint representation of all agents of interests as well as a top-down view of static scene through an LSTM encoder and a pre-trained CNN (Krizhevsky et al. 2012) respectively. Furthermore, the encoded context can be divided into a spatial grid that is used to model future trajectory distribution as a mixture model where each attention head of the multi-headed mechanism corresponds to a single mixture component. Thus, allowing each attention head to focus on a distinct parts of the encoded scene resulting in a diverse and more accurate multi-modal future predictions.

Vast majority of motion forecasting approaches focus on formulating the discussed problem as a regression task. Recently proposed method called *CoverNet* (Phan-Minh et al. 2020) proposes to frame this as a classification task over a set of generated trajectories. The main motivation arises from an assumption that given the current state of an agent within an environment, there exist a finite amount of relatively reasonable actions that an agent can execute within a short period of the future time horizon (e.g. 6 seconds). First, the set of plausible trajectories is generated using either fixed approach where a subset of future motions is sub-sampled from the training set without considering the observed agent's state, or using the dynamic approach where trajectories are generated based on the currently observed state of the agent with the dynamic model. Next, the scene context (top-down view of the surrounding) is encoded with a pre-trained ResNet-50 and further concatenated with an agent's current motion state. Finally, the concatenated encoding is fed to prediction layer which outputs probabilities over the fixed set of trajectories. Results of experiments conducted on public urban self-driving datasets demonstrated that CoverNet can outperform previously proposed methods by over 30% on a 6 seconds prediction horizon, whilst ensuring that a plausible set of possible trajectories is

being covered.

The use of deep learning based methods within the area of motion forecasting has not been explored solely from the point of view of autonomous vehicles. A large body of work has also examined applications of these models in the area of human trajectory prediction with a great emphasis on modeling interactions between e.g. pedestrians in crowded spaces. An example of such work was presented by Alahi et al. (2016) where a LSTM based network (*Social LSTM*) was applied to implicitly model complex interactions between people, thus allowing to account for numerous non-linear behaviors which in essence lead to a more intelligent and robust prediction of agents' future paths. Most importantly, authors addressed LSTM's inability to capture spatial dependencies between multiple sequences through a novel pooling layer refereed to as *social pooling*. The social pooling mechanism uses a grid-based approach where encoded LSTM's hidden states corresponding to an agent of interest as well as its nearby neighbours are placed inside a tensor such that their spatial relationship can be partially preserved. The *social* tensor as well as agent's coordinates over the observed time-steps are further encoded and then finally decoded to output an estimate of future positions.

An extension of Social LSTM, namely Social GAN (S-GAN) was further presented by Gupta et al. (2018) where a pooling mechanism was replaced with a MLP and max pooling operation resulting in a decreased computational costs as well as an increase of overall performance. Moreover, the authors demonstrated the use of generative adversarial networks (GANs) (Goodfellow et al. 2014) to account for the multi modal nature of the task. In addition, a novel *variety loss* was introduced to encourage the network to learn a wider distribution of the possible trajectories with respect to agent's observed past data whilst being consistent in producing paths that are socially plausible. Formerly discussed approaches introduced numerous interesting ideas with respect to modelling of social constrains, however, when navigating

through complex and stochastic environments it is crucial to consider not only social but also physical limitations imposed by the surrounding. *SoPhie* (Sadeghian et al. 2019) addresses both of these limitations by employing a soft attention mechanism (Xu et al. 2015) for both physical features extracted from image representation of the scene, as well as social features i.e. hidden states of observed agents and their joint representation.

On the other hand the following work by Nikhil & Tran Morris (2018) also focuses on predicting the motion of pedestrians, however, instead of using a RNN based network to process the sequential data it utilises a convolutional layers to encode and predict future coordinates. Given only the past coordinates of observed agent the proposed CNN based model is able to achieve a competitive results with some of the state-of-the-art models e.g. 0.59/1.22 vs 0.61/1.121 ADE/FDE against S-GAN-P, whilst dramatically decreasing the inference time per frame from 0.067 seconds (S-GAN-P) to 0.002 seconds.

More recent approach within the area of pedestrian motion prediction presented by Giuliari et al. (2020) explored a novel use of transformer networks (Vaswani et al. 2017) to model temporal dependencies between past and future states. During recent years transformer networks demonstrated a superior performance on a diverse set of language tasks (e.g. text generation, sentiment analysis) (Radford et al. 2018, 2019, Brown et al. 2020) significantly outperforming LSTM based models due to their improved ability of handling long-term dependencies as well as the use of attention mechanisms which further forces the model to focus on the salient parts of e.g. given sentence. In the discussed work the authors examined vanilla transformer as well bidirectional transformer (BERT) (Devlin et al. 2018) against numerous approaches on the TrajNet benchmark (Sadeghian et al. 2018). Despite the inability of modelling social interactions in crowded environment between pedestrians, the proposed transformer model has been able to achieve competitive performance against

other approaches whilst demonstrating greater capabilities of predicting multiple future motions further into the future.

The idea of pooling mechanism for the purpose of modelling the context of on-road social interactions was also extended within the domain of AVs. For instance the following work by Deo & Trivedi (2018*a*) proposes a use of pooling layer based on convolutional layer as well as max-pooling operation as opposed to the methodologies presented in similar prior work by Alahi et al. (2016) and Gupta et al. (2018). More specifically, states of all observed agents in the scene are encoded with an LSTM encoder and placed within a tensor (social tensor) to preserve spatial relationship. Then, instead of using an MLP for further encoding, a convolutional layer and the max-pooling operation are applied on top of the obtained tensor to extract local salient features corresponding to interactions between agents. Lastly, based on the social encoding as well as encoded state of the vehicle of interest a distribution of possible future motions can be predicted with respect to six different discrete manoeuvre classes.

A similar approach was also demonstrated by Zhao et al. (2019) where apart from considering an encoded context of on-road social interaction a spatial features of the road were also encoded to account for physical context of the surrounding. Additionally, a conditional GAN (Mirza & Osindero 2014) dependent on the scene context and past trajectories of all agents was trained to account for multi-modality of the task.

Another example of utilising the pooling mechanism for modelling of on-road social interactions was presented by Messaoud et al. (2019) with an extension of attention mechanism. In their work, Messaoud *et al.* used a grid-like approach to preserve spatial relationship between target vehicle and surrounding agents which further enabled the model to learn local dependencies through the use of convolutional layers with depthwise kernel as demonstrated by Chollet (2017). Furthermore, considering

that employing convolutional layers only enables the model to capture local interactions between nearby vehicles, a multi-headed attention was implemented in order to ensure that a weighted non-local encoding of the relationship between agents can also be obtained. Despite promising results, the proposed model was examined solely on a highway datasets and it is therefore arguable whether its effectiveness would be persistent within a more challenging environments.

More recent approach where social pooling was employed was presented by Zhang et al. (2020). The proposed model also follows the methodology of placing tracked agents within a grid map to retain spatial structure, however, in comparison to previous studies where e.g. a convolutional operation with max pooling operation is used to create a so called Social tensor, authors propose to instead use dilated convolutional operation (Yu & Koltun 2015) to reduce loss of data caused by the pooling operation. Additionally, a stacked sparse auto-encoders (Du et al. 2016) were introduced to extract and encode salient features that correspond to agents' states. Nevertheless, the described methodology was only examined within a highway scenario and was not compared against any of the recently published methods for further validation of its performance.

3.4.4 Summary

In the realm of short-term motion prediction, diverse methodologies have been explored, spanning classical machine learning and cutting-edge deep learning approaches allowing to predict accurate motion of agents for up to 6 seconds into the future. Classification and regression models, like Support Vector Machines, exhibit effectiveness in discerning behaviors, while clustering algorithms, including K-Means and Gaussian Mixture Models, uncover motion patterns through unsupervised analysis. The surge in deep learning, witnessed in models like DESIRE, DROGON, and FaF, showcases the superior capacity of CNNs, RRNs, and Transformers in captur-

ing intricate interactions and temporal dependencies. Innovations like CoverNet and TraPHic introduce trajectory classification and hybrid architectures, while SoPhie and transformer-based models extend their prowess to human trajectory prediction. Challenges persist in adapting these methods to diverse real-world scenarios, warranting ongoing research for robust, adaptable solutions in short-term motion prediction for autonomous vehicles.

3.5 Conclusion

This chapter focused on reviewing the literature in the field of motion prediction, primarily from the perspective of automotive vehicles. The importance of the task that is trajectory forecasting has been thoroughly outlined and discussed. First of all, section 3.1 examined more classical methodologies based on the kinematic and dynamic physical motion models that often combine additional filtering algorithms such as Kalman filter to account for the noise in the data. And although those methods often provide a straightforward set of solutions they generally require several assumptions e.g. a constant lateral motion to remain fixed during the prediction horizon and can therefore be only applied within environments that manifest a low level of complexity. In addition, the accuracy of physic's based models diminishes rapidly for predictions that are non-linear and are longer than e.g. 1 second as they rely solely on a low level properties of vehicle's motion, further ignoring and not considering numerous external factors such as road topology, on-road interactions, manoeuvres etc.

Next, section 3.2 and 3.3 explored probabilistic methodologies such as Monte Carlo sampling as well as methods that are based on the Bayesian framework, for example Hidden Markov models. The discussed techniques demonstrated a great number of definite advantages over both kinematic and dynamic models by for instance allow-

ing to capture conditional relationships between variables, or by enabling to obtain an approximate estimation of desired motion using a large number of samples. Nevertheless, in spite of numerous benefits, there are still various critical limitations that the probabilistic based approaches manifest including short prediction horizon, inability to accurately model multiple plausible motions, low capacity of capturing the temporal dependency of future movement based on the past actions of the tracked vehicle, e.g. HMM must satisfy Markov property and can thus compute motion that depends exclusively on the current state.

Lastly, in section 3.4 an extensive review of recent machine learning techniques has been conducted. First, the more classical ML algorithms such as SVM, K-Means, GMM and others were examined, and despite the significant improvements that those methods offer over previously discussed techniques that rely on e.g. physical models there is still a number of substantial disadvantages that prevents them from achieving low prediction error in complex environments. For example, as with previously discussed approaches the problem of a accurate anticipation for the longer horizon than e.g. 1-2 seconds persists with the dramatic accuracy decrease as the future prediction horizon increases. Moreover, section 3.4.3 covered a broad review of most recent and advanced trajectory prediction methods based on deep learning algorithms that are currently achieving state-of-the-art results on the motion forecasting task. As discussed in section 2.6 in recent years tasks that involve learning from data has been primarily approached from the perspective of utilising DL based algorithms mainly due to their ability to process raw, large amounts of data which then allows those methods to automatically learn relevant abstract representation required for the task at hand without the need of manual feature engineering. Yet, even recently proposed methods present several limitations with respect to the anticipation of agents' future motion, especially in a more advanced surroundings such as urban environments where a large space of possible non-linear options must be considered. For example, all recent DL methods that employ some form of HD

map data (generally in a form of top-down rasterised 2D images) focus on a large portion of the surrounding to encode the global context of the surrounding which might be sub-optimal considering that a smaller chunks of the map that capture local surrounding can be used to obtain a more robust encoding of the tracked agent. Additionally, the standard technique of encoding those maps involves employing one of the popular CNN architectures e.g. ResNet-50 (He et al. 2016) with large number of parameters that were initially pre-trained on a distinct task as well as dataset. Generally, this method presents satisfactory results. However, it has been recently demonstrated that for tasks that heavily depend on localization of salient features it might be more beneficial to train the model from scratch directly on the task at hand. As a result, the next chapter will explore an alternative way of using map data to encode the captured information about the tracked agent in a form of local maps. In addition, the subsequent chapters will investigate and demonstrate a variant of the feature extractor that is based on a novel neural network architecture which will aim to minimize the model's complexity and alleviate some of the drawbacks that standard CNN models exhibit.

Chapter 4

Investigating Local Maps for Short-term Motion Prediction

This chapter will explore the usage of local geometrical maps and will aim to establish a set of baseline models for the subsequent parts of the thesis in the domain of vehicle motion prediction. First, an overview of various public autonomous driving datasets will be presented in order to determine their suitability for further experiments. Next, the problem formulation as well as notation within this chapter will be outlined. Furthermore, the concept of local geometrical maps as well as the process of creating them will be introduced. Finally, a set of experiments and results will be presented to demonstrate the performance of baseline models as well as advantages of using local maps.

4.1 Introduction

The majority of recent approaches within the area of motion prediction has primarily focused on exploiting the capabilities of deep learning models to learn complex representations that would allow for an accurate anticipation of the tracked vehicle's

short-term movement. As presented in the previous chapter, one of the popular techniques that has been frequently employed to improve robustness and accuracy of the predicted trajectories is to incorporate the global context of the environment in the form of e.g. high definition maps (see Figure: 4.1) which encodes various useful properties of the road such as lanes, walkways, other road participants and so forth. Generally, a predominant manner in which those HD maps are utilised involves first transforming them into a rasterised top-down image of the surrounding, and then using this spatial representation of the road as an auxiliary data for the DL model during both training and inference as presented for example in the work by Phan-Minh et al. (2020) or Cui et al. (2019). Moreover, the usual approach when using the rasterised map is to extract a large portion of the map (e.g. a portion that encompasses 60×60 meters) with respect to agent's position to allow the network to detect and encode salient features as well as enable the agent to 'see' what lies in front of it. Whilst this approach demonstrated plausible results it is still unclear whether indeed a large portion of the environment must be observable by the model to make an accurate short-term prediction of the tracked agent's motion. In contrast, this chapter will focus on exploring and creating a more representative encoding with respect to a single agent by exploiting its spatial representation within an environment through a semantic, hierarchical view of a local chunk (e.g. 20×20 meters) of a rasterised HD map.

4.2 General Setup

4.2.1 Dataset and Benchmark

An accurate prediction of agent's future motion is greatly influenced by numerous observable factors, primarily, as suggested by the conducted literature review, by its past motion data such velocity, yaw angle etc. In addition, a vast majority of recent

motion prediction techniques also demonstrated several advantages of incorporating some form of map data which introduces information about the environment's global context into the processing pipeline. For instance, using a previously considered

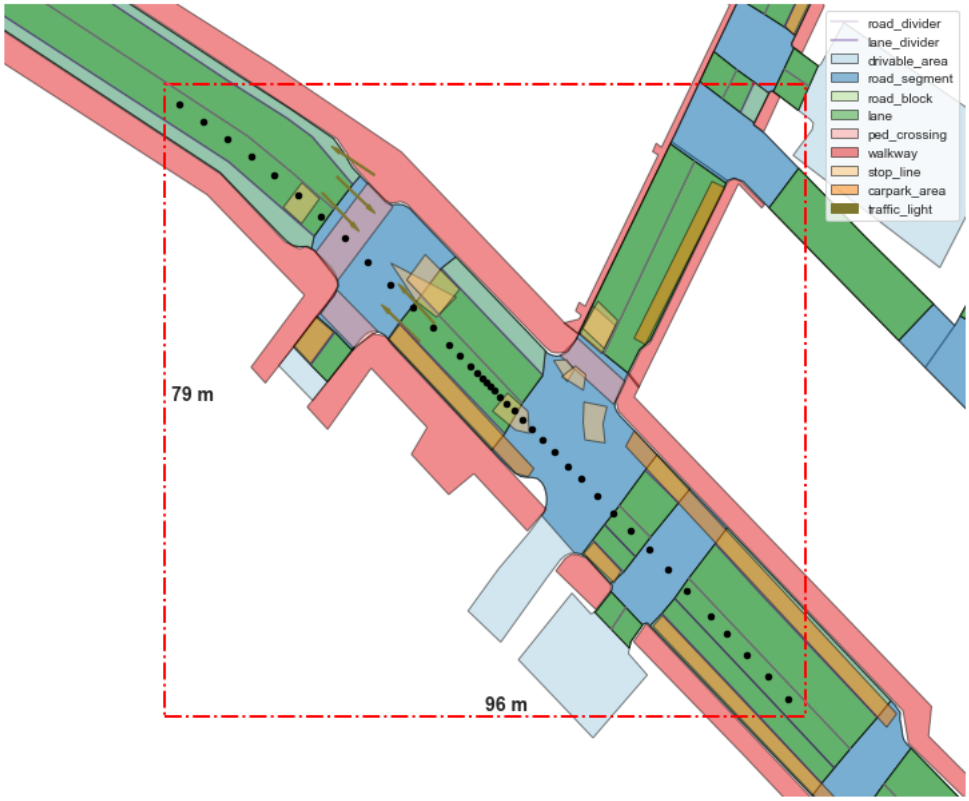

Figure 4.1: An example of a rasterised top-down view of the HD map from nuScenes dataset (Caesar et al. 2020) where a temporal motion of an ego vehicle has been represented with black dots, and a number of different road layers has been specifically colour coded.

rasterised top-down view of HD maps with encoded road attributes where each attribute is specifically colour coded as demonstrated in Figure 4.1 should enable a CNN based feature extractor to learn the relevant road features with respect to the spatial context of the scene as previously presented by Choi et al. (2019) and Marchetti et al. (2020). It is therefore reasonable to explore and primarily consider datasets that provide the access to such data for subsequent experiments. Table 4.1 provides a comparison summary between various popular publicly available autonomous driving datasets.

Dataset	Locations	Images	Anns.	Map Type	Map Layers
KITTI (Geiger et al. 2012)	Karlsruhe	15k	15k	None	0
Cityscapes (Cordts et al. 2016)	Germany (50 cities)	25k	25k	None	0
ApolloScape (Huang et al. 2019)	China (4 regions)	144k	144k	None	0
H3D (Patil et al. 2019)	San Francisco	83k	27k	None	0
A2D2 (Geyer et al. 2020)	Germany (3 cities)	392.5k	12.5k	None	0
A*3D (Pham et al. 2020)	Singapore	39k	19k	None	0
Waymo (Sun et al. 2020)	USA (3 cities)	1m	**200k**	None	0
Argoverse (Chang et al. 2019)	Miami Pittsburgh	490k	22k	**Raster, Vector**	3
nuScenes (Caesar et al. 2020)	Boston Singapore	**1.4m**	**40k**	**Raster**	**11**

Table 4.1: An overview of various public autonomous driving datasets that are often used within the research community. Note that only two datasets (where map type is not none) provide an access to some sort of map data.

As presented in the table 4.1 despite the existence of numerous publicly available autonomous driving datasets there exists a limited number of them that provides an access not only to annotations related to agents' past movement but also to a map data which offers a contextual knowledge of the surrounding and a strong prior for further scene understanding. The two main candidates for experiments are Argoverse and nuScenes. The Argoverse datasets contains the lowest amount of annotations (22k) out of those potential sets, and only provides an access to two map layers (driveable area and ground height) as well to an additional map in a vectorised form where semantic road data is represented as a localized graph as

opposed to the raster counterpart. Moreover, it's main limitation arises from the fact that each recorded sequence of a trajectory is only 5 seconds long meaning that a predictive model is restricted to a prediction horizon that is shorter than 5 seconds (considering at least a single time step of observed data as a model's input).

On the other hand, nuScenes contains a large number of annotated frames (around 40k) with each recorded scene being at least 20 seconds long. In addition, the dataset provides the access to rasterised top-down semantic maps with 11 different types of map layers and it includes scenes that were recorded in Boston as well as Singapore which provides the access to both left as well as right hand traffic. Moreover, nuScenes offers the full availability of its development kit which enables the easy access to various properties of the annotated samples such as the positions, rotation and velocity and many more including semantic maps that can be rendered with different combinations of semantic layers.

The nuScenes dataset extends its value beyond the provision of high-quality data. It also offers a comprehensive toolkit that facilitates effective experimentation and evaluation of models. This toolkit includes baseline physics-based models, providing researchers with a foundational starting point for benchmarking performance. Additionally, it incorporates state-of-the-art model implementations drawn from the literature, enabling easy comparative analysis and contextualizing the performance of novel models. The availability of well-established evaluation metrics within the toolkit ensures consistency and reliability in the evaluation process. Moreover, nuScenes maintains a leaderboard for the relevant task, promoting direct comparison of model performance and ultimately fostering innovation and progress in the field.

In conclusion, nuScenes exhibits several distinct advantages over comparable datasets, justifying its selection as the primary source of experimental data. This dataset, along with its accompanying toolkit, will be used for both the training of novel mod-

els and their benchmarking against established models from the literature throughout this thesis and its subsequent technical chapters. The availability of well-defined, standardized metrics within nuScenes further reinforces its suitability for reliable and objective performance evaluation.

4.2.2 Problem Formulation and Notation

First of all, it is assumed that the vehicle equipped with the proposed method is also fitted with an appropriate tracking sensor that is capable of producing data corresponding to the state of a tracked agent at some fixed interval e.g. 2Hz. Additionally, it is assumed that the vehicle is capable of accessing the HD map data that defines the following road layers; road segments, drivable areas, lanes and walkways. Next, the goal of the trained model is to predict a sequence of the agent's future position differences for τ time-steps into the future with respect to its last observed position at time t. The matrix containing a set of future predictions is further denoted as $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_{t+1}, \hat{\boldsymbol{y}}_{t+2}, \cdots, \hat{\boldsymbol{y}}_{t+\tau}]$ where $\hat{\boldsymbol{y}}_{t+1}$ corresponds to a vector containing the difference in position at time $t+1$ such that $\hat{\boldsymbol{y}}_{t+1} = (\Delta\hat{x}_{t+1}, \Delta\hat{y}_{t+1})$. Furthermore, let $\boldsymbol{Y} = [\boldsymbol{y}_{t+1}, \boldsymbol{y}_{t+2}, \cdots, \boldsymbol{y}_{t+\tau}]$ encapsulate the ground-truth future position differences and let $\boldsymbol{S} = [\boldsymbol{s}_{t-\rho}, \cdots, \boldsymbol{s}_{t-1}, \boldsymbol{s}_t]$ represent a matrix of the tracked agent's motion state from time-step $t-\rho$ to the initial time-step t where ρ represents the observed time horizon and where:

$$\boldsymbol{s}_t = [\boldsymbol{v}, \boldsymbol{a}, \Delta\vartheta] \tag{4.1}$$

denotes a vector of features containing velocity (x, y) and acceleration (x, y) vectors as well as a scalar value for the heading change rate respectively. Moreover, a normalised tensor $\mathbf{D} = [\mathbf{L}_{t-\rho}, \cdots, \mathbf{L}_{t-1}, \mathbf{L}_t]$ is defined which encapsulates the spatial data that represents a set of rasterised top-down views of a local surrounding with

respect to agent's position over the entire observed time horizon. Then, each $\mathbf{L} \in \mathbf{D}$ is unraveled into separate semantic layers such that $\mathbf{L}_t = [\boldsymbol{L}_{t,0},\ \boldsymbol{L}_{t,1}, \cdots, \boldsymbol{L}_{t,n}]$ where n is equal to number of road layers e.g. drivable areas, and $\boldsymbol{L}_{t,i}$ is equal to a sparse matrix that encapsulates spatial data of semantic layer of type i (see Figure: 4.2), note that the process of constructing semantic layers will be explained in details in the next subsection. Finally, both state matrix $\boldsymbol{S}$ as well as the ground truth matrix $\boldsymbol{Y}$ are standardized so that each independent feature has a mean of 0 and standard deviation of 1, and the image data, in this case the tensor $\mathbf{D}$, is normalized so that values are within the range of 0-1.

4.3 Local Semantic Layers

The following summarises the process of creating disentangled map chunks as depicted in Fig. 4.2.

First, let $\{(x_t, y_t) \mid \boldsymbol{p}_t \in \boldsymbol{P}\}$ denote a set of tracked agent's observed coordinates from time t to $t - \rho$ where each $\boldsymbol{p}_t \in \boldsymbol{P}$ is used to define the origin of extraction for a map chunk $\mathbf{L}_t$ at time t. Moreover, a single map chunk $\mathbf{L}_t$ is defined by n sparse matrices (all pixels but the ones corresponding to the given layer are set to zero) $[\boldsymbol{L}_{t,0},\ \boldsymbol{L}_{t,1}, \cdots, \boldsymbol{L}_{t,n}]$ where the $\boldsymbol{L}_{t,i}$ corresponds to a semantic layer of type i. The following types of layers are considered within this work:

$$\mathsf{S} = \{road\_segments, drivable\_area, lane, walkway\} \tag{4.2}$$

Thus, each local semantic layer with respect to $\boldsymbol{p}_t$ can be further extracted by:

$$\boldsymbol{L}_{t,i} = \Phi_{\text{get\_l}}(\boldsymbol{p}_t, \lambda, \mathsf{s}), \quad \forall\ \mathsf{s} \in \mathsf{S} \tag{4.3}$$

where λ defines the extraction offset in meters with a default resolution of 3 pixels

per meter. The local chunk of the layer of interest is then extracted from $x_t - 3\lambda$ to $x_t + 3\lambda$ in a horizontal direction, and from $y_t - 3\lambda$ to $y_t + 3\lambda$ in a vertical direction, hence, the final size of the $\boldsymbol{L}_{t,i}$ is $2\lambda \times 2\lambda$ meters. In addition, an extra layer that portrays the agent is created by rendering it at its origin defined by $\boldsymbol{p}_t$, with its initial orientation facing up. Next, a rotation angle $\hat{\theta}$ with respect to the agent's yaw angle θ is computed as:

$$\hat{\theta} = ((\pi/2) + \text{sign}(-\theta) \cdot |\theta|) \cdot 180/\pi \tag{4.4}$$

which defines the rotation angle by which the agent is rotated so that its orientation is aligned with its global heading direction. Lastly, the extracted images (layers) are transformed from RGB to grayscale, and the up-scaled so that the final size of each layer is equal to 64×64 pixels.

4.4 Experiments and Results

This section will focus on establishing and evaluating a set of deterministic models that will be used as baseline models for further experiments.

4.4.1 Experimental Setup

Results of subsequent experiments are obtained by training and evaluating models within the subsequent subsections on the nuScenes dataset (see Section: 4.2.1) which provides the access to 1000 scenes, each approximately 20 seconds long that were collected in two different cities with both left and right hand traffic. The collected scenes provide a wide diversity with regards to weather conditions, traffic situations as well as traffic density. In addition, scenes and objects (e.g. vehicles, pedestrians) were accurately annotated at the rate of $2Hz$ and modeled as cuboid, providing

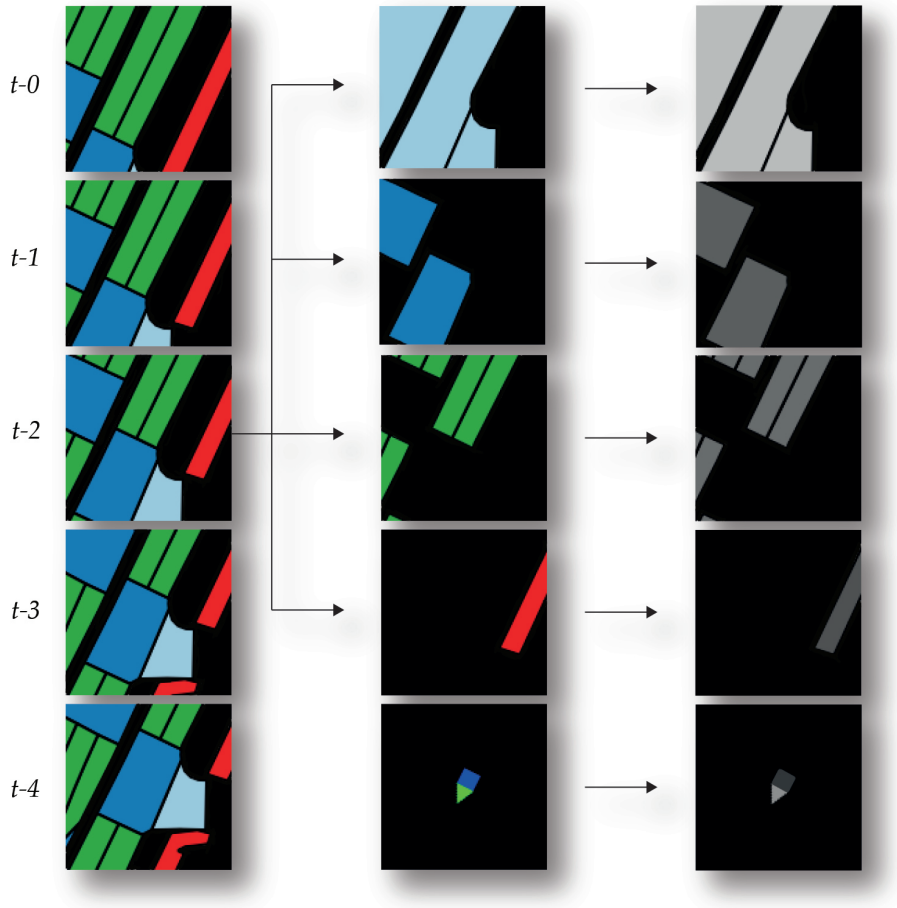

Figure 4.2: Disentanglement process of a map chunk at time $t-2$. The map is turned into several geometrical layers each representing a single part of the whole map. In addition, an agent is rendered (bottom middle and right images) with its origin corresponding to the position on the chunk of interest.

further access to the object's position, velocity and yaw angle (the acceleration is further derived with the use of velocity). The dataset is further split in accordance to nuScene's *train* and *val* split sets [1]. Since the *test* set has not been annotated the *train* set is split into train (80%) and validation (20%) sets and the *val* set is used as a test set. For every data sample the observed time horizon is set to 5 time-steps which corresponds to 2 seconds of agent's past state, and the predictions are made for every $t \in \{1, 2, 3, 4, 5, 6\}$ seconds of the future time horizon. During the initial experiment the size of a local chunk of a map is set to $\lambda = 10$ meters. Furthermore, each model is trained for 100 epochs with Adam optimizer (Kingma &

[1] Available on the official repository

Ba 2014) with a constant learning rate of $5e-4$ and a batch size of 128. All models are implemented with PyTorch (Paszke et al. 2019) and trained as well as tested on a single Nvidia RTX 2080Ti. To account for the fact that models are subject to random variation in training, the final results are computed as the mean and standard deviation (std) over 5 independent training runs. Networks are optimised by minimising the mean squared error loss function between predicted ground truth and predicted trajectory:

$$\mathcal{J} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2 \tag{4.5}$$

The quantitative results are reported by employing the standard metrics from the literature for the task of motion predictions that measure the difference between ground truth and predictions in meters i.e. average displacement error (ADE) between all predictions of sample i:

$$\text{ADE} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2} \tag{4.6}$$

and final displacement error (FDE) between sample's final prediction at time $t+\tau$:

$$\text{FDE} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(\boldsymbol{y}_{i,t+\tau} - \hat{\boldsymbol{y}}_{i,t+\tau})^2} \tag{4.7}$$

4.4.2 Establishing Baseline Models

First of all, two classic physics-based models will be introduced to serve as a baseline methods against other DL models introduced within this subsection. First model will be based on an assumption that vehicle's velocity as well as the yaw angle remain constant throughout the prediction horizon (constant velocity and yaw), second model on the other hand will compute the minimum average point-wise Euclidean distance over four different physics-based models; constant velocity and

yaw, constant velocity and yaw rate, constant acceleration and yaw, and finally a constant acceleration and yaw rate. Next, as defined in section 4.2.2 both the input as well as the output data are of sequential type, and thus it is reasonable to establish at least one model that can process this type of data e.g. a RNN based network. However, in order to measure the impact that a sequence of observed features has on the predicted outputs, a model based on the feed-forward network that will be trained on features observed at the last time-step t will also be examined. Lastly, since the aim of this chapter is to examine the impact of semantic maps, a trivial model based on a CNN architecture will also be evaluated against a CNN model that uses global maps as an input. The following lists the initial set of models considered during this experiment:

1. **Cons. Vel. & Head.** - A dynamic physics-based model that assumes that the agent's velocity as well as heading angle remain constant throughout the duration of prediction horizon.

2. **Physics Oralce** - An extended version of basic physics-based models which computes future predictions using numerous different constant physics-based models and which chooses the final output to be the one with the minimum $L2$ distance with respect to the ground truth data.

3. **FCSingleTS** - A simple neural network based solely on three fully-connected layers with hidden size of 128, 64 and 2τ respectively, and with a ReLU non-linearity (Nair & Hinton 2010) in-between whose input consists of the agent's state at the last observed time-step t i.e. $\boldsymbol{s}_t$.

4. **FCMultiTS** - An extension of FCSingleTS model whose input not only considers the last observable state of the agent but the entire sequence $\boldsymbol{S}$.

5. **LSTM** - An RNN model based on a LSTM unit with a hidden dimension of 128 units for the purpose of examining whether a model that can naturally deal

with sequence data outperforms a simple network based on fully-connected layers.

6. **GRU** - A more computationally efficient RNN model based on a GRU unit with a hidden dimension of 128 units for comparison purposes against LSTM model.

7. **CNN**$_L$ - A variant of a basic CNN based model that is built of a fully-connected layer with 128 units as well as three convolutional layers whose output channels are equal to 16, 32 and 32 respectively. The default kernel size of 3×3, a stride of 1 and a *same* padding mode is adopted for each convolutional layer. The fully-connected layer processes the flattened matrix $\boldsymbol{S}$ in order to encapsulate the knowledge of the entire observed history, whereas convolutional layers encode the information from the local semantic layers (with a default size of 20×20 meters). For the basic baseline experiments the semantic layers are not split into separate geometrical layers, and instead of using a sequence of local maps a single map at the last observable timestep t is used as an input to the network which is further depicted in the first column of Figure 4.2 at $t-0$. Finally, the output of fully-connected layer as well as the convolutional layers is concatenated and processed by the final fully-connected layer whose output is a predicted trajectory $\hat{\boldsymbol{Y}}$.

8. **CNN**$_G$ - A second variant of the CNN model based on CNN$_L$ which uses the same architecture, however, instead of using local maps as an input, it uses global rasterised chunk of a map (60×60 meters) centered at the agent's location.

The following presents results of the baseline models trained on the nuScene dataset for up to 6 seconds of the future prediction horizon. First, Tables 4.2 and 4.3 present results of the trained models with respect to the average displacement error which

Model	Future Time Horizon (seconds) - ADE		
	1s	2s	3s
Const. Vel. & Head.	0.48	0.96	1.60
Physics Oracle	0.41	0.76	1.26
FCSingleTS	0.27 ± 0.00	0.62 ± 0.02	1.18 ± 0.03
FCMultiTS	0.24 ± 0.00	0.58 ± 0.00	1.11 ± 0.05
LSTM	0.24 ± 0.00	0.57 ± 0.02	**1.05** ± 0.02
GRU	0.24 ± 0.00	0.60 ± 0.01	1.06 ± 0.01
CNN_G	**0.22** ± 0.00	0.57 ± 0.02	**1.05** ± 0.00
CNN_L	**0.22** ± 0.01	**0.56** ± 0.00	**1.05** ± 0.00

Table 4.2: The average displacement error of proposed baseline models for the first 3 seconds of the future time horizon with each (in case of DL models) containing the mean ADE and std over 5 different training runs.

Model	Future Time Horizon (seconds) - ADE		
	4s	5s	6s
Const. Vel. & Head.	2.42	3.31	4.33
Physics Oracle	1.92	2.63	3.49
FCSingleTS	1.80 ± 0.08	2.54 ± 0.04	3.40 ± 0.07
FCMultiTS	1.74 ± 0.05	2.45 ± 0.05	3.31 ± 0.06
LSTM	1.68 ± 0.03	2.41 ± 0.01	3.22 ± 0.01
GRU	1.69 ± 0.01	2.43 ± 0.01	3.26 ± 0.02
CNN_G	1.61 ± 0.00	2.40 ± 0.02	3.21 ± 0.01
CNN_L	**1.58** ± 0.01	**2.38** ± 0.00	**3.17** ± 0.01

Table 4.3: Further results of the ADE for the future predictions between 4-6 seconds. Again, each cell with DL model presents a mean error and std of 5 independent training runs.

computes an average error over the entire predicted trajectory of all samples. As can be seen from both tables, physics based models demonstrate the worst performance with the constant velocity and heading model performing significantly worse than the physics oracle as well as other deep learning models. The large performance gap is further shown in Figure 4.3 where the error of the constant velocity and heading model grows exponentially as the prediction horizon increases. On the other hand however, the physics oracle model presents a similar performance as the

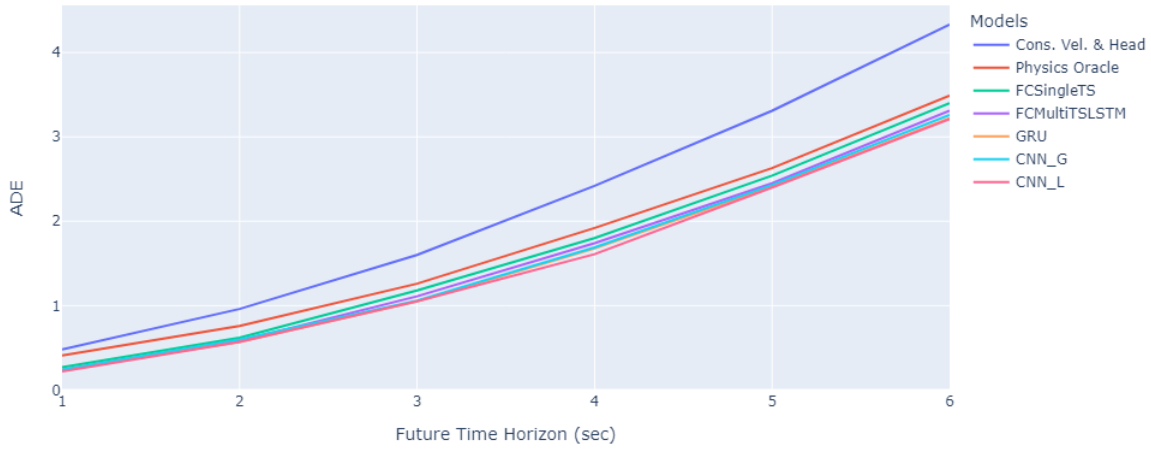

Figure 4.3: A plot of ADE for all models over the entire set of future prediction time-steps. As can be seen the gap between constant velocity and heading model and other models increases exponentially for more distant predictions.

FCSingleTS with the relative ADE difference of only 0.09 meters for the 6 second prediction horizon. Next, observing the mean performance of the DL based models it can be clear that as expected the FCSingleTS network which only considers the state of the agent at time t as its input yields the highest error across all future time-steps where the most apparent difference is visible for predicted trajectories with longer time horizon e.g. $>$ 3 seconds. The FCMultiTS model demonstrates further improvement over the FCSingleTS model which indicates that incorporating more temporal information about the agent is beneficial and results in more accurate predictions. Moreover, the use of networks that are designed to work with temporal data i.e. LSTM and GRU in this case, presents further improvement as those networks process the data in sequential order, thus encoding the temporal information encapsulated within the observed state of an agent. Comparing the performance of both LSTM and GRU it is clear that LSTM outperforms GRU on almost all future time-steps with respect to the ADE error (apart from 5 seconds). In addition, the temporal based networks (as well as subsequent CNN models) demonstrate

lower variance in between training runs as opposed to models based purely on fully-connected layers. Finally, both CNN models which encapsulate not only the state of the agent but also the rasterised maps (global or local) presents the best performance, outperforming temporal models by at least 1.5% as well as significantly outperforming models based solely on fully-connected layers by over 4.2% on the longest prediction horizon (6 seconds).

Furthermore, the analysis of final displacement error in Table 4.4 and 4.5 provide a greater insight into the mean performance of baseline models for the prediction of the furthest position (time-step $t + \tau$) within the anticipated trajectory sequence.

Model	Future Time Horizon (seconds) - FDE		
	1s	2s	3s
Const. Vel. & Head.	0.66	1.82	3.32
Physics Oracle	0.41	0.76	2.55
FCSingleTS	0.37 ± 0.00	1.21 ± 0.06	2.43 ± 0.08
FCMultiTS	**0.34** ± 0.00	1.13 ± 0.07	2.27 ± 0.03
LSTM	0.36 ± 0.00	1.11 ± 0.01	2.22 ± 0.02
GRU	0.37 ± 0.00	1.12 ± 0.01	2.27 ± 0.02
CNN_G	0.35 ± 0.01	1.09 ± 0.02	2.21 ± 0.01
CNN_L	**0.34** ± 0.01	**1.08** ± 0.01	**2.19** ± 0.01

Table 4.4: The final displacement error of the examined baseline models for the first 3 seconds of the predicted trajectories. Again, for the DL based models the mean FDE and std is reported over 5 independent training runs.

First of all, as observed, the relative performance between models for FDE metric is consistent with results of ADE where the physics based models demonstrate poorest performance, again with the significant decrease in accuracy of predicted motion for the constant velocity and heading model as the future time-horizon increases (see Figure: 4.4). Yet again, the CNN based predictors presents top performance, significantly outperforming all models but LSTM and GRU on the final prediction of the agents' positions with the CNN_L also outperforming the version of the model that processes the global map. Finally, results from both ADE and FDE related

Model	Future Time Horizon (seconds) - FDE		
	4s	5s	6s
Const. Vel. & Head.	5.30	7.62	10.23
Physics Oracle	4.13	6.13	8.38
FCSingleTS	4.11 ± 0.08	5.94 ± 0.03	8.29 ± 0.07
FCMultiTS	3.93 ± 0.05	5.82 ± 0.07	8.19 ± 0.07
LSTM	3.76 ± 0.03	5.72 ± 0.03	7.86 ± 0.10
GRU	3.82 ± 0.02	5.75 ± 0.03	7.91 ± 0.05
CNN_G	3.76 ± 0.01	5.70 ± 0.03	7.80 ± 0.02
CNN_L	**3.73** ± 0.01	**5.68** ± 0.04	**7.76** ± 0.01

Table 4.5: Final FDE results for the furthest time-horizons from 4 to 6 seconds into the future with each cell containing the mean (and std for DL models) results of all models.

tables demonstrate that for all DL models there exist an extremely small variance between predictions with respect to different training runs, however, just as in the ADE comparison, the fully-connected based models yield relatively higher variance than more complex counterparts. Interestingly, as can be observed from results with both ADE and FDE the CNN model that uses smaller local maps consistently outperforms its counterpart that uses global maps by a small margin. It is likely that this difference can be attributed to the fact that smaller maps are more specific to the state of the agent rather than a surrounding as a whole, and due to their size carry less noise within the data.

Ultimately, the set of experiments carried out in this section provides numerous valuable insights that will guide further studies. The set of baseline models defined in this section reveal a clear hierarchy of importance for the task at hand. Firstly, the fully-connected models (FCSingleTS and FCMultiTS) demonstrate the value of temporal information. FCSingleTS, limited to the last observed state, underperforms compared to FCMultiTS, which incorporates the entire trajectory history. Secondly, RNN models (LSTM and GRU) outperform fully-connected models, emphasizing the significance of modeling temporal dependencies. LSTM's slight advantage over

GRU hints at its greater ability to capture long-term patterns within the agent's movement. Finally, the excellent performance of CNN-based models (CNN_L and CNN_G) underscores the importance of spatial context provided by maps. Interestingly, CNN_L's use of focused local maps yields slightly better results than CNN_G's global map approach, suggesting that a concentrated view of the agent's surroundings may reduce noise and highlight the most relevant decision-making cues.

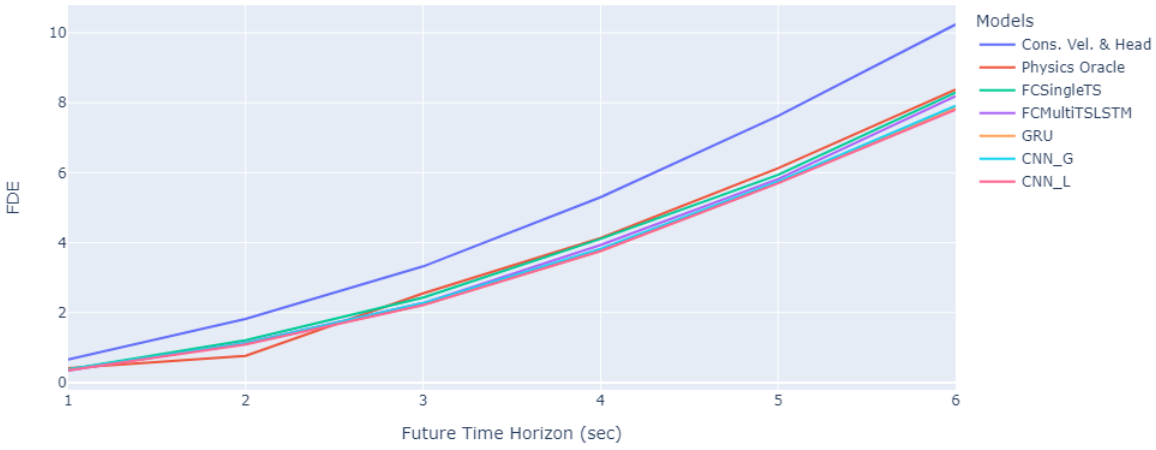

Figure 4.4: A plot that depicts the final displacement error for all models for up to 6 seconds of the prediction horizon.

4.4.3 Unraveling of Semantic Layers

Next, as demonstrated in Figure 4.2 and discussed in section 4.3 the set of semantic layers can be unraveled into several geometrical layers where each layer represents a single part of the whole map, e.g. a walkway or lanes. It can be argued that the unraveling process of semantic layers can potentially provide an additional source of information for the model to learn from. Moreover, in the previous experiment where a number of various types of models were examined to determine a baseline model it became evident that during the training models benefit not only from spatial data

but also from the temporal data (state of the tracked agent), therefore it is also reasonable to examine the effect of including local maps in a temporal manner.

This experiment will focus on examining whether the best performing baseline model from previous section (CNN based model) can be further improved by training it with a) a set of unraveled semantic layers as well as b) a set of temporal semantic layers. Furthermore, the set of separate geometrical layers will also include an additional layer that will portray an agent with respect to its current state as demonstrated in Figure 4.2. The experimental setup remains unchanged and the baseline CNN_L model is trained with the same training setup as models in the previous section with the sole difference being the input data:

1. $\textbf{CNN}_{DL}$ - The baseline CNN model that takes as an input a single unraveled map chunk $\mathbf{L}_t = [\boldsymbol{L}_{t,0},\ \boldsymbol{L}_{t,1}, \cdots, \boldsymbol{L}_{t,n}]$ at the last observable timestep t $(t-0)$. Each separate geometrical layer $\boldsymbol{L}_{t,i}$ is encoded through convolutional layers and then finally concatenated with the state of an agent $\boldsymbol{S}$ and finally processed by the fully-connected layer to make final predictions.

2. $\textbf{CNN}_{TDL}$ - The extension of the baseline CNN model where apart from using the disentangled local layers at the last observable timestep, the entire history of local maps from time t to $t-\rho$ is encoded and used to predict future movement. Again, each layer for each timestep is encoded separately, however, this time instead of using a fully-connected layer to encode the state of an agent, a LSTM layer with 128 units is used. This allows to concatenate the encoded state of an agent s_t at time t with corresponding encoded semantic layers $\mathbf{L}_t$ and then use that as an input to fully-connected layer to predict future movement of an agent.

Results of the experiment are presented in tables 4.6 and 4.7 (ADE and FDE respectively). Note that for this experiment only a future time horizon of 4, 5 and 6

seconds was considered as predictions with respect to shorter times are significantly less critical for this work. As initially expected, the inclusion of additional sources of data whether in a form of geometrical layers or in a form of temporal maps provides a significant advantage during the learning process, allowing the baseline model to improve further over all future time horizons, gaining a slight but not necessarily a significant improvement on both ADE and FDE on the 6 second horizon for up to 3.7% and 2.4% respectively. This confirms that providing the model with more fine-grained and temporally-aware spatial information enhances its predictive ability.

Model	Future Time Horizon (seconds) - ADE		
	4s	5s	6s
CNN_L	1.58 ± 0.01	2.38 ± 0.00	3.17 ± 0.01
CNN_{DL}	1.55 ± 0.03	2.36 ± 0.01	3.12 ± 0.00
CNN_{TDL}	**1.51** ± 0.02	**2.31** ± 0.01	**3.05** ± 0.02

Table 4.6: ADE results for the time-horizons from 4 to 6 seconds with each cell containing the mean and std results of three variants of the CNN baseline model.

Model	Future Time Horizon (seconds) - FDE		
	4s	5s	6s
CNN_L	3.73 ± 0.01	5.68 ± 0.04	7.76 ± 0.01
CNN_{DL}	3.65 ± 0.02	5.60 ± 0.01	7.68 ± 0.03
CNN_{TDL}	**3.55** ± 0.01	**5.51** ± 0.02	**7.57** ± 0.02

Table 4.7: FDE results for the time-horizons from 4 to 6 seconds with each cell containing the mean and std results of three variants of the CNN baseline model.

4.4.4 Finding an Optimal Map Size

Lastly, during the previous two experiments the size of a local map chunk, or rather an area of the agent's surrounding that it encapsulates remained as a fixed size of

$\lambda = 20$ meters. Given the default setting the model was able to achieve promising results, however, it is still crucial to conduct further examination to find the optimal map size (in meters). The subsequent experiment examines following the set of $\lambda \in \{5, 10, 15, 20\}$ settings at a 6 second prediction horizon to determine whether the displacement errors can be further reduced. As with previous experiments the same training setup for the model of interest, in this case the CNN_{TDL} are being adopted to ensure that a fair comparison is conducted. Results of the experiment are presented in figure 4.5.

As demonstrated, various configurations of the λ parameter result in a minor difference for both ADE and FDE. What is however apparent is that setting the $\lambda = 5$ yields the least optimal results which suggests that the size of the captured surrounding might considerably restrict the learning process due to the limited amount of information that is carried within the rasterised maps. In addition, it can be noted that the optimal configuration of λ appears to be when $\lambda = 15$, the difference however between other settings seems insignificant. It is nonetheless interesting to see that a bigger size of the map does not guarantee better results which suggests that larger maps might carry a significant amount of noise that negatively contributes the training of the model. In conclusion, the experiment on optimal map size provides insights into the trade-offs associated with different configurations. While a slightly larger map size appears optimal, there's a diminishing return with excessively large maps.

4.5 Conclusion

This chapter focused on introducing an alternative methodology towards a task of a short-term motion prediction of vehicles from a perspective of an autonomous vehicle in a complex urban setting. First, an alternative approach of using rasterised

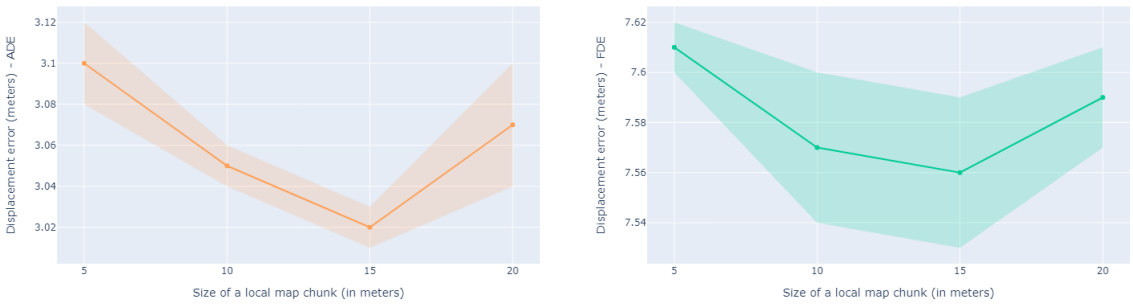

Figure 4.5: An ADE and FDE results of the CNN_{TDL} model trained for a 6 seconds of future time horizon on various settings of map size λ.

top-down 2D HD maps has been proposed and examined. Instead of a typical usage of rasterised maps where global maps that encapsulate large chunks of the surrounding environment are utilised as an additional source of information for the model, a local map chunks that can be separated into an individual semantic pieces were used instead. This is the first time the concept of local maps has been utilised for the task at hand. As demonstrated in section 4.4 the addition of auxiliary information in a form of rasterised maps enabled the trained model with a CNN feature extractor to perform slightly better than e.g. a LSTM based model where only temporal information about past movement of the agent was provided. In addition, the model that utilized local map chunks as opposed to the global counterparts provided further improvement with respect to the final performance of the model. It can be argued that it is highly possible that the improvement of the model that was trained with local maps as opposed to global maps can be attributed to the fact that the information carried with global maps carry a significant amount of noise that does not contribute to the training. Next, the impact of unraveling of semantic maps was further examined to determine whether representing the rasterised map data in a form of separate geometrical layers can be beneficial. In addition, the experiment also examined advantages of incorporating map data over time as opposed to using the 2D view from the last observable time-step. As demonstrated, the baseline

model with the lowest error with respect to two main metrics (ADE & FDE) has proved to be the CNN which incorporated both the separate local map chunks as well as maps that were observed over longer time horizon further suggesting that it is indeed advantageous to include the temporal information in a spatial form. Lastly, an ablation study has been carried out to explore and find the optimal size of the local map chunk, and as presented in the last subsection of section 4.4 maps with higher spatial range do not necessarily contribute positively towards learning process.

The next chapter will explore an alternative CNN based method for extracting features from rasterised map data to determine whether a large pre-trained models are indeed, as usually presented in the literature, an adequate option for the task of motion prediction.

Chapter 5

Improving Deterministic Motion with Capsule Networks

5.1 Introduction

A typical technique for processing spatial data, as presented in the literature survey as well as previous chapter, usually requires employing some sort of a CNN based feature extractor which encodes the rasterised map input and then extracts a set of meaningful road features that are subsequently combined with e.g. the observed agent's motion data in order to compute the final prediction. A common choice for a CNN based feature extractor involves using one of the popular CNN models, for example ResNet-50 which is initially pre-trained on a very large and diverse dataset such as the *ImageNet* dataset (Russakovsky et al. 2015). In principle, this approach has been adopted as a standard technique due to its general effectiveness in computer vision, however, recent studies demonstrate that training models from scratch on the domain specific dataset leads to results that are often on pair (or slightly better) with their pre-trained counterparts when given enough training iterations (He et al. 2019). Additional findings also suggests that pre-training on for instance

an ImageNet appears to be less beneficial for tasks that rely heavily on localisation and are primarily advantageous to classification related tasks. Furthermore, the ImageNet dataset is composed of over 1 million annotated diverse images belonging to 1000 different classes (see Figure: 5.1) which is significantly more complex than rasterised maps used for motion prediction meaning that the deep layers of a large vision models such as a pre-trained ResNet which learned highly abstract representations of those classes could provide little to no use when trying to extract relevant representations from maps that are mainly constructed out of very low-level features such lanes, edges etc. Additionally, majority of popular pre-trained vision models often contain a large number of parameters, e.g. ResNet50 contains 25 millions trainable parameters whereas its deeper version named ResNet152 contains over 60 million parameters, and although for tasks such as image classification or object recognition such complexity is necessary it is arguable whether this approach is viable for learning valuable representation from relatively naive rasterised maps.

The discussed points suggest that the use of large feature extractors that are pre-trained on a very distinct domain might be redundant or even harmful as similar or better level of accuracy can be potentially achieved with significantly lighter models that are trained end-to-end from scratch. Therefore, this chapter will explore an approach towards utilisation of map data for the deterministic motion forecasting of an on-road vehicles from the perspective of autonomous vehicle with a novel CNN feature extractor based on the capsule network which aids to alleviate the drawbacks that a standard CNN architectures exhibit i.e. lack of equivariance to other affine transformations but translation, and inability of achieving invariance without use of additional pooling mechanism (Goodfellow et al. 2016).

Figure 5.1: An example of six images from different classes taken from the ImageNet training set.

5.2 Capsule Neural Networks

As previously discussed, a typically employed approach that is utilised to encode a scene context is to use the rasterised HD map data in order to extract representative spatial road features with the use of a CNN, often a variant of ResNet, by either leveraging benefits of transfer learning (Wang et al. 2019) and fine tuning certain parts of the feature extractor for the task at hand, or by training the network in an end-to-end fashion. Nonetheless, despite achieving state-of-the-art performance in recent years on several computer visions tasks such as object classification (Touvron et al. 2020), detection (Acharya et al. 2020) and semantic segmentation (Tao et al. 2020), the CNN still exhibit certain crucial drawbacks (Goodfellow et al. 2016).

First, convolutional layers are unable to account for the equivariance with respect to various transformations. An arbitrary function $f(x)$ is said to be equivariant to function $g(x)$ if it's output reflects the same changes that were applied to the input x through $g(x)$ i.e. $f(g(x)) = g(f(x))$. For example, let $g(\cdot)$ denote a translation function that shifts each pixel of an input image $\boldsymbol{I}$ one pixel to the right such that $\boldsymbol{I'}(x, y) = \boldsymbol{I}(x - 1, y)$. Next, let $f(\cdot)$ denote a convolution operation, if $f(\cdot)$ were to be equivariant with respect to $g(\cdot)$, then the output of $f(g(\boldsymbol{I}))$ would be the

same as firstly applying the convolutional operation and then the translation i.e. $g(f(\boldsymbol{I}))$. CNNs exhibit the property of being equivariant solely to the translation, other transformations such as rotation and scaling will often cause relevant neurons to not fire and thus fail when detecting salient features within spatial domain. The translation equivariance is achieved through parameter sharing as the same learned weights (kernel) shift through the entire input image, thus enabling the convolutional layer to detect features of interest irrespective of their positions.

Secondly, in addition to the issue of equivariance, CNNs lack the ability to handle spatial invariance i.e. given an arbitrary affine transformation A e.g. translation, the convolutional layer denoted by $f(\cdot)$ would be invariant to such translation if $f(\cdot)$ satisfied the following $f(x) = f(A(x))$. One approach that has been extensively employed within CNN's pipeline that helps to achieve an approximation of local translation invariance is known as pooling operation which has been previously discussed in section 2.3. The output of pooling operation e.g. max-pooling summarises feature maps computed by a former convolutional layer by sliding a filter, often a 2×2 with a stride of 2, throughout layer's input tensor and taking a maximum value within each small window. This operation results in each feature map preserving only the values with highest activation whilst simultaneously being downsampled by a factor of 2. Evidently, pooling operation provides certain benefits as downsampled feature maps cause subsequent convolution layers to have significantly less learnable parameters, thereby making the network computationally less expensive, and as previously mentioned enabling a model to have a capability of achieving a small translation invariance. Nonetheless, these properties might not be always desirable, for instance, for a task of object classification where the position of given object is not as crucial as its sole presence the pooling will provide advantages, however, if a given task is highly dependant on the position of local features as well as their global relationship then pooling will effectively eliminate such information. Figure 5.2 demonstrates a case of an image that contains two variations of the representa-

tion of human's face i.e. a face on the left preservers the relationship between local features whereas the face on the right contains local features at arbitrary positions. Due to weight sharing as well as the pooling mechanism it is highly possible that a CNN trained to classify whether a human face is present or not (binary classification) would in both cases produce a positive outcome i.e. face being present.

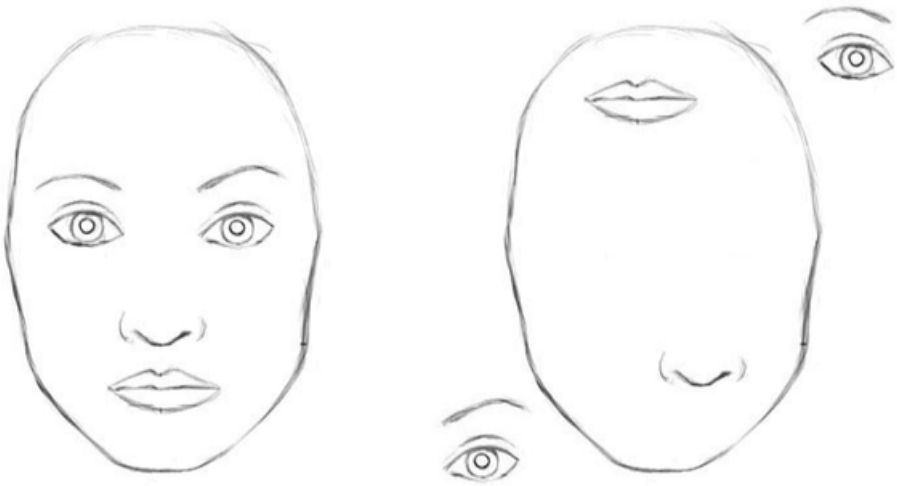

Figure 5.2: An image of two human faces with a face on the left preserving hierarchical relationship between local features, and face on the right having local features in arbitrary positions (*CNN face detection failure* n.d.).

To tackle these issues Hinton et al. (2011) proposed a novel type of neural network known as capsule network that implements the idea of using capsules (locally invariant group of neurons) to learn various properties (e.g. pose) of the same object and encode them in an output vector whose length corresponds to the probability of that object being present. Encoded parameters can be further conceptualised as an object's instantiation parameters that enable the model to learn more robust and equivariant representation of features with respect to change in viewpoint. Furthermore, Sabour et al. (2017) introduced the *routing-by-agreement* mechanism by which capsules from lower levels decide which of their output vectors should be sent to higher level capsules. In essence, output vectors from lower level capsules are

used to predict the output of higher level capsules, predictions are then compared with actual outputs to iteratively compute "agreement" (cosine similarity) between lower and higher capsules. For instance, the mere presence of a nose or eyes (lower level capsules) should not be a sole indicator that the face (higher level capsules) exists within an image, a hierarchical relationship (e.g. rotation) between low and high level features should have a high impact on the final prediction.

The outlined advantages of capsule networks over standard CNN architectures strongly motivate for examination of this type of networks for the task of motion prediction. Therefore, the subsequent set of experiments will investigate how a DL motion predictor with a capsule based feature extractor can be utilised to improve the overall performance of the system whilst maintaining the efficiency of the model with respect to its complexity. First, the architecture as well as the computational flow of the proposed capsule based encoder will be described in details.

5.3 Network Architecture and Computational Flow

The architecture of the proposed and examined CapsNet based feature extractor loosely follows the implementation from the original work of Sabour et al. (2017), however, the proposed model in this case is constructed as a four-part network as demonstrated in Figure 5.3. Let (k, s, o) denote the tuple that specifies a convolution layer's kernel size, stride size, and the number of output channels respectively (the padding is set to zero for all convolutional layers). First, a shallow convolutional base $\Phi_{\text{base}}(\cdot)$ is defined to extract local, low-level features from semantic layers which is composed of a single convolutional layer with the following set of hyperparameters; $(k = 9, s = 2, o = 64)$ followed by a variant of ReLU activation function i.e. Exponential Linear Unit (ELU) non-linearity (Clevert et al. 2015) which is defined

as:

$$\mathrm{ELU}(\boldsymbol{z}) = \begin{cases} \boldsymbol{z} & \boldsymbol{z} > 0 \\ \alpha(e^{\boldsymbol{z}} - 1) & \boldsymbol{z} \leq 0 \end{cases} \tag{5.1}$$

and provides certain benefits that aid with mitigating the issues that can be otherwise introduced by ReLU such as dying ReLU (Trottier et al. 2017). Next, the second part of the encoder i.e. lower level (primary) capsules $\Phi_{\text{lower}}(\cdot)$ is defined to learn parameters for more trivial parts of the input data. Every capsule is a $4D$ unit, where each of its dimensions corresponds to the scalar output value of an independent set of consecutive convolutional layers defined by the following set of hyperparameters; $(k = 9, s = 2, o = 32)$, $(k = 2, s = 2, o = 16)$ for the first and second layer respectively, hence, the number of convolutional layers within $\Phi_{\text{lower}}(\cdot)$ is equal to 4×2. Instead of using e.g. the ELU activation function in-between convolutional layers which treats each value as being independent, a *'squashing'* non-linearity introduced by Sabour et al. (2017) is employed in order to normalize the input vector $\boldsymbol{z}$ so that the magnitude of short and long vectors is squashed to almost 0 and just below 1 respectively:

$$\Phi_{\text{squash}}(\boldsymbol{z}) = \frac{\|\boldsymbol{z}\|^2}{1 + \|\boldsymbol{z}\|^2} \frac{\boldsymbol{z}}{\|\boldsymbol{z}\|} \tag{5.2}$$

Since low level features of input images e.g. edges and lanes resemble strong similarities among different types of semantic layers, it is reasonable to use a single $\Phi_{\text{lower}}(\cdot)$ layer to learn and extract their parameters. Nonetheless, to account for the fact that their final representation differs significantly, a single higher capsule layer $\Phi_{\text{higher}}(\cdot)$ is defined per each type of semantic layer in the subsequent part of the encoder. Hence, each type of semantic layer $\boldsymbol{L}_{t,i}$ is encoded through its own respective higher capsule $\Phi_{\text{higher\_i}}(\cdot)$ that outputs a $32D$ vector of its latent representation. Moreover, the encoder's final part is defined as a single capsule $\Phi_{\text{final}}(\cdot)$ which outputs a $128D$ vector containing a joined latent representation of all semantic layers. Lastly, for

each $\Phi_{\text{higher}}(\cdot)$ as well as $\Phi_{\text{final}}(\cdot)$ learnable parameters are sampled from Gaussian distribution and then scaled by ϵ such that $\epsilon \boldsymbol{w} \sim \mathcal{N}(\mu, \sigma^2)$ where $\epsilon = 0.1$, $\mu = 0$ and $\sigma^2 = 1$.

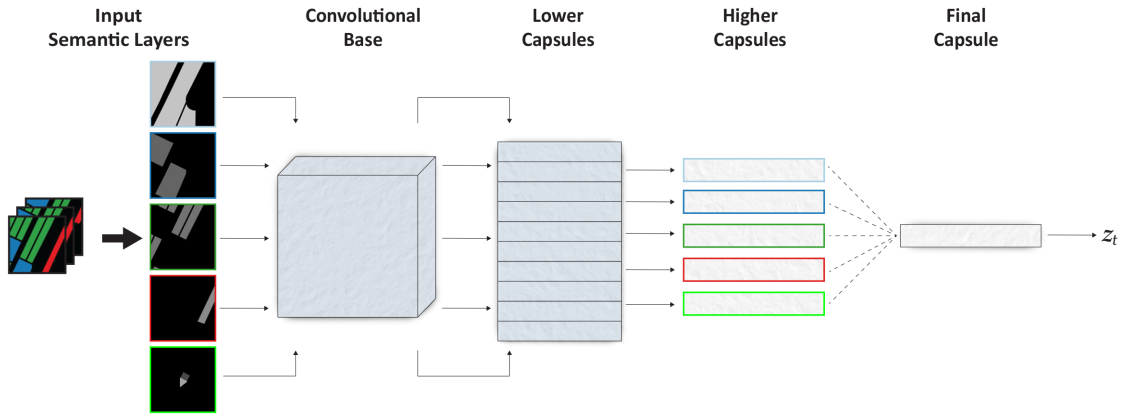

Figure 5.3: An overview of the proposed Capsule Net encoder. A single map chunk at time t is disentangled into separate geometrical layers. Each layer is then passed separately to the convolutional base, followed by lower capsules and then to its corresponding higher capsule. Finally, outputs of all higher capsules are concatenated and passed to the final capsule to compute the final encoding.

Next, the computational flow of data through the proposed network that yields the short-term future motion of an agent of interest is described, including both the capsule encoder as well as the motion decoder. Although the description represents the computation process of a single agent, it is trivial to extend the method for a multi-agent scenario. For the purpose of simplicity and to focus on the main aspect of the method, a single-agent case is maintained.

First, for each observed time-step t a set of semantic layers $\mathbf{L}_t$ are encoded through the capsule encoder. Each disentangled grayscale layer $\boldsymbol{L}_{t,i}$ is transformed individually to compute its output representation vector by first running the matrix through a convolutional base:

$$\mathbf{Z}_{\boldsymbol{t,i}} = \Phi_{\text{base}}(\boldsymbol{L}_{t,i}) \tag{5.3}$$

where $\mathbf{Z}_{\boldsymbol{t,i}}$ is the output tensor containing 64 feature maps of size 28×28. Next,

the $\mathbf{Z}_{t,i}$ is further passed into the $\Phi_{\text{lower}}(\cdot)$ layer to compute lower capsules:

$$\boldsymbol{Z}_{t,i} = \Phi_{\text{lower}}(\mathbf{Z}_{t,i}) \tag{5.4}$$

where $\boldsymbol{Z}_{t,i}$ is the squashed output matrix with $400 \times 4D$ capsules (second convolution layer outputs 16 feature maps of size 5×5). Furthermore, $\boldsymbol{Z}_{t,i}$ is passed through its respective $\Phi_{\text{higher\_i}}(\cdot)$ to get the $32D$ output vector $\boldsymbol{z}_{t,i}$ whose scalar values corresponds to the layer's $\boldsymbol{L}_{t,i}$ latent instantiation parameters:

$$\boldsymbol{z}_{t,i} = \Phi_{\text{higher\_i}}(\boldsymbol{Z}_{t,i}) \tag{5.5}$$

This process is then repeated for each semantic layer with each output being concatenated to create an input matrix for the final capsule that outputs a final $128D$ vector $\boldsymbol{z}_t$ at time t:

$$\boldsymbol{z}_t = \Phi_{\text{final}}(\text{concat}(\boldsymbol{z}_{t,0}, \boldsymbol{z}_{t,1}, \cdots, \boldsymbol{z}_{t,n})) \tag{5.6}$$

In addition, each state vector $\boldsymbol{s}_t$ is encoded through a fully-connected layer with 128 output units, followed by the ELU activation which is then concatenated with a corresponding $\boldsymbol{z}_t$ to form the matrix of shape $[\rho, 256]$ where ρ refers to the number of observed time-steps. The matrix is then encoded in the temporal manner through the use of the LSTM layer with hidden-state size of 128. Finally, the last hidden-state (final encoded output) of the LSTM layer is passed through the decoder (fully-connected layer) with $2(x, y) \times \tau$ units where τ corresponds to the future time-horizon, in order to get the final future predictions. The output of the decoder is then reshaped to create the target matrix $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_{t+1}, \hat{\boldsymbol{y}}_{t+2}, \cdots, \hat{\boldsymbol{y}}_{t+\tau}]$ with each vector containing the predicted future position (x, y) of the tracked agent, relative to its position at the last observed time-step t.

5.4 Experiments and Results

5.4.1 Initial Ablation Study

First, in order to demonstrate the benefits of the proposed method, an initial ablation study is conducted between the described capsule based network and its simpler variants (note that throughout the remainder of this chapter the main model defined in the subsection 5.3 will be referred to as **MotionCaps**):

1. **SimpleCaps** - A similar yet basic version of the proposed network that is composed of the capsule based extractor constructed only with the convolutional base Φ_{base} as well as a layer composed of lower level (primary) capsules such as that of Φ_{lower} as well as final layer Φ_{final}, thus omitting the higher capsules Φ_{higher} that are responsible for an independent encoding of different types of semantic layers. In addition, instead of processing each type of semantic layer independently through the Φ_{final}, a whole local map at time t (see the first column of Figure 4.2) is passed to the encoder, thus performing a similar computational process as that of the **CNN** model described in 4.4.1. The output of this backbone feature extractor is a $128D$ vector $\boldsymbol{z_t}$ at time t that is then concatenated with the corresponding state vector $\boldsymbol{s_t}$.

2. **SimpleLayerCaps** - A subsequent variant of the **SimpleCaps** model that instead of using a local map as a whole as its input, uses a set of separate semantic geometrical layers $\mathbf{L}_t$ (just like the proposed capsule extractor) as depicted in Figure 4.2. However, yet again, this model does not include the higher capsules Φ_{higher} for individual processing of each type of layers, but instead processes each $\boldsymbol{L}_{t,i}$ through the final capsule layer Φ_{final} and then aggregates each output of the Φ_{final} (i.e. $128D$ vector) through a summation operation. Again, the result of applying this feature extractor on the input

layers is a $128D$ vector $\boldsymbol{z_t}$ at time t.

The setup for this experiment is similar to the setup explained in subsection 4.4.1 where the above models are trained 5 times (independent runs) with Adam optimizer for 100 epochs per each training run, however, this time only results for every $t \in \{4, 5, 6\}$ future horizon are reported as it is more crucial to focus on the longer prediction horizon. In addition, a learning rate scheduler is adopted during the training which decays the initial learning rate $(5e-4)$ by $\gamma = 0.1$ at epoch 5 and 20.

Tables 5.1 and 5.2 demonstrate results from the conducted experiment. In addition, a performance of a CNN_{TDL} model from subsection 4.4.4 has been added to the table as it is essential to first explore and compare the performance of the basic capsule based motion decoder (SimpleCaps) to emphasize on the sole benefits of using the proposed architecture. The comparison between the basic capsule-based model and its CNN_{TDL} counterpart reveals a notable performance advantage for the capsule-based approach. Specifically, the capsule-based model achieves a significant reduction in both ADE and FDE errors, outperforming the CNN_{TDL} counterpart by 5.9% and 6.8% respectively on the 6-second prediction horizon. This highlights the effectiveness of the proposed capsule-based architecture in capturing complex motion dynamics compared to traditional convolutional neural networks. Interestingly, the performance gain of the capsule-based model remains consistent across all future time-steps, with notable improvements observed as the prediction horizon increases. This suggests that the additional complexity introduced by higher-level capsules in MotionCaps is justified, as it enables more accurate predictions over longer time horizons. Furthermore, while the performance gap between iterations of capsule-based models is marginal, MotionCaps consistently outperforms other variants across all future time-steps on both metrics. This suggests that the inclusion of higher-level capsules in MotionCaps contributes to its superior performance,

allowing for more effective encoding of diverse semantic layers and complex motion patterns. Unexpectedly, the second iteration of the model (SimpleLayerCaps) performs worse than its more basic version, however, it is still outperformed by MotionCaps. The reason behind this performance gap could be explained by the fact that SimpleLayerCaps uses a larger array of image data (separate semantic layers) than SimpleCaps which uses a single image of a local environment, thus requiring a higher level of complexity in order to be able to learn much richer internal representation. This is in fact further proven with MotionCaps which utilizes the same type of data as SimpleLayerCaps, yet the model complexity is increased with an addition of a higher capsule layer per each type of semantic layer.

Model	Future Time Horizon (seconds) - ADE		
	4s	5s	6s
CNN_{TDL}	1.51 ± 0.02	2.31 ± 0.01	3.05 ± 0.02
SimpleCaps	1.41 ± 0.03	2.09 ± 0.01	2.87 ± 0.02
SimpleLayerCaps	1.45 ± 0.01	2.16 ± 0.02	2.95 ± 0.01
MotionCaps	**1.38** ± 0.02	**2.04** ± 0.01	**2.80** ± 0.04

Table 5.1: The mean and std ADE between three variants of Capsule based motion prediction over 5 independent runs.

Model	Future Time Horizon (seconds) - FDE		
	4s	5s	6s
CNN_{TDL}	3.55 ± 0.01	5.51 ± 0.02	7.57 ± 0.02
SimpleCaps	3.25 ± 0.02	5.02 ± 0.03	7.05 ± 0.02
SimpleLayerCaps	3.29 ± 0.01	5.09 ± 0.02	7.11 ± 0.02
MotionCaps	**3.21** ± 0.01	**4.97** ± 0.01	**6.99** ± 0.03

Table 5.2: The mean and std FDE between three variants of Capsule based motion prediction over 5 independent runs.

5.4.2 Performance Comparison of Capsule Encoder vs Popular CNN Models

In the introductory section of this chapter the topic of using pre-trained networks as a means of transferring the already acquired knowledge (transfer learning) from a larger and often more general datasets towards a different task has been covered. As discussed, the idea of using transfer learning has been adopted as a regular method for numerous tasks that rely on deep learning models. However, recent findings suggest that focusing on the sole use of pre-trained models might not necessarily yield an optimal solution due to certain limitations, for example, a decrease in a fine-tuned model's performance on tasks that are heavily dependent on a precise localisation of salient features within the spatial domain (He et al. 2019). In order to further examine the effectiveness of the proposed capsule based model, a set of models with popular architectures (e.g. ResNet) that are often employed in the literature will be initially trained from scratch for the task of motion prediction and compared against the MotionCaps model. Then, the same set of models will be trained again, however, this time with the weights being initially pre-trained on the ImageNet and then fine-tuned for the task at hand to demonstrate the difference in their performance. The following models are considered during this experiment[2]:

1. ResNet - One of the most popular type of convolutional architectures that is often being used as a baseline backbone in various computer vision tasks. Prior to the introduction of ResNet architecture the training of very deep networks, as discussed in the original paper, seemed infeasible due to the *degradation* problem which causes the deeper network's performance to saturate and then rapidly degrade as compared to its shallow counterpart. ResNet introduced what is known as *Residual Block*, where instead of learning a mapping $H(\boldsymbol{x})$ through a stack of layers the residual block aims to learn $F(\boldsymbol{x}) = H(\boldsymbol{x}) -$

[2]All considered pre-trained models are available on the official PyTorch repository

$\boldsymbol{x}$ and thus the final mapping function becomes $H'(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x}$ where the second term i.e. addition of $\boldsymbol{x}$ is referred to as *skip connection*. The paper demonstrated that learning residual function $F(\boldsymbol{x})$ and introducing skip connections to the computational graph eases the learning process and does indeed allow to effectively train much deeper networks.

2. DenseNet (Huang et al. 2017) - Use of skip connections has also proved to be an effective strategy for another type of CNN architecture inspired by ResNet known as *DenseNet*. The main idea behind DenseNet architecture concentrates on reusing network's feature maps. More specifically, a set of features maps from an earlier layer is passed not just to the next layer but to all subsequent layers within a *Dense Block* (hence DenseNet). In addition, instead of adding the data from incoming skip connections (as is the case for ResNet), the DenseNet concatenates it allowing subsequent layers to reuse feature maps from earlier parts of the network.

3. Inception (Szegedy et al. 2015) - The standard way of defining a CNN simply involves stacking number of convolutional layers one after another such that the network becomes deep and complex enough to learn the task at hand. The Inception model proposed a new paradigm of defining the CNN architecture where a number of filters operate on the same computational level. For example, given an input image with a single channel $\boldsymbol{I} \in \mathbb{R}^{w \times h \times 1}$ and a naive convolutional layer with a single 3×3 kernel the output of such a layer would be a single feature map produced as a result of convolving this kernel over the input image. On the other hand however, the Inception block implements a number of kernels (often stacked on top of another w.r.t. convolutional operations) of various sizes for example 1×1, 3×3, 5×5, whose output feature maps are concatenated to produce the final output. The main reason behind the usage of kernels of various spatial sizes is to give the CNN the ability to look at

the input at a different spatial resolutions. Additionally, subsequent versions of Inception network Szegedy et al. (2016) introduced numerous improvements such as a reduction of the output's depth through a usage of 1×1 kernels, a reduction of computational complexity through factorization of kernels e.g. a 5×5 convolution (25 weights) can be represented by two 3×3 convolutional operations (18 weights), and even further factorization of any $n \times n$ kernels into two separate $1 \times n$ and $n \times 1$ kernels.

4. MobileNet (Howard et al. 2019) - The last CNN architecture that will be considered in the following experiment was proposed and designed to be suited for mobile vision applications. The MobileNet family of CNNs aim to maximize the performance of the model on a given task whilst simultaneously reducing the computational intensity. MobileNet achieves its performance with use of *Depth-wise separable convolution*, an operation that is constructed with two separate layers. Firstly, the depth-wise convolution is applied onto the input image $\mathbf{I} \in \mathbb{R}^{w \times h \times c}$ by convolving a single kernel over each channel (thus the number of kernels within the layer is equal to c) which results in an output tensor $\mathbf{I'} \in \mathbb{R}^{w' \times h' \times c}$. Then a point-wise convolution layer is applied onto $\mathbf{I'} \in \mathbb{R}^{w' \times h' \times c}$ by convolving n 1×1 kernels to produce linear combination of the output of the former layer resulting in an output tensor $\mathbf{O'} \in \mathbb{R}^{w' \times h' \times n}$.

Furthermore, during initial experiments it was discovered that using the default output of models that were proposed for comparison purposes (e.g. ResNet-50) as a direct input to the following, temporal modules within the network quickly leads to either significant overfitting or exploding gradients early in the training. The primary cause of this instability during training is a result of a relatively large size of an output vector of the convolutional feature extractor (e.g. output of ResNet-50 in this case is an vector of 2048 units) in comparison to the encoded size of the motion state vector $\boldsymbol{s_t}$ (128 units). To account for this and ensure that a

comparison between the proposed networks is indeed fair and that the motion state vector is not overwhelmed by features extracted from semantic maps an additional fully-connected layer with 128 units is stacked on top of each model in order to downsample the output. Next, the output of each feature extractor is concatenated with an encoded motion state $\boldsymbol{s_t}$ of an observed agent to form a matrix of shape $[\rho, 256]$ where ρ refers to the number of observed time-steps. The subsequent steps follow exactly the same procedure as MotionCaps i.e. the concatenated matrix with extracted visual features as well as the encoded motion state is decoded through an LSTM layer and finally through the fully-connected layer to yield a predicted motion in the form of a target matrix $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_{t+1}, \hat{\boldsymbol{y}}_{t+2}, \cdots, \hat{\boldsymbol{y}}_{t+\tau}]$. Finally, each model is also trained in an identical fashion as MotionCaps, adopting the same optimizer, scheduler, number of epochs, learning rate as well as other relevant hyperparameters.

Results of the experiment are presented in table 5.3 with each cell containing the ADE and FDE errors for the corresponding time horizon (between 4 and 6 seconds) as well as with an additional column that shows the complexity of each backbone feature extractor in terms of number of its parameters. The results in 5.3 clearly demonstrate that the motion prediction based on the capsule feature extractor (MotionCaps) consistently outperforms other backbone feature extractors across all time horizons. Despite having significantly fewer parameters compared to the other networks, MotionCaps achieves lower ADE and FDE errors, indicating its effectiveness in capturing motion dynamics. This suggests that the capsule-based architecture effectively encodes and utilizes spatial-temporal information for accurate motion prediction. Additionally, it is interesting to see that the performance of each model does not correlate with its complexity, as primarily demonstrated by MotionCaps and DenseNet, but rather the architecture and its benefits.

Furthermore, figures 5.4 and 5.5 demonstrate the training and validation loss of each model over the 100 epochs of the training process with the latter figure showing the

Model	Future Time Horizon (seconds) - ADE/FDE			Params (Backbone)
	4s	5s	6s	
MobileNet V2	1.60/3.69	2.29/5.37	3.03/7.29	3.50m
Inception V3	1.48/3.51	2.22/5.29	2.95/7.23	27.16m
ResNet-50	1.39/3.28	2.10/5.15	2.82/7.12	23.50m
DenseNet-121	1.36/3.21	2.03/4.99	2.79/6.94	7.97m
MotionCaps	**1.34**/**3.17**	**1.99**/**4.91**	**2.74**/**6.89**	0.95m

Table 5.3: The ADE and FDE errors between proposed backbone feature extractors from the literature and the Capsule based motion predictor. Additionally, the number of parameters of each backbone feature extractor has been included to demonstrate the performance with respect to model's complexity.

progress after the first 20 epochs in order to demonstrate the point at which each model starts to overfit. Figure 5.4 illustrates the training and validation loss trends over 100 epochs for each model. It is noteworthy that while all models reach a local minimum early in the training (around the 10th epoch), MotionCaps exhibits a slower convergence rate compared to other models. This prolonged convergence may indicate the complexity of the capsule-based architecture and the time required for it to effectively learn motion patterns from the data. Nevertheless, as observed closer in figure 5.5 despite this, the capsule based model eventually reaches and outperforms other models on both training and validation loss. Ultimately, further in the training it can be seen that the performance of all models stops improving, the validation curve diverges from the training curve which leads to overfitting with MobileNet and Inception showing poorest performance as well as stability over the course of the training.

Lastly, as mentioned in the beginning of this sub-section, a typical way of approaching a number of deep learning tasks is to start with a previously trained model that can be then fine-tuned for the task at hand. In addition, it was also discussed that the idea of fine-tuning might not necessarily be suitable if the task on which the model was initially pre-trained is significantly different. For example, using a

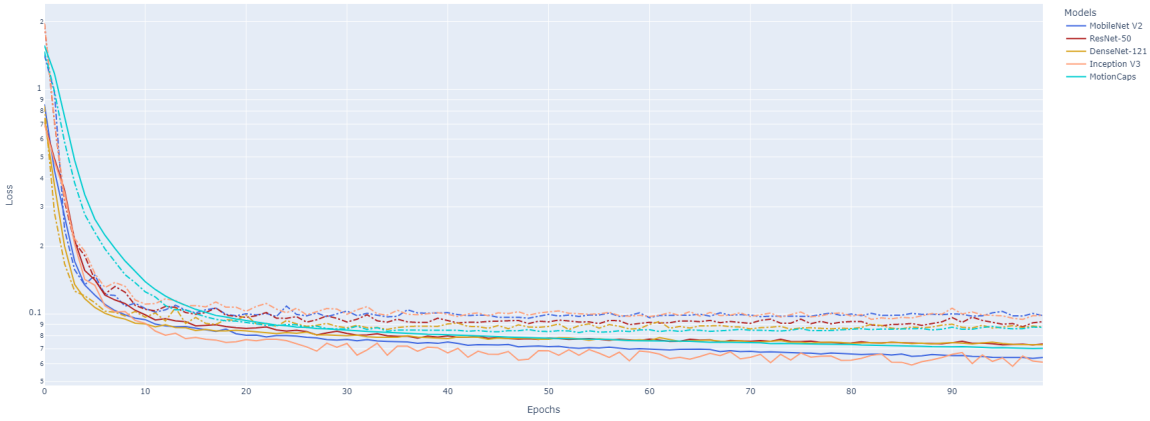

Figure 5.4: The training (solid line) and validation loss (dash line) of each model over 100 epochs. Training loss is presented with a solid line whereas the dashed line shows the validation loss.

model trained for a classification task might not necessarily yield suitable results if the latter task relies heavily on precise localisation of features. Figure 5.6 presents results of models that were previously proposed for comparison purposes in terms of their validation loss with the solid line showing results of those models trained from scratch and the dash line showing results of models being initially pre-trained on ImageNet and then fine tuned for the task of motion prediction. As can be seen from the results below, it is evident that all pre-trained models demonstrate significantly poorer performance than the same variants that were trained from scratch directly for the task at hand. Furthermore, the results in figure 5.6 reveal that pre-trained models exhibit significantly poorer performance compared to models trained from scratch, as indicated by their higher validation loss. Notably, pre-trained models also require more epochs to reach a certain local minimum, typically around the 60th epoch, after which their validation loss starts to plateau. This suggests that pre-training on ImageNet and fine-tuning for motion prediction may not effectively leverage the spatial-temporal features required for accurate motion prediction, leading to suboptimal performance and prolonged training convergence.

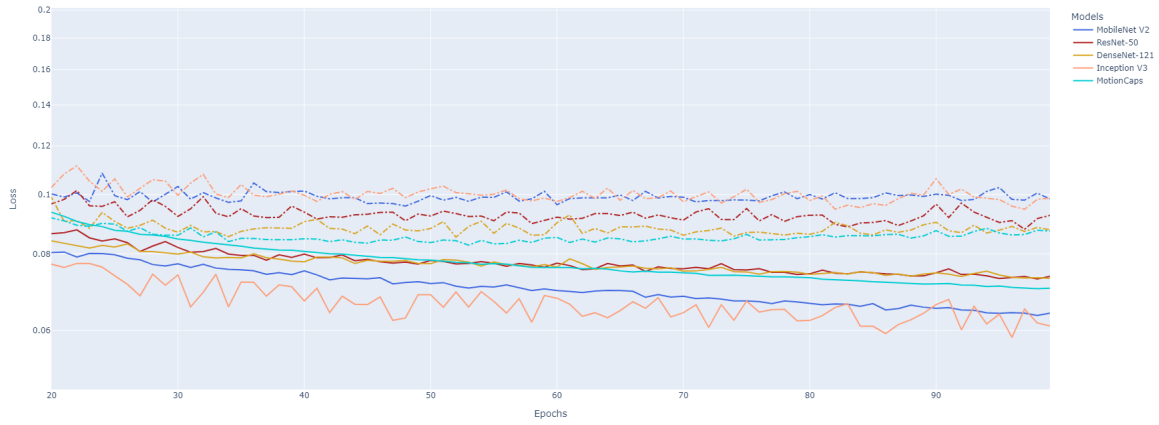

Figure 5.5: The training (solid line) and validation loss (dash line) of each model starting from epoch 20 after each model hits some local minimum demonstrated the stability of each model as well as a point of where overfitting starts to emerge.

5.4.3 Mode Collapse

As demonstrated throughout this chapter the motion detector that introduces the usage of semantic maps as well as the novel spatial feature extractor based on a capsule network yields promising results. Nevertheless, the main issue still remains, the proposed model can only predict a single possible trajectory given the data about past movement of the observed agent which greatly limits its potential. In addition, its main limitation i.e. deterministic prediction of a single motion, can also lead to a possibility of what is known as a mode collapse where during optimization a model's weights are adjusted in such a way that its ability to predict more than a small set of output types is greatly restricted.

First, in order to gain greater insight into the model's performance its loss over the entire test set, over three different future time horizons (4,5 and 6 seconds) is closely examined. Figure 5.7 demonstrates initial results of the experiment where three box plots reveal the distribution of loss obtained by running MotionCaps over the test set. As can be seen in the figure each distribution demonstrates a comparatively

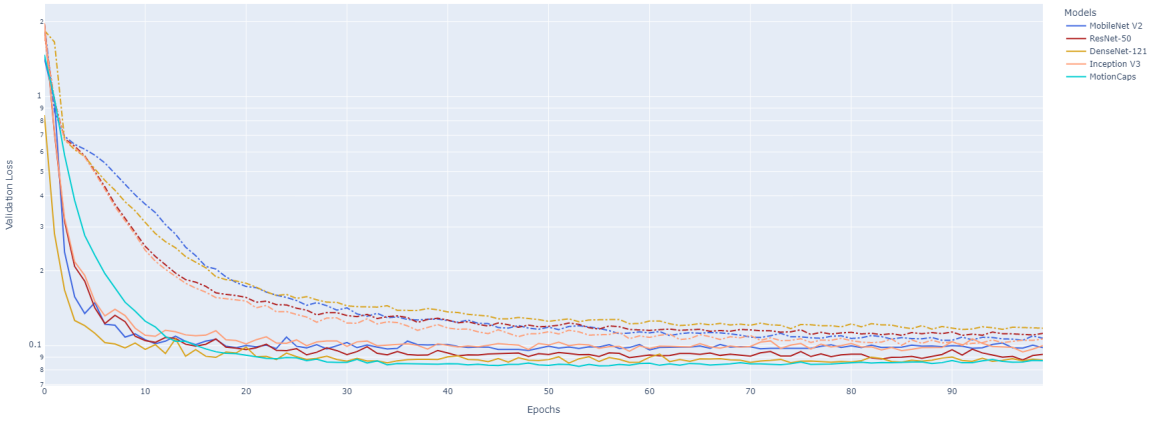

Figure 5.6: The validation loss of each model and its pre-trained variation (dashed line) over 100 epochs.

similar pattern of being right skewed where a large number of samples (up to the third quartile) yields a relatively small loss. However, it can also be clearly seen that there is a large number of outliers that strongly deviate from the mean of the distribution.

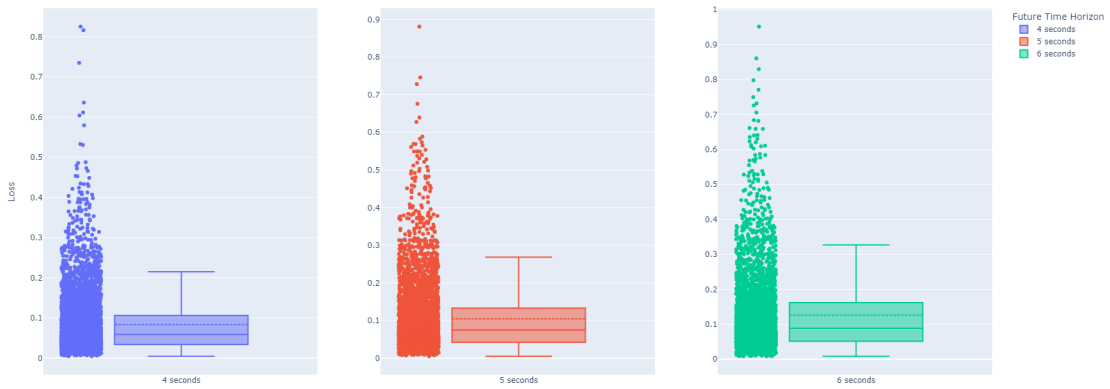

Figure 5.7: The distribution of the loss computed with MotionCaps over three different future time windows with test set. The plot presents the box plot as well as all individual data points along x-axis which provides an insight into how spread those loss values are for each individual time-step.

The distribution of loss values depicted in 5.7 indicates overall stability in the model's

performance during training. However, the presence of outliers suggests instances where the model struggles to make accurate predictions, potentially due to complex or rare motion patterns. Addressing these outliers could lead to further improvements in the model's robustness and generalization capabilities. It is therefore highly likely that there exists a pattern or patterns of motion that the model is either struggling to learn, or a general set of patterns that the model focused too strongly on and is hence only able to make an accurate prediction when such patterns occur.

To further confirm this hypothesis, the test motion samples that were obtained by running the model over the entire test set are sorted with respect to their loss value. Then, a set of $n = 150$ easy (n samples with lowest loss) and a set of hard (n samples with highest loss) ground truth samples per three future time horizons (4,5 and 6 seconds) are visualised to determine differences with respect to the motion patterns that yield lowest and highest loss. As can be seen in figure 5.8 where various trajectories are being visualized there is a significant difference with respect to motion patterns between easy and hard samples over all three time horizons. More specifically, it appears that all samples that present little difficulty to the model seem to follow a notably similar, linear pattern of agents moving in a straight motion. In addition, looking closely at those patterns one can notice that the velocity of the tracked agents also remains relatively stable, thus making the prediction significantly simpler. Furthermore, observing the visualised patterns that are in fact difficult for the model to predict correctly it can be clearly noticed that the space of those movements is significantly more difficult, containing patterns that are highly non-linear. Moreover, looking closely it can also be observed that the velocity within those movements varies considerably.

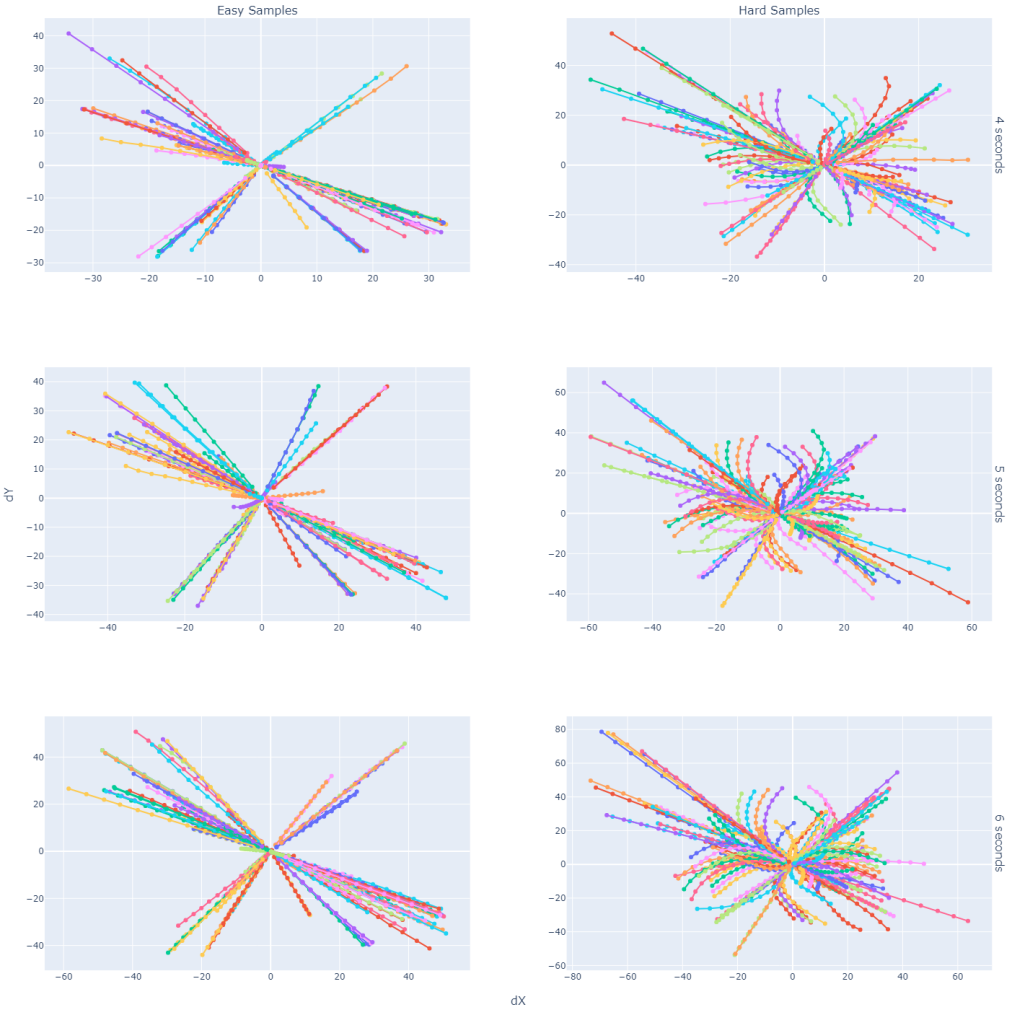

Figure 5.8: Visualisation of 150 easy (left columns) and hard (right columns) ground truth motion samples form test set over three future time horizons (defined on the right side of y-axis).

5.5 Conclusion

In this chapter a novel method for extracting features from rasterised map data based a on capsule network has been proposed and thoroughly examined. First and foremost, an initial ablation study has been conducted in order to establish the initial performance of the proposed model.

A comparison experiment between a basic CNN based motion predictor from section 4.4.4 as well as MotionCaps and its basic variants has been conducted. The results

presented in the discussed section demonstrated that the model with the proposed capsule based feature extractor did indeed outperform a model that instead used the basic CNN backbone. In addition, through further ablation study it was also established that the model can also benefit from the usage of local, disentangled semantic maps (each describing a different category of the road layer e.g. walkway, drivable area) assuming that each semantic layer is encoded with a dedicated capsule layer.

Moreover, MotioCaps model has been additionally compared against a set of motion predictors with various, popular CNN backbones from the literature such as ResNet, MobileNet, Inception and DenseNet both by training the chosen networks with randomly initialized weights as well as with the use of weights that were pre-trained on ImageNet. The practice of using pre-trained models has been generally adopted as a de facto method within the computer vision domain, however, as argued in section 5.2 this is not often beneficial if the source domain within which the model was pre-trained strongly deviates from the target domain, especially when the task at hand is highly sensitive towards a localization of small spatial features. As then demonstrated in subsection 5.4.2 models that were trained from scratch achieved substantially better performance indicating that their counterparts that were initially pre-trained on e.g. the classification task are less suited for learning of salient features from rasterised maps. In addition, the comparison of MotionCaps against motion detectors that employed one of the popular CNN architecture demonstrated that the capsule based predictor not only accomplished comparable results against other models, but also managed to reach its performance using considerably less parameters and with a more stable training.

Finally, limitations of the proposed motion predictor were investigated to determine model's modes of failure. First of all, the MotionCaps model was used to evaluate the NuScene test set and gather statistics with respect to the loss over the entire

test set across three different prediction horizons. From the demonstrated results it is clear that the general performance for all future time horizons remains relatively stable, however, there still exists a large amount of samples that do cause the model to fail considerably yielding high prediction error. Ultimately, the test samples that produced the lowest and highest prediction error revealed that the model has collapsed during the training and is strongly biased towards accurate prediction of linear motion. Nevertheless, despite limitations that were discovered during the final analysis within this chapter the proposed capsule based motion predictor, MotionCaps, has managed to demonstrate satisfactory performance, surpassing other methods that were used during the comparison analysis. In the next chapter the MotionCaps model will be extended further in order to advance its functionality with an introduction of generative models to account for the uncertain, multi-modal nature of the task.

Chapter 6

Introducing Stochasticity for a Multi-modal Motion

6.1 Introduction

The previous chapter explored and employed a novel architecture for predicting short-term future motion of tracked agents from the perspective of autonomous vehicles. The proposed model utilized a convolutional branch based on capsule networks in order to process a series of local, disentangled semantic layers which greatly increased its final performance. Nevertheless, the problem of forecasting short-term motion of nearby vehicles presents an inherently challenging issue as the space of agents' possible future movements is not strictly limited to a set of single trajectories. The technique proposed in the previous chapter as well as number of recently proposed methods that demonstrate plausible results concentrate primarily on forecasting the fixed number of deterministic predictions (Cui et al. 2019), or on classifying over a wider variety of trajectories that were previously generated using for example a dynamic model (Phan-Minh et al. 2020). This set of approaches however, is not sufficient for the problem at hand as it is highly unlikely for agents

to continuously follow uniform, deterministic movement in any given environment.

One way to address the discussed issue is to introduce stochasticity into the prediction process, thereby enabling the model to vary its output. Therefore, in this chapter the issue of multi-modal motion prediction will be tackled by leveraging benefits of generative models, in particular conditional variational auto-encoders that approximates a posterior distribution of future trajectories by conditioning the decoder on the past data of a tracked agent. As previously discussed, a number of recently suggested approaches have explored the use of various DL methods to address the discussed task, however, these still manifest several drawbacks such as the complete lack of multi-modality, or in a case where the proposed model does exhibit the ability to output numerous predictions given an input, it is still often limited to several deterministic predictions. Variational auto-encoders (VAE) (Kingma & Welling 2013) on the other hand come from the family of generative models and are thus not restricted to a particular set of deterministic predictions.

6.2 Conditional Variational Auto-encoders

6.2.1 Auto-encoders

First, in order to gain a better understanding of variational autoencoders it is essential to introduce the notion of autoencoders which are types of neural networks trained in an unsupervised fashion that aim to learn the compressed representation of the data in order to then reconstruct it. More specifically, an autoencoder, first introduced by McClelland et al. (1987) can be perceived as a neural network that is build of two primary parts (see Figure: 6.1) i.e. an encoder network $c = f(x)$ that takes an input x and produces its compressed representation c, as well as a decoder network $\hat{x} = g(c)$ that aims to reconstruct the input x from its compressed

latent representation c such that $\hat{x} \approx x$ (Goodfellow et al. 2016). Autoencoders were traditionally used within the area of dimensionality reduction and in a case where an autoencoder does not use non-linear activations it can learn a similar latent representation of the data as *Principal Component Analysis* (PCA) (Hotelling 1933). Thus, those types of networks with non-linear activation functions can be further considered as a generalization of PCA that learns a complex non-linear manifold of the data as opposed to the low dimensional hyperplane that PCA aims to find.

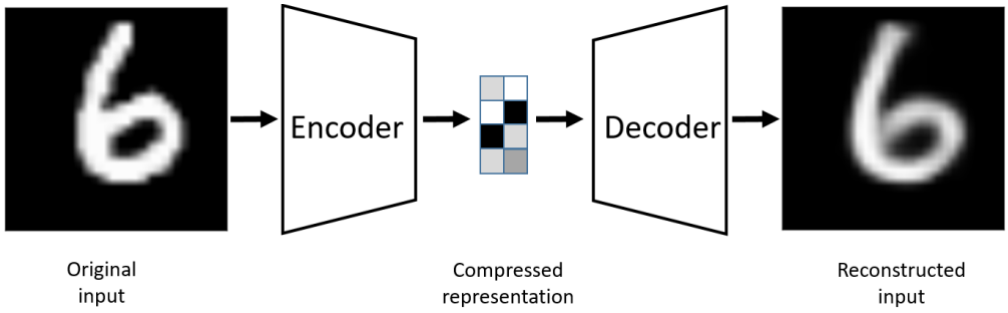

Figure 6.1: An example of an autoencoder. An original input image is processed through the encoder to produce a compressed representation of an image, then a decoder takes this latent representation and aims to decode it such that the output resembles the original input (Bank et al. 2020).

The use of autoencoders varies, as previously mentioned those types of networks can be used for example for data compression, however, their usage has also been found to be particularly beneficial for retrieval of corrupted data. In such case, the autoencoder is generally referred to as denoising auto-encoder (DAE) (Vincent et al. 2008). As an example, let x be the original input data and let $\tilde{x}$ be a corrupted version of x through an addition of random noise e.g. $\tilde{x} = x + n$ where $n \sim \mathcal{N}(\mu, \sigma^2)$. Then, the aim of the denoising autoencoder is to minimize some objective function $\mathcal{J}(x, g(f(\tilde{x})))$ such that the resulting network can remove the noise from the input as opposed to simply learning to copy the input as done by conventional autoencoders as demonstrated in figure 6.2.

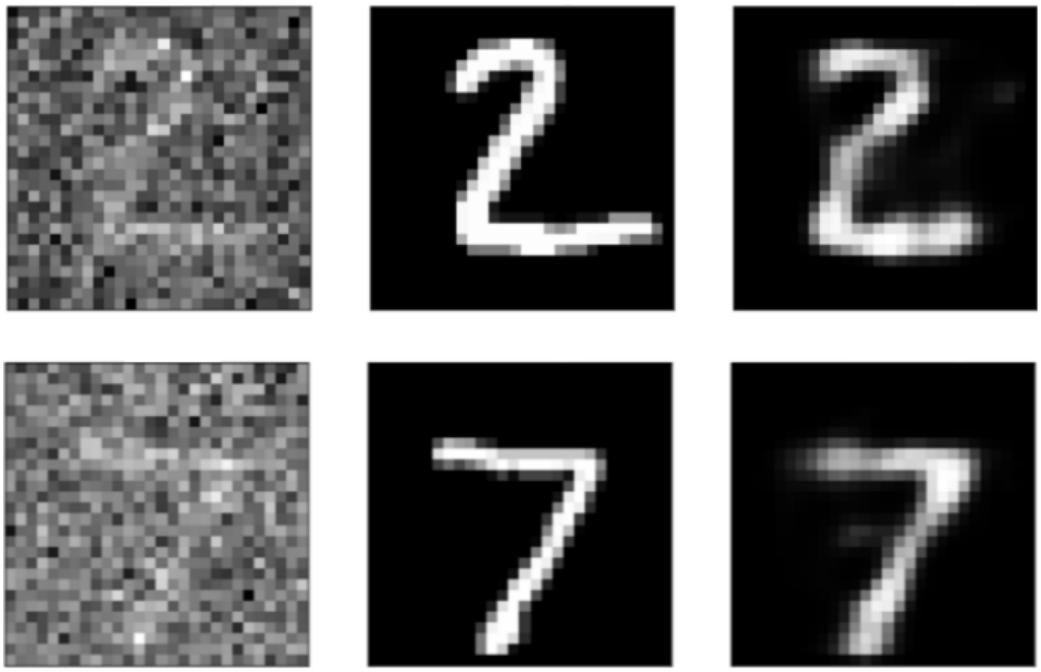

Figure 6.2: An example of results obtained from training the denoising autoencoder. The noisy input to the network is presented on the left with the original data in the centre an a denoised output of the model on the right (Jordan 2018).

6.2.2 Variational Auto-encoders

Variational auto-encoders which are considered probabilistic generative models, extend the idea of conventional, often deterministic auto-encoders by aiming to constrain the learning process to learn parameters of a probability distribution (often Gaussian) that models the data instead of learning an arbitrary mapping function. A simple VAE at its core just like the conventional auto-encoder is constructed of a sequence of two part neural networks, an encoder as well as decoder which in a case of VAE are often referred to as recognition and generative networks (or models) respectively. Figure 6.3 demonstrates a simple example of the VAE framework.

To further understand the core idea behind VAE let $X = \{x^i\}_{i=1}^N$ denote a dataset with N i.i.d. samples where x^i can be either continuous or discrete. Furthermore, it is assumed that the data is generated by some random process which involves a latent continuous random variable z^i generated from a prior distribution $P_\theta(z)$ with

the random data sample x being generated from a conditional distribution over z i.e. $P_\theta(x \mid z)$. Therefore, the initial objective is to be able to sample from the well representative latent space z given the observed data x which requires calculating the posterior probability distribution $P_\theta(z \mid x)$ which using Bayes' rule is further defined as:

$$P_\theta(z \mid x) = \frac{P_\theta(x, z)}{P_\theta(x)} = \frac{P_\theta(x \mid z)P_\theta(z)}{P_\theta(x)} \tag{6.1}$$

where

$$P_\theta(x) = \int P_\theta(x, z) = \int P_\theta(x \mid z)P_\theta(z)dz \tag{6.2}$$

However, $P_\theta(x)$ and therefore the posterior $P_\theta(z \mid x)$ is intractable due to all possible configurations of the continuous latent variable z. Hence, in order to turn this into a tractable problem a VAE framework introduces an encoder (recognition) network, denoted as $Q_\phi(z \mid x)$ where ϕ includes weights and biases of the encoder, which aims to approximate parameters of the true posterior distribution $P_\theta(z \mid x)$ such that:

$$Q_\phi(z \mid x) \approx P_\theta(z \mid x) \tag{6.3}$$

For example, if $P_\theta(z \mid x)$ is a Gaussian distribution then the encoder $Q_\phi(z \mid x)$ will be trained such that it can estimate parameters μ, σ^2 given x, thus:

$$Q_\phi(z \mid x) = \mathcal{N}(z \mid \mu(x, \phi), \sigma^2(x, \phi)) \tag{6.4}$$

To train an encoder network a non symmetrical measure of the similarity between two probability distributions is being used, more precisely a Kullback-Leibler divergence D_{KL} is defined as:

$$D_{KL}(Q_\phi(z \mid x) \mid\mid P_\theta(z \mid x)) = \mathbb{E}_{Q_\phi}\left[\log \frac{Q_\phi(z \mid x)}{P_\theta(z \mid x)}\right] \tag{6.5}$$

which has the property of being non-negative $D_{KL}(Q_\phi(z \mid x) \mid\mid P_\theta(z \mid x)) \geq 0$ and

zero if, and only if, $Q_\phi(z \mid x)$ equals the true posterior distribution. By further factoring Kullback-Leibler divergence and applying Bayes' rule to the $P_\theta(z \mid x)$ the D_{KL} becomes:

$$\begin{aligned} D_{KL}(Q_\phi(z \mid x) \mid\mid P_\theta(z \mid x)) &= \mathbb{E}_{Q_\phi}\left[\log\frac{Q_\phi(z \mid x)}{P_\theta(z \mid x)}\right] \\ &= \mathbb{E}_{Q_\phi}[\log Q_\phi(z \mid x)] - \mathbb{E}_{Q_\phi}[\log P_\theta(z \mid x)] \\ &= \mathbb{E}_{Q_\phi}[\log Q_\phi(z \mid x)] - \mathbb{E}_{Q_\phi}\left[\log\frac{P_\theta(z,x)}{P_\theta(x)}\right] \\ &= \mathbb{E}_{Q_\phi}[\log Q_\phi(z \mid x)] - \mathbb{E}_{Q_\phi}[\log P_\theta(z,x)] + \mathbb{E}_{Q_\phi}[\log P_\theta(x)] \\ &= \mathbb{E}_{Q_\phi}[\log Q_\phi(z \mid x)] - \mathbb{E}_{Q_\phi}[\log P_\theta(z,x)] + \log P_\theta(x) \end{aligned} \tag{6.6}$$

The goal is to find the variational parameters ϕ of the encoder that minimize the divergence between the approximated and true posterior:

$$Q^*_\phi(z \mid x) = \arg\min_\phi D_{KL}(Q_\phi(z \mid x) \mid\mid P_\theta(z \mid x)) \tag{6.7}$$

However, due to the intractable nature of $P_\theta(x)$ the $D_{KL}(Q_\phi(z \mid x) \mid\mid P_\theta(z \mid x))$ cannot be computed directly. Instead, an alternative objective function known as *evidence lower bound* (ELBO) (or *variational lower bound*) can be used during optimization:

$$\text{ELBO}_{\theta,\phi}(x) = \mathbb{E}_{Q_\phi}[\log P_\theta(x,z)] - \mathbb{E}_{Q_\phi}[\log Q_\phi(z \mid x)] \tag{6.8}$$

Combining this further with the D_{KL} allows to rewrite the evidence as:

$$\log P_\theta(x) = \text{ELBO}_{\theta,\phi}(x) + D_{KL}(Q_\phi(z \mid x) \mid\mid P_\theta(z \mid x)) \tag{6.9}$$

and since D_{KL} is always equal or greater than 0 it means that maximizing ELBO is equivalent to minimizing the D_{KL}. Interestingly, as stated by Kingma et al. (2019) the optimization of ELBO will approximately maximize the marginal likelihood

$P_\theta(x)$ and minimize the D_{KL} between the approximated and true posterior, thus, leading to a better generative model.

Furthermore, given the latent output of the stochastic encoder $Q_\phi(z \mid x)$ the next part is to train the probabilistic decoder (generative network) $P_\Phi(x \mid z)$ to reconstruct the input data such that:

$$P_\Phi(x \mid z) \approx x \tag{6.10}$$

The choice of loss function for the decoder is strongly dependant on the task at hand, however, a simple mean squared error is a frequent choice for learning to reconstruct the data:

$$\mathcal{J}_{decoder} = \|P_\Phi(x \mid z) - x\|^2 \tag{6.11}$$

6.2.3 Conditional Variational Auto-encoders

In the previous subsection a probabilistic generative model named variational auto-encoder was introduced. As demonstrated, the VAE establishes a reliable framework for learning parameters of probability distribution and allowing to then sample from the learned distribution in order to generate a plausible output. Nevertheless, the VAE is inherently limited as the decoder cannot be controlled to produce a specific output. For example, given a MNIST dataset of handwritten digits (LeCun et al. 1998) the task with use of the VAE would involve learning the probability distribution such that when sampled from the distribution the decoder would be able to reconstruct the handwritten digits. However, in such case the latent input to the decoder would carry no information as to which digit the decoder should aim to reconstruct, and thus the final output would be a randomly generated digit.

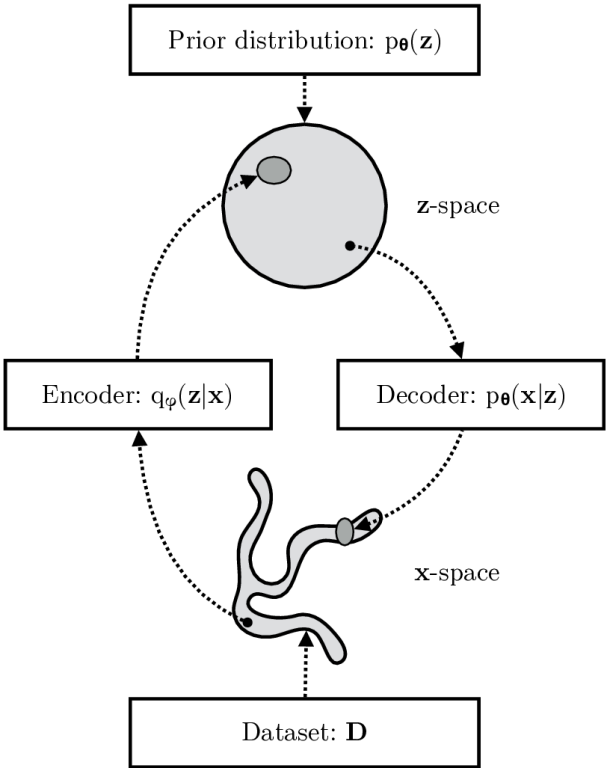

Figure 6.3: The figure by Kingma et al. (2019) demonstrates the mapping process that a VAE is aiming to learn (note that the notation in the figure might differ from the notation used in this subsection). A probabilistic encoder $Q_\phi(z \mid x)$ is trained to map the input x to the latent variable z. Then, a stochastic decoder $P_\Phi(x \mid x)$ is trained to reconstruct the input data x given a latent input z.

To mitigate this issue and allow for a more controlled process of generating data, a further extension of VAE was proposed, namely a conditional variational auto-encoder which learns to encode and decode the latent space based not solely on the input x and latent variable z but also on some random variable y as depicted in figure 6.4.

More specifically, the encoder is now modelled as $Q_\phi(z \mid x, y)$ and the decoder as $P_\Phi(x \mid z, y)$ where for instance, in a case of training to generate the MNSIT dataset the y could represent the specific digit (often encoded with one hot encoding) that the model should aim to output. Moreover, the evidence now becomes $\log P_\theta(y \mid x)$ and is further formulated as:

$$\log P_\theta(y \mid x) = \text{ELBO}_{\theta,\phi}(x, y) + D_{KL}(Q_\phi(z \mid x, y) \mid\mid P_\theta(z \mid x, y)) \tag{6.12}$$

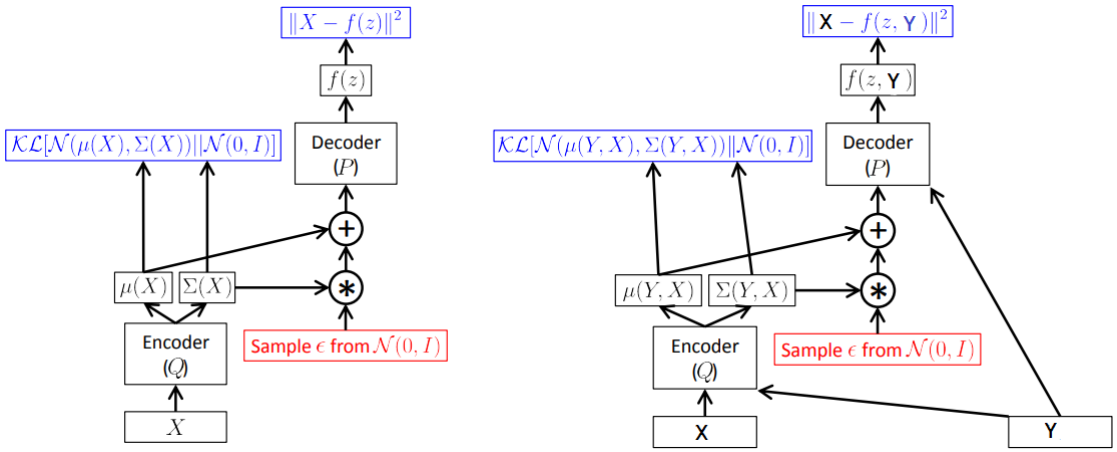

Figure 6.4: A graphical comparison of the VAE (left) and CVAE (right). As depicted, the CVAE model now considers an additional input y which conditions the encoder and decoder accordingly to a given label y (Doersch 2016).

with the variational lower bound from equation 6.8 now being rewritten as:

$$\text{ELBO}_{\theta,\phi}(x, y) = \mathbb{E}_{Q_\phi}[\log P_\theta(y, z \mid x)] - \mathbb{E}_{Q_\phi}[\log Q_\phi(z \mid x, y)] \tag{6.13}$$

As demonstrated in this section, the VAE framework (as well as its extension towards CVAE) proposes an elegant solution for tackling the lack of multi-modality in numerous tasks, among them a prediction of future motion of on-road agents. Therefore, the next section will explain the proposed architecture of the mulit-modal motion predictor based on the CVAE framework.

6.3 Network Architecture and Computational Flow

This section provides an overview of the architecture as well as the computational flow of the proposed network that aims to address the main issue from the previous chapter, the lack of multi-modal output.

First, as in previous chapters, it is assumed that the vehicle equipped with motion prediction capabilities is equipped with an appropriate tracking module that for re-

trieval of information that corresponds to the past state of the tracked agent, and in addition it is assumed that additional data in form of HD maps is also available. First, let τ, ρ denote time-steps for the prediction and observed time horizon respectively. Next, let $\boldsymbol{S} = [\boldsymbol{s}_{t-\rho}, \cdots, \boldsymbol{s}_{t-1}, \boldsymbol{s}_t]$ represent a standardised matrix of the tracked agent's motion state from time-step $t-\rho$ to the initial time-step t where $\boldsymbol{s}_t = [v, a, \Delta\vartheta]$ denotes a vector of scalar features containing velocity, acceleration and heading change rate respectively. Moreover, let $\boldsymbol{P} = [\boldsymbol{p}_{t-\rho}, \cdots, \boldsymbol{p}_{t-1}, \boldsymbol{p}_t]$ correspond to the agent's observed past positions, and $\boldsymbol{Y} = [\boldsymbol{y}_{t+1}, \boldsymbol{y}_{t+2}, \cdots, \boldsymbol{y}_{t+\tau}]$ to its future ground-truth positions with each vector in both matrices containing (x, y) coordinates in the agent's frame of reference.

Next, let the normalised matrix $\boldsymbol{M}$ define a global chunk of a rasterised HD map which captures 100 meters of the surrounding area in front of the vehicle and 5 meters from the rear. The matrix $\boldsymbol{M}$ is further extracted with accordance to the agent's position at the initial time-step t and rotated towards its frame of reference. Furthermore, a normalised tensor $\mathbf{D} = [\mathbf{L}_{t-\rho}, \cdots, \mathbf{L}_{t-1}, \mathbf{L}_t]$ is defined which encapsulates local map chunks $(20m \times 20m)$ with respect to the tracked agent. Then, each $\mathbf{L} \in \mathbf{D}$ is unraveled into separate semantic layers such that $\mathbf{L}_t = [\boldsymbol{L}_{t,0},\ \boldsymbol{L}_{t,1}, \cdots, \boldsymbol{L}_{t,n}]$ where n is equal to the number of road layers and $\boldsymbol{L}_{t,i}$ is equal to a sparse matrix that encapsulates spatial data of semantic layer of type i. An example of both global and local HD map data is presented in Fig 6.5.

The goal is to train a generative model based on the CVAE framework which is generally composed, as previously mentioned, as a three-part model with $Q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$ (recognition network), $P_\theta(\boldsymbol{z}|\boldsymbol{x})$ (conditional prior) and $P_\Phi(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ (generative network) parameterised by Φ, ϕ and θ with each part of the model being commonly defined as a MLP. In $P_\theta(\boldsymbol{z}|\boldsymbol{x})$ the latent variable $\boldsymbol{z}$ is conditioned on an arbitrary input $\boldsymbol{x}$, however, this can be further relaxed such that $P_\theta(\boldsymbol{z}|\boldsymbol{x}) = P_\theta(\boldsymbol{z})$ therefore making $\boldsymbol{z}$ statistically independent (Kingma et al. 2014). Finally, the main interest

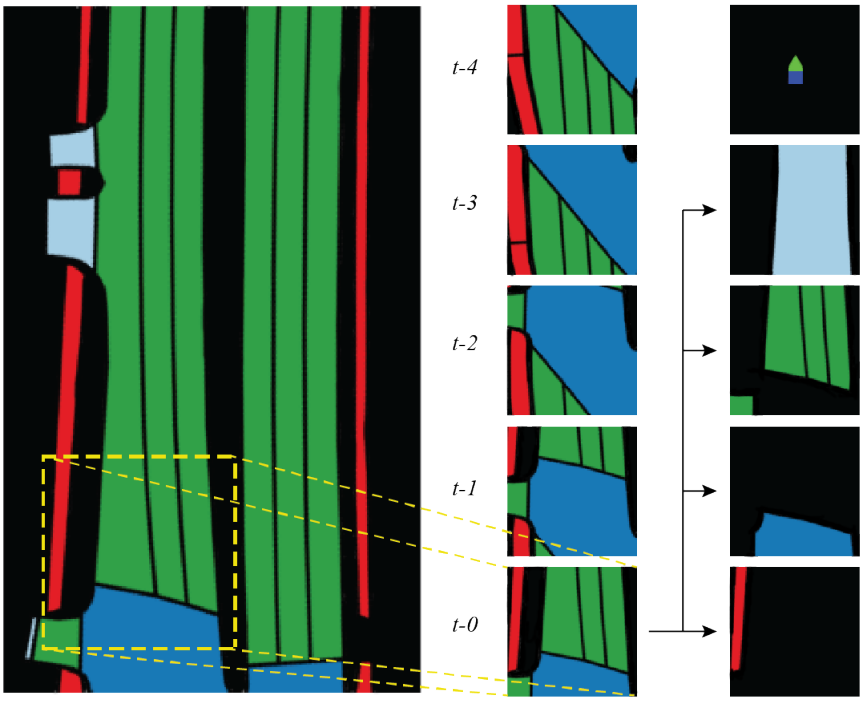

Figure 6.5: An example of a global map $\boldsymbol{M}$ (left) as well as the unraveling process of a local map chunk $\mathbf{L}_t$ (middle) which is turned into several geometrical layers (right) each representing a single part of the whole map. In addition, we draw an agent (top right) with its origin corresponding to the position on the chunk of interest.

is to use the trained generator to predict a diverse set of k trajectories with respect to a tracked entity which is denoted as $\hat{\mathbf{Y}} = [\hat{\boldsymbol{Y}}_0, \hat{\boldsymbol{Y}}_1, \cdots, \hat{\boldsymbol{Y}}_k]$ with each matrix $\hat{\boldsymbol{Y}}_i$ being further constructed as $\hat{\boldsymbol{Y}}_i = [\hat{\boldsymbol{y}}_{i,t+1}, \hat{\boldsymbol{y}}_{i,t+2}, \cdots, \hat{\boldsymbol{y}}_{i,t+\tau}]$ where $\hat{\boldsymbol{y}}_{i,t+1}$ corresponds to a vector containing future coordinates of the trajectory i at time $t+1$ such that $\hat{\boldsymbol{y}}_{i,t+1} = (\hat{x}_{i,t+1}, \hat{y}_{i,t+1})$ in the agent's frame of reference.

6.3.1 State Encoding

The basis of the network for encoding the captured state of the agent follows those from the previous chapter. More specifically, the set of local semantic maps $\mathbf{D}$ is encoded through the capsule encoder (for details see Section 5.3). A single layer $\boldsymbol{L}_{t,i} \in \mathbf{L}_t$ at time t is passed through the feature extractor to extract the final latent representation vector of each semantic map $\mathbf{L}$ over the observed time horizon. The

matrix $\boldsymbol{L}_{t,i}$ is passed through the convolutional base with *Leaky ReLU* non-linearity (Maas et al. 2013) (with the default value for the negative slope of $1e-2$) such that $\hat{\mathbf{L}}_{t,i} = \Phi_{\text{l\_base}}(\boldsymbol{L}_{t,i})$ which results in the tensor $\hat{\mathbf{L}}_{t,i}$ which encapsulates activations of local low-level features. The extracted features are then passed through lower capsules $\hat{\boldsymbol{L}}_{t,i} = \Phi_{\text{l\_lower}}(\hat{\mathbf{L}}_{t,i})$ which results in the output matrix $\hat{\boldsymbol{L}}_{t,i}$ that contains m capsules where each capsule is an $n-$dimensional vector with entries corresponding to instantiation parameters of detected features e.g. lines. As in the previous chapter, instead of using scalar based activation function, a squashing non-linearity (see Equation: 5.2) is used instead. The output of the $\Phi_{\text{l\_lower}}(\cdot)$ is further encoded with its respective higher capsule layer i such that $\hat{\boldsymbol{l}}_{t,i} = \Phi_{\text{l\_higher\_i}}(\hat{\boldsymbol{L}}_{t,i})$ with vector $\hat{\boldsymbol{l}}_{t,i}$ encapsulating encoded parameters of the layer $\boldsymbol{L}_{t,i}$. The above operation is further repeated for each $\boldsymbol{L}_{t,i} \in \mathbf{L}_t$ with the final encoding of each layer being concatenated to create an input matrix for the final capsule:

$$\hat{\boldsymbol{l}}_t = \Phi_{\text{l\_final}}(\Phi_{\text{concat}}(\hat{\boldsymbol{l}}_{t,0}, \hat{\boldsymbol{l}}_{t,1}, \cdots, \hat{\boldsymbol{l}}_{t,n-1})) \tag{6.14}$$

where $\hat{\boldsymbol{l}}_t$ encodes the final representation of $\mathbf{L}_t$. Repeating this process for each $\mathbf{L}_t \in \mathbf{D}$ yields matrix $\hat{\boldsymbol{L}}$ whose entries encapsulate encoding of all semantic layers from time $t-\rho$ to t such that $\hat{\boldsymbol{L}} = [\hat{\boldsymbol{l}}_{t-\rho}, \cdots, \hat{\boldsymbol{l}}_{t-1}, \hat{\boldsymbol{l}}_t]$.

Furthermore, in order to obtain the final encoding of the agent's state the matrix $\hat{\boldsymbol{L}}$ is concatenated with corresponding elements from motion state matrix $\boldsymbol{S}$:

$$\hat{\boldsymbol{s}}_t = \Phi_{\text{concat}}(\hat{\boldsymbol{l}}_t, \Phi_{\text{state}}(\boldsymbol{s}_t)), \quad \forall\, \hat{\boldsymbol{l}}_t \in \hat{\boldsymbol{L}}, \boldsymbol{s}_t \in \boldsymbol{S} \tag{6.15}$$

to create encoded state matrix $\hat{\boldsymbol{S}} = [\hat{\boldsymbol{s}}_{t-\rho}, \cdots, \hat{\boldsymbol{s}}_{t-1}, \hat{\boldsymbol{s}}_t]$ where $\Phi_{\text{state}}(\cdot)$ is a single fully-connected layer with Leaky ReLU. Lastly, the matrix $\hat{\boldsymbol{S}}$ is passed through a LSTM layer to compute the final state representation in a temporal manner $\hat{\boldsymbol{s}} = \Phi_{\text{state\_lstm}}(\hat{\boldsymbol{S}})$ with vector $\hat{\boldsymbol{s}}$ corresponding to $\Phi_{\text{state\_lstm}}(\cdot)$ final hidden-state.

6.3.2 Recognition Network

In addition to the computed encoding of the tracked agent's state $\hat{\boldsymbol{s}}$ as described in the previous subsection, the scene context $\boldsymbol{M}$ as well as an agent's ground-truth future $\boldsymbol{Y}$ and past motion $\boldsymbol{P}$ must also be encoded. First of all, the surrounding global context $\boldsymbol{M}$ is passed through another, independent feature extractor in a similar manner as $\mathbf{D}$ such that:

$$\hat{\boldsymbol{m}} = \Phi_{\text{m\_higher}}(\Phi_{\text{m\_lower}}(\Phi_{\text{m\_base}}(\boldsymbol{M}))) \tag{6.16}$$

The feature extractor is based on the capsule network proposed in the previous chapter with a slight modification as can be noted by looking at the equation 6.16. Instead of reusing the same architecture, the feature extractor is pruned such that it only relies on the convolutional base as well as the lower and higher capsules. The final capsule layer has been omitted due to the fact that the rasterised scene context $\boldsymbol{M}$ is a single image as opposed to multiple images as in a case of local semantic layers $\mathbf{L}_t$.

Next, both $\boldsymbol{Y}$ and $\boldsymbol{P}$ are flattened along their temporal dimensions to create vectors $\boldsymbol{g}$ (future) and $\boldsymbol{p}$ (past) of shape $2 \times \tau$ and $2 \times \rho$ respectively. Both vectors through their respective fully-connected layers with Leaky ReLU to get $\hat{\boldsymbol{g}} = \Phi_{\text{gt}}(\boldsymbol{g})$ and $\hat{\boldsymbol{p}} = \Phi_{\text{past}}(\boldsymbol{p})$ (initial experiments demonstrated no improvement when encoding $\boldsymbol{Y}$ and $\boldsymbol{P}$ through LSTM layers and therefore it was decided to use fully-connected networks instead to account for computational efficiency).

The recognition network is further defined in the CVAE framework as $Q_\Theta(\boldsymbol{z}|\hat{\boldsymbol{g}}, \boldsymbol{c})$ where Q_Θ is constructed as a two-headed MLP each composed of two fully-connected layers with Leaky ReLU and batch normalisation (Ioffe & Szegedy 2015), and the vector $\boldsymbol{c}$ is the result of the following $\boldsymbol{c} = \Phi_{\text{concat}}(\hat{\boldsymbol{p}}, \hat{\boldsymbol{m}})$. Q_Θ aims to approximate the prior distribution $P_\phi(\boldsymbol{z})$ which is modeled as a Gaussian such that $P_\phi(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

by predicting distribution parameters i.e. $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$ of a latent variable $\boldsymbol{z}$ which results in $\boldsymbol{z} \sim Q_\Theta(\boldsymbol{z}|\hat{\boldsymbol{g}}, \boldsymbol{c}) = \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}})$ and leads to learning of the hidden representation of $\hat{\boldsymbol{g}}$ given $\boldsymbol{c}$.

6.3.3 Motion Generator

The generation network which is also referred to as motion generator is formulated as $P_\theta(\hat{\boldsymbol{Y}}|\boldsymbol{z}, \boldsymbol{c})$ and defined as an MLP with four fully-connected layers and Leaky ReLU non-linearity in-between layers. In addition to using $\boldsymbol{z}$ and $\boldsymbol{c}$ during the generation process, the agent's encoded state $\hat{\boldsymbol{s}}$ is also added as an additional intermediate input to the second fully-connected layer. The final layer of the generator yields a vector of size $2(x, y) \times \tau$ where τ corresponds to the future time horizon. The output of the generator is finally reshaped to create target matrix $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_{t+1}, \hat{\boldsymbol{y}}_{t+2}, \cdots, \hat{\boldsymbol{y}}_{t+\tau}]$ with each entry vector containing predicted future coordinates (x, y) in the agent's coordinate frame from time $t+1$ to $t+\tau$. During the training phase the latent variable $\boldsymbol{z}$ is sampled from Q_Θ i.e. $\boldsymbol{z} \sim Q_\Theta(\boldsymbol{z}|\hat{\boldsymbol{g}}, \boldsymbol{c})$, however, during testing the encoding of ground-truth motion $\hat{\boldsymbol{g}}$ is not available and therefore $\boldsymbol{z}$ is directly sampled from the prior distribution $\boldsymbol{z} \sim P_\phi(\boldsymbol{z})$. Note that generating k samples can be performed by sampling $\boldsymbol{z}$ from the prior for k times to create $\boldsymbol{Z} = [\boldsymbol{z}_0, \boldsymbol{z}_1, \cdots, \boldsymbol{z}_k]$ with $\boldsymbol{z}_i$ corresponding to the latent sample at index i, and then decoding the i^{th} motion sequence such that:

$$\hat{\boldsymbol{Y}}_i = P_\theta(\hat{\boldsymbol{Y}}_i|\boldsymbol{z}_i, \boldsymbol{c}) \quad \forall\ \boldsymbol{z}_i \in \boldsymbol{Z} \tag{6.17}$$

The summary of the proposed model's architecture is demonstrated in figure 6.6.

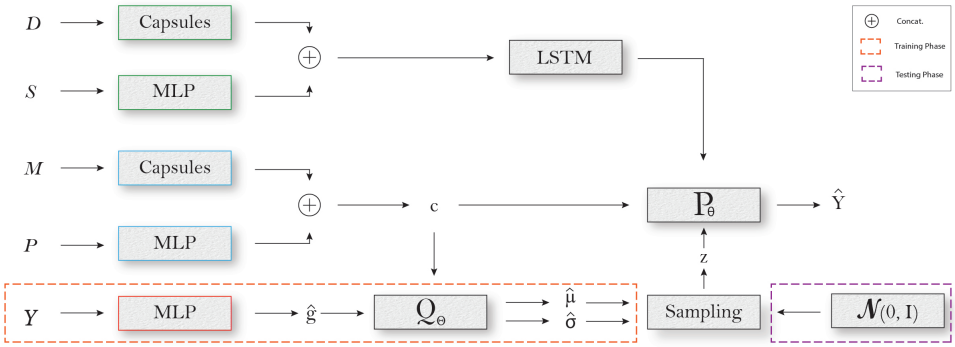

Figure 6.6: A simplified overview of the proposed network. During training parameters of the prior distribution are obtained with the recognition network which is used to produce a diverse set of training samples. Then, during testing the past ground-truth motion $\boldsymbol{Y}$ of a tracked agent is not available and z is therefore sampled directly from prior i.e. $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then used to decode k samples which are encapsulated in $\hat{\mathbf{Y}}$.

6.4 Experiments and Results

This section will focus on exploring and evaluating an extension of the deterministic model proposed in the previous chapter by extending it via the CVAE framework to account for multi-modality of the problem that is predicting future motion of agents in complex scenes. First of all, an ablation study will be conducted between a motion predictor based on a standard VAE against a CVAE with various modes of $\boldsymbol{c}$ to examine a) whether CVAE does actually presents any benefits by conditioning the decoder and b) what sort of condition yields lowest error. The architecture of the standard VAE and the CVAE follows a similar pattern, with the main difference being the conditional prior. Note, throughout the remainder of this chapter the proposed CVAE model will be referred to as **MMST** (multi-modal stochastic trajectories).

6.4.1 Ablation Study - VAE vs CVAE

The experimental setup for the following experiments follows a similar pattern as in previous chapters with the main difference being the loss function used for the

training and the metric used for evaluation. Both VAE and CVAE predictors are optimized by minimising the following loss function:

$$\mathcal{J} = \alpha\mathcal{J}_1 + \beta\mathcal{J}_2 \tag{6.18}$$

where $\mathcal{J}_1$ aims to minimise the distance between $Q_\Theta(\boldsymbol{z}|\hat{\boldsymbol{g}})$ (VAE) or $Q_\Theta(\boldsymbol{z}|\hat{\boldsymbol{g}}, \boldsymbol{c})$ (CVAE) as well as the prior $P_\phi(\boldsymbol{z})$ such that:

$$\begin{aligned} \mathcal{J}_1 &= -D_{KL}(Q_\Theta(\boldsymbol{z}|\hat{\boldsymbol{g}}, \boldsymbol{c})||P_\phi(\boldsymbol{z})) \\ &= -\frac{1}{2}\sum_{i=1}^{n}(1 + log((\hat{\boldsymbol{\sigma}}_i)^2) - (\hat{\boldsymbol{\mu}}_i)^2 - (\hat{\boldsymbol{\sigma}}_i)^2) \end{aligned} \tag{6.19}$$

and $\mathcal{J}_2$ serves as a reconstruction loss between ground truth and the prediction:

$$\begin{aligned} \mathcal{J}_2 &= MSE(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) \\ &= \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i^k\|^2 \end{aligned} \tag{6.20}$$

Both loss terms are balanced by setting $\alpha = 1$ and $\beta = 1e-2$. Furthermore, both types of networks are trained for up to 400 epochs with the early stopping, Adam optimizer and a batch size of 64. Next, for every data sample the observed time horizon is again set to 2 seconds (5 time-steps) of the agent's past states i.e. $\rho \equiv 2$ seconds and the prediction horizon is set to $\tau \equiv 6$ seconds of the agent's future motion. Moreover, the quantitative metrics are now adapted in order to reflect the nature of the generative models, more specifically, instead of considering only a single predicted sample both ADE and FDE now take into consideration k predictions such that:

$$\text{minADE}_k = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\min_k \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i^k\|^2} \tag{6.21}$$

and

$$\text{minFDE}_k = \frac{1}{n}\sqrt{\sum_{i=1}^{n} \min_k \|\boldsymbol{y}_{i,t+\tau} - \hat{\boldsymbol{y}}^k_{i,t+\tau}\|^2} \tag{6.22}$$

First of all, the performance of CVAE against VAE is examined with various modes of $\boldsymbol{c}$ in order to determine the most optimal combination of input data as well as to determine whether conditioning a generative model yields lower minADE and minFDE. For various modes of $\boldsymbol{c}$ the CVAE based networks are adjusted by adding/removing relevant layers w.r.t. input data.

Results of the initial experiment are presented in Table 6.1 where both metrics demonstrate the error with respect to a large number of samples ($k = 100$). As demonstrated in the table, there exists a significant performance difference between the VAE and CVAE based motion predictors of at least (0.03/0.06) and up to (0.18/0.30) for both minADE and minFDE. These findings unequivocally underscore the advantageous nature of conditioning the generative model on the latent variable $\boldsymbol{c}$, as facilitated by the CVAE framework.

Furthermore, the analysis of various modes of $\boldsymbol{c}$ demonstrates that the best combination is in fact the past motion of the observed agent as well as the rasterised global map that encapsulates significantly larger portion of surrounding than local maps. Moreover, it appears that utilising past motion provides the greatest improvement across the board which potentially stems from the fact that the past motion correlates strongly with the future movement of an agent. On the other hand, conditioning the motion generator solely with either global or local maps offers slightly worse improvement over the standard VAE.

In essence, the findings underscore the pivotal role of conditioning within the CVAE framework, emphasizing the nuanced interplay between past motion, global context, and local cues in shaping the predictive capabilities of the model. These insights not only advance the understanding of multi-modal motion prediction but also of-

fer practical implications for enhancing the performance of generative models in complex dynamic environments.

Generative Model	Condition $\boldsymbol{c}$			minADE/minFDE ($k = 100$)
	Past Motion	Global Map	Local Maps	
VAE				2.21/4.63
CVAE	✓			2.07/4.40
		✓		2.12/4.49
			✓	2.13/4.52
	✓	✓		**2.03/4.33**
		✓	✓	2.10/4.45
	✓	✓	✓	2.06/4.38

Table 6.1: Quantitative results of an ablation study between CVAE and VAE as well as between various modes of conditioning of the CVAE based model.

Next, further results presented in Table 6.2 demonstrate the performance of MMST based on both VAE and CVAE (with most optimal $\boldsymbol{c}$) from the previous experiment across the number of different k samples. In addition, the MotionCaps network introduced in the previous chapter has been also considered in order to further present the relative performance of deterministic and stochastic networks. Looking at the table it is clear that the CVAE based MMST presents the best results in comparison to other methods which further supports results from the previous experiment. However, what is interesting are the results of stochastic models when $k = 1$ in which case the best performing model seems to be the deterministic MotionCaps. Nonetheless, when observing the performance of stochastic models it can be noted that the model (MMST based on CVAE) with minimum error across both metrics and almost all k modes tend to perform worse on $k = 1$. This behaviour might signify that initially the variance w.r.t. plausible movement of the agent could be narrow and only encapsulate the most common motion patterns (hence lower minADE/minFDE for MotionCaps and MMST based on VAE) but with a better model the learned variance could be more suited towards predictions of more complex movement.

Model	minADE/minFDE			
	$k = 1$	$k = 20$	$k = 50$	$k = 100$
MotionCaps	**2.74/6.89**	-	-	-
MMST (VAE)	3.97/9.49	2.78/6.20	2.38/5.02	2.21/4.63
MMST (CVAE)	4.26/10.17	**2.49/5.58**	**2.17/4.72**	**2.03/4.33**

Table 6.2: Quantitative results of an ablation study between CVAE and VAE as well as between various modes of conditioning of the CVAE based model.

Lastly, it is crucial to analyze the inference time of the examined models with regards to the number of sampled trajectories to determine whether sampling with a higher rate introduces a speed bottleneck. Results of the experiment are presented in Figure 6.7 where three models considered in Table 6.2 were tested on $k \in \{1, 20, 50, 100, 500, 1000\}$ with the deterministic model (MotionCaps) being considered only on $k = 1$ for comparison purposes. As demonstrated, the MotionCaps is in fact fastest when predicting for a single sample with the inference speed of $55.02ms$ against VAE's $57.12ms$ and CVAE's $60.44ms$, which is in fact relatively negligible considering that MotionCaps is limited to predicting a single trajectory per given input. On the other hand, it is clear that CVAE is slightly slower than VAE across all modes of k whilst also having significantly higher variance. However, despite the varying complexities introduced by an increased number of sampled trajectories, the inference time remains relatively stable, indicating that scaling the number of samples does not necessarily incur a substantial speed penalty. This observation underscores the scalability and efficiency of the models, suggesting that a large number of samples can be obtained without compromising computational speed.

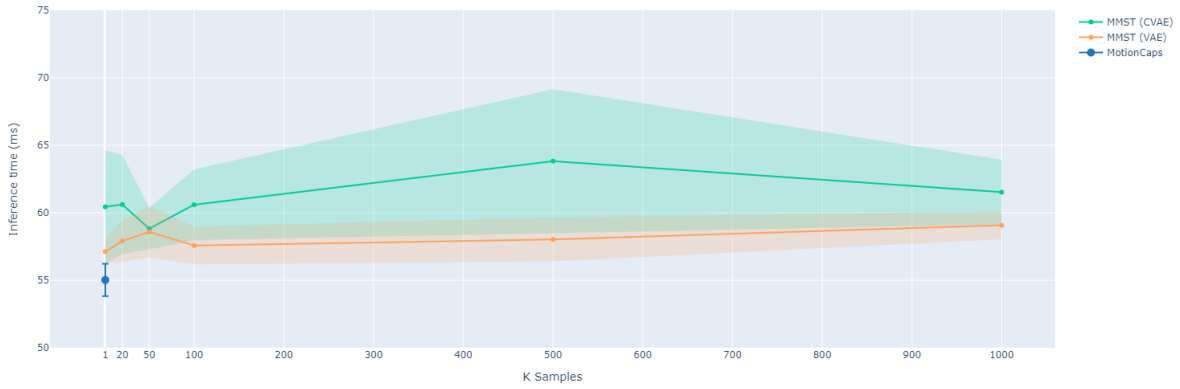

Figure 6.7: An inference time study demonstrating the speed of predicting k samples with previously examined models.

6.4.2 Facilitating the Learning Process with Minimum over N

Next, the work presented by Gupta et al. (2018) where a generative model (based on GAN) was employed for a similar task but in the domain of human trajectory prediction, argue that with a simple objective function such as $L2$ the model is strongly restricted the during optimization process which leads to learning of a limited space of plausible outcomes causing the model to often produce a set of "average" predictions. In order to aid with the learning process and encourage the network to produce a more diverse set of samples the authors proposed to utilise a novel variety loss (also known as Minimum over N or just MoN) which considers n output predictions produced by a model by randomly sampling from $\mathcal{N}(0, 1)$ instead of considering only a single prediction when optimizing the model. Furthermore, during the learning process, the MoN uses the most optimal prediction out of n predictions which is defined by an arbitrary distance metric e.g. $L2$. The formulation of MoN aims to encourage the network to cover a wider space of plausible movements that correspond to the past motion of agents. More specifically, the MoN is formulated

as:

$$MoN(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \sum_{i=1}^{N} \min_{n} \mathrm{d}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i^n) \tag{6.23}$$

where N refers to the number of data samples, n to the number of modes used during training and $\mathrm{d}(\cdot)$ to an arbitrary distance function such as $\mathrm{d}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i^n) = \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i^n\|^2$.

The following experiment examines the effect of MoN when applied during the training of MMNST. The training procedure follows a similar process as the one defined in subsection 6.4.1, however, this time the network is trained for 360 epochs with the *SPS* optimizer with adaptive learning rate (Loizou et al. 2021) and the reconstruction loss is formulated as:

$$\begin{aligned} \mathcal{J}_2 &= MoN(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) \\ &= \sum_{i=1}^{N} \min_{n} \mathrm{d}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i^n) \end{aligned} \tag{6.24}$$

with the default distance function being $\mathrm{d}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i^n) = L2(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i^n) = \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i^n\|^2$.

First, various values of n are examined with regards to the training of the model where $n \in \{1, 16, 32, 64, 128, 256\}$, results are presented in Table 6.3. Note that for sake of comparison a $n = 1$ mode has been included to demonstrate the performance gain of models trained with MoN against a default MMNST that was trained with a default $L2$ objective as presented in the 6.4.1. Above all, it is crucial to emphasize the difference between a model trained without a MoN (MMST_1) against models trained with the alternative criterion.

As presented, the initial performance increase is drastic with an improvement of both minADE_k and minFDE_k by at least 21% and 24% respectively ($k = 10$, MMST_1 vs MMST_{256}). This highlights the pivotal role of MoN in enhancing the predictive capabilities of the model, leading to more accurate trajectory predictions.Furthermore, results show that for the proposed method the optimal number of samples during training is ≈ 32 which yields the least error across three out of four sampling sce-

narios ($k = 10$). Interestingly, the performance of the model does not exhibit a linear relationship with the number of training samples (n), as performance tends to plateau or even decrease with larger n values (above $n = 32$). This observation suggests that there might be a saturation point beyond which increasing the number of training samples does not lead to further improvements in prediction accuracy. Furthermore, it is notable that for $k = 10$, the best results are achieved when $n = 16$, indicating that the MMST trained with MoN can achieve optimal performance with relatively smaller sample sizes. This finding has practical implications, as it suggests that the model can be trained effectively even with limited data, potentially reducing computational costs and training time.

MoN_n	$\text{minADE}_k/\text{minFDE}_k$			
	$k = 10$	$k = 25$	$k = 50$	$k = 100$
MMST_1	2.78/6.33	2.40/5.31	2.17/4.72	2.03/4.33
MMST_{16}	**1.77/3.81**	1.32/2.51	1.09/1.87	0.92/1.38
MMST_{32}	1.82/3.92	**1.25/2.42**	**1.05/1.82**	**0.89/1.32**
MMST_{64}	1.99/4.37	1.43/2.79	1.14/2.00	0.95/1.46
MMST_{128}	2.07/4.53	1.45/2.84	1.17/2.02	0.94/1.44
MMST_{256}	2.19/4.80	1.52/3.00	1.19/2.11	0.97/1.50

Table 6.3: Comparison study between different settings of n during training with respect to MoN loss. A minADE/minFDE errors are provided in meters for four different sampling values of k.

Next, Table 6.4 demonstrates results of employing three different distance functions $d(\cdot)$ during training of the model (n is set to the fixed value of 32 as it presented most optimal performance during the previous experiment). Two popular metric functions $L1$ and $L2$ as well as their combination which is further balanced by the $\lambda = 0.5$ parameter are examined. Results clearly demonstrate advantages of using $L2$ which outperforms other metrics on the majority of scenarios. Interestingly, the balanced combination of $L2$ and $L1$ does present the best results in a couple of cases and performs relatively similarly to $L2$ in others. This suggests that the

appropriate adjustment of the λ parameter might lead to $L2$ being outperformed in certain scenarios. Further fine-tuning of the parameter could potentially optimize the trade-off between the two distance functions, resulting in improved prediction accuracy.

Distance Function	$\text{minADE}_k/\text{minFDE}_k$			
	$k = 10$	$k = 25$	$k = 50$	$k = 100$
$L1$	1.94/4.00	1.45/2.72	1.20/2.05	1.00/1.51
$L2$	1.82/3.92	**1.25/2.42**	**1.05**/1.82	**0.89/1.32**
$\lambda(L1 + L2)$	**1.79/3.83**	1.30/2.49	1.07/**1.81**	0.90/1.36

Table 6.4: Ablation study between different distance functions. Again, four different sampling rates (k) are used during the testing.

Lastly, an analysis of displacement errors (Fig. 6.8 & 6.9) with respect to a large number of sampled trajectories ($k \leq 2^{13}$) are presented. As expected the displacement error for both metrics declines as the number of sampled trajectories grows. It is also noticed that all variations of MMST perform similarly well with models trained on larger n (i.e 128 and 256) achieving slightly lower minADE of about ≈ 0.40. In addition, a similar trend can be noticed with respect to the minFDE with all variations of MMST reaching roughly equivalently low error as k grows. Interestingly, when $k > 2^9$ the minFDE gets smaller than minADE with lowest displacement of roughly ≈ 0.17. This disparity underscores the model's ability to capture more accurate predictions of final positions as the number of sampled trajectories increases, highlighting its robustness in handling diverse and complex prediction scenarios.

6.4.3 Comparison with Methods from the Literature

Finally, a performance comparison of the MMST against other methods from the literature (which at the time of experimentation were considered the state-of-the-art methods) is presented, more specifically the following are considered during the

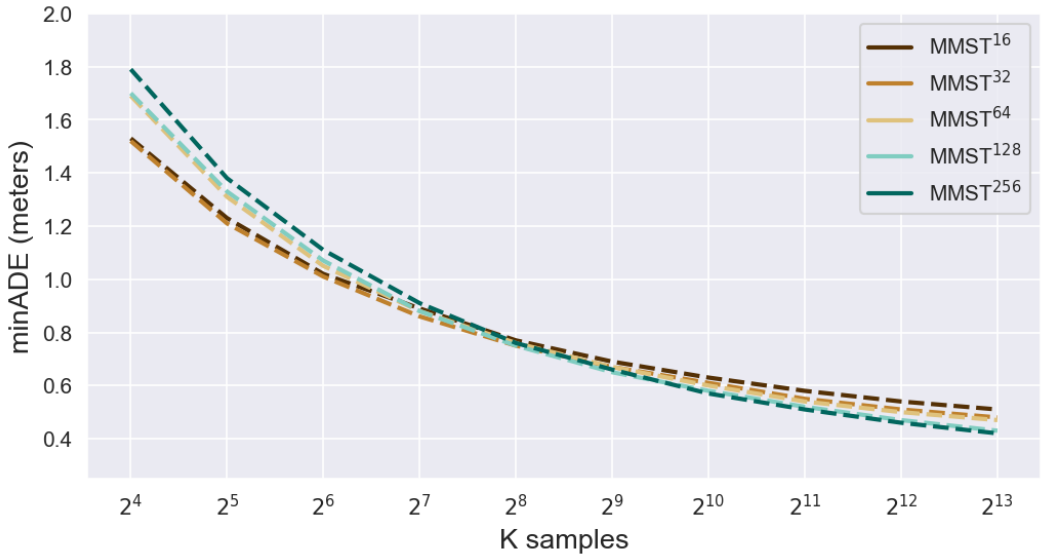

Figure 6.8: Results of the minADE error with regards to several MMST's variants where $2^4 \leq k \leq 2^{13}$.

comparison study:

1. **CoverNet** (Phan-Minh et al. 2020): This method as previously discussed in section 3.4.3 proposes to frame the multi-modal probabilistic trajectory prediction from a classification point of view by pre-generating n trajectories and then training the model to reduce the loss with respect to a classified sample that is closest to the actual ground-truth. For this experiment, the model is re-trained by following settings outlined in the original paper across three different error tolerance settings i.e. $\epsilon \in \{2, 4, 8\}$ with pre-trained ResNet-50 to extract road features and encode context of the environment.

2. **MTP** (Cui et al. 2019): Another method that uses a rasterised top-down view of the environment and encodes its salient features with use of ResNet-50. As with CoverNet the training settings for MTP are also adopted with accordance to the details outlined within the original paper. The model is re-trained to output $n \in \{64, 128, 256\}$ deterministic trajectories along with their probabilities and then sample k most probable motions.

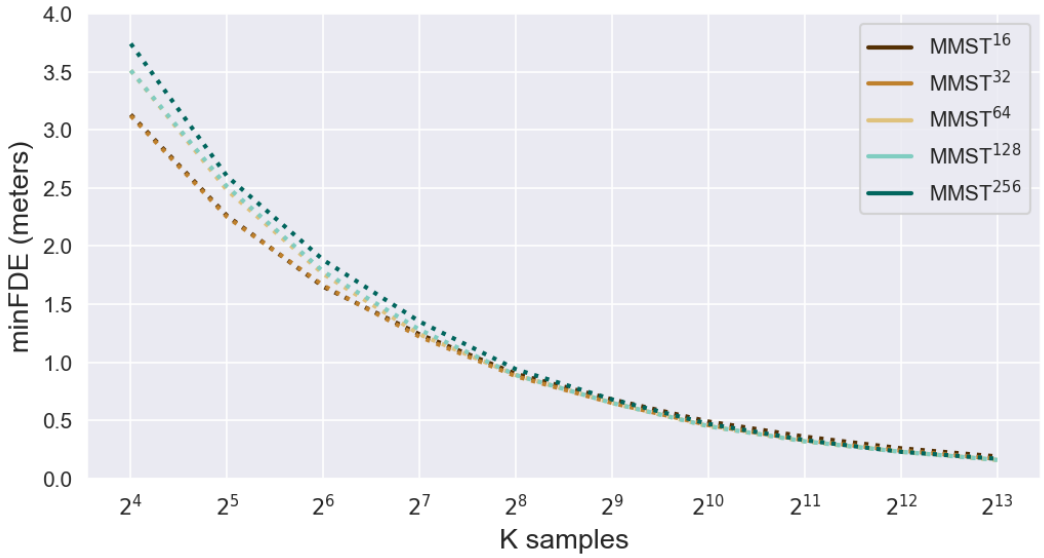

Figure 6.9: Results of the minFDE error with regards to several MMST's variants where $2^4 \leq k \leq 2^{13}$.

Results of the comparison study are presented in Table 6.5. As demonstrated, the proposed generative model either outperforms both approaches or produces results that are relatively similar to MTP. It can be noticed that MMST yields significantly higher error when $k = 1$ but then instantaneously reaches similar results when $k > 1$. In addition, MMST tends to outperform other methods across most k samples on the minADE metric whilst reaching significantly lower errors when $k = 200$. Furthermore, it can be observed that the proposed model achieves a relatively low error whilst at the same time significantly reducing the number of parameters within the network which is attributed to the use of the custom backbone feature extractor based on the capsule network. Both MTP and CoverNet employ ResNet-50 to encode the context of the environment which lead to early overfitting during re-training of these models. Moreover, the pre-trained ResNet-50 is trained on a distinct domain (Deng et al. 2009), and it is therefore arguable whether this approach actually leads to an encoding of meaningful features from the rasterised HD map. It is important to note that MMST is not restricted with respect to the number of samples it can generate as compared to both CoverNet and MTP which are limited to producing a deterministic output that must contain an equal amount of

trajectories with regards to n training modes. This flexibility positions MMST as a versatile and adaptable solution for multi-modal trajectory prediction tasks, capable of accommodating varying prediction requirements and complexities.

Model	$\text{minADE}_k/\text{minFDE}_k$				#Params
	$k = 10$	$k = 50$	$k = 100$	$k = 200$	
CoverNet^8_{64}	2.43/4.10	2.21/3.11	-	-	32.0m
CoverNet^8_{415}	2.31/4.33	1.51/2.02	1.40/1.60	1.34/1.39	33.5m
CoverNet^8_{2206}	2.32/4.60	1.38/2.02	1.17/1.46	1.04/1.09	40.9m
MTP_{64}	1.84/**3.70**	1.07/1.85	-	-	38.5m
MTP_{128}	2.39/5.18	1.23/2.04	0.93/**1.25**	-	45.0m
MTP_{256}	2.29/5.04	1.14/1.87	0.94/1.28	0.88/1.13	58.1m
MMST_{32}	**1.82**/3.92	**1.05/1.82**	**0.89**/1.32	**0.78/0.98**	7.4m

Table 6.5: Comparison study of the final model vs two recently proposed methods from the literature. Apart from providing results using minADE/minFDE the number of learnable parameters (millions) is also included to demonstrate the difference in complexity of all examined methods. All models were trained on n samples indicated by the subscript. Additionally, for CoverNet^ϵ the number of modes that each variant is trained on is adjusted with accordance to ϵ value.

6.5 Conclusion

The final chapter of this thesis concentrated on building upon the work introduced in the former chapters. The motion prediction model based on the capsule network which employed local semantic maps as an additional source of information for modelling of the state of the tracked agent has been extended with use of a stochastic model, more specifically a conditional variational auto-encoder which allowed to account for the multi-modal nature of the problem. First, an introduction to the problem as well as a general overview of auto-encoders and variational auto-encoders has been presented, stating numerous reasons as to why a previously introduced motion predictor can benefit from incorporating the CVAE framework into its pipeline.

Next, the network architecture as well as the computational flow of the proposed model has been presented to demonstrate how CVAE will be used for the problem of short-term motion prediction. Furthermore, the number of experiments has been carried out to examine the effectiveness of the proposed model. First of all, the performance of the proposed CVAE based model was examined against the classic (similar) VAE based model to establish whether conditioning the generative model yields a positive outcome, the number of different variants of CVAE were considered to determine the most successful model combination. In addition, the generative model was then examined against the capsule based motion predictor both with respect to its prediction quality as well as its speed, demonstrating that the proposed generative model can be used to produce a large number of predictions with almost a constant time. Then, to further ease the learning process an alternative objective function, Minimum over N, has been introduced and thoroughly examined to allow the model to learn a more diverse set of plausible trajectories. Finally, the performance of the proposed CVAE based motion predictor has been further compared against other methods from the literature that also focused on multi-modal prediction with use of rasterised maps. As demonstrated in the last section, the proposed model significantly outperformed other methods whilst drastically reducing the overall size of the network.

Chapter 7

Conclusions and Future Work

This chapter provides a coherent summary of the work conducted in this thesis. Starting of with the discussion of the contributions that were made throughout this work, then addressing the limitations and finally exploring the future work within the area of intelligent vehicle motion prediction.

7.1 Summary of the Thesis

The main purpose of this thesis was to first of all, investigate and explore the usage of an alternative data source in the form of rasterized HD maps that capture a local portion of the tracked agent's surroundings in order to increase the model's performance. This problem was tackled in chapter 4 where a number of experiments were conducted in order to validate the following hypothesis:

Incorporating local information about the context of the tracked agent's surrounding can be exploited to increase the general performance of a motion prediction model.

In chapter 4 a novel approach that utilizes a small portion of a rasterised 2D HD maps has been proposed. It was argued that employing the standard methodology

of using global maps that aim to capture a significantly larger portion of the surroundings might not be optimal and that instead using local maps captured over time can be better suited to encode the state of the tracked agent, and therefore increase the efficiency of the model. First, a set of experiments was conducted in order to establish a set of baseline models and to determine the advantages of using various types of data such as temporal data (agent's physical state) and spatial data (global or local rasterised maps). Results of experiments initially demonstrated that achieving most optimal performance with the basic model can be accomplished with the use of the agent's tracked physical data such as its past motion, position etc, as well as the spatial data either global or local with the model that used local maps yielding slightly better results. Then, a subsequent set of experiments aimed at improving the baseline model by first unraveling the local map into a set of separate maps where each defines a certain road layer. Then the model was further extended to process a set of local semantic maps in a temporal manner where instead of using the local chunk of a map from the last observable timestep, a set of local maps collected over a fixed period of time was utilized to further advance the model's performance. Finally, the last experiment explored various sizes of local maps to determine the most optimal setting for the training and inference. Based on the conducted experiments and obtained results it can be concluded that exploiting the local semantic maps, especially when incorporated in a temporal manner in which those maps are utilized can noticeably increase the performance of the model in regards to short-term motion prediction.

Additionally, the problem of reducing the model's complexity was also addressed as it was argued that models deployed within autonomous vehicles should achieve a certain level of inference speed which is highly dependent on the complexity of the model which must therefore be significantly lighter as opposed to large vision/language models for example, that are used to analyse data off-line. This issue was tackled in order to validate the following hypothesis:

The computational complexity of the motion predictor can be significantly reduced by training a model based on a robust capsule network from scratch directly on the target domain.

Results of the proposed methodologies and subsequent experiments are presented in chapters 5 and 6. Chapter 5 introduces a motion prediction model based on a alternative architecture of a neural network for processing of the spatial data, namely a capsule network. First, it was argued that the typical feature extractors that are employed in recent methods from the literature are often very large (number of parameters), pre-trained on the distinct domain (often classification tasks) which can greatly harm their performance if the target domain differs significantly, and use standard convolutional layers with pooling layers on top which can further reduce a model's performance as it discards a large amount of information about objects. Initial experiments in this chapter explored different architectures of motion prediction based on capsule feature extractor, more specifically, it was determined that the most optimal performance can be achieved when an unraveled set of local maps captured over time is encoded with the dedicated higher capsules which resulted in significantly better performance as opposed to other baseline models. Next, the performance of the most optimal model was further examined against a set of popular CNN feature extractors such as ResNet, DenseNet and many others. All employed feature extractors were trained either from scratch or by simply fine tuning already pre-trained weights on the task at hand. Results clearly demonstrate that using a capsule based feature extractor can potentially lead to a model that achieves a higher level of accuracy on the task of motion prediction, and in addition allows to significantly reduce the total number of parameters leading to a noticeable lighter model. Lastly, the final part of this chapter explored modes of failures of the model to gain further understanding of the cases that the model is particularly struggling with. As clearly demonstrated, this experiment revealed that certain types of movements (strongly non-linear trajectories) significantly decrease the prediction accuracy of

the model. Based on the obtained results it was further decided that in order to alleviate the discovered issues an extension of the deterministic model towards a multi-modal prediction must be made.

In the last technical chapter (chapter 6) this issue was further addressed with an extension of the model through the introduction of a generative model based on the conditional variational auto-encoder framework. As argued, the space of plausible movement for each vehicle in the vicinity is non-deterministic and is not limited to a single, correct motion. Therefore, to improve the model further a CVAE method has been employed to allow for a stochastic, multi-modal output. Initial experiments aimed at comparing the previously proposed capsule based model against two models, one based on the classical VAE framework and the other based on its extension with the conditional part. From results it was established that the CVAE based motion predictor achieves the best performance, outperforming the VAE best model as well as the deterministic model. In addition, either of the stochastic models demonstrated that there is no noticeable speed difference when sampling for a very large number of motion samples against prediction of a single sample. Next, the training process of the CVAE based model was further improved with an introduction of an alternative loss function known as Minimum over N (MoN). The use of loss function has been previously proposed in other tasks that involved usage of generative models and thus it was only logical to try to examine whether this can be used to further advance the performance of the motion predictor. Presented results clearly demonstrate the superior performance of the model that was trained with MoN loss. Lastly, the performance of the final motion prediction model based on a combination of capsule network and CVAE framework was compared against some of the methods that at the time of experimentation were considered as state-of-the-art models within the area of vehicle motion prediction. All models that were used for comparison purposes were tested on a public dataset with each model producing a multi-modal output. Yet again, the model proposed in this thesis (see Table 6.5)

demonstrated an excellent performance, significantly outperforming other methods whilst at the same time maintaining low complexity.

7.2 Limitations and Avenues for Future Work

The aim of this thesis was to improve on current motion prediction models for autonomous vehicles to better predict the short-term movement of other vehicles in close vicinity through the development of novel methodologies based on deep learning methods. Based on results obtained from chapter 4, 5 and 6 and through research and development of a novel motion prediction model it can be concluded that the defined hypotheses were correct. Nevertheless, there are a number of possible future avenues that can still be explored in this area which will be further discussed in this section.

7.2.1 Physical and Social Constrains

Precise anticipation of a vehicle's trajectory, as well as the movements of pedestrians, bicycles, and other entities on the road, is essential for achieving significantly safer autonomous systems. However, accomplishing this can prove challenging as motion prediction models typically concentrate on forecasting future movements based solely on limited observations of the agent's state within a brief time frame. An interesting area of research that was not explored in this thesis revolves around the prediction of an agent's future movement based on the knowledge of physical and social constrains. As was presented in the final section of chapter 5, the model can learn to accurately predict a large number of motion patterns that involve straight movement. However, in numerous cases, especially when driving in complex, urban environments, certain movements might simply not be plausible due to physical constrains such as buildings, footpaths, other on-road agents etc, and the model should

be able to recognize that such trajectories are invalid and should not be predicted. Moreover, there are a number of complex social interactions that are constantly happening either between vehicles or between vehicles and pedestrians. Forecasting of human trajectories has already been explored in the following work by Gupta et al. (2018), however, in the context of vehicle motion prediction this aspect is still missing. As with physical constrains, the prediction of vehicle trajectories is also strongly constrained by social interactions and this should be included in the training process to allow for not only more precise predictions but also for forecasting of the trajectories that are socially acceptable.

7.2.2 Probabilities of Predicted Trajectories

Models presented in this thesis gradually evolved from simple and shallow models to deeper deterministic motion predictors that finally demonstrated abilities to predict potentially an unlimited number of motion patterns through usage of a generative framework. Nonetheless, some of the models from the literature that were used during comparison studies in the last section of chapter 6 also presented capacities of assigning probabilities of agents performing one of numerous predicted trajectories. Typically the motion detectors that can output both sets of trajectories and their respective probabilities are constrained to predict a limited set of deterministic motions as presented in previous methods. The final model proposed in this thesis can produce potentially an unlimited number of stochastic trajectories, however, it is missing the crucial ability to assign probabilities to reach of predicted trajectories. This area of research where a generative model can also accurately predict probability of taking a certain trajectory by a vehicle is crucial for more accurate analysis of potentially dangerous on-road situations. Moreover, this should be further combined with the knowledge about physical and social interactions as a next step in order to more accurately reflect those movements.

7.2.3 Public Datasets

Unfortunately, in this thesis only one public dataset was suitable for carrying out experiments that allowed the validation of the defined hypothesis. Lack of large, public datasets not only in the area of motion prediction but also in other tasks related to autonomous vehicles has an immense effect on the research community where researchers are greatly restricted from carrying out experiments as well as the development of models that could considerably contribute to numerous problems that are yet to be solved in the general area of autonomous vehicles. It is important to acknowledge that despite having the access to even the most advanced deep learning models, one would be still largely limited in terms of advancing such models if the dataset used for training does not include a large number of complex and representative scenarios. As demonstrated in the last section of chapter 5, the proposed deterministic model learned to accurately predict mostly straight movement but struggled with more complex motion. This inability of the proposed model to precisely handle more complicated trajectories is greatly influenced simply by the fact that these sorts of movements were not as common as other, more simple motion patterns.

7.2.4 Novel Architectures

Lastly, in recent years a number of novel deep learning architectures have been developed that could potentially lead to even further improvements in the area of motion prediction. For example, a transformer (Vaswani et al. 2017) architecture has been employed with enormous success in a number of tasks within the area of natural language processing. More recently however, transformers have also brought a lot of success in the area of computer vision (Dosovitskiy et al. 2020). This suggests that novel architectures based on transformers could be explored for this type of work as

it is suited both for spatial and temporal data. Recent years have also seen a rapid growth in the area of graph neural networks. This type of architecture appears to be greatly suitable to the task such as modelling of interactions between on-road users.

Bibliography

Abbeel, P. & Ng, A. Y. (2004), Apprenticeship learning via inverse reinforcement learning, *in* 'Proceedings of the twenty-first international conference on Machine learning', p. 1.

Acharya, M., Hayes, T. L. & Kanan, C. (2020), 'Rodeo: Replay for online object detection', *arXiv preprint arXiv:2008.06439* .

Akabane, A. T., Pazzi, R. W., Madeira, E. R. & Villas, L. A. (2017), Modeling and prediction of vehicle routes based on hidden markov model, *in* '2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)', IEEE, pp. 1–5.

Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L. & Savarese, S. (2016), Social lstm: Human trajectory prediction in crowded spaces, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 961–971.

Ammoun, S. & Nashashibi, F. (2009), Real time trajectory prediction for collision risk estimation between vehicles, *in* '2009 IEEE 5th International Conference on Intelligent Computer Communication and Processing', IEEE, pp. 417–422.

Aoude, G. S., Desaraju, V. R., Stephens, L. H. & How, J. P. (2011), Behavior classification algorithms at intersections and validation using naturalistic data, *in* '2011 IEEE Intelligent Vehicles Symposium (IV)', IEEE, pp. 601–606.

Aoude, G. S. & How, J. P. (2009), Using support vector machines and bayesian filtering for classifying agent intentions at road intersections, Technical report.

Augustin, D., Hofmann, M. & Konigorski, U. (2018), Motion pattern recognition for maneuver detection and trajectory prediction on highways, *in* '2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)', IEEE, pp. 1–8.

Bank, D., Koenigstein, N. & Giryes, R. (2020), 'Autoencoders', *arXiv preprint arXiv:2003.05991* .

Bar-Shalom, Y., Li, X. R. & Kirubarajan, T. (2004), *Estimation with applications to tracking and navigation: theory algorithms and software*, John Wiley & Sons.

Barth, A. & Franke, U. (2008), Where will the oncoming vehicle be the next second?, *in* '2008 IEEE Intelligent Vehicles Symposium', IEEE, pp. 1068–1073.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R. et al. (2018), 'Relational inductive biases, deep learning, and graph networks', *arXiv preprint arXiv:1806.01261* .

Batz, T., Watson, K. & Beyerer, J. (2009), Recognition of dangerous situations within a cooperative group of vehicles, *in* '2009 IEEE Intelligent Vehicles Symposium', IEEE, pp. 907–912.

Bengio, Y., Simard, P., Frasconi, P. et al. (1994), 'Learning long-term dependencies with gradient descent is difficult', *IEEE transactions on neural networks* **5**(2), 157–166.

Berndt, H., Emmert, J. & Dietmayer, K. (2008), Continuous driver intention recognition with hidden markov models, *in* '2008 11th International IEEE Conference on Intelligent Transportation Systems', IEEE, pp. 1189–1194.

Bishop, C. M. (2006), *Pattern recognition and machine learning*, springer.

Bishop, R. (2005), *Intelligent vehicle technology and trends*.

Bliss, J. P. & Acton, S. A. (2003), 'Alarm mistrust in automobiles: how collision alarm reliability affects driving', *Applied ergonomics* **34**(6), 499–509.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J. et al. (2016), 'End to end learning for self-driving cars', *arXiv preprint arXiv:1604.07316* .

Broadhurst, A., Baker, S. & Kanade, T. (2005), Monte carlo road safety reasoning, *in* 'IEEE Proceedings. Intelligent Vehicles Symposium, 2005.', IEEE, pp. 319–324.

Broggi, A., Bertozzi, M., Fascioli, A., Bianco, C. G. L. & Piazzi, A. (1999), 'The argo autonomous vehicle's vision and control systems', *International Journal of Intelligent Control and Systems* **3**(4), 409–441.

Broughton, J. & Baughan, C. (2002), 'The effectiveness of antilock braking systems in reducing accidents in great britain', *Accident Analysis & Prevention* **34**(3), 347–355.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. et al. (2020), 'Language models are few-shot learners', *arXiv preprint arXiv:2005.14165* .

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G. & Beijbom, O. (2020), nuscenes: A multimodal dataset for autonomous driving, *in* 'Proceedings of the IEEE/CVF conference on computer vision and pattern recognition', pp. 11621–11631.

Chandra, R., Bhattacharya, U., Bera, A. & Manocha, D. (2019), Traphic: Trajectory prediction in dense and heterogeneous traffic using weighted interactions, *in*

'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 8483–8492.

Chandra, R., Bhattacharya, U., Roncal, C., Bera, A. & Manocha, D. (2019), Robusttp: End-to-end trajectory prediction for heterogeneous road-agents in dense traffic with noisy sensor inputs, *in* 'ACM Computer Science in Cars Symposium', pp. 1–9.

Chang, M.-F., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D. et al. (2019), Argoverse: 3d tracking and forecasting with rich maps, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 8748–8757.

Choi, C. & Dariush, B. (2019), Looking to relations for future trajectory forecast, *in* 'Proceedings of the IEEE/CVF International Conference on Computer Vision', pp. 921–930.

Choi, C., Patil, A. & Malla, S. (2019), 'Drogon: A causal reasoning framework for future trajectory forecast', *arXiv preprint arXiv:1908.00024* .

Chollet, F. (2017), Xception: Deep learning with depthwise separable convolutions, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1251–1258.

Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. (2014), 'Empirical evaluation of gated recurrent neural networks on sequence modeling', *arXiv preprint arXiv:1412.3555* .

Clevert, D.-A., Unterthiner, T. & Hochreiter, S. (2015), 'Fast and accurate deep network learning by exponential linear units (elus)', *arXiv preprint arXiv:1511.07289* .

CNN face detection failure (n.d.), `http://sharenoesis.com/wp-content/uploads/2010/05/7ShapeFaceRemoveGuides.jpg`. Accessed: 2021-04-08.

Committee, S. O.-R. A. V. S. et al. (2018), 'Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles', *SAE International: Warrendale, PA, USA* .

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S. & Schiele, B. (2016), The cityscapes dataset for semantic urban scene understanding, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 3213–3223.

Cortes, C. & Vapnik, V. (1995), 'Support-vector networks', *Machine learning* **20**(3), 273–297.

Cui, H., Radosavljevic, V., Chou, F.-C., Lin, T.-H., Nguyen, T., Huang, T.-K., Schneider, J. & Djuric, N. (2019), Multimodal trajectory predictions for autonomous driving using deep convolutional networks, *in* '2019 International Conference on Robotics and Automation (ICRA)', IEEE, pp. 2090–2096.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009), Imagenet: A large-scale hierarchical image database, *in* '2009 IEEE conference on computer vision and pattern recognition', Ieee, pp. 248–255.

Deo, N. & Trivedi, M. M. (2018*a*), Convolutional social pooling for vehicle trajectory prediction, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops', pp. 1468–1476.

Deo, N. & Trivedi, M. M. (2018*b*), Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms, *in* '2018 IEEE Intelligent Vehicles Symposium (IV)', IEEE, pp. 1179–1184.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805* .

Dickmanns, E. D., Behringer, R., Dickmanns, D., Hildebrandt, T., Maurer, M., Thomanek, F. & Schiehlen, J. (1994), The seeing passenger car'vamors-p', *in* 'Intelligent Vehicles' 94 Symposium, Proceedings of the', IEEE, pp. 68–73.

Djuric, N., Radosavljevic, V., Cui, H., Nguyen, T., Chou, F.-C., Lin, T.-H., Singh, N. & Schneider, J. (2020), Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving, *in* 'The IEEE Winter Conference on Applications of Computer Vision', pp. 2095–2104.

Doersch, C. (2016), 'Tutorial on variational autoencoders', *arXiv preprint arXiv:1606.05908* .

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. et al. (2020), 'An image is worth 16x16 words: Transformers for image recognition at scale. arxiv 2020', *arXiv preprint arXiv:2010.11929* .

Du, B., Xiong, W., Wu, J., Zhang, L., Zhang, L. & Tao, D. (2016), 'Stacked convolutional denoising auto-encoders for feature representation', *IEEE transactions on cybernetics* **47**(4), 1017–1027.

Dumoulin, V. & Visin, F. (2016), 'A guide to convolution arithmetic for deep learning', *arXiv preprint arXiv:1603.07285* .

Eidehall, A. & Petersson, L. (2008), 'Statistical threat assessment for general road scenes using monte carlo sampling', *IEEE Transactions on intelligent transportation systems* **9**(1), 137–147.

Ferguson, D., Baker, C., Likhachev, M. & Dolan, J. (2008), A reasoning framework for autonomous urban driving, *in* '2008 IEEE Intelligent Vehicles Symposium', IEEE, pp. 775–780.

for Health Statistics, N. C. (2017), *Health, United States, 2016, with chartbook on Long-term trends in health*, number 2017, Government Printing Office.

Forney, G. D. (1973), 'The viterbi algorithm', *Proceedings of the IEEE* **61**(3), 268–278.

Fraichard, T. & Asama, H. (2004), 'Inevitable collision states—a step towards safer robots?', *Advanced Robotics* **18**(10), 1001–1024.

Fritzke, B. (1995), A growing neural gas network learns topologies, *in* 'Advances in neural information processing systems', pp. 625–632.

Fukushima, K. (1980), 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position', *Biological cybernetics* **36**(4), 193–202.

Gao, J., Sun, C., Zhao, H., Shen, Y., Anguelov, D., Li, C. & Schmid, C. (2020), Vectornet: Encoding hd maps and agent dynamics from vectorized representation, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 11525–11533.

Gatys, L. A., Ecker, A. S. & Bethge, M. (2016), Image style transfer using convolutional neural networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 2414–2423.

Geiger, A., Lenz, P. & Urtasun, R. (2012), Are we ready for autonomous driving? the kitti vision benchmark suite, *in* '2012 IEEE conference on computer vision and pattern recognition', IEEE, pp. 3354–3361.

Geyer, J., Kassahun, Y., Mahmudi, M., Ricou, X., Durgesh, R., Chung, A. S., Hauswald, L., Pham, V. H., Mühlegg, M., Dorn, S. et al. (2020), 'A2d2: Audi autonomous driving dataset', *arXiv preprint arXiv:2004.06320* .

Gillespie, T. D. (1992), *Fundamentals of vehicle dynamics*, Vol. 400, Society of automotive engineers Warrendale, PA.

Giuliari, F., Hasan, I., Cristani, M. & Galasso, F. (2020), 'Transformer networks for trajectory forecasting', *arXiv preprint arXiv:2003.08111* .

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep learning*, MIT press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014), Generative adversarial nets, *in* 'Advances in neural information processing systems', pp. 2672–2680.

Graves, A., Mohamed, A.-r. & Hinton, G. (2013), Speech recognition with deep recurrent neural networks, *in* '2013 IEEE international conference on acoustics, speech and signal processing', IEEE, pp. 6645–6649.

Guérillot, D., Bruyelle, J. et al. (2017), Uncertainty assessment in production forecast with an optimal artificial neural network, *in* 'SPE Middle East Oil & Gas Show and Conference', Society of Petroleum Engineers.

Gupta, A., Johnson, J., Fei-Fei, L., Savarese, S. & Alahi, A. (2018), Social gan: Socially acceptable trajectories with generative adversarial networks, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 2255–2264.

Gurney, K. (1997), 'An introduction to neural networks'.

Hajela, P. & Berke, L. (1991), 'Neurobiological computational models in structural analysis and design', *Computers & Structures* **41**(4), 657–667.

Hammersley, J. (2013), *Monte carlo methods*, Springer Science & Business Media.

Hartigan, J. A. & Wong, M. A. (1979), 'Algorithm as 136: A k-means clustering algorithm', *Journal of the royal statistical society. series c (applied statistics)* **28**(1), 100–108.

He, K., Girshick, R. & Dollár, P. (2019), Rethinking imagenet pre-training, *in* 'Proceedings of the IEEE/CVF International Conference on Computer Vision', pp. 4918–4927.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

Hillenbrand, J., Spieker, A. M. & Kroschel, K. (2006), 'A multilevel collision mitigation approach—its situation assessment, decision making, and performance tradeoffs', *IEEE Transactions on intelligent transportation systems* **7**(4), 528–540.

Hinton, G. E., Krizhevsky, A. & Wang, S. D. (2011), Transforming auto-encoders, *in* 'International conference on artificial neural networks', Springer, pp. 44–51.

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.

Hofmann, T. & Buhmann, J. M. (1997), 'Pairwise data clustering by deterministic annealing', *Ieee transactions on pattern analysis and machine intelligence* **19**(1), 1–14.

Hotelling, H. (1933), 'Analysis of a complex of statistical variables into principal components.', *Journal of educational psychology* **24**(6), 417.

Houenou, A., Bonnifait, P., Cherfaoui, V. & Yao, W. (2013), Vehicle trajectory prediction based on motion model and maneuver recognition, *in* '2013 IEEE/RSJ international conference on intelligent robots and systems', IEEE, pp. 4363–4369.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V. et al. (2019), Searching for mobilenetv3, *in* 'Proceedings of the IEEE/CVF International Conference on Computer Vision', pp. 1314–1324.

Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K. Q. (2017), Densely connected convolutional networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 4700–4708.

Huang, J. & Tan, H.-S. (2006), Vehicle future trajectory prediction with a dgps/ins-based positioning system, *in* '2006 American Control Conference', IEEE, pp. 6–pp.

Huang, X., Wang, P., Cheng, X., Zhou, D., Geng, Q. & Yang, R. (2019), 'The apolloscape open dataset for autonomous driving and its application', *IEEE transactions on pattern analysis and machine intelligence* **42**(10), 2702–2719.

Hubel, D. H. & Wiesel, T. N. (1968), 'Receptive fields and functional architecture of monkey striate cortex', *The Journal of physiology* **195**(1), 215–243.

Hubert, M. & Van Driessen, K. (2004), 'Fast and robust discriminant analysis', *Computational Statistics & Data Analysis* **45**(2), 301–320.

Hülnhagen, T., Dengler, I., Tamke, A., Dang, T. & Breuel, G. (2010), Maneuver recognition using probabilistic finite-state machines and fuzzy logic, *in* '2010 IEEE Intelligent Vehicles Symposium', IEEE, pp. 65–70.

Ioffe, S. & Szegedy, C. (2015), Batch normalization: Accelerating deep network training by reducing internal covariate shift, *in* 'International conference on machine learning', PMLR, pp. 448–456.

Jang, J.-S. R., Sun, C.-T. & Mizutani, E. (1997), 'Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence [book review]', *IEEE Transactions on automatic control* **42**(10), 1482–1484.

Jazwinski, A. H. (2007), *Stochastic processes and filtering theory*, Courier Corporation.

Jensen, F. V. et al. (1996), *An introduction to Bayesian networks*, Vol. 210, UCL press London.

Jordan, J. (2018), 'Introduction to autoencoders', *Jeremy Jordan, Mar* .

Joseph, J., Doshi-Velez, F., Huang, A. S. & Roy, N. (2011), 'A bayesian nonparametric approach to modeling motion patterns', *Autonomous Robots* **31**(4), 383.

Julier, S. J. & Uhlmann, J. K. (2004), 'Unscented filtering and nonlinear estimation', *Proceedings of the IEEE* **92**(3), 401–422.

Kalchbrenner, N. & Blunsom, P. (2013), Recurrent continuous translation models, *in* 'Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing', pp. 1700–1709.

Kalman, R. E. (1960), 'A new approach to linear filtering and prediction problems'.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Kingma, D. P., Mohamed, S., Jimenez Rezende, D. & Welling, M. (2014), 'Semi-supervised learning with deep generative models', *Advances in neural information processing systems* **27**.

Kingma, D. P. & Welling, M. (2013), 'Auto-encoding variational bayes', *arXiv preprint arXiv:1312.6114* .

Kingma, D. P., Welling, M. et al. (2019), 'An introduction to variational autoencoders', *Foundations and Trends® in Machine Learning* **12**(4), 307–392.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* 'Advances in neural information processing systems', pp. 1097–1105.

Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E. & How, J. P. (2009), 'Real-time motion planning with applications to autonomous urban driving', *IEEE Transactions on control systems technology* **17**(5), 1105–1118.

LaValle, S. M. (1998), 'Rapidly-exploring random trees: A new tool for path planning'.

LeCun, Y., Bengio, Y. & Hinton, G. (2015), 'Deep learning', *nature* **521**(7553), 436.

LeCun, Y., Bengio, Y. et al. (1995), 'Convolutional networks for images, speech, and time series', *The handbook of brain theory and neural networks* **3361**(10), 1995.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E. & Jackel, L. D. (1990), Handwritten digit recognition with a back-propagation network, *in* 'Advances in neural information processing systems', pp. 396–404.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324.

Lee, D., Gu, Y., Hoang, J. & Marchetti-Bowick, M. (2019), 'Joint interaction and trajectory prediction for autonomous driving using graph neural networks', *arXiv preprint arXiv:1912.07882* .

Lee, J. D., Hoffman, J. D. & Hayes, E. (2004), Collision warning design to mitigate driver distraction, *in* 'Proceedings of the SIGCHI Conference on Human factors in Computing Systems', pp. 65–72.

Lee, N., Choi, W., Vernaza, P., Choy, C. B., Torr, P. H. & Chandraker, M. (2017), Desire: Distant future prediction in dynamic scenes with interacting agents, *in*

'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 336–345.

Li, X. R. & Jilkov, V. P. (2003), 'Survey of maneuvering target tracking. part i. dynamic models', *IEEE Transactions on aerospace and electronic systems* **39**(4), 1333–1364.

Lin, C.-F., Ulsoy, A. G. & LeBlanc, D. J. (2000), 'Vehicle dynamics and external disturbance estimation for vehicle path prediction', *IEEE Transactions on Control Systems Technology* **8**(3), 508–518.

Litman, T. (2017), *Autonomous vehicle implementation predictions*, Victoria Transport Policy Institute Victoria, Canada.

Liu, G., Reda, F. A., Shih, K. J., Wang, T.-C., Tao, A. & Catanzaro, B. (2018), Image inpainting for irregular holes using partial convolutions, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 85–100.

Loizou, N., Vaswani, S., Laradji, I. H. & Lacoste-Julien, S. (2021), Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence, *in* 'International Conference on Artificial Intelligence and Statistics', PMLR, pp. 1306–1314.

Luo, W., Yang, B. & Urtasun, R. (2018), Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net, *in* 'Proceedings of the IEEE conference on Computer Vision and Pattern Recognition', pp. 3569–3577.

Lytrivis, P., Thomaidis, G. & Amditis, A. (2008), Cooperative path prediction in vehicular environments, *in* '2008 11th International IEEE Conference on Intelligent Transportation Systems', IEEE, pp. 803–808.

Maas, A. L., Hannun, A. Y. & Ng, A. Y. (2013), Rectifier nonlinearities improve neural network acoustic models, *in* 'Proc. icml', Vol. 30, Citeseer, p. 3.

Mandalia, H. M. & Salvucci, M. D. D. (2005), Using support vector machines for lane-change detection, *in* 'Proceedings of the human factors and ergonomics society annual meeting', Vol. 49, SAGE Publications Sage CA: Los Angeles, CA, pp. 1965–1969.

Marchetti, F., Becattini, F., Seidenari, L. & Bimbo, A. D. (2020), Mantra: Memory augmented networks for multiple trajectory prediction, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 7143–7152.

McCall, J. C., Wipf, D. P., Trivedi, M. M. & Rao, B. D. (2007), 'Lane change intent analysis using robust operators and sparse bayesian learning', *IEEE Transactions on Intelligent Transportation Systems* **8**(3), 431–440.

McCarthy, J. (1997), 'Ai as sport'.

McClelland, J. L., Rumelhart, D. E., Group, P. R. et al. (1987), *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, Vol. 2, MIT press.

McCulloch, W. S. & Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133.

Mechner, D. A. (1998), 'All systems go', *Sciences* **38**(1), 32.

Messaoud, K., Deo, N., Trivedi, M. M. & Nashashibi, F. (2020), 'Trajectory prediction for autonomous driving based on multi-head attention with joint agent-map representation'.

Messaoud, K., Yahiaoui, I., Verroust-Blondet, A. & Nashashibi, F. (2019), Non-local social pooling for vehicle trajectory prediction, *in* '2019 IEEE Intelligent Vehicles Symposium (IV)', IEEE, pp. 975–980.

Mikolov, T. (2012), 'Statistical language models based on neural networks', *Presentation at Google, Mountain View, 2nd April* **80**.

Mirza, M. & Osindero, S. (2014), 'Conditional generative adversarial nets', *arXiv preprint arXiv:1411.1784* .

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. (2015), 'Human-level control through deep reinforcement learning', *Nature* **518**(7540), 529–533.

Morris, B., Doshi, A. & Trivedi, M. (2011), Lane change intent prediction for driver assistance: On-road design and evaluation, *in* '2011 IEEE Intelligent Vehicles Symposium (IV)', IEEE, pp. 895–901.

Nair, V. & Hinton, G. E. (2010), Rectified linear units improve restricted boltzmann machines, *in* 'Icml'.

Nasios, N. & Bors, A. G. (2006), 'Variational learning for gaussian mixture models', *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **36**(4), 849–862.

Nikhil, N. & Tran Morris, B. (2018), Convolutional neural network for trajectory prediction, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 0–0.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K. & Mordvintsev, A. (2018), 'The building blocks of interpretability', *Distill* **3**(3), e10.

Organization, W. H. et al. (2023), *Pedestrian safety: a road safety manual for decision-makers and practitioners*, World Health Organization.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. et al. (2019), 'Pytorch: An imperative

style, high-performance deep learning library', *Advances in neural information processing systems* **32**, 8026–8037.

Patil, A., Malla, S., Gang, H. & Chen, Y.-T. (2019), The h3d dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes, *in* '2019 International Conference on Robotics and Automation (ICRA)', IEEE, pp. 9552–9557.

Pepy, R., Lambert, A. & Mounier, H. (2006), Reducing navigation errors by planning with realistic vehicle model, *in* '2006 IEEE Intelligent Vehicles Symposium', IEEE, pp. 300–307.

Petit, S. (2017), 'World vehicle population rose 4.6% in 2016', *Wards Auto* .

Petti, S. & Fraichard, T. (2005), Safe motion planning in dynamic environments, *in* '2005 IEEE/RSJ International Conference on Intelligent Robots and Systems', IEEE, pp. 2210–2215.

Pham, Q.-H., Sevestre, P., Pahwa, R. S., Zhan, H., Pang, C. H., Chen, Y., Mustafa, A., Chandrasekhar, V. & Lin, J. (2020), A* 3d dataset: Towards autonomous driving in challenging environments, *in* '2020 IEEE International Conference on Robotics and Automation (ICRA)', IEEE, pp. 2267–2273.

Phan-Minh, T., Grigore, E. C., Boulton, F. A., Beijbom, O. & Wolff, E. M. (2020), Covernet: Multimodal behavior prediction using trajectory sets, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 14074–14083.

Polychronopoulos, A., Tsogas, M., Amditis, A. J. & Andreone, L. (2007), 'Sensor fusion for predicting vehicles' path for collision avoidance systems', *IEEE Transactions on Intelligent Transportation Systems* **8**(3), 549–562.

Pomerleau, D. A. (1989), Alvinn: An autonomous land vehicle in a neural network, *in* 'Advances in neural information processing systems', pp. 305–313.

Pomerleau, D. & Jochem, T. (1996), 'Rapidly adapting machine vision for automated vehicle steering', *IEEE expert* **11**(2), 19–27.

Rabiner, L. R. (1989), 'A tutorial on hidden markov models and selected applications in speech recognition', *Proceedings of the IEEE* **77**(2), 257–286.

Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018), 'Improving language understanding by generative pre-training'.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. (2019), 'Language models are unsupervised multitask learners', *OpenAI blog* **1**(8), 9.

Rajamani, R. (2011), *Vehicle dynamics and control*, Springer Science & Business Media.

Rasmussen, C. E. (2003), Gaussian processes in machine learning, *in* 'Summer School on Machine Learning', Springer, pp. 63–71.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016), You only look once: Unified, real-time object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 779–788.

Ren, S., He, K., Girshick, R. & Sun, J. (2015), Faster r-cnn: Towards real-time object detection with region proposal networks, *in* 'Advances in neural information processing systems', pp. 91–99.

Reynolds, D. A. (2009), 'Gaussian mixture models.', *Encyclopedia of biometrics* **741**.

Rosenblatt, F. (1957), *The Perceptron, a Perceiving and Recognizing Automaton Project Para*, Report: Cornell Aeronautical Laboratory, Cornell Aeronautical Laboratory.

URL: *https://books.google.co.uk/books?id=P_XGPgAACAAJ*

Rothman, D. (2018), *Artificial Intelligence By Example: Develop machine intelligence from scratch using real artificial intelligence use cases*, Packt Publishing Ltd.

Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', *arXiv preprint arXiv:1609.04747* .

Rudin-Brown, C. M. & Parker, H. A. (2004), 'Behavioural adaptation to adaptive cruise control (acc): implications for preventive strategies', *Transportation Research Part F: Traffic Psychology and Behaviour* **7**(2), 59–76.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. et al. (1988), 'Learning representations by back-propagating errors', *Cognitive modeling* **5**(3), 1.

Rupprecht, C., Laina, I., DiPietro, R., Baust, M., Tombari, F., Navab, N. & Hager, G. D. (2017), Learning in an uncertain world: Representing ambiguity through multiple hypotheses, *in* 'Proceedings of the IEEE International Conference on Computer Vision', pp. 3591–3600.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. et al. (2015), 'Imagenet large scale visual recognition challenge', *International journal of computer vision* **115**(3), 211–252.

Sabour, S., Frosst, N. & Hinton, G. E. (2017), 'Dynamic routing between capsules', *arXiv preprint arXiv:1710.09829* .

Sadeghian, A., Kosaraju, V., Gupta, A., Savarese, S. & Alahi, A. (2018), 'Trajnet: Towards a benchmark for human trajectory prediction', *arXiv preprint* .

Sadeghian, A., Kosaraju, V., Sadeghian, A., Hirose, N., Rezatofighi, H. & Savarese, S. (2019), Sophie: An attentive gan for predicting paths compliant to social and physical constraints, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 1349–1358.

Schreier, M., Willert, V. & Adamy, J. (2014), Bayesian, maneuver-based, long-term trajectory prediction and criticality assessment for driver assistance systems, *in* '17th International IEEE Conference on Intelligent Transportation Systems (ITSC)', IEEE, pp. 334–341.

Schubert, R., Richter, E. & Wanielik, G. (2008), Comparison and evaluation of advanced motion models for vehicle tracking, *in* '2008 11th international conference on information fusion', IEEE, pp. 1–6.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. & LeCun, Y. (2013), 'Overfeat: Integrated recognition, localization and detection using convolutional networks', *arXiv preprint arXiv:1312.6229* .

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016), 'Mastering the game of go with deep neural networks and tree search', *nature* **529**(7587), 484.

Singh, S. (2015), Critical reasons for crashes investigated in the national motor vehicle crash causation survey, Technical report.

Sohn, K., Lee, H. & Yan, X. (2015), Learning structured output representation using deep conditional generative models, *in* 'Advances in neural information processing systems', pp. 3483–3491.

Srikanth, S., Ansari, J. A., Ram, R. K., Sharma, S., Murthy, J. K. & Krishna, K. M. (2019), Infer: Intermediate representations for future prediction, *in* '2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 942–949.

Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B. et al. (2020), Scalability in perception for au-

tonomous driving: Waymo open dataset, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 2446–2454.

Sutton, R. S., Barto, A. G. et al. (1998), *Introduction to reinforcement learning*, Vol. 135, MIT press Cambridge.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. & Rabinovich, A. (2015), Going deeper with convolutions, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1–9.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2016), Rethinking the inception architecture for computer vision, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 2818–2826.

Taheri, S. (1992), 'An investigation and design of slip control braking systems integrated with four-wheel steering.'.

Tan, P.-N., Steinbach, M. & Kumar, V. (2016), *Introduction to data mining*, Pearson Education India.

Tao, A., Sapra, K. & Catanzaro, B. (2020), 'Hierarchical multi-scale attention for semantic segmentation', *arXiv preprint arXiv:2005.10821* .

Teh, Y. W. (2010), 'Dirichlet process.'.

Thrun, S. (2002), 'Probabilistic robotics', *Communications of the ACM* **45**(3), 52–57.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G. et al. (2006), 'Stanley: The robot that won the darpa grand challenge', *Journal of field Robotics* **23**(9), 661–692.

Tipping, M. E. (2000), The relevance vector machine, *in* 'Advances in neural information processing systems', pp. 652–658.

Tipping, M. E. (2001), 'Sparse bayesian learning and the relevance vector machine', *Journal of machine learning research* **1**(Jun), 211–244.

Touvron, H., Vedaldi, A., Douze, M. & Jégou, H. (2020), 'Fixing the train-test resolution discrepancy: Fixefficientnet', *arXiv preprint arXiv:2003.08237* .

Tran, Q. & Firl, J. (2013), Modelling of traffic situations at urban intersections with probabilistic non-parametric regression, *in* '2013 IEEE Intelligent Vehicles Symposium (IV)', IEEE, pp. 334–339.

Tran, Q. & Firl, J. (2014), Online maneuver recognition and multimodal trajectory prediction for intersection assistance using non-parametric regression, *in* '2014 IEEE Intelligent Vehicles Symposium Proceedings', IEEE, pp. 918–923.

Trottier, L., Giguere, P. & Chaib-Draa, B. (2017), Parametric exponential linear unit for deep convolutional neural networks, *in* '2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)', IEEE, pp. 207–214.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C. et al. (2008), 'Autonomous driving in urban environments: Boss and the urban challenge', *Journal of Field Robotics* **25**(8), 425–466.

Vasquez, D. & Fraichard, T. (2004), Motion prediction for moving objects: a statistical approach, *in* 'IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004', Vol. 4, IEEE, pp. 3931–3936.

Vasquez, D., Fraichard, T., Aycard, O. & Laugier, C. (2008), 'Intentional motion on-line learning and prediction', *Machine Vision and Applications* **19**(5-6), 411–425.

Vasquez, D., Fraichard, T. & Laugier, C. (2009), 'Incremental learning of statistical motion patterns with growing hidden markov models', *IEEE Transactions on Intelligent Transportation Systems* **10**(3), 403–416.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017), Attention is all you need, *in* 'Advances in neural information processing systems', pp. 5998–6008.

Veeraraghavan, H., Papanikolopoulos, N. & Schrater, P. (2006), Deterministic sampling-based switching kalman filtering for vehicle tracking, *in* '2006 IEEE Intelligent Transportation Systems Conference', IEEE, pp. 1340–1345.

Veeraraghavanm, H. & Papanikolopoulos, N. (2004), Combining multiple tracking modalities for vehicle tracking at traffic intersections, *in* 'IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004', Vol. 3, IEEE, pp. 2303–2308.

Velodyne LiDAR (n.d.).
URL: *https://www.velodynelidar.com/*

Vidal, E., Thollard, F., De La Higuera, C., Casacuberta, F. & Carrasco, R. C. (2005), 'Probabilistic finite-state machines-part i', *IEEE transactions on pattern analysis and machine intelligence* **27**(7), 1013–1025.

Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.-A. (2008), Extracting and composing robust features with denoising autoencoders, *in* 'Proceedings of the 25th international conference on Machine learning', pp. 1096–1103.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. & Lang, K. J. (1989), 'Phoneme recognition using time-delay neural networks', *IEEE transactions on acoustics, speech, and signal processing* **37**(3), 328–339.

Wang, K., Gao, X., Zhao, Y., Li, X., Dou, D. & Xu, C.-Z. (2019), Pay attention to features, transfer learn faster cnns, *in* 'International Conference on Learning Representations'.

Waymo Technology (n.d.).
URL: *https://waymo.com/tech/*

Weston, J., Chopra, S. & Bordes, A. (2014), 'Memory networks', *arXiv preprint arXiv:1410.3916* .

Wiest, J., Höffken, M., Kreßel, U. & Dietmayer, K. (2012), Probabilistic trajectory prediction with gaussian mixture models, *in* '2012 IEEE Intelligent Vehicles Symposium', IEEE, pp. 141–146.

Winner, H., Witte, S., Uhler, W. & Lichtenberg, B. (1996), 'Adaptive cruise control system aspects and development trends', *SAE transactions* pp. 1412–1421.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R. & Bengio, Y. (2015), Show, attend and tell: Neural image caption generation with visual attention, *in* 'International conference on machine learning', pp. 2048–2057.

Yu, F. & Koltun, V. (2015), 'Multi-scale context aggregation by dilated convolutions', *arXiv preprint arXiv:1511.07122* .

Zadeh, L. A. (1965), 'Fuzzy sets', *Information and control* **8**(3), 338–353.

Zhang, H., Wang, Y., Liu, J., Li, C., Ma, T. & Yin, C. (2020), 'A multi-modal states based vehicle descriptor and dilated convolutional social pooling for vehicle trajectory prediction', *arXiv preprint arXiv:2003.03480* .

Zhao, T., Xu, Y., Monfort, M., Choi, W., Baker, C., Zhao, Y., Wang, Y. & Wu, Y. N. (2019), Multi-agent tensor fusion for contextual trajectory prediction, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 12126–12134.

Zhu, J.-Y., Park, T., Isola, P. & Efros, A. A. (2017), Unpaired image-to-image translation using cycle-consistent adversarial networks, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 2223–2232.

Zurada, J. M. (1992), *Introduction to artificial neural systems*, Vol. 8, West publishing company St. Paul.