LEARNING TO DRIVE: EXPLOITING DEEP MODELS FOR AUTONOMOUS DRIVING

by

Wenjie Luo

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Learning to Drive: Exploiting Deep Models for Autonomous Driving

Wenjie Luo
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2019

Building self-driving vehicles is exciting and promising. It is going to transform the way we live and provide safety, efficiency, and mobility for everyone. In this thesis, I present a collection of our work in the direction of building smart self-driving vehicles from understanding convolutional neural networks (CNNs) to the full autonomy stacks, including perception, prediction, and planning.

First, we study the property of CNNs and provide theoretical analysis on the receptive field. We use Fourier transform and center limit theorem to show that the effective receptive field only occupies a fraction of the theoretical one, especially in deep models. This can provide insights for future CNNs design.

Next, we push the state of the art for tasks in the autonomy stacks using deep CNNs. For depth estimation with stereo cameras, we develop a deep matching network that simplifies previous approaches by using a dot-product layer and incorporates a multi-class classification loss, allowing the network to calibrate scores among larger contexts. These greatly improve the runtime and achieve much better matching results. Furthermore, we extend the idea to optical flow where the matching is done in 2D space. Combined with an instance segmentation algorithm, we achieve state-of-the-art results on KITTI optical flow benchmark.

Third, we propose new formulations for the self-driving system, *i.e.* using a joint model for 3D detection, prediction, and tracking. Our model takes a sequence of consecutive LiDAR sweeps and predicts the bounding boxes for vehicles at the current time step as well as one second into the future. Tracking is done by greedily checking the overlap between current detections and predictions from the past. This approach bridges the gap between perception and prediction to achieve better performance both in accuracy and runtime. Then, we take this one step further to construct a deep structured model for interpretable neural motion planning, where the CNN also predicts a 3D cost volume, expanding on time and x-y spatial dimensions. The final planning trajectory is generated with a sampling-based inference algorithm. We conduct offline testing and show that our approach is better than several baselines.

Acknowledgements

I still remember the first day we came to Toronto when the three of us drove a 15 foot truck from Chicago all the way to Toronto in winter. It was really cold. However, when I look back now at the past five years spent here, I feel a lot of warm moments. I feel so lucky and proud to be part of the machine learning group at the University of Toronto, with so many talented and hard-working people around. Ph.D. is not low-hanging fruit, and I could not imagine where I am right now from five years ago. All the accomplishments I made, if any, are due to the tremendous help and encouragement from people around me.

First of all, I would like to thank my advisor Raquel Urtasun, who is a fantastic researcher and human being. I thank her for all the support and guidance she provided during the past seven years, including my master studies in Chicago and Ph.D. studies in Toronto. She offered me very hands-on help in the early days, and always poked me to think mathematically. During the later stages of my Ph.D. studies, I learned more from Raquel to think broadly and systematically, to discover new problems. Also, I thank her for being an amazing role model. Even after working with her for a few years, I am still impressed by her dedication to research. Furthermore, I thank her for providing all the opportunities along the way, from showing me the way to graduate school in Chicago to providing me the opportunity to join Toronto and also the Uber self-driving R&D team.

There are also lots of other people I would like to thank. I thank Alexander G. Schwing for all the help in research and life in general. I learned from Alex to think in detail, to stay humble and care for others. I thank Jian Yao and Qinqing Zheng for helping me settle down and navigate through research in the early days. I thank Shenlong Wang and Kaustav Kundu for helpful discussions and all the fun time. I thank Yujia Li for showing me his structural thinking and coding, as well as being a good friend. I thank Min Bai, Bin Yang, and Wenyuan Zeng for working tirelessly on our projects and always pushing the limits.

I am also grateful to have the opportunity to spend time with so many talented people in the ML group over the past few years: Jimmy Ba, Marcus A. Brubaker, Lluis Castrejon, Liang-Chieh Chen, Xiaozhi Chen, Dominic Cheng, Hang Chu, Davi Frossard, Roger Grosse, Namdar Homayounfar, Jamie Ryan Kiros, Renjie Liao, Ruiyu Li, Justin Liang, Chenxi Liu, Shu Lu, Wei-chiu Ma, Chris Maddison, James Martens, Gellert Mattyus, Nina Merkle, Xiaojuan Qi, Mengye Ren, Edgar Simo-Serra, Jake Snell, Yang Song, Kevin Swersky, Nitish Srivastava, Charlie Tang, Makarand Tapaswi, Eleni Triantafillou, Ivan Vendrov, Tingwu Wang, Yuwen Xiong, Ziyu Zhang, Yukun Zhu and many others.

I would also like to thank my brilliant colleagues in Uber Advanced Technologies Group, including but not limited to Sergio Casas, Nemanja Djuric, Inmar Givoni, Rui Hu, Luke Li, Ming Liang, Siva Manivasagam, Abbas Sadat, Sean Segal, Simon Suo, Carl Wellington, Kelvin Wong.

I thank Ruslan Salakhutdinov and Martin Min for hosting internships.

I thank Sanja Fidler, Richard Zemel, David Fleet, and Stefan Roth for being my committee members, giving insightful comments on the thesis and all the help over the years.

Lastly, I would like to thank my parents for their endless love and support. Although they are thousands of miles away, they are always ready to help me whenever I need it. I am so proud to have them as parents, and I know they are also proud of me (although they do not speak English, they promise to read my thesis with the help of translation tools).

One more thing, I would like to thank my girlfriend Tianxing Xu. I thank her for all the support, for taking care of the mess at home when I am busy, for bringing love and laughter to our home, for encouraging me to do good work even though she does not know what I am doing;) Look forward to many years ahead.

Contents

Chapter 1

Introduction

Technology has significantly driven the development of our society and transformed our daily life, from the usage of fire to electricity. Self-driving vehicles have drawn more and more attention over the past few years as we keep pushing the boundaries of technology and maturing self-driving techniques. It is widely recognized that fully self-driving vehicles would bring lots of benefits, including increased safety and mobility, reduced costs, increased efficiency, etc. Safety benefits include the reduction of traffic accidents and corresponding injuries as well as damages. It can provide enhanced mobility for everyone, including children, the elderly, disabled people, etc. It is also able to increase the efficiency from the perspective of fuel usage, less congestion, better traffic flow, and significantly less requirement for parking space, leading to lower personal cost as well as a lower environmental burden.

While self-driving is exciting, it is undoubtedly a difficult task. The problems range from hardware and software to government regulations and concerns on privacy and security. While all of these are important factors, the scope of this thesis will be on autonomy software. Thus throughout this thesis, we use the term "self-driving technology" interchangeably with "autonomy software".

The development of self-driving can date back to the 1920s [149]. Significant engineering efforts were put into this area since then, and machine learning technology is playing an increasingly important role over time. It is worth mentioning the breakthrough of deep neural networks on image recognition tasks [93], where Krizhevsky *et al.* showed the superior performance of deep learning models over previous hand-engineered approaches. Since then, various deep learning models and different convolutional neural networks (CNNs) have been proposed to tackle a broad set of tasks with multiple types of sensor data such as RGB images/videos, LiDAR point clouds, and magnetic resonance images (MRI). It is certainly interesting to see how much further we can push self-driving technology with the help of deep learning models, from both accuracy and runtime perspectives. On the other hand, despite the great success of CNNs, there is not much theoretical understanding about them, making them black-box algorithms. A theoretical analysis would help better understand the behavior of CNNs and bring more intuition to architectural design for specific problems, *i.e.*, we could potentially develop new algorithms for existing tasks in the autonomy stacks for better performance on both accuracy and runtime. In addition, with more advanced and powerful tools in hand, it is also worthwhile to revisit some of the design choices we made a long time ago on the autonomy stacks, to research in a different direction that could potentially bring a revolutionary solution.

With that in mind, this thesis is tackling problems in the following three directions, with the ultimate goal of building a smarter self-driving vehicle:

Understanding CNNs: We provide theoretical analysis on the receptive field (*i.e.* field of view) of convolutional neural networks. The receptive field is a fundamental attribute of a CNN. It describes the size of the area one network can see on the input. It is crucial for model design; for example, if a network has a receptive field size smaller than an object in the image, it would not be able to detect the object. In theory, we can compute the size of the receptive field directly by considering the number of layers, convolutional kernel size and corresponding stride. However, this number might not reflect the real effective receptive field. Thus, we provide theoretical analysis on the receptive field and introduce the notion of effective receptive field size. We show that the effective receptive field size does not grow as fast as we used to think. This work provides good intuitions on modern neural network architectural design and can lead to better models for solving different tasks in the self-driving stack.

Pushing state of the art: While CNNs have been shown to outperform previous approaches in different high-level perception tasks such as image recognition and detection, they are not widely used for low-level vision tasks such as depth estimation and optimal flow. These tasks are essential for self-driving as they provide richer, critical information about the surrounding world with a cheaper camera sensor. We first proposed a deep matching network to incorporate context information for depth estimation using stereo cameras. It achieves much better matching performance in the context of depth estimation while being two orders of magnitude faster than previous methods. We also adapted it to optical flow, where the matching is performed in 2D space. Together with object-aware techniques, we achieved state-of-the-art optical flow performance on the KITTI [53] benchmark at the time of publication of the corresponding paper. These works apply deep neural networks to low-level vision tasks and achieve better results than previous approaches. It provides an excellent alternative solution for self-driving sensing with cheaper sensors.

New formulation for the autonomy stacks: The success of deep neural networks on various perception and prediction tasks brings a natural question: should we rethink about the architecture of autonomy stacks? The separation of perception, prediction, and planning, in general, might not be optimal. Thus, in this direction, we look at an alternative approach to formulate the problem. Firstly, we propose a joint model for perception and short-term prediction. In particular, we train a neural network to only take a few consecutive LiDAR sweeps of data as input and output 3D detection, tracking across time as well as the prediction into the future. This is a real-time solution with the advantage of jointly training for better performance, and uncertainty can be propagated throughout the whole network. Secondly, we take this one step further to predict a time-dependent spatial cost-volume. The cost-volume is widely used in planning to tell whether the self-driving vehicle should be driving on a specific location at a particular time. Given the predicted cost-volume, we utilize a sampling-based inference algorithm with the help of a trajectory sampler to generate the final planning trajectory. Together, these give an interpretable neural planner that consumes raw LiDAR data and a high-definition (HD) map. The model also provides intermediate 3D detection and prediction results for interpretability purposes.

1.1 Background

The physical region one lives in on a daily basis has expanded dramatically over the past decades. There is an increasing demand for mobility, resulting in more and more people relying on vehicles every day. It is not difficult to imagine that self-driving technologies can benefit our society. The potential benefits include increased safety, mobility, efficiency, and user experience, in addition to reduced costs and environmental impact.

To be more specific, human drivers inevitably make mistakes for various reasons, including distraction, lack

of sleep, and alcohol consumption. However, a self-driving vehicle controlled by a computer would not feel tired, *i.e.*, it will maintain the same performance over time. As a result, self-driving vehicles would provide a much safer driving service statistically. Secondly, human drivers need serious training for different driving scenarios, and there are minimum and maximum age limits. In Ontario (one of the largest provinces in Canada), one has to practice at least one year with an experienced driver before taking a road test. The driver's license system also has different categories, which require different training for operating different types of standard vehicles, such as passenger vehicles, school buses, etc. It is indeed a nontrivial task to obtain a driver's license; thus a self-driving vehicle could provide mobility to a much broader set of people, including children, the elderly, disabled people or others unable to drive. Thirdly, as we have more vehicles on the road, we would experience more congestion and traffic jams. In busy or confusing areas such as intersections, traffic lights are used to control the traffic flow. There are also efforts using AI techniques to predict the traffic flow and make better control signals accordingly. However, this is not an optimal solution, as vehicles on the road still need to decelerate and accelerate constantly. In a world with only self-driving vehicles, it can be more collaborative so that vehicles coming from any direction can go through an intersection without stopping, making traffic flow more efficient. As a result, better traffic flow permits a better usage of fuel, reducing the cost for everyone while being more environmentally friendly. Self-driving vehicles also facilitate business models of transportation as a service, especially via the sharing economy. Furthermore, vehicles are only used 5% of the time on average, *i.e.*, vehicles are idle in parking lots most of the time. Thus, one self-driving vehicle can serve many people to not only save the cost of owning/maintaining a vehicle, but also the space for parking.

While the future world of self-driving vehicles looks very exciting, it also introduces new problems, such as liability, legal frameworks, government regulations, privacy, security as well as the potential loss of driving-related jobs in the transportation industry. The liability issue brings up a difficult but realistic question: if a self-driving vehicle causes an accident, who should be blamed? It also involves legal frameworks and government regulations to ensure that each self-driving vehicle on the road has been properly tested and has sufficient backup plans when unexpected issues arise. This is even more crucial during the transition phase when we have both human drivers and self-driving vehicles on the road. Additionally, as we enter the cyber world and rely more and more on computers, privacy and security become an important issue. A hacker could potentially get the trace of specific users or even take over the control of the self-driving vehicles. Another concern, which is generally relevant for all AI technology, is the loss of related human jobs. As one can imagine, self-driving technology will reduce the demand for human drivers. Despite the various problems related to self-driving vehicles, we are optimistic that society can gradually adapt to new technology, as we have seen many times throughout history.

1.1.1 Definition of Self-Driving:

The self-driving vehicle (sometimes referred to as a robot car, autonomous car, or driverless car) is a vehicle that can sense the surrounding environment and move to desired destinations with little or no human input. There are different definitions for the level of self-driving-ness, and the widely used one is from the Society of Automotive Engineers (SAE), which consists of 6 levels [179] (SAE J3016 Standard).

- Level 0: Automated system issues warnings and may momentarily intervene but has no sustained vehicle control.
- Level 1 ("hands on"): The driver and the automated system share control of the vehicle. Examples are Adaptive Cruise Control (ACC), where the driver controls steering and the automated system controls speed; and Parking Assistance, where steering is automated while speed is under manual control. The driver must be

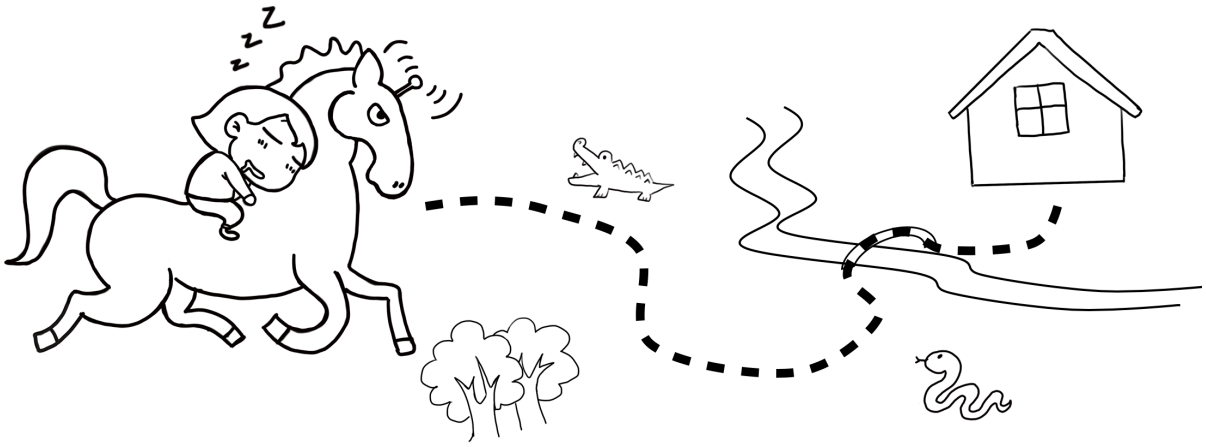


Figure 1.1: Decodes ago, horses were trained to take us home. Drawing by Tianxing.

ready to retake full control at any time. Lane Keeping Assistance (LKA) Type II is a further example of level 1 self-driving.

- Level 2 ("hands off"): The automated system takes full control of the vehicle (accelerating, braking, and steering). The driver must monitor the driving and be prepared to intervene immediately at any time if the automated system fails to respond properly. The shorthand "hands off" is not meant to be taken literally. In fact, contact between hand and wheel is often mandatory during SAE 2 driving, to confirm that the driver is ready to intervene.

- Level 3 ("eyes off"): The driver can safely turn their attention away from the driving tasks, *e.g.*, the driver can text or watch a movie. The vehicle will handle situations that call for an immediate response, like emergency braking. The driver must still be prepared to intervene within some limited time, specified by the manufacturer, when called upon by the vehicle to do so. As an example, the 2018 Audi A8 Luxury Sedan was the first commercial car to claim to be capable of level 3 self-driving. This particular car has a so-called Traffic Jam Pilot. When activated by the human driver, the car takes full control of all aspects of driving in slow-moving traffic below 60 kilometers per hour (37 mph). The function works only on highways with a physical barrier separating one stream of traffic from oncoming traffic.

- Level 4 ("mind off"): As level 3, but no driver attention is ever required for safety, *e.g.*, the driver may safely go to sleep or leave the driver's seat. Self-driving is supported only in limited spatial areas (geofenced) or under special circumstances, like traffic jams. Outside of these areas or circumstances, the vehicle must be able to safely abort the trip, *e.g.*, park the car, if the driver does not retake control.

- Level 5 ("steering wheel optional"): No human intervention is required at all.

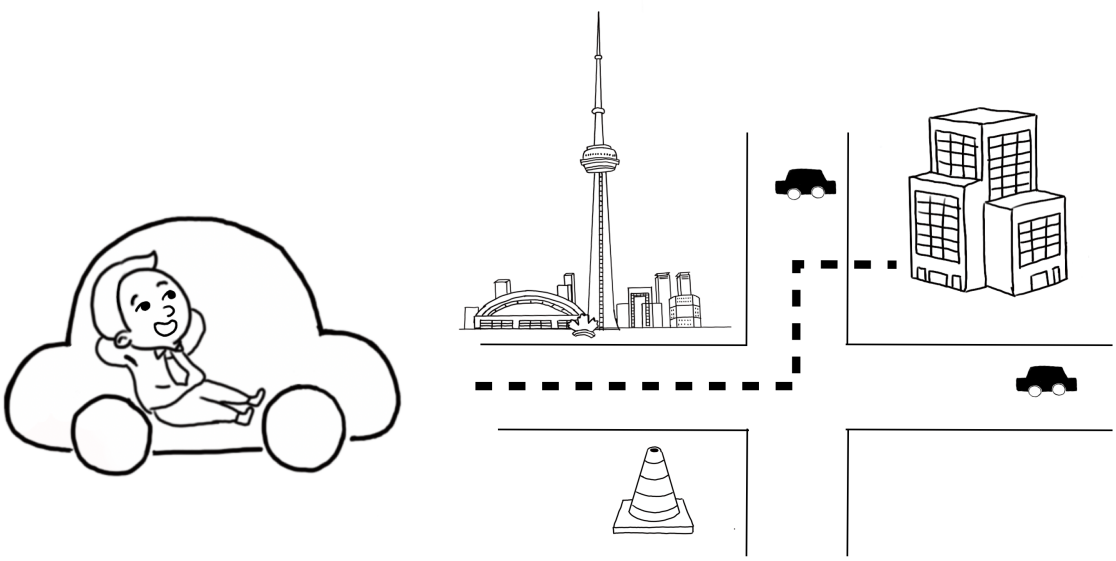


Figure 1.2: Nowadays, the self-driving vehicle is developed to take everyone and everything everywhere. Drawing by Tianxing.

1.1.2 History

Developing robots to help humans with various tasks has always been a focus for both the research community and industry. We have seen a great number of robots invented that indeed make our daily life easier and more enjoyable. For example, we have vacuum robots that can clean the house automatically, drones that can follow people and take pictures or videos automatically, and autopilot planes that fly millions of people every day. Among all these areas, mobility is one of the most important factors. We used to train horses to take us home (Fig. 1.1). Nowadays we are training robots to move everyone, everything everywhere (Fig. 1.2).

While it may seem like self-driving vehicles have gone from blockbuster or science fiction to reality in just a few years, and the technology emerged overnight, the development of self-driving has a long history. It dates back to at least the 1920s [149], when people started talking about it. More promising attempts took place in the 1950s, and the development of self-driving vehicles has taken on a faster pace. In the early 1980s, Ernst Dickmanns and his team equipped a Mercedes-Benz van with cameras and other sensors and re-engineered it to control the steering wheel, throttle, and brake via computer command. They performed initial experiments in Bavaria on streets without traffic. Later in 1986/1987, they made the car capable of driving fully by itself up to 96 kilometers per hour, but only on empty roads. In the early 1990s, Dean Pomerleau from Carnegie Mellon University proposed a neural network to output the steering angle directly from raw image input [136]. This is the first known attempt of self-driving vehicles using a neural network with an end-to-end training procedure. It was demonstrated to work in certain conditions and was far more efficient than alternative attempts at the time, which manually divided images into “road” and “non-road” categories. Furthermore, in 1995, Pomerleau and fellow researcher Todd Jochem took their self-driving vehicle on the road. They managed to drive autonomously approximately 3K miles coast-to-coast from Pittsburgh, Pennsylvania to San Diego, California (although they needed to control speed and to brake themselves), of which 98.2 % was autonomously controlled. It was dubbed “No Hands Across America”. In the 2000s, the U.S. government started to get involved. It funded military efforts

to demonstrate the ability of crewless ground vehicles to navigate miles without crashing into obstacles such as trees and rocks. There is also a demonstration for a hierarchical system, where not only individual vehicles are controlled, but also groups of vehicles are coordinated with each other in order to achieve given high-level goals together. Entering the 2010s, self-driving vehicles have become superstars. Advances in artificial intelligence technology, including algorithms, computing power, and huge amounts of data, all play important roles in this progress. Numerous major companies and research organizations have developed working self-driving prototype vehicles, including car manufacturers such as Mercedes-Benz, General Motors, Continental Automotive Systems, Autoliv Inc., Bosch, Nissan, Toyota, Audi, and Volvo. Technology companies such as Google, Baidu, and Uber started self-driving efforts as well. The bright future of self-driving also spins off many startups, such as Zoox, Pony.ai, Argo.ai, Aurora, and Nuro. We refer interested readers to related materials and the references [178], a, b, c. In the following, we will briefly talk about two noticeable projects in the history of self-driving development.

DARPA Grand Challenge: The DARPA Grand Challenge is a series of prize competitions for developing long-distance autonomous vehicles. They were hosted by the Defense Advanced Research Projects Agency (DARPA), which is the most prominent research organization of the United States Department of Defense. The U.S. Congress authorized DARPA to offer prize money of one million dollars for the first Grand Challenge to facilitate robotic development, with the ultimate goal of making one-third of ground military forces autonomous by 2015. Following the 2004 event, the prize money increased to two million [176]. The competition was open to teams and organizations from around the world, as long as there was at least one U.S. citizen on the team. It attracted major universities and large corporate interests such as CMU, Stanford, and MIT to participate. The first DARPA Grand Challenge (2004) was created to spur the development of technologies needed to create the first fully autonomous ground vehicles capable of completing a substantial off-road course within a limited time. None of the robot teams at the time could finish the route of 240 km along the Interstate 15. The third event (2007), the DARPA Urban Challenge, extended the initial Challenge to autonomous operation in a mock urban environment, where multiple teams were able to finish the 96 km course. In particular, Tartan Racing, a team from CMU and GM with a 2007 Chevy Tahoe car, won the first place and claimed the 2 million dollar prize. The second-place finisher, earning the 1 million dollar prize, was the Stanford Racing Team with their entry "Junior," a 2006 Volkswagen Passat. Coming in third place was team VictorTango, winning the $500,000 prizes with their 2005 Ford Escape hybrid, "Odin." The most recent Challenge, the 2012 DARPA Robotics Challenge, focused on autonomous emergency-maintenance robots. We refer interested readers to [176] for more details.

EUREKA Prometheus Project: The Eureka PROMETHEUS Project (**PRO**gra**M**me for a **E**uropean **T**raffic of **H**ighest **E**fficiency and **U**nprecedented **S**afety, 1987-1995) was a project from the European Union and was one of the largest R&D projects in the domain of self-driving vehicles. It received 749,000,000 euros in funding from various EUREKA member states. It also attracted many universities and car manufacturers to participate. As the partners recognized the wide variety of skills required for fully autonomous vehicles, the project was formulated differently for different sub-projects. In particular, the committee proposed three sub-projects on industrial research, including driver assistance by computer systems (PRO-CAR), vehicle-to-vehicle communication (PRO-NET), vehicle-to-environment communication (PRO-ROAD), and four on basic research including methods and systems of artificial intelligence (PRO-ART), custom hardware for intelligent processing in vehicles (PRO-CHIP), methods and standards for communication (PRO-COM), traffic scenario for new assessments, and the introduction of new systems (PRO-GEN). The project spurred progress on different aspects of self-driving technology, including vision enhancement, lane keeping support, and collision avoidance. We refer interested readers to [177]

for more details.

1.2 Challenges

From decades of development on self-driving technology, it is now clear that a full self-driving system involves a diverse and advanced set of techniques, including hardware design and maintenance; accurate high-definition mapping; accurate and real-time localization systems; robust perception pipelines, including but not limited to detection; tracking; and prediction, as well as planning and control modules that compute plausible trajectories and execute them in real-world environments. While all of them have drawn lots of attention and been widely studied in the research community, in the following, I highlight some of the challenges this thesis will focus on.

Sensing: When a human driver is on the road, she or he would first take information from the surrounding world using human sensors, *e.g.* eyes. Human eyes can be treated as stereo cameras, and our visual system can correspondingly infer the depth of the scene from these sensor inputs. Similarly, this 3D information about the surrounding environment is also crucial for self-driving vehicles as we need to navigate to destinations without collision and obey rules on the road. Currently, there are two different ways to get 3D information: one is to use a 3D sensor such as LiDAR, while the other is to use stereo cameras similar to the setting of the human visual system. Using LiDAR sensors provides very accurate 3D information as they actively shoot out light and measure the distance via the time-lapse upon receiving the reflection. However, due to the nature of this approach and the limitation of current LiDAR technology, the point cloud we receive from one LiDAR sweep is usually very sparse, and the sparsity grows exponentially over distance. The working range for a standard Velodyne HDL-64E is only up to 100 meters. The other cheaper option is to use stereo cameras. However, they bring more significant challenges—estimating depth from stereo cameras is not a trivial task. From geometry, we know that depth is inversely proportional to the disparity between rectified images. Thus, it is often treated as a matching problem between image patches. We have seen useful algorithms in datasets with manually controlled environments such as the Middlebury dataset; however, in the real-world environment, the performance is heavily affected and limited by specularities, occlusions, repetitive patterns, and even saturation due to improper exposure. Additionally, self-driving systems would require real-time performance, thus leaving a very limited budget for perception in general, not to mention sensing which is only a small part of perception.

3D Detection: Given 3D information of the scene, another essential ingredient for perception is 3D detection, where the task is to detect surrounding objects in the 3D space. This is a challenging task as it requires recognizing the target objects and localizing them in 3D space, with excellent accuracy and low latency. Failing to do either of these tasks could lead to a catastrophic failure of the whole system. Imagine if the self-driving system fails to detect an obstacle in front—it could just run into it, causing damage or even injuries. The 3D detection task has two main-stream approaches based on what type of 3D sensing framework it relies on, *i.e.*, the first one utilizes LiDAR sensor data and the other one utilizes information from stereo images. The LiDAR detection approach can produce more accurate 3D detection results since the sensor data (LiDAR point cloud) can provide more accurate depth information. The limitation for utilizing LiDAR is that currently it only works for short distances, *e.g.* a commonly-used Velodyne LiDAR could only give very sparse LiDAR points at the distance beyond 100 meters, bringing extra challenges on long-range detections. In practice, the effective range could be even shorter due to different environmental and weather conditions. Additionally, the computation limit is also a key challenge for LiDAR-based methods, as LiDAR data genuinely lives in 3D space. A naive 3D convolution approach would be

orders of magnitude slower than what we can afford for self-driving systems. On the other hand, image-based methods have an advantage on long-distance objects as we could easily have access to high-resolution cameras, as well as cameras designed specifically for long range. However, the drawback is also clear that high-resolution brings heavy computation, and the depth information could be very noisy due to the limitation of current stereo techniques.

Prediction: Knowing where all the surrounding objects are only at the current time (*i.e.* 3D detection) is not enough; a self-driving system would also need to know where each object goes into the future. This allows planning for a safe and smooth trajectory that avoids collision and provides a comfortable riding experience, *i.e.*, without jerk and sharp turns. There are several challenges for prediction. Firstly, it is a multi-modal problem. Each object could potentially take on very different trajectories, even from the same starting point. For example, a pedestrian standing at the intersection could go straight to cross the intersection or stand there waiting for others; similarly, a vehicle at an intersection could either go straight or make a turn. The self-driving vehicle needs to take into account all these possibilities to make its corresponding movement. Secondly, since there are multiple objects on the road, the prediction system would need to consider the interaction among them. This is also a non-trivial task for human drivers, and in practice, we often use hand gestures or make eye-contact to signal each other. For the self-driving system, however, it is challenging to model the interaction as the possibility of all interactions grows exponentially with time and number of objects. Also, each object's action and location live in continuous space, which makes the problem even more challenging. Thirdly, the prediction of other objects should also comply with traffic rules most of the time. This brings extra challenges to the system, especially in a dynamically changing environment.

Planning & Cost Map: The planning module takes the output from previous perception and prediction to plan a safe, smooth, and progressive trajectory towards a given destination. This is a difficult task as we need to comply with traffic rules and other dynamic objects. It is often achieved by manually designing a cost map representing the 'goodness' of each position that the self-driving car can take within the planning horizon. As there is no ground-truth for this cost map, it requires enormous engineering effort and domain knowledge to design a proper cost map that can encode the rules and driving behavior we want. In practice, this process often takes a long time and does not scale to all kinds of driving scenarios. In addition, we need to consider the mistakes from previous perception and prediction systems. A robust planning system would require rigorous safety checks and consideration of both uncertainties from previous modules and the potential inability to generate feasible trajectory due to the noise in perception and prediction results. Currently, there is still no perfect solution for all of these issues. Furthermore, in contrast to supervised tasks, such as image classification and detection, there is no standard metric for planning. Simply taking a ride in the self-driving vehicle cannot provide comprehensive information about the status of the system. It requires a sophisticated simulation system to test it, usually at the level of driving millions or even billions of miles.

Interpretability: While deep neural networks have been successfully applied to a wide variety of problems, their lack of interpretability is a common criticism. This is a crucial factor for safety-critical applications such as self-driving vehicles. It is essential for both online and offline settings. In an online setting, interpretability can provide safety checks and make users confident about the system. In an offline setting, interpretability can give a better analysis of failure cases. However, getting the interpretability of the decision made by a deep neural network is not obvious, as it is a highly non-linear function. Previously, researchers have conducted

different experiments and proposed various ways to understand neural networks from the perspective of activation visualization [196, 203, 118] showing how the network evolves during training and how different layers can extract different levels of abstraction. There are also studies about neural networks from the perspective of adversarial samples [131, 48, 126] showing that neural networks are not as robust as we used to think. However, these studies are mostly on the empirical side, while theoretical analysis on understanding neural networks remains a challenging and promising research direction.

All these problems involve fundamental computer vision techniques ranging from the low-level perception of depth estimation and optical flow to the high-level perception of object detection, recognition, and tracking. It also requires better machine learning techniques, including but not limited to how to perform training and inference efficiently, use data more efficiently, build insight into the model, and have interpretability of all these algorithms.

1.3 Contributions

The contribution of this thesis can be summarized as *moving from manually tuned rules to learning-based methods for self-driving*. Previous self-driving techniques utilized many manually-tuned rules or hand-crafted features for each of the building blocks; for example, Birchfield and Tomasi[12] uses slanted surfaces for depth estimation, and hand-crafted features such as SIFT[110] and HOG [40] were among the best choice for detection tasks in the early days. On the other hand, with the help of large-scale datasets [41, 53] and advanced computing architecture, especially NVIDIA GPUs and CUDA computing platform, deep neural networks achieve amazing results on various tasks [72, 6, 93, 69]. In this thesis, I present a list of my work during my Ph.D. studies to provide a theoretical understanding of CNNs from the perspective of the receptive field, improving the performance on existing tasks in the self-driving stack, as well as proposing new formulations that jointly reason on multiple tasks. All these efforts are towards the goal of building a smarter self-driving vehicle. In detail:

- CNNs have been widely used for various tasks ranging from dense prediction of depth estimation [195] and segmentation [23] to high-level tasks of image classification [93] and detection [140, 69]. However, they are not necessarily well understood, especially regarding the receptive field, which plays an important role in designing network architectures. In Chapter 2, we give a theoretical analysis on the receptive field of CNNs, introduce the notion of the effective receptive field (ERF), which is the area that has a non-negligible influence on the output. We prove, using Fourier analysis and central limit theorem, that the effective receptive field follows a Gaussian distribution and the size shrinks at the rate of $\frac{1}{\sqrt{n}}$ *w.r.t.* theoretical receptive field size, where n is the number of layers. We also discuss ERF behavior on different non-linearities, dropout, sub-sampling, and dilated convolution.

- Presented in Chapter 3, we develop a deep learning method for depth estimation from stereo images. We utilize the geometric property for stereo images and formulate the depth estimation as image patches matching along the epipolar line. Previously, Zbontar and LeCun[195] adopt a siamese network to perform deep matching; however, it comes with the limitations of being too slow, requiring 1 minute of GPU computation, and providing inadequate matching because the scores are not calibrated along the epipolar line. Instead, we utilize a simple dot product on top of the siamese network as our matching network, allowing us to do deep matching two orders of magnitude faster. Also, we treat it as a multi-class classification with cross entropy loss instead of a binary classification. This gives a much better matching performance as the matching scores are calibrated and the network can exploit context information as well. Experiments on KITTI [53] show significant improvement in matching.

- In Chapter 4, we take the previous model one step further to do optical flow, where the problem can be formulated as patch matching in 2D space. Previous methods exploit different matching techniques but ignore the semantic information of the scene. In this work, we extend our deep matching method to 2D space and introduce a voting mechanism for post-processing. This can filter out regions with noisy matching results caused by occlusions, specularities, etc. Furthermore, we exploit the semantic information of the scene by splitting dynamic objects and infer their rigid transformations separately. Experiments on KITTI [53] show state-of-the-art performance at the time.

- Equipped with neural networks, we are rethinking the design choices of the past on the self-driving stack, especially from the perspective of the split between different subtasks, including 3D detection, tracking, and prediction. Previously there are lots of successful algorithms in each of these domains [138, 107, 50, 116, 4]; however, optimizing different tasks separately cannot guarantee a good solution for the whole system, as uncertainty can hardly be propagated through different modules, and the heavy feature computation cannot be shared between different modules. In Chapter 5, we take a different approach and propose to use one model for both perception and prediction tasks. It utilizes temporal information and optimizes the objective function of multiple tasks jointly. The benefits are two-fold. First, it is more accurate as we optimize jointly. Second, it is much faster, achieving real-time performance, as heavy feature computation can be shared between both perception and prediction. Experiments on a large-scale self-driving dataset show our method achieves better results on detection, tracking, and prediction tasks than previous real-time solutions.

- Following the direction of Chapter 5, in Chapter 6, we take it one step further and build an end-to-end neural planner with interpretable intermediate results. Previous work on planning took perception and prediction output as input to construct the cost map manually and solved the corresponding constrained optimization problem. However, the design of manual cost functions requires lots of engineering efforts, and it is not easy to scale to all driving scenarios. In this work, we design a holistic model that takes as input raw LIDAR data and an HD map and produces interpretable intermediate representations in the form of 3D detections and their future trajectories, as well as a cost volume defining the goodness of each position that the self-driving car can take within the planning horizon. We then sample a set of diverse, physically-possible trajectories and choose the one with the minimum learned cost. Importantly, our cost volume can naturally capture multi-modality and uncertainty. We demonstrate the effectiveness of our approach in real-world driving data captured in several cities in North America. Our experiments show that the learned cost volume can generate safer planning than all the baselines.

1.4 Relationship to Published Work

All related research works in this thesis have been peer-reviewed and published in computer vision or machine learning related conferences. In the following, I list the corresponding papers for each chapter. Note * denotes equal contribution:

Chapter 2: Wenjie Luo*, Yujia Li*, Raquel Urtasun, and Richard Zemel. Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. Poster in Neural Information Processing Systems (NIPS), 2016.

Chapter 3: Wenjie Luo, Alexander G. Schwing, and Raquel Urtasun. Efficient Deep Learning for Stereo Matching. Spotlight in Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

Chapter 4: Min Bai*, Wenjie Luo*, Kaustav Kundu and Raquel Urtasun. Exploiting Semantic Information and Deep Matching for Optical Flow. Poster in European Conference on Computer Vision (ECCV), 2016.

Chapter 5: Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net. Oral in Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

Chapter 6: Wenyuan Zeng*, Wenjie Luo*, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, Raquel Urtasun. End-to-end Interpretable Neural Motion Planner. Oral in Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

1.5 Thesis Structure

For the rest of this thesis, I will present the main technical part in Chapter 2-6 in the following order: the study of convolutional neural networks from the receptive field perspective; depth estimation from stereo images with our deep matching networks; optical flow using monocular camera using our object-aware algorithm; joint 3D detection, tracking and prediction framework and finally the end-to-end neural interpretable planner that learns the time-dependent cost-volume and generates corresponding planning trajectory. Lastly, I conclude in Chapter 7 with future research directions.

Chapter 2

Understanding CNNs from Effective Receptive Field

Self-driving requires solving a broad set of problems in computer vision, ranging from low-level depth estimation and flow estimation to high-level tasks, such as 3D detection and tracking. One of the fundamental problems in computer vision is how to extract useful features from sensor data, such as images. Researchers have developed different hand-crafted features for various tasks using domain knowledge of the problem [110, 40]. These approaches dominated all kinds of computer vision tasks for a long time until the emergence of convolutional neural networks (CNNs). This was especially the case since Krizhevsky *et al.*[93] showed ground-breaking results on Imagenet challenge.

The availability of large-scale datasets and powerful computation resources, especially GPUs, certainly played an essential role in this process. At the same time, convolutional neural networks do have certain design advantages. Firstly, they can learn to adjust their weights to extract desired features. Secondly, the multi-layer structure allows the network to extract features from different semantic levels. For example, in image recognition tasks, the network would learn to extract simple features in the first few layers, such as lines and circles, while for the top layers, the network would be tailored for high-level features. Thirdly, it shares parameters among kernels at different spatial locations. This gives spatial invariance that benefits most vision tasks and makes the training/optimization easier since it reduces the total number of parameters significantly.

Despite the success of convolutional neural networks, the theoretical understanding underneath is still limited, and it is often treated as a black-box algorithm. Researchers have been looking at different visualization techniques on convolutional neural networks [154, 204, 118] as well as developing adversary samples [94, 131], in order to get a better understanding of them.

In this chapter, we take a different perspective and look at the *receptive field* for theoretical analysis. The *Receptive field*, or *field of view*, is one of the basic concepts in deep convolutional neural networks. Each unit in a particular layer in the network has a receptive field over the original input tensor (*e.g.* images). Unlike in fully connected networks, where the value of each unit depends on the entire input to the network, a unit in convolutional networks only depends on a region of the input. This region in the input is the receptive field for that unit.

The concept of the receptive field is important for understanding and diagnosing how deep CNNs work. Since anywhere in an input image outside the receptive field of a unit does not affect the value of that unit, it is necessary to control the receptive field carefully to ensure that it covers the entire relevant image region. In many tasks,

especially dense prediction tasks like semantic segmentation, stereo depth estimation, and optical flow estimation, it is critical for each output pixel to have a big receptive field, such that no important information is missing when making the prediction. Consider the task of semantic segmentation of an urban scene for autonomous driving. The output pixels should have a receptive field big enough to cover large objects that span a significant region in the image, like buses and trucks that are close to the camera. If these receptive fields are too small, prediction errors are unavoidable. In fact, this is indeed one of the common failure modes of deep CNN based segmentation systems. A similar argument applies equally well for object detection. Even for image-level recognition tasks like object classification, where the information across the whole image is usually aggregated with a few fully connected layers on top of the convolution feature maps, having a big receptive field for the units in the convolutional layers may still be beneficial. With bigger receptive fields, the convolutional layers can aggregate information across larger input regions, and the network can potentially make use of the convolutional layers more efficiently.

The receptive field size of a unit can be increased in many ways. One option is to stack more layers to make the network deeper, which increases the receptive field size linearly in theory, as each extra layer increases the receptive field size by the kernel size. Sub-sampling, on the other hand, increases the receptive field size multiplicatively. Modern deep CNN architectures like the VGG networks [153] and Residual Networks [70, 69] use a combination of these techniques.

However, simply stacking more layers might not be as good as we used to think. In this chapter, we carefully study the receptive field of deep CNNs, focusing on problems in which there are many output units (*e.g.* pixel-wise dense prediction, such as segmentation). In particular, we discover that not all pixels in a receptive field contribute equally to an output unit's response. We measure the effect of an input pixel on an output pixel by the magnitude of the gradient backpropagated directly from that output to the input pixel. The magnitude of the gradient tells us how much a change in the input changes the output and is a natural measure of importance. Intuitively, it is easy to see that pixels at the center of the receptive field have a much larger impact on the output. In the forward pass, central pixels can propagate information to the output through many different paths, while the pixels in the outer area of the receptive field have very few paths to propagate its impact. In the backward pass, gradients from an output unit are propagated across all the paths, and therefore the central pixels have a much larger magnitude for the gradient from that output.

This observation leads us to further study the distribution of impact within a receptive field on the output. Surprisingly, we can prove that in many cases, the distribution of impact in a receptive field distributes as a Gaussian. Note that in earlier work [182], this Gaussian assumption about a receptive field is used without theoretical justification. This result further leads to some intriguing findings, in particular, that the effective area in the receptive field, which we call the *effective receptive field*, only occupies a fraction of the *theoretical receptive field*, since Gaussian distributions generally decay quickly from the center.

In the following, we first present the theoretical study in Section 2.1 followed by some empirical observations in Section 2.2 that support our theory and aim to understand the effective receptive field for deep CNNs better. In addition, we discuss a few potential ways to increase the effective receptive field size in Section 2.3.

2.1 Properties of Effective Receptive Fields

Convolutional neural networks were inspired by biological processes in that the connectivity patterns between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. In CNNs, the receptive field of a neuron at the top layer corresponds to all the locations on the input, where there exists a path from the input location to

the target neuron. In theory, it can be easily computed with standard convolution kernels. For example, if the kernel has a spatial size of 3x3, stacking n layers of such a kernel will give a receptive field of size: $(3-1)*n+1$ by $(3-1)*n+1$.

In this chapter, we aim to mathematically characterize how much each input pixel in a receptive field can impact the output of a unit n layers up the network, and study how the impact distributes within the receptive field of that output unit. To simplify notation, we consider only a single channel on each layer, but similar results can be easily derived for convolutional layers with more input and output channels.

Assume the pixels on each layer are indexed by (i, j), with their center at $(0, 0)$. Denote the (i, j)th pixel on the pth layer as $x^p_{i,j}$, with $x^0_{i,j}$ as the input to the network, and $y_{i,j} = x^n_{i,j}$ as the output on the nth layer. We want to measure how much each $x^0_{i,j}$ contributes to $y_{0,0}$. We define the *effective receptive field* (ERF) of this central output unit as the region containing any input pixel with a non-negligible impact on that unit.

The measure of the impact we use in this chapter is the partial derivative $\partial y_{0,0}/\partial x^0_{i,j}$. It measures how much $y_{0,0}$ changes as $x^0_{i,j}$ changes by a small amount; it is, therefore, a natural measure of the importance of $x^0_{i,j}$ with respect to $y_{0,0}$. However, this measure depends not only on the weights of the network, but is in most cases also input-dependent, so most of our results will be presented in terms of expectations over the input distribution.

The partial derivative $\partial y_{0,0}/\partial x^0_{i,j}$ can be computed with back-propagation. In the standard setting, back-propagation propagates the error gradient with respect to a certain loss function. Assuming we have an arbitrary loss l, by the chain rule we have $\frac{\partial l}{\partial x^0_{i,j}} = \sum_{i',j'} \frac{\partial l}{\partial y_{i',j'}} \frac{\partial y_{i',j'}}{\partial x^0_{i,j}}$.

Then to get the quantity $\partial y_{0,0}/\partial x^0_{i,j}$, we can set the error gradient $\partial l/\partial y_{0,0} = 1$ and $\partial l/\partial y_{i,j} = 0$ for all $i \neq 0$ and $j \neq 0$, then propagate this gradient from there back down the network. The resulting $\partial l/\partial x^0_{i,j}$ equals the desired $\partial y_{0,0}/\partial x^0_{i,j}$. Here we use the back-propagation process without an explicit loss function, and the process can be easily implemented with standard neural network tools.

In the following, we first consider linear networks, where this derivative does not depend on the input and is purely a function of the network weights and (i, j), which clearly shows how the impact of the pixels in the receptive field distributes. Then we move forward to consider more modern architecture designs and discuss the effect of nonlinear activations, dropout, sub-sampling, dilation convolution and skip connections on the ERF.

2.1.1 The simplest case: a stack of convolutional layers of weights all equal to one

Consider the case of n convolutional layers using $k \times k$ kernels with stride one, one single channel on each layer and no nonlinearity, stacked into a deep linear CNN. In this analysis, we ignore the biases on all layers. We begin by analyzing convolution kernels with weights all equal to one.

Denote $g(i, j, p) = \partial l/\partial x^p_{i,j}$ as the gradient on the pth layer where p ranges from 1 to $n-1$, and let $g(i, j, n) = \partial l/\partial y_{i,j}$. Then $g(\cdot, \cdot, 0)$ is the desired gradient image of the input. The back-propagation process effectively convolves $g(\cdot, \cdot, p)$ with the $k \times k$ kernel to get $g(, , p-1)$ for each p.

In this particular case, the kernel is a $k \times k$ matrix of 1's, so the 2D convolution can be decomposed into the product of two 1D convolutions. Therefore, we focus exclusively on the 1D case. We have the initial gradient signal $u(t)$ and kernel $v(t)$ formally defined as

$$u(t) = \delta(t), \quad v(t) = \sum_{m=0}^{k-1} \delta(t-m), \quad \text{where } \delta(t) = \begin{cases} 1, & t = 0 \\ 0, & t \neq 0 \end{cases} \tag{2.1}$$

and $t = 0, 1, -1, 2, -2, ...$ indexes the pixels.

The gradient signal on the input pixels is simply $o = u * v * \cdots * v$, convolving u with n such v's. To compute

this convolution, we can use the Discrete Time Fourier Transform to convert the signals into the Fourier domain, and obtain

$$U(\omega) = \sum_{t=-\infty}^{\infty} u(t)e^{-j\omega t} = 1, \quad V(\omega) = \sum_{t=-\infty}^{\infty} v(t)e^{-j\omega t} = \sum_{m=0}^{k-1} e^{-j\omega m} \tag{2.2}$$

Applying the convolution theorem, we have the Fourier transform of o is

$$\mathcal{F}(o) = \mathcal{F}(u * v * \cdots * v)(\omega) = U(\omega) \cdot V(\omega)^n = \left(\sum_{m=0}^{k-1} e^{-j\omega m} \right)^n \tag{2.3}$$

Next, we need to apply the inverse Fourier transform to get back $o(t)$:

$$o(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \left(\sum_{m=0}^{k-1} e^{-j\omega m} \right)^n e^{j\omega t} \mathrm{d}\omega \tag{2.4}$$

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-j\omega s} e^{j\omega t} \mathrm{d}\omega = \begin{cases} 1, & s = t \\ 0, & s \neq t \end{cases} \tag{2.5}$$

We can see that $o(t)$ is simply the coefficient of $e^{-j\omega t}$ in the expansion of $\left(\sum_{m=0}^{k-1} e^{-j\omega m} \right)^n$.

Case $k = 2$: Now let's consider the simplest nontrivial case of $k = 2$, where $\left(\sum_{m=0}^{k-1} e^{-j\omega m} \right)^n = (1 + e^{-j\omega})^n$. The coefficient for $e^{-j\omega t}$ is then the standard binomial coefficient $\binom{n}{t}$, so $o(t) = \binom{n}{t}$. It is quite well known that binomial coefficients distributes with respect to t like a Gaussian as n becomes large (see for example [109]), which means the scale of the coefficients decays as a squared exponential as t deviates from the center. When multiplying two 1D Gaussian together, we get a 2D Gaussian, therefore in this case, the gradient on the input plane is distributed like a 2D Gaussian.

Case $k > 2$: In this case the coefficients are known as "extended binomial coefficients" or "polynomial coefficients", and they too distribute like Gaussian, see for example [46, 129]. This is included as a special case for the more general case presented later in Section 2.1.3.

2.1.2 Random weights

Now let's consider the case of random weights. In general, we have

$$g(i, j, p-1) = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} w_{a,b}^p g(i + a, i + b, p) \tag{2.6}$$

with pixel indices properly shifted for clarity, and $w_{a,b}^p$ is the convolution weight at (a, b) in the convolution kernel on layer p. At each layer, the initial weights are independently drawn from a fixed distribution with zero mean and variance C. We assume that the gradients g are independent of the weights. This assumption is in general not true if the network contains nonlinearities, but for linear networks, these assumptions hold. As $\mathbb{E}_w[w_{a,b}^p] = 0$, we can then compute the expectation

$$\mathbb{E}_{w,input}[g(i, j, p-1)] = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} \mathbb{E}_w[w_{a,b}^p] \mathbb{E}_{input}[g(i + a, i + b, p)] = 0, \quad \forall p \tag{2.7}$$

Here the expectation is taken over w distribution as well as the input data distribution. The variance is more interesting, as

$$\mathrm{Var}[g(i,j,p-1)] = \sum_{a=0}^{k-1}\sum_{b=0}^{k-1}\mathrm{Var}[w_{a,b}^{p}]\mathrm{Var}[g(i+a,i+b,p)] = C\sum_{a=0}^{k-1}\sum_{b=0}^{k-1}\mathrm{Var}[g(i+a,i+b,p)] \tag{2.8}$$

This is equivalent to convolving the gradient variance image $\mathrm{Var}[g(,,p)]$ with a $k \times k$ convolution kernel full of 1's, and then multiplying by C to get $\mathrm{Var}[g(,,p-1)]$.

Based on this we can apply the same analysis as in Section 2.1.1 on the gradient variance images. The conclusions carry over easily that $\mathrm{Var}[g(.,.,0)]$ has a Gaussian shape, with only a slight change of having an extra C^n constant factor multiplier on the variance gradient images, which does not affect the relative distribution within a receptive field.

2.1.3 Non-uniform kernels

More generally, each pixel in the kernel window can have different weights, or as in the random weight case, they may have different variances. Let's again consider the 1D case, $u(t) = \delta(t)$ as before, and the kernel signal $v(t) = \sum_{m=0}^{k-1} w(m)\delta(t-m)$, where $w(m)$ is the weight for the mth pixel in the kernel. Without loss of generality, we can assume the weights are normalized, *i.e.* $\sum_m w(m) = 1$.

Applying the Fourier transform and convolution theorem as before, we get

$$U(\omega)\cdot V(\omega)\cdots V(\omega) = \left(\sum_{m=0}^{k-1} w(m)e^{-j\omega m}\right)^n \tag{2.9}$$

the space domain signal $o(t)$ is again the coefficient of $e^{-j\omega t}$ in the expansion; the only difference is that the $e^{-j\omega m}$ terms are weighted by $w(m)$.

These coefficients turn out to be well studied in the combinatorics literature, see for example [46] and the references therein for more details. In [46], it was shown that if $w(m)$ are normalized, then $o(t)$ exactly equals the probability $p(S_n = t)$, where $S_n = \sum_{i=1}^{n} X_i$ and X_i's are i.i.d. multinomial variables distributed according to $w(m)$'s, *i.e.* $p(X_i = m) = w(m)$. Notice the analysis there requires that $w(m) > 0$. But we can reduce to variance analysis for the random weight case, where the variances are always nonnegative while the weights can be negative. The analysis for negative $w(m)$ is more difficult and is left to future work. However, empirically, we found the implications of the analysis in this section still applies reasonably well to networks with negative weights.

From the central limit theorem point of view, as $n \to \infty$, the distribution of $\sqrt{n}(\frac{1}{n}S_n - \mathbb{E}[X])$ converges to Gaussian $\mathcal{N}(0, \mathrm{Var}[X])$ in distribution. This means, for a given n large enough, S_n is going to be roughly Gaussian with mean $n\mathbb{E}[X]$ and variance $n\mathrm{Var}[X]$. As $o(t) = p(S_n = t)$, this further implies that $o(t)$ also has a Gaussian shape. When $w(m)$'s are normalized, this Gaussian has the following mean and variance:

$$\mathbb{E}[S_n] = n\sum_{m=0}^{k-1} mw(m), \quad \mathrm{Var}[S_n] = n\left(\sum_{m=0}^{k-1} m^2 w(m) - \left(\sum_{m=0}^{k-1} mw(m)\right)^2\right) \tag{2.10}$$

This indicates that $o(t)$ decays from the center of the receptive field squared exponentially according to the Gaussian distribution. The rate of decay is related to the variance of this Gaussian. If we take one standard deviation as the *effective receptive field (ERF) size*, then this size is $\sqrt{\mathrm{Var}[S_n]} = \sqrt{n\mathrm{Var}[X_i]} = O(\sqrt{n})$.

On the other hand, as we stack more convolutional layers, the theoretical receptive field grows linearly, therefore relative to the theoretical receptive field, the ERF actually *shrinks* at a rate of $O(1/\sqrt{n})$, which we found surprising.

In the simple case of uniform weighting, we can further see that the ERF size grows linearly with kernel size k. As $w(m) = 1/k$, we have

$$\sqrt{\text{Var}[S_n]} = \sqrt{n}\sqrt{\sum_{m=0}^{k-1}\frac{m^2}{k} - \left(\sum_{m=0}^{k-1}\frac{m}{k}\right)^2} = \sqrt{\frac{n(k^2-1)}{12}} = O(k\sqrt{n}) \tag{2.11}$$

Remarks: The result derived in this section, *i.e.*, the distribution of impact within a receptive field in deep CNNs converges to Gaussian, holds under the following conditions: (1) all layers in the CNN use the same set of convolution weights. In general, this is not true; however, when we apply the analysis of variance, the weight variance on all layers are usually the same up to a constant factor. (2) The convergence derived is convergence "in distribution", as implied by the central limit theorem. This means that the cumulative probability distribution function converges to that of a Gaussian, but at any single point in space, the probability can deviate from the Gaussian. (3) The convergence result states that $\sqrt{n}(\frac{1}{n}S_n - \mathbb{E}[X]) \to \mathcal{N}(0, \text{Var}[X])$, hence S_n approaches $\mathcal{N}(n\mathbb{E}[X], n\text{Var}[X])$, however the convergence of S_n here is not well defined as $\mathcal{N}(n\mathbb{E}[X], n\text{Var}[X])$ is not a fixed distribution, but instead it changes with n. Additionally, the distribution of S_n can deviate from Gaussian on a finite set. But the overall shape of the distribution is still roughly Gaussian.

2.1.4 Nonlinear activation functions

Nonlinear activation functions are an integral part of every neural network. We use σ to represent an arbitrary nonlinear activation function. During the forward pass, on each layer, the pixels are first passed through σ and then convolved with the convolution kernel to compute the next layer. This ordering of operations is a little non-standard but equivalent to the more usual ordering of convolving first and passing through nonlinearity, and it makes the analysis slightly easier. The backward pass, in this case, becomes

$$g(i, j, p-1) = {\sigma^p_{i,j}}' \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} w^p_{a,b} g(i+a, i+b, p) \tag{2.12}$$

where we abused notation a bit and use ${\sigma^p_{i,j}}'$ to represent the gradient of the activation function for pixel (i, j) on layer p.

For ReLU nonlinearities, ${\sigma^p_{i,j}}' = \mathbf{I}[x^p_{i,j} > 0]$ where $\mathbf{I}[.]$ is the indicator function. We have to make some extra assumptions about the activations $x^p_{i,j}$ to advance the analysis, in addition to the assumption that it has zero mean and unit variance. A standard assumption is that $x^p_{i,j}$ has a symmetric distribution around 0 [67]. If we make an extra simplifying assumption that the gradients σ' are independent from the weights and g in the upper layers, we can simplify the variance as $\text{Var}[g(i, j, p-1)] = \mathbb{E}[{\sigma^p_{i,j}}'^2] \sum_a \sum_b \text{Var}[w^p_{a,b}]\text{Var}[g(i+a, i+b, p)]$, and $\mathbb{E}[{\sigma^p_{i,j}}'^2] = \text{Var}[{\sigma^p_{i,j}}'] = 1/4$ is a constant factor. Following the variance analysis we can again reduce this case to the uniform weight case.

Sigmoid and Tanh nonlinearities are harder to analyze. Here we only use the observation that when the network is initialized, the weights are usually small and therefore, these nonlinearities will be in the linear region, and the linear analysis applies. However, as the weights grow bigger during training, their effect becomes hard to analyze.

2.1.5 Dropout

Dropout [157] is a technique that sets each unit in a neural network randomly to zero during training, which has found great success as a regularizer to prevent deep networks from over-fitting. Assume the network has a dropout probability of r uniformly across all units. Here we do the variance analysis, Eq. 2.6 becomes

$$g(i,j,p-1) = \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} w_{a,b}^{p} z_{i+a,j+b}^{p} g(i+a,j+b,p) \tag{2.13}$$

where $z_{i+a,i+b}^{p}$ are independent Bernoulli variables with probability $1-r$ to be 1. In this case it is easy to see the expectation of gradient is still 0, and $\mathbb{E}[z_{i+a,j+b}^{p}] = 1-r$, Eq. 2.8 becomes

$$\mathrm{Var}[g(i,j,p-1)] = C(1-r)^2 \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} \mathrm{Var}[g(i+a,j+b,p)]. \tag{2.14}$$

Note the variance now does not factorize into a product of individual variances, as the Bernoulli variables has a non-zero mean. Nevertheless, this case again reduces to the uniform weight case.

2.1.6 Subsampling and dilated convolutions

Subsampling reduces the resolution of the convolutional feature maps and makes each of the following convolutional layers operate on a larger scale. Therefore, it is a great way to increase the receptive field.

Subsampling followed by convolutional layers can be equivalently implemented as changing all the convolutional layers after subsampling from dense convolutions to dilated convolutions[191]. Thus we can apply the same theory we developed above to understand networks with subsampling layers. However, with exponentially growing receptive field introduced by the subsampling or exponentially dilated convolutions, many more layers are needed to see the Gaussian shape clearly.

For models with small depth, it is more practical to understand the effective receptive field as a mixture of Gaussians for networks with subsampling. For example, consider a network with a stack of convolutional layers, followed by a subsampling layer and then another stack of convolutional layers. For the lower stack of convolutional layers before subsampling, each pixel has a Gaussian effective receptive field as the previous analysis showed. For the higher stack of convolutional layers, the effective receptive field for an output unit is Gaussian distributed across the pixels immediately after subsampling, which translates to a mixture of Gaussians, where the mixture weights are themselves Gaussian distributed.

2.1.7 Skip connections

Skip-connections have been found in the human brain. They have also been introduced to deep neural networks and have since become another type of popular architecture designs for deep neural networks in general. Recent state-of-the-art models for image recognition, in particular, the Residual Networks (ResNets) [69] make extensive use of skip connections. The ResNet architecture is composed of residual blocks; each residual block has two pathways, one is a path of q (usually 2) convolutional layers plus nonlinearity and batch-normalization, the other one is a path of a skip connection that goes directly from the input to the output. The output is simply a sum of the results of the two pathways.

In a ResNet of D residual blocks, there are 2^D possible paths to go from input to the output. On a path that

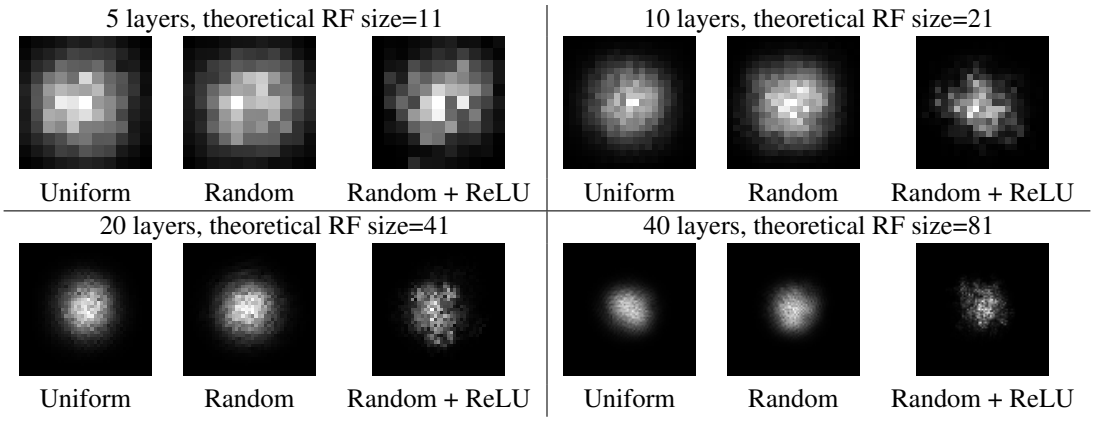


Figure 2.1: Comparing the effect of number of layers, random weight initialization and nonlinear activation on the ERF. Kernel size is fixed at 3×3 for all the networks here. Uniform: convolutional kernel weights are all ones, no nonlinearity; Random: random kernel weights, no nonlinearity; Random + ReLU: random kernel weights, ReLU nonlinearity.

selects d skip connections in all D residual blocks, we get an effective CNN with $(D-d)q$ layers which has an ERF that is Gaussian. The overall ERF is a sum of all these Gaussians, weighted by binomial coefficients $\binom{D}{d}$. We don't have an explicit expression for the ERF size yet, but it is smaller than the biggest receptive field possible, which is achieved when the pathway that goes through the convolutional layers are chosen in all residual blocks. This translates to an ERF size proportional to $\sqrt{Dq}$. On the other hand, the term with the biggest binomial coefficient has approximately $D/2 \cdot q$ convolutional layers and the ERF size is proportional to $\sqrt{Dq/2}$, which is $1/\sqrt{2}$ of the biggest receptive field, and an even smaller fraction of the theoretical receptive field size.

2.2 Experiments

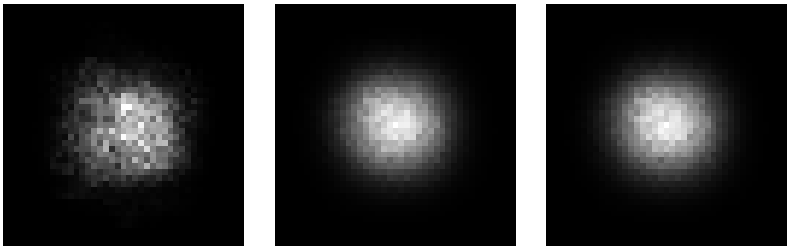
In this section, we empirically study the ERF for various deep CNN architectures. We first use artificially constructed CNN models to verify the theoretical results in our analysis. We then present our observations on how the ERF changes during the training of deep CNNs on real datasets. For all ERF studies, we place a gradient signal of 1 at the center of the output plane and 0 everywhere else. We can than back-propagate this gradient map through the network to get a response map as $\tilde{I}$. Notice the response map will be input dependent, *i.e.*, different input image I will have different response map $\tilde{I}$.

2.2.1 Verifying theoretical results

We first verify our theoretical results in artificially constructed deep CNNs. For computing the ERF, we use random inputs, and for all the random weight networks, we followed [67, 57] for proper random initialization. In this section, we verify the following results:

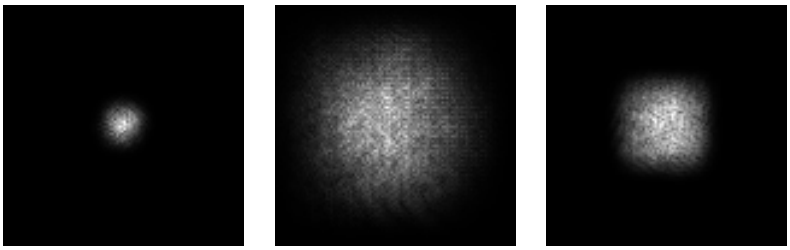
ERFs are Gaussian distributed: As shown in Fig. 2.1, we can observe perfect Gaussian shapes for uniformly and randomly weighted convolution kernels without nonlinear activations, and near-Gaussian shapes for randomly weighted kernels with nonlinearity.

Adding the ReLU nonlinearity makes the distribution a bit less Gaussian, as the ERF distribution depends on the input as well. Another reason is that ReLU units output exactly zero for half of its inputs and it is very easy



ReLU Tanh Sigmoid

Figure 2.2: Effective receptive field visualization for 20 layer networks of different nonlinearities. The result is averaged across 100 runs with random weights as well as random inputs.



Conv-Only Subsample Dilation

Figure 2.3: Effective receptive field visualization for 15 layer networks of different architectures. The result is averaged across 100 runs with random weights as well as random inputs.

to get a zero output for the center pixel on the output plane, which means no path from the receptive field can reach the output; hence the gradient is all zero. Here the ERFs are averaged over 20 runs with a different random seed. Fig. 2.2 shows the ERF for networks with 20 layers of random weights, with different nonlinearities. Here the results are averaged both across 100 runs with different random weights as well as different random inputs. In this setting, the receptive fields are a lot more Gaussian-like.

$\sqrt{n}$ absolute growth and $1/\sqrt{n}$ relative shrinkage: In Fig. 2.4, we show the change of ERF size and the relative ratio of ERF over theoretical receptive field *w.r.t* number of convolution layers. The best fitting line for ERF size gives a slope of 0.56 in the log domain, while the line for ERF ratio gives a slope of -0.43. This indicates ERF size is growing linearly *w.r.t* $\sqrt{N}$ and ERF ratio is shrinking linearly *w.r.t.* $\frac{1}{\sqrt{N}}$. Note here we use 2 standard deviations as our measurement for ERF size, *i.e.* any pixel with a value greater than $(100 - 95.45)\%$ of the center point is considered as in ERF. The ERF size is represented by the square root of the number of pixels within ERF, while the theoretical RF size is the side length of the square in which all pixel has a non-zero impact on the output pixel, no matter how small. All experiments here are averaged over 20 runs.

Subsampling & dilated convolution increases receptive field: In this experiment, we show the effects of different architecture components. In particular, Fig. 2.3 shows the effect of subsampling and dilated convolution. The reference baseline is a convolutional neural network with 15 dense convolution layers only. Its ERF is shown in the left-most figure. We then replace 3 of the 15 convolutional layers with stride-2 convolution to get the ERF for the 'Subsample' figure and replace them with dilated convolution with factor 2,4 and 8 for the 'Dilation' figure. As we can see, both of them are able to increase the effective receptive field significantly. Note the 'Dilation' figure shows a rectangular ERF shape typical for dilated convolutions.

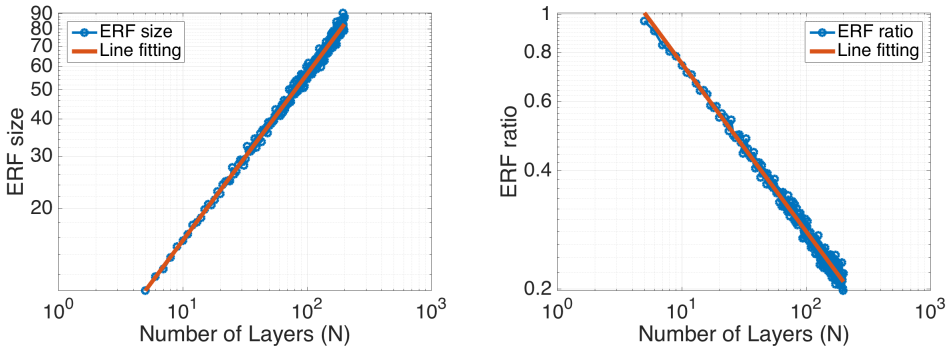


Figure 2.4: Absolute growth (left) and relative shrink (right) for ERF.

2.2.2 How the ERF evolves during training

In this part, we take a look at how the ERF of units in the top-most convolutional layers of a classification CNN and a semantic segmentation CNN evolve during training. For both tasks, we adopt the ResNet architecture which makes extensive use of skip-connections. As the analysis shows, the ERF of this network should be significantly smaller than the theoretical receptive field. This is indeed what we have observed initially. Intriguingly, as the network learns, the ERF gets bigger, and at the end of the training, it is significantly larger than the initial ERF.

Classification - CIFAR10 For the classification task, we trained a ResNet with 17 residual blocks on the CIFAR-10 dataset. At the end of the training, this network reached a test accuracy of 89%. Note that in this experiment, we did not use pooling or downsampling, and exclusively focus on architectures with skip-connections. The accuracy of the network is not state-of-the-art but still quite high. In Fig. 2.5 we show the effective receptive field on the 32×32 image space at the beginning of training (with randomly initialized weights) and at the end of training when it reaches best validation accuracy. Note that the theoretical receptive field of our network is actually 74×74, bigger than the image size, but the ERF is still not able to fully fill the image. Comparing the results before and after training, we see that the effective receptive field has grown significantly.

Semantic Segmentation - CamVid For the semantic segmentation task, we used the CamVid dataset for urban scene segmentation. We trained a "front-end" model [191] which is a purely convolutional network that predicts the output at a slightly lower resolution. This network plays the same role as the VGG network does in many previous works [108]. We trained a ResNet with 16 residual blocks interleaved with 4 subsampling operations, each with a factor of 2. Due to these subsampling operations, the output is $1/16$ of the input size. For this model, the theoretical receptive field of the top convolutional layer units is quite big at 505×505. However, as shown in Fig. 2.5, the ERF only gets a fraction of that with a diameter of 100 pixels at the beginning of training. Again we observe that during training the ERF size increases and at the end it reaches almost a diameter around 150 pixels.

Notice those two tasks are very different in terms of the receptive field. For the classification task, it only outputs a single vector for each input, and it only has one error signal spatially for each image. While for the semantic segmentation task, the output consists of a vector for each pixel in the output image, correspondingly we will have more error signal. Thus the ERF could be aggregated and suffer less from the Gaussian effect of the receptive field.

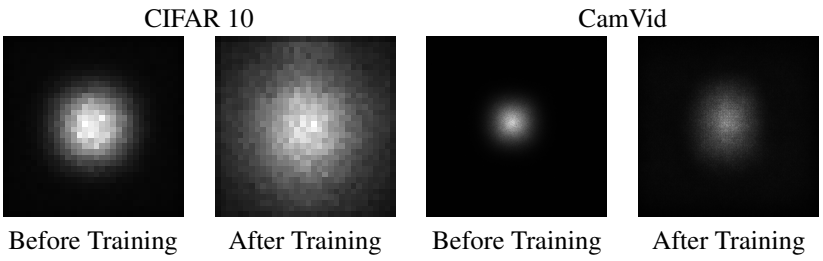


Figure 2.5: Comparison of ERF before and after training for models trained on CIFAR-10 classification and CamVid semantic segmentation tasks. CIFAR-10 receptive fields are visualized in the image space of 32×32.

2.3 Reduce the Gaussian Damage

The above analysis shows that the ERF only takes a small portion of the theoretical receptive field, which is undesirable for tasks that require a large receptive field such as dense prediction problems like semantic image segmentation, stereo, optical flow estimation, etc. Even for classification tasks, having a larger receptive field for the convolution layers may help the model learn better as each pixel in the higher layers of the convolutional network then takes information from a larger area of the input.

New Initialization: One simple way to increase the effective receptive field is to manipulate the initial weights. We propose a new random weight initialization scheme that makes the weights at the center of the convolution kernel have a smaller scale than the weights on the outside; this diffuses the concentration on the center out to the periphery. Practically, we can initialize the network with any initialization method, then scale the weights according to a distribution that has a lower scale at the center and higher scale on the outside.

In the extreme case, we can optimize the $w(m)$'s to maximize the ERF size or equivalently the variance in Eq. 2.10. Solving this optimization problem leads to the solution that put weights equally at the 4 corners of the convolution kernel while leaving everywhere else 0. However, using this solution to do random weight initialization is too aggressive, and leaving a lot of weights to 0 makes learning slow. A softer version of this idea usually works better.

We have trained a CNN for the CIFAR-10 classification task with this initialization method, with several random seeds. In a few cases we get a 30% speed-up of training compared to the more standard initialization [57, 67]. But overall the benefit of this method is not always significant. We note that no matter what we do to change $w(m)$, the effective receptive field is still distributed like a Gaussian so the above proposal only solves the problem partially.

Architectural changes: A potentially better approach is to make architectural changes to the CNNs, which may change the ERF in more fundamental ways. For example, instead of connecting each unit in a CNN to a local rectangular convolution window, we can sparsely connect each unit to a larger area in the lower layer using the same number of connections. Dilated convolution [191] belongs to this category, but we may push even further and use sparse connections that are not grid-like.

An alternative way to increase the ERF given a fixed number of weights is to do weight sharing within kernels. There are different variants of 'weight sharing' previous developed. [62] proposes to prune the network by throwing away connections that have small weights and cluster weights into groups to share the same weights. [24] proposes to use a hashing function to represent the index mapping. thus it would be more memory efficient as it does not need to store the index table for all the locations of every kernel. But it also loses the ability to potentially

learn better sharing scheme within kernels as it is determined by the hashing function. Dilation convolution[191] is also a special case of 'weight sharing' as it has a fixed sharing scheme and the shared weights are all zeros. Other architecture changes are also possible, and the space is open for exploration.

2.4 Discussion

Deep convolutional neural networks have recently been the driving force behind the significant improvement of the state-of-the-art for many computer vision tasks, including image recognition [93, 70], object detection [140, 26], semantic segmentation [108, 5], image captioning [182]. The receptive field is a fundamental notion in CNNs' designs for many vision tasks, as the output must respond to large enough areas in the image to capture information about large objects (that is also the reason why bottom layers can only learn local features, while top layers can learn more global features).

In this chapter, we study the characteristics of receptive fields of units in deep convolutional neural networks. We introduce the notion of an effective receptive field and show that it has a Gaussian distribution and only occupies a fraction of the full theoretical receptive field. We analyze the effective receptive field in several architectural designs and how it is affected by nonlinear activations, dropout, sub-sampling, and skip connections. This analysis leads to suggestions for ways to address its tendency to be too small.

The theory we develop for the effective receptive field also correlates well with some empirical observations. One such empirical observation is that the current commonly used random initialization leads some deep CNNs to start with a small effective receptive field, which then grows during training. This potentially indicates a bad initialization bias.

Connection to biological neural networks: In our analysis, we have established that the effective receptive field in deep CNNs actually grows a lot slower than we used to think. This indicates that a lot of local information is still preserved, even after many convolution layers. This finding contradicts some long-held relevant notions in deep biological networks. A popular characterization of mammalian visual systems involves a split into "what" and "where" pathways [165]. Progressing along the "what" or "where" pathway, there is a gradual shift in the nature of connectivity: receptive field sizes increase, and spatial organization becomes looser until there is no obvious retinotopic organization; the loss of retinotopy means that single neurons respond to objects such as faces anywhere in the visual field [86]. This functional difference between ventral (object) and dorsal (spatial) pathways has been discovered in both monkeys and humans by measuring neural activity while subjects performed object identity and spatial location tasks [65].

A second relevant effect of our analysis is that it suggests that convolutional networks may automatically create a form of foveal representation. The fovea of the human retina extracts high-resolution information from an image only in the neighborhood of the central pixel. Sub-fields of equal resolution are arranged such that their size increases with the distance from the center of the fixation. At the periphery of the retina, lower-resolution information is extracted from larger regions of the image. Some neural networks have explicitly constructed representations of this form [97]. However, because convolutional networks form Gaussian receptive fields, the underlying representations will naturally have this character.

Connection to previous work on CNNs: While receptive fields in CNNs have not been studied extensively, He *et al.*[67], Glorot and Bengio [57] conduct similar analyses, in terms of computing how the variance evolves

through the networks. They developed a good initialization scheme for convolution layers following the principle that variance should not change much when going through the network.

Researchers have also utilized visualizations in order to understand how neural networks work. Mahendran and Vedaldi [118] showed the importance of using natural-image priors as well as what the activation of the convolutional layer would represent. Zeiler and Fergus [196] used deconvolutional nets to show the relation of pixels in the image and the neurons that are firing. Zhou *et al.*[203] did an empirical study involving receptive field and used it as a cue for localization. There are also visualization studies using gradient ascent techniques [48] that generate interesting images [126]. These all focus on the unit activations, or feature map, instead of the effective receptive field, which we investigate here.

Connection to later work on CNNs: Following our study, researchers have also developed various architectures that we found interesting and related to our analysis on effective receptive field. Zhao *et al.* [202] proposed a network that aggregates features from different layers in deep CNNs with proper upsampling. This is equivalent to adding skip connections between different layers on different spatial locations. From our analysis, this can greatly increase the effective receptive field size, which can also explain the effectiveness of their model. Furthermore, deformable convolution [39, 205] was proposed to operate on the learned location instead of the regular grid used by conventional convolution operator. Following our analysis, this can also reduce the Gaussian damage, thus increasing the effective receptive field size as it breaks the symmetry of the standard convolutions across the spatial locations. They have also shown great visualization on how the receptive field changed with their model, which also demonstrates the increase of ERF.

To summarize, in this chapter, we carefully studied the receptive fields in deep CNNs and established a few surprising results about the effective receptive field size. In particular, we have shown that the distribution of impact within the receptive field is asymptotically Gaussian, and the effective receptive field only takes up a fraction of the full theoretical receptive field. Empirical results echoed the theory we established. We believe this is just the start of the study of the effective receptive field, which provides a new angle to understand deep CNNs. In the future, we hope to study further the factors that impact effective receptive field in practice and how we can gain more control over them.

Chapter 3

Efficient Deep Learning for Stereo Matching

In chapter 2, we studied in detail the effective receptive field of convolutional neural networks. From this chapter onwards, we will look at specific problems in the domain of self-driving. One of the most important tasks in the self-driving stack is sensing and capturing information of the surrounding world. In particular, reconstructing the scene in 3D is one of the most important tasks for autonomous driving as we need to know how far we are from different obstacles in order to avoid them. 3D sensors such as LiDAR are commonly employed to ease this process. However, LiDAR sensors are very expensive and each could cost about 80K dollar, a price that is much higher than a standard vehicle. Thus, utilizing cameras is an attractive alternative as it is typically a more cost-effective solution. Despite decades of research, estimating depth from a stereo pair is still an open problem. Dealing with occlusions, large saturated areas, and repetitive patterns are some of the remaining challenges.

Many approaches have been developed that try to aggregate information from local matches. Cost aggregation, for example, averages disparity estimates in a local neighborhood. Similarly, semi-global block matching [74] and Markov random field based methods [156, 184] combine pixelwise predictions and local smoothness into an energy function. However, all these approaches employ cost functions that are hand-crafted or where only a linear combination of features is learned from data.

In the past few years, we have witnessed a revolution in high-level vision where deep representations are learned directly from pixels to solve many scene understanding tasks with unprecedented performance. These approaches are currently state of the art in tasks such as detection, segmentation, and classification.

Very recently, convolutional networks have also been exploited to learn how to match for the task of stereo estimation [195, 193]. Current approaches learn the parameters of the matching network by treating the problem as binary classification; Given a patch in the left image, the task is to predict if a patch in the right image is the correct match. Towards this goal, they generate training examples by randomly sampling patches in the left image and generating for each patch either a positive or a negative example. While [194] showed great performance in challenging benchmarks such as KITTI [54], it is computationally very expensive, requiring a minute of computation in the GPU. This is because they exploited a siamese architecture followed by concatenation and further processing via a few more fully-connected layers to compute the final score, and accordingly, the memory requirements are also high.

In contrast, in this chapter, we propose a matching network which can produce very accurate results in less than a second of GPU computation. Towards this goal, we exploit a product layer which simply computes the

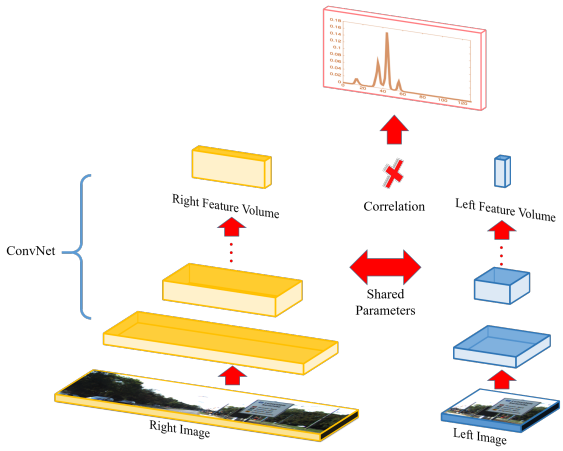


Figure 3.1: To learn informative image patch representations, we employ a siamese network which extracts marginal distributions over all possible disparities for each pixel.

inner product between the two representations of a siamese architecture. We train our network by treating the problem as multi-class classification, where the classes are all possible disparities. This allows us to get calibrated scores, resulting in much better matching performance when compared to [194]. We refer the reader to Fig. 3.1 for an illustration of our approach. We demonstrate the effectiveness of our approach on the challenging KITTI benchmark and show competitive results when exploiting smoothing techniques. For this work, the code and data can be found online at: http://www.cs.toronto.edu/~wenjie/cvpr16.html.

3.1 Background

In order to build a self-driving vehicle, or any robotic system that interacts with the environment, we need sensors to collect information from the surrounding world. Cameras are the most popular sensors as they are cost-effective and robust to different lighting or weather conditions. Various types of cameras have been developed for different use cases; for example, long-range cameras can provide more detail on objects far away, though with a narrow field of view. Fisheye cameras, on the other hand, can see a wider field of view, which is suitable for observing objects very close to the camera. Although cameras can provide rich information about the surrounding environment, the data we get from them (*i.e.* images) is a projection of the 3D world to the 2D image plane; thus, the depth information is lost. As we know, humans (along with many other animals) evolved to have two eyes, providing a stereo vision so they can *compute* depth.

In self-driving, depth is one of the most critical pieces of information we need. As we live in a 3D world, in order to navigate and drive in real-world environments, we need to know the position and status of all surrounding objects (*e.g.* vehicles, pedestrians, and traffic lights/signs) in 3D world. A simple example is when drivers are following the preceding vehicle, they need to estimate how far it is from them, *i.e.* the depth of the vehicle.

Luckily, we live in a structured world. We can use the domain knowledge of geometry for the stereo vision problem. In the remaining of this chapter, we will first introduce the geometry background on stereo vision in Section 3.1.1 that translates the depth estimation problem to the matching problem. Next, in Section 3.1.2, we provide related work on depth estimation using stereo images.

3.1.1 Stereo Geometry

Image is the projection of the 3D world onto a 2D image plane; thus, depth is lost. However, if we use two cameras that provide stereo images, we could reconstruct the depth. This is also how humans estimate the depth of the scene, with human eyes acting as stereo cameras.

Given a rectified image pair (I^L, I^R), the depth of every pixel can be computed via

$$z = \frac{k}{d},$$

where k is the distance between the two cameras multiplied by the distance from the lens to the image, and d is the disparity, which is the horizontal distance in the rectified image pair. Under this framework, the most important thing is to find the corresponding image pixels between the two images. This is a non-trivial task, as (1) finding correspondences between pixels involves modeling/comparing subtle differences, (2) it requires a large amount of computation, since each pixel could have appeared at hundreds of possible locations on the right image(large space of d) and we could easily have hundreds of thousands of pixels in a single image.

3.1.2 Related Work

Over the past decades, many stereo algorithms have been developed. In this section, we restrict ourselves mostly to a subset of methods that exploit learning, where we can formulate the problem as an energy minimization task.

Intensity-based methods typically employ a combination of building blocks to perform matching cost computation, cost aggregation, disparity computation, and disparity refinement, as shown in the Middlebury leaderboard [147].

Among the most successful methods in the KITTI benchmark [54, 124] is the work by Vogel *et al.* [168]. The authors introduce a collection of planar and rigidly moving local segments. They design a sophisticated energy function which includes terms for occlusion, shape, motion, and segmentation.

Early learning based approaches focused on correcting an initially computed matching cost [91, 92]. Learning has also been utilized to tune the hyper-parameters of the energy-minimization task. Among the first to train these hyper-parameters were [199, 146, 101], which investigated different forms of probabilistic graphical models, ranging from Markov random fields and its conditional counterpart to structured support vector machines.

Slanted plane approaches model groups of pixels with slanted 3D planes. They are very competitive in autonomous driving scenarios, where robustness is the key. They have a long history, dating back to [12] and were shown to be very successful on the Middleburry benchmark [147, 90, 16, 172] as well as on KITTI [184, 185, 186].

Holistic models which solve jointly many tasks have also been explored. The advantage is that many tasks in low-level and high-level vision are related, and one can benefit from solving them together. For example, a combination of approaches which jointly solve for object segmentation and depth have been developed [14, 15, 13, 96, 60]. Guney and Geiger [59] investigated the utility of high-level vision tasks such as object recognition and semantic segmentation for stereo matching.

Estimating the confidence of each match is key when employing stereo estimates as a part of a pipeline. Learning methods were successfully applied to this task, *e.g.*, by combining several confidence measures via a random forest classifier [61], or by incorporating random forest predictions into a Markov random field [156].

Convolutional neural networks(CNN) have been shown to perform very well on high-level vision tasks such as image classification, object detection, and semantic segmentation. More recently, CNNs have been applied to low-level vision tasks such as optical flow prediction [42]. In the context of stereo estimation, [194] utilize

CNN to compute the matching cost between two image patches. In particular, they used a siamese network which takes the same sized left and right image patches with a few fully-connected layers on top to predict the matching cost. They trained the model to minimize a binary cross-entropy loss. In similar spirit to [194], [193] investigated different CNN based architectures for comparing image patches. They found concatenating left and right image patches as different channels works best but at the cost of being very slow.

Our work here is most similar to [194, 193] with two main differences. First, we propose to learn a probability distribution over all disparity values using a smooth target distribution. Our model learns to refine the entire cost volume which results in a much smoother prediction across all possible disparities. As a consequence, we are able to capture correlations between the different disparities implicitly. This contrasts to [194] which performs independent binary predictions on image patches where extracted information is more local and correlations between different disparities are harder to capture. Second, on top of the convolution layers, we use a simple dot-product layer to join the two branches of the network instead of a few fully-connected layers. This allows us to do orders of magnitude faster computation. We note that the concurrent work unpublished at the time of submission of our paper [195, 29] also introduced a dot-product layer.

We will describe the details of our architecture in the following Section 3.2, as well as different smoothing techniques that can further enhance the performance in Section 3.3. We conduct a detailed evaluation of the design choices in Section 3.4.

3.2 Deep Learning for Stereo Matching

We are interested in computing a disparity image given a stereo image pair. Throughout this chapter, we assume that the image pairs are rectified; thus the epipolar lines are aligned with the horizontal image axis, and we follow the large body of existing literature by phrasing this disparity map estimation task as an energy minimization problem.

Let $y_i \in \mathbf{Y}_i$ represent the disparity associated with the i-th pixel, and let $|\mathbf{Y}_i|$ be the cardinality of the set (typically 128 or 256). Stereo algorithms estimate a 3-dimensional cost volume by computing for each pixel in the left image a score for each possible disparity value. This is typically done by exploiting a small patch around the given pixel and a simple hand-crafted representation of each patch. In contrast, in this work, we exploit convolutional neural networks to learn how to match.

In detail, the score for each pixel is typically computed by exploiting a patch around the given pixel. For each pixel i in the left image, we can compute the cost volume by computing the scores for the different disparity values. The score is obtained by matching a patch defined on the neighborhood of the i-th pixel with all $|\mathbf{Y}_i|$ locations along the epipolar line of the corresponding rectified right image. The employed neighborhood is typically small, and simple hand-crafted feature representations are exploited. Recently [194] proposed to use a convolutional neural net for stereo estimation, the idea being to learn how to match two image patches.

Convolution neural networks (CNNs) are very powerful in learning representations for high-level vision tasks, such as object recognition. In this chapter, we design a CNN architecture, which is suitable for the low-level vision problem of stereo estimation.

We utilize a siamese architecture, where each branch processes the left or the right image respectively. We refer the reader to Fig. 3.2 for an illustration. In particular, each branch takes an image patch as input, and passes it through a set of layers, each consisting of a spatial convolution with a small filter-size (*e.g.*, 5×5 or 3×3), followed by spatial batch normalization and a rectified linear unit (ReLU). Note that we remove the ReLU from the last layer in order to not lose the information encoded in the negative values. In our experiments, we exploit a

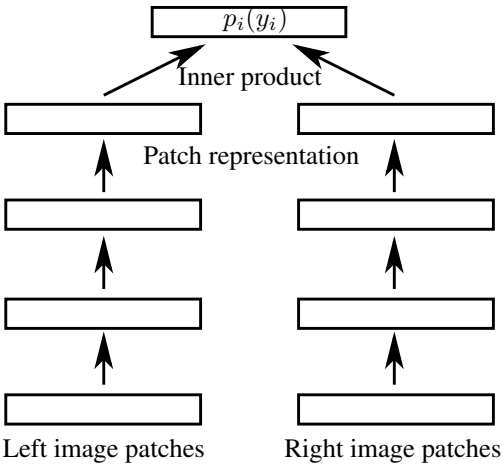


Figure 3.2: Our four-layer siamese network architecture which has a receptive field size of 9.

different number of filters per layer, either 32 or 64, and share the parameters between the two branches.

In contrast to existing approaches which exploit concatenation followed by further processing, we use a product layer which simply computes the inner product between the two representations to compute the matching score. This simple operation speeds up the computation significantly as the resulting representation is much lower dimensional. Fig. 3.2 depicts an example of a 4-layer network with filter-size 3×3, which results in a receptive field of size 9×9.

Our network can learn a good representation of an image pixel using its surrounding neighborhood. At the same time, we introduce a dot-product operation to compute the similarity metric efficiently.

3.2.1 Training

We use small left image patches extracted at random from the set of pixels for which ground truth is available to train the network. This strategy provides us with a diverse set of examples and is memory efficient. In particular, each left image patch is of a size equivalent to the size of our network's receptive field. We use a larger patch for the right image which expands both the size of the receptive field as well as all possible disparities (*i.e.*, displacements). The output of the two branches of the siamese network is hence a single 64-dimensional representation for the left branch, and $|\mathbf{Y}_i| \times 64$ for the right branch. These two vectors are then passed as input to an inner-product layer which computes a score for each of the $|\mathbf{Y}_i|$ disparities. This allows us to compute a softmax for each pixel over all possible disparities.

During training, we minimize cross-entropy loss with respect to the weights w that parameterize the network

$$\min_{w} \sum_{i,y_i} p_{\mathrm{gt}}(y_i) \log p_i(y_i, w).$$

Since we are interested in a 3-pixel error metric we use a smooth target distribution $p_{\mathrm{gt}}(y_i)$, centered around the ground-truth y_i^{GT}, *i.e.*,

$$p_{\mathrm{gt}}(y_i) = \begin{cases} \lambda_1 & \text{if } y_i = y_i^{GT} \\ \lambda_2 & \text{if } |y_i - y_i^{GT}| = 1 \\ \lambda_3 & \text{if } |y_i - y_i^{GT}| = 2 \\ 0 & \text{otherwise} \end{cases}.$$

	>2 pixel		>3 pixel		>4 pixel		>5 pixel		End-Point		Runtime(s)
	Non-Occ	All	Non-Occ	All	Non-Occ	All	Non-Occ	All	Non-Occ	All	
MC-CNN-acrt [194]	15.02	16.92	12.99	14.93	12.04	13.98	11.38	13.32	4.39 px	5.21 px	20.13
MC-CNN-fast [194]	17.72	19.56	15.53	17.41	14.41	16.31	13.60	15.51	4.77 px	5.63 px	0.20
Ours(19)	**10.87**	**12.86**	**8.61**	**10.64**	**7.62**	**9.65**	**7.00**	**9.03**	**3.31** px	**4.2** px	**0.14**

Table 3.1: Comparison of the output of the matching network across different error metrics on the KITTI 2012 validation set.

Unary	CA	SGM[195]	Post[195]	Slanted[186]	Ours(9)	Ours(19)	Ours(29)	Ours(37)	MC-CNN-acrt[194]	MC-CNN-fast[194]
✓					16.69	8.61	7.64	6.61	12.99	15.53
✓	✓				12.14	7.48	6.86	6.09	6.32	-
✓	✓	✓			4.57	3.99	4.12	3.96	3.34	4.53
✓	✓	✓	✓		4.11	3.73	3.99	3.88	3.22	3.73
✓	✓	✓	✓	✓	3.96	3.64	3.81	3.83	3.36	3.83

Table 3.2: Comparison of different smoothing methods. The table illustrates non-occluded 3 pixel error on the KITTI 2012 validation set.

In this work, we set $\lambda_1 = 0.5$, $\lambda_2 = 0.2$ and $\lambda_3 = 0.05$. Note that this contrasts with cross entropy for classification, where $p_{gt}(y_i)$ is a delta function placing all its mass on the annotated groundtruth configuration.

We train our network using backpropagation with stochastic gradient descent AdaGrad [45]. Similar to moment-based stochastic gradient descent, AdaGrad adapts the gradient based on historical information. Contrasting moment based methods, it emphasizes rare but informative features. We adapt the learning rate every few thousand iterations as detailed in the experimental section.

3.2.2 Inference

In contrast to the training procedure where we compose a mini-batch by randomly sampling locations from different training images, we can improve the performance during testing. Our siamese network computes a 64-dimensional feature representation for every pixel i. To efficiently obtain the cost volume, we compute the 64-dimensional representation only once for every pixel i, and during computation of the cost volume, we re-use its values for all disparities that involve this location.

3.3 Smoothing Deep Net Outputs

Given the unaries obtained with a CNN, we compute predictions for all disparities as well as its probability at each image location. Note that simply outputting the most likely configuration for every pixel is not competitive with modern stereo algorithms, which exploit different forms of cost aggregation, post-processing, and smoothing. This is particularly important to deal with complex regions with occlusions, saturation, or repetitive patterns. To be more specific, even though our proposed CNN architecture uses more global information than networks trained for binary prediction, we have not yet considered the correlations between the outputs at nearby pixels.

Over the past decade, many different MRFs have been proposed to solve the stereo estimation problem. Most approaches define each random variable to be the disparity of a pixel and encode smoothness between consecutive or nearby pixels. An alternative approach is to segment the image into regions and estimate a slanted 3D plane

for each region. In this chapter, we investigate the effect of different smoothing techniques that leverage the cost volume returned by our matching network. Towards this goal, we formulate the stereo matching as inference in several different Markov random fields (MRFs) in order to smooth the matching results produced by our convolutional neural network. In particular, we investigate cost aggregation, semi-global block matching as well as the slanted plane approach of [186] as means of smoothing. In the following, we briefly review these techniques:

Cost aggregation: We exploited a very simple cost aggregation approach, which simply performs average pooling over a window of size 5×5. This is a local smoothing technique, which refines the unaries by averaging the marginal probabilities $p_i(y_i)$ of neighboring pixels.

Semi global block matching: Semi-global block matching augments the unary energy term obtained from convolutional neural nets by introducing additional pairwise potentials which encourage smooth disparities. In this chapter we assume the energy to be composed of unary and pairwise terms, specifically,

$$E(y) = \sum_{i=1}^{N} E_i(y_i) + \sum_{(i,j)\in\mathcal{E}} E_{i,j}(y_i, y_j),$$

where $\mathcal{E}$ refers to 4-connected grid and the unary energy $E_i(y_i)$ is the output of the neural net. We use the standard four-connected neighborhood system.

We define the pairwise energy as

$$E_{i,j}(y_i, y_j) = \begin{cases} 0 & \text{if } y_i = y_j \\ c_1 & \text{if } |y_i - y_j| = 1 \\ c_2 & \text{otherwise} \end{cases},$$

with variable constants $c_1 < c_2$. We follow the approach of [194], where c_1 and c_2 is decreased if there is strong evidence for edges at the corresponding locations in either the left or the right image. We refer the reader to their paper for more details.

Slanted plane: To construct a depth-map, this approach performs block-coordinate descent on energy involving appearance, location, disparity, smoothness, and boundary energies. More specifically, we first over-segment the image using an extension of the SLIC energy [2]. For each superpixel, we then compute slanted plane estimates [186] which should adhere to the depth-map evidence obtained from the convolutional neural network. We then iterate these two steps to minimize the energy function. We refer the interested reader to [186] for details.

Sophisticated post-processing: In [195], a three-step post-processing is designed to perform interpolation, subpixel enhancement, and refinement. The interpolation step resolves conflicts between the disparity maps computed for the left and right images by performing a left-right consistency check. Subpixel enhancement fits a quadratic function to neighboring points to obtain an enhanced depth-map. To smooth the disparity map without blurring the edges, the final refinement step applies a median filter and a bilateral filter. We only use the interpolation step as we found that the other two do not always further improve the performance in our case.

	> 2 pixel		> 3 pixel		> 4 pixel		> 5 pixel		End-Point		Runtime(s)
	Non-Occ	All	Non-Occ	All	Non-Occ	All	Non-Occ	All	Non-Occ	All	
MC-CNN-acrt [194]	15.20	16.83	12.45	14.12	11.04	12.72	10.13	11.80	4.01 px	4.66 px	22.76
MC-CNN-fast [194]	18.47	20.04	14.96	16.59	13.18	14.83	12.02	13.67	4.27 px	4.93 px	0.21
Ours(37)	**9.96**	**11.67**	**7.23**	**8.97**	**5.89**	**7.62**	**5.04**	**6.78**	**1.84** px	**2.56** px	0.34

Table 3.3: Comparison of the output of the matching network across different error metrics on the KITTI 2015 validation set.

Unary	CA	SGM[195]	Post[195]	Slanted[186]	Ours(9)	Ours(19)	Ours(29)	Ours(37)	MC-CNN-acrt[194]	MC-CNN-fast[194]
✓					15.25	8.95	7.23	7.13	12.45	14.96
✓	✓				11.43	8.00	6.60	6.58	7.78	-
✓	✓	✓			5.18	4.74	4.62	4.73	3.48	5.05
✓	✓	✓	✓		4.41	4.23	4.31	4.38	3.10	4.74
✓	✓	✓	✓	✓	4.25	4.20	4.14	4.19	3.11	4.79

Table 3.4: Comparison of smoothing methods using different CNN output. The table illustrates the non-occluded 3 pixel error on the KITTI 2015 validation set.

3.4 Experimental Evaluation

We evaluate the performance of different convolutional neural network structures and different smoothing techniques on the KITTI 2012 [54] and 2015 [124] datasets.

Before training, we normalize each image to have zero mean and standard deviation of one. We create our training image patches by randomly selecting image pixels with available ground truth disparity as not all pixels in the image have ground truth depth associated with them. Depending on the architecture of the network, we then crop the pixels' surrounding image patch of appropriate dimension $w \times h$. As input for the other branch of the siamese network, we extract a wider image region containing patches for all possible disparities $|\mathbf{Y}_i|$. Therefore the resulting dimension is $(w + |\mathbf{Y}_i|) \times h$.

We initialize the parameters of our networks using a uniform distribution. We employ the AdaGrad algorithm [45] and use a learning rate of $1e^{-2}$. The learning rate is decreased by a factor of 5 after 24k iterations and then further decreased by a factor of 5 every 8k iterations. We use a batch size of 128. The network is trained for 40k iterations which take around 6.5 hours on an NVIDIA Titan-X.

3.4.1 KITTI 2012 Results

The KITTI 2012 dataset contains 194 training and 195 test images. To compare the different network architectures described below, we use as a training set 160 image pairs randomly selected, and the remaining 34 image pairs as our validation set.

Comparison of Matching Networks: We first show our network's matching ability and compare it to existing matching networks [194, 195]. In this experiment, we do not employ smoothing or post-processing, but just utilize the raw output of the network. Following KITTI, we employ the percentage of pixels with disparity errors larger than a fixed threshold as well as an end-point error as metrics. We refer to our architecture as 'Ours(19).'

	> 2 pixels		> 3 pixels		> 4 pixels		> 5 pixels		End-Point		Runtime
	Non-Occ	**All**	**Non-Occ**	**All**	**Non-Occ**	**All**	**Non-Occ**	**All**	**Non-Occ**	**All**	(s)
StereoSLIC [185]	5.76	7.20	3.92	5.11	3.04	4.04	2.49	3.33	0.9 px	1.0 px	2.3
PCBP-SS [185]	5.19	6.75	3.40	4.72	2.62	3.75	2.18	3.15	0.8 px	1.0 px	300
SPS-st [186]	4.98	6.28	3.39	4.41	2.72	3.52	2.33	3.00	0.9 px	1.0 px	2
Deep Embed [29]	5.05	6.47	3.10	4.24	2.32	3.25	1.92	2.68	0.9 px	1.1 px	3
MC-CNN-acrt [195]	3.90	5.45	2.43	3.63	**1.90**	2.85	**1.64**	2.39	**0.7 px**	0.9 px	67
Displets v2 [59]	**3.43**	**4.46**	**2.37**	**3.09**	1.97	**2.52**	1.72	**2.17**	**0.7 px**	**0.8 px**	265
Ours(19)	4.98	6.51	3.07	4.29	2.39	3.36	2.03	2.82	0.8 px	1.0 px	**0.7**

Table 3.5: Comparison to stereo state-of-the-art (as of 2016) on the test set of the KITTI 2012 benchmark.

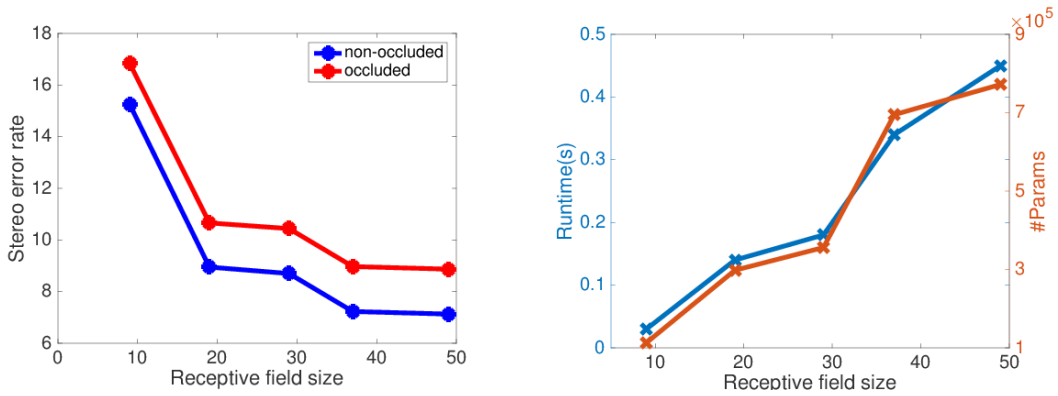
	All/All			All/Est			Noc/All			Noc/Est			Runtime
	D1-bg	D1-fg	D1-all	D1-bg	D1-fg	D1-all	D1-bg	D1-fg	D1-all	D1-bg	D1-fg	D1-all	(s)
MBM [47]	4.69	13.05	6.08	4.69	13.05	6.08	4.33	12.12	5.61	4.33	12.12	5.61	**0.13**
SPS-St [186]	3.84	12.67	5.31	3.84	12.67	5.31	3.50	11.61	4.84	3.50	11.61	4.84	2
MC-CNN [195]	**2.89**	8.88	3.89	**2.89**	8.88	3.88	**2.48**	7.64	3.33	**2.48**	7.64	3.33	67
Displets v2 [59]	3.00	**5.56**	**3.43**	3.00	**5.56**	**3.43**	2.73	**4.95**	**3.09**	2.73	**4.95**	**3.09**	265
Ours(37)	3.73	8.58	4.54	3.73	8.58	4.54	3.32	7.44	4.00	3.32	7.44	4.00	1

Table 3.6: Comparison to stereo state-of-the-art (as of 2016) on the test set of KITTI 2015 benchmark.

It consists of 9 layers of 3×3 convolutions resulting in a receptive field size of 19×19 pixels. As shown in Table 3.1, our 9-layer network achieves a 3-pixel non-occluded stereo error of 8.61% after only 0.14 seconds of computation. In contrast, [194] obtains 12.99% after a significantly longer time of 20.13 seconds. Their faster version [195] requires 0.20 second, which results in a much lower performance of 15.53%. As shown in the table, our network outperforms previously designed convolutional neural networks by a large margin on all criteria.

Smoothing Comparison: Next, we evaluate different algorithms for smoothing and post-processing when employing different network sizes. In particular, we evaluate cost aggregation, semi-global block matching, and slanted plane smoothing which were described in the previous section. We also experiment with different receptive field sizes for our network which corresponds to changing the depth of our architecture. As before, we use 'Ours(n)' to refer to our architecture with a receptive field size of $n \times n$ pixel. We investigated $n = 9, 19, 29, 37$. We use kernels of size 3×3 for $n = 9$ and $n = 19$, while the kernels were of size 5×5 for $n = 39$. To achieve a receptive field of 29, we use five layers of 5×5 and four layers of 3×3. This keeps the number of layers bounded to 9.

As shown in Table 3.2, networks with different receptive field sizes result in errors ranging from 6.61% (for $n = 37$ to 16.69% for $n = 9$. The corresponding error for [195] is 12.99% for their slow and more accurate model, and 15.53% for their fast model. After smoothing, the differences in stereo error achieved by the networks are no longer significant. All of them achieve an error of slightly less than 4%. Since depth-maps tend to be very smooth, we think that an aggressive smoothing helps to flatten the noisy unary potentials. Also, we observe that utilizing simple cost aggregation to encourage local smoothness further helps to improve the results slightly. This is because such techniques eliminate small isolated noisy areas. From column 3, we can see adding some local smoothness does help improve the results as it can eliminate small isolated noise due to image imperfection, etc. While the post-processing proposed in [195] focuses on occlusions and sub-pixel enhancement, [186] adds extra robustness to non-textured areas by fitting slanted planes to model depth discontinuities. Both methods improve the semi-global block matching output slightly. Our best performing model combination achieves a 3-pixel stereo



(a) Stereo error using only the matching network (unaries).

(b) Runtime and number of parameters over the receptive field size.

Figure 3.3: Evaluation of stereo error (a), runtime and number of parameters (b).

error of 3.64%.

We note that with our naive implementation of the algorithm, using the larger models, we cannot fit the entire image into GPU memory at once during test time. For consistency, we drop the top 100 rows of all the images since there is no ground-truth for the top of the image in KITTI. This operation results in a 0.01% difference in computing stereo error.

Comparison to state of the art[1]: To evaluate the test set performance we trained our model having a receptive field of 19 pixels, *i.e.*, "Ours(19)," on the entire training set. The obtained test set performance is shown in Table 3.5. Since we did not particularly focus on finding a good combination of smoothness and unaries, our performance is slightly below the current state-of-the-art.

Qualitative Analysis: Examples of stereo estimates by our approach are depicted in Fig. 3.4. Each row represents one example, and the first column represents the left input image out of the stereo image pair, the center column shows the final depth estimation in terms of disparities and the rightmost column give the stereo error image in terms of the error in disparities, higher value (brighter) means bigger error. We can see that our algorithm gives a smooth depth image, while also able to capture the shapes of trees and vehicle etc. Looking at the error image, we found that our approach suffers from texture-less regions such as the window of the vehicle, repetitive patterns such as fences, etc.

3.4.2 KITTI 2015 Results

The KITTI 2015 dataset is an augmented dataset over KITTI 2012. It consists of 200 training and 200 test images. Instead of the gray-scale images used for the KITTI 2012 dataset, it provides RGB stereo images. To compare the different network architectures, we randomly selected 160 image pairs as the training set and use the remaining 40 image pairs for validation purposes.

Comparison of Matching Networks: We first show our network's matching ability and compare it to existing matching networks [194, 195]. In this experiment, we do not employ smoothing or post-processing, but just

[1]We are comparing to the 'state of the art' at the time of publication of the corresponding paper [112], *i.e.* year of 2016

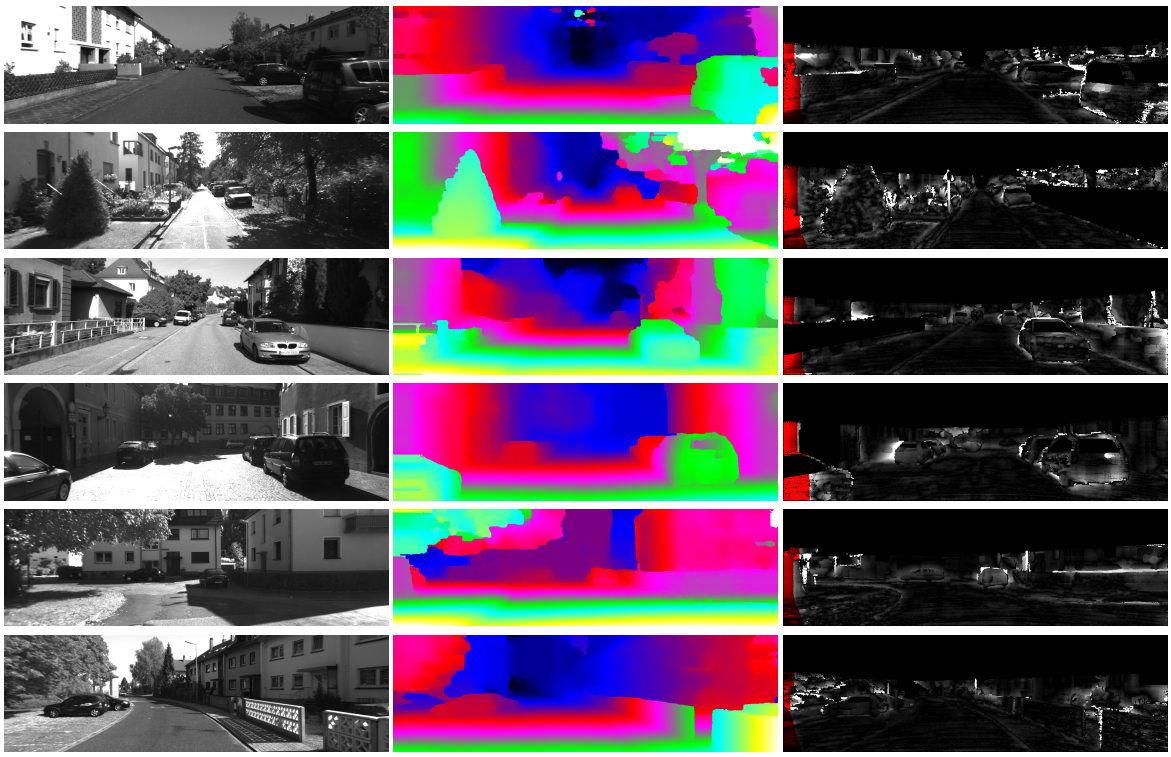


Figure 3.4: KITTI 2012 test set: (left) original image, (center) stereo estimates, (right) stereo errors.

utilize the raw output of the network. We refer to our architecture via 'Ours(37).' It consists of 9 layers of 5×5 convolutions resulting in a receptive field size of 37×37 pixels. As shown in Table 3.3, we compare the stereo prediction error using a different pixel range and provide results for both the occluded and non-occluded version. Our 9-layer network achieves a 3-pixel stereo error of 7.23% after only 0.34 seconds of processing time, whereas [194] obtains 12.45% after a significantly longer processing time of 22.76 seconds. Their faster version [195] requires 0.21 seconds but results in a much lower performance of 14.96% when compared to our approach. Again, our network outperforms previously designed convolutional neural networks by a large margin on all criteria.

Smoothing Comparison: Table 3.4 shows results of applying different post processing techniques to different network architectures. Similar to KITTI 2012 dataset, we observe that the difference in network performance vanishes after applying smoothing techniques. Our best performing combination achieves a 3-pixel error of 4.14% on the validation set.

Influence of Depth and Filter Size: Next, we evaluate the influence of the depth and receptive field size of our CNNs in terms of matching performance and running time. Fig. 3.3a shows matching performance as a function of the networks' receptive field sizes. Notice that the depth of the networks depends linearly on the theoretical receptive field size as we are using a fixed kernel size of 3×3 for each layer. We observe that an increasing receptive field size achieves better performance. However, when the receptive field is very large, the improvement is subtle, since the network starts to overlook the details of small objects and local regions with occlusions and depth discontinuities. Our findings are consistent for both non-occluded and all pixels. As shown in Fig. 3.3b, the running time and number of parameters are highly correlated. Note that models with larger receptive field do not

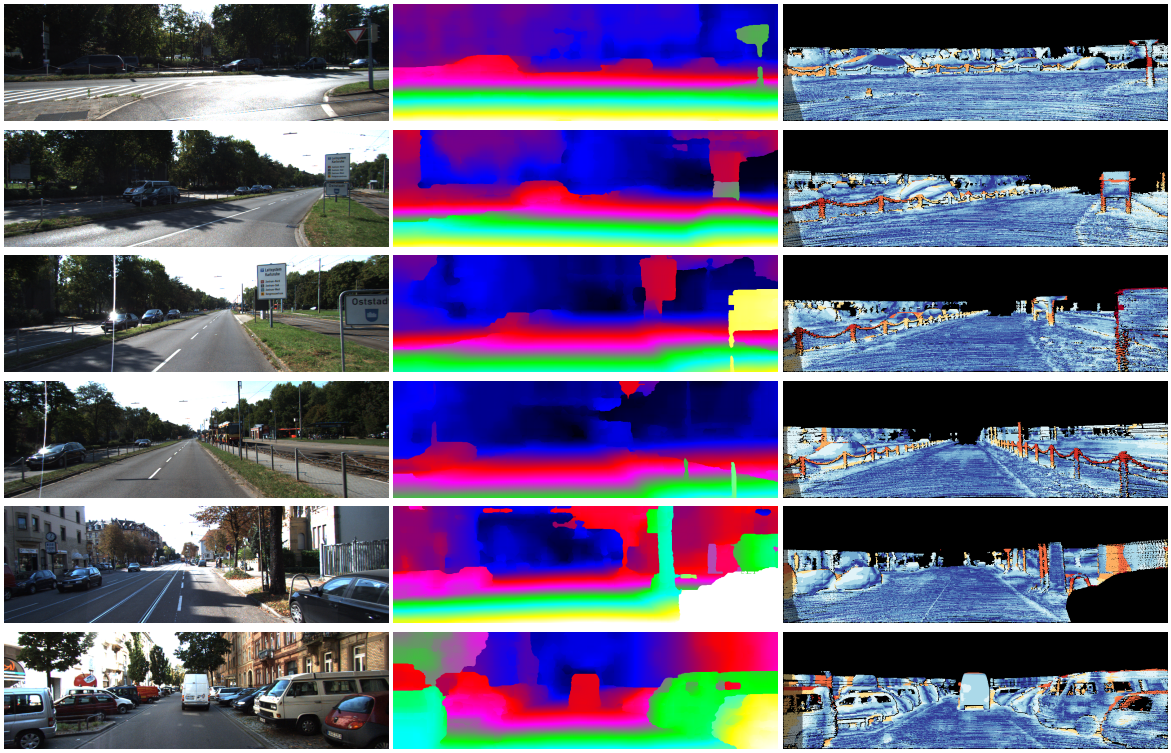


Figure 3.5: KITTI 2015 test set: (left) original image, (center) stereo estimates, (right) stereo errors.

necessarily have more parameters since the number of trainable weights also depends on the number of filters and the channel size of each convolutional layer.

Comparison to state of the art[2]: To evaluate the test set performance, we choose the best model with current smoothing techniques which has a receptive field of 37 pixels, *i.e.*, "Ours(37)." The obtained test set performance is shown in Table 3.6. We achieve on-par results with state of the art in significantly less time.

Qualitative Results: We provide results from the test set in Fig. 3.5. Similar to Fig. 3.4, each row gives one example, and the first column shows the sample left input image (notice it is an RGB image comparing to grayscale image in Fig. 3.4.1), the second column shows the predicted disparity image, while the third column shows the error image (red color tone indicates big error). Again, we observe that our algorithm can produce smooth and consistent disparity images, but it suffers from texture-less regions as well as regions with repetitive patterns such as the sky.

3.5 Discussion

In this chapter, we looked at the depth estimation problem using stereo cameras. Getting depth information (*i.e.* 3D information) about the scene is crucial for self-driving, as driving is performed in 3D world. In contrast to the traditional approaches that use hand-crafted features, convolutional neural networks have been recently shown to perform extremely well for stereo estimation. However, the previous architectures rely on siamese networks

[2]We are comparing to the 'state of the art' at the time of publication of the corresponding paper [112], *i.e.* year of 2016

which exploit concatenation followed by further processing layers, requiring a minute on the GPU to process a stereo pair. In this chapter, we have proposed a matching network which can produce very accurate results in less than a second of GPU computation. Our key contribution is to replace the concatenation layer and subsequent processing layers by a single product layer that computes the matching score. We trained the networks using cross-entropy over all possible disparities. This allows us to get calibrated scores because the network can exploit more contextual information which results in much better matching performance when compared to existing approaches. We have also investigated the effect of different smoothing techniques to improve the performance further.

Despite the big improvement in accuracy, there are also limitations to our approach. Firstly, our approach is still based on matching with image patches; this has the limitation that the learned feature can only capture local information. As a result, the matching purely based on these features would fail in a region that requires global context for discrimination. Secondly, the running time of our approach, although improved by two orders of magnitude, is still not good enough for online usage, especially with limited GPU resources.

Since the development of our approach, the field of stereo depth estimation has advanced a lot. First, DispNet [120] was proposed as an end-to-end framework for depth estimation that directly regress depth from input image pairs. They achieve real-time performance, *i.e.* 60 ms per frame, since there is no explicit matching for all pixels. Further, Kendall *et al.* [88] incorporated the epi-polar matching into the network by exhaustively concatenating features from all possible disparities locations. Correspondingly, their intermediate representation was a 4D feature volume. In addition, they proposed a differentiable soft argmin operation to enable sub-pixel disparity regression directly from the 4D feature volume. Later, PSM-Net [20] further improved the performance by extending the idea to include stacked hourglass network [130] and spatial pyramid pooling [68]. All these works rely on the neural networks to have large receptive fields to capture global information, either by designing deeper models or utilizing spatial pyramid pooling operation. On the other hand, Liu *et al.* [106] proposed spatial propagation network that embedded a feature propagation procedure in the network. This can generate more global image features and was shown to work well for semantic segmentation. Later, Cheng *et al.* [31, 32] extended this idea to feature propagation in 3D in the context of stereo depth estimation. Their experiments showed that matching using those aggregated features greatly improved the performance of stereo depth estimation in different datasets and achieved state of the art results on the KITTI dataset [53].

Chapter 4

Deep Matching for Optical Flow

In chapter 3, we developed a new depth estimation algorithm exploiting deep neural networks. Stereo depth estimation does provide richer information, *i.e.*, 3D information about the surrounding environment allowing the self-driving vehicle to operate in the 3D real world. However, it is still reasoning for one timestep. For self-driving, it is essential to look at temporal information, to understand how the environment is changing over time. Optical flow is one of the crucial tasks falling in this domain and has been widely studied for years. It operates on image pairs from adjacent time frames and is used to estimate the motion of all pixels in the image. Fig. 4.1 shows an example of the desired output of optical flow, where each pixel has an associated vector indicating their motion into the next frame.

Different than stereo depth estimation, where a stereo image pair is used, in this chapter, we focus on the optical flow problem using a monocular camera. This is the fundamental problem of flow estimation, comparing to using stereo cameras to estimate depth and flow together (often referred to as scene flow). Despite many decades of research, estimating dense optical flow is still an open problem. Large displacements, textureless regions, specularities, shadows, and big changes in illumination continue to pose difficulties. Furthermore, flow estimation is computationally very demanding, as the typical range for a pixel's potential motion can contain more than 30K possibilities. This poses many problems for discrete methods. Therefore most recent methods rely on continuous optimization [141, 159]. Similar to depth estimation, where we can formulate the problem as image patch matching along the epipolar line, optical flow can be formalized as an image patch matching problem in 2D search space. Thus, it is straightforward to utilize deep neural networks for optical flow as well.

In this chapter, we are interested in computing optical flow in the context of autonomous driving. We argue that strong priors can be exploited in this context to make the estimation more robust (and potentially faster). In particular, we build on the observation that the scene is typically composed of a static background, as well as a relatively small number of traffic participants which move rigidly in 3D. To exploit such intuition, we need to reliably identify the independently moving objects and estimate their motion. Past methods typically attempt to segment the objects based solely on the motion. However, this is a chicken and egg problem: an accurate motion estimation is necessary for accurate motion segmentation, yet the latter also circularly depends upon the former.

In contrast, we propose an alternative approach, which relies solely on exploiting semantics to identify the potentially moving objects. Note that semantic segmentation is not sufficient as different vehicles might move very differently, yet form a single connected component due to occlusion. Instead, we exploit instance-level segmentation, which provides us with a different segmentation label for each vehicle. Given the instance segmentation, our approach then formulates the optical flow problem as a set of epipolar flow estimation problems, one for each

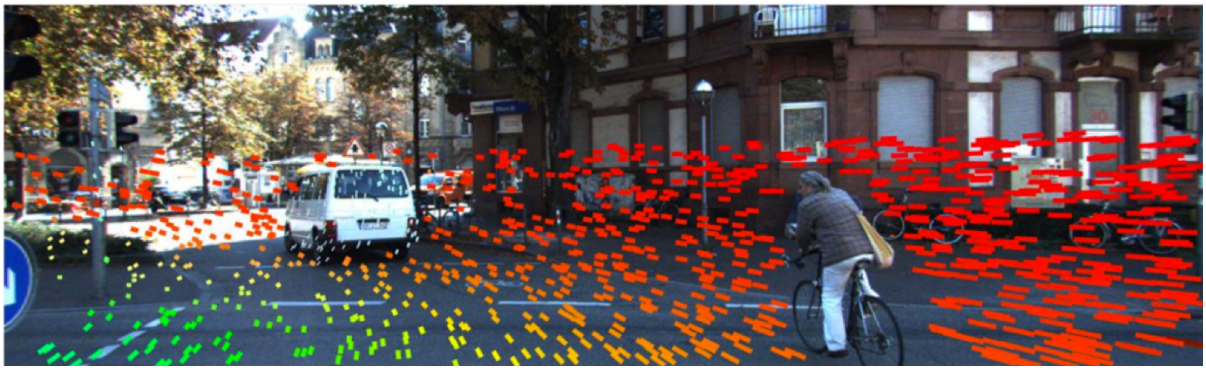


Figure 4.1: Optical flow example.

moving object. We treat the background as a special object, whose motion is solely due to the ego-car's motion. This contrasts [183, 95], as we estimate a different epipolar geometry (*i.e.*, fundamental matrix) for each moving object instead of assuming that the whole scene is static and the ego-car is the only thing moving. As shown in our experimental evaluation, this results in much better flow estimates for moving objects. Since we formulate the problem as a set of epipolar flow problems, the search space is reduced from a 2D area to a 1D search along the epipolar line. This has benefits both in terms of the computational complexity, as well as the robustness of our proposed approach. We refer the reader to Fig. 4.2 for an illustration of our approach.

The success of our approach relies on accurate fundamental matrix estimation for each moving object, as well as accurate matching. To facilitate this, our second contribution is a new convolutional net that learns to perform flow matching and can estimate the uncertainty of its matches. This allows us to reject outliers, leading to better estimates for the fundamental matrix of each moving object. We smooth our predictions using semi-global block matching [73], where each match from the convolutional net is restricted to lie on its epipolar line. We post-process our flow estimate using left-right consistency to reject outliers, followed by EpicFlow [141] for the final interpolation. Additionally, we take advantage of slanted plane methods [95] for background flow estimation to increase smoothness for texture-less and saturated regions. For the background, we take advantage of slanted plane methods [95], which provide further smoothness for texture-less and saturated regions.

We demonstrate the effectiveness of our approach in the challenging KITTI 2015 flow benchmark [122] and show that our approach outperforms all previously published approaches by a large margin at the time of publishing the corresponding paper [7].

In the following, we first review related work in Section 4.1 and then discuss our convolutional net for flow estimation in Section 4.2. We then present our novel approach that encodes flow as a collection of rigidly moving objects in Section 4.3, followed by our experimental evaluation in Section 4.4.

4.1 Related Work

The classical approach for optical flow estimation involves building an energy model, which typically incorporates image evidence such as gradient consistency [111, 75], warping [134], or matches [141] as unary terms. Additionally, there is a pairwise term to encourage smoothness. There are various methods for energy minimization and embedding of additional priors. This section summarizes several major categories.

The study of [159] shows that classical approaches to optical flow estimation are mainly gradient based meth-

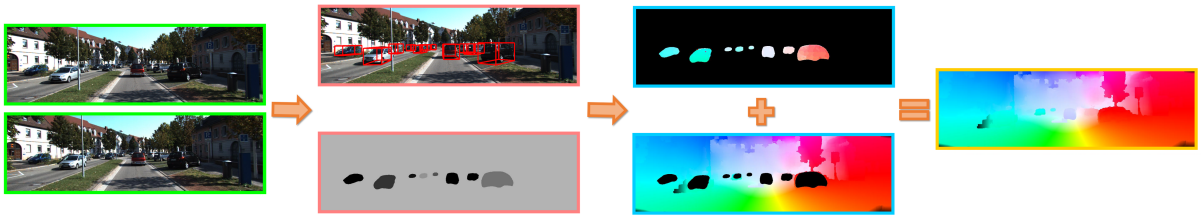


Figure 4.2: Full pipeline of our approach. We take the input image, segment the potentially moving vehicles from the background, estimate the flow individually for every object and the background, and combine the flow for the final result.

ods [111, 75]. Unfortunately, these are typically unsuitable for estimating large displacements (often encountered in traffic scenes) due to inconsistent image patch appearances. Both coarse-to-fine strategies [44] as well as inference at the original image resolution are employed [1, 173]. EpicFlow [141] is a global approach that is very often used to interpolate sparse flow fields taking into account edges [174]. As shown in our experiments, its performance can be improved even further when augmented with explicit reasoning about moving objects.

Many approaches formulate flow as inference in a Markov random field (MRF) [188, 11, 99, 123, 34]. Message passing or move making algorithms are typically employed for inference. One of the most successful optical flow methods in the context of autonomous driving is DiscreteFlow [123], which reduces the search space by utilizing only a small number of proposals. These are shared amongst neighbors to increase matching performance and robustness. An MRF is then employed to encourage smoothness. After some post processing, the final flow is interpolated using EpicFlow [141]. [189] segments images using superpixels and approximates flow of each superpixel as homographies of 3D planes. Unlike our method, these methods do not exploit the fact that the background is static and only a few objects move.

Concurrent to our work, [150] also employs semantics to help optical flow. In particular, they identify three classes of components: static planar background, rigid moving objects, and elements for which a compact motion model cannot be defined. A different model is then adapted for each of the three classes to refine DiscreteFlow [123]. An affine transformation and a smooth deformation is fitted to moving vehicles, and homographies are fitted to planar backgrounds. In contrast, we use a stronger 3D epipolar motion constraint for both foreground vehicles and the entire static background. Our experiments shows that this results in much better flow estimates.

In a series of papers, Yamaguchi et al. [183, 95] exploited epipolar constraints to reduce the correspondence search space. However, they assume that the scene is static and only the camera moves, and thus cannot handle independently moving objects. 3D priors about the physical world have been used to estimate scene flow. [167] assumes a piecewise planar scene and piecewise rigid motions. Stereo and temporal image pairs are used to track these moving planes by proposing their position and orientation. [122] tracks independently moving objects by clustering super-pixels. However, both [122, 167] require two cameras.

Our approach is also related to multibody flow methods (*e.g.*, [166, 198, 143]), which simultaneously segment, track, and recover structure of 3D scenes with moving objects. However, [166] requires noiseless correspondences, [198] uses a stereo setup, and [143] has a simple data term which, unlike our approach, does not exploit deep learning.

Recent years have seen a rise in the application of deep learning models to low level vision. In the context of stereo, [194] uses a siamese network to classify matches between two input patches as either a match or not. Combined with smoothing, it achieves the best performance on the KITTI stereo benchmark. Similarly, [30] uses convolutional neural nets (CNNs) to compute the matching cost at different scales. Different CNN architec-

tures were investigated in [193]. Luo et al. [113] exploited larger context and trained the network to produce a probability distribution over disparities, resulting in better matching. Deep learning has also been used for flow estimation [42], where the authors proposed a convolution-deconvolution network (*i.e.*, FlowNet) which is trained end-to-end, and achieves good results in real-time.

4.2 Deep Learning for Flow Estimation

The goal of optical flow is to estimate a 2D vector encoding the motion between two consecutive frames for each pixel location on the first image frame, *i.e.*, we need to find the correspondent location on the second image. The typical assumption is that a local region (*e.g.*, image patch) around each pixel will look similar in both frames. Flow algorithms then search for the pixel displacements that produce the best score. This process is referred to as matching. It thus requires computing the similarity of two candidate locations. While a pixel itself contains very limited information, various methods usually leverage its surrounding patch to generate representative features for later comparison. Traditional approaches adopt hand-crafted features, such as SIFT [110], DAISY [164], census transform [144] or image gradients to represent each image patch. These are matched using a simple similarity score, *e.g.*, via an inner product on the feature space. However, these features are not very robust. Flow methods based on only matching perform poorly in practice. To address this, sophisticated smoothing techniques have been developed [183, 141, 174, 123].

Deep convolutional neural networks have been shown to perform extremely well in high-level semantic tasks such as classification, semantic segmentation, and object detection. Recently, they have been successfully trained for stereo matching [113, 194], producing state-of-the-art results in the challenging KITTI benchmark [53]. Following this trend, in our work, we adopt a deep convolution neural network to learn feature representations that are tailored to the optical flow estimation problem. Our network can learn a good representation of pixels using its surrounding neighborhood. In the following, we will describe how we can apply deep convolution neural networks to image matching in the context of flow estimation.

4.2.1 Network Architecture

Our network takes two consecutive frames as input and processes them in two branches of a siamese network to extract features. The two branches are then combined with a product layer to create a matching score for each possible displacement. We refer the reader to Fig. 4.3 for an illustration of our convolutional net. In particular, it consists of two parts: a Siamese network at the bottom and a matching network at the top. The Siamese network contains two branches whose weights are shared and extracts features that are useful for matching. It will process the first and second image frame separately at the same time. The matching network is a dot product layer, which is used to compute similarity score using features generated from the bottom network.

Our network uses nine convolutional layers, where each convolution is followed by batch normalization [84] and a rectified non-linear unit (ReLU). We use 3×3 kernels for each convolution layer. With a stride of one pixel and no pooling, this gives us a receptive field size of 19×19. The number of filters for each convolution layer varies. As shown in Fig. 4.3, we use the following configuration for our network: 32, 32, 64, 64, 64, 128, 128, 128. Note that although our network has nine layers, the number of parameters is only around 620K. Therefore, our network is much smaller than networks used for high-level vision tasks, such as AlexNet or VGG, which have 60 and 135 million parameters, respectively. As our last layer has 128 filters, the dimension of our feature vector for each pixel is also 128.

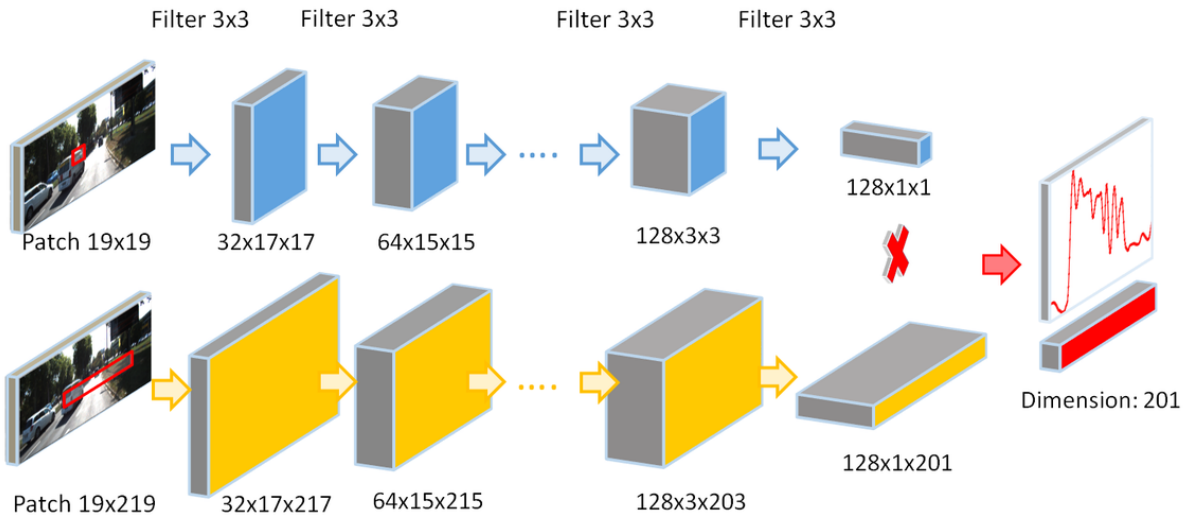


Figure 4.3: **Network Overview**: A siamese convolutional net is followed by a product layer that computes a score for each displacement. During training, for each pixel we compute a softmax over a horizontal or vertical 1D displacement, and minimize cross-entropy.

4.2.2 Learning

To train the network, we use small image patches extracted at random from the set of pixels for which ground truth is available. This strategy is beneficial, as it provides us with a diverse set of training examples (as nearby pixels are very correlated). Furthermore, it is more memory efficient. Let $\mathcal{I}$ and $\mathcal{I}'$ be two images captured by the same camera at two consecutive times. Let (x_i, y_i) be the image coordinates of the center of the patch extracted at random from $\mathcal{I}$, and let (f_{x_i}, f_{y_i}) be the corresponding ground truth flow. We use a patch of size 19×19 since this is the size of our total receptive field. Since the magnitude of (f_{x_i}, f_{y_i}) can be very large, we create a larger image patch in the second image $\mathcal{I}'$. Including the whole search range is computationally very expensive, as this implies computing 30K scores. Instead, we reduce the search space and construct two training examples per randomly drawn patch, one that searches in the horizontal direction and another in the vertical direction, both centered on the ground truth point $(x + f_{x_i}, y + f_{y_i})$. The horizontal training example is shown in Fig. 4.3. Thus, their size is $19 \times (19 + R)$ and $(19 + R) \times 19$, respectively. Note that this poses no problem as we use a convolutional net. In practice, we use $R = 200$. We find the network performance is not very sensitive to this hype-parameter.

As we do not use any pooling and a stride of one, the siamese network outputs a single feature vector from the left branch and $(1 + R)$ feature vectors from the right branch corresponding to all candidate flow locations. Note that by construction, the ground truth is located in the middle of the patch extracted in $\mathcal{I}'$. The matching network on top then computes the corresponding similarity score for each possible location. We simply ignore the pixels near the border of the image and do not use them for training.

We learn the parameters of the model by minimizing cross entropy, where we use a soft-max over all possible flow locations. We thus optimize:

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \sum_{s_i} p_i^{GT}(s_i) \log p_i(s_i, \mathbf{w}).$$

where $\mathbf{w}$ are the parameters of the network, and N is the total number of training examples. N is double the number of sample patches, as we generate two training examples for each patch. In practice, we generate 22

million training examples from the 200 image pairs. Further, s_i is the ground truth location index for patch i in the second image. Recall that the second image patch was of size $(19+R)$ or $(R+19)$. Finally, p_i^{GT} is the target distribution, and p_i is the predicted distribution for patch i according to the model (*i.e.*, output of the soft-max).

Note that when training neural nets, p^{GT} is typically assumed to be a delta function with non-zero probability mass only for the correct hypothesis. Here, we use a more informative loss, which penalizes depending on the distance to the ground truth configuration. We thus define

$$p_i^{GT}(s_i) = \begin{cases} \lambda_1 & \text{if } s_i = s_i^{GT} \\ \lambda_2 & \text{if } |s_i - s_i^{GT}| = 1 \\ \lambda_3 & \text{if } |s_i - s_i^{GT}| = 2 \\ 0 & \text{o.w.} \end{cases} .$$

This allows the network to be less strict in discriminating patches within 3-pixels from the ground truth. In practice we choose $\lambda_1 = 0.5$, $\lambda_2 = 0.2$ and $\lambda_3 = 0.05$.

4.2.3 Inference

In contrast to training where we select small image patches, during inference, we need to evaluate all the pixels for the first image frame. Using the same routine as for learning would result in as many forward passes as the number of pixels in the image, which is computationally very expensive. Similar to the following stereo approaches [194, 113] that focus on stereo estimation with a similar network architecture, we can efficiently compute the feature vectors for all pixels with the siamese network using only one forward pass. A similar trick was also used when training FastRCNN [56], where features for all regions proposal are computed by one forward pass.

Optical flow is more challenging than stereo matching because the search space is approximately 200 times larger, as one has to search over a 2D space. A standard searching window of size 400×200 would require 300GB space to store the whole cost volume for a single image, which is prohibitive. Instead, we propose to use only the first top-K candidates for every location. This also enables the network to handle better texture-less regions as detailed in the next section. Notice that the matching is performed at each location independently, thus in order to get smooth matching results, we apply different simple post-processing techniques. This can improve the results, especially at textureless regions, as well as to better deal with occlusion and specularities.

Cost aggregation: We first utilize a simple cost aggregation to smooth the matching results, which can be noisy as the receptive field is only 19×19. Cost aggregation is an iterative process which, for every location i, updates the cost volume c_i using the cost values of neighborhood locations *i.e.*, $c_i^t(s_i) = \frac{\sum_{j \in \mathcal{N}(i)} c_j^{t-1}(s_i)}{N}$, where $\mathcal{N}(i)$ is the set of neighbor locations of i, $c_i^t(s_i)$ is the cost volume at location i during the t-th aggregation iteration, s_i is the flow configuration id, and $c_i^0(s_i)$ is the raw output from our network. Note that applying cost aggregation multiple times is equivalent to performing a weighted average over a larger neighborhood. In practice, we use 4 iterations of cost aggregation and a 5×5 window size. Because we only store top-K configurations in our cost volume to reduce memory usage, neighboring locations have different sets of label ids. Thus, we perform cost aggregation on the union of label sets and store only the top-K results after aggregation as final results $c_i^T(s_i)$. Note that one can interpret $c_i^T(s_i)$ as a score of the network's confidence. We thus threshold the cost $c_i^T(s_i)$ to select the most confident matches. The threshold is selected such that on average 60% of the locations are estimated as confident. This simple thresholding on the cost aggression allows us to eliminate most specularities

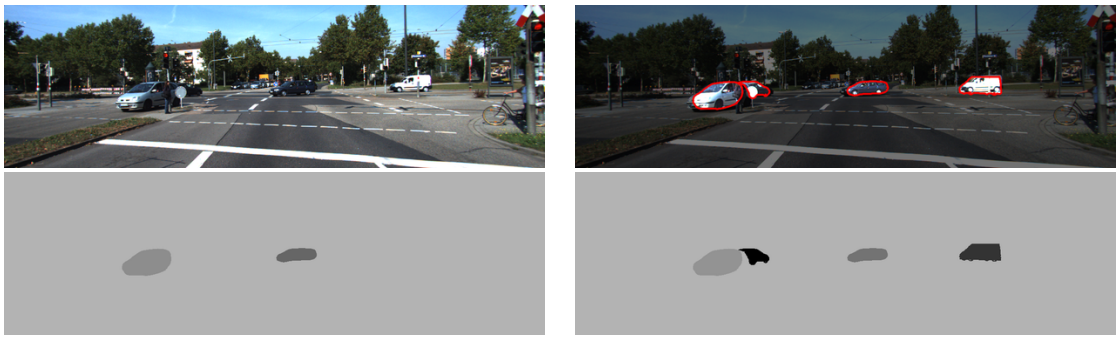


Figure 4.4: **Top left**: KITTI image. **Top right**: Instance segmentation masks overlaid on input image. **Bottom left**: Car segmentation masks from [201]. **Bottom right**: Segmentation instances augmented by 3D detection [25] followed by CAD model fitting [51].

and shadows. In texture-less regions, the sparse top-K predicted matches sets of neighboring pixels have very little overlap. Combined with cost aggregation, scores of erroneous matches decrease through the aggregation iterations, thus eliminating erroneous matches. Another possible solution would be using uncertainty estimation by computing the entropy at each pixel. However, our experiments show that selecting top-K combined with simple thresholding works much better than thresholding the entropy. In practice, we used K = 30 as it balances memory usage and performance.

4.3 Object-Aware Optical Flow

In this section, we discuss our parameterization of the optical flow problem as a result of the projection of the 3D scene flow. In particular, we assume that the world encountered in autonomous driving scenarios consists of independently moving rigid objects. The ego-car, where the camera is located, is a special object, which is responsible for the optical flow of the static background.

Our approach builds on the observation that if the 3D motion of an object is rigid, it can be parameterized with a single transformation. This is captured by the fundamental matrix, which we denote by $F \in \mathcal{R}^{3\times3}$ with $\text{rank}(F) = 2$. Let $\mathcal{I}$ and $\mathcal{I}'$ be two images captured by a single camera at two consecutive times, then for any point in a rigidly moving object, the following well-known epipolar constraint holds

$$\tilde{\mathrm{p}}_i'^{\top} F \tilde{\mathrm{p}}_i = 0$$

where $\mathrm{p}_i = (x_i, y_i)$ and $\mathrm{p}_i' = (x_i', y_i')$ are the projection of a 3D point p_i into the two images, and $\tilde{\mathrm{p}} = (x, y, 1)$ is p in homogeneous coordinates. Further, the line defined by $l_i' = F_i \tilde{\mathrm{p}}_i$ is the epipolar line in $\mathcal{I}'$ corresponding to point p, passing through both the epipole in $\mathcal{I}'$ and p_i'.

4.3.1 Segmenting Traffic Participants

Since only pixels belonging to one independently moving vehicle obey the same epipolar constraint, it is necessary to obtain a segmentation of the scene into independently moving objects. This is traditionally done by clustering the motion estimates. In this chapter, we take an alternative approach and use semantics to infer the set of potential traffic participants. Towards this goal, we exploit instance-level segmentation, which segments each

traffic participant into a different component. Note that we aim at an upper bound on the number of moving objects, as some of the vehicles might be parked.

To compute instance-level segmentations, we exploit the approach of [201], which uses a multi-resolution CNN followed by a fully connected conditional random field to create global labeling of the scene in terms of instances. Since only labeled training data for *cars* was available; the method is unable to detect vans and trucks. This results in high precision but lower recall. To partially alleviate this shortcoming, we augment the instance segmentation results with extra segmentations which are computed by performing 3D detection [25] followed by CAD model fitting. In particular, we simply go over all the CAD models and select the one which best aligns with the 3D box, following the technique in [51]. Since this process has higher recall but lower precision than the instances of [201], we only add new segmentation masks if they do not overlap with the previously computed masks. We refer the reader to Fig. 4.4 for an example. This process provides us with a segmentation of the scene in terms of rigidly moving objects. We now discuss how to estimate flow for each moving object as well as for the background.

4.3.2 Foreground Flow Estimation

Our first goal is to reliably estimate the fundamental matrix describing the motion of each moving object. We consider this motion to be the combination of the vehicle's motion and the motion of the ego-car, that is the 3D motion whose projection we observe as optical flow. This is a challenging task, as moving objects can be very small and contain many specularities. We take advantage of the fact that our convolutional net outputs an uncertainty estimate, and only use the most confident matches for this task. In particular, we use RANSAC with the 8 point algorithm [64] to estimate the fundamental matrix of each moving object independently. We then choose the hypothesis with a smaller median squared error, where the error is defined as the shortest distance between each matching point

Following [183], we consider the optical flow $\mathrm{u_p} = (\mathrm{u}_x, \mathrm{u}_y)$ at point p to decompose into its rotational and translational components. Thus

$$\mathrm{u}_{\mathrm{p}_k} = \mathrm{u_w}(\mathrm{p}_k) + \mathrm{u_t}(\mathrm{p}_k, Z_{\mathrm{p}_k})$$

, where $\mathrm{u_w}(\mathrm{p})$ is a component of the flow of pixel p due to the rotation, and $\mathrm{u_t}(\mathrm{p}, Z_{\mathrm{p}})$ is a component of the flow from the translation of the object relative to the camera. Note that the direction Z here is perpendicular to the image plane of $\mathcal{I}'$. If the rotation is small, the rotational component can be linearized. We estimate the linear coefficients using matched point pairs, with the additional constraint that the point $\mathrm{p} + \mathrm{u_w}(\mathrm{p})$ must lie on the epipolar line in the second image.

Upon application of the aforementioned linear transformation to $\mathcal{I}$, the image plane of the image patches corresponding to the object is now parallel and related to each other only by a relative translation. This reduces the problem to either an epipolar contraction or epipolar expansion, where matching point pairs both lie on the same epipolar line. Therefore, the search for a matching point is reduced to a 1D search along the epipolar line. The flow at a given point is then parameterized as the disparity along the epipolar line between its rectified coordinates and its matching point.

To smooth our results, we exploit semi-global block matching (SGM) [73]. In particular, we parameterized the problem using disparity along the epipolar line as follows:

$$E(d) = \sum_{\mathrm{p}_k} C'(\mathrm{p}_k, d_{\mathrm{p}_k}) + \sum_{\mathrm{p}_k, \mathrm{p}'_k \in \mathcal{N}} S(d_{\mathrm{p}_k}, d_{\mathrm{p}'_k})$$

with $C'(\mathrm{p}_k, d_{\mathrm{p}_k})$ being the matching similarity score computed by our convolutional net with local cost aggregation to increase robustness to outliers. Note that the vz-ratio parameterization of disparity in [183] is unsuitable for foreground objects, as it relies on a significant relative motion in the z-direction (perpendicular to the image plane). This assumption is often violated by foreground vehicles, such as those crossing an intersection in front of the static observer. We use a standard smoothing term

$$S(d_{\mathrm{p}_k}, d_{\mathrm{p}'_k}) = \begin{cases} \lambda_1 & \text{if } |d_{\mathrm{p}_k} - d_{\mathrm{p}'_k}| = 1 \\ \lambda_2 & \text{if } |d_{\mathrm{p}_k} - d_{\mathrm{p}'_k}| > 1 \\ 0 & \text{otherwise} \end{cases}$$

with $\lambda_2 > \lambda_1 > 0$. In practice, $\lambda_2 = 256$ and $\lambda_1 = 32$. After SGM, we use left-right consistency check to filter out outliers. The output is a semi-dense flow estimate.

Occasionally, the fundamental matrix estimation for an object fails due to either too few confident matches or too much noise in the matches. In this case, we directly use the network's matching to obtain a flow-field. Finally, we use the edge-aware interpolation of EpicFlow [141] to interpolate the missing pixels by performing one step of variational smoothing. This produces a fully dense flow-field for all objects.

4.3.3 Background Flow Estimation

To estimate the background flow, we mostly follow Yamaguchi et al. [183]. However, we make two significant changes which greatly improve its performance. First, we restrict the matches to the areas estimated to be background by our semantic segmentation. We use RANSAC and the 8-point algorithm with SIFT to estimate the fundamental matrix. Note that this simple approach is sufficient as background occupies most of the scene. Similar to the foreground, the flow u_p at a point p is considered to be a sum of a rotational and a translational component: $\mathrm{u}_\mathrm{p} = \mathrm{u}_\mathrm{w}(\mathrm{p}) + \mathrm{u}_\mathbf{t}(\mathrm{p}, Z_\mathrm{p})$. Again, we linearize the rotational component. To find the matching point p′ for p, we search along the epipolar line l', and parameterize the displacement vector as a scalar disparity.

Further, the disparity at point p_i can be written as

$$d(\mathrm{p}, Z_\mathrm{p}) = |\mathrm{p} + \mathrm{u}_\mathrm{w}(\mathrm{p}) - \mathrm{o}'| \frac{\frac{v_z}{Z_\mathrm{p}}}{1 - \frac{v_z}{Z_\mathrm{p}}}$$

where v_z is the forward (Z) component of the ego-motion, o′ is the epipole and $\omega_\mathrm{p} = \frac{v_z}{Z_\mathrm{p}}$.

We use SGM [73] to smooth the estimation. However, we parameterize the flow in terms of the vz-ratio instead of directly using disparity as in the case of foreground flow estimation. We perform inference along four directions and aggregate the results. Finally, we post-process the result by checking left-right consistency to remove outliers. This provides us with a semi-dense estimate of flow for the background pixels.

Unfortunately, no matches are found by the matching pair process in occluded regions such as portions of road or buildings that disappear from view as the vehicle moves forward. An additional significant improvement over [183] is a 3D geometry-inspired extrapolation. Let $\delta_\mathrm{p} = |\mathrm{p} + \mathrm{u}_\mathrm{w}(\mathrm{p}) - \mathrm{o}'|$ be the distance between the point p and o′. For a planar surface in the 3D world, δ_p is inversely proportional to Z_p. Since v_z is constant for all points after the linearized rotational flow component $\mathrm{u}_\mathrm{w}(\mathrm{p})$ is removed, the vz-ratio is also proportional to δ_p. For each point p where the vz-ratio is not estimated, we search along the line segment joining p to o′ to collect a set of up to 50 vz-ratios at pixels p′ and calculate their associated $\delta_{\mathrm{p}'}$. We take advantage of semantic information to exclude points belonging to moving foreground vehicles. Using this set, we fit a linear model which we use to estimate the missing vz-ratio at p.

Method	Non occluded px			All px		
	Fl-bg	Fl-fg	Fl-all	Fl-bg	Fl-fg	Fl-all
HS [159]	30.49 %	50.59 %	34.13 %	39.90 %	53.59 %	42.18 %
DeepFlow [173]	16.47 %	31.25 %	19.15 %	27.96 %	35.28 %	29.18 %
EpicFlow [141]	15.00 %	29.39 %	17.61 %	25.81 %	33.56 %	27.10 %
MotionSLIC [183]	6.19 %	64.82 %	16.83 %	14.86 %	66.21 %	23.40 %
DiscreteFlow [123]	9.96 %	**22.17 %**	12.18 %	21.53 %	**26.68 %**	22.38 %
SOF [150]	8.11 %	23.28 %	10.86 %	14.63 %	27.73 %	16.81 %
Ours	**5.75 %**	22.28 %	**8.75 %**	**8.61** %	26.69 %	**11.62 %**

Table 4.1: **KITTI Flow 2015 Test Set**: we compare our results with top scoring published monocular methods that use a single image pair as input

Method	Non occluded px			All px		
	Fl-bg	Fl-fg	Fl-all	Fl-bg	Fl-fg	Fl-all
EpicFlow [141]	16.14 %	28.75 %	18.66 %	27.28 %	31.36 %	28.09 %
MotionSLIC [183]	6.32 %	64.88 %	17.97 %	15.45 %	65.82 %	24.54 %
DiscreteFlow [123]	10.86 %	**20.24 %**	12.71 %	22.80 %	**23.32 %**	22.94 %
Ours	**6.21 %**	21.97 %	**9.35 %**	**9.38 %**	24.79 %	**12.14 %**

Table 4.2: **KITTI Flow 2015 5-Flod Validation**: we compare our results with the state of the art (at the time of publication of the corresponding paper [7]) by averaging performance over 5 different splits of the KITTI training dataset into training/testing.

We employ a slanted plane model similar to MotionSLIC [183] to compute a dense and smooth background flow field. This assumes that the scene is composed of small, piecewise planar regions. In particular, we model the vz-ratios of the pixels in each superpixel with a plane defined as $\frac{v_z}{Z_\mathrm{p}} = A(x-x_c)+B(y-y_c)+C$. Here, (A, B, C) are the plane parameters, and (x_c, y_c) are the coordinates of the center of the superpixel. We simultaneously reason both about the assignments of pixels to planes, the plane parameters, and the types of boundaries between superpixels (*i.e.*, coplanar, hinge, occlusion). The inference is performed by block coordinate descent.

4.4 Experimental Evaluation

We evaluated our approach on the widely used self-driving dataset: KITTI Optical Flow 2015 benchmark [53], which consists of 200 training and 200 testing image pairs. There are several challenges, including specularities, moving vehicles, sensor saturation, large displacements, and texture-less regions. The benchmark's scoring method is as follows. Image pixels have two attributes - occluded / non-occluded and background/foreground. The former attribute indicates whether the same 3D world point visible in $\mathcal{I}$ is still visible in $\mathcal{I}'$. The latter attribute indicates whether the pixel belongs to a moving foreground object (Fl-fg) or the static background (Fl-bg). The foreground and background pixels can be combined into Fl-all. The estimated flow at a pixel is deemed correct when it deviates less than 3 px or 5 % (whichever is greater) from the ground truth flow. All of our analysis, henceforth also uses these definitions.

We trained our siamese convolutional network for 100k iterations using stochastic gradient descent with Adam [89]. We used a batch size of 128 and an initial learning rate of 0.01 with a weight decay of 0.0005. We divided the learning rate by half at iterations 40K, 60K, and 80K. Note that since we have 22 million training examples, the network converges before completing one full epoch. This shows that 200 images are more than enough to train the network. Training takes 17 hours on an NVIDIA-Titan Black GPU. However, performance improves

Method	Non occluded px			All px		
	Fl-bg	Fl-fg	Fl-all	Fl-bg	Fl-fg	Fl-all
[139]	**6.17** %	25.30 %	9.98 %	**9.31 %**	28.11 %	12.69 %
[201]	6.18 %	24.61 %	9.82 %	9.35 %	27.31 %	12.54 %
[139] augmented with [25]	**6.17** %	22.06 %	**9.35 %**	**9.31 %**	25.08 %	12.15 %
[201] augmented with [25]	6.21 %	**21.97 %**	**9.35 %**	9.38 %	**24.79 %**	**12.14 %**

Table 4.3: Flow estimation with various instance segmentation algorithms

	Within Detected Object Masks		Within Ground Truth Object Masks	
Method	Non-occ px error %	All px error %	Non-occ px error %	All px error %
EpicFlow [141]	26.77 %	29.93 %	28.75 %	31.36 %
DiscreteFlow [123]	18.76 %	22.42 %	20.24 %	23.32 %
Ours	**15.91 %**	**19.72 %**	**15.42 %**	**18.62 %**

Table 4.4: Foreground flow estimation within detected object masks and within ground truth masks

Source for Matches for F Estimation	Non-occluded px error %	All px error %
EpicFlow [141]	25.02 %	27.58 %
DiscreteFlow [123]	23.40 %	26.05 %
Our Matching Network	**21.97 %**	**24.79 %**

Table 4.5: Foreground flow estimation errors when F is estimated from various sources

only slightly after 70k iterations.

In this section, we first analyze our method's performance in comparison with the state-of-the-art. Additionally, we explore the impact of various stages of our pipeline.

Comparison to state of the art[1]: We first present our results[2] on the KITTI Optical Flow 2015 test set, and compare our approach to published monocular approaches that exploit a single temporal image pair as input. As shown in Table 4.1, our approach significantly outperforms all published approaches. Our approach is particularly effective on the background, outperforming MotionSLIC [183]. Moreover, our method's foreground performance is on par with the leading foreground estimation technique DiscreteFlow [123]. The test set images are fairly correlated, as many pairs are taken from the same sequence. To provide further analysis, we also computed results on the training set with 5-fold validation. In particular, for each fold, 160 images are used for training, and the remaining 40 are used for testing. The same improvements on the test set can be seen in Table 4.2.

Influence of instance segmentation: Table 4.3 shows performance when using [139] and [201] to create the instance segmentations. We also explore augmenting them by fitting CAD models with [51] to the 3D detections of [25]. Note that a combination of segmentation and detection is beneficial. A limiting factor of our foreground flow estimation performance arose when we missed moving vehicles when estimating our instances. Using our five folds on the training set, we explore what happens when we have perfect objects masks. Towards this goal, we first examine the foreground flow estimation performance only on our detected vehicle masks. The left half of Table 4.4 shows that within the vehicle masks we detect, our flow estimation is significantly more accurate than our competitors in the same regions. Moreover, the right half of the same table shows that the same is true within

[1]We are comparing to the 'state of the art' at the time of publication of the corresponding paper [7], *i.e.* year of 2016

[2]We exploit the instances of [139] for our submission to the evaluation server.

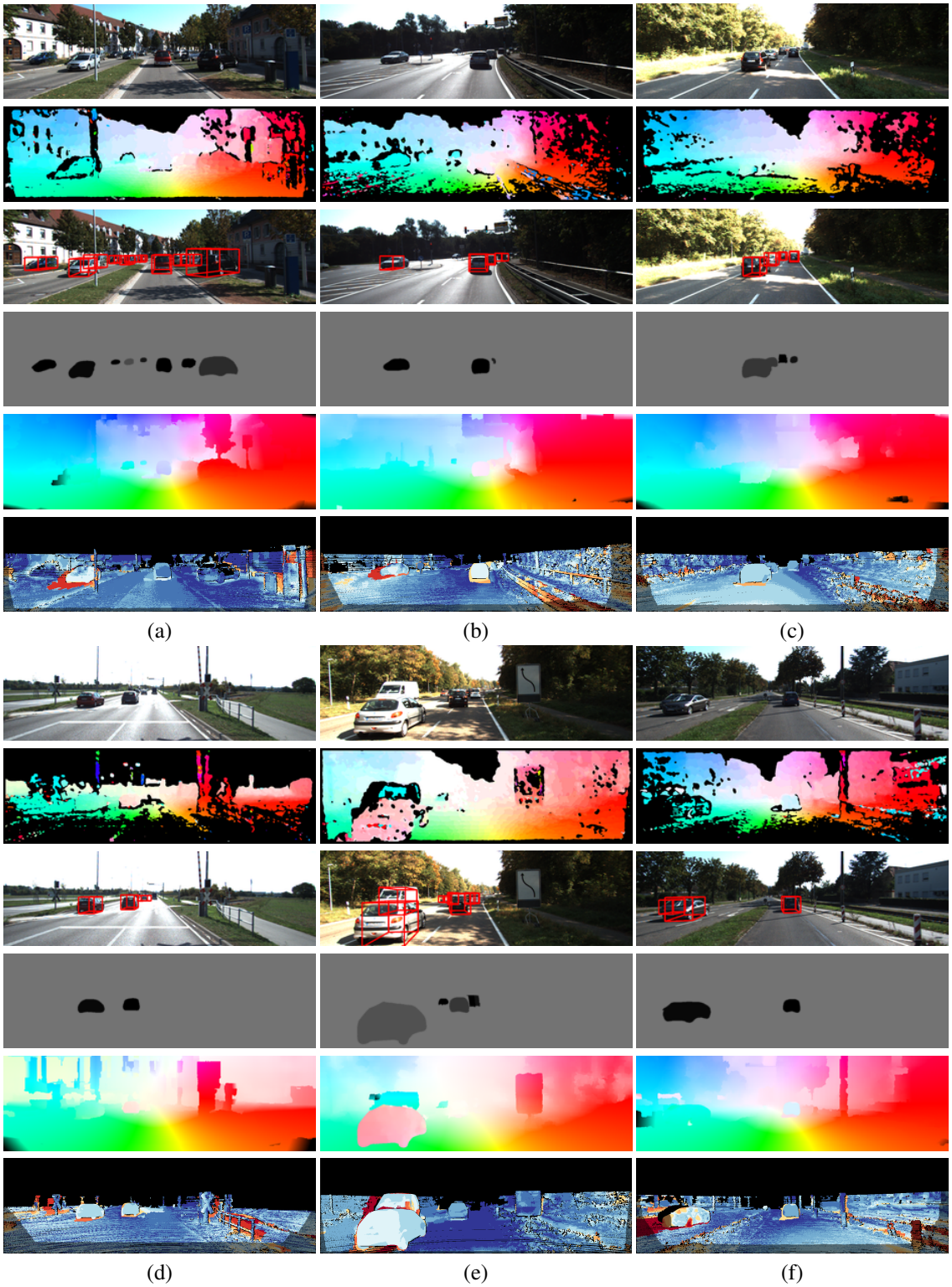
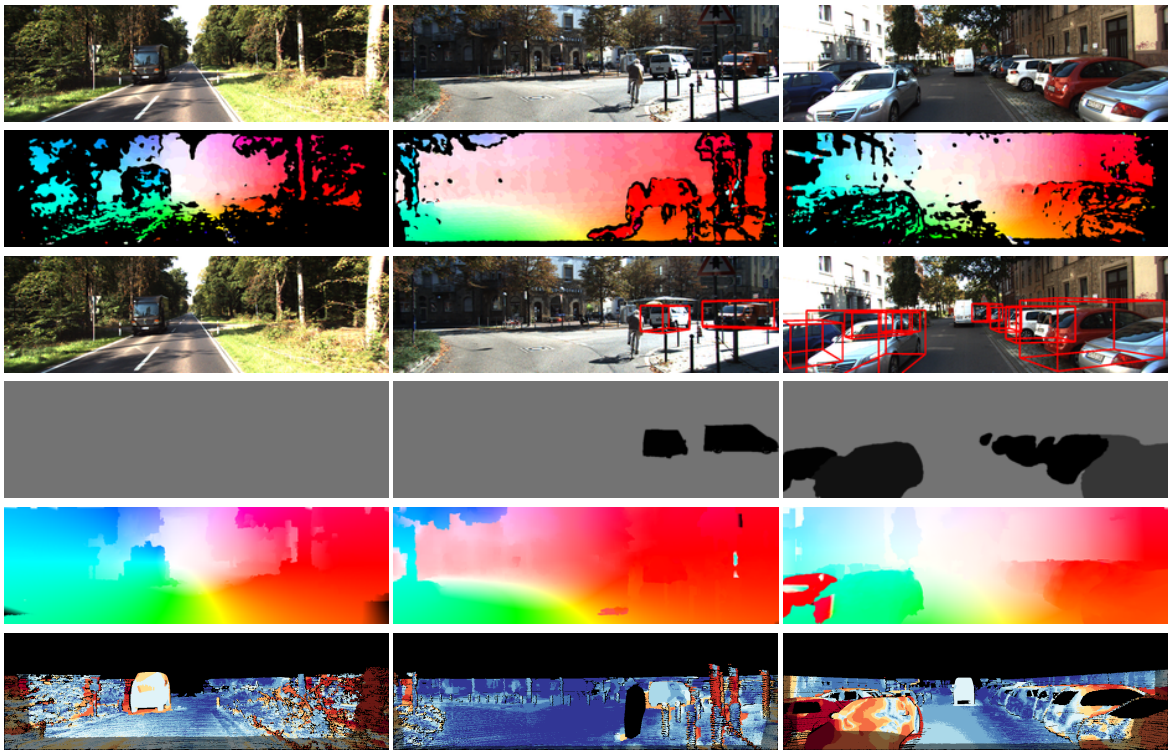


Figure 4.5: Examples of successful flow estimations. Within each group, from top to bottom: first frame of input image, confident flow produced by our network, 3D car detection results, instance segmentation output augmented by 3D car detection, final flow field, and flow field error.



(a) (b) (c)

Figure 4.6: Failure cases for our algorithm: (a) fails to segment the van; (b) incorrect CAD model fitting; (c) Incorrect estimation of fundamental matrix

the ground truth object masks. Thus, if the instances were further improved (*e.g.*, by incorporating temporal information when computing them), our method can be expected to achieve more improvement over the leading competitors.

Estimating Fundamental Matrix: Having an accurate fundamental matrix is critical to the success of our method. While the strong epipolar constraint offers great robustness to outliers, it can also cause many problems if it is wrongly estimated. We now compare different matching algorithms employed to compute the fundamental matrices and use the rest of our pipeline to estimate flow. As shown in Table 4.5, selecting only confident matches from our network to estimate the fundamental matrix is significantly better than using the flow field estimations from other algorithms, including DiscreteFlow.

Qualitative Analysis: Fig. 4.5 shows qualitative results, where each column depicts the original image, the network most confident estimates, the 3D detections of [25], our final instance segmentations, our final flow field, and its errors. Our convolutional net can predict accurate results for most regions in the image. It leaves holes in regions including textureless areas like the sky, occlusion due to the motion of the car and specularities on the windshield. The 3D detector can detect almost all cars, regardless of their orientation and size. Our final object masks used to label foreground objects are very accurate and contain many cars of different sizes and appearances. Note that different shades represent distinct car instances whose fundamental matrices are estimated separately. As shown in the last two rows, we produce very good overall performance.

Failure Modes: Our technique has several failure modes. If a car is not segmented, the estimation of flow defaults to using the epipolar constraint of the background. This happens particularly often with trucks and vans, as we do not have training examples of these types of vehicles to train our segmentation and detection networks. Fig. 4.6(a) shows an example where a van is not segmented. By coincidence, its true epipolar lines are almost identical with those calculated using the background fundamental matrix. As such, its flow estimation is still mostly correct. If object masks contain too many background pixels (which are outliers from the perspective of foreground fundamental matrix estimation), our algorithm can also fail. This is commonly associated with objects identified by the 3D object detector rather than the instance-segmentation algorithm, as the 3D detection box might be misaligned with the actual vehicle. Moreover, fitting CAD models to monocular images is not a trivial task. The right-most car in Fig. 4.6(b) is such an example. The other failure mode of our approach is the wrong estimation of the fundamental matrix, which can happen when the matches are very sparse or contain many outliers. Fig 4.6(c) shows such an example, where the fundamental matrix of the left-most car is incorrectly estimated due to the sparseness in confident matching results (demonstrated in the second row).

4.5 Discussion

In this chapter, we tackled the problem of estimating optical flow from a monocular camera in the context of autonomous driving. We built on the observation that the scene is typically composed of a static background, as well as a relatively small number of traffic participants that move rigidly in 3D. We have shown how instance-level segmentation and 3D object detection can be used to segment the different vehicles and proposed a new convolutional network that can accurately match patches. We proposed to estimate the traffic participants using instance-level segmentation. For each traffic participant, we used the epipolar constraints that govern each independent motion for faster and more accurate estimation. Our second contribution is a new convolutional net that learns to perform flow matching and can estimate the uncertainty of its matches. This is a core element of our flow estimation pipeline. We demonstrated the effectiveness of our approach on the challenging KITTI 2015 flow benchmark. We have shown in extensive experiments that our approach outperforms published approaches (at the time of publication of the corresponding paper [7]) by a large margin.

On the other hand, there are different ways to improve our approach further. First of all, a better instance segmentation mask would improve the performance of optical flow in our case. Different superior methods on instance segmentation have been proposed since the development of our approach in this chapter *i.e.* year of 2016. Bai and Urtasun [8], inspired by watershed transform, proposed an end-to-end convolutinal network that achieved state of the art results on Cityscapes dataset [36]. Li *et al.* [102] proposed a fully convolutional instance-aware segmentation method that combined instance mask proposals [37] and fully convolutinal segmentation networks [108]. The idea is to predict a set of position-sensitive output channels that simultaneously predict object classes, boxes, and masks. Later, Mask-RCNN [66] was proposed to use two parallel headers to jointly predict object mask and perform recognition. It utilized RoIAlign operation to address the issues [102] had on overlapping instances and spurious edges. Secondly, the application of optical flow in the domain of autonomous driving is not only limited to rigid object, *i.e.* vehicles, but also other non-rigid dynamic objects such as pedestrians, cyclists. Further, the self-driving vehicle needs to detect unknown/non-categorized moving objects, *e.g.* flying plastic bags or basketball, on the road. A safe autonomous driving system needs to be robust to these rare cases. Using optical flow could help in these cases by first generating the optical flow of all pixels. Then, we can perform clustering to group pixels, which will inform us if there exist moving objects in the scene. Thirdly, due to the limitation of computation resources on the vehicle, we need to further improve the algorithm from the runtime perspective.

Nowadays, lots of tasks in the autonomy stacks require GPU hungry algorithms, *i.e.* deep learning models. We would need to share both the computation power (*i.e.* GPU time) as well as the limited GPU memory. Various end-to-end flow estimation models [83, 82] have been proposed following FlowNet [42]. They all directly regress optical flow from images, without explicitly doing image patch matching. These end-to-end frameworks could run faster since there is no need for dense patch matching. However, in order to comparable performance, they all utilize very large neural networks that make these approaches less applicable. SPyNet [137], being 96% smaller than FlowNet in terms of model parameters, utilizes spatial pyramid network to handle large motions and achieves similar performance as FlowNet. Following this idea, PWC-Nets [161, 160] also utilize pyramid processing and incorporate warping as well as cost volume into their networks, making the models 17 times faster with better performance than FlowNet. Hui *et al.* [79, 80] proposed LiteFlownet that also outperforms FlowNet while being 25.3 times smaller and 3.1 times faster. Recently, Yin *et al.* [190] propose a hierarchical matching procedure for optical flow with the help of pyramid features. They are able to achieve state-of-the-art performance on KITTI and run real time, *i.e.* 80ms per frame.

Chapter 5

Joint 3D Detection, Tracking and Prediction

In previous chapters, we have developed algorithms to help the self-driving vehicle better understand the environment from the low-level vision perspective. In particular, we have developed a convolutional neural network to estimate the depth of the scene in chapter 3. In chapter 4, we took it one step further to apply the underlying deep matching method to estimate optical flow, *i.e.* the motion of each pixel in the image.

All the previous algorithms focus on image data which can provide very detailed information about the surrounding environment. There are different kinds of cameras with different user cases, *i.e.* long-range cameras with long focal length for seeing things far away and fish-eye cameras with extreme short focal length for seeing close objects. In certain cases, modern cameras can perform better than human eyes, *e.g.* some cameras perform better in low-light condition, while other cameras have wider field-of-view or can see further. Another advantage of using cameras is that it is a cost-effective solution. The price of high-quality cameras dropped significantly over the past decades. This makes it easier for large scale deployment. One of the good examples is Tesla, which heavily relies on cameras for their L3 self-driving package: autopilot. Comparing to a standard LiDAR, cameras can be three orders of magnitude cheaper, costing around 100 dollars while a LiDAR could cost 100 thousand dollars.

However, there are also limitations for cameras. Modern cameras still use the pin-hole camera model, where image data is a projection of the 3D real-world on the 2D image plane. Thus, the depth information is lost. For self-driving, we need to know not only where the objects are on the 2D image plane, but also where they are *w.r.t.* self-driving vehicle in 3D space, *i.e.* the depth for each object. There are two kinds of approaches we can take to tackle this problem. First, one can compute the depth for all pixels using stereo images, then perform 2D detection on the 2D image to locate objects. In chapter 3, we developed a deep neural network based algorithm for estimating depth from stereo images, where we achieved two orders of magnitude faster runtime then previous methods. However, it is still not ready yet for real-world self driving both from speed and accuracy perspectives. While it is important to acquire dense depth for the surrounding environment, in practice, not all pixels are equally important. Thus, lots of computation for stereo algorithms are not needed. Another approach is to do 3D detection from stereo images directly, without directly reasoning the depth of all pixels in the image. While this is an interesting and promising direction [25], the accuracy is still not good enough for self driving. The fundamental challenge here is the difficulty of estimating depth from 2D image data.

LiDAR can come to our rescue. LiDAR uses pulsed laser light with a sensor to receive the laser bounding

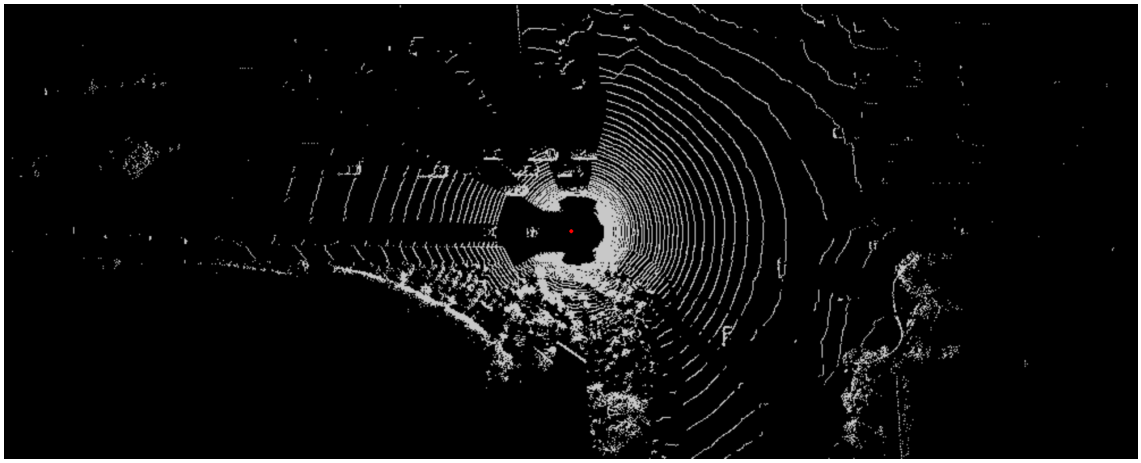


Figure 5.1: Top-down view of LiDAR point cloud data.

back. It measures the time each laser traverses forward and backward. It requires careful mechanical engineering to make it work properly and precisely, *i.e.*, it is hard to build and maintain, correspondingly very expensive. However, since LiDAR uses light and measure the delta time, the depth it measures is very accurate. This technology has been widely used in different areas such as geodesy, geomatics, archaeology, etc. It has also been used to build units that are suitable for self driving. One of the widely used LiDAR unit for self driving is Velodyne HDL-64. It is mounted on top of the self-driving vehicle and has a rotation unit spreading out laser; thus, it provides 360-degree 3D information around the self-driving vehicle. Fig. 5.1 shows a top-down view of a sample LiDAR point cloud.

Having accurate 3D information is not enough for self-driving; we need to have a high-level understanding of the surrounding environment. This includes 3D detection, tracking, and motion forecasting. In a traditional autonomy stack, these modules are usually learned independently, and uncertainty is rarely propagated. This can result in catastrophic failures as downstream processes cannot recover from errors that appear at the beginning of the pipeline.

An alternative solution could be to jointly reason about 3D detection, tracking, and motion forecasting given data captured by a 3D sensor *i.e.* LiDAR. Instead of tackling these three problems separately, a joint reasoning pipeline has three advantages: (1) we can share the heavy feature computation among different modules. As most perception tasks employ large convolutional neural networks, GPU computation is becoming a scarce resource. Thus, sharing feature computation among different modules can save computation and makes a large model run real-time; (2) it can fix the distribution mismatch between different modules. A sequential but separate pipeline would pass information through the specific format, *e.g.* objects' bounding boxes, between detection, tracking, and motion forecasting. This requires heavy engineering effort to make sure that the downstream system is tuned accordingly. Furthermore, it is difficult to propagate uncertainty throughout different tasks as downstream modules cannot affect previous modules; (3) from the optimization perspective, learning and optimizing jointly would achieve better results compared to learning separately.

Thus, in this chapter, we propose a novel deep neural network that performs 3D convolutions across space and time over a bird's-eye-view representation of the 3D world. It generates 3D detection bounding boxes, tracklets, and correspondingly motion forecasting for all vehicles in the scene at the same time. It is very efficient in terms of both memory and computation compared to using 4D convolution on 3D space and time dimensions. By jointly reasoning about these tasks, our holistic approach is more robust to occlusion as well as sparse data at the

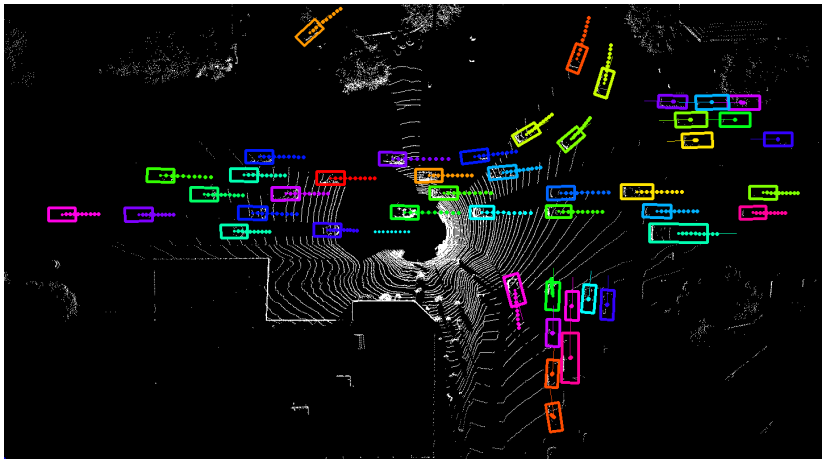


Figure 5.2: This figure shows the top down view of LiDAR point cloud with our 3D detection results on vehicles, the color for each bounding box represents tracking information while the dots (waypoints) associated to each vehicle represent its future location in discrete time steps. Notice when the waypoints overlap, it means the vehicle is static.

range. Our experiments on a new large-scale dataset captured in several North American cities show that we can outperform the state of the art by a large margin. Importantly, by sharing computation, we can perform all tasks in as little as 30 ms. Fig. 5.2 gives an example of the problem we are tackling.

5.1 Related Work

Over the past few years many methods that exploit convolutional neural networks to produce accurate 2D object detections, typically from a single image, have been developed. Convolutional neural networks have shown great performance on perceiving images, learning to extract related features. With better features extracted, we see better performance on all kinds of detection tasks, including 2D and 3D. On the other hand, tracking and motion forecasting also benefits from the powerful feature extractor and achieve better results when incorporating human domain knowledge. In the following, we introduce the advances in these areas in the past years briefly.

2D Object Detection: 2D object detection advances a lot over the past few years with the help of the convolutional neural network. People have developed different algorithms with different intuition; these approaches typically fell into two categories depending on whether they exploit a first step dedicated to creating object proposals. Modern *two-stage detectors* [140, 66, 38, 77], utilize region proposal networks (RPN) to learn the region of interest (RoI) where potential objects are located. In a second stage, the final bounding box locations are predicted from features that are average-pooled over the proposal RoI. Mask-RCNN [66] also took this approach, but used RoI aligned features addressing the boundary and quantization effect of RoI pooling. Furthermore, they added segmentation branch to take advantage of dense pixel-wise supervision, achieving state-of-the-art results on both 2D image detection and instance segmentation. On the other hand, *one-stage detectors* skip the proposal generation step, and instead learn a network that directly produces object bounding boxes. Notable examples are YOLO [138], SSD [107] and RetinaNet [105]. One-stage detectors are computationally very appealing and are typically real-time, especially with the help of recently proposed architectures, *e.g.* MobineNet [76], SqueezeNet [180]. One-stage detectors were outperformed significantly by two stage-approaches until Lin *et al.*[105] shown state-of-the-art results by exploiting a focal loss and dense predictions.

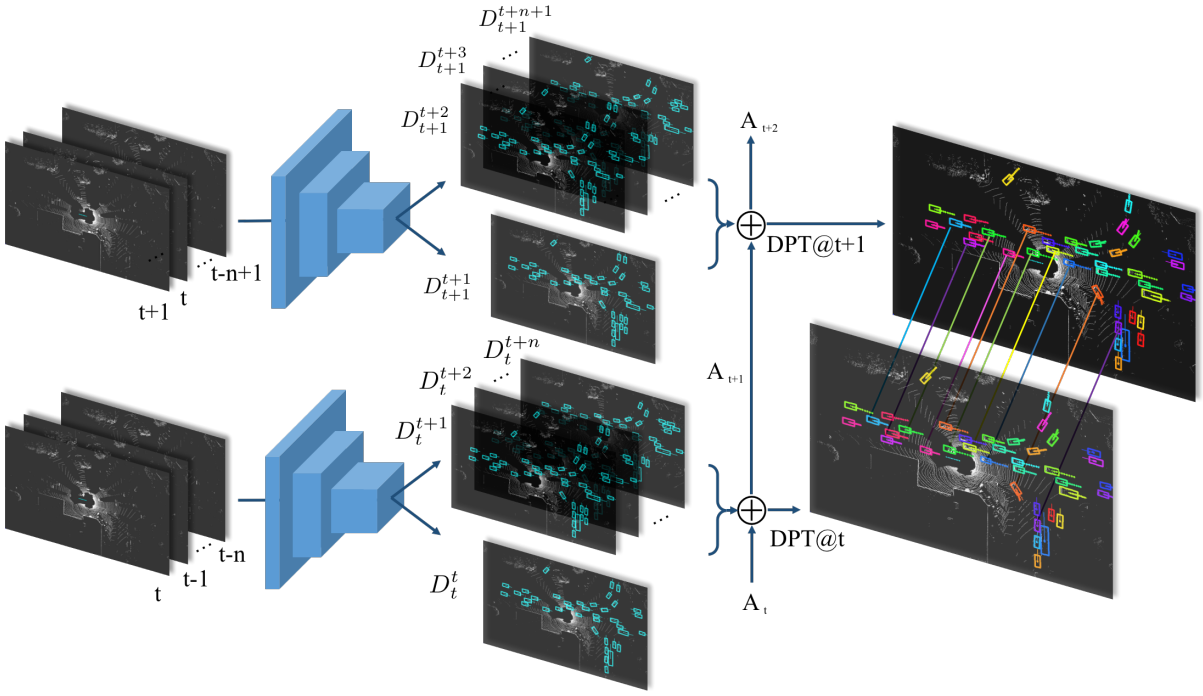
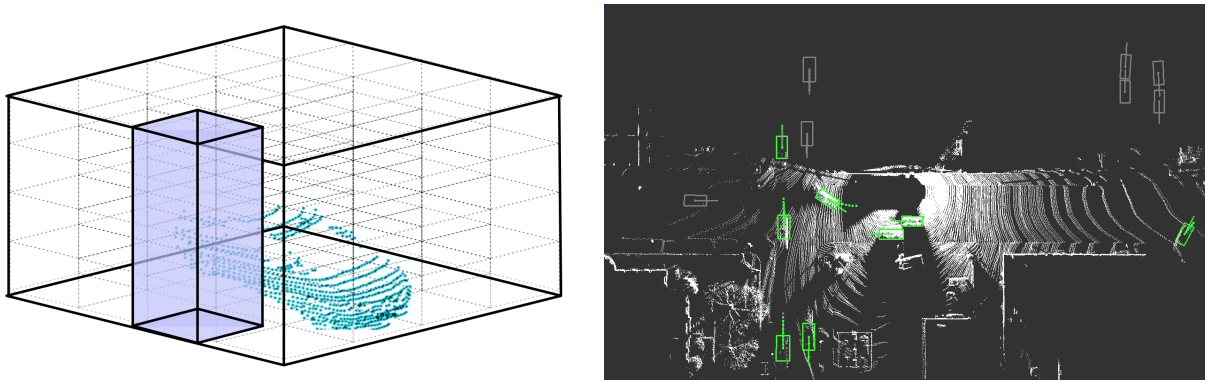


Figure 5.3: **Overview of our approach:** Our FaF network takes multiple frames as input and performs detection, tracking and motion forecasting.

3D Object Detection: In robotics applications such as autonomous driving, we are interested in detecting objects in 3D space. The ideas behind modern 2D image detectors can be transferred to 3D object detection in various ways. Chen *et al.*[27] used stereo images to perform 3D detection. Li [100] used 3D point cloud data and proposed to use 3D convolutions on a voxelized representation of point clouds. Chen *et al.*[28] combined image and 3D point clouds with a fusion network. They exploited 2D convolutions in BEV; however, they used hand-crafted height features as input. They achieved promising results on KITTI [55] but only ran at 360ms per frame due to heavy feature computation on both 3D point clouds and images. This is very slow, particularly if we are interested in extending these techniques to handle temporal data, which will be more problematic when including temporal data.

Object Tracking: Detection by itself is not enough for Self-driving; we also need to track objects over time. Over the past few decades, many approaches have been developed for object tracking. In this section, we briefly review the use of deep learning methods in tracking. In [115], pretrained convolutional neural networks were used to extract features and perform tracking with correlation, where similarly in [170, 71], regression is used. In contrast, Wang and Yeung [171] used an autoencoder to learn a good feature representation that helps tracking. Tao *et al.*[163] used siamese matching networks to perform tracking. Nam and Han [128] fine-tuned a convolutional neural network at inference time to track object within the same video.

Motion Forecasting: In addition to tracking, motion forecasting looks at the problem of predicting where each object will be in the future given multiple past frames. Lee *et al.*[98] proposed to use recurrent networks for long term prediction. Alahi *et al.*[4] used LSTMs to model the interaction between pedestrian and perform prediction accordingly. Ma *et al.*[116] proposed to utilize concepts from game theory to model the interaction between



(a) (b)

Figure 5.4: a **Voxel Representation:** treat height dimension directly as input feature dimension. b **Sample Data:** Overlaid temporal & motion forecasting data. *Green* represents ground truth bbox w/ 3D point. *Grey* represents ground truth bbox w/o 3D point.

pedestrian while predicting future trajectories. Other work has also focussed on short term prediction of dynamic objects [58, 135]. [169] performed prediction for dense pixel-wise short-term trajectories using variational autoencoders. [158, 119] focused on predicting the next future frames given a video, without explicitly reasoning about per-pixel motion.

Multi-task Approaches: Feichtenhofer *et al.*[50] proposed to do detection and tracking jointly from videoes. They model the displacement of corresponding objects between two input images during training and decode them into object tubes during inference time.

5.2 Joint 3D Detection, Tracking, and Motion Forecasting

Different from all the above related work, in this chapter we propose a single network that takes advantage of temporal information and tackles the problem of 3D detection, tracking, and short term motion forecasting in the scenario of self driving. Our input representation is a 4D tensor encoding an occupancy grid of the 3D space over several time frames. We exploit 3D convolutions over space and time to produce fast and accurate predictions. As point cloud data is inherently sparse in 3D space, our approach saves lots of computation as compared to doing 4D convolutions over 3D space and time. We demonstrate the effectiveness of our model on a large-scale dataset captured from multiple vehicles driving in North-America and show that our approach significantly outperforms the state-of-the-art. We name our approach Fast and Furious (FaF), as it can create very accurate estimates in as little as 30 ms.

In the following, we first describe our data parameterization in Sec. 5.2.1 including voxelization and how we incorporate temporal information. In Sec. 5.2.2, we present our model's architecture followed by the objective we use for training the network (Sec. 5.2.3).

5.2.1 Data Parameterization

In this section, we first describe our single frame representation of the surrounding world. We then extend our representation to exploit multiple frames.

Voxel Representation: In contrast to image detection where the input is a dense RGB image, point cloud data is inherently sparse and provides geometric information about the 3D scene. To get a representation where convolutions can be easily applied, we quantize the 3D world to form a 3D voxel grid. We then assign a binary indicator for each voxel encoding whether the voxel is occupied. We say a voxel is occupied if there exists at least one LiDAR point in the voxel's 3D space. As the grid is a regular lattice, convolutions can be directly used. We do not utilize 3D convolutions on our single frame representation as this operation will waste most computation since the grid is very sparse, *i.e.*, most of the voxels are not occupied. Instead, we performed 2D convolutions and treated the height dimension as the channel dimension. This allows the network to learn to extract information in the height dimension. This contrast approaches such as MV3D [28], which perform quantization on the x-y plane and generate a representation of the z-dimension by computing hand-crafted height statistics. Note that if our grid's resolution is high, our approach is equivalent to applying convolution on every single point without losing any information. We refer the reader to Fig. 5.4a for an illustration of how we construct the 3D tensor from 3D point cloud data.

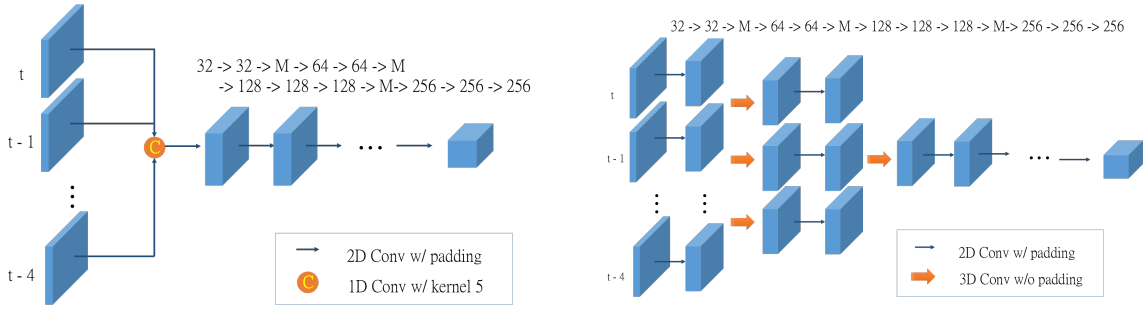
Adding Temporal Information: In order to perform motion forecasting, it is crucial to consider temporal information. Towards this goal, we take all the 3D points from the past n frames and perform a change of coordinates to represent them in the current vehicle coordinate system. This is important in order to undo the ego-motion of the vehicle where the sensor is mounted. This transformation can be easily performed using the transformation matrix. In this work, we assume the transformation matrix is given. In practice, this is usually computed by information from GPS/IMU etc with proper localization algorithms. After performing this transformation, we compute the voxel representation for each frame. Now that each frame is represented as a 3D tensor, we can append multiple frames' along a new temporal dimension to create a 4D tensor. This not only provides more 3D points as a whole but also gives cues about the vehicle's heading and velocity, enabling us to do motion forecasting. As shown in Fig. 5.4b, we overlay multiple frames for visualization purposes. As we can see, static objects are well aligned with denser LiDAR points while dynamic objects have 'shadows' which represents their motion.

5.2.2 Model Formulation

We have seen exciting results on image detection from Faster-RCNN [140], R-FCN [38], Mask-RCNN [66], YOLO [138], SSD [107], where they shown carefully designed CNNs can extract the features to predict both bounding box coordinates and semantic information. In this work, we follow the single-stage detector approach and tailor it for detection using point cloud data.

Our single-stage detector takes a 4D input tensor and regresses directly to object bounding boxes at different timestamps without using region proposals. We investigate two different ways to exploit the temporal dimension on our 4D tensor: early fusion and late fusion. They represent a tradeoff between accuracy and efficiency, and they differ on at which level the temporal dimension is aggregated. We will introduce the detail in the following.

Early Fusion: Our first approach aggregates temporal information at the very first layer. As a consequence, it runs as fast as using a single frame detector. However, it might lack the ability to capture complex temporal features as this is equivalent to producing a single point cloud from all frames, but weighing the contribution of the different timestamps differently. In particular, as shown in Fig. 5.5, given a 4D input tensor, we first use a 1D convolution with a kernel size n on temporal dimension to reduce the temporal dimension from n to 1. We share the weights among all feature maps, *i.e.*, also known as group convolution. We then perform convolution



(a) Early fusion

(b) Later fusion

Figure 5.5: We propose two ways for modeling temporal information.

and max-pooling following VGG16 [155] with each layer number of feature maps reduced by half. Note that we remove the last convolution group in VGG16, resulting in only ten convolution layers.

Late Fusion: In this case, we gradually merge the temporal information. This allows the model to capture high-level motion features. We use the same number of convolution layers and feature maps as in the early fusion model, but instead perform 3D convolution with a kernel size $3 \times 3 \times 3$ for 2 layers without padding on temporal dimension, which reduces the temporal dimension from n to 1, and then perform 2D spatial convolution with a kernel size 3×3 for other layers. We refer the reader to Fig. 5.5 for an illustration of our architecture.

Both early and late fusion models produce feature maps with width and height $\frac{1}{8}$ to the original input size and a feature vector of dimension 256. We then add two branches of convolution layers, as shown in Fig. 5.6. The first one performs binary classification to predict the probability of each bounding box being a vehicle. The second one predicts the bounding box over the current frame as well as $n-1$ frames into the future. Motion forecasting is possible as our approach exploits multiple frames as input, and thus can learn to estimate useful features such as velocity and acceleration.

Following SSD [107], we use multiple predefined boxes for each feature map location. As we utilize a BEV representation, our network can exploit priors about physical sizes of objects. Here we use boxes corresponding to 5 meters in the real world with the aspect ratio of $1:1, 1:2, 2:1, 1:6, 6:1$ and 8 meters with an aspect ratio of $1:1$. In total, there are 6 predefined boxes per feature map location denoted as $a^k_{i,j}$ where $i = 1, ..., I, j = 1, ..., J$ is the location in the feature map and $k = 1, ..., K$ ranges over the predefined boxes (*i.e.*, size and aspect ratio). Using multiple predefined boxes allows us to reduce the variance of regression target, thus makes the network easy to train. Notice that we do not use predefined heading angles. Furthermore, we use both $\sin$ and $\cos$ values to avoid the 180 degrees ambiguity.

In particular, for each predefined box $a^k_{i,j}$, our network predicts the corresponding normalized location offset $\hat{l_x}, \hat{l_y}$, log-normalized sizes $\hat{s_w}, \hat{s_h}$ and heading parameters $\hat{a}_{\sin}, \hat{a}_{\cos}$.

We build our model based on the intuition that so long as the feature vector encodes temporal information with big enough receptive field, it could also capture objects' velocity, acceleration, etc., capable of performing short term prediction. Thus, we make the bounding box regression branch predict multiple frames into the future. This can also give a clue about whether a car is static or not, which is crucial for motion planner in the downstream for developing the self-driving vehicle.

Decoding Tracklets: At each timestamp, our model outputs the detection bounding boxes for n timestamps. Reversely, each timestamp will have current detections as well as $n-1$ past predictions. Thus we can aggregate the

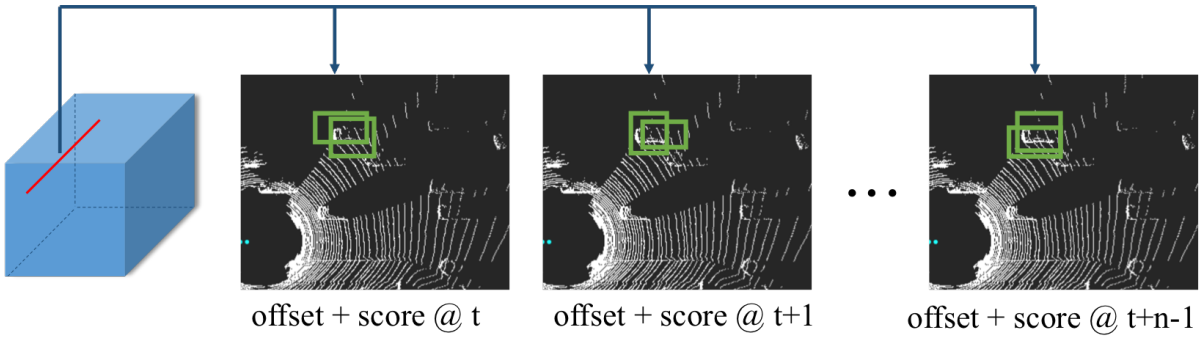


Figure 5.6: **Motion forecasting:** each feature vector is used to predict the bounding box offset and detection score for current time step t as well as future time step t+1, ..., t+n-1.

information for the past to produce accurate tracklets without solving any trajectory-based optimization problem. Note that if detection and motion forecasting are perfect, we can decode perfect tracklets. In practice, we use the average as the aggregation function. When there is overlap between detections from current and future predictions from the past, they are considered to be the same object, and their bounding boxes will be averaged. Intuitively, the aggregation process helps particularly when we have strong past predictions but no current evidence, *e.g.*, if the object is currently occluded or a false negative from detection. This allows us to track through occlusions over multiple frames. On the other hand, when we have strong current evidence but no prediction from the past, then there is evidence for a new object.

5.2.3 Loss Function and Training

We train the network to minimize a combination of classification and regression loss. In the case of regression we include both the current frame as well as our n frames forecasting into the future. That is

$$\ell(w) = \sum \left(\alpha \cdot \ell_{\text{cla}}(w) + \sum_{i=t,t+1,\ldots,t+n} \ell^t_{\text{reg}}(w) \right) \tag{5.1}$$

where t is the current frame and w represents the model parameters.

We employ as classification loss binary cross-entropy computed over all locations and predefined boxes:

$$\ell_{\text{cla}}(w) = \sum_{i,j,k} q_{i,j,k} \log p_{i,j,k}(w) \tag{5.2}$$

Here i, j, k are the indices on feature map locations and predefined box identity, $q_{i,j,k}$ is the class label (*i.e.* $q_{i,j,k}$ =1 for vehicle and 0 for background) and $p_{i,j,k}$ is the predicted probability for vehicle.

To define the regression loss for our detections and future predictions, we first need to find their associated ground truth. We defined their correspondence by matching each predefined box against all ground truth boxes. In particular, for each predicted box, we first find the ground truth box with the biggest overlap in terms of intersection over union (IoU). If the IoU is bigger than a fixed threshold (0.4 in practice), we assign this ground truth box as $\bar{a}^k_{i,j}$ and assign 1 to its corresponding label $q_{i,j,k}$. Following SSD [107], if there exists a ground truth box not assigned to any predefined box, we will assign it to its highest overlapping predefined box ignoring the fixed threshold. Note that multiple predefined boxes can be associated to the same ground truth, and some

predefined boxes might not have any correspondent ground truth box, meaning their $q_{i,j,k} = 0$.

Thus we define the regression targets as

$$l_x = \frac{x - x^{GT}}{w^{GT}} \qquad l_y = \frac{y - y^{GT}}{h^{GT}}$$
$$s_w = \log \frac{w}{w^{GT}} \qquad s_h = \log \frac{h}{h^{GT}}$$
$$a_{sin} = \sin(\theta^{GT}) \qquad a_{cos} = \cos(\theta^{GT})$$

We use a weighted smooth L1 loss over all regression targets where smooth L1 is defined as:

$$\text{smooth}_{L_1}(\hat{x}, x) = \begin{cases} \frac{1}{2}(\hat{x} - x)^2 & \text{if } |\hat{x} - x| < 1 \\ |\hat{x} - x| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{5.3}$$

Hard Data Mining Due to the imbalance of positive and negative samples, we use hard negative mining during training. We define positive samples as those predefined boxes having corresponding ground truth box, *i.e.*, $q_{i,j,k} = 1$. For negative samples, we rank all candidates by their predicted score $p_{i,j,k}$ from the classification branch and take the top negative samples with a ration of 3 in practice.

5.3 Experiments

In this section, we will evaluate our algorithm on real-world data for all tasks. In order to do that, we would need a large scale dataset that provides sequential LiDAR point cloud data, accurate ego-motion, as well as ground truth labels across time for both detection and tracking. Unfortunately, there is no such publicly available dataset that can evaluate 3D detection, tracking, and motion forecasting together. The biggest public autonomous driving dataset currently is KITTI [55]; however, KITTI detection benchmark does not provide 3D point cloud data with temporal information, and tracking benchmark is only on 2D image. Thus we collected a more comprehensive dataset that provides accurate temporal information. It is two orders of magnitude bigger in scale involving more dynamic scenes and also allows us to evaluate detection, tracking and motion forecasting at the same time.

5.3.1 Experiment Setup

Dataset: Our dataset is collected by a roof-mounted LiDAR on top of a vehicle driving around several North-American cities over different time in the day and different seasons in the year. It consists of 546,658 frames collected from 2762 different scenes. Each scene consists of a continuous sequence with an average length of 20 seconds. Our validation set consists of 5,000 frames collected from 100 scenes, *i.e.*, 50 continuous frames are taken from each sequence. There is no overlap between the geographic area where the training and validation are collected in order to showcase strong generalization. Our labels might contain vehicles with no 3D point on them as the labelers have access to the full sequence in order to provide accurate annotations. Our labels contain 3D rotated bounding box as well as track id for each vehicle, allowing to evaluate tracking on the whole sequence. Notice the dataset only provides 3D point cloud data for each frame without the ground information, *i.e.*, each LiDAR point's height is necessarily positive as the reference height 0 is from the first frame. For example, if the car is driving downhill, then the LiDAR point received later will have a negative height as the reference height 0 location is up on the hill. This makes the detection more challenging as it requires the algorithm to be more robust

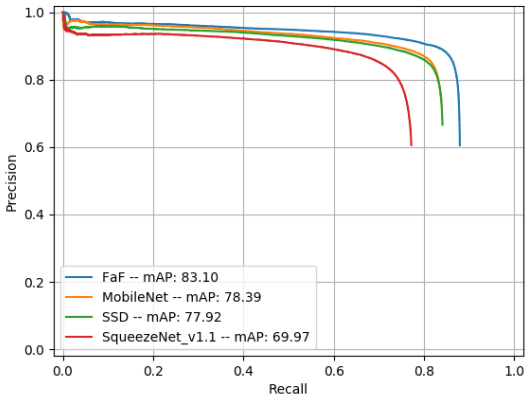


Figure 5.7: **P/R curve**

IoU	0.5	0.6	0.7	0.8	0.9	Time [ms]
SqueezeNet_v1.1 [81]	85.80	81.06	69.97	43.20	3.70	9
SSD [107]	90.23	86.76	77.92	52.39	5.87	23
MobileNet [76]	90.56	87.05	78.39	52.10	5.64	65
FaF	**93.24**	**90.54**	**83.10**	**61.61**	**11.83**	30

Table 5.1: Detection performance on 144×80 meters region, with object having ≥ 3 number 3D points.

to the change on height dimension. Although the dataset contains label for different dynamic objects including *vehicles, pedestrian, cyclists etc*, in this work, we focus on *vehicles* only, which includes car, van, and bus.

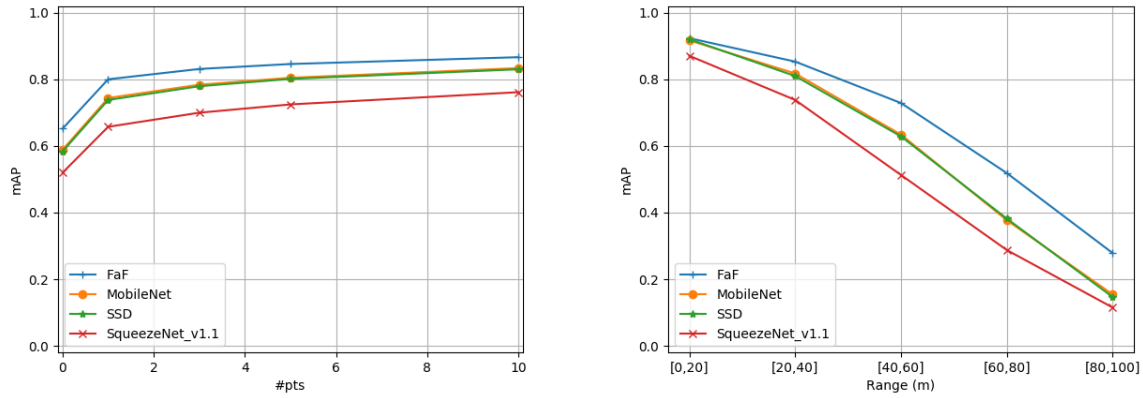
Training Setup: At training time, we use a spatial X-Y region of size 144×80 meters, where each grid cell is 0.2×0.2 meters. On the height dimension, we take the range from -2 to 3.5 meters with a 0.2-meter interval, leading to 29 bins. Notice we use -2 as bottom height as the dataset does not provide ground height information. This gives us a tensor of $29x400x720$ for one frame of LiDAR point cloud data. For temporal information, we take all the 3D points from the past five timestamps. Thus our input is a four-dimensional tensor consisting of time, height, X, and Y.

For both our early-fusion and late-fusion models, we train from scratch using Adam optimizer [89] with a learning rate of 1e-4. The model is trained on a 4 Titan XP GPU server with a batch size of 12. We train the model for 100K iteration with learning rate halved at 60K and 80K iterations respectively. Also we use $\alpha = 2$ in Eq. 5.1 to re-weight loss from box size and heading, and $\gamma = 3$ as the ratio of negative-positive samples during hard data mining.

5.3.2 Quantitative Results

Here we provide detailed quantitative results for 3D detection, tracking, and motion forecasting.

Detection Results: We compare our model against state-of-the-art real-time detectors including SSD [107], MobileNet [76] and SqueezeNet [81]. Note that these detectors are all developed to work on 2D detection from RGB images. To make them competitive, we also build our predefined boxes into their system, which further easy



(a) (b)

Figure 5.8: a: mAP on different number of minimum 3D points. b: mAP over distance.

the task for those detectors. The region of interest is $144 \times 80M$ centered at ego-car during inference time. We keep the same voxelization for all models and evaluate detections against the ground truth vehicle bounding boxes with a minimum of three 3D points. Vehicles with less than three points are considered don't care regions. We consider a detection correct if it has an IoU against any ground-truth vehicle bounding box larger than 0.7. Note that for a vehicle with a typical size of 3.5×6 meters, 0.7 IoU means we can have at most miss 0.35 meters along the width and 0.6 meters along the length. Fig. 5.7 shows the precision recall curve for all approaches, where our model can achieve better performance, especially higher recall at the same precision level, which is crucial for autonomous driving.

Furthermore, Tab. 5.1 shows mAP numbers using different IoU thresholds for all detectors. We can see that our method can outperform all other methods. Particularly at IoU 0.7, we achieve 4.7% higher mAP than MobileNet [76] while being twice faster, and 5.2% better than SSD [107] with similar running time. This direct comparison shows the effectiveness of our approach that using temporal information and learning jointly can achieve much better performance.

Also, in order to have a more comprehensive understanding of the performance of different approaches, we report mAP performance as a function of the minimum number of 3D points, which is used to filter ground truth bounding boxes during test time. This provides specific information on how the model performances on objects with sparse LiDAR points. Note that a high level of sparsity is due to occlusion or long distance vehicles. As shown in Fig. 5.8a, our method can outperform other methods at all levels. We also evaluate with a minimum of 0 point, to show the importance of exploiting temporal information.

Furthermore, we are also interested in knowing how the model performs as a function of vehicle distance. Towards this goal, we extend the predictions to be as far as 100 meters away. Fig. 5.8b shows the mAP with IoU 0.7 on vehicles within different distance ranges. We can see that all methods are doing well on nearby vehicles, while our method is significantly better at long range. Note that all methods perform poorly at 100 meters due to lack of 3D points at that distance.

Ablation Study: We conducted ablation experiments within our framework to show how important each of the components is. We fixed the training setup for all experiments. As shown in Tab. 5.2, using temporal information with early fusion gives 3.7% improvement on mAP at IoU 0.7 compared to a model trained with single frame

5 Frames			Joint		IoU Threshold					Runtime
Single	Early	Later	Forecasting	Tracking	0.5	0.6	0.7	0.8	0.9	[ms]
✓					89.81	86.27	77.20	52.28	6.33	9
	✓				91.49	88.57	80.90	57.14	8.17	11
		✓			92.01	89.37	82.33	58.77	8.93	29
		✓	✓		92.02	89.34	81.55	58.61	9.62	30
		✓	✓	✓	**93.24**	**90.54**	**83.10**	**61.61**	**11.83**	30

Table 5.2: Ablation study, on 144×80 region with vehicles having $\geq$3 number 3D points.

	MOTA	MOTP	MT	ML
FaF	**80.9**	85.3	**75**	**10.6**
Hungarian	73.1	**85.4**	55.4	20.8

Table 5.3: Tracking performance

input. While later fusion uses the same information as early fusion, it can get 1.4% extra improvement as it can model more complex temporal features. Besides, adding prediction loss gives similar detection results on the current frame alone, however it enables us to decode tracklets and provides evidence to output smoother detections, thus giving the best performance, *i.e.* 6% points better on mAP at IoU 0.7 than single frame detector.

Tracking: Our model can output detections with track ids directly. In this part, we evaluate the raw tracking output without adding any sophisticated tracking pipeline on top. Tab. 5.3 shows the comparison between our model's output and a Hungarian method on top of our detection results. We follow the KITTI protocol [55] and compute MOTA, MOTP, Mostly-Tracked (MT) and Mostly-Lost (ML) across all 100 validation sequences. The evaluation script uses IoU 0.5 for the association and score of 0.9 for thresholding both methods. We can see that our final output achieves 80.9% in MOTA, 7.8% better than Hungarian, as well as 20% better on MT, 10% lower on ML, while still having similar MOTP.

Motion Forecasting: We evaluate the forecasting ability of the model by computing the average L1 and L2 distances of the vehicles' center location. As shown in Fig. 5.9, we can predict ten frames into the future with L2 distance only less than 0.33 meter. Note that due to the nature of the problem, we can only evaluate on true positives, which in our case has a corresponding recall of 92.5% on all objects. Also, as we can tell, the L1/L2 error grows exponentially over time. This shows the inherent difficulty of the motion forecasting problem, as long term forecasting can be multi-modality while our approach only gives uni-model. It might also require more information such as a high-definition map to better prediction where each vehicle would go into the future.

5.3.3 Qualitative Results

In this section, we show the visualization of different sets of samples from the validation set. As shown in Fig. 5.10, we use the top-down view image over a 144×80 meters region. We draw LiDAR point cloud in white and all vehicles in color. Each vehicle has its unique color, and it is consistent over time, showing the effect of tracking. We use waypoints for each vehicle to represent their future location as motion forecasting. In our work, we prediction ten waypoints for each vehicle, corresponding to 1 second into the future. By looking at the waypoints, we can tell the speed of each vehicle. For example, the last row in Fig. 5.10 gives an example where predicted waypoints are far away from each other, showing the high speed of these vehicles. On the contrary,

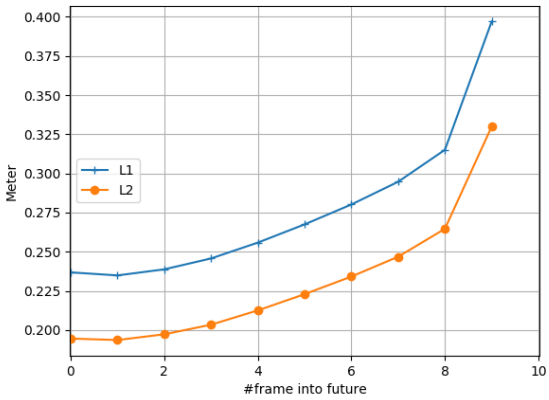


Figure 5.9: Motion forecasting performance

in the first row of Fig. 5.10, we can see lots of vehicles with predicted waypoints overlap at the same location, meaning they are static vehicles. In total, we provide 5 sequences in Fig. 5.10, including heavy traffic, complex scenes and fast-changing scenarios, etc. We can see that our algorithm can give accurate bounding boxes for vehicles with a good heading estimate. Furthermore, the consistent color across different time shows that our algorithm can track all vehicles at the same time, even with very heavy traffic such as the second row in Fig. 5.10.

On the other hand, we also provide a set of failure cases in Fig. 5.11 to better understand the limitations of our algorithm, as well as inspire future work. The first row in Fig. 5.11 gives an example where the predicted waypoints are not consistent (check the green car in the middle, turning right). This is because the waypoints are independently predicted in our algorithm. Thus, there is no guarantee that the predicted waypoints are physically feasible, giving a weird trajectory in this case. The second example shows a failure case where the detection is incorrect. Check the red car in the middle whose rotation is predicted wrong. This results in inconsistent detection/prediction results across time and can be very confusing for the whole self-driving system. This is because LiDAR points are sparse, being occluded in this case. The third example shows a failure case on turning. Check the green car in the middle that is turning left. We also draw the ground truth waypoints in white. As we can see, our predicted waypoints do not perform a good turn, *i.e.* not as sharp as it should be. This reveals the fact that making prediction only based on the motion is difficult. We should exploit more input, such as the HD map, to better regularize where each vehicle can drive.

5.4 Discussion

The traditional self-driving solution includes different models/components to address the sub-problems of 3D detection, tracking, and motion forecasting. It is indeed a good solution in practice as it allows a big organization to utilize divide and conquer method. They can split teams to focus on each of these sub-tasks and progress at the same time. However, the limitation is that massive engineering effort is required to hook up these models/components. We need to tune carefully to make sure the output of the upstream modules is compatible and optimized for the input of downstream modules. It is also not efficient to iterate the system to achieve better performance since a small change in upstream modules would require fine-tuning the whole system again. As a result, often, when people upgrade the upstream module, they do not necessarily see a direct improvement of the

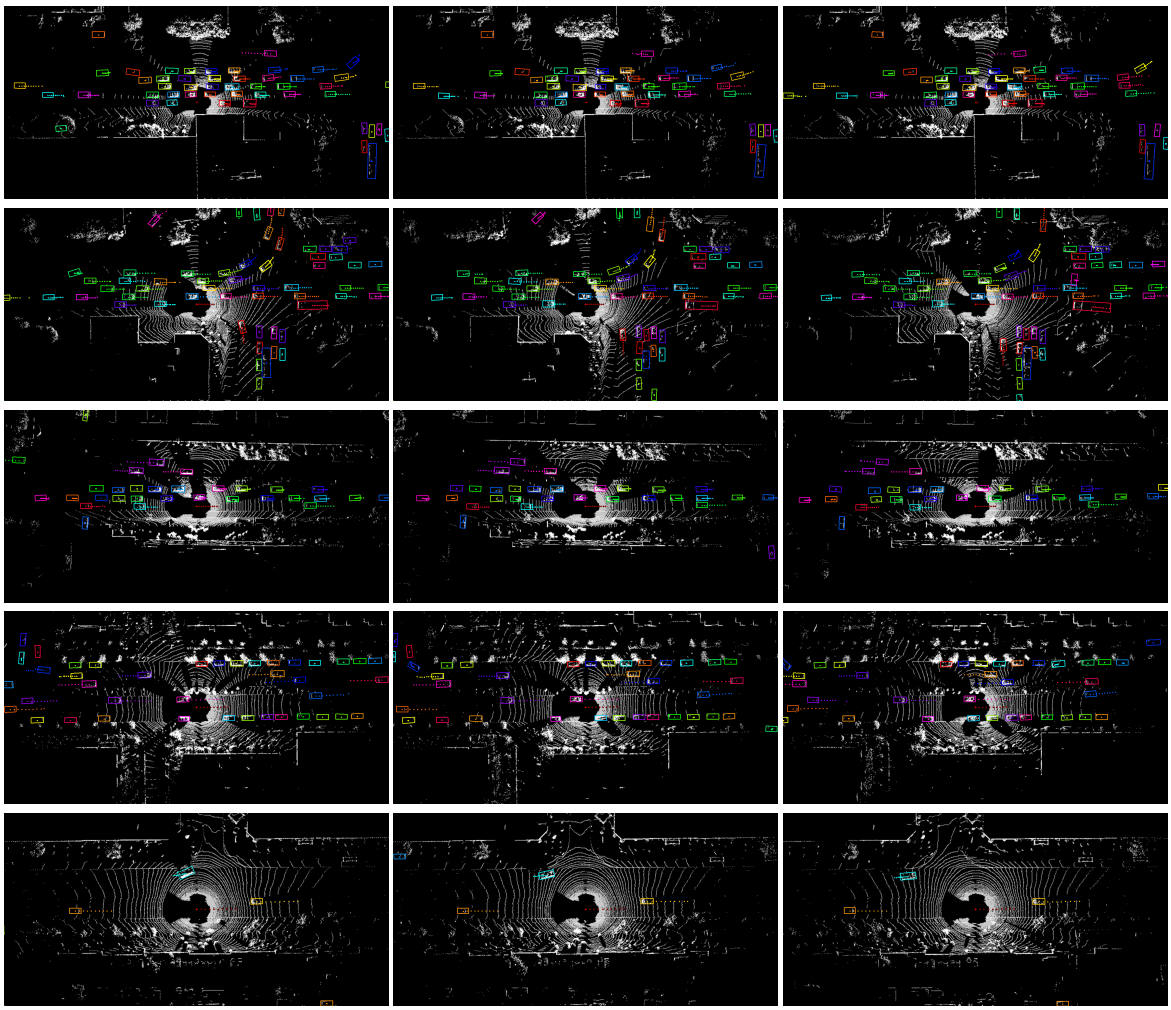


Figure 5.10: **Visualization:** Each row represents one sequence. Each figure the top-down view of our model output, including detection (represented by oriented bounding boxes), tracking (represented by color) and motion forecasting (represented by waypoints).

entire system, making it challenging to integrate real improvement to the full stack.

On the contrary, we have proposed the holistic model that reasons jointly about 3D detection, motion forecasting, and tracking. We exploit the temporal information and provide two ways of incorporating temporal information, *i.e.*, early-fusion and later-fusion that exploit the trade-off between runtime and accuracy. Using large scale real-world driving data, we demonstrate that our model performs better on all tasks compared to baselines. This is because we can train the model jointly with temporal information, as well as optimize it jointly with all tasks. More importantly, as we use one model for different tasks, the heavy feature computation is shared among all tasks, making our algorithm very efficient in terms of computation. In practice, we can achieve real-time performance, running at 30Hz, without substantial code optimization.

Despite the promising results we have seen, there are also certain limitations to our approach. First of all, a real-world self-driving vehicle would need to understand the movement of all surrounding objects, not only limited to vehicles. We need to extend this work to pedestrians, cyclists, etc. Secondly, the information our model uses to make motion forecasting is only from objects' history motion. This, however, could be problematic in

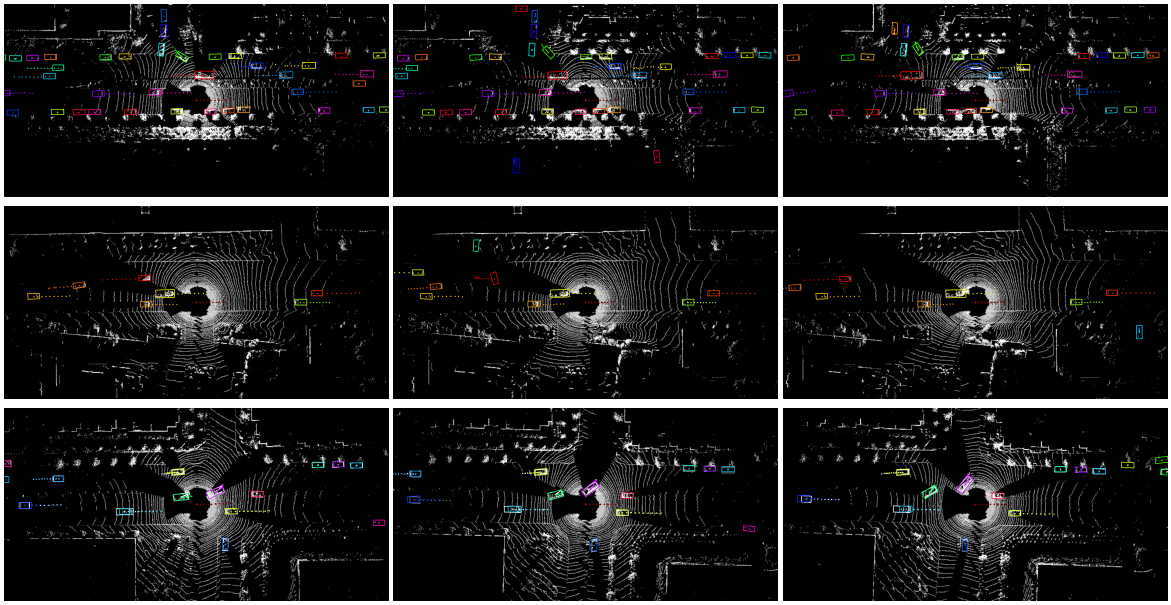


Figure 5.11: **Failure Cases:** Each row represents one sequence. Each figure the top-down view of our model output, including detection (represented by oriented bounding boxes), tracking (represented by color) and motion forecasting (represented by waypoints).

many cases. For example, when the vehicle is approaching a *STOP* sign, our model would not predict it to stop as we do not have traffic sign and traffic lights as input. Another example is when the vehicle is approaching an intersection; our model would not predict it making a turn unless the vehicle starts making the turn (*i.e.*, generating the motion of turning). To fix these issues, we need to input an HD map as well as the dynamic element of the map (*i.e.*, traffic lights, etc.) to our model. In fact, [19] did the work along this direction. They also took it one step further to predict high-level actions that can better help motion planner understand the surrounding environment. Bansal et al. [10] and Chou et al. [33] also utilized temporal information to help predict motion using a single neural network.

Chapter 6

Neural Interpretable Planner

In the previous chapters, we developed methods based on deep learning to help self-driving vehicles understand the surrounding environment. In particular, chapter 3 was about estimating the depth of the scene using stereo cameras. In chapter 4, we extended the deep matching method to flow estimation. Moreover, a joint model for detection, tracking, and motion forecasting was introduced in chapter 5. It unified different components in traditional autonomy software systems to achieve better results. While previous chapters focus on improving the performance of important tasks such as perception and prediction, we also need to examine the impact of these improvements on the end task, *i.e.* self-driving capability. This involves optimizing the usage of the output of the perception and prediction models for the downstream modules. In other words, how can we transform the understanding of the world into something more suitable for downstream modules?

In traditional self-driving autonomy systems, a *cost map* would be constructed to represent the good region for a self-driving vehicle to drive through. This *cost map* is manually designed to incorporate human knowledge and handcrafted rules to make sure that a self-driving vehicle behaves as expected. More explicitly, the cost map is required to encode the information about the surrounding environment. For example, it takes as input the information of the present and predicted future locations of both dynamic and static objects. If a detection model detects a vehicle in front of the ego car, there should be a much higher cost at the detected location, implying that the self-driving vehicle should not be driving there. This can also be interpreted as the penalty of driving to a specific location at a specific time step. In the context of reinforcement learning, people have been using value function to represent the same idea, *i.e.*, each state has a value that indicates how good or bad it is if the agent arrived there.

Although we can manually design the cost map based on the results from perception and prediction, the performance will be bounded by the perception and prediction models as information can only be passed through specific and limited forms, *e.g.* detection bounding boxes. To be more specific, we have various types of sensors giving comprehensive information represented as $\mathbf{A}$ as the input to our system. The perception and prediction modules can be treated as encoders that transform the information $\mathbf{A}$ into another form understandable by humans represented as $\mathbf{B}$, *e.g.* objects' bounding boxes and their future locations. Later, the planning module, which can be treated as a decoder, takes this information $\mathbf{B}$ to perform corresponding actions to drive safely toward the desired location. However, this approach is suboptimal, as the information captured by $\mathbf{B}$ is limited. Useful information such as uncertainty and unknown obstacles are ignored when we only use bounding boxes and predicted future locations from successful detections. Alternatively, we can use a deep learning model to learn the cost map instead of manually constructing it. The advantages include a reduced dependency on human domain knowledge,

ease of scaling and maintenance. It also removes the limitation on the flow of information introduced by the explicit and restrictive bounding box representation.

In this chapter, we propose a neural motion planner for self driving in complex urban scenarios, including perceiving traffic-light, yielding, and handling simple interactions with multiple road-users. We bridge the gap between perception, prediction, and planning modules by learning the cost map directly from sensor data. Towards this goal, we design a holistic model that takes as input raw LiDAR data and an HD map, and produces a space-time cost volume defining the goodness of each position that the self-driving car can occupy within the planning horizon. At the same time, our model gives interpretable intermediate representations in the form of 3D detections and their future trajectories. Our planner then samples a set of diverse, physically feasible trajectories and selects the one with the minimum learned cost for execution. Importantly, the non-parametric cost volume can capture the uncertainty and multi-modality in various possible SDV trajectories, *e.g.*, changing lane vs. keeping lane.

We demonstrate the effectiveness of our approach using a real-world driving dataset captured in several cities in North America. Our experiments show that our model provides good interpretable representations and better performance. For detection and motion forecasting, our model outperforms recent neural architectures specifically designed on these tasks. For motion planning, our model generates safer planning compared to the baselines.

6.1 Background

As is the case in many application domains, the field of autonomous driving has been transformed in the past few years by the success of deep learning. Existing approaches that leverage this technology can be divided into two main types: models that are trained in an end-to-end fashion and traditional engineering systems.

End-to-end driving approaches [136, 17] take the output of the sensors (*e.g.*, LiDAR, images) and use it as input to a neural network that outputs control signals, *e.g.*, steering command and acceleration. The main benefit of this framework is its simplicity, as only a few lines of code can build a model. As well, labeled training data can be easily obtained automatically by recording human driving using a self driving vehicle platform. In practice, this approach suffers from compounding errors due to the nature of self-driving control being a sequential decision problem, and the requirement of massive amounts of data to generalize. Furthermore, interpretability is difficult to obtain for analyzing the mistakes of the network. It is also hard to incorporate sophisticated prior knowledge about the scene, *e.g.*, vehicles should not collide with other actors and should drive on the road surface.

In contrast, most self-driving car companies, utilize a *traditional engineering system*, where the problem is divided into subtasks: perception, prediction, motion planning, and control. Perception is in charge of estimating all actors' position and motion given the current and past evidence. This involves solving tasks such as 3D object detection and tracking. Prediction, on the other hand, tackles the problem of estimating the future positions of all actors as well as their intentions (*e.g.*, changing lanes, parking). Finally, motion planning takes the output from previous modules and generates a safe trajectory for the SDV to execute. This framework has interpretable intermediate representations by construction, and prior knowledge can be easily exploited, for example, in the form of high definition maps (HD maps).

However, solving each of these subtasks is hard. Most self-driving companies have large engineering teams working on each sub-problem in isolation. As a consequence, an advance in one sub-system does not easily translate to an overall system performance improvement. In addition, uncertainty estimates are difficult to propagate, and computation is not shared among different modules. This leads to longer reaction times of the SDV and makes the overall system less reliable. Furthermore, each of the sub-systems is trained with a different task-specific objective, which is not directly related to the overall system performance. For example, 3D detection tries

to maximize AP, where each actor has the same weight. However, in a driving scenario, high-precision detection of actors that may influence the SDV motion, *e.g.*, through interaction (cutting in, sudden stopping), is more critical.

In the following, we first give a brief introduction about related work for motion planning in the domain of self-driving. Then, we introduce the basics about inverse reinforcement learning as well as deep structured models which form the fundamentals of our approach.

6.1.1 Related Work

Imitation Learning: Imitation learning (IL) uses expert demonstrations to learn a policy that maps states to actions directly. It is arguably able to bypass the problem of disconnection between different modules as information can flow through the whole module instead of only by the form of detected objects. Using IL for self-driving vehicles was introduced in the pioneering work of [136] where a direct mapping from the sensor data to steering angle and acceleration is learned. A similar philosophy has been followed since then, *e.g.*, in [17], but with much larger neural networks and massive training data recorded by manual driving. In contrast, with the help of a high-end driving simulator CARLA [43], Codevilla *et.al.* [35] exploit conditional models with additional high-level commands such as *continue*, *turn-left*, *turn-right*. Muller *et.al.* [127] incorporate reasoning about road segmentation as intermediate interpretable representations. During training, they use segmentation as supervision, and at inference time, steering commands are converted from learned segmentation mask from camera data, *i.e.* the segmentation is trained separately. In practice, IL approaches suffer from the compounding error due to the nature of self-driving control being a sequential decision problem. Furthermore, these approaches require a massive amount of data, and generalize poorly, *e.g.*, to situations drifting out of the lane. Also, another limitation is on interpretability. It is very hard to diagnose when IL fails as there is no intermediate output showing if the self-driving vehicle understands the surrounding world well. It is essentially a black-box to human and could not provide safety guarantee for critical usage such as self-driving vehicles.

RL & IRL: On the other hand, reinforcement learning (RL) is a natural fit for sequential decision problems as it considers the interaction between the environment and the agent (a self-driving car in this case). It is designed to learn from the interaction between an agent and the environment; thus it can avoid the compounding error issue as the agent can explore everywhere in the environment instead of in a fixed set of roads that human drives on. A good simulator is essential to do that. In particular, it needs to be able to simulate the worst case scenario. Following the success of Alpha GO [152], RL has been applied to self-driving in [133, 87]. In contrast to reinforcement learning, inverse reinforcement learning (IRL), as the name suggests, looks at the inverse problem of reinforcement learning, *i.e.* learning the reward function for a given task through human demonstration. [181, 206] develop IRL algorithms to learn drivable region for self-driving cars. [142] further infer possible trajectories with a symmetrical cross-entropy loss. However, all these approaches have only been tested on the simulated dataset or small real-world dataset, and it is unclear if RL and IRL can scale to more realistic settings. Furthermore, these methods do not produce interpretable representations, which are desirable in safety-critical applications.

Optimization Based Planners: Motion planning has long been treated as an independent task that uses the outputs of perception and prediction modules to formulate an optimization problem, usually by manually engineering a cost function [18, 125, 208, 49]. The preferred trajectory is then generated by minimizing this cost function. In practice, to simplify the optimization problem, many approaches assume the objective to be, for example, quadratic [22], decompose lateral and longitudinal planning as two tasks [49, 3] or represent the search space into

speed and path [85, 52]. In [3] A* is used to search the space of possible longitudinal and lateral motions in a traffic scene and find a low-cost, collision-free trajectory. Similarly, the *Baidu* motion planner [49] uses dynamic programming to find an approximate path and speed profile, followed by a quadratic programming optimization of a manually designed cost function. In [207], the trajectory planning problem is formulated as a continuous optimization and used in practice to demonstrate 100km of autonomous driving. In sampling-based approaches, a set of trajectories is generated and evaluated against a predefined cost, among which, the one with minimum cost is chosen [175, 148]. Such approaches are attractive since they are highly parallelizable [121]. The drawback of all these hand-engineered approaches is that they require perfect perception and prediction output, thus are not robust to real-world driving scenarios. Also, it is not easy to propagate uncertainty through different modules. In practice, tremendous engineering efforts are required to fine-tune the whole system for the real world.

Planning under uncertainty: Planning methods for robust and safe driving in the presence of uncertainty have also been explored [9, 197, 63]. Uncertainty in the intention of other actors is the main focus of [9, 197]. In [63], possible future actions of other vehicles and collision probability are used to account for the uncertainty in obstacles positions. Compared to these approaches, our planner naturally handles uncertainty by learning a non-parametric cost function.

Holistic Models: These models can provide interpretability while achieving great performance on perception, prediction. Chen *et.al.* [21] propose to learn a mapping from the sensor data to accordance, such as distance to left boundary/leading vehicle. This is then fed into a controller that generates steering command and acceleration. Sauer *et.al.* [145] further propose a conditional version of this model that takes direction command as additional input. Unfortunately, these methods are only demonstrated in simulation and still far away from real-world usage. On the other hand, Luo et al. [114] propose a joint model for perception and prediction from raw LiDAR data and [19] extends it to predict each vehicles' intention. All the methods above are trained for tasks that provide interpretable perception/prediction outputs to be used in motion planning. However, no feedback is back-propagated from the motion planning module.

6.2 Deep Structured Interpretable Planner

The end-to-end learnable motion planner model we proposed here generates accurate 3D trajectories over a planning horizon of a few seconds. Our model takes as input LiDAR point clouds and a high definition map (HD-map) and produces interpretable intermediate representations in the form of 3D detections as well as their future motion forecast over the planning horizon. Our final output representation is a space-time cost volume that represents the "goodness" of each possible location that the SDV can take within the planning horizon. The initial cost volume is computed solely via the feed-forward convolutional neural network. Then, our final plan can be estimated by sampling a set of physically plausible trajectories, evaluating them using our learned cost volume, and selecting the trajectory with the lowest cost.

Existing imitation learning approaches [136, 17, 35] directly regress the steering angle from raw sensor data, but do not generalize well and have difficulty capturing the multi-modality nature of the problem. In contrast, our approach can provide great interpretability through the form of detection and motion forecasting for each dynamic object as well as handling multimodality naturally (*e.g.*, changing lanes vs. keeping current lane). This is encoded in the cost volume, as it can give multiple low-cost regions. Traditional planners use manually designed cost functions using the outputs of perception and prediction systems. However, these individual components suffer

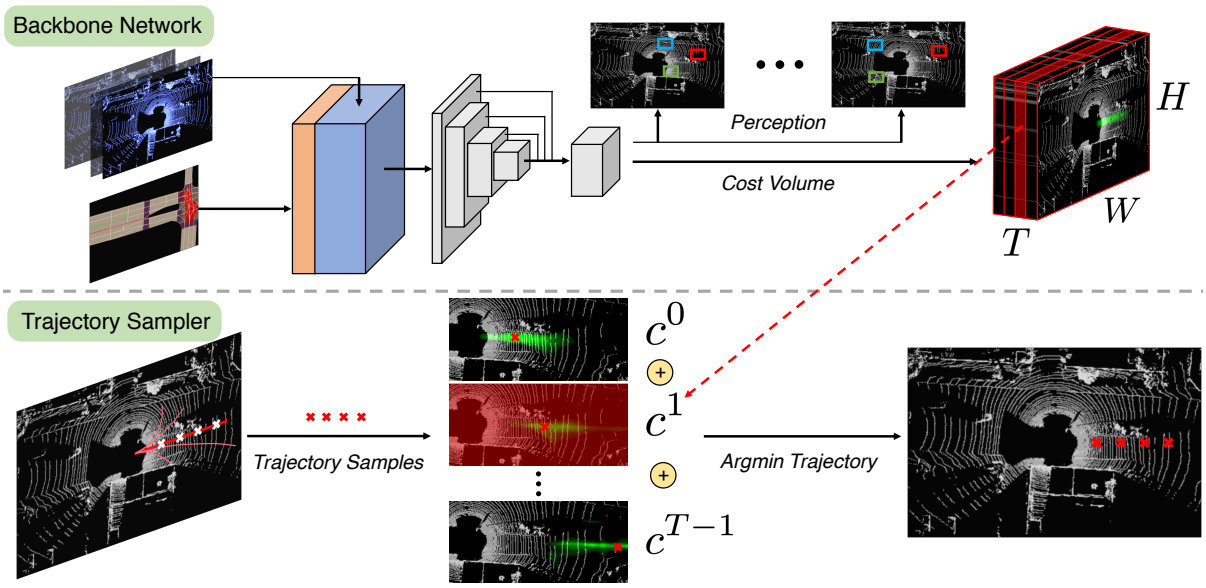


Figure 6.1: Our interpretable and end-to-end motion planner. Backbone network takes LiDAR data and maps as inputs, and outputs bounding boxes of other actors for future time steps (perception), as well as a cost volume for planning with T filters. Next, for each trajectory proposal from the sampler, its cost is indexed from different filters of the cost volume and summed together. The trajectory with the minimal cost will be our final planning.

from distribution mismatch between their inputs and outputs, as they are trained and tuned separately. In contrast, our approach is trained jointly and can learn better representations for the end task. Furthermore, our model can handle uncertainty naturally as this is represented in the cost. It does not require the costly parameter tuning when constructing the cost volume from perception/prediction outputs, and can learn concepts that are difficult to specify by hand, such as "slowing down when approaching occlusion".

We learn our model end-to-end with a multi-task objective. Our planning loss encourages the human-driven trajectories to have lower cost computed from our learned cost volume compared to other trajectories. Note that this loss is sparse, as a ground-truth trajectory only occupies a small portion of the space of all possible trajectories. As a consequence, learning with this loss alone is slow and difficult. To mitigate this problem, we introduce another perception loss that encourages the intermediate representations to produce accurate 3D detections and motion forecasting. This ensures the interpretability of the intermediate representations, and provides a direct explanation of the model's perception of the surroundings. On the other hand, it enables much faster learning.

6.2.1 Deep Structured Planning

More formally, let $\mathbf{s} = \{\mathbf{s}^0, \mathbf{s}^1, \cdots, \mathbf{s}^{T-1}\}$ be a trajectory spanning over T time steps into the future, with $\mathbf{s}^t$ the location in bird's eye view (BEV) at time step t. We formulate the planning problem as a deep structured minimization problem as follows

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \sum_t c^t(\mathbf{s}^t) \tag{6.1}$$

where c^t is our learned cost volume indexed at time step t, which is a 2D tensor with the same size as our region of interest. Notice in Eq. 6.1, $\mathbf{s}$ represents how the vehicle can move in the real world; thus, there are constraints on how the vehicles can move. For example, there is a limit on acceleration, deceleration, and the angle a vehicle can turn. In this work, the minimization problem in Eq. 6.1 is approximated by sampling a set of physically valid

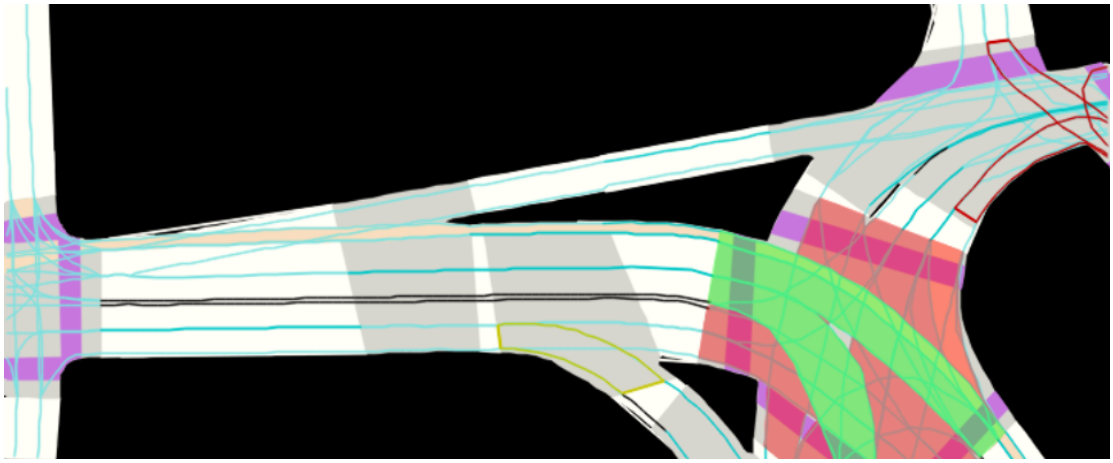


Figure 6.2: Visualization of rasterized map data. Different color represents different map component.

trajectories **s** and picking the one with minimum cost. Our base model employs a convolutional network backbone to compute this cost volume. It first extracts features from both LiDAR and maps and then feeds this feature map into two branches of convolution layers that output 3D detection and motion forecasting as well as the planning cost volume respectively. In this section, we describe our input representation and network in details.

Input representation: Our approach takes raw point clouds as inputs, captured by a LiDAR mounted on top of the SDV. We employ $T' = 10$ consecutive sweeps as observations; this can help the model capture the motion of all actors to make accurate predictions on where they are going. This information is crucial to plan a safe trajectory avoiding future collisions.

Each LiDAR sweep consists of an unstructured point cloud collected at different timestep, with around 100K points on average. For those sweeps, we first compensate ego-motion, *i.e.* project the point clouds from the past ten frames into the same coordinate system centered at SDV's current location. The transformation requires accurate ego-motion estimation, which is not in the scope of this section. Thus here we assume it is known and provided by the dataset. In practice, it can be estimated using GPS/IMU with standard localization algorithms. On the other hand, raw LiDAR points are sparse and characterized by continuous coordinates, preventing us from doing standard convolution operation directly. Thus to make the input point cloud data amenable to standard convolutions, we follow [19] and rasterize the space into a 3D occupancy grid, where each voxel has a binary value indicating whether it contains a LiDAR point. This results in a 3D tensor of size $H\text{x}W\text{x}(ZT')$, where Z, H, W represents the height and x-y spatial dimensions respectively. Note that we have concatenated different time steps along the Z dimension to avoid using 3D convolutions, which are memory and computation intensive.

Access to a map is also a key for accurate motion planning, as we need to drive according to traffic rules (*e.g.*, stop at a red light, follow the lane, change lanes only when allowed). Towards this goal, we exploit HD maps that contain information about the semantics of the scene such as the location of lanes, their boundary type (*e.g.*, solid, dashed) and the location of stop signs. Also, dynamic elements such as the status of a traffic light are crucial for real-world driving. Similar to [19], we rasterize the map to form a M channels tensor, where each channel represents a different map element, including road, intersections, lanes, lane boundaries, traffic lights, etc. In detail, we represent each semantic component in the map with a binary map (*i.e.*, 1 or -1). Roads and intersections are represented as filled polygons covering the whole drivable surface. Lane boundaries are parameterized as poly-lines representing the left and right boundaries of lane segments. Note that we use three binary masks to

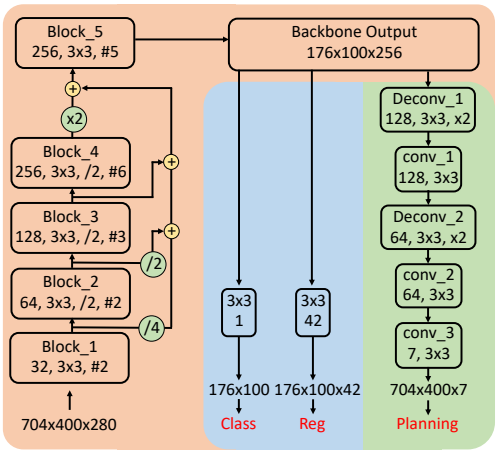


Figure 6.3: Details on backbone network.

distinguish lane types, as lane boundaries can be crossed or not, or only in certain situations. Lane surfaces are rasterized to differentiate between straight, left, and right turns, as this information is helpful for intention prediction. We also use two extra binary masks for bike and bus lanes as a way to input a prior of non-drivable road areas. Furthermore, traffic lights can change the drivable region dynamically. We encode the state of the traffic light into the lanes they govern. We rasterize the surface of the lane, succeeding the traffic light in one out of three binary masks depending on its state: green, yellow or red. One extra layer is used to indicate whether its governing traffic light protects those lanes, *i.e.*, cars in other lanes must yield. This situation happens in turns when the arrow section of the traffic light is illuminated. We estimate the traffic light states using cameras in our self-driving vehicle. We also infer the state of some unobserved traffic lights that directly interact with known traffic light states. For example, a straight lane with unknown traffic light state that collides with a protected turn with green traffic light state can be safely classified as being red. Lastly, traffic signs are also encoded into their governed lane segments, using two binary masks to distinguish between yields and stops. In total, there are 17 binary masks used as map features, resulting in a 3D tensor that represents the map. Fig. 6.2 shows an example, where different elements (*e.g.*, lane markers in cyan, crossings in magenta, alpha blended traffic lights with their state colored) are depicted.

Our final input tensor is thus 3D tensor of size $H\text{x}W\text{x}(ZT' + M)$.

Backbone: Our backbone is adapted from the detection network of [187] and consists of five blocks. Each block has $\{2, 2, 3, 6, 5\}$ *Conv2D* layers with filter number $\{32, 64, 128. 256, 256\}$, filter size 3x3 and stride 1. There are *MaxPool* layers after each of the first 3 blocks. A multi-scale feature map is generated after the first four blocks as follows. Similar to [202], we resize the feature maps from each of the first four blocks to 1/4 of the input sizes and concatenate them together to increase the effective receptive field [112]. These multi-scale features are then fed into the 5-th block. The whole backbone has a downsampling rate of 4. We refer the reader to Fig. 6.3 for more details.

Perception Header: The perception header has two components formed of convolution layers, one for classification and one for regression.

To reduce the variance of regression targets, we follow SSD [107] and employ multiple predefined anchor boxes $a^k_{i,j}$ at each feature map location, where subscript i, j denotes the location on the feature map and k indexes

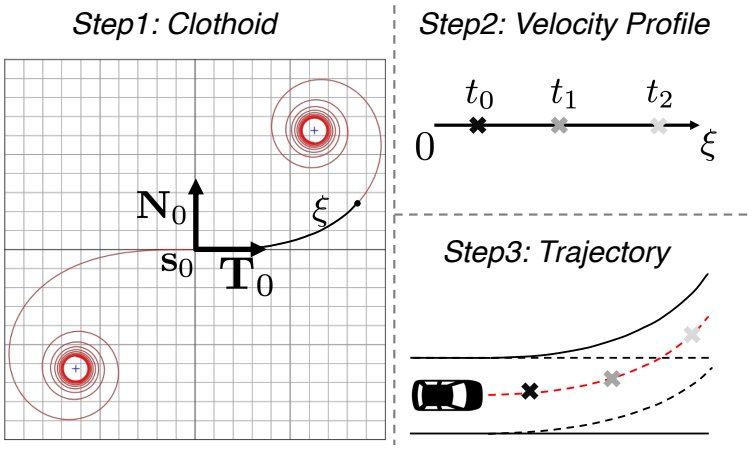


Figure 6.4: Trajectory Representation

over the anchors. In total, there are 12 anchors at each location, with different sizes, aspect ratios, and orientations. The classification branch outputs a score $p^k_{i,j}$ for each anchor indicating the probability of a vehicle at each anchor's location. The regression branch also outputs regression targets for each anchor $a^k_{i,j}$ at different time-steps. This includes localization offset l^t_x, l^t_y, size s^t_w, s^t_h and heading angle a^t_{sin}, a^t_{cos}. The superscript t stands for time frame, ranging from 0 (present) to $T-1$ into the future. Regression is performed at every timestep, thus producing motion forecasting for each vehicle.

Cost Volume Head: The cost volume head consists of several convolution and deconvolution layers. To produce a cost volume c at the same resolution as our bird-eye-view (BEV) input, we apply two deconvolution layers on the backbone's output with filter number $\{128, 64\}$, filter size 3x3 and stride 2. Each deconvolution layer is also followed by a convolution layer with filter number $\{128, 64\}$, filter size 3x3, and stride 1. We then apply a final convolution layer with filter number T, which is our planning horizon. Each filter generates a cost volume c^t for a future timestep t. This allows us to evaluate the cost of any trajectory $\mathbf{s}$ by simply indexing in the cost volume c. In our experiments, we also clip the cost volume value between -1000 to +1000 after the network. Applying such bounds prevents the cost value shifting arbitrarily, and makes tuning hyper-parameters easier. We next describe our output trajectory parameterization.

6.2.2 Efficient Inference

Given the input LiDAR sweeps and the HD map, we can compute the corresponding cost volume c by feed-forward convolutional operations as described above. The final trajectory can then be computed by minimizing Eq. (6.1). Note, however, that this optimization is NP-hard due to the exponentially large number of possible trajectories. We thus rely on sampling to obtain a low-cost trajectory. Towards this goal, we sample a wide variety of diverse trajectories that can be executed by the SDV and produce as final output the one with minimal cost according to our learned cost volume. This is guaranteed by using a physically plausible trajectory sampler. For this procedure to be successful, we need to be able to efficiently sample trajectories that are physically possible, and we need to be able to evaluate the cost volume efficiently. In this section, we describe in detail how we sample, as well as how we evaluate our cost volume efficiently.

Output Parameterization: A trajectory can be defined by the combination of the spatial path (a curve in the 2D plane) and the velocity profile (how fast we go along this path).

For sampling a spatial path, the most naive way is to sample a set of points $(x, y) \in \Re^2$ in space, corresponding to the preferred location over different time steps. This approach has the advantage of diversity as it is guaranteed to cover the best trajectories possible. However, the limitation is that the sampling space is way too large, while at the same time most of them are not physically possible for self-driving cars due to the limits in speed, acceleration, and turning angle. Thus a better way is to follow a dynamic model for self-driving car and sample only in the space defined by the dynamic model. This can greatly reduce the sampling space while making sure all sampled spatial path defined by the set of points in cartesian space are executable by the self-driving vehicle.

In this work, we employ the bicycle model [132], which is widely used for planning in self-driving cars. This model implies that the curvature κ of the vehicle's path is approximately proportional to the steering angle ϕ (angle between the front wheel and the vehicle): $\kappa = 2tan(\phi)/L \approx 2\phi/L$, where L is the distance between the front and rear axles of the SDV. This is a good approximation as ϕ is usually small.

We then utilize a *Clothoid* curve, also known as Euler spiral or Cornu spiral, to represent the 2D path of the SDV [151]. We refer the reader to Fig. 6.4 for an illustration. The curvature κ of a point on this curve is proportional to its distance ξ alone the curve from the reference point, *i.e.*, $\kappa(\xi) = \pi\xi$, Considering the bicycle model, this linear curvature characteristic corresponds to steering the front wheel angle with constant angular velocity. The canonical form of a Clothoid can be defined as

$$\mathbf{s}(\xi) = \mathbf{s_0} + a\left[C\left(\frac{\xi}{a}\right)\mathbf{T_0} + S\left(\frac{\xi}{a}\right)\mathbf{N_0}\right] \tag{6.2}$$

$$S(\xi) = \int_0^{\xi} sin\left(\frac{\pi u^2}{2}\right) du \tag{6.3}$$

$$C(\xi) = \int_0^{\xi} cos\left(\frac{\pi u^2}{2}\right) du \tag{6.4}$$

Here, $\mathbf{s}(\xi)$ defines a Clothoid curve on a 2D plane, indexed by the distance ξ to reference point $\mathbf{s_0}$, a is a scaling factor, $\mathbf{T_0}$ and $\mathbf{N_0}$ are the tangent and normal vector of this curve at point $\mathbf{s_0}$. $S(\xi)$ and $C(\xi)$ are called the *Fresnel integral*, and can be efficiently computed. In order to fully define a trajectory, we also need a longitudinal velocity $\dot{\xi}$ (velocity profile) that specifies the SDV motion along the path $\mathbf{s}(\xi)$: $\dot{\xi}(t) = \ddot{\xi}t + \dot{\xi}_0$, where $\dot{\xi}_0$ is the initial velocity of the SDV and $\ddot{\xi}$ is a constant forward acceleration. Combining this and (6.2), we can obtain the trajectory points $\mathbf{s}$ in Eq. (6.1).

Sampling: Since we utilize Clothoid curves, sampling a path corresponds to sampling the scaling factor a in Eq. (6.2). This scaling factor controls the shape of a sampled path. Since the final trajectory can only come from sampled candidates, we need to make sure the set of trajectories we sample diversified enough. On the other hand, considering the standard city driving speed limit of 15m/s (equivalent to 54 km/h), we sample a from the range of 6 to 80m. Once a is sampled, the shape of the curve is fixed. We then use the initial SDV's steering angle (curvature) to find the corresponding position on the curve. Note that Clothoid curves cannot handle circle and straight line trajectories well; thus, we sample them separately. The probability of using straight-line, circle, and Clothoid curves are 0.5, 0.25, 0.25, respectively. Also, we only use a single Clothoid segment to specify the path of SDV, which we think is enough for the short planning horizon. In addition, we sample constant accelerations $\ddot{\xi}$ ranging from $-5m/s^2$ to $5m/s^2$ which specifies the SDV's velocity profile. Combining sampled curves and

velocity profiles, we can project the trajectories to discrete time steps and obtain the corresponding waypoints (See Fig 6.4) for which to evaluate the learned cost.

6.2.3 End-to-End Learning

Our ultimate goal here in this chapter is to plan a safe trajectory while following the rules of traffic. We want the model to understand where obstacles are and where they will be in the future to avoid collisions. Therefore, we use a multi-task training with supervision from detection, motion forecasting as well as the human-driven trajectories for the ego-car. Note that we do not have supervision for cost volume. We thus adopt max-margin loss to push the network to learn to discriminate between good and bad trajectories. The overall loss function is then:

$$\mathcal{L} = \mathcal{L}_{\text{perception}} + \beta \mathcal{L}_{\text{planning}}. \tag{6.5}$$

This multi-task loss not only directs the network to extract useful features but also make the network output interpretable results. This is crucial for self-driving as it helps understand failure cases and improves the system. In the following, we describe each loss in more details.

Perception Loss: Our perception loss includes classification loss, for distinguishing a vehicle from the background, and regression loss, for generating precise object bounding boxes. For each predefined anchor box, the network outputs a classification score as well as several regression targets. This classification score $p^k_{i,j}$ indicates the probability of the existence of a vehicle at this anchor. We employ a cross-entropy loss for the classification defined as

$$\mathcal{L}_{cla} = \sum_{i,j,k} \left(q^k_{i,j} \log p^k_{i,j} + (1 - q^k_{i,j}) \log(1 - p^k_{i,j}) \right), \tag{6.6}$$

where $q^k_{i,j}$ is the class label for this anchor (*i.e.*, $q^k_{i,j} = 1$ for vehicle and 0 for background). The regression outputs include information of position, shape and heading angle at each time frame t, namely

$$l_x = \frac{x^a - x^l}{w^a} \quad l_y = \frac{y^a - y^l}{h^a},$$

$$s_w = \log \frac{w^a}{w^l} \quad s_h = \log \frac{h^a}{h^l},$$

$$a_{sin} = sin(\theta^a) - sin(\theta^l) \quad a_{cos} = cos(\theta^a) - cos(\theta^l),$$

where superscript a means anchor and l means label. We use a weighted smooth L1 loss over all these outputs. The overall *perception loss* is

$$\mathcal{L}_{perception} = \sum \left(\mathcal{L}_{cla} + \alpha \sum_{t=0}^{T} \mathcal{L}^t_{reg} \right). \tag{6.7}$$

Note that the regression loss is summed over all vehicle correlated anchors, from the current time frame to our prediction horizon T. Thus it teaches the model to predict the position of vehicles at every time frame.

To find the training label for each anchor, we associate it to its neighboring ground-truth bounding box, similar to [107, 114]. In particular, for each anchor, we find all the ground-truth boxes with intersection over union (IoU) higher than 0.4. We associate the highest one among them to this anchor and compute the class label and regression targets accordingly. We also associate any non-assigned ground-truth boxes with their nearest neighbor. The remaining anchors are treated as background and are not considered in the regression loss. Note

that one ground-truth box may associate to multiple anchors, but one anchor can at most be associated with one ground-truth box. During training, we also apply hard negative mining to overcome imbalance between positive and negative samples.

Planning Loss: Learning a reasonable cost volume is challenging as we do not have ground-truth. To overcome this difficulty, we minimize the max-margin loss where we use the ground-truth trajectory as a positive example, and randomly sampled trajectories as negative examples. The intuition behind is to encourage the ground-truth trajectory to have minimal cost, and others to have higher costs. More specifically, assume we have a ground-truth trajectory $\{(x^t, y^t)\}$ for the next T time steps, where (x^t, y^t) is the position of our vehicle at the t time step. Define the cost volume value at this point (x^t, y^t) as $\hat{c}^t$. Then, we sample N negative trajectories, the i^{th} among which is $\{(x_i^t, y_i^t)\}$ and the cost volume value at these points are c_i^t. The overall max-margin loss is defined as

$$\mathcal{L}_{\text{planning}} = \sum_{\{(x^t,y^t)\}} \left(\max_{1 \le i \le N} \left(\sum_{t=1}^{T} \left[\hat{c}^t - c_i^t + d_i^t + \gamma_i^t \right]_+ \right) \right) \tag{6.8}$$

The inner-most summation denotes the discrepancy between the ground-truth trajectory and one negative trajectory sample, which is a sum of per-timestep loss. $[]_+$ represents a ReLU function. This is designed to be inside the summation rather than outside, as it can prevent the cost volume at one time-step from dominating the whole loss. d_i^t is the distance between negative trajectory and ground-truth trajectory $||(x^t, y^t) - (x_i^t, y_i^t)||_2$, which is used to encourage negative trajectories far from the ground-truth trajectory to have much higher cost. γ_i^t is the traffic rule violation cost, which is a constant if and only if the negative trajectory t violates traffic rules at time t, *e.g.* moving before red-lights, colliding with other vehicles etc. This is used to determine how 'bad' the negative samples are, as a result, it will penalize those rule violated trajectories more severely and thus avoid dangerous behaviors. After computing the discrepancy between the ground-truth trajectory and each negative sample, we only optimize the worst case by the $\max$ operation. This encourages the model to learn a cost volume that discriminates good trajectories from bad ones.

6.3 Experiments

In this section, we evaluate our approach on a large scale real-world driving dataset. The dataset was collected over multiple cities across North America, over different seasons. In order to create a balanced dataset for self-driving, we need to consider different aspects and characteristics of the dataset. In particular, we take into account the diversity of time of the day, season, traffic conditions. As a result, we made the dataset consists of 6,500 scenarios with about 1.4 million frames, the training set consists of 5,000 scenarios, while validation and test have 500 and 1,000 scenarios, respectively. Our dataset has annotated 3D bounding boxes (bounding box size and heading in top-down view) of all objects including vehicles, pedestrians, and cyclists for each frame. Each object has a unique id; thus, it is tracked over time. In this chapter's experiments, we focus our attention on the vehicle to vehicle interactions and leave the detection of pedestrians, cyclists, and other road users to future work.

For all experiments, we utilize the same spatial region, which is centered at the SDV, with 70.4 meters both in front and back, 40 meters to the left and right, and height from -2 meters to 3.4 meters. This corresponds to a 704x400x27 tensor with an interval of 0.2 meters per pixel on all dimensions. Each time step corresponding to 100ms and we use an input sequence of 10 frames at 10Hz, while the output is 7 frames at 2Hz, resulting in a planning horizon of 3 seconds.

Method	L2 (m)			Collision Rate (%)						Traffic Violation (%)		
	1.0s	2.0s	3.0s	0.5s	1.0s	1.5s	2.0s	2.5s	3.0s	1.0s	2.0s	3.0s
Ego-motin	0.281	0.900	2.025	**0.00**	**0.01**	0.20	0.54	1.04	1.81	0.51	2.72	6.73
IL	**0.231**	**0.839**	**1.923**	**0.00**	**0.01**	0.19	0.55	1.04	1.72	0.44	2.63	5.38
Acc	0.403	1.335	2.797	0.05	0.12	0.27	0.53	1.18	2.39	**0.24**	**0.46**	**0.64**
Manual Cost	0.402	1.432	2.990	**0.00**	0.02	0.09	0.22	0.79	2.21	0.39	2.73	5.02
Ours(3s)	0.314	1.087	2.353	**0.00**	**0.01**	**0.04**	**0.09**	**0.33**	**0.78**	0.35	0.77	2.99

Table 6.1: Planning Metrics

In the following, we first show quantitative analysis for planning on a wide variety of metrics measuring collision, similarity to the human trajectory, and traffic rule violations. Next, we demonstrate the interpretability of our approach, through quantitative analysis of detection and motion forecasting, as well as visualization of the learned cost volume. Last, we provide an ablation study to show the effects of different loss functions and different temporal history lengths.

6.3.1 Planning Results

For open-loop planning evaluation, there is no standard well-recognized metrics. In order to get a comprehensive understanding of our planner, we first explain the set of metrics we used in the following.

- **L2 Distance to Real Trajectory**: This evaluates how far away the planned trajectory is from the real executed trajectory in the form of L2 distance between planned waypoints and real waypoints, *i.e.* $\mathcal{L}_{\text{dist}} = \sum_{t=1}^{T} \sqrt{(x_t - \hat{x}_t)^2 + (y_t - \hat{y}_t)^2}$, where x_t, y_t are planned waypoint at time t and $\hat{x}_t, \hat{y}_t$ are waypoint at time t from human preferable trajectories. Notice real trajectory is just one of the many possible trajectories that humans think is good; thus, a lower number indicates a higher precision.

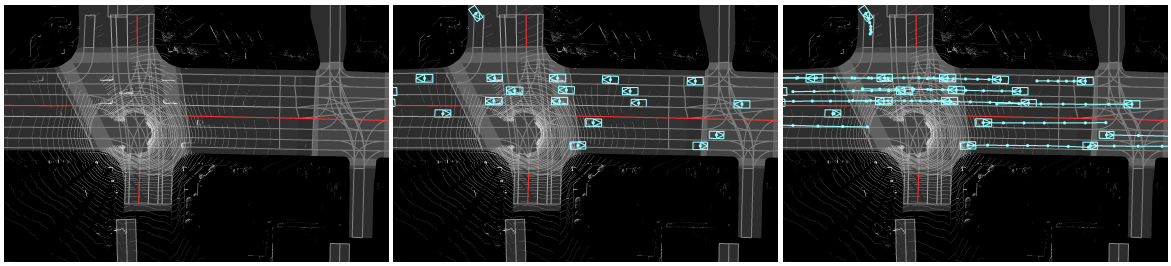
- **Future Potential Intervention Rate**: This is used to evaluate if a planned trajectory will overlap with other vehicles in the future. For a given time step t in future, We count the occurrence of all overlap up to time t as one intervention and compute the percentage of this happening; thus, a lower number is preferred.

- **Lane Violation**: It counts the occurrence of SDV *crossing* a solid yellow line. The number shows the accumulated percentage of this happening and lower is better. The *crossing* here is defined as if SDV touches the line.

Given the metrics, the other important element is the baselines. In the following, we introduce a few baselines from using learning algorithms to standard non-learning algorithms.

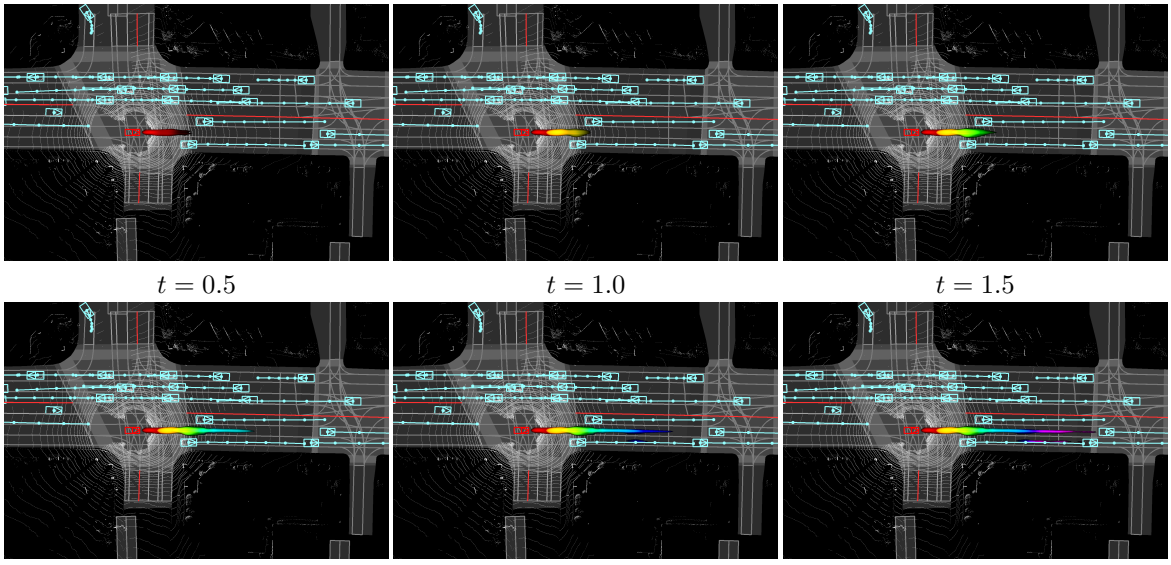
- **Ego-motion forecasting (Ego-motion)**: Ego-motion provides a strong cue of how the SDV would move in the future. This baselines takes only SDV's past position as input and uses a 4-layer MLP to predict the future locations. This baseline is expected to work well when driving in very simple driving condition, *i.e.*, keep going straight; however, it has a significant drawback that it does not reason about the surrounding vehicles, map information, etc.

- **Imitation Learning (IL)**: We follow the imitation learning framework [136, 17, 35], and utilize a deep network to extract features from raw LiDAR data and rasterized map. For fair comparison, we use the same backbone described (Sec. 6.2.1) and same input parameterization (Sec. 6.2.1) as our approach. Also, the same MLP from *Ego-motion forecasting* baseline is used to extract features from ego-motion. These two



Input representation Detection Motion forecasting

Figure 6.5: **Pipeline Visualization - Perception** We show a sample input representation of our model and the corresponding detection results (bounding boxes in cyan) as well as motion forecasting results (represented by waypoints) for 3 seconds into the future.



$t = 0.5$ $t = 1.0$ $t = 1.5$

$t = 2.0$ $t = 2.5$ $t = 3.0$

Figure 6.6: **Pipeline Visualization - Cost Map** Here we show the cost-map for each time step into future. In total, we prediction 6 frames into future on 2Hz. Cost-map are overlapped with different color for better visualization.

features are then concatenated and fed into a three-layer MLP to compute the final prediction. In addition to reasoning from ego-motion, this baseline can reason from the surrounding environment; however, it lacks the interpretability and generalization might be an issue.

- **Adaptive Cruise Control (ACC)**: This baseline implements the simple behavior of following the leading vehicle, which is widely used in the automation industry. The vehicle follows the lane center-line, while adaptively adjusting its speed to maintain a safe distance from the vehicle ahead. When there is no lead vehicle, a safe speed limit is followed. Traffic control, such as traffic lights and stop signs, are observed as a stationary obstacle, similar to a stopped lead vehicle.

- **Plan w/ Manual Cost (Manual)**: This baseline uses the same trajectory parameterization and sampling procedure as our approach. However, it utilizes a manually designed cost using perception and motion forecasting outputs. In detail, we rasterize all possible roads the SDV can take going forward and set it to a low cost of 0; all detected objects' bounding boxes is set to be 255 meaning a high-cost area; the cost of any other area is set to a default value 100. This baseline is designed to show the effectiveness of our learned

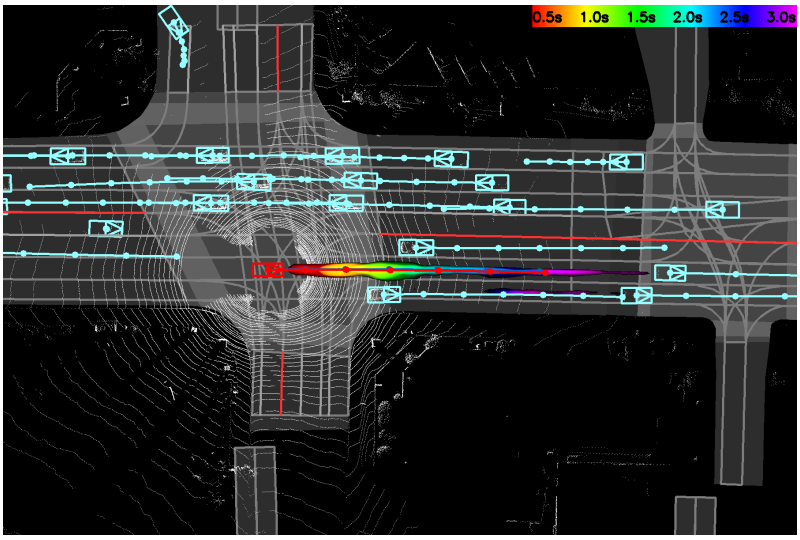


Figure 6.7: Overall visualization with planned trajectory in red.

cost volume as it utilizes the same sampling procedure as our approach but just a different cost volume.

The planning comparison results are shown in Tab. 6.1. Our approach has a lower future collision rate at all time steps by a large margin. Note that Ego-motion and IL baselines give lower L2 numbers as they optimize directly for this metric; however, they are not good from the planning perspective as they have difficulty reasoning about other actors and frequently collide with others. Comparing to the manual cost baseline and ACC, we achieve both better regression numbers and better collision rates, showing the advantage of our learned cost volume over manually designed cost. For lane violation, Notice ACC is designed to follow the lane; thus, it has 0 violation by definition. However, comparing to other baselines, we achieve a much smaller traffic violation number, showing our model can reason and learn from the map.

6.3.2 Interpretability

Interpretability is crucial for self-driving, as it can help the diagnosis of failure cases both offline and online. During an offline analysis, we can diagnose the system better with intermediate interpretation and more easily identify the root cause of failure cases. During online testing, good interpretability provides a real-time feedback of the performance of the system, and makes it easier for drivers or passengers to monitor the system.

In the following, we first show the visualization of the full pipeline, from the input representation to detection, motion forecasting, cost-map, and planning results. Then, we provide quantitative results on 3D detection and motion forecasting, where our approach achieves on-par or better results on metrics compared to methods designed specifically for these individual tasks. For the cost map, we show qualitatively that it provides multi-modality naturally and learns to reason for different time steps in the future. Additionally, it produces trajectories that obey traffic rules and avoid collisions.

Pipeline Visualization: As shown in Fig. 6.5, we give an example input on the left. Note that for better visualization, we only show one frame LiDAR points with smaller spatial dimension, while our model uses ten frames with a region of size 140x80 meters. The input tensor also encodes map information, including road, lane boundary, crossing, etc. (detailed in 6.2.1), the red line means the boundary vehicles must not cross. In the middle, we

Method	Detection mAP @ IoU (pts $\geq$ 1)				
	0.5	0.6	0.7	0.8	0.9
MobileNet[76]	86.1	78.3	60.4	27.5	1.1
FaF[114]	89.8	82.5	68.1	35.8	2.5
IntentNet[19]	**94.4**	89.4	75.4	43.5	3.9
PIXOR[187]	93.4	89.4	78.8	52.2	**7.6**
Ours	94.2	**90.8**	**81.1**	**53.7**	7.1

Table 6.2: Detection mAP Result

Method	L2 along trajectory (m)				L2 across trajectory (m)				L1 (m)				L2 (m)			
	0s	1s	2s	3s	0s	1s	2s	3s	0s	1s	2s	3s	0s	1s	2s	3s
FaF[114]	0.29	0.49	0.87	1.52	0.16	0.23	0.39	0.58	0.45	0.72	1.31	2.14	0.37	0.60	1.11	1.82
IntentNet[19]	0.23	0.42	0.79	1.27	0.16	0.21	0.32	0.48	0.39	0.61	1.09	1.79	0.32	0.51	0.93	1.52
Ours	**0.21**	**0.37**	**0.69**	**1.15**	**0.12**	**0.16**	**0.25**	**0.37**	**0.34**	**0.54**	**0.94**	**1.52**	**0.28**	**0.45**	**0.80**	**1.31**

Table 6.3: Motion Forecasting Metric

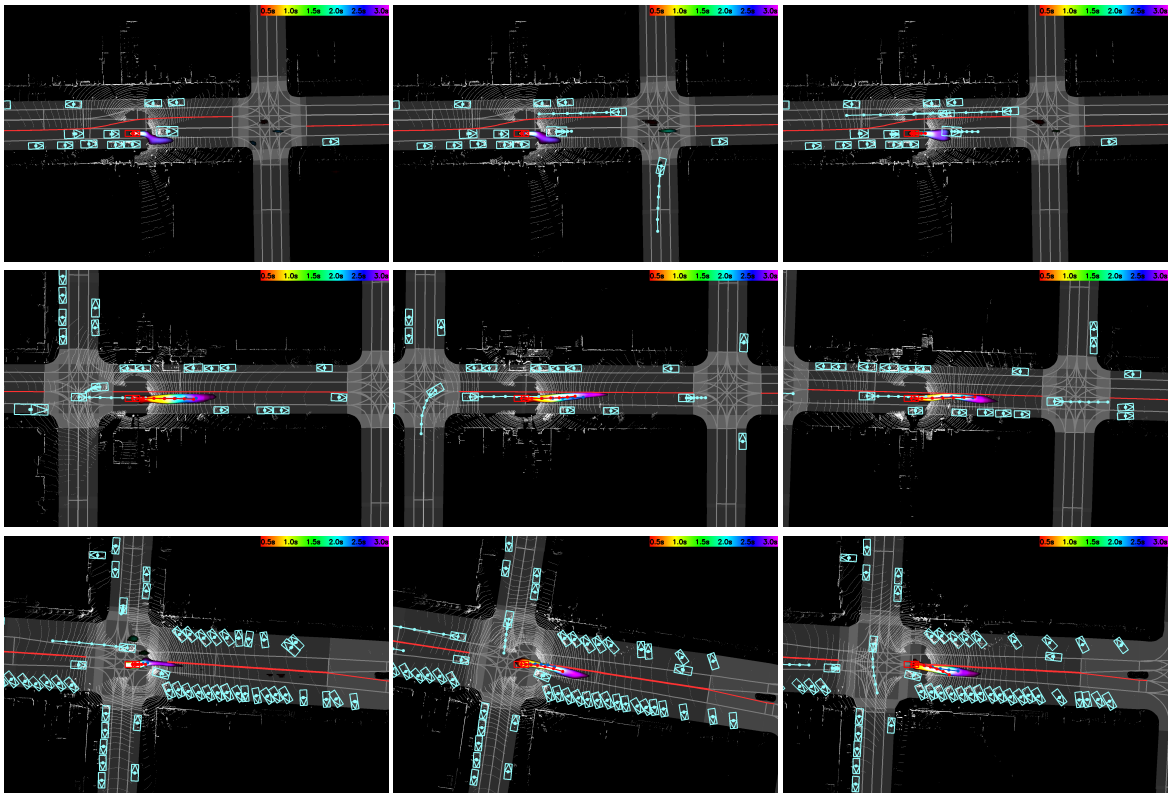
show the detection bounding boxes (in cyan) from our model. We can see that our model captures all vehicles with proper bounding boxes and consistent headings. On the right, we overlay the motion forecasting results on top. Each vehicle's future trajectory is shown with 6 points, representing its predicted location in 0.5, 1.0,..., 3.0 seconds into the future. By looking at the waypoints, we can also tell the predicted speed of the vehicles. In particular, when all waypoints overlap with each other, it means the vehicle is static (either stopped or parked) at the time.

In Fig. 6.6, we give an example on the learned cost map for all different time steps. Our model is trained to predict the cost-map for 6 frames into future with 2Hz, *i.e.* at $t = 0.5, 1.0, ..., 3.0$ seconds. We can clearly see that our model is able to learn time-dependent cost, and the cost follows the lane perfectly in this case.

In Fig. 6.7, we show the final visualization including all components of our approach for the example shown in both Fig. 6.5 and Fig. 6.6. In addition to the input HD map and LiDAR data, detection bounding boxes, motion forecasting, and cost-map, we show the planned trajectory in red. We can see that the planned trajectory is following the lane perfectly in this example, and it is consistent with our learned cost-map, demonstrate the effectiveness of our sampling-based inference algorithm.

Detection: Here we show quantitative results on detection. We compare against several state-of-the-art real-time detectors, validating that our holistic model understands the environment. Our baselines include a MobileNet adapted from [76], FaF[114], IntentNet[19] and PIXOR[187], which are specifically designed for LiDAR-based 3D object detection. The metric is mAP with different IoU thresholds, and vehicles without LiDAR points are not considered. As shown in Tab. 6.2, our model archives best results on 0.7 IoU threshold, which is the metric of choice for self-driving. In particular tit improves from PIXOR [187] by 2.3 points at 0.7 IoU threshold. Qualitative results can also be found in Fig. 6.8, Fig. 6.9 and Fig. 6.10.

Motion Forecasting: Tab. 6.3 shows quantitative motion forecasting results, including L1 and L2 distance to ground-truth locations. We also provide the L2 distance from our predictions to the ground-truth position along and perpendicular to the ground-truth trajectory. These help to distinguish between errors due to wrong velocities or direction estimations. We use baselines from [114, 19], which are designed for motion forecasting with raw LiDAR data. Our model performances better in all metric and all time steps. In particular, we improve from FaF [114] 1.52m to 1.15m for L2 along trajectory and 0.58m to 0.37m for L2 across trajectory at 3 second into future,



t = 0s t = 2s t = 4s

Figure 6.8: **Cost Volume across Time - Avoid Collision:** Each figure shows the learned cost volume for different future time steps overlapped using the color scheme in Fig. 6.6. Each row represents one sequence at different time steps.

which is 25% and 36% improvement respectively. Compared to IntentNet [19] which uses high-level intentions as additional information for training, we improve from 1.27m to 1.15m for L2 along trajectory and 0.48m to 0.37m for L2 across trajectory at 3 second into future, which is 10% and 23% improvement respectively. Qualitative results are shown in Fig. 6.8, Fig. 6.9 and Fig. 6.10.

Cost Map Visualization: In order to better demonstrate the effectiveness of our learned cost map, we provide more visualizations of the learned cost map across different time steps in different scenarios as shown in Fig. 6.8, Fig. 6.9 and Fig. 6.10. These figures give a top-down view of the scene, showing the map (using the format same as Fig. 6.5), LiDAR point clouds in red and planning results (in red) as well as detection (in cyan), motion forecasting (represented as cyan points and polyline connecting them) and learned cost map where we use the same color scheme introduced in Fig. 6.6.

As shown in Fig. 6.8, each figure shows the learned cost map represented by different colors for different time steps into the future. In particular, each row is one example and all figures are centered at the self-driving vehicle for simplicity. As we can see, in the first row of Fig. 6.8, the cost map correctly captures the stopped vehicle in front of the self-driving vehicle and provides the option to change lanes. The example in the second row from Fig. 6.8 shows that the cost map can avoid obstacles, even if it is not detected as dynamic objects of predetermined classes (*i.e.*, vehicles in this case). This is a perfect example which shows that our model can learn more useful information from the raw sensor compared to standard perception/prediction modules. The example in the last

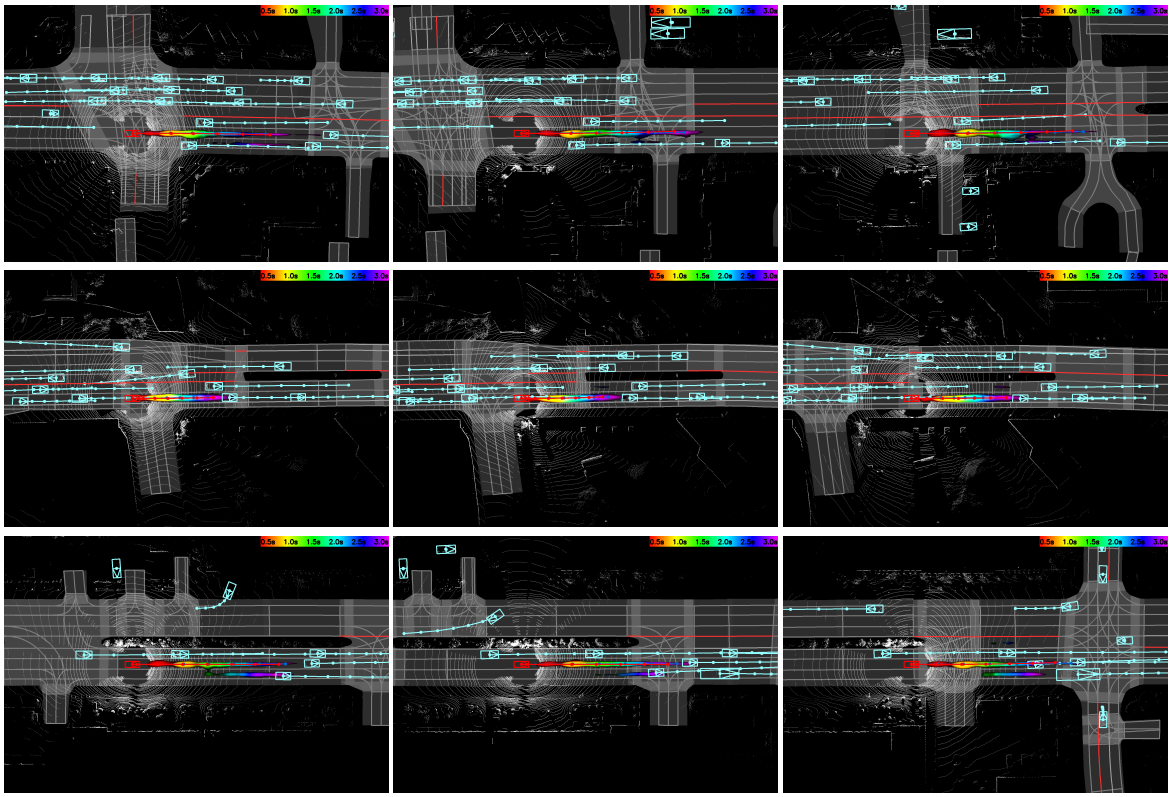


Figure 6.9: **Cost Volume across Time - Follow Lane:** Each figure shows the learned cost volume for different future time steps overlapped using the color scheme in Fig. 6.6. Each row represents one sequence at different time steps.

row gives another nudging case where our learned cost map can avoid the unexpected vehicle in front.

In a similar format, Fig. 6.9 gives three examples where the self-driving vehicle is driving in heavy traffic and on narrow roads. As we can see, our model gives accurate time-dependent cost map, capturing proper speed and direction. The final planned trajectories shown in red are stable and accurate.

Also, we provide another three examples shown in Fig. 6.10, where our model can learn multi-modality, *i.e.*, giving multiple low-cost regions going forward. When the self-driving vehicle is approaching intersections, our model provides multiple options, *i.e.* either going straight or making a turn. Once the self-driving vehicle is in or passing the intersection, the cost map becomes more uni-modal and provides accurate cost map going forward.

6.3.3 Ablation Study

In this section, we conduct ablation studies for a detailed understanding of our approach. The corresponding results are reported in Table 6.4. Our best model is Model 5. Compared to Model 1, which is optimized only for detection and motion forecasting, it can achieve similar performance in terms of detection and motion forecasting. Model 2 is trained only with planning loss, without the supervision of object bounding boxes. It performs worse in terms of planning metrics, showing the importance of intermediate supervision. Model 3 exploits different input length. As we can see, a longer input sequence gives better results on all metrics. This is again expected as longer input sequence gives a better estimation of motion not only for self-driving vehicles but also surrounding

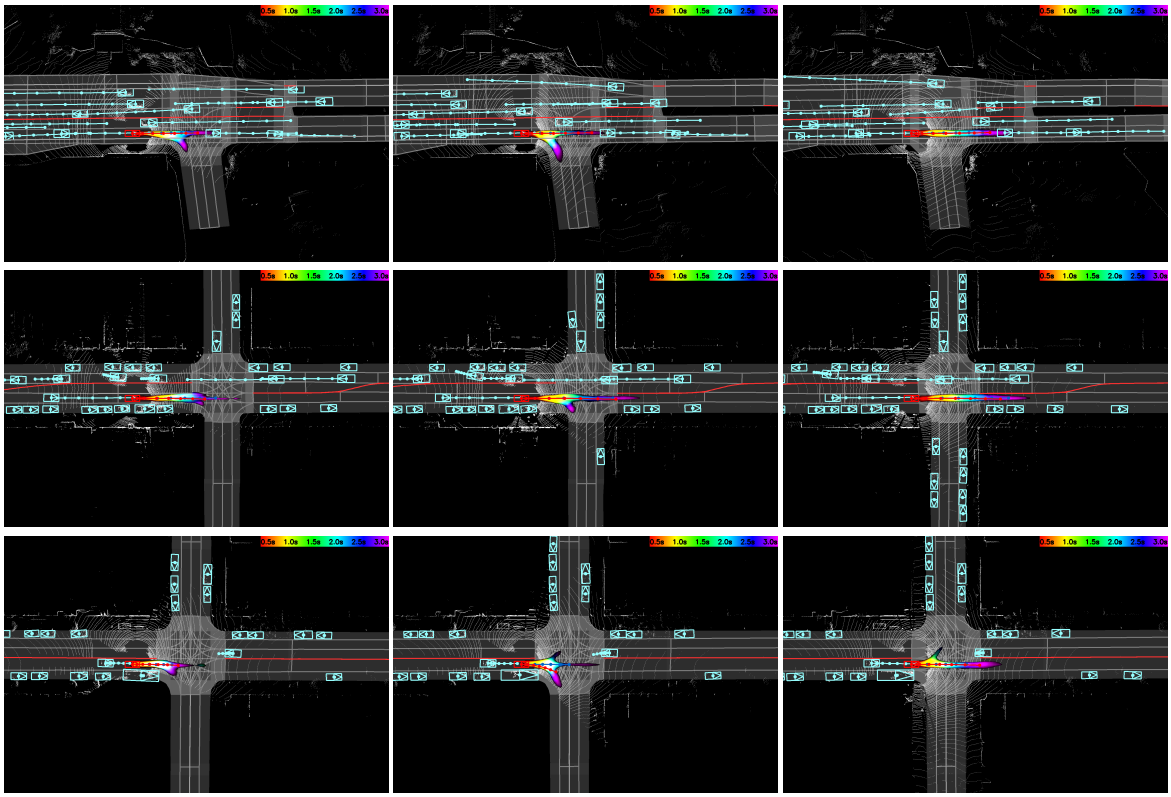


Figure 6.10: **Cost Volume across Time - Multi-Modality:** Each figure shows the learned cost volume for different future timesteps overlapped using the color scheme in Fig. 6.6. Each row represents one sequence at different timesteps.

dynamic objects. Model 4 is trained without the traffic rule penalty γ in Eq. 6.8. It performs worse on planning, as it has no prior knowledge about a collision, and is not explicitly trained to avoid collisions.

Traffic Lights A/B Test: In addition to vehicles on the road, another type of important elements in real-world driving is the traffic light. They direct traffic, assign right-of-way in an intersection, etc. Thus, the self-driving vehicle needs to understand the implicit meaning of traffic lights and plan the action accordingly, *i.e.*, following the traffic rules. Traffic lights are dynamic elements, meaning the same light can have different states at different time steps. We have shown the quantitative performance of our approach in the whole dataset in Tab. 6.1. Here, we additionally present ablation studies (A/B test) to examine the effect of traffic lights on the behavior of our model through changes in the resulting cost map. In particular, we pick a few interesting scenarios, especially near intersections. We manually change the traffic light in front of a self-driving vehicle to be *GREEN* in one experiment and *RED* in the other. By comparing the cost map from both experiments, we can see the effects of traffic lights. As shown in Fig. 6.11, each column is a pair of examples. The first row has traffic lights being *RED* and the second row has *GREEN* lights. We can see that when the lights are *RED*, the low cost regions in the cost maps are less spread out and mostly concentrated at the same location. When the lights are *GREEN*, the learned cost maps extend naturally forward, and at the same time avoid obstacles (*e.g.* third example). As a result, the final planned trajectories represent a stop action under *RED* lights and driving forward action under *GREEN* lights. This demonstrates that our model can learn to understand the concept of traffic lights and behave

ID	Loss		Input		Penalty	mAP@IoU		Prediction L2 (m)			Collision Rate (%)			Traffic Violation (%)		
	Det	Plan	5	10		0.5	0.7	1s	2s	3s	1s	2s	3s	1s	2s	3s
1	✓			✓		94.1	**81.3**	0.48	0.84	1.34	-	-	-	-	-	-
2		✓		✓		-	-	-	-	-	**0.01**	0.23	1.42	0.37	1.06	3.85
3	✓	✓	✓		✓	93.6	80.1	0.46	0.83	1.35	**0.01**	0.15	0.93	0.36	0.86	3.09
4	✓	✓		✓		**94.2**	81.1	**0.45**	**0.80**	**1.30**	**0.01**	0.29	1.40	0.36	1.02	3.26
5	✓	✓		✓	✓	**94.2**	81.1	**0.45**	**0.80**	1.31	**0.01**	**0.09**	**0.78**	**0.35**	**0.77**	**2.99**

Table 6.4: Ablation Study

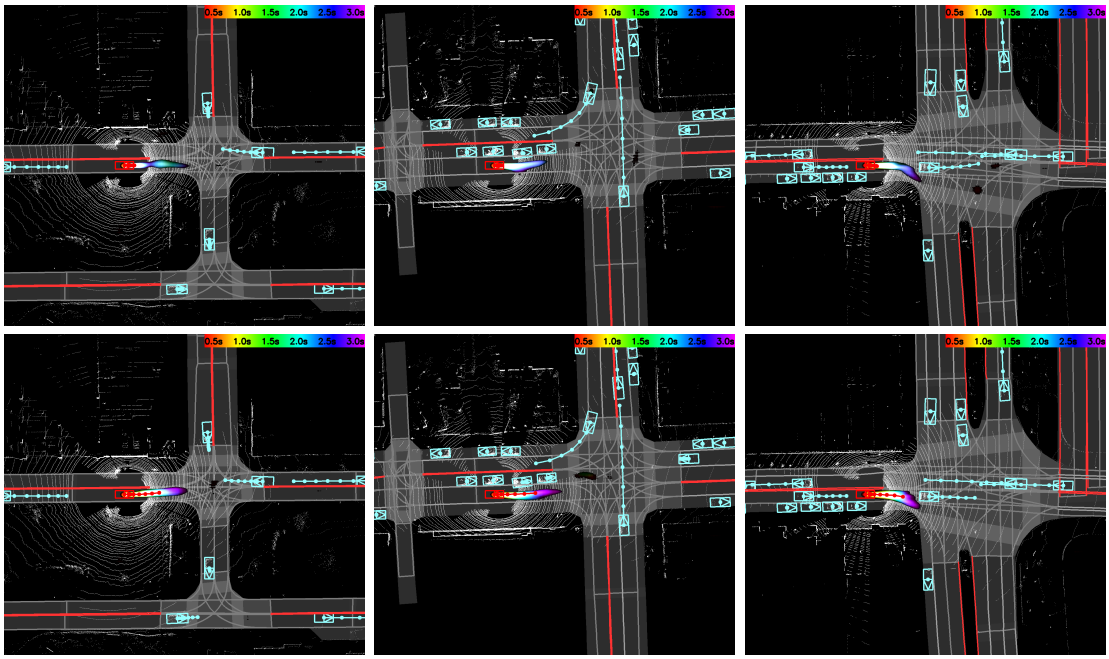


Figure 6.11: **Traffic Light A/B Test:** We show the learned cost map with traffic light manually on and off on same scenarios. First row has traffic light *RED* and second row has *GREEN* lights.

accordingly.

6.3.4 Close-Loop Test

In order to show the effectiveness of our approach for real-world self-driving situations, we need to perform closed-loop tests. This is an essential step as the self-driving vehicle will always execute the trajectory it plans during real-world online driving. There is no human intervention at every time step to change the execution (the safety driver is only used to make sure self-driving vehicle do not go catastrophically wrong and would allow the self-driving vehicle to choose an action when multiple options exist). In the following, we first describe the setup we use and show a set of samples covering different aspects of the problem.

Close-Loop Setup: The ideal setup for these experiments is to use a simulator that allows the self-driving vehicle to drive anywhere in that virtual world, and receive raw sensor data (LiDAR) as well as corresponding HD maps in real-time. While there is no such simulator publicly available, we can simplify the problem by making use of the recorded scenarios from the dataset we collected. In particular, we use the ground truth labels as input instead of raw sensor data. First, we take a recorded scenario and the corresponding HD map, *i.e.*, we know the

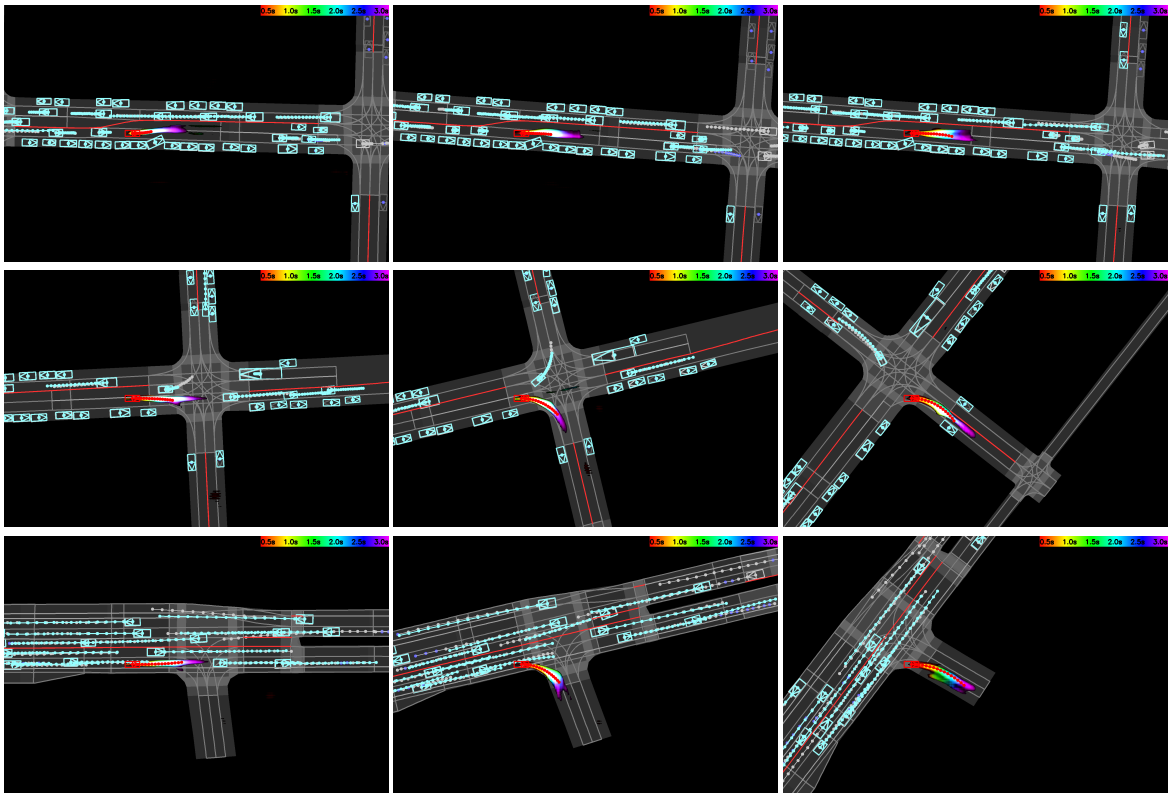


Figure 6.12: **Close-Loop Test:** Here we use different settings for input, *i.e.* rasterization from ground-truth labels. This is due to the limitation of sensor simulation.

location of all dynamic objects in different time steps. Also, the dynamic objects follow their existing trajectories, *i.e.*, they would not react to the self-driving vehicle. Second, we rasterize a top-down view image using dynamic objects' bounding boxes and locations in the past to generate the input tensor for our model. We train our model using the same algorithm but with the simulated input data. During inference time, instead of executing the human action, we make the self-driving vehicle take the first action from the previously planned trajectory exactly as a closed-loop test would need. Note that [10] also uses a similar setting for the closed-loop test. In this way, we can bypass the requirement of simulating LiDAR sensor data. This, however, is against the idea of learning from sensor data and leverage the most information from it. However, this can demonstrate the generalization of our algorithm in close-loop settings.

Sample Visualization Here we show a set of examples from the closed-loop test. We use the five frames spanning the past 0.8 seconds as the input; our model is trained to predict the cost map and plan for the 16 frames covering 3 seconds into the future. Fig. 6.12 shows three examples in the closed-loop test. Each row depicts one example. The first column represents time 0, the second column represents 2 seconds later, and the third column represents 4 seconds later. We observe that the intermediate 3D detection and motion forecasting remain of high quality. This is expected as the problem is much simpler as compared to the real world application. More importantly, the first example shows that we can nudge smoothly in response to a vehicle performing a parallel parking maneuver. The other two examples show the case where our algorithm can follow the lane in heavy traffic as well as make turns to avoid obstacles.

6.4 Discussion

In this chapter, we have proposed a neural motion planner that learns to drive safely. To the best of our knowledge, this is the first of its kind to learn to drive from sensor data directly with interpretable intermediate results. We have designed a holistic model that takes LiDAR data and an HD map as input and produces interpretable intermediate representations in the form of 3D detections and their future trajectories, as well as a time-dependent cost map defining the goodness of each position that the self-driving car can take within the planning horizon. Our planer then samples a set of physically possible trajectories and chooses the one with the minimum cost. We have demonstrated the effectiveness of our approach in very complex real-world scenarios in several cities of North America and show that we can learn to drive safely. We have demonstrated that our approach gives better planning results in terms of future collision rate, traffic rule violation, etc. compared to standard approaches ranging from simple ego-motion regression to complicated imitation learning, adaptive cruise control and planning with a manual cost. We visualized the results including 3D detections, motion forecasting, time-dependent cost map as well as planned trajectories. Furthermore, we performed A/B testing using traffic lights with manually determined statuses to demonstrate that our approach can learn the concept of traffic lights and plan accordingly. Last but not least, we perform closed-loop testing in a slightly different setting due to the lack of a perfect simulator. These preliminary results on the closed-loop test show that our approach can drive appropriately in certain scenarios.

Despite the exciting and promising results we have achieved so far, there are certain limitations. First, our sampler uses Euler spiral in order to get physically possible trajectories for the planner. The curvature of a target point on the Euler spiral is proportional to the distance from the starting point to the target point. This is a natural fit for vehicle that changes direction at most once, but not for complex long-term motion. A simple example is the lane change behavior, where a single Euler spiral cannot model the full lane change trajectory. To fix this issue, we can extend to multiple Euler spirals, while making sure the curvature is continuous among all segments. This could potentially solve the problem of drifting too far away when we only need to make a quick lane change. On the other hand, our approach currently only consider vehicles on the road; however, there are more categories of dynamic objects in the real world, such as pedestrians and cyclists. We need to extend the target classes to cover these traffic participants. Furthermore, we use rasterized HD maps as input to the network currently. However, this may be suboptimal, and we should exploit different representation of the map, *e.g.*, to encode the map in a way that respects the traffic rules by definition.

Learning the cost map and the corresponding planner from sensor data is a hard problem. In this chapter, we propose the first approach in this direction and show promising results. Additionally, it showcases the merit of using deep structured models. In particular, we use a deep neural network to learn to produce useful unary features from sensor data, and use a sampling-based inference method to generate the final planning trajectory. We could potentially add more well-known prior knowledge into the structured model. This could improve the performance because deep neural network models require a large amount of corresponding data to learn certain behavior. Unfortunately, we often only have limited data on rare and important corner cases. Thus, we can incorporate priors into the structured models to capture those rare cases, making the learning process of the whole system easier. In conclusion, a hybrid system with learning and predefined rules could be the best way to achieve autonomous driving.

Chapter 7

Conclusion

In this thesis, I present the work done during my Ph.D. studies on building smarter self-driving vehicles. In particular, I focus on developing algorithms that learn from data for better performance as well as a better understanding of existing models. In the following, I will summarize the work and give a short discussion on future work.

7.1 Summary

Building a self-driving vehicle is an exciting but difficult task. While previous approaches utilize lots of manually tuned rules, we exploit deep neural networks to tackle this problem.

First of all, as convolutional neural networks have been widely used in a variety of vision tasks, we provide a theoretical analysis of CNNs from the perspective of the receptive field in chapter 2. We carefully study its properties in deep CNNs and establish a few surprising results about the effective receptive field size using Fourier transform. We have shown that the distribution of impact within the receptive field is asymptotically Gaussian, even with a 2D rectangular kernel. In deep CNNs, the effective receptive field only takes up a small fraction of the full theoretical receptive field. We have also conducted empirical experiments for image classification and semantic segmentation tasks. The results echoed the theory we established. This work on the receptive field is just a start on the study of CNNs' properties that provides a different angle for a better understanding. It indeed spurred later work on different state-of-the-art model architectures for various vision tasks.

Secondly, we exploit the power of deep neural networks in different well-defined vision tasks within the autonomy stacks and achieve better performance in terms of both accuracy and runtime compared to previous methods. In particular, chapter 3 looks at the problem of depth estimation using stereo cameras. Stereo depth estimation can be treated as an image patch matching problem. Different than the previous method which builds a cumbersome siamese network, we propose a simple siamese network with dot-product as an explicit distance metric on top of the CNN. The intuition behind this is that we want the CNN to learn specific features tailored to and compatible with the distance metric used. We also treat the patch matching problem as a multi-class classification problem instead of binary classification. This allows the network to calibrate the matching score across a larger context, thus being more accurate. We demonstrate the effectiveness of our algorithm on the KITTI dataset and show much better matching performance while being two orders of magnitude faster. In the following, chapter 4, we take it one step further to apply this novel matching network for the optical flow problem. We adopt the matching network to the 2D searching space and perform smoothing only on the topK matching results due to memory limitations. As a result, we can remove regions with repetitive patterns, specularity, etc. and provide

more reliable matching results. Furthermore, we treat each traffic participant (obtained by an instant segmentation algorithm) separately and assume rigid motion for each of them. We evaluate our proposed approach on the challenging KITTI 2015 optical flow benchmark and achieve state-of-the-art results at the time of publication.

Thirdly, we revisit the design choices of current autonomy stacks and propose new formulations in chapter 5 and chapter 6. After witnessing the tremendous progress of neural networks in different machine learning domains, it is straightforward to reconsider whether certain design choices are still the best options. In chapter 5, we propose a joint model for 3D detection, prediction, and tracking to close the gap between perception and prediction. We utilize a single neural network to learn features from sequential LiDAR data. Multiple prediction heads are used to detect bounding boxes at different time-stamps. Tracking is performed greedily via checking the overlap between predicted bounding boxes using previous evidence and current evidence. The benefits include (1) better accuracy, as we can learn them jointly and optimize globally; (2) faster runtime, as we can share the heavy feature computation between different modules. We demonstrate the effectiveness of our algorithm on a self-collected large-scale driving dataset. We show that our approach outperforms the state-of-the-art by a large margin and is more robust to occlusion, as well as sparse data at long range. Importantly, by sharing computation, we can perform all tasks in as little as 30 ms. Furthermore, in chapter 6, we take the framework one step further to include a planning module. We propose a deep structure model that uses a CNN for bounding box detection and prediction, as well as learning a time-dependent cost-map over the planning horizon. The cost-map defines the 'goodness' of each position that the self-driving vehicle can take within the planning horizon. Then, a sampling-based inference procedure is used to infer the desire planning trajectory further. We utilize Clothoid curves to achieve physically plausible trajectories. Furthermore, as we do not have the ground-truth labels for the cost map, we adapt max-margin loss to train the neural network to output a cost map. We evaluate our approach on a large-scale real-world driving dataset and achieve better performance than the baselines, including ego-motion forecasting, imitation learning, adaptive cruise control, and manual-cost based approach. The ablation study results also show that training jointly with perception and prediction can improve the planning performance, and our model can understand the effect of traffic lights. Further, the corresponding visualization of the cost map as well as the planned trajectories demonstrate lane following, collision avoidance, and multi-modality.

7.2 Future Work

While we developed different algorithms for self-driving from various perspectives, we are still far from building a perfect self-driving stack. However, there are reasons to believe that learning-based methods, especially different variances of deep neural networks, will be able to push the boundaries of modern self-driving techniques.

Various self-driving datasets [36, 53, 192, 78, 117] have been collected over the past few years, focusing on different aspects of the self-driving stack including semantic segmentation, 3D detection, and tracking. These large-scale datasets allow us to validate novel architectures and train bigger / complex models that are capable of handling multiple tasks. We have seen clear improvement from using a larger dataset, as it can exploit the power of deep neural networks while making it less prone to overfitting. However, the benefits of data decrease as we keep increasing the size of the dataset, *i.e.* we have diminishing returns with data usage. In the real world, we have the long-tail distribution of scenarios we need to consider. For a safety-critical application such as self-driving, it is too expensive to ignore certain scenarios, even if they rarely happen. This brings up very interesting research problems regarding how can we design models to easily adapt to new scenarios while not forgetting what is learned, and how much data—or what type of data—should we be collecting to improve the current system? Most of the current deep neural network models designed to perform supervised learning tasks such as perception

or prediction are trained to minimize certain losses on the whole dataset. The model will likely ignore cases that rarely happen, sacrificing performance there to make improvements on more popular scenarios. However, in the self-driving domain, we care about rare/edge cases to deliver a safe self-driving vehicle. For the edge cases we have seen in the datasets, we need to re-balance them during training, while for unseen edge cases, we need to make our model easily adaptable while not forgetting what has been already learned. This is also referred to as lifelong learning or continual learning. On the other hand, identifying the edge cases is also nontrivial. Different models will have different properties and, correspondingly, different edge cases. While it is true that we could use human labelers to identify specific cases and provide corresponding labels/corrections, it is certainly a very expensive and non-scalable approach. Thus, it is promising to look at active learning, while the learning procedure is performed by actively selecting samples that can best benefit the trained model.

Besides, with richer datasets and more insight about neural networks, various architectures [93, 153, 162, 69, 202] have been proposed that achieve better results on widely used benchmarks. We could also benefit from adapting and exploiting new architectures. While previous researchers with domain knowledge have been designing the architecture manually, it is worthwhile to use machine learning techniques to search for architectures automatically. Neural architecture searching is one of the promising directions that exploits the power of deep learning in the space of architecture design. More importantly, as the number of GPU hungry algorithms in the autonomy stacks increases, the performance and runtime tradeoff becomes critical. One can always reduce the size of the network to achieve lower runtime and less energy consumption; however, the performance would greatly decrease. People have proposed various architectures that run real-time with a relatively small performance drop [76, 200, 81]. However, there are still improvements needed for self-driving. On the other hand, model compression looks at the problem of how to efficiently compress a trained model without too much performance drop through quantization or clipping. Overall, exploring model architectures will certainly benefit the development of self-driving vehicles.

Furthermore, in this thesis, I propose algorithms dealing with images or LiDAR data separately. As mentioned before, each sensor has its advantages and disadvantages. They should be used together to get a full understanding of the surrounding environment. A simple example is that using LiDAR mounted on top of the vehicle would result in blind spots around the vehicle, *i.e.*, it cannot see pedestrians walking close to the vehicle, etc. Thus, we would need cameras to provide visual evidence for these areas. Another example is that both camera and LiDAR might not work well in severe weather conditions. Additionally, image and LiDAR data cannot provide accurate speed information, which is important for a self-driving vehicle, as it needs to understand the behavior of surrounding traffic participants very accurately. Thus, we would need Radar that can operate in various weather conditions and measure the velocity of other objects accurately. While using different models for different sensor data does work well to some extent, the information is not shared across different domains, and the model can hardly reason about different sensor data at the same time. Thus, it becomes clear that we need to develop algorithms that consume multiple sensor data and optimize together, *i.e.*, multi-sensor fusion [104, 103]. The advantages of multi-sensor fusion are two-fold. First of all, it can reduce computation as certain neural network layers are shared across different sensor domains. Secondly, learning jointly can achieve better results compared to training separately and manually merging the results. The advantages of a multi-sensor fusion are clear; however, it is not a trivial task. The challenge includes finding the correspondence between different sensor data and fusing them effectively and efficiently, *i.e.*, a LiDAR-image model should outperform a LiDAR-only or image-only model, while being faster than running two models sequentially. Furthermore, the model should be designed to be robust to sensor failure, *i.e.* a LiDAR-image model should still perform similarly to as LiDAR-only model while the camera fails to capture images at certain times or areas.

Another promising direction is to extend further the structure model we proposed in chapter 6. While a neural network can be trained to perform well on all kinds of supervised learning tasks such as stereo depth estimation (chapter 4), optical flow (chapter 2), 3D detection, prediction (chapter 5), etc., it is very hard to encode rules into it. However, there are many predefined driving rules on the road, such as obeying the traffic lights, staying under speed limits, and driving in proper lanes. We could indeed hope the neural network learn these behaviors from large amounts of real-world driving data, but satisfying these rules at inference time is never guaranteed, and driving rules could vary widely across different areas. For example, while in most cities a left-turn vehicle will wait for an oncoming vehicle in an unprotected left-turn scenario, it is the opposite in the Pittsburgh area—a practice often referred to as 'Pittsburgh left.' Thus, a better solution would be to explicitly specify different driving rules as well as any fixed prior knowledge the human driver would have. A straightforward solution is to encode this information as potential in a structured model, where neural network output also serves as a potential term. Thus, the final output would be computed via inference on the structured model. Chapter 6 provides the first attempt in this direction; however, the structure model part is still very limited and does not exploit predefined rules during inference. On the other hand, building the structured model could introduce the causality (if using a directed model) between different components explicitly and could provide better interpretability for the whole system.

Finally, there are lots of remaining challenges in building a realistic simulator for self-driving. Real-world tests for self driving would be very expensive and even dangerous in certain cases. Simulation, on the other hand, provides a faster (*e.g. utilizing parallelism*) and safer test. Simulators have been used to test prediction and planning modules with predefined traffic participants or real driving logs. However, simulating sensor data is also essential. This enables us to test methods such as those proposed in chapter 5 and 6, where neural networks consume sensor data directly. This would also allow us to create arbitrary adversary sensor data to test the robustness of the system.

While I think all the above are promising directions for building a smart self-driving vehicle, it is indeed a difficult problem involving both general research and engineering work. I feel very lucky to be able to witness and participate in the development of self-driving. This thesis only scratches the surface of self-driving technology. I will always stay humbled and excited to see further breakthroughs in this area and, of course, the bright future of self-driving vehicles.

Bibliography

[1] A.Bruhn, J.Weickert, and C.Schnoerr. Lucas/Kanade Meets Horn/Schunck: Combining Local and Global Optic Flow Methods. *International Journal of Computer Vision*, 61(3):211–231, 2004.

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2012.

[3] Zlatan Ajanovic, Bakir Lacevic, Barys Shyrokau, Michael Stolz, and Martin Horn. Search-based optimal motion planning for automated driving. *arXiv preprint arXiv:1803.04868*, 2018.

[4] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971, 2016.

[5] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Min Bai, Wenjie Luo, Kaustav Kundu, and Raquel Urtasun. Exploiting semantic information and deep matching for optical flow. In *The European Conference on Computer Vision (ECCV)*, pages 154–170. Springer, 2016.

[8] Min Bai and Raquel Urtasun. Deep watershed transform for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5221–5229, 2017.

[9] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-aware motion planning. In *Algorithmic foundations of robotics X*, pages 475–491. Springer, 2013.

[10] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[11] L. Bao, Q. Yang, and H. Jin. Fast edge-preserving PatchMatch for large displacement optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[12] Stan Birchfield and Carlo Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999.

[13] M. Bleyer, C. Rhemann, and C. Rother. Extracting 3D scene-consistent object proposals and depth from stereo images. In *The European Conference on Computer Vision (ECCV)*, 2012.

[14] M. Bleyer, C. Rother, and P. Kohli. Surface stereo with soft segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[15] M. Bleyer, C. Rother, P. Kohli, D. Scharstein, and S. Sinha. Object stereo - joint stereo matching and object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[16] Michael Bleyer and Margrit Gelautz. A layered stereo matching algorithm using image segmentation and global visibility constraints. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2005.

[17] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[18] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[19] Sergio Casas, Wenjie Luo, and Raquel Urtasun. Intentnet: Learning to predict intention from raw sensor data. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 947–956. PMLR, 29–31 Oct 2018.

[20] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5410–5418, 2018.

[21] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730. IEEE, 2015.

[22] J. Chen, W. Zhan, and M. Tomizuka. Constrained iterative lqr for on-road autonomous driving motion planning. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7, Oct 2017.

[23] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 40(4):834–848, 2018.

[24] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. *arXiv preprint arXiv:1504.04788*, 2015.

[25] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3D Object Detection for Autonomous Driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[26] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *Advances in neural information processing systems (NIPS)*, 2015.

[27] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals using stereo imagery for accurate object class detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2017.

[28] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[29] Zhuoyuan Chen, Xun Sun, Liang Wang, Yinan Yu, and Chang Huang. A deep visual correspondence embedding model for stereo matching costs. In *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[30] Zhuoyuan Chen, Xun Sun, Liang Wang, Yinan Yu, and Chang Huang. A deep visual correspondence embedding model for stereo matching costs. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 972–980, 2015.

[31] Xinjing Cheng, Peng Wang, and Ruigang Yang. Depth estimation via affinity learned with convolutional spatial propagation network. In *The European Conference on Computer Vision (ECCV)*, pages 103–119, 2018.

[32] Xinjing Cheng, Peng Wang, and Ruigang Yang. Learning depth with convolutional spatial propagation network. *arXiv preprint arXiv:1810.02695*, 2018.

[33] Fang-Chieh Chou, Tsung-Han Lin, Henggang Cui, Vladan Radosavljevic, Thi Nguyen, Tzu-Kuo Huang, Matthew Niedoba, Jeff Schneider, and Nemanja Djuric. Predicting motion of vulnerable road users using high-definition maps and efficient convnets. 2018.

[34] C.Lei and Y.H.Yang. Optical Flow Estimation on Coarse-to-Fine Region-Trees using Discrete Optimization. In *The IEEE International Conference on Computer Vision (ICCV)*, 2009.

[35] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[36] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[37] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *The European Conference on Computer Vision (ECCV)*, pages 534–549. Springer, 2016.

[38] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems (NIPS)*, pages 379–387, 2016.

[39] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR, abs/1703.06211*, 1(2):3, 2017.

[40] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005.

[41] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. Ieee, 2009.

[42] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Smagt, D. Cremers, and T. Brox. FlowNet: Learning Optical Flow with Convolutional Networks. In *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[43] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[44] D.Sun, S.Roth, and M.J.Black. Secrets of Optical Flow Estimation and Their Principles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[45] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[46] Steffen Eger. Restricted weighted integer compositions and extended binomial coefficients. *Journal of Integer Sequences*, 16(13.1):3, 2013.

[47] Nils Einecke and Julian Eggert. A multi-block-matching approach for stereo. In *Intelligent Vehicles Symposium (IV)*, 2015.

[48] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341, 2009.

[49] Haoyang Fan, Fan Zhu, Changchun Liu, Liangliang Zhang, Li Zhuang, Dong Li, Weicheng Zhu, Jiangtao Hu, Hongye Li, and Qi Kong. Baidu apollo em motion planner. *arXiv preprint arXiv:1807.08048*, 2018.

[50] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[51] Sanja Fidler, Sven Dickinson, and Raquel Urtasun. 3D Object Detection and Viewpoint Estimation with a Deformable 3D Cuboid Model. In *Advances in neural information processing systems (NIPS)*, 2012.

[52] Thierry Fraichard and Christian Laugier. Path-velocity decomposition revisited and applied to dynamic trajectory planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 40–45. IEEE, 1993.

[53] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

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

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

[56] Ross Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[57] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[58] Haifeng Gong, Jack Sim, Maxim Likhachev, and Jianbo Shi. Multi-hypothesis motion planning for visual object tracking. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 619–626. IEEE, 2011.

[59] Fatma Guney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[60] C. Haene, L. Ladický, and M. Pollefeys. Direction Matters: Depth Estimation with a Surface Normal Classifier. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[61] R. Haeusler, R. Nair, and D. Kondermann. Ensemble learning for confidence measures in stereo vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[62] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[63] Jason Hardy and Mark Campbell. Contingency planning over probabilistic obstacle predictions for autonomous road vehicles. *IEEE Transactions on Robotics*, 29(4):913–929, 2013.

[64] R. Hartley. In Defence of the Eight-Point Algorithm. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 1997.

[65] James V Haxby, Barry Horwitz, Leslie G Ungerleider, Jose Ma Maisog, Pietro Pietrini, and Cheryl L Grady. The functional organization of human extrastriate cortex: a pet-rcbf study of selective attention to faces and locations. *The Journal of Neuroscience*, 14(11):6336–6353, 1994.

[66] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 37(9):1904–1916, 2015.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *The European Conference on Computer Vision (ECCV)*, 2016.

[71] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *The European Conference on Computer Vision (ECCV)*, pages 749–765. Springer, 2016.

[72] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[73] H. Hirschmuller. Stereo Processing by Semigloabl Matching and Mutual Information. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2008.

[74] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(2):328–341, 2007.

[75] B.K.P. Horn and B.G. Schunck. Determining Optical Flow. *Artificial Intelligence*.

[76] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[77] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[78] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The apolloscape dataset for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 954–960, 2018.

[79] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8981–8989, 2018.

[80] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. A lightweight optical flow cnn–revisiting data fidelity and regularization. *arXiv preprint arXiv:1903.07414*, 2019.

[81] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[82] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2462–2470, 2017.

[83] Eddy Ilg, Tonmoy Saikia, Margret Keuper, and Thomas Brox. Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation. In *The European Conference on Computer Vision (ECCV)*, pages 614–630, 2018.

[84] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[85] Kamal Kant and Steven W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The international journal of robotics research*, 5(3):72–89, 1986.

[86] Nancy Kanwisher, Josh McDermott, and Marvin M Chun. The fusiform face area: a module in human extrastriate cortex specialized for face perception. *The Journal of Neuroscience*, 17(11):4302–4311, 1997.

[87] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. *arXiv preprint arXiv:1807.00412*, 2018.

[88] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, and Peter Henry. End-to-end learning of geometry and context for deep stereo regression. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 66–75. IEEE, 2017.

[89] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[90] Andreas Klaus, Mario Sormann, and Konrad Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *International Conference on Pattern Recognition (ICPR)*, 2006.

[91] D. Kong and H. Tao. A method for learning matching errors for stereo computation. In *BMVC*, 2004.

[92] D. Kong and H. Tao. Stereo matching via learning multiple experts behaviors. In *BMVC*, 2006.

[93] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, pages 1097–1105, 2012.

[94] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[95] K.Yamaguchi, D.McAllester, and R.Urtasun. Efficient Joint Segmentation, Occlusion Labeling, Stereo and Flow Estimation. In *The European Conference on Computer Vision (ECCV)*, 2014.

[96] L. Ladický, J. Shi, and M. Pollefeys. Pulling Things out of Perspective. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[97] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In *Advances in neural information processing systems (NIPS)*, pages 1243–1251, 2010.

[98] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[99] V. Lempitsky, S. Roth, and C. Rother. FusionFlow: Discrete-Continuous Optimization for Optical Flow Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[100] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. *arXiv preprint arXiv:1611.08069*, 2016.

[101] Y. Li and D. P. Huttenlocher. Learning for stereo vision using the structured support vector machine. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[102] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2359–2367, 2017.

[103] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7345–7353, 2019.

[104] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *The European Conference on Computer Vision (ECCV)*, pages 641–656, 2018.

[105] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[106] Sifei Liu, Shalini De Mello, Jinwei Gu, Guangyu Zhong, Ming-Hsuan Yang, and Jan Kautz. Learning affinity via spatial propagation networks. In *Advances in neural information processing systems (NIPS)*, pages 1520–1530, 2017.

[107] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *The European Conference on Computer Vision (ECCV)*, 2016.

[108] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[109] L Lovsz, J Pelikn, and K Vesztergombi. Discrete mathematics: elementary and beyond, 2003.

[110] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[111] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[112] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in neural information processing systems*, pages 4898–4906, 2016.

[113] Wenjie Luo, Alex Schwing, and Raquel Urtasun. Efficient deep learning for stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2016.

[114] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net.

[115] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Hierarchical convolutional features for visual tracking. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 3074–3082, 2015.

[116] Wei-Chiu Ma, De-An Huang, Namhoon Lee, and Kris M Kitani. Forecasting interactive dynamics of pedestrians with fictitious play. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 774–782, 2017.

[117] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017.

[118] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196. IEEE, 2015.

[119] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[120] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 2016.

[121] Matthew McNaughton. Parallel algorithms for real-time motion planning. 2011.

[122] M. Menze and A. Geiger. Object Scene Flow for Autonomous Vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[123] M. Menze, C. Heipke, and A. Geiger. Discrete Optimization for Optical Flow. In *GCPR*, 2015.

[124] Moritz Menze and Andreas Geiger. Object Scene Flow for Autonomous Vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[125] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[126] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, 20, 2015.

[127] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladen Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

[128] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4293–4302, 2016.

[129] Thorsten Neuschel. A note on extended binomial coefficients. *Journal of Integer Sequences*, 17(2):3, 2014.

[130] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *The European Conference on Computer Vision (ECCV)*, pages 483–499. Springer, 2016.

[131] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.

[132] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[133] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[134] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert. Highly accurate optical flow computation with theoretically justified warping. *International Journal of Computer Vision*, 2006.

[135] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 261–268. IEEE, 2009.

[136] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[137] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4161–4170, 2017.

[138] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[139] M. Ren and R. Zemel. End-to-end instance segmentation and counting with recurrent attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[140] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems (NIPS)*, 2015.

[141] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C.Schmid. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[142] Nicholas Rhinehart, Kris M Kitani, and Paul Vernaza. R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting. In *The European Conference on Computer Vision (ECCV)*, pages 794–811. Springer, Cham, 2018.

[143] A. Roussos, C. Russell, R. Garg, and L. Agapito. Dense Multibody Motion Estimation and Reconstruction from a Handheld Camera. In *ISMAR*, 2012.

[144] R.Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *The European Conference on Computer Vision (ECCV)*, 1994.

[145] Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. *arXiv preprint arXiv:1806.06498*, 2018.

[146] D. Scharstein and C. Pal. Learning conditional random fields for stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[147] Daniel Scharstein and Richard Szeliski. Middlebury stereo vision page. *Online at http://www.middlebury.edu/stereo*, 2002.

[148] J. Schlechtriemen, K. P. Wabersich, and K. Kuhnert. Wiggling through complex traffic: Planning trajectories constrained by predictions. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1293–1300, June 2016.

[149] Milwaukee Sentinel. Phantom autowill tour city. *The Milwaukee Sentinel*, page 4, 1926.

[150] L. Sevilla-Lara, D. Sun, V. Jampani, and M. Black. Optical Flow with Semantic Segmentation and Localized Layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[151] Dong Hun Shin, Sanjiv Singh, and W Whittaker. Path generation for a robot vehicle using composite clothoid segments. *IFAC Proceedings Volumes*, 25(6):443–448, 1992.

[152] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[153] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[154] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[155] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[156] A. Spyropoulos, N. Komodakis, and P. Mordohai. Learning to detect ground control points for improving the accuracy of stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[157] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[158] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning, (ICML)*, pages 843–852, 2015.

[159] Deqing Sun, Stefan Roth, and Michael J Black. volume 106, pages 115–137, 2014.

[160] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Models matter, so does training: An empirical study of cnns for optical flow estimation. *arXiv preprint arXiv:1809.05571*, 2018.

[161] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8934–8943, 2018.

[162] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[163] Ran Tao, Efstratios Gavves, and Arnold WM Smeulders. Siamese instance search for tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1420–1429, 2016.

[164] E. Tola, V. Lepetit, and P. Fua. DAISY: An Efficient Dense Descriptor Applied to Wide Baseline Stereo. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2010.

[165] Leslie G Ungerleider and James V Haxby. whatand wherein the human brain. *Current opinion in neurobiology*, 4(2):157–165, 1994.

[166] R. Vidal, Y. Ma, S. Soatto, and S. Sastry. Two-View Multibody Structure from Motion. *International Journal of Computer Vision*, 68(1):7–25, 2006.

[167] C. Vogel, K. Schindler, and S. Roth. 3D Scene Flow Estimation . *International Journal of Computer Vision*, 2015.

[168] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015.

[169] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *The European Conference on Computer Vision (ECCV)*, pages 835–851. Springer, 2016.

[170] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 3119–3127, 2015.

[171] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *Advances in neural information processing systems (NIPS)*, pages 809–817, 2013.

[172] Zeng-Fu Wang and Zhi-Gang Zheng. A region based stereo matching algorithm using cooperative optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[173] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. DeepFlow: Large Displacement Optical Flow with Deep Matching. In *The IEEE International Conference on Computer Vision (ICCV)*, 2013.

[174] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. DeepFlow: Large displacement optical flow with deep matching. In *The IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, dec 2013.

[175] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 987–993. IEEE, 2010.

[176] Wikipedia contributors. Darpa grand challenge — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=DARPA_Grand_Challenge&oldid=866546657`.

[177] Wikipedia contributors. Eureka prometheus project — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Eureka_Prometheus_Project&oldid=874294598`.

[178] Wikipedia contributors. History of self-driving cars — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=History_of_self-driving_cars&oldid=876111593.

[179] Wikipedia contributors. Self-driving car — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Self-driving_car&oldid=879848335.

[180] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. *arXiv preprint arXiv:1612.01051*, 2016.

[181] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[182] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning, (ICML)*, 2015.

[183] K. Yamaguchi, D. Mcallester, and R. Urtasun. Robust Monocular Epipolar Flow Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[184] Koichiro Yamaguchi, Tamir Hazan, David McAllester, and Raquel Urtasun. Continuous markov random fields for robust stereo estimation. In *The European Conference on Computer Vision (ECCV)*, 2012.

[185] Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Robust monocular epipolar flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[186] Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *The European Conference on Computer Vision (ECCV)*, 2014.

[187] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7652–7660, 2018.

[188] H. Yang, W. Lin, and J. Lu. DAISY filter flow: A generalized discrete approach to dense correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[189] J. Yang and H. Li. Dense, Accurate Optical Flow Estimation with Piecewise Parametric Model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[190] Zhichao Yin, Trevor Darrell, and Fisher Yu. Hierarchical discrete distribution decomposition for match density estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6044–6053, 2019.

[191] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[192] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[193] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4353–4361, 2015.

[194] Jure Zbontar and Yann LeCun. Computing the stereo matching cost with a convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[195] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *arXiv preprint arXiv:1510.05970*, 2015.

[196] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *The European Conference on Computer Vision (ECCV)*, pages 818–833. Springer, 2014.

[197] Wei Zhan, Changliu Liu, Ching-Yao Chan, and Masayoshi Tomizuka. A non-conservatively defensive strategy for urban autonomous driving. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pages 459–464. IEEE, 2016.

[198] G. Zhang, J. Jia, and H. Bao. Simultaneous Multi-Body Stereo and Segmentation. In *The IEEE International Conference on Computer Vision (ICCV)*, 2011.

[199] L. Zhang and S. M. Seitz. Estimating optimal parameters for MRF stereo from a single image pair. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2007.

[200] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018.

[201] Z. Zhang, S. Fidler, and R. Urtasun. Instance-Level Segmentation with Deep Densely Connected MRFs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[202] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.

[203] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[204] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable basis decomposition for visual explanation. In *The European Conference on Computer Vision (ECCV)*, pages 119–134, 2018.

[205] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. *arXiv preprint arXiv:1811.11168*, 2018.

[206] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[207] J. Ziegler, P. Bender, T. Dang, and C. Stiller. Trajectory planning for bertha a local, continuous method. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 450–457, June 2014.

[208] Julius Ziegler, Philipp Bender, Thao Dang, and Christoph Stiller. Trajectory planning for berthaa local, continuous method. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 450–457. IEEE, 2014.