SCENE FLOW FOR AUTONOMOUS NAVIGATION

A Thesis
Presented to
The Academic Faculty

By

Vaibhav Dedhia

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2018

SCENE FLOW FOR AUTONOMOUS NAVIGATION

Approved by:

Dr. Patricio A. Vela, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ghassan AlRegib
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Mark Davenport
School of Electrical and Computer Engineering
Georgia Institute of Technology

This thesis is dedicated to Prachi Dosani, who has been my life since I met her.

ACKNOWLEDGEMENTS

First and foremost, I would first like to thank my thesis advisor Dr. Patricio A. Vela of the School of Electrical and Computer Engineering at Georgia Institute of Technology. He consistently assisted and steered me in the right direction whenever he thought I needed it. I sincerely hope I continue to have opportunities to interact with him for the rest of my career.

My sincere thanks also go to Justin Smith at IVALab who supported me throughout various research projects and my thesis with insightful comments and discussions. I cannot adequately express how thankful I am. This thesis would not have been possible without his intellectual contribution and guidance throughout my entire research period.

I would also like to thank the rest of my thesis committee members, Dr. Ghassan AlRegib and Dr. Mark Davenport for sharing their precious time.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Today, there are various different paradigms for vision based autonomous navigation: mediated perception approaches that parse an entire scene to make driving decision, a direct perception approach that estimates the affordance of driving that maps an input image to small number of key perception indicators that directly relate to the affordance of road/traffic state for driving. Also, deep learning models trained for specific tasks such as obstacle classification, detecting drivable spaces have been used as modules for autonomous navigation of vehicles.

Recent applications of deep learning to navigation have generated end-to-end navigation solutions whereby visual sensor input is mapped to control signals or to motion primitives. It is accepted that these solutions cannot provide the same level of performance as a global planner. However, it is less clear how such end-to-end systems should be integrated into a full navigation pipeline. We evaluate the typical end-to-end solution within a full navigation pipeline in order to expose its weaknesses. Doing so illuminates how to better integrate deep learning methods into the navigation pipeline.

For the thesis, we evaluate global path planning using sampling based path planning algorithms. Global planners assume that the world is static and location of obstacle is known. However, for autonomous navigation scenerio, this assumption does not hold true. A need arises to be able to detect the obstacles in the scene, localize them and then make appropriate changes to the decisions for navigation. We train Convolutional Neural Network based deep networks for object recognition that are very effective for detecting the objects in the scene such as vehicles, pedestrian etc. We also propose methods to track the objects in the scene in three dimensions thus effectively localizing the objects in the scene.

CHAPTER 1
INTRODUCTION

1.1 Overview

Autonomous vehicles have raised wide interest in recent years with various applications such as inspection, monitoring and mapping. To navigate autonomously, the vehicles need to detect and avoid 3D obstacles in real time. Various range sensors such as laser [1], stereo cameras [2], and RGBD depth cameras [3] could be used to build 3D map of the environment. A typical approach to vision based navigation problem is to use geometric 3D reconstruction techniques such as structure from motion (SfM) and simultaneous localization and mapping (SLAM). They usually track visual features to build sparse [4] or semidense 3D map [5]. However, monocular SLAM may not be robust especially in challenging low-texture or degenerated environments. Besides, their map is usually relatively sparse and cannot fully represent 3D obstacles. On the other hand, for the image in Fig. 1, a human can easily understand the 3D layout with obstacles and generate the best motion commands.

CNNs have revolutionized most of the detection and classification tasks. Prior to the widespread adoption of CNNs, most of detection and classification tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images. Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations. While CNNs with learned features have been in commercial use for over twenty years, their adoption has exploded in the last few years because of two recent

developments. First, large, labeled data sets such as the Large Scale Visual Recognition Challenge (ILSVRC) have become available for training and validation. Second, CNN learning algorithms have been implemented on the massively parallel graphics processing units (GPUs) which tremendously accelerate learning and inference.

To mimic humans behavior, there are also some learning based algorithms to predict paths directly from raw RGB image especially with Convolutional neural networks (CNNs) [6] [7] [8]. To train CNN models, a large dataset with motion command labels is needed. Previous work mainly utilized human demonstration to collect datasets with labels but it has some problems such as being time-consuming to collect large demonstration data and usually biased to certain scenarios where demonstrations happen.

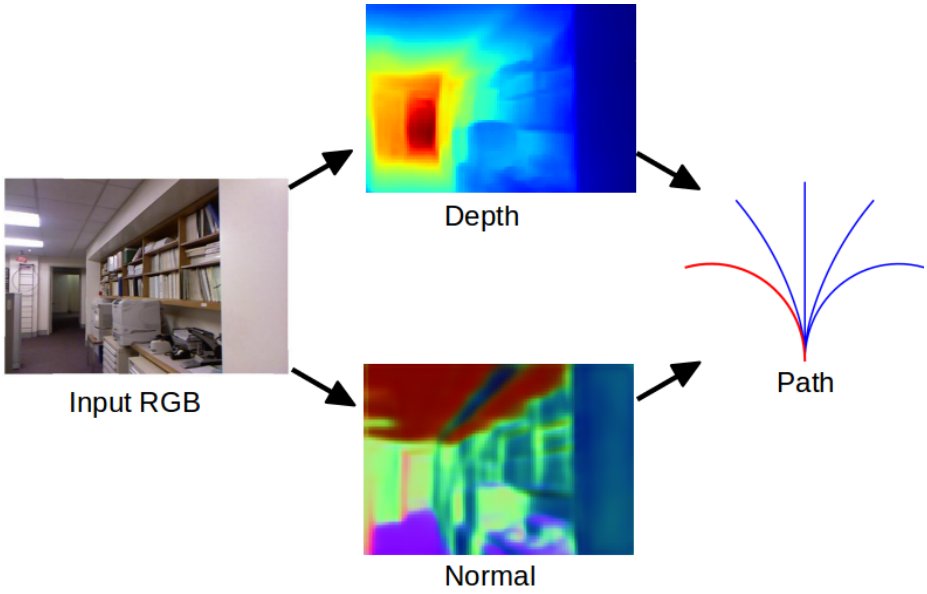

Figure 1.1: Results of Object Detection using Faster RCNN

For example, in Fig. 1.1, people might select straight forward or left turning based on their own analysis of risk and task requirements. They only predict the instantaneous steering command (left, right, etc.) so it requires the learning algorithm to predict frequently. It is also difficult to evaluate whether robots hitting obstacles or not as it depends on the how long the steering command last.

In this thesis, we propose an alternative approach for planning required for autonomous

robots. Instead of predicting a steering angle directly from the image, we use the images to recognize the objects in the scene as well as localize them. We use Object Detection techniques to detect different objects present in the scene and then use Scene Flow i.e 3D equivalent of Optical Flow. We use deep learning to train a model that we propose is more efficient as compared to traditional methods as it is free of any underlying assumptions. It can track objects over very large displacements and can handle scenes with varying illumination as well. We use stereoscopic image pair to estimate the flow and disparity in 3D. This also helps us tackle the problem of tracking objects even when they are partially occluded. This detection and tracking model can then be fed to path planner and controller to influence the path planning decisions. We propose that the resulting visual navigation strategies work well at collision avoidance and have more efficient performance as compared to traditional navigation algorithms while operating in real-time.

1.2 Thesis Organization

The research focuses on object detection strategies including Region Based CNN, Fast RCNN, Faster RCNN. Object detection models are trained, and their performance is tested on datasets. We report and compare the performances of each of these methods. For object tracking, the research covers the concepts of Optical Flow and Disparity to calculate the 3D Flow for the scene i.e Scene Flow.

Chapters 2 highlights the approaches used for sampling based path planning algorithms. Chapter 2 explains in detail the different path planning algorithms, integration of Kinect to capture depth images and implementation of these strategies of SRV robot for navigation task.

Chapter 3 discusses the problem of object detection and few limitation of traditional detection techniques. The chapter provides brief about the state of the art deep learning architectures used for object detection, highlights the architecture of the network used, along with the results obtained.

Chapters 4 explains object tracking methods using Optical Flow and Scene Flow. Chapter 4 introduces the reader to concepts of motion estimation technique called Optical Flow, which can be used for tracking objects. It points out limitations of traditional techniques and suggests deep learning based solutions that perform well. It then covers the concepts of disparity map for a stereoscopic image pair and methods to estimate disparity. The chapter concludes with integration of optical flow and disparity methods to estimate the Scene Flow.

The thesis concludes with Chapter 5 which discusses the inferences from the object detection and tracking methods explained in the previous chapters and proposes future work. Finally, the Appendix contains the code used for object detection and the C++ code used for DNN based object detection.

CHAPTER 2
BACKGROUND AND PREVIOUS WORK

2.1 Path Planning Algorithms

Path-planning is an important primitive for autonomous mobile robots that lets robots find the shortest or otherwise optimal path between two points. Otherwise optimal paths could be paths that minimize the amount of turning, the amount of braking or whatever a specific application requires. Path-planning is an important primitive for autonomous mobile robots that lets robots find the shortest or otherwise optimal path between two points. Otherwise optimal paths could be paths that minimize the amount of turning, the amount of braking or whatever a specific application requires. Path-planning requires a map of the environment and the robot to be aware of its location with respect to the map.

We explored the Global and Local path plannnig algorithms.Global planning involves using the initial position and the destination position, and then computing the path. The local planning involves dynamically scanning the environment for obstacles and avoiding it. We studied and implemented the discrete planning algorithms like Dijkstra's Algirithm, Breadth first, Depth first, A-Star, Iterative Deepening etc. We also explored sampling based path planning algorithms like Rapidly Exploring Random Trees (RRTs), Probabilistic Road Maps (PRMs) etc. Brief details of some of these algorithms is given below. Both type of algorithms can be implemented successfully using Single-Tree Search, Balanced Bi-directional search or with multiple trees. Though handling all the trees might be complex, this approach has a better chance at giving us a result when there are multiple bottlenecks or bug-traps in the Configuration Space. RRTs are used when we have a single query, whereas PRMs are used when we work with multiple queries.

2.1.1 Rapidly Exploring Random Trees

A Rapidly-exploring Random Tree (RRT) is algorithm that is designed for efficiently searching nonconvex high-dimensional spaces. RRTs are constructed incrementally in a way that quickly reduces the expected distance of a randomly-chosen point to the tree. RRTs are particularly suited for path planning problems that involve obstacles and differential constraints (nonholonomic or kinodynamic). RRTs can be considered as technique for generating open-loop trajectories for nonlinear systems with state constraints. RRT can be intuitively considered as a Monte-Carlo way of biasing search into largest Voronoi regions.

The algorithm involves sampling random points in the configuration space and it tries to connect the point to the nearest node of the existing tree without hitting or going through any Obstacle. It tries to connect to the goal every once in a while (if it is a unidirectional forward tree) and stops when it has reached the goal. Smoothening was implemented along with RRTs, which reduces the cost of reaching the goal by skipping few nodes and connecting to the next node which doesnt involve going through obstacles.

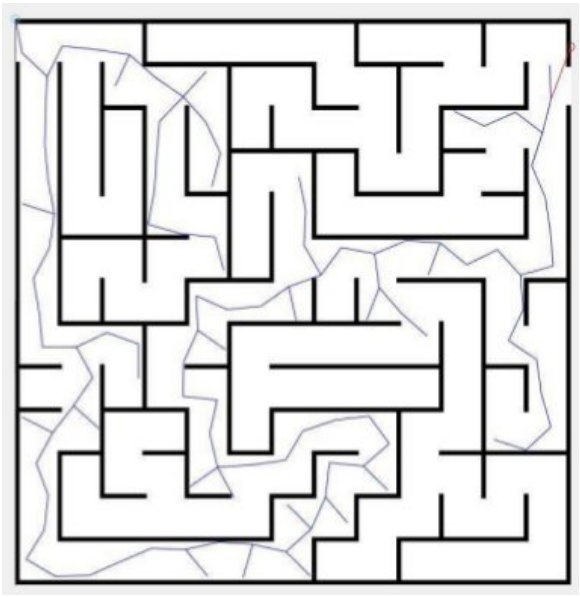

Figure 2.1: RRT applied on a Maze World

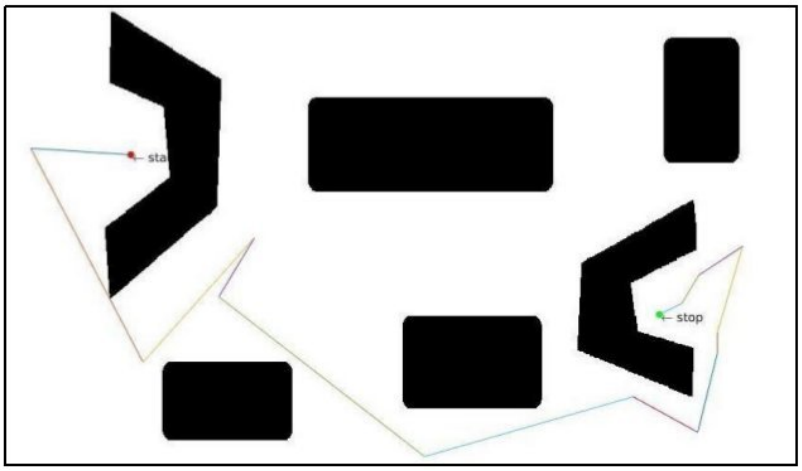

Figure 2.2: RRT applied to a world without smoothening

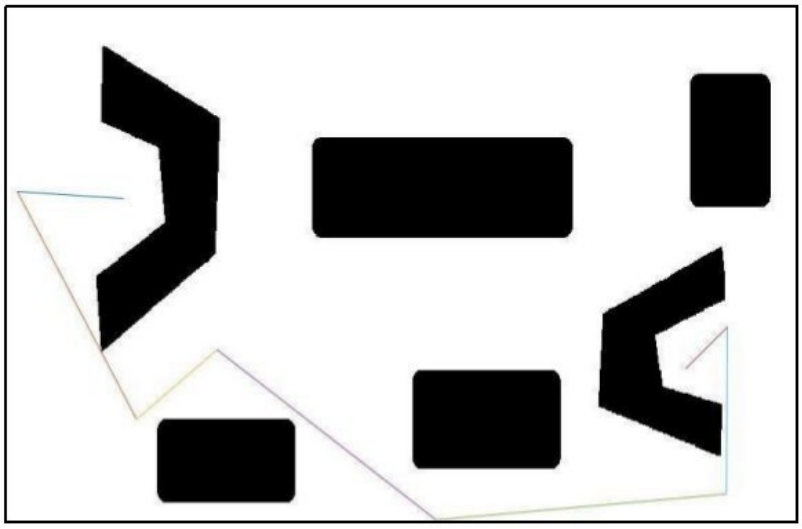

Figure 2.3: RRT applied to a world with smoothening

2.1.2 A* Algorithm

A* is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at

the predetermined goal node.

At each iteration of its main loop, A* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal node. Specifically, A* selects the path that minimizes

$$f(n) = g(n) + h(n) \tag{2.1}$$

where n is the last node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic that estimates the cost of the cheapest path from n to the goal. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node.

Here, we show the A* algorithm exploring the world and the best solution found for the given scene with some initial start position to reach the end position i.e goal.

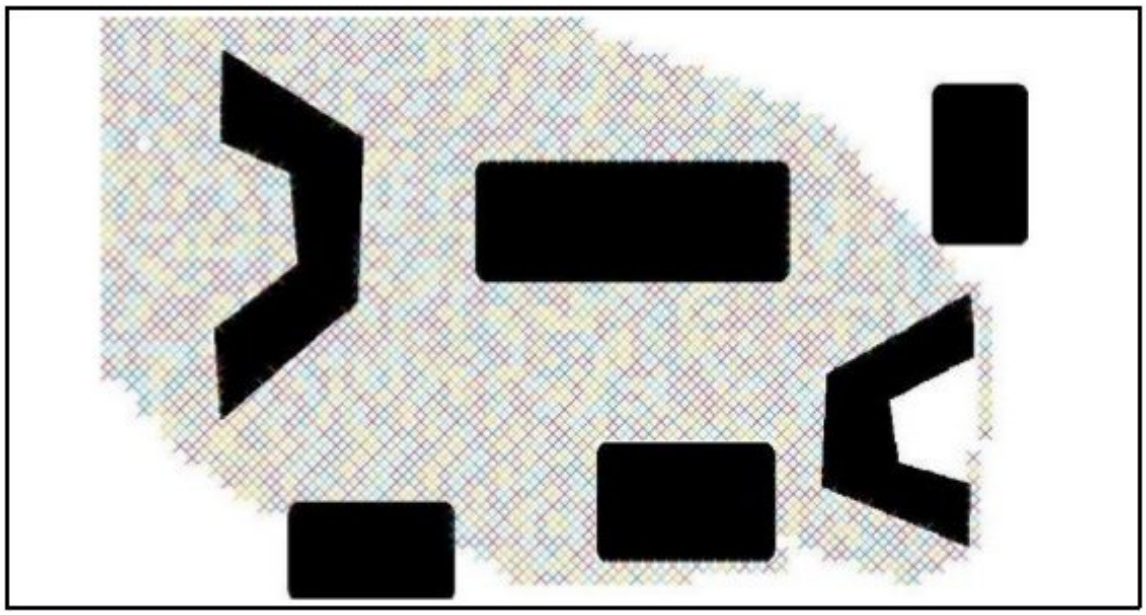

Figure 2.4: A* algorithm exploring in World

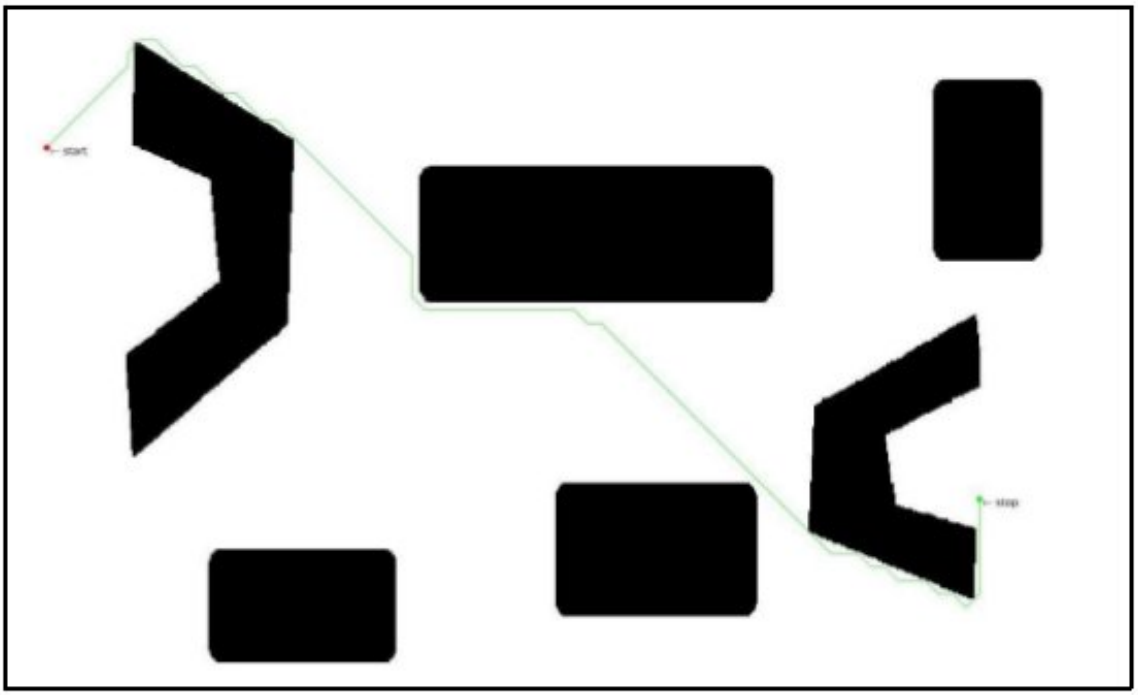

Figure 2.5: The best path recognized by A*

2.2 Integrating algorithms with Robot

After having the algorithms work for scenes generated on the matlab, we experimented the algorithms on real world scenes to test the algorithms in terms of its performance and to get better understading of their working. We implemented these algorithms on SRV bot with help of images obtained from overhead Kinect.

2.2.1 Working with kinect for Depth Images

The Kinect is a device which captures both Depth image and RGB image (both 640*480).As we were working indoors, Kinect was a good resource to use as an Overhead camera. We implemented the methods to read images with the Kinect using the Freenect Class. We used Cardboards, bins, barrels as the obstacles, and using the depth image and defining the range of values on the depth image, we obtained the obstacles that were at certain height in our world.

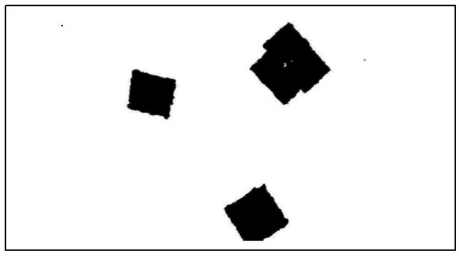

Figure 2.6: Depth Image from Kinect

Since the cameras used for Depth and RGB are not the same, we offseted one of the images to account for the shift in image. The overhead camera thus helped in understanding the world i.e location of various objects in the scene and to learn the position as well as orientation of the Bot.

2.2.2 SRV Robot

The Bot SRV-1 is a palm sized robot. It operates with a 500MHZ Blackfin BF537 processor and a mini video camera.

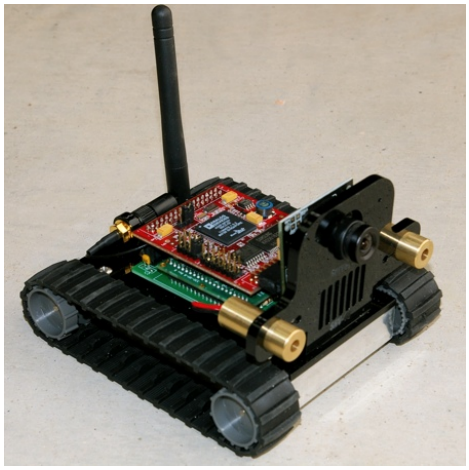

Figure 2.7: SRV Robot

Its accompanied with 802.11b/g Wi-Fi with which the SRV-1 interfaces wirelessly with a remote PC. It is initially setup using its serial port interface with the necessary information. We tweaked the JAVA code provided with the bot, inorder to enable us to control the Bot through Matlab.

2.2.3 Integration and Execution

We read the Depth and the RGB images from the Kinect, obtained the position obstacles and the Bot and the goal, passed the initial and goal positions to the path planner and obtained a feasible path from the initial to the goal position. The path is represented by an array data structure, containing series of consecutive vertices along the path.

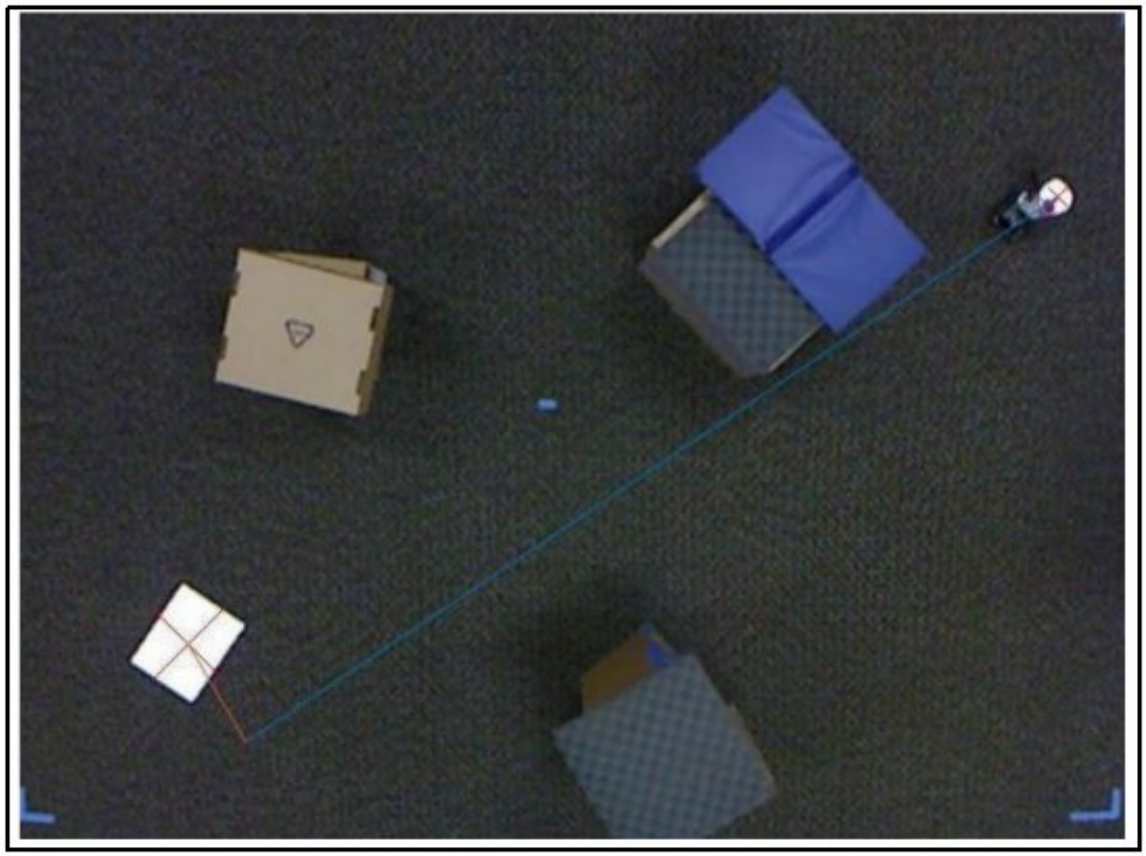

Figure 2.8: Path found with RRTs for the scene

Using the driveFor() method the robot is controlled by sending commands through the W-LAN using TCP/IP protocol. To control the Bot along the path, the current position of the Bot and the next vertex in the path is used. To drive the Bot to the next vertex, the Bot is first rotated towards the vertex and then translated towards it in steps. Separate methods for Translation and Orientation is made. The new position of the Bot is obtained by updating the image from the overhead camera and necessary adjustments are made to

the command we give to each wheel using the driveFor() method. This process is repeated till the destination is reached.

The SRV robot is then controlled via Matlab to follow this path.

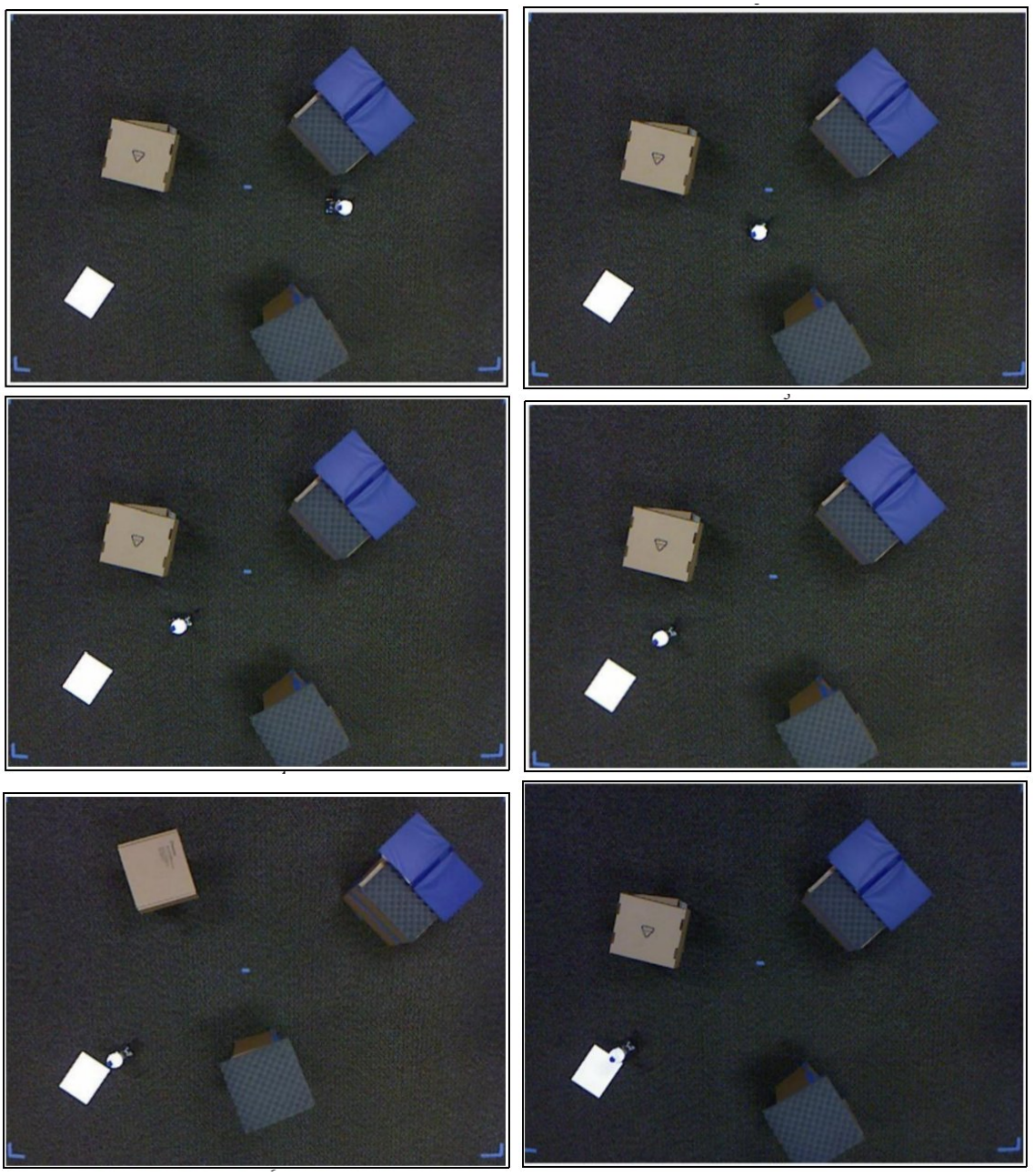

Figure 2.9: Robot following the path to reach the goal

CHAPTER 3
METHODS FOR OBJECT DETECTION

For path planning algorithm in the previous chapter we assumed that the scene was static i.e objects in the scene were not moving and we already knew where they were located. This is not the case in real scenes. While driving, we have other objects in the scene that can effect the driving decision. Objects include but are not restriced to pedestrian, other vehicles, animals etc. We also need to follow traffic signals, road signs and react accordingly. So, instead of relying on global information we need to actively look for such objects and localize them so that our agent can react accordingly.

Object detection deals with finding instances of real-world objects, while handling variation with respect to the objects shape, orientation and motion. Object detection algorithms typically take in a raw image as input and output the bounding box coordinates for the detected object. Object detection plays a significant role in programming autonomous vehicles, as it helps the vehicle perceive the environment just like a human driver would do. Originally, a lot of techniques like Object Detection using Cascade [9] , Histogram of Oriented Gradients [10] used classical machine learning for detection. These methods encoded a very low level characteristics of the objects and therefore were not able to distinguish well among the different labels. The processing time for each of them was not very reliable for high speed detection as required for complex task like autonomous driving. Also, the detection rate achieved with these algorithms was very low.

Deep learning (Convolutional networks) based methods have become the state of the art in object detection in image. They construct a representation in a hierarchial manner with increasing order of abstraction from lower to higher levels of neural network. Starting from Girshick et. al. paper, a flurry of papers have been published which have either focused on improving run-time efficiency or accuracy. These have used more or less the same pipeline

involving CNN(convolutional neural network)as feature extractor. In this chapter we look at some of the challenges that are to be tackled will detecting objects and then show results with different algorithms that we tested for purpose of object detection.

3.1 Challenges for Object Detection

Object detection combines the tasks of classification and localization, and outputs the object along with its pixel coordinates in the input image. This poses a few challenges as,

- **Varying number of objects:** The number of objects detected in an input image is not predefined, and due to this the output of the detector cannot be a vector of a fixed size. Traditional machine learning approaches use a sliding window detector to find all the objects in the image.
- **Size of the objects:** In image classification tasks, the biggest object in the image is classified into one of the known categories of objects. As opposed to this, objects of varying sizes must be detected in a detection task. In traditional machine learning approaches, sliding windows of varying sizes are used to detect objects in various scales.
- **Architecture** An object detection task combines object classification and localization. The architecture used for detecting objects must thus include the requirements of both these tasks into consideration and a combined model must be created.

3.2 Different Methods for Object Detection

In recent times, lot of different methods for object recognition using deep learning has been proposed that perform well on the detection task. One of the main ideas behind deep learning is automatic feature extraction. Deep learning aims to automatically learn the features from noisy data - this is a major challenge for traditional machine learning and the performance of a machine learning algorithm drops significantly with noisy data. A

deep network is usually made up of an input layer, many hidden layers that learn the high-level features using some activation functions, followed by an output layer. While training, random values are assigned as the networks weights, and the network learns the optimal weights using an algorithm called the backpropagation.

One of the earliest methods is the OverFeat [11], published in 2013. It shows how a multi-scale sliding window can be used efficiently in a convolutional neural network (ConvNet) to detect objects. Soon after OverFeats publication, Ross Girshick published the R-CNN [12] (Region based Convolutional Neural Networks) and this approach had nearly a 50% improvement to OverFeat. In this approach, possible objects and regions are extracted and then the regions are classified. R-CNN evolved into Fast R-CNN[9] in which a CNN is applied to the entire image instead of extracting the object and region proposals separately. To make process significantly faster, the Faster R-CNN [13] was then introduced. Instead of the selective search that was used in the previous two R-CNN approaches for region proposals, the Faster R-CNN uses a Region Proposal Network' which makes the entire model completely trainable end-to-end. Another object detection approach, You Only Look Once (YOLO) [14] uses a single CNN to predict the bounding boxes and class probabilities from the input image. This method has a great inference time and allows object detection to be performed in real-time. SSD [15] (Single Shot Multi-box detection) is a method that improved upon YOLO and used multiple sized convolutional feature maps while R-FCN (Region based Fully Convolutional Network) improved upon Faster RCNN by using only convolutional layers.

Here we will look at three methods that worked well for classes particular for the autonomous driving scenario:

- **Region Based CNN** After the rise of deep learning, the obvious idea was to replace HOG based classifiers with a more accurate convolutional neural network based classifier. However, there was one problem. CNNs were too slow and computationally very expensive. It was impossible to run CNNs on so many patches generated by

sliding window detector. R-CNN solves this problem by using an object proposal algorithm called Selective Search which reduces the number of bounding boxes that are fed to the classifier to close to 2000 region proposals.

- **Fast RCNN** Still, RCNN was very slow. Because running CNN on 2000 region proposals generated by Selective search takes a lot of time. Fast RCNN fixed this limitation by calculating the CNN representation for entire image only once and then using this representation to calculate the effective CNN representation for each patch generated by Selective Search. This can be done by performing a pooling type of operation on just that section of the feature maps of last conv layer that corresponds to the region. It increased the speed by a huge factor

- **Faster RCNN** Though Fast RCNN increased the speed by a factor of 10, it was still slow for real time object detection. Slowest part in Fast RCNN was Selective Search or Edge boxes. Faster RCNN replaces selective search with a very small convolutional network called Region Proposal Network to generate regions of Interests.

3.2.1 Region based CNN

The goal of R-CNN is to take in an image, and correctly identify where the main objects in the image.

- **Inputs**: Image
- **Outputs**: Bounding boxes + labels for each object in the image.

It propose a bunch of boxes in the image and see if any of them actually correspond to an object. R-CNN creates these bounding boxes, or region proposals, using a process called Selective Search. Selective Search looks at the image through windows of different sizes, and for each size tries to group together adjacent pixels by texture, color, or intensity to identify objects.

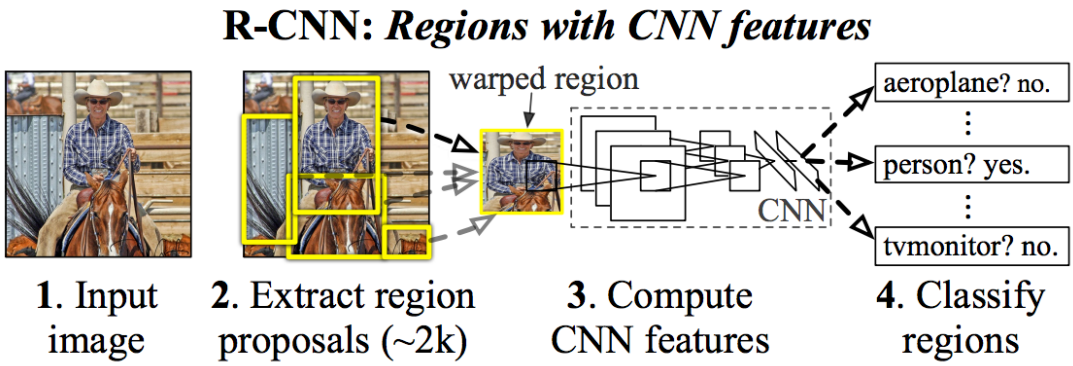

Figure 3.1: After creating a set of region proposals, R-CNN passes the image through a modified version of AlexNet to determine whether or not it is a valid region. Source: https://arxiv.org/abs/1311.2524.

Once the proposals are created, R-CNN warps the region to a standard square size and passes it through to a modified version of AlexNet. On the final layer of the CNN, R-CNN adds a Support Vector Machine (SVM) that simply classifies whether this is an object, and if so what object. Having found the objects for each proposed bounding box, R-CNN runs a simple linear regression on the region proposal to generate tighter bounding box coordinates to get our final result.

3.2.2 Fast R-CNN

R-CNN works really well, but is really quite slow for a few simple reasons:

- **Region Proposal**: It requires a forward pass of the CNN (AlexNet) for every single region proposal for every single image (thats around 2000 forward passes per image!).

- **models**: It has to train three different models separately - the CNN to generate image features, the classifier that predicts the class, and the regression model to tighten the bounding boxes. This makes the pipeline extremely hard to train.

Fast R-CNN, solved both these problems by proposing the uses of RoI (Region of Interest) Pooling. During the forward pass of the CNN for each image, a lot of proposed regions for the image invariably overlapped causing us to run the same CNN computation again and again (2000 times!). Instead of running the CNN on image multiple times, we can run the CNN just once per image and then find a way to share that computation across the 2000 proposals using RoIPool (Region of Interest Pooling).

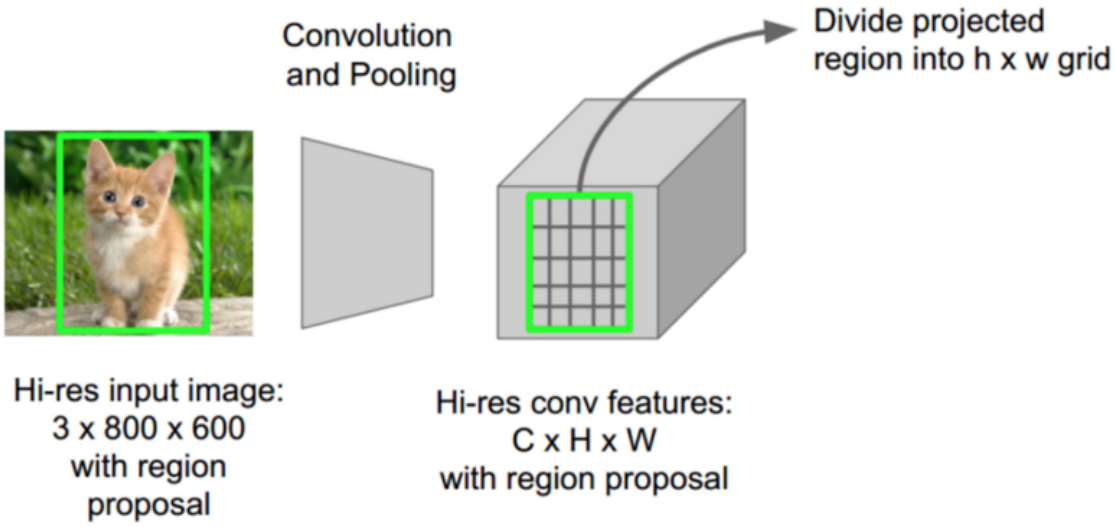

Figure 3.2: In RoIPool, a full forward pass of the image is created and the conv features for each region of interest are extracted from the resulting forward pass. Source: Stanfords CS231N slides by Fei Fei Li, Andrei Karpathy, and Justin Johnson.

At its core, RoIPool shares the forward pass of a CNN for an image across its subregions. In the image above, notice how the CNN features for each region are obtained by selecting a corresponding region from the CNNs feature map. Then, the features in each region are pooled (usually using max pooling). So all it takes us is one pass of the original image as opposed to 2000! Fast R-CNN also jointly trains the CNN, classifier, and bounding box regressor in a single model. Where earlier we had different models to extract image features (CNN), classify (SVM), and tighten bounding boxes (regressor), Fast R-CNN instead used a single network to compute all three.

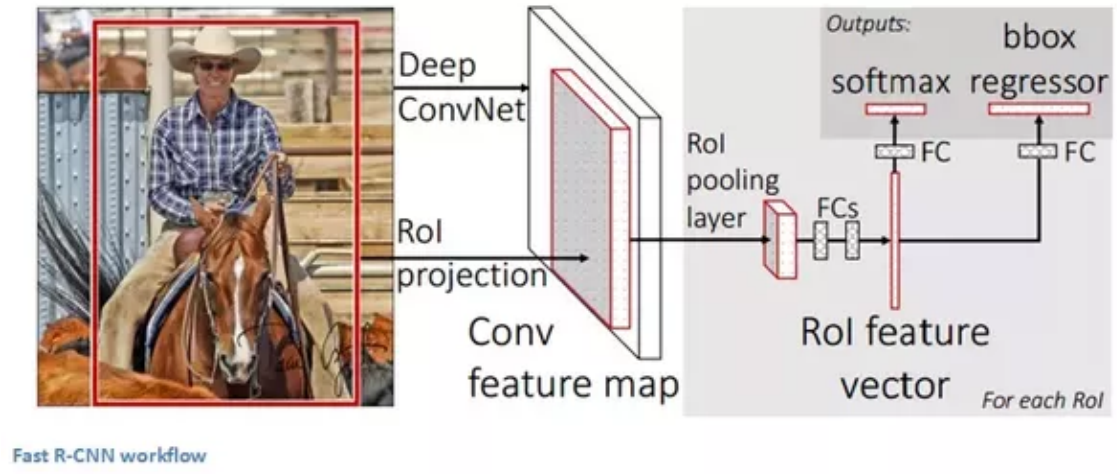

Figure 3.3: Fast R-CNN combined the CNN, classifier, and bounding box regressor into one, single network.
Source: https://www.slideshare.net/simplyinsimple/detection-52781995.

Fast R-CNN replaced the SVM classifier with a softmax layer on top of the CNN to output a classification. It also added a linear regression layer parallel to the softmax layer to output bounding box coordinates. In this way, all the outputs needed came from one single network. The Fast RCNN method offers multiple advantages over RCNN which can be listed as:

- The Fast R-CNN performs tasks of Object Detection 10 times faster as compared to R-CNN.
- Results in faster training and efficient back-propagation.

3.2.3 Faster R-CNN

Even with all these advancements, there was still one remaining bottleneck in the Fast R-CNN processŁŁthe region proposer. As we saw, the very first step to detecting the locations of objects is generating a bunch of potential bounding boxes or regions of interest to test. In Fast R-CNN, these proposals were created using Selective Search, a fairly slow process that was found to be the bottleneck of the overall process.

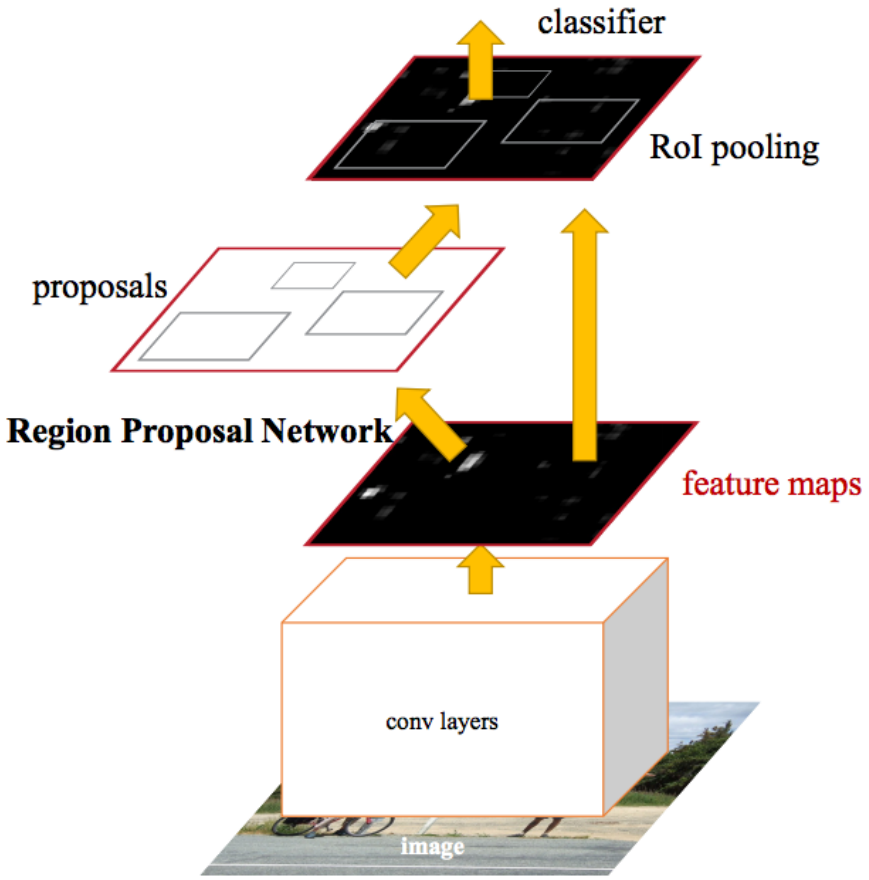

Figure 3.4: Faster R-CNN has single CNN for region proposals and classifications. Source: https://arxiv.org/abs/1506.01497.

Faster R-CNN provides a method to make the region proposal step almost cost free through use of Region Proposal Network. The insight of Faster R-CNN was that region proposals depended on features of the image that were already calculated with the forward pass of the CNN (first step of classification).

Here is how the Region Proposal Network works

- At the last layer of an initial CNN, a 3x3 sliding window moves across the feature map and maps it to a lower dimension (e.g. 256-d).
- For each sliding-window location, it generates multiple possible regions based on k fixed-ratio anchor boxes (default bounding boxes).

- Each region proposal consists of a) an objectness score for that region and b) 4 coordinates representing the bounding box of the region

We look at each location in our last feature map and consider k different boxes centered around it: a tall box, a wide box, a large box, etc. For each of those boxes, we output whether or not we think it contains an object, and what the coordinates for that box are. This is what it looks like at one sliding window location:

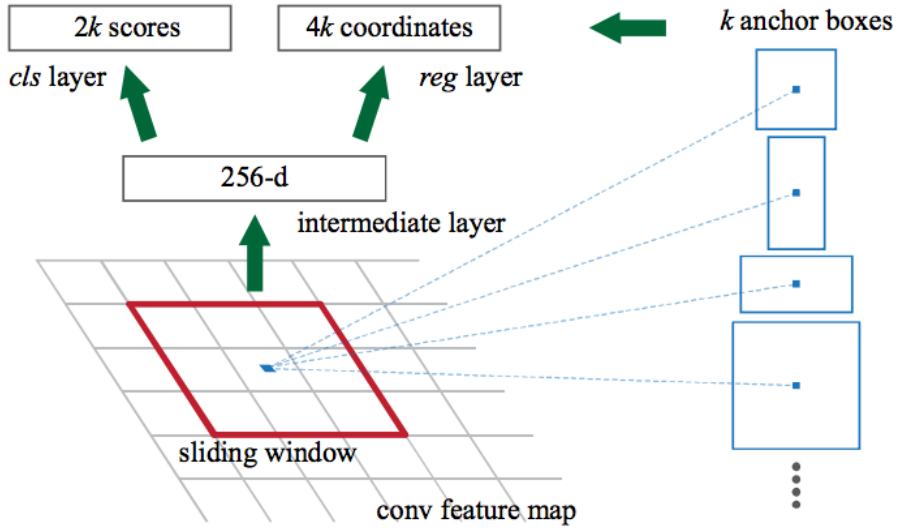

Figure 3.5: Sliding window approach. Source: https://arxiv.org/abs/1506.01497.

The 2k scores represent the softmax probability of each of the k bounding boxes being on object. Notice that although the RPN outputs bounding box coordinates, it does not try to classify any potential objects: its sole job is still proposing object regions. If an anchor box has an objectness score above a certain threshold, that boxs coordinates get passed forward as a region proposal.

Once we have our region proposals, we feed them straight into what is essentially a Fast R-CNN. We add a pooling layer, some fully-connected layers, and finally a softmax classification layer and bounding box regressor.

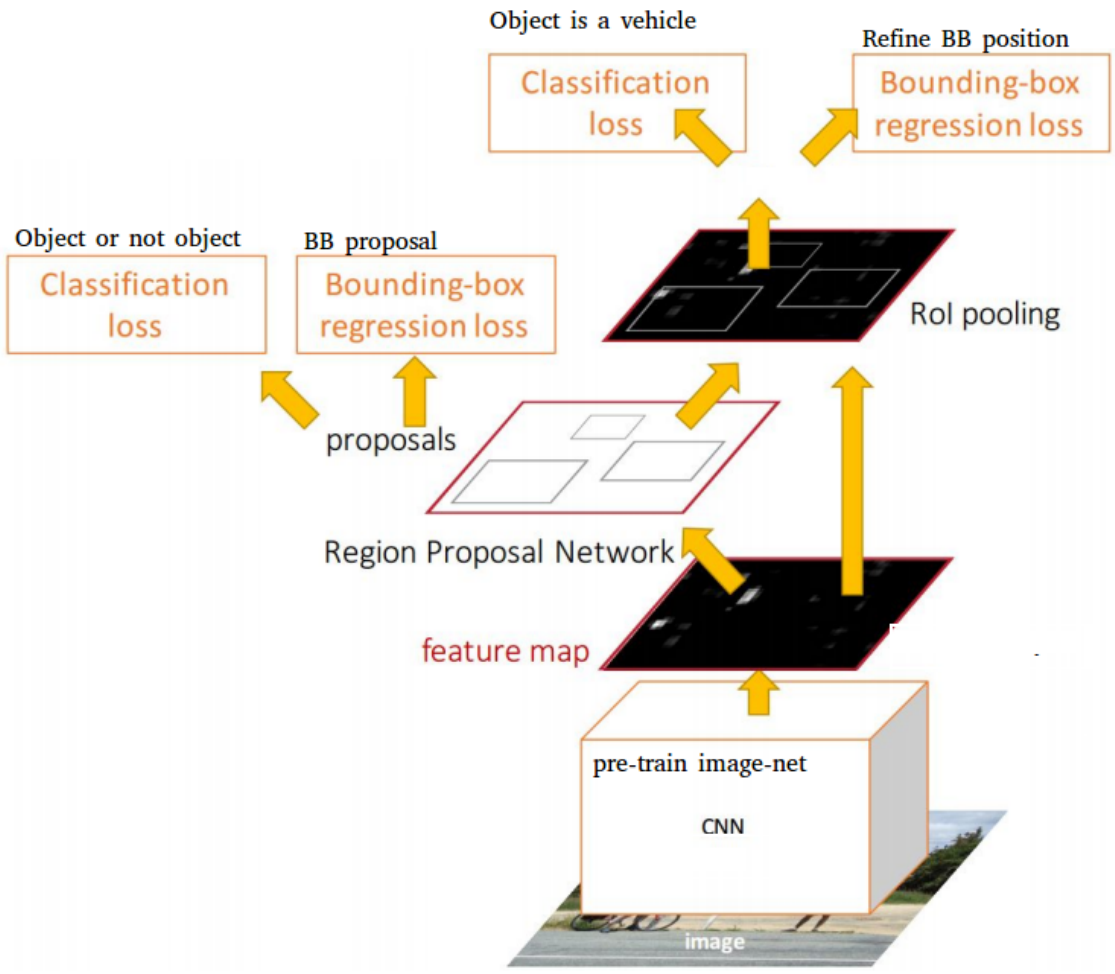

Figure 3.6: Architecture of Faster R-CNN. Source: https://arxiv.org/abs/1506.01497.

3.3 Results for Object Detection

We trained models using Fast RCNN and Faster RCNN. Though the training time for Faster RCNN was significantly more as compared to that of Fast RCNN, it performs detections 10 times faster than Fast RCNN. Hence, we choose Faster RCNN as the object detection method for our purposes.

Here we show the results obtained using Faster RCNN for test images on KITTI Vision Benchmark Suite and Oxfords Robotic Car dataset.

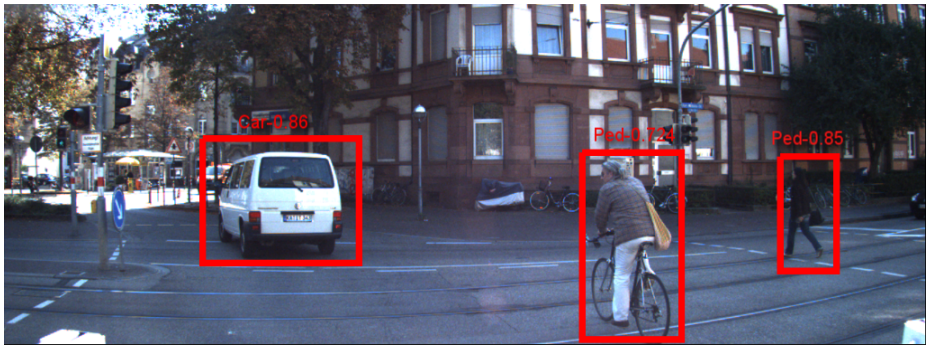

Figure 3.7: Results of Object Detection using Faster RCNN

The figure above shows various objects in the image detected. The model predicts that the object to the leftmost is a Car with probability of 0.86. Similarly other two objects are classified as pedestrian with a probability of 0.724 and 0.85 respectively. As can be seen the bounding boxes are quite tight around the object detected.

Figure 3.8: Results of Object Detection using Faster RCNN

Here, the model detects the cars with probability of 0.791 and 0.84 respectively.

Here we show the compare the performances of these methods:

	R-CNN	Fast R-CNN	Faster R-CNN
Test Time per Image	50 Seconds	2 Seconds	0.2 Seconds
Speed Up	1x	25x	250x

Figure 3.9: Time comparision between three methods.

3.4 Limitations of Object Detection

As can be seen from the examples above, the Object Detection methods represent the detection hypothesis using a bounding box, which denotes the area in image the detector thinks an object is present. While a rectangular is the simplest way to indicate a region of the image, it is not optimal for many applications. If the object itself is not rectangular, not all the pixels in the bounding box belong to the object. The detection method fails to detect objects if they are occluded. Here we show some of the many cases we encountered when the detection method failed to detect occluded objects:

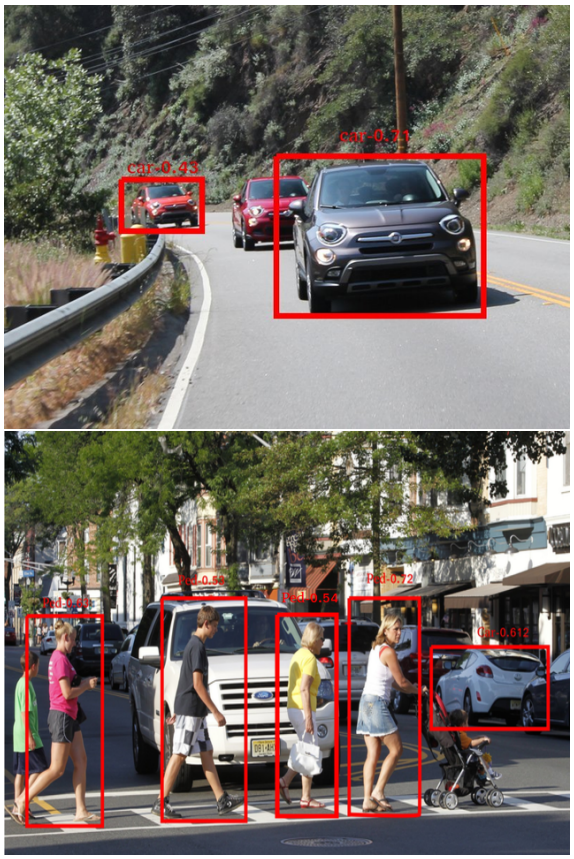

Figure 3.10: Examples where Object Detection fails to detect objects

In the top image, two cars are detected while the middle car is not detected. Also the bounding box of the first car is not tight enough. In the bottom image, although the pedestrians are detected, the occluded car does not get detected.

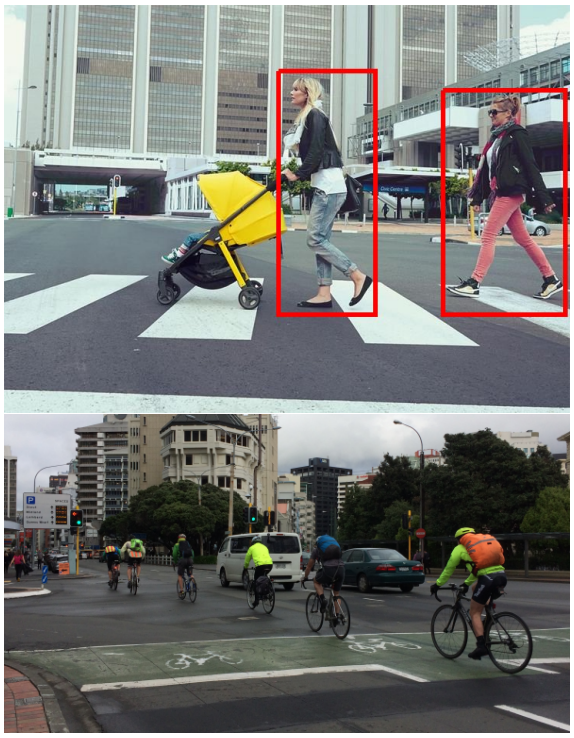

Figure 3.11: Examples where Object Detection fails to detect objects

In the top image, two people crossing the road are detected while the middle baby stroller is not detected. In the bottom image, although lot of cyclist and cars are present, they are not detected.

CHAPTER 4
SCENE FLOW USING DEEP LEARNING

While object detection is an important task, it requires lot of examples of object that we want to detect. If we want to detect an object, we need lots of different examples of that object with different poses to be able to handle detection for cases when we are looking at object from different viewpoints. In-class variance adds more complexity to the learning model and much more richer dataset is required inorder to train a model with all these constraints. For instance, if we are training a model that can detect car, it should be able to handle cases when the car is facing directly at the camera and when is it being looked at from the side view. Car can be partially blocked or it can viewed at from different angle. Also, there are different type of cars like hatchback, sedan, SUV and differnt companies have diffeernt builds adding lot of variance. More the variance, more is the data required for training a successful model. While deep learning has made it possible to handle a lot of complexities, data collection and training is a cumbersome task and resultant model might not be best at detection.

Instead of relying on object detection for detecting obstacles, understanding the geometry and motion within urban scenes, using either monocular or stereo imagery can help understand the scene better and is relevant for autonomous driving. Optical flow has been typically used for motion estimation. Here, we see limitations of optical flow and describe the Scene Flow method which gives us more accurate estimate of geometry and motion.

4.1 Optical Flow

One of the traditional ways of tracking objects is using Optical Flow. Optical Flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movemement of object or camera. It is 2D vector field where each vector is a displacement

vector showing the movement of points from first frame to second. Other than tracking, it has many applications in areas like structure from motion, video stabilization, video compression etc.

Consider a pixel $I(x, y, t)$ in first frame. It moves by distance (dx, dy) in next frame taken after dt time. So since those pixels are the same and intensity does not change, we can say,

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

Then take taylor series approximation of right-hand side, remove common terms and divide by dt to get the following equation:

$$f_x u + f_y v + f_t = 0$$

where:

$$f_x = \frac{\partial f}{\partial x} \quad ; \quad f_y = \frac{\partial f}{\partial y}$$

$$u = \frac{\mathrm{d}x}{\mathrm{d}t} \quad ; \quad v = \frac{\mathrm{d}y}{\mathrm{d}t}$$

Above equation is called Optical Flow equation. In it, we can find f_x and f_y, they are image gradients. Similarly f_t is the gradient along time. But (u, v) is unknown. We cannot solve this one equation with two unknown variables.

There are several methods to calculate the problem of optical flow:

- Lucas-Kanade Method
- Horn-Schunck Method
- Farneback

All of these methods have some underlying assumptions upon which they are built. These assumptions lead to shortcomings with regards to the task of object tracking:

- **Illumination**

  Changes in illumination can greatly affect the flow vectors and will lead to the creation of flow vectors at postions without motion. Since the method assumes brightness constancy over the image sequence, objects cannot be tracked when there is an illumination change in the images.

- **Large Movements**

  Large movements may not be detected correctly because they produce great displacements instead of smooth motion. To detect large movements, either the frame rate must be increased, or the image size must be decreased.

- **Occlusion**

  Since optical flow tracks only the apparent movement of pixels between frames, the object is not tracked if it is occluded behind another object. It depends only on the motion of pixels, and hence does not detect and track an occluded object.

- **Movement of Camera and Object**

  This method works well when either the camera or the object is in motion, but not when there is a relative movement in both the camera and the object. For a real-time vehicle tracking system, this method cannot be used to effectively track other objects from a moving vehicle since the motion of pixels in the entire frame will be large.

4.2 Scene Flow for Tracking Objects

To overcome the shortcomings of Optical Flow using the traditional techniques, we propose use of Scene Flow using Deep Learning. Scene flow is the three-dimensional motion field of points in the world, just as optical flow is the two-dimensional motion field of points in an image. Any optical flow is simply the projection of the scene flow onto the image plane

of a camera. One representation of the scene motion is therefore a dense three-dimensional vector field defined for every point on every surface in the scene.So, we have both depth and motion (equivalently 2.5D motion) for each image pixel - as opposed to 2D motion in image plane for the case of optical flow. Instead of 2D metric values, we can use a pixel-space formulation to decouple the task from camera intrinsics.

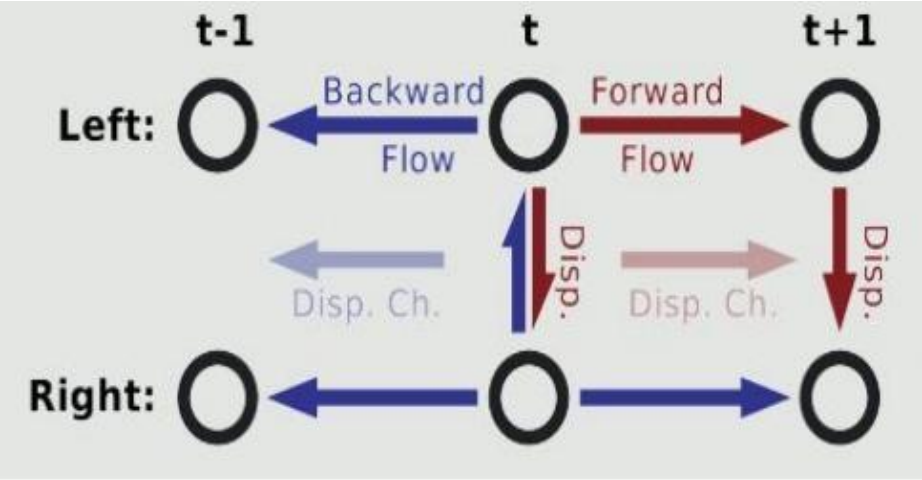

Figure 4.1: Scene Flow Design.

4.3 Components of Scene Flow

The Scene Flow can be split up into:

- Optical Flow
- Disparities
- Disparity changes

The last component fills gaps left by occlusions and creates a dense map of Scene Flow. The final architecture for Scene Flow is made up of two base architectures

- FlowNet
- DispNet

4.3.1 FlowNet: Estimate Optical Flow

FlowNet is a Convolutional Neural Network (CNN) based deep learning technique that learns to estimate the optical flow for the given sequence of images. The inputs to the model are image frames at time $(t-1)$ and time t and it estimates the displacement of all the pixels in the scene.

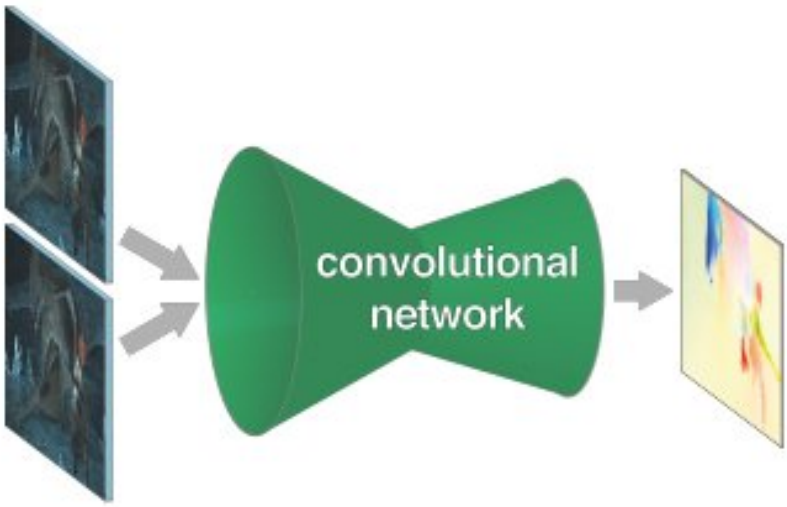

Figure 4.2: FlowNet which learns to estimate optical flow, being trained end-to-end

While optical flow estimation needs precise per-pixel localization, it also requires finding correspondences between two input images. This involves not only learning image feature representations, but also learning to match them at different locations in the two images. It differs from other standard CNN based models like AlexNet as it uses a correlation layer that explicitly provides matching capabilities. The idea is to exploit the ability of convolutional networks to learn strong features at multiple levels of scale and abstraction and to help it with finding the actual correspondences based on these features. The layers on top of the correlation layer learn how to predict flow from these matches.

Training such a network to predict generic optical flow requires a sufficiently large training set. Although data augmentation does help, the existing optical flow datasets are still too small to train a network on par with state of the art. Thus, we use synthetically

generated dataset, Flying Chairs dataset which consists of random background images on which we overlay segmented images of chairs. These data have little in common with the real world, but CNNs trained on just these data generalize well to realistic datasets, even without fine-tuning.

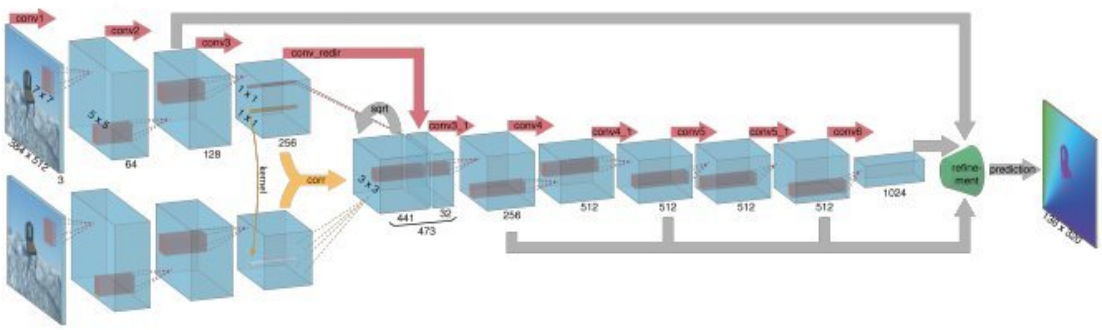

Figure 4.3: FlowNet Architecture

The images are feed to two separate, yet identical processing streams and are combined at a later stage as shown in figure 4.3. With this architecture the network is constrained to first produce meaningful representations of the two images separately and then combine them on a higher level. To aid the network in finding the correspondences between two images a correlation layer that performs multiplicative patch comparisons between two feature maps is used. Given two multi-channel feature maps f_1 , f_2 : $R_2 \rightarrow R_c$, with w, h, and c being their width, height and number of channels, our correlation layer lets the network compare each patch from f_1 with each path from f_2.

Dataset

Unlike traditional approaches, neural networks require data with ground truth not only for optimizing several parameters, but to learn to perform the task from scratch. In general, obtaining such ground truth is hard, because true pixel correspondences for real world scenes cannot easily be determined. We use Middlebury dataset, KITTI dataset, MPI Sintel dataset for training. Middlebury dataset contains only 8 image pairs for training, with ground truth

flows generated using four different techniques. Displacements are very small. The KITTI dataset [14] is larger (194 training image pairs) and includes large displacements, but contains only a very special motion type. The ground truth is obtained from real world scenes by simultaneously recording the scenes with a camera and a 3D laser scanner. This assumes that the scene is rigid and that the motion stems from a moving observer. Moreover, motion of distant objects, such as the sky, cannot be captured, resulting in sparse optical flow ground truth. These dataset are too small to train large CNNs. To provide enough training data, we use a simple synthetic dataset named Flying Chairs.

Training Details

The exact architectures of the networks we train are shown in Fig. 4.3. Overall, we try to keep the architectures of different networks consistent: they have nine convolutional layers with stride of 2 (the simplest form of pooling) in six of them and a ReLU nonlinearity after each layer. We do not have any fully connected layers, which allows the networks to take images of arbitrary size as input. We choose Adam as optimization method because for our task it shows faster convergence than standard stochastic gradient descent with momentum. We fix the parameters of Adam: $\beta 1 = 0.9$ and $\beta 2 = 0.999$. Since, in a sense, every pixel is a training sample, we use fairly small mini-batches of 8 image pairs. We start with learning rate λ = 1e4 and then divide it by 2 every 100k iterations after the first 300k. As training loss we use the endpoint error (EPE), which is the standard error measure for optical flow estimation. It is the Euclidean distance between the predicted flow vector and the ground truth, averaged over all pixels.

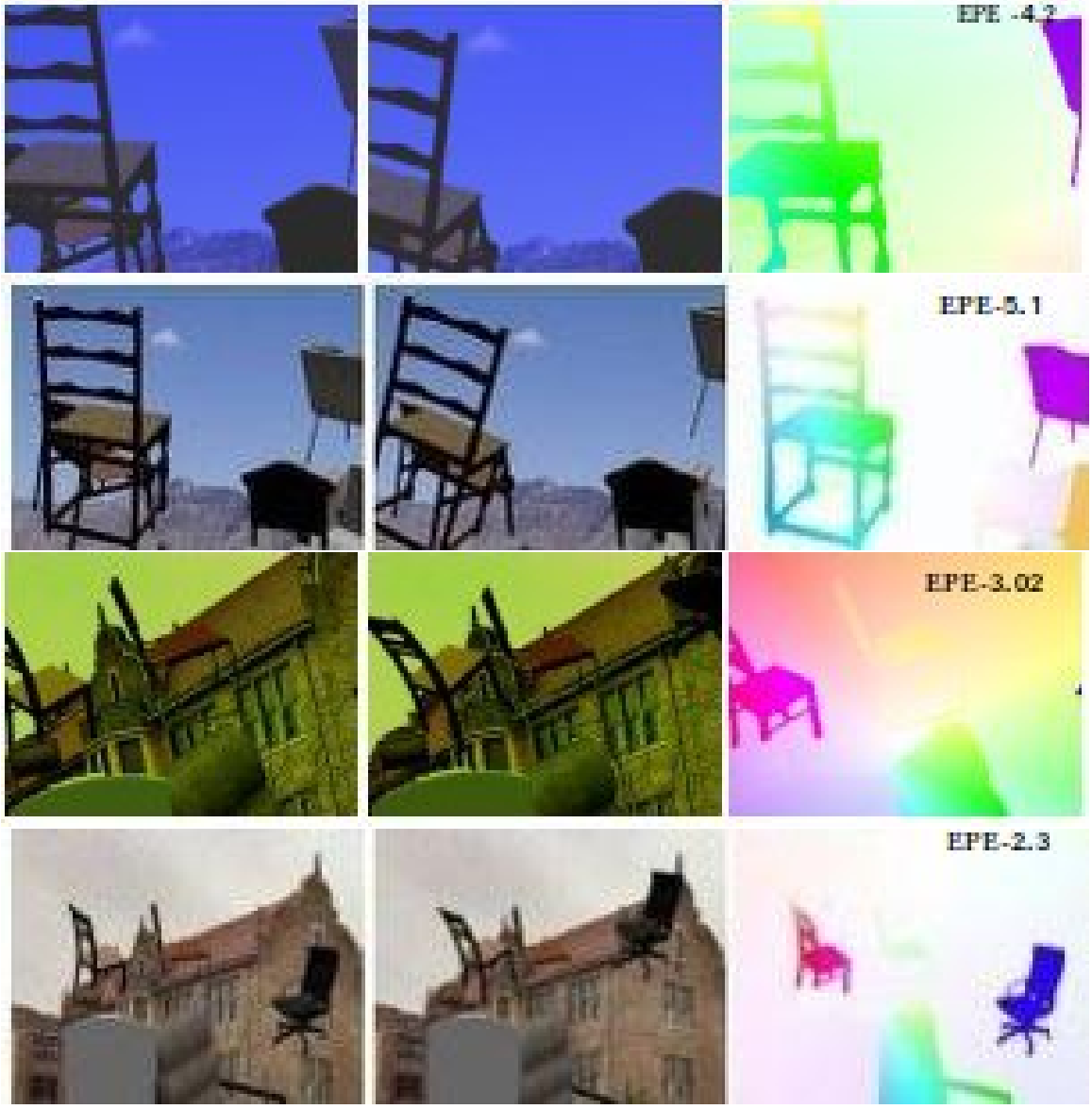

Figure 4.4: Optical Flow estimation using FlowNet for examples from Simulated Dataset. The first and the second column show the left and right images, while th third coumn shows corresponding color coded flow field. First two rows are examples from simulated dataset, while last rows show the augmented examples.

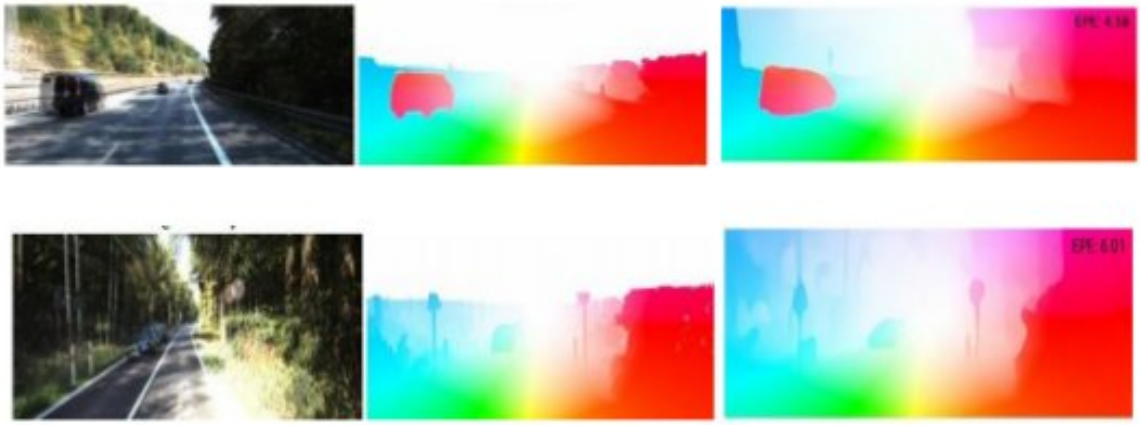

Figure 4.5: Optical Flow Estimation for Real Scenes using FlowNet. The first column shows consecutive images overlayed on each other, the second column shows the Groundtruth Optical Flow and third column shows the Optical Flow estimated using FlowNet. Average End Point Error(EPE) of 4.2 for this example

Results

We evaluated all methods on a system with an Intel Xeon E5 with 2.40GHz and an Nvidia GTX 1080. Our model was evaluated on test data from Sintel, KITTI and Middlebury datasets. We achieved 6.07 EPE on Sintel, 8.26 EPE on KITTI and 3.87 on Middlebury dataset. The per frame runtime of our model was 1.05 secs.

4.3.2 DispNet: Estimate Disparity

DispNet is a Convolutional Neural Network (CNN) based deep learning technique that learns to estimate the disparity for the given pair of stereoscopic images. Disparity refers to the distance between two corresponding points in the left and right image of a stereo pair. The inputs to the model are right and left image frames and it generates the disparity map for entire image.

The model that we use to train the DispNet is very similar to what we used for FlowNet as in both cases we need to learn the low level features and then learn some correspondences between two images. The only difference is that we expect it to learn displacement for every pixel in frames at time $t-1$ and time t in case of FlowNet and disparity between

corresponding pixels in stereo pair in case of DispNet.

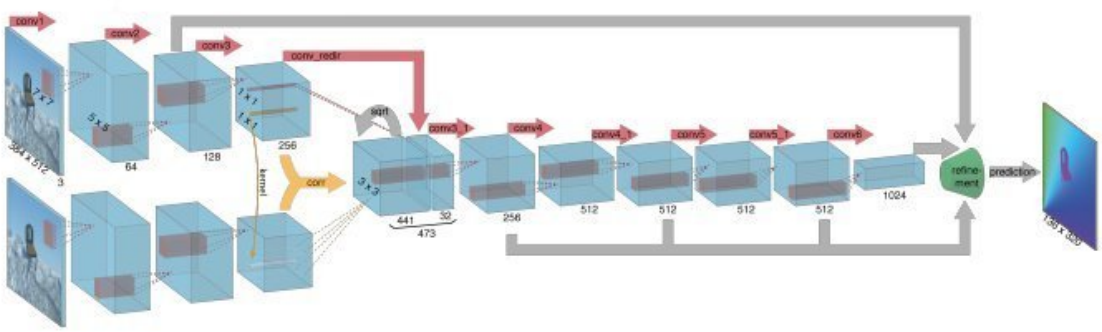

Figure 4.6: DispNet Architecture

Training Details

As the architecure of the DispNet is similar to that of FlowNet, we use similar training parameters. We choose Adam as optimization method because for our task it shows faster convergence than standard stochastic gradient descent with momentum. We fix the parameters of Adam: $\beta 1 = 0.9$ and $\beta 2 = 0.999$. Since, in a sense, every pixel is a training sample, we use fairly small mini-batches of 8 image pairs. We start with learning rate λ = 1e4 and then divide it by 2 every 100k iterations after the first 300k. As training loss we use the endpoint error (EPE), which is the standard error measure for disparity estimation.

Results

The results obtained for the DispNet are shown below:

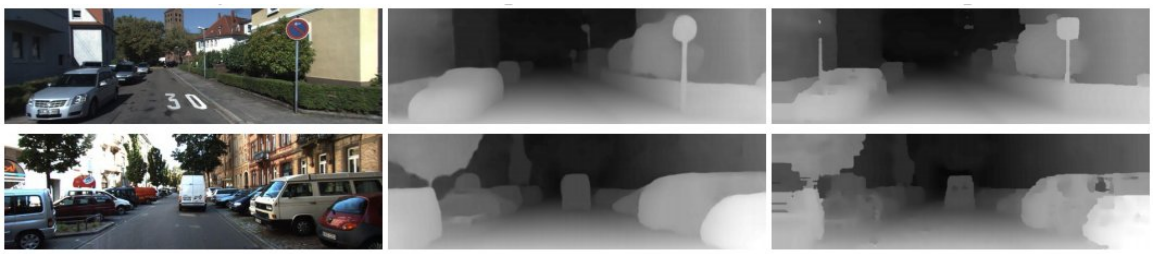

Figure 4.7: Disparity Estimation for Real Scenes using DispNet. The first column is the one of the images from the stereo-pair, second column is the groundtruth disparity and third column is the result obtained from the DispNet.

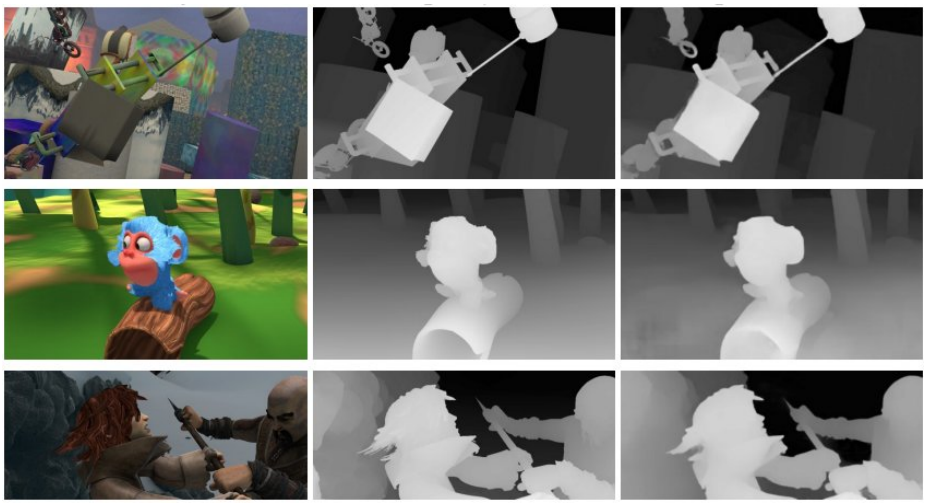

Figure 4.8: Disparity Estimation for using DispNet for examples from simulated dataset. The first column is the one of the images from the stereo-pair, second column is the groundtruth disparity and third column is the result obtained from the DispNet.

We evaluated all methods on a system with an Intel Xeon E5 with 2.40GHz and an Nvidia GTX 1080. Our model was evaluated on test data from Sintel, KITTI and Middlebury datasets. We achieved 2.19 EPE on KITTI and 2.02 on FlyingThings3D and 6.38 on Sintel Dataset. The per frame runtime of our model was 2.13 secs.

4.4 SceneFlowNet: Estimate SceneFlow

For Scene Flow algorithms, we assume a standard stereo camera rig consisting of calibrated left and right cameras that is built-in in most passenger cars and commercial vehicles. As input for our approach, two temporally adjacent frame pairs are used. These four images provide sufficient information to estimate 3D scene flow.

To estimate a full scene flow representation in 3D, we want to combine depth and 2D motion information. Depth is estimated by DispNet using only first image pair of the left and right camera at a time $t-1$ and using second image pair of left and right camera at time t. Motion information is estimated by estimated optical flow obtained using FlowNet using only consecutive images from the left camera over time and similarly for the right camera. Both complement each other regarding scene flow. Optical flow does not have

depth information, and stereo depth can be generalized to change over time.

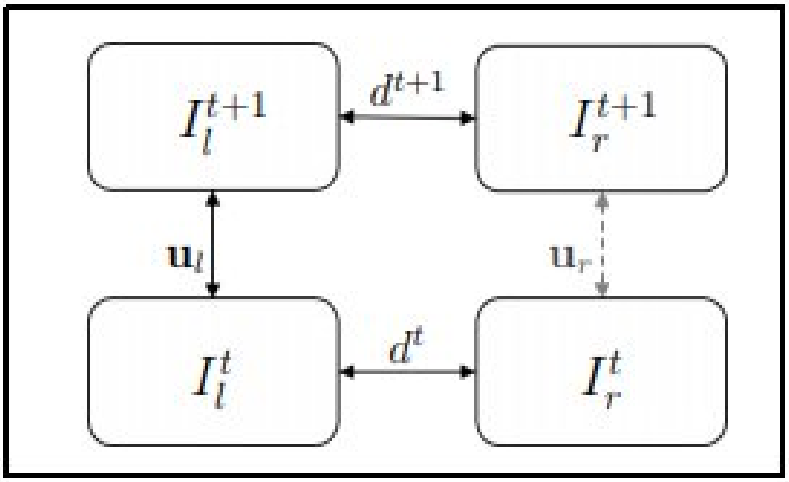

Figure 4.9: Relation of two stereo image pairs

In figure 4.9, subscripts denote the view point(left/right) and superscripts the time step. We consider the left and the right image at time t = 0 the reference frame. Each stereo image pair is related by the according disparity map, while the temporal image pairs are related by 2D optical flow.

4.4.1 Combining Optical Flow and Disparity

The DispNet and FlowNet trained using the architecture described above are used to build the model to estimate Scene Flow. This model is called SceneFlowNet. We use them in bootstrapping form. The integrated SceneFlow model is four layered network where the results of the flow obtained from FlowNet in the previous layer is used as an input to the depth network in the next layer and similarly the output of the depth network in the previous layer is an input to flow network in the next layer. The reasoning behind bootstrapping network is that depth estimation helps us fine-tine the optical flow estimation and optical flow estimation in turn helps us fine-tune the depth prediction and resultant output is an efficient SceneFlow model. Since we want to train this model end-to-end, the submodules

i.e FlowNet and DispNet have to be end-to-end trainable models as well. Hence, we use DispNet for disparity estimation instead of some traditional techniques.

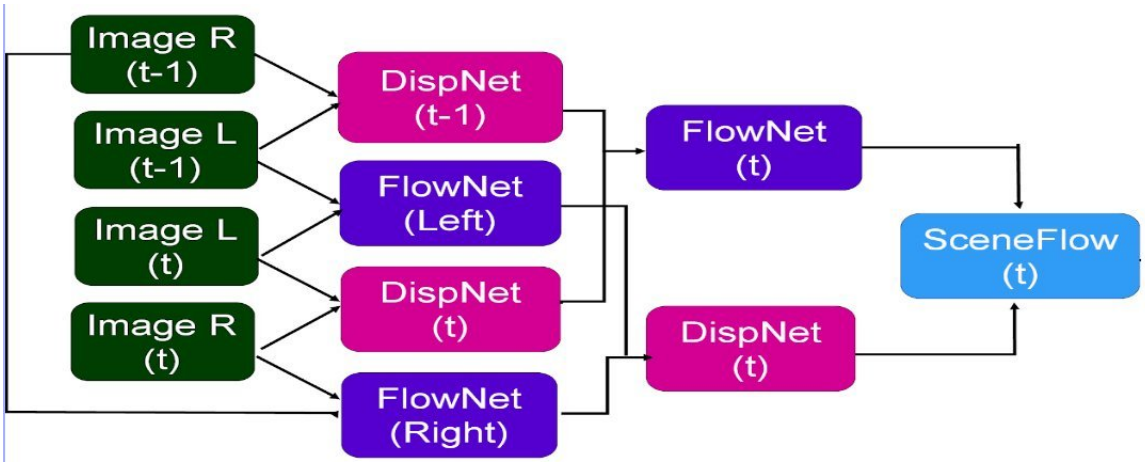

Figure 4.10: SceneFlowNet architecture

The input to the model is the stereoscopic image pair at time $t-1$ and time t. The DispNet calculates the disparity $\mathcal{D}_{t-1}$ for left image and right image at time $t-1$ and disparity $\mathcal{D}_t$ for left image and right image at time t. The FlowNet calculates Flow $\mathcal{F}_l$ between left images at time $t-1$ and time t. Similarly, it also calculates the Flow $\mathcal{F}_r$ between right images at time $t-1$ and t.

From these four mappings, it is possible to reconstruct a scene flow field that we define as follows:

$$\mathcal{S}_f : \Omega \rightarrow R^4$$

where $\mathcal{S}_f$ is the output SceneFlow map, Ω is the dimension of input image (all four images have same dimension Ω). Each image point $x = (x, y)^T$ is mapped to four values $\mathcal{S}_f(x) = (u, v, d_0, d_1)^T$ which fully describe the 3D motion and 3D geometry given the camera intrinsics and extrinsics. The four components are the values of the optical flow field, the disparity at the current time step, and the disparity value at the next time step where the optical flow is pointing to.

The results from each of them are interwined in form of bootstrap to fine tune the Scene

Flow Estimation. However, if the optical flow leaves the image boundaries, or the disparity maps contain gaps, we can not reconstruct the full scene flow at every pixel position. Thus, our scene flow estimate is non-dense by nature.

4.5 Results

The most important part of our proposed method is that it extends 2D motion information to three dimensional space using depth. Compared to optical flow, scene flow additionally describes the motion in direction of viewing, providing full 3D motion information that many applications like autonomous vehicles can potentially benefit from. The evaluation of scene flow results is based on 3D motion vectors. In addition to the disparity and optical flow maps of the reference frame it depends on disparity estimates for the second time step which we represent in the reference frame as well i.e. they are warped via the optical flow. This guarantees that for every pixel of the reference frame a 3D scene flow vector can be stored via two disparity maps and one flow map. Our evaluation takes place in image space (disparity and flow space) as all measurements in the training datasets have been obtained in image space.

4.5.1 Accuracy

We test our SceneFlowNet on the KITTI 2015 benchmark [16]. We evaluate the performance of disparity estimates for two frames $(\mathcal{D}_1, \mathcal{D}_2)$, flow estimates $(\mathcal{F}_r)$ for the reference frame, and scene flow estimates $(\mathcal{S}_F)$ for all the pixels in the frame. Errors in disparity and flow evaluation are counted if disparity or flow estimation exceeds 3 pixels and 5% of its true value. In Scene Flow evaluation, the error is counted if any pixel in any of the three estimates exceed the criterion. We report an Average Estimate of Pixel Error (EPE) of 26.23 for $\mathcal{D}_1$, EPE of 20.76 for $\mathcal{D}_2$, EPE of 30.33 for $\mathcal{F}_r$ and EPE of 36.97 for $\mathcal{S}_F$ for scenes with occlusions(OCC Error). An EPE of 13.45 for $\mathcal{D}_1$, EPE of 10.32 for $\mathcal{D}_2$, EPE of 16.72 for $\mathcal{F}_r$ and EPE of 14.67 for $\mathcal{S}_F$ for Non Occlusions scene(NOC Error) is achieved.

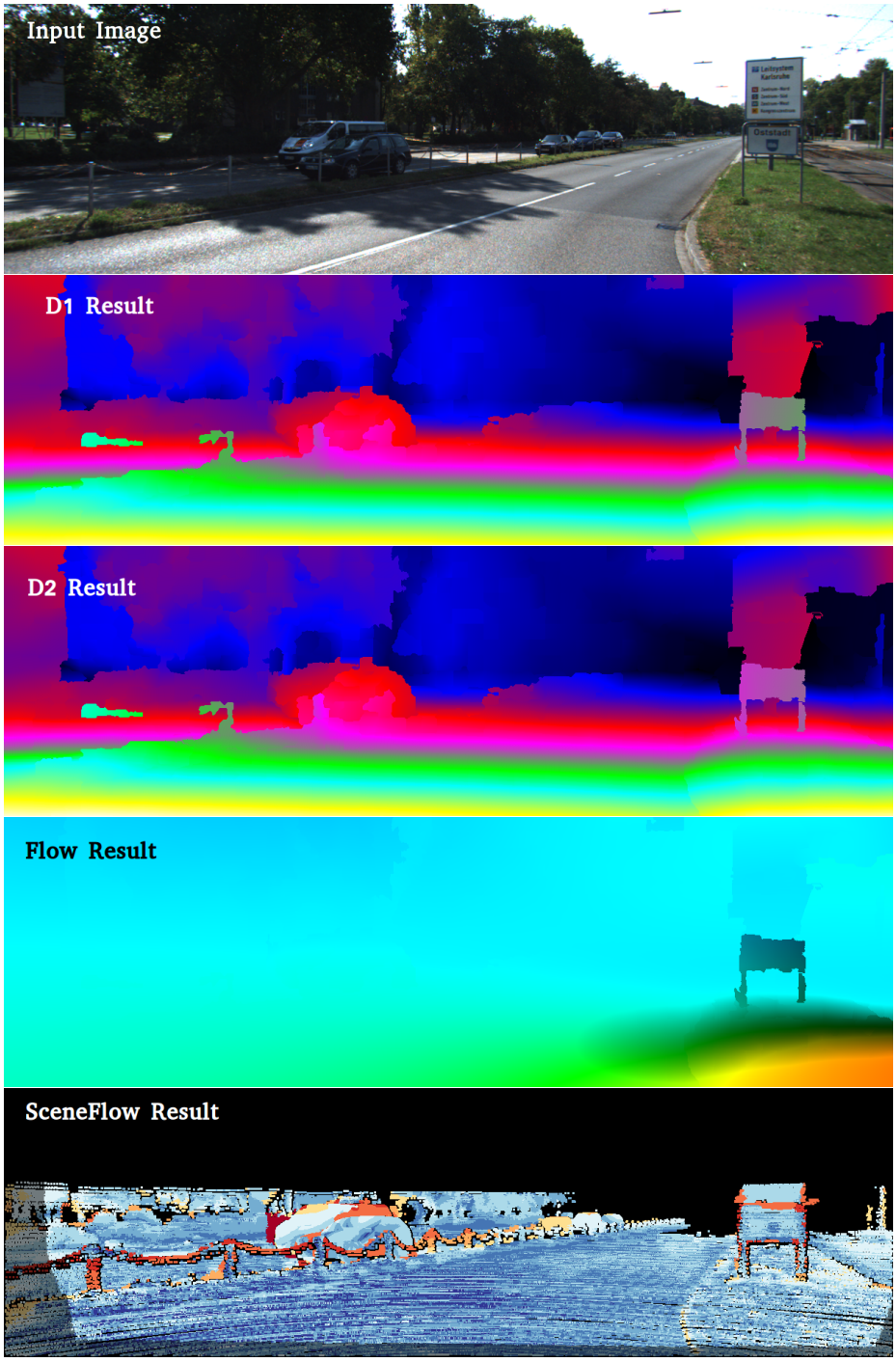

Figure 4.11: Visualization of SceneFlow Results for one of the example from KITTI Vision Dataset.Here, the images are input sample, disparity map at reference frame $\mathcal{D}_1$, disparity map at next time stamp $\mathcal{D}_2$, the optical flow and the sceneflow results respectively.

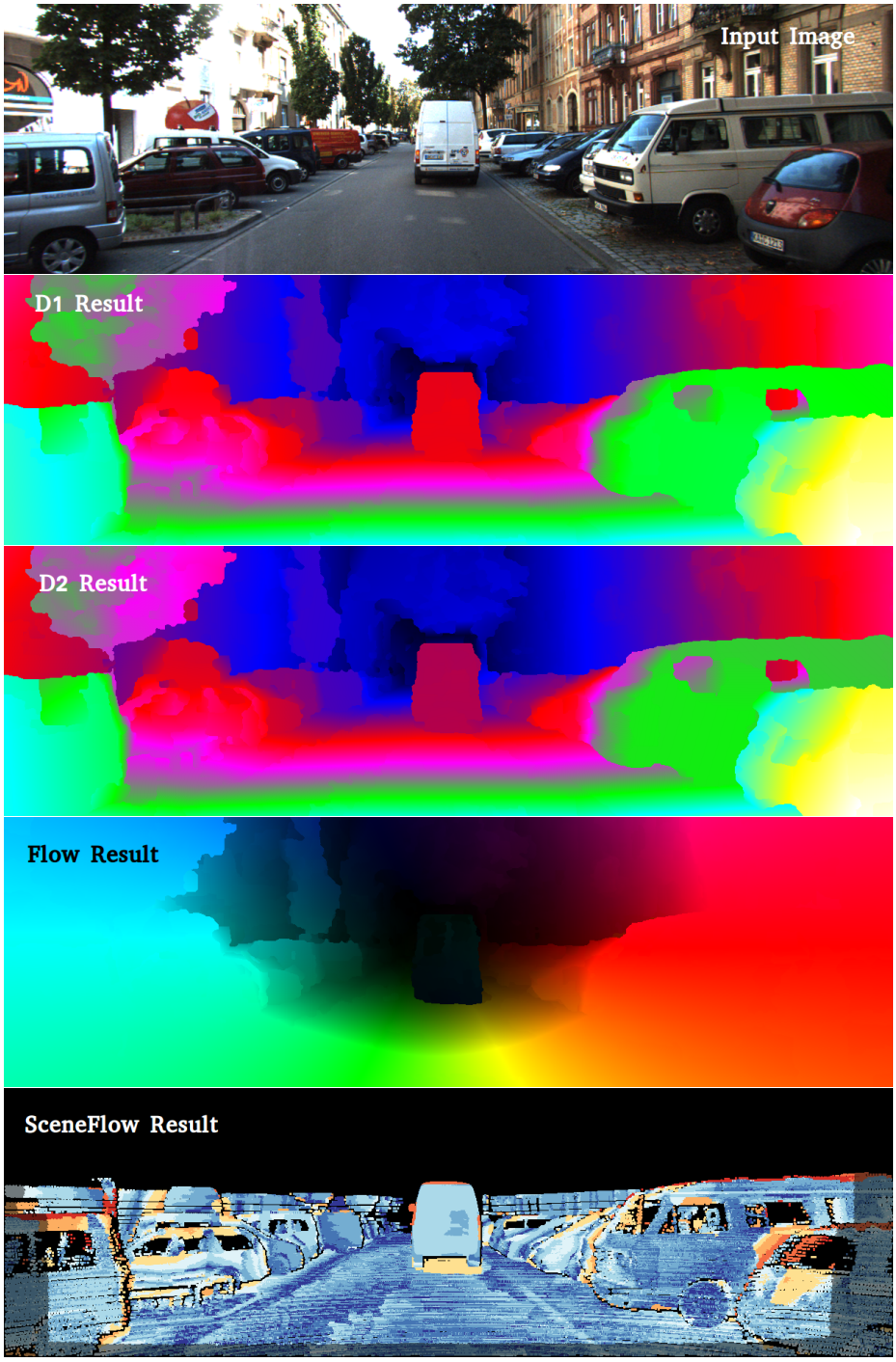

Figure 4.12: Visualization of SceneFlow Results for one of the example from KITTI Vision Dataset.Here, the images are input sample, disparity map at reference frame $\mathcal{D}_1$, disparity map at next time stamp $\mathcal{D}_2$, the optical flow and the sceneflow results respectively.

CHAPTER 5
CONCLUSION & FUTURE WORK

5.1 Conclusion

The concepts of Computer Vision and Deep Learning used for the perception module of autonomous systems have been illustrated in the thesis along with the details of methods used for Object Detection and SceneFlow Estimation. Here we give some of the most notable results obtained from each of the methods to give reader a better understanding of various approaches used and the results obtained using them.

5.2 Conclusion: Object Detection

We trained models for object detection using three different methods. The conclusion for these object detections method can be listed as:

- Faster RCNN is the fastest amongst the three methods for detecting objects in the image frames. It can process images at 0.2 sec / image and thus can effectively process around 5 frames/sec. With speed of 5fps, it can be used in application requiring real time detections.

- The speed of Faster RCNN comes with very large training time. This is result of having the need to train the Region Proposal Network in addition of normal architecture of CNN for image classification.

- The most commonly used anchors in the Faster RCNN are in ratio of 1:1, 1:2 and 2:1. However, models trained with anchors of these ratios are not fit for large vehicles or are very close to vehicles resulting in no detections. Different anchor ratios when included result in more robust detections.

5.3 Conclusion: Scene Flow

- Building on recent progress in design of convolutional network architectures, we have shown that it is possible to train a network to directly predict optical flow from two input images.
- The training data needs to more be realistic. The artificial Flying Chairs dataset including just affine motions of synthetic rigid objects is not sufficiently accurate to predict optical flow in natural scenes which usually have much more complex structure.
- The results for disparity estimation using DispNet are very accurate as it learns low level representations of features and finds corresponding points very effectively.
- DispNet being a backward differentiable deep model supports end-to-end training and thus can be integrated in the architecture of Scene Flow.

5.4 Future Work

The Object Detection methods use detect the objects with certain confidence score and return the bounding boxes around the object. These bounding boxes also include regions that do not belong to the object. The need for detecting objects at pixel level arises and pixel level segmentation is an active area of research. Mask-RCNN is one such method that does pixel level segmentation for Object Detection. Conducting research and experiments to test the efficiency of such methods for autonomous driving scenerios can increase the efficiency of tasks.

The Scene Flow predicted by the model is not accurate for scenes which have objects that do not have unique textures and features. In such cases, it comes difficult for it to be able to distinguish between different objects resulting in poor Optical Flow estimation and thus poor Scene Flow.

We can address this problem by providing more sophisticated dataset that has a wider range of objects included so that it learn representation for vaster range of objects. Also, since the training data used for our purpose is synthetically generated dataset using limited rigid objects with some affine motions, providing more realistic dataset or data collected from real scenerios might prove very useful.

Appendices

APPENDIX A

CODE FOR CALCULATING THE OPTICAL FLOW USING FLOWNET MODEL

```
from __future__ import print_function

import os, sys, numpy as np
import argparse
from scipy import misc
import caffe
import tempfile
from math import ceil

parser = argparse.ArgumentParser()
parser.add_argument('caffemodel', help='path to model')
parser.add_argument('deployproto',
         help='path to deploy prototxt template')
parser.add_argument('img0', help='image 0 path')
parser.add_argument('img1', help='image 1 path')
parser.add_argument('out',  help='output filename')
parser.add_argument('--gpu',
     help='gpu id to use (0, 1, ...)', default=0, type=int)
parser.add_argument('--verbose',
     help='whether to output all caffe logging',
        action='store_true')

args = parser.parse_args()

if(not os.path.exists(args.caffemodel)):
       raise BaseException('No caffemodel: '+args.caffemodel)
if(not os.path.exists(args.deployproto)):
       raise BaseException('No deploy-proto: '+args.deployproto)
if(not os.path.exists(args.img0)):
       raise BaseException('img0 does not exist: '+args.img0)
if(not os.path.exists(args.img1)):
       raise BaseException('img1 does not exist: '+args.img1)

num_blobs = 2
input_data = []
img0 = misc.imread(args.img0)
if len(img0.shape) < 3:
     input_data.append(img0[np.newaxis, np.newaxis, :, :])
else:
     input_data.append(img0[np.newaxis, :, :, :].
```

```
        transpose(0, 3, 1, 2)[:, [2, 1, 0], :, :])

img1 = misc.imread(args.img1)
if len(img1.shape) < 3:
     input_data.append(img1[np.newaxis, np.newaxis, :, :])
else:
     input_data.append(img1[np.newaxis, :, :, :].
        transpose(0, 3, 1, 2)[:, [2, 1, 0], :, :])

width = input_data[0].shape[3]
height = input_data[0].shape[2]
vars = {}
vars['TARGET_WIDTH'] = width
vars['TARGET_HEIGHT'] = height

divisor = 64.
vars['ADAPTED_WIDTH'] = int(ceil(width/divisor) * divisor)
vars['ADAPTED_HEIGHT'] = int(ceil(height/divisor) * divisor)

vars['SCALE_WIDTH'] = width / float(vars['ADAPTED_WIDTH']);
vars['SCALE_HEIGHT'] = height / float(vars['ADAPTED_HEIGHT']);

tmp = tempfile.NamedTemporaryFile(mode='w', delete=True)

proto = open(args.deployproto).readlines()
for line in proto:
    for key, value in vars.items():
        line = line.replace(tag, str(value))

    tmp.write(line)

tmp.flush()

if not args.verbose:
    caffe.set_logging_disabled()
caffe.set_device(args.gpu)
caffe.set_mode_gpu()
net = caffe.Net(tmp.name, args.caffemodel, caffe.TEST)

input_dict = {}
for blob_idx in range(num_blobs):
    input_dict[net.inputs[blob_idx]] = input_data[blob_idx]

#
# There is some non-deterministic nan-bug in caffe
# it seems to be a race-condition
#
```

```
print('Network forward pass using %s.' % args.caffemodel)
i = 1
while i<=5:
    i+=1

    net.forward(**input_dict)

    containsNaN = False
    for name in net.blobs:
        blob = net.blobs[name]
        has_nan = np.isnan(blob.data[...]).any()

        if has_nan:
            print('blob %s contains nan' % name)
            containsNaN = True

    if not containsNaN:
        print('Succeeded.')
        break
    else:
        print('***** FOUND NANs, RETRYING ********')

blob = np.squeeze(net.blobs['predict_flow_final'].
                  data).transpose(1, 2, 0)

def readFlow(name):
    if name.endswith('.pfm') or name.endswith('.PFM'):
        return readPFM(name)[0][:,:,0:2]

    f = open(name, 'rb')

    header = f.read(4)
    if header.decode("utf-8") != 'PIEH':
        raise Exception('Flow file header doesnt contain PIEH')

    width = np.fromfile(f, np.int32, 1).squeeze()
    height = np.fromfile(f, np.int32, 1).squeeze()

    flow = np.fromfile(f, np.float32,
                width * height * 2).reshape((height, width, 2))

    return flow.astype(np.float32)

def writeFlow(name, flow):
    f = open(name, 'wb')
    f.write('PIEH'.encode('utf-8'))
    np.array([flow.shape[1], flow.shape[0]],
```

```
135             dtype=np.int32).tofile(f)
136     flow = flow.astype(np.float32)
137     flow.tofile(f)
138     f.flush()
139     f.close()
140 
141 writeFlow(args.out, blob)
```

APPENDIX B

CODE FOR CALCULATING THE DISPARITY MAP USING DISPNET MODEL

```
#!/usr/bin/env python
import os, sys
import subprocess
from math import ceil

my_dir = os.path.dirname(os.path.realpath(__file__))
os.chdir(my_dir)

caffe_bin = 'bin/caffe.bin'
img_size_bin = 'bin/get_image_size'

template = 'model/deploy.tpl.prototxt'

# ============================================================


def get_image_size(filename):
    global img_size_bin
    dim_list = [int(dimstr) for dimstr in str(subprocess.check_output([img_s
    if not len(dim_list) == 2:
        print('Could not determine size of image %s' % filename)
        sys.exit(1)
    return dim_list


def sizes_equal(size1, size2):
    return size1[0] == size2[0] and size1[1] == size2[1]


def check_image_lists(lists):
    images = [[], []]

    with open(lists[0], 'r') as f:
        images[0] = [line.strip() for line in f.readlines() if len(line.stri
    with open(lists[1], 'r') as f:
        images[1] = [line.strip() for line in f.readlines() if len(line.stri

    if len(images[0]) != len(images[1]):
        print("Unequal amount of images in the given lists (%d vs. %d)" % (
        sys.exit(1)
```

```
41
42      if not os.path.isfile(images[0][0]):
43          print('Image %s not found' % images[0][0])
44          sys.exit(1)
45
46      base_size = get_image_size(images[0][0])
47
48      for idx in range(len(images[0])):
49          print("Checking image pair %d of %d" % (idx+1, len(images[0])))
50          img1 = images[0][idx]
51          img2 = images[1][idx]
52
53          if not os.path.isfile(img1):
54              print('Image %s not found' % img1)
55              sys.exit(1)
56
57          if not os.path.isfile(img2):
58              print('Image %s not found' % img2)
59              sys.exit(1)
60
61          img1_size = get_image_size(img1)
62          img2_size = get_image_size(img2)
63
64          if not (sizes_equal(base_size, img1_size) and sizes_equal(base_size,
65              print('The images do not all have the same size. (Images: %s or
66              sys.exit(1)
67
68      return base_size[0], base_size[1], len(images[0])
69
70  if not (os.path.isfile(caffe_bin) and os.path.isfile(img_size_bin)):
71      print('Caffe tool binaries not found. Did you compile caffe with tools
72      sys.exit(1)
73
74  if len(sys.argv)-1 != 2:
75      print("Use this tool to test DispNet on images\n"
76            "Usage for single image pair:\n"
77            "    ./demo_dispnet.py IMAGE1 IMAGE2\n"
78            "\n"
79            "Usage for a pair of image lists (must end with .txt):\n"
80            "    ./demo_dispnet.py LIST1.TXT LIST2.TXT\n")
81      sys.exit(1)
82
83  img_files = sys.argv[1:]
84  using_lists = False
85  list_length = 1
86
87  if img_files[0][-4:].lower() == '.txt':
```

```
88      print("Checking the images in your lists...")
89      (width, height, list_length) = check_image_lists(img_files)
90      using_lists = True
91      print("Done.")
92  else:
93      print("Image files: " + str(img_files))
94
95      # Check images
96
97      for img_file in img_files:
98          if not os.path.isfile(img_file):
99              print('Image %s not found' % img_file)
100             sys.exit(1)
101
102
103     # Get image sizes and check
104     img_sizes = [get_image_size(img_file) for img_file in img_files]
105
106     print("Image sizes: " + str(img_sizes))
107
108     if not sizes_equal(img_sizes[0], img_sizes[1]):
109         print('Images do not have the same size.')
110         sys.exit(1)
111
112     width = img_sizes[0][0]
113     height = img_sizes[0][1]
114
115 # Prepare prototxt
116 subprocess.call('mkdir -p tmp', shell=True)
117
118 if not using_lists:
119     with open('tmp/img1.txt', "w") as tfile:
120         tfile.write("%s\n" % img_files[0])
121
122     with open('tmp/img2.txt', "w") as tfile:
123         tfile.write("%s\n" % img_files[1])
124 else:
125     subprocess.call(['cp', img_files[0], 'tmp/img1.txt'])
126     subprocess.call(['cp', img_files[1], 'tmp/img2.txt'])
127
128 divisor = 64.
129 adapted_width = ceil(width/divisor) * divisor
130 adapted_height = ceil(height/divisor) * divisor
131 rescale_coeff_x = width / adapted_width
132
133 replacement_list = {
134     '$ADAPTED_WIDTH': ('%d' % adapted_width),
```

```
    '$ADAPTED_HEIGHT': ('%d' % adapted_height),
    '$TARGET_WIDTH': ('%d' % width),
    '$TARGET_HEIGHT': ('%d' % height),
    '$SCALE_WIDTH': ('%.8f' % rescale_coeff_x)
}

proto = ''
with open(template, "r") as tfile:
    proto = tfile.read()

for r in replacement_list:
    proto = proto.replace(r, replacement_list[r])

with open('tmp/deploy.prototxt', "w") as tfile:
    tfile.write(proto)

# Run caffe

args = [caffe_bin, 'test', '-model', 'tmp/deploy.prototxt',
        '-weights', 'model/DispNetCorr1D-K_CVPR2016.caffemodel',
        '-iterations', str(list_length),
        '-gpu', '1']

cmd = str.join(' ', args)
print('Executing %s' % cmd)

subprocess.call(args)

print('\nThe resulting disparity is stored in dispnet-corr1d-pred-NNNNNNN.fl
```

REFERENCES

[1] A. Bachrach, R. He, and N. Roy, "Autonomous flight in unknown indoor environments," *International Journal of Micro Air Vehicles*, pp. 217–28, 2009.

[2] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Vision-based state estimation for autonomous rotorcraft mavs in complex environments," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1758–1764, 2013.

[3] Z. Fang, S. Yang, S. Jain, G. Dubey, S. Maeta, S. Roth, S. Scherer, Y. Zhang, and S. Nuske, "Robust autonomous flight in constrained and visually degraded environments," *Springer Field and Service Robotics*, pp. 411–425, 2016.

[4] R. Mur-Artal, J. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *Robotics, IEEE Transactions*, vol. 31(5), pp. 1147–1163, 2015.

[5] J. Engel, T. Schps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," *European Conference on Computer Vision (ECCV)*, pp. 834–849, 2014.

[6] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deep driving: Learning affordance for direct perception in autonomous driving," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2722–2730, 2015.

[7] A. Giusti, J. Guzzi, D. Ciresan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, and G. D. Caro, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letter*, 2016.

[8] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," *In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proc. IEEE Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. I–511–I–518, 2001.

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *Proc. IEEE Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. 886–893, 2005.

[11] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *International Conference on Learning Representations (ICLR)*, 2014.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proc. IEEE Soc. Conf. Comput. Vis. Pattern Recognit.(CVPR)*, 2014.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, pp. 91–99, 2015.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Soc. Conf. Comput. Vis. Pattern Recognit.(CVPR)*, 2016.

[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *European Conference on Computer Vision*, pp. 21–37, 2016.

[16] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," *CVPR*, 33543361, 2012.

[17] C. Vogel, K. Schindler, and S. Roth, "3d scene flow estimation with a piecewise rigid scene model," *IJCV*, 2015.

[18] M. Neoral and J. ochman, "Object scene flow with temporal consistency," *22nd Computer Vision Winter Workshop, Pattern Recongition and Image Processing Group*, 2017.