EXTRACTING COGNITION OUT OF IMAGES
FOR THE PURPOSE OF AUTONOMOUS DRIVING

CHENYI CHEN

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
OPERATIONS RESEARCH AND FINANCIAL ENGINEERING
ADVISER: ALAIN L. KORNHAUSER

MAY 2016
Abstract

Autonomous driving is a broadly recognized solution to serious traffic problems such as accidents and congestions. It is a very broad topic that extends across cognition, artificial intelligence, and control. While this thesis primarily focuses on the cognition aspect, others are also considered. Here, the thesis proposes several computer vision algorithms for autonomous driving, encompassing three major parts:

In part one, experiments on motion-based object recognition are presented. The proposed method differentiates objects according to their speed. In part two, the artificial intelligence aspect of autonomous driving is considered. Research on training an autonomous driving AI agent through reinforcement learning is introduced.

In part three, the key part of the thesis, a direct perception approach is proposed to drive a car in a highway environment. In this approach, an input image is mapped to a small number of key perception indicators that directly relate to the affordance of a road/traffic state for driving. This representation provides a set of compact yet complete descriptions of the scene to enable a simple controller to drive autonomously on highways. Using synthetic images from a virtual environment, a deep convolutional neural network (ConvNet) is trained for direct perception. Experiments show that the model can effectively drive a car in a very diverse set of virtual environments, and it provides good estimation of affordance indicators from real driving images. To further improve the performance of the direct perception-based system, the issue of temporal information is considered by studying the Long Short Term Memory (LSTM) unit and its influence on the affordance indicator estimation. Quantitative results show that adding the LSTM unit does help to improve the system’s performance.

Finally, as object detection is closely related to autonomous driving, in Appendix A a deep learning-based small object detection approach is proposed. The applicability of the state-of-the-art object detection algorithms to the small object detection task is studied.
Acknowledgements

Foremost, I would like to thank my advisers, Alain Kornhauser and Jianxiong Xiao, for their continuous support and guidance throughout my PhD studies. Prof. Kornhauser brought me to the field of autonomous driving, while Prof. Xiao guided me through all my research projects on computer vision. In the past four years, they have taught me how to think and write logically, and how to approach a challenging problem and do good research.

I was fortunate to collaborate with Ming-Yu Liu and Oncel Tuzel from MERL on the small object detection project. This work would not have been possible without their insightful suggestions and invaluable advice. I would also like to thank Ari Seff for his help and suggestions on the DeepDriving project.

I am grateful to my girl friend, Xiaoli Wang, without whom my PhD life in Princeton would not have been so amazing. Finally, I would like to thank my parents for their persistent support and encouragement, my life would not be possible without them.
To my parents.

To my girl friend, Xiaoli Wang.
Contents

Abstract ........................................................................ iii
Acknowledgements ........................................................ iv
List of Tables ................................................................. x
List of Figures ................................................................. xiii

1 Introduction ................................................................. 1
  1.1 Motivation ............................................................. 1
    1.1.1 Primary sensor: LiDAR vs. camera ....................... 4
    1.1.2 Fundamental concept: recording vs. non-recording .... 5
    1.1.3 Virtual environment for system development .......... 8
    1.1.4 System decomposition ......................................... 11
  1.2 Thesis outline ......................................................... 19

2 Literature Review ........................................................ 20
  2.1 Autonomous driving systems .................................... 20
  2.2 Computer vision for autonomous driving .................... 21
    2.2.1 Object recognition ............................................. 22
    2.2.2 3D street scene reconstruction and vehicle localization .. 25
    2.2.3 Scene understanding .......................................... 26
    2.2.4 Datasets for autonomous driving research .............. 27
  2.3 Decision making for autonomous driving .................... 27
2.4 Deep learning and convolutional neural network (ConvNet) ........................................ 28
   2.4.1 Basics of deep learning ................................................................. 29
   2.4.2 Deep learning-based research and applications ................................. 33
   2.4.3 Deep learning for driving .............................................................. 35
2.5 Object detection ......................................................................................... 36

3 Time to Contact: Recognizing Moving Objects Using Motion Patterns ............... 40
   3.1 Introduction ............................................................................................ 40
   3.2 Basics of Time to Contact ...................................................................... 41
   3.3 Implementation ....................................................................................... 48
   3.4 Results and analysis ................................................................................ 51
      3.4.1 Time to Contact for single object .................................................. 51
      3.4.2 Time to Contact map for multiple objects ...................................... 52
      3.4.3 Quantitative assessment ................................................................ 57
      3.4.4 Comparison with baseline ............................................................. 64
   3.5 Summary .................................................................................................. 70

4 Training an Autonomous Driving AI Agent with Reinforcement Learning .......... 71
   4.1 Introduction ............................................................................................ 71
   4.2 Reinforcement learning for driving ......................................................... 72
      4.2.1 Basics ............................................................................................. 72
      4.2.2 AI agent for lane changing and velocity control ............................ 77
      4.2.3 AI agent for steering control .......................................................... 80
   4.3 Apprenticeship learning for driving ......................................................... 82
   4.4 Deep reinforcement learning for driving ............................................... 88
   4.5 Summary .................................................................................................. 92
5 DeepDriving: Direct Perception for Autonomous Driving

5.1 Introduction ......................................................... 93
5.2 Learning affordance for driving perception through virtual environment
   5.2.1 Mapping from an image to affordance ....................... 97
   5.2.2 Mapping from affordance to action ....................... 100
5.3 Implementation .................................................. 103
5.4 TORCS evaluation ................................................. 105
   5.4.1 Qualitative assessment .................................. 106
   5.4.2 Comparison with baselines .............................. 107
5.5 Testing on real-world image data ................................ 112
   5.5.1 Smartphone video ....................................... 112
   5.5.2 Car distance estimation on the KITTI dataset ......... 114
   5.5.3 Comparison with DPM-based baseline ................. 116
5.6 Visualization ..................................................... 119
5.7 Summary ......................................................... 121

6 Extending DeepDriving with Temporal Information .......................... 122

6.1 Introduction ....................................................... 122
   6.1.1 The role of temporal information ....................... 122
6.2 Short term motion information ................................ 125
6.3 Long Short Term Memory (LSTM) ............................ 127
6.4 Network architecture ........................................... 129
6.5 Implementation .................................................. 133
6.6 Results and analysis .......................................... 136
6.7 Further discussion .............................................. 140
6.8 Summary ......................................................... 142

7 Conclusions and Future Work ........................................ 143
List of Tables

3.1 Mean Total Absolute Error (in frames) on Sequence-17. The parameter $k$ is used in ridge regression. In both steps of the Case IV method, there is a $k_i$ for ridge regression, and we choose $k_1 = k_2 = k$. In the table, an empty spot refers to the result of the specific method and parameter is not computed (since not necessary). 60

3.2 Mean Total Absolute Error (in frames) on Sequence-06. The parameter $k$ is used in ridge regression. In both steps of the Case IV method, there is a $k_i$ for ridge regression, and we choose $k_1 = k_2 = k$. In the table, an empty spot refers to the result of the specific method and parameter is not computed (since not necessary). 63

3.3 Mean Total Absolute Error (in frames) of the IBD-based TTC method and the optical flow-based TTC method on Sequence-17. 67

3.4 Mean Total Absolute Error (in frames) of the IBD-based TTC method and the optical flow-based TTC method on Sequence-06. 69

5.1 Mean Absolute Error on the testing set for the Caltech lane detector baseline. angle is in radians, the rest are in meters. 109
5.2 Standard Deviation of absolute errors on the testing set for the Caltech lane detector baseline. \textit{angle} is in radians, the rest are in meters.

5.3 Mean Absolute Error on the testing set for the GIST baseline. \textit{angle} is in radians, the rest are in meters.

5.4 Standard Deviation of absolute errors on the testing set for the GIST baseline. \textit{angle} is in radians, the rest are in meters.

5.5 Mean Absolute Error (in meters) on the KITTI testing set. Errors are computed by both penalizing (column 1$\sim$3) and not penalizing false positives (column 4$\sim$6).

5.6 Standard Deviation of absolute errors (in meters) on the KITTI testing set. Errors are computed by both penalizing (column 1$\sim$3) and not penalizing false positives (column 4$\sim$6).

6.1 Mean Absolute Error of different models on the testing set. \textit{angle} is in radians, the rest are in meters. “GT” refers to that the ground truth values are used as feedback for training, while “OP” refers to that the network’s actual outputs are used as feedback for training.

6.2 Standard Deviation of different models on the testing set. \textit{angle} is in radians, the rest are in meters. “GT” refers to that the ground truth values are used as feedback for training, while “OP” refers to that the network’s actual outputs are used as feedback for training.

6.3 Number of Misclassification of different models on the testing set. “GT” refers to that the ground truth values are used as feedback for training, while “OP” refers to that the network’s actual outputs are used as feedback for training.
A.1 Statistics of our small object detection dataset. 151
A.2 Median relative area of the object categories in the PASCAL VOC dataset. 151
A.3 Recall rate of the region proposal generation methods. 156
A.4 Up-sampling effects. Both networks are trained and tested with DPM proposals, 500 per image per category. 158
A.5 Results of ContextNet. All the networks are trained (2000 per image) and tested (500 per image) with modified RPN proposals. 160
A.6 Four scales of the object size. The size is counted using the longest edge of the re-scaled box, in pixel. 165
A.7 Original results of various object detection networks and re-scored results of the co-occurrence ConvNet. The average precision is computed using $\text{IoU} \geq 0.5$ (upper table) and $\text{IoU} \geq 0.3$ (lower table), respectively. 168
A.8 Results of DPM, AlexNet R-CNN, and VGG16 R-CNN. The AlexNet in row 2 is trained and tested with DPM proposals, 500 per image per category. The AlexNet in row 3 and the VGG16 in row 4 are trained (2000 per image) and tested (500 per image) with modified RPN proposals. 170
A.9 Results of ContextNet. Both networks are trained (2000 per image) and tested (various) with modified RPN proposals. 170
## List of Figures

1.1 Motor vehicle deaths in the US by year from 1899 to 2014. Data from [189]. ................................. 2

1.2 An example of LiDAR. (b) Upper: 3D point cloud generated by LiDAR, lower: the corresponding image of the environment, courtesy of the KITTI dataset. .................................. 4

1.3 Closed-loop testing for the autonomous driving system. . . . . . . . . . . . . . . . . . . . . . . 8

1.4 Functionality decomposition of the autonomous driving system. 11

1.5 Computer vision tasks that are related to autonomous driving. 12

1.6 Open-loop testing for the cognition module. ............................ 14

1.7 Combining the cognition module and the decision making/AI module. ................................. 17

1.8 Combining all the three modules as an end-to-end system. . 18

1.9 Combining the decision making/AI module and the control module. ................................. 18

2.1 An example of convolution. The filter convolves with the input feature map and generates a new feature map. ................................. 30

2.2 An example of pooling. (a) 2 * 2 average pooling. (b) 2 * 2 max pooling. (c) The pooling operation allows small translation invariance and helps to reduce the size of the input feature map. . . . . . . . . . . . . 31

2.3 An example of dropout. .............................................. 32
3.1  Four cases of the planar object’s relative motion. Courtesy of

3.2  Self-made equipments for carrying out the experiments. . . 49

3.3  Pipeline of computing TTC and TTC map. . . . . . . . . . . 50

3.4  TTC result of Sequence-02 using the Case I method. In (b) the
blue line is the estimated TTC, and the red line is the ground truth.
In the experiment, time is measured by “frames”. . . . . . . . . . . 51

3.5  TTC results of Sequence-06 using the four different methods.
The blue line is the estimated TTC, and the red line is the ground
truth. In the experiments, time is measured by “frames”. . . . . . . 52

3.6  Examples of SLIC segmentation. . . . . . . . . . . . . . . . . 53

3.7  TTC map results (at a randomly chosen time step) of
Sequence-03 using various methods. (b) ~ (d) The TTC value
of each super pixel is color coded and indicated by the colorbar. . . 54

3.8  TTC map results (at a randomly chosen time step) of
Sequence-03 using various methods and ridge regression. The
TTC value of each super pixel is color coded and indicated by the
colorbar. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

3.9  Labeling and segmenting each image in Sequence-17. (a) and
(b) Label the vertices of the planar object. (c) and (d) Partition the
planar object into 4 * 4 segmentation. . . . . . . . . . . . . . . . . 58

3.10 TTC results of the 4*4 segmented regions in Sequence-17. The
Case II method with ridge regression $k = 0.1$ is implemented. The blue
line is the estimated TTC, and the red line is the ground truth. In the
experiment, time is measured by “frames”. . . . . . . . . . . . . . . 59
3.11 TTC results of the 4*4 segmented regions in Sequence-17. The Case IV method with ridge regression $k_1 = k_2 = 1$ is implemented. The blue line is the estimated TTC, and the red line is the ground truth. In the experiment, time is measured by “frames”.

3.12 Labeling and segmenting each image in Sequence-06. (a) and (b) Label the vertices of the planar object. (c) and (d) Partition the planar object into 4*4 segmentation.

3.13 TTC results of the 4*4 segmented regions in Sequence-06. The Case II method without ridge regression is implemented. The blue line is the estimated TTC, and the red line is the ground truth. In the experiment, time is measured by “frames”.

3.14 TTC results of the 4*4 segmented regions in Sequence-06. The Case IV method with ridge regression $k_1 = k_2 = 1$ is implemented. The blue line is the estimated TTC, and the red line is the ground truth. In the experiment, time is measured by “frames”.

3.15 Implementing optical flow on Sequence-17. In (b) the optical flow is color coded, where hue indicates orientation and saturation indicates magnitude. In (c) and (d) the motion fields are color coded, the warmer the more positive the motion fields, the colder the more negative the motion fields.

3.16 TTC results of the entire planar object in Sequence-17. The TTC results are computed using the optical flow-based method (both from $v_x$ field (in green) and $v_y$ field (in blue) respectively) and the Case II method (in magenta). The ground truth is drawn in red. In the experiment, time is measured by “frames”.
3.17 **TTC results of the 4 × 4 segmented regions in Sequence-17.**

The TTC results are computed using the optical flow-based method (both from $v_x$ field (in green) and $v_y$ field (in blue) respectively). The ground truth is drawn in red. In the experiment, time is measured by “frames”.

66

3.18 **Implementing optical flow on Sequence-06.** In (b) the optical flow is color coded, where hue indicates orientation and saturation indicates magnitude. In (c) and (d) the motion fields are color coded, the warmer the more positive the motion fields, the colder the more negative the motion fields.

67

3.19 **TTC results of the entire planar object in Sequence-06.** The TTC results are computed using the optical flow-based method (both from $v_x$ field (in green) and $v_y$ field (in blue) respectively) and the Case II method (in magenta). The ground truth is drawn in red. In the experiment, time is measured by “frames”.

68

3.20 **TTC results of the 4 × 4 segmented regions in Sequence-06.**

The TTC results are computed using the optical flow-based method (both from $v_x$ field (in green) and $v_y$ field (in blue) respectively). The ground truth is drawn in red. In the experiment, time is measured by “frames”.

68

4.1 **An example of the reward function for driving.** The reward values are manually designed to illustrate the idea.

72

4.2 **Pipeline of training a Q-net for driving which takes a state vector as input.**

76

4.3 **Testing the Q-net for autonomous driving.**

77
4.4 Screenshots of the driving simulator. The host car is represented by the red solid box, while the traffic cars are represented by the yellow solid boxes.

4.5 Illustration of the state $s$. (b) The AI agent monitors an area of two car length for collision, if a traffic car’s rear bump is in this area, it has a collision with the host car. The host car is represented by the red solid box, while the traffic car is represented by the yellow solid box.

4.6 Screenshots of the host car driven by the AI agent for steering control. (a) The training is only done on a circular track. (b) and (c) In testing, the AI agent can follow tracks of any shape. The black curve is the track to be followed, while the red curve is the trajectory of the host car.

4.7 Apprenticeship learning algorithm.

4.8 Illustration of the SVM problem.

4.9 Pipeline of apprenticeship learning.

4.10 Simplified driving testbed for apprenticeship learning. The host car is represented by the red solid box, while the traffic cars are represented by the yellow solid boxes.

4.11 Pipeline of training a ConvNet-based Q-net for driving which takes a image sequence as input.

4.12 Testing the ConvNet-based Q-net for autonomous driving.

4.13 Deep reinforcement learning algorithm.

4.14 Screenshots of two selected driving games in DOSBox.

5.1 Three paradigms for autonomous driving.

5.2 Eight examples of driving scenarios from an ego-centric perspective. The lanes monitored for making driving decisions are marked with light gray.
5.3 **Illustration of our affordance representation.** A lane changing maneuver needs to traverse the “in lane system” and the “on marking system”. (f) shows the designated overlapping area used to enable smooth transitions. .................................................. 98

5.4 **Complete list of affordance indicators in our direct perception representation.** All the affordance indicators are measured in an ego-centric perspective, in which the center of the host car is the origin of the coordinate system. The active range of each affordance indicator is listed in the brackets (assume the lane width is 4 meters), where angle is in radians, the rest are in meters. .................................................. 100

5.5 **Illustration of the control strategy.** Assume overtaking from the right is permitted by the local driving regulation. (d) and (e) explain the constraint for overtaking on sharp curves in order to reduce control complexity. .................................................. 101

5.6 **Controller logic.** The control frequency is determined by the frequency of the while loop (e.g. 10Hz for the TORCS implementation). 102

5.7 **Examples of the 7 tracks used for training.** Each track is customized to the configuration of one-lane, two-lane, and three-lane with multiple asphalt darkness levels. The rest of the tracks are used in the testing set. .................................................. 104

5.8 **Examples of the 22 cars used in the training set.** The rest of the cars are used in the testing set. .................................................. 104

5.9 **System architecture.** The ConvNet processes the TORCS image and estimates 13 indicators for driving. Based on the indicators and the current speed of the car, a controller computes the driving commands which will be sent back to TORCS to drive the host car in the game. 105
5.10 Testing the TORCS-based system. The estimation is shown as an empty box, while the ground truth is indicated by a solid box. For testing on real videos, without the ground truth, only the estimation is shown. .......................... 106

5.11 GIST baseline. Procedure of mapping GIST feature to the 13 affordance indicators for driving using SVR and SVC. .......................... 110

5.12 Comparison of car perception accuracy between different models. Legend: green, ConvNet full; dark blue, ConvNet sub; light blue, GIST half; red, GIST whole. .......................... 111

5.13 Screenshots of testing in special driving scenarios. .......................... 113

5.14 Car distance estimation on the KITTI dataset. (a) The coordinate system is defined relative to the host car. The space is partitioned into three areas, and the objective is to estimate the coordinate of the closest car in each area. (b) Our direct perception approach is compared with the DPM-based mediated perception. The central crop of the KITTI image (indicated by the yellow box in the upper left image and shown in the lower left image) is sent to the far range ConvNet. The bounding boxes output by DPM are shown in red, as are its distance projections in the LiDAR visualization (right). The ConvNet outputs and the ground truth are represented by green and black boxes, respectively. .......................... 116

5.15 Comparison of car distance estimation accuracy between different models. Legend: green, ConvNet penalizing false positives; red, DPM + Projection penalizing false positives; dark blue, ConvNet not penalizing false positives; light blue, DPM + Projection not penalizing false positives. .......................... 118
5.16 Activation patterns of neurons. The neurons’ activation patterns display strong correlations with the host car’s heading, the location of lane markings, and traffic cars. 119

5.17 Response map of our KITTI-based ConvNet. The level of the response is color coded, the warmer the color, the stronger the response. The ConvNet has strong responses over the locations of nearby cars. 120

5.18 Response map of our TORCS-based ConvNet. The level of the response is color coded, the warmer the color, the stronger the response. The ConvNet has strong responses over the locations of lane markings. 120

6.1 Concatenating the RGB image and the two raw motion fields as a 5-channel input. The first 3 channels of the 5-channel input come from the current frame (RGB), while the remaining 2 channels are the motion fields $v_x$ and $v_y$ (grayscale). 126

6.2 Concatenating the RGB image and the color coded optical flow image as a 6-channel input. The first 3 channels of the 6-channel input come from the current frame (RGB), while the remaining 3 channels come from the color coded optical flow results (RGB). 126

6.3 Decomposition of LSTM unit. 127

6.4 Architecture of the network with one-layer LSTM. This model is denoted by “1-L”. 130

6.5 Architecture of the network with two-layer LSTM. This model is denoted by “2-L”. 130

6.6 Architecture of the network with one-layer LSTM and feedback. This model is denoted by “1-L FB”. 131

6.7 Architecture of the network with two-layer LSTM and feedback. This model is denoted by “2-L FB”. 132
6.8 Architecture of the network with two-layer LSTM, feedback, and skip structure. This model is denoted by “2-L Skip FB”.

6.9 Relative errors (%) of the affordance estimation of different LSTM-based models compared with the DeepDriving baseline. The error terms of the DeepDriving model are set to 100%. If any LSTM-based model outperforms the DeepDriving baseline on any indicator, the corresponding relative error is smaller than 100%.

6.10 Relative errors (%) of the classifiers of different LSTM-based models compared with the DeepDriving baseline. The error terms of the DeepDriving model are set to 100%. If any LSTM-based model outperforms the DeepDriving baseline on any indicator, the corresponding relative error is smaller than 100%.

6.11 2D convolution vs. 3D convolution.

7.1 New lane marking models in TORCS.

7.2 Pavement markings at an intersection. Left: screenshot of TORCS, right: real image.

A.1 Detecting small objects with low-res input is a must-have component of a robust vision pipeline.

A.2 Single resolution and multi-resolution structures for RPN.

A.3 ContextNet: the neural network for integrating context information. The two front-end sub-networks take proposal region patches and context region patches as input respectively, the back-end sub-network takes in the concatenation of the two feature vectors and computes the final classification score.
A.4 **Example of how co-occurrence improves the accuracy of small object detection.** By knowing the location of monitor and keyboard, many high score false positives can be eliminated since their locations are not likely to have a mouse.

A.5 **Co-occurrence ConvNet structure.** The classification results of small object region proposals and the ground truth bounding boxes of the big objects are converted to input images of the co-occurrence ConvNet. Spatial relation filters convolve with the corresponding big object location maps, and the resulting feature maps concatenate with the small object score map. All these feature maps are used to produce a new score map for the small object region proposals.

A.6 **Handling various scales.** The small object region proposals and the big object bounding boxes are partitioned into four scales according to the size. Large region proposals and bounding boxes are down-sampled to match the size of the spatial relation filter.

A.7 **Spatial relation filter of “mouse” and “keyboard”**.

A.8 **Spatial relation filter of “mouse” and “monitor”**.

A.9 **Spatial relation filter of “toilet paper” and “toilet”**.

A.10 **Spatial relation filter of “faucet” and “sink”**.

A.11 **Precision-recall curves of the co-occurrence ConvNet.** The ConvNet is built on top of the Full AlexNet (R-CNN)’s results, and the curves are computed using $IoU \geq 0.3$.

A.12 **Examples of the detection results on some testing images.**
A.13 Comparison of methods on small objects and PASCAL. In both (a) and (b), a marker represents the mAP of a detector on an object category. Specifically, red represents Faster R-CNN on PASCAL objects, green represents our ContextNet on our small objects, light and dark blue represent DPM on PASCAL objects and our small objects, respectively.

A.14 The proposal patches that have the largest excitation to the neurons in $fc6$ of proposal sub-network. Please refer to the main text for further discussions.

A.15 The context patches that have the largest excitation to the neurons in $fc6$ of context sub-network. Please refer to the main text for further discussions.
Chapter 1

Introduction

1.1 Motivation

The car is an essential component of daily life in most parts of the world. Till 2016, over 250 million road vehicles are registered in the United States. This large and increasing number of vehicles is exacerbating traffic problems such as accidents and congestion. According to global road crash data [49], nearly 1.3 million people die in fatal traffic accidents each year, and the cost is $518 billion USD globally. It is anticipated that road traffic injuries will become the fifth leading cause of death if no action is taken.

Fatality data from 1899 to 2014 (Figure 1.1) reveals an interesting temporal variation. From 1899 to the 1970s, the yearly number of deaths exhibits an obviously increasing trend. Then, the number starts to go down gradually. The increasing pattern can be explained by more and more vehicles being produced and utilized by people, while the decreasing pattern can be explained by the advancement of crash mitigation technologies. In 1968, seat belts were mandated in the United States, and since then, various other passive safety technologies have been developed and equipped on vehicles. Those technologies, although only focused on reducing the
severity of crashes rather than avoiding crashes, saved many lives throughout the world.

![Motor vehicle deaths in the US by year from 1899 to 2014. Data from [189].](image)

Figure 1.1: Motor vehicle deaths in the US by year from 1899 to 2014. Data from [189].

Into the new century, it is widely believed that technology will continue increasing the efficiency and safety of transportation, and one of the major breakthroughs will be autonomous driving - a kind of active safety technology that aims to avoid crashes. According to studies [3](NHTSA, 2008) and [1](NHTSA, 2012), the majority of traffic crashes are caused by human error, including inappropriate maneuvers and distracted drivers. By automating the vehicles, human errors can be significantly reduced since a computer will never get drunk or be distracted, and it can be designed to often execute appropriate maneuvers that avert the crash entirely. In addition, smaller headway and better driving behaviors can be achieved, which will lead to less traffic congestion. Autonomous vehicles will also benefit specific groups. People who are too young, too old, or those that have disabilities and are not able to drive themselves will enjoy the mobility afforded by the new technology. In summary, autonomous driving will likely have a profound impact on society.
The idea of autonomous driving dates back to the 1930s. Originally, it was assumed that autonomous driving could only take place in special roadways where use was dedicated to automated vehicles. However, the DARPA Grand Challenge in 2005 opened up the mindset of the public, and it became accepted that putting intelligence into the system is possible to make the vehicles drive autonomously on non-exclusive roads. Nowadays, although the academia is still conducting cutting-edge research on autonomous driving, many industrial entities have entered this arena. For example, Google is intensively testing its self-driving cars in California; NVIDIA has released its autonomous driving development platform (Drive PX) to the market. IT companies such as Apple, Uber, Baidu, and traditional car manufacturers such as Mercedes, Volkswagen, Ford, and Toyota all have established research groups working on autonomous driving. Startups such as Zoox and Cruise Automation (acquired by General Motors) are also developing their own systems.

Autonomous driving is an extremely challenging problem because it requires high accuracy and reliability, while everything must be done in real time. For instance, if a regular machine learning algorithm can produce 99% accuracy, it will be recognized as an excellent algorithm. However, if the pedestrian detection algorithm on the autonomous driving vehicles can only guarantee 99% accuracy, this is not enough as the missing 1% may cause fatal accidents. So autonomous driving must utilize the best technologies at the frontier of computer vision, machine learning, and artificial intelligence. Currently, no existing algorithm in any related research field is perfect enough to guarantee such performance, so developing novel and more advanced algorithms for autonomous driving is of great importance both in academia and industry. In this thesis, we focus on the cognition aspect of autonomous driving, and we aim to propose novel computer vision algorithms to help the vehicle better understand its driving environment.
1.1.1 Primary sensor: LiDAR vs. camera

Currently, to guarantee functionality and reliability, perception for autonomous driving is built upon multiple sensor inputs, such as cameras, ultrasound, radars, laser scanners, etc., rather than any single one. Although a camera is always on the list of input sensors, in most of the demonstrated systems, it is not the key component. Instead, autonomous driving systems such as the Google self-driving car primarily rely on a special kind of laser scanner - LiDAR (Figure 1.2a). A LiDAR scanner has the appearance of a rotating hat on the top of an autonomous driving vehicle. While working, it shoots out an array of laser beams, which cover a 360 degree area around the host vehicle. When the laser beams hit any surface, e.g. ground or other vehicles, they are reflected back and captured by the LiDAR. After processing the reflected laser beams, a 3D point cloud of the environment around the host vehicle (Figure 1.2b) is reconstructed. Each point in the reconstruction includes a 3D coordinate and a reflectance rate. Thus, laser beams reflected from fluorescent surfaces, such as traffic signs and lane markings, can be easily differentiated according to their higher reflectance rate.

Figure 1.2: An example of LiDAR. (b) Upper: 3D point cloud generated by LiDAR, lower: the corresponding image of the environment, courtesy of the KITTI dataset.
Generally, LiDAR is a very useful sensor. Based on physics and geometry, the reconstructed 3D point cloud is very accurate and reliable, which significantly reduces the complexity of perception. However, there are drawbacks with LiDAR as well. First, the resolution of LiDAR is relatively low, since the scan of the environment can only be done progressively (see Figure 1.2b, upper). Second, although an accurate 3D point cloud reconstruction can be produced, much useful information is lost. For example, the color, texture, and appearance of objects are not preserved in the 3D point cloud. Due to this limitation, LiDAR cannot be used for detecting specific traffic signs or traffic lights. Third, the physical principle of LiDAR makes it impossible to function in rain or snow; rain drops and snowflakes reflect laser beams, which produces large errors in the 3D point cloud.

As human beings, we rely on our vision to perceive the environment. Thus, we believe that autonomous driving systems can be designed by similar principles with the camera playing an important role. It may be argued that the existing infrastructures have been built to be readily perceived by human vision rather than by laser scanners, consequently, vision has the fundamental advantage. Any infrastructure improvement allowing better perception by human beings should also benefit corresponding computer vision algorithms. All the drawbacks of LiDAR can be alleviated by using a camera, which only costs a few tens of dollars. Almost all the information regarding the environment can be faithfully stored in images, so the only thing we need to worry about is developing more advanced computer vision algorithms that can better utilize the image. In this thesis, we focus only on computer vision algorithms that take camera images as input.

1.1.2 Fundamental concept: recording vs. non-recording

Autonomous driving, especially autonomous driving in urban regions, is extremely challenging. To localize the vehicle and navigate it through a complex environment,
in addition to detecting each object of interest, the autonomous driving system needs to possess a high-level understanding of the scene. Various approaches have been proposed to perform this task, and all these approaches can be categorized into two concepts: recording and non-recording.

Recording is a trick which reduces the complexity of scene understanding and localization. An example of a system utilizing recording is the Google self-driving car. The basic idea is as follows: before letting a vehicle drive itself through a region, a human driver first drives the vehicle around and takes a recording of the region: LiDAR 3D point clouds, image sequences, etc. The recorded environment is then manually labeled by human workers. The elements being labeled are static but critical for driving, such as traffic signs, traffic lights, intersection and lane configurations. The next time the vehicle is navigating through the same region autonomously, it can localize itself by actively matching the sensor output to the annotated map in its database. For instance, before the vehicle enters an intersection, it already knows the configuration of the intersection and the location of all the traffic signs and traffic lights ahead. In this way, recording provides a shortcut to information that is necessary for a vehicle to navigate through the complex environment. Such information is usually much more difficult for computer vision algorithms to extract from raw sensor outputs on-the-fly. The recording approach is widely used for urban driving, primarily because traffic networks in urban regions are very complicated and we currently do not have an algorithm capable of understanding the urban street scene as well as the pre-recording and annotation method.

However, the drawbacks of the recording approach are obvious. Recording only partially solves the cognition problem since dynamic objects such as moving vehicles, cyclists and pedestrians cannot be recorded. Because such a system relies heavily on the annotated map for localization, the vehicle can only drive in regions that are already recorded. Whenever the vehicle is placed in a new region, it does not know
how to drive. Moreover, in order to support autonomous driving, the annotated map needs to perfectly coincide with the real world. As urban environment is subject to quick changes, the annotated map has to be updated frequently. If we are considering a small region, this may not be a big issue, but if we want the autonomous driving vehicle to go anywhere in the world, the scale of the map does matter.

In contrast to the recording approach, there is another concept that does not require any pre-recording of the region. The idea is similar to how human beings drive. Humans have very strong reasoning abilities. Based on prior knowledge, our reasoning skills can help us understand the situation even in a totally unfamiliar environment. When we human beings drive in an unknown region, sometimes we do feel uncomfortable, but we can use fundamental visual cues to support our driving, allowing us to continue navigation. Thus, we believe the optimal autonomous driving system should drive in a similar manner. Basically, the non-recording approach requires a knowledge-base to store high-level general knowledge of the most common driving scenes, and at the same time, it relies on a strong reasoning module to generalize the high-level knowledge to specific cases. Since such a non-recording system does not rely on any annotated map for localization and navigation, it can be deployed in any region. However, such a system does require much more advanced object detection and scene understanding algorithms, which may not be achieved by today’s computer vision technology. For example, with the annotated map, a system knows the exact location of each traffic light, so none of them will be missed. In contrast, if the system has no knowledge of the locations of traffic lights beforehand, the detection algorithm has to actively search for them in every input image, which is very likely to suffer from false positives and false negatives. Similarly, with the annotated map, street scene understanding algorithms may not be very critical, since the vehicle already knows the configuration of the road ahead. However, without the annotated map, it does need an advanced scene understanding algorithm to estimate the configuration.
In summary, both concepts have their pros and cons. The recording approach is more practical currently, and it could be a shortcut to autonomy. But to fundamentally achieve autonomous driving in the end, we must focus on algorithms that lead to the non-recording approach. In this thesis, we assume no pre-recorded information is used, and algorithms are proposed following the non-recording concept.

1.1.3 Virtual environment for system development

Different from ordinary computer vision problems, the vehicle perception is conducted in a dynamic interactive environment. To illustrate, assume at time step $t$ the autonomous driving system takes in a view of its surroundings. After processing, the output steering angle and acceleration/brake level are sent back to the vehicle’s mechanical system to drive it through $t$ to $t + 1$. The new view of the surroundings at $t + 1$ is generated by altering the view at $t$ according to the vehicle’s motion. So in this sense, the current output of the autonomous driving system will affect its future input. Such a mechanism is recognized as a closed-loop feedback system, which is demonstrated in Figure 1.3.

Figure 1.3: Closed-loop testing for the autonomous driving system.
Testing such a system is more difficult than testing non-feedback ones. A pre-recorded video is not eligible for testing the autonomous driving system, since everything in the video is fixed. To evaluate how the system works, we have to let it drive in an interactive environment. The most ideal case is having a real vehicle testbed and a dedicated testing area, allowing the autonomous driving system to be tested in a real scene. However, such a setting is very expensive and requires resources which may not be available to most researchers. An alternative approach is to perform testing in a virtual environment/driving simulator, which is easy to set up, safe to operate, and cost-economic. Current computer graphics technologies are able to support fairly good virtual reality. A simulator can be created with realistic graphics models and physics engines, and the algorithms developed in the virtual environment can be further generalized to the real world.

In addition to providing a closed-loop interactive testing environment, a simulator has other features that a real environment can never achieve. In computer vision research field, algorithms for object detection, semantic segmentation, scene understanding, etc., are mostly machine learning-based and are usually trained in a supervised learning manner. This means we need to collect a large quantity of labeled data for training. Collecting ground truth labels for images is a very difficult process, and crowdsourcing is currently the most popular approach to generate labeled datasets in computer vision research. However, labeling a million-sample dataset through crowdsourcing is very expensive, and the manual labels are subject to human errors. Simulators that are based on computer graphics models provide an alternative way to solve the labeling problem. A simulator consists of a physics engine and a graphics engine. The physics engine is used to compute/update physical parameters of the virtual world at each simulation step, while the graphics engine is used to render each frame according to the physics engine’s output. With a simulator, the images and their corresponding ground truth labels can be automatically
extracted from the graphics engine and the physics engine, respectively. No human intervention is involved in the process, so the accurate labels are actually provided for free.

Another advantage of a simulator over the real environment for autonomous driving research is that we can create many dangerous and urgent situations that are very rare in real life. If we only rely on real data, we may not be able to collect enough samples for those rare but critical situations, preventing the algorithm from being properly trained for those situations. However, with a simulator, we can easily synthesize as many samples as needed. Moreover, if we want to train an autonomous driving AI agent using the reinforcement learning approach, the virtual environment is our only choice. In reinforcement learning, the AI agent learns to take optimal actions through a trial-and-error process. This means to learn “crash” is bad, the AI agent needs to receive a penalization from crashing first. Such a training scheme is infeasible in a real environment in which crashing is always intolerable.

Due to these many wonderful features, there is an increasing amount of work in the computer vision community that takes advantage of simulator/computer graphics models [190, 166, 101, 143, 165, 75, 12]. In this thesis, the majority of experiments are conducted within a driving simulator, and the model can be generalized to real world cases reasonably well. However, we do need to be careful about the domain difference issue [99, 149]. Images rendered from computer graphics models are different from real images. A powerful computer graphics model may reduce the gap but cannot eliminate it. So when we generalize a model that is only trained with synthetic images to the real world, we should expect a certain amount of degradation in performance. However, since simulators have become popular in computer vision research, the topic of “domain adaptation” has been considered by many researchers [54](Ganin & Lempitsky, 2014). We believe the domain difference issue will be solved soon.
1.1.4 System decomposition

Autonomous driving systems can be extremely complicated due to the inclusion of a large number of subsystems. In general, an autonomous driving system can be decomposed into three modules based on the major tasks (Figure 1.4): a cognition module, a decision making/AI module, and a control module.

![Diagram of System Decomposition](image)

Figure 1.4: Functionality decomposition of the autonomous driving system.

The cognition module serves as the eye of the autonomous driving system, and it converts the sensor outputs, e.g. images or 3D point clouds, into certain types of physical representations that can be directly processed by the downstream module. All the perception tasks for driving are covered in this module. However, besides “perceiving” the environment by detecting critical objects such as vehicles and lane markings, the cognition module also needs to “understand” the scene by discovering interactions between objects and extracting high-level information for localization and navigation. For example, when making a left/right turn in an urban intersection, the autonomous driving vehicle has to yield to pedestrians. Because pedestrians have various behaviors, the computer needs to understand each individual’s body language and correctly predict his/her intention before making any decision. Such a task lies beyond the domain of “perception” and requires real “cognition” of the driving environment.

The cognition results are passed to the decision making/AI module. This module serves as the brain of the autonomous driving system - it determines the high-level decisions that the host vehicle should take, such as planning a path ahead, switching to the adjacent lane, making a left turn in an intersection, slowing down, stopping,
accelerating, and so on. The decisions are then sent to the final module - the control module. The control module functions as the hand/foot of the autonomous driving system. It executes the decisions by implementing optimal control algorithms to compute the steering angle and acceleration/brake level, and these driving commands are directly applied to the host vehicle’s mechanical system.

1.1.4.1 Cognition module

Cognition is the first module in our decomposition of the autonomous driving system. It is crucial since it serves as the eye of the system. Without good sight, we human beings cannot drive a car, and this is also true for a computer. The cognition module is built upon computer vision technologies. There are many tasks within the domain of computer vision, and Figure 1.5 illustrates the major tasks that are related to autonomous driving.

![Diagram of Autonomous Driving Vision]

Figure 1.5: **Computer vision tasks that are related to autonomous driving.**

The most critical vision task for autonomous driving is object detection. In the cognition module, a collection of detectors are implemented on the input images to search for vehicles, pedestrians, lane markings, traffic signs, traffic lights, etc. Those detectors output markers (e.g. bounding box, splines) on top of the detections. Object tracking algorithms track detected objects across consecutive frames, and such algorithms are usually built upon a Kalman filter or particle filter. Semantic segmentation algorithms partition the input image into multiple regions, where each region is composed of pixels belonging to a single coherent object. They are very
helpful in discovering the road area (or drivable area) in images. Optical flow, as well as other motion-based algorithms, jointly processes multiple consecutive frames to generate the motion field of the input images. 3D reconstruction and visual odometry work similarly. They take a sequence of images as input, and can be used to localize the vehicle in a complex scene and compute the camera trajectory. Stereo vision is very useful in autonomous driving; it functions in the same way as our two eyes. By processing two displaced images (left view and right view), the algorithm outputs a corresponding depth map, from which the distance from each point to the camera can be derived. While all these tasks are jigsaw puzzle pieces of the cognition problem, scene understanding is a high-level task that aims to get a whole picture of the scene. Scene understanding can help to estimate the configuration of the road/intersection ahead and predict corresponding traffic patterns. Such information is crucial for the navigation and localization of a vehicle.

Only a few years ago, to solve each of the tasks illustrated in Figure 1.5, a spectrum of diverse algorithms needed to be proposed. However, “deep learning” has recently dominated the computer vision research community. Unlike any specific algorithm, deep learning is a very general methodology that can be used for many vision tasks as listed in Figure 1.5. In most tasks, the deep learning-based algorithms have demonstrated superior performance over their traditional counterparts. In this thesis, we primarily work on the cognition module and focus on developing novel deep learning-based algorithms for autonomous driving.

Although the entire autonomous driving system requires a closed-loop interactive environment for testing, the procedure can be simplified when conducting testing only on the cognition module. The cognition results are measurements of the physical world, which are objective and unique, so a certain type of “ground truth” always exists. Given a pre-recorded image sequence of the driving environment and the corresponding labels, error can be computed between the algorithm’s actual output and
the ground truth. If the cognition algorithm is effective, its output should accurately approximate the ground truth, thus achieving low error. In this sense, it is possible to build an open-loop system to evaluate the cognition module as illustrated in Figure 1.6. This idea partially reduces the complexity of testing. Thus, if we only focus on computer vision algorithms for cognition, a real testbed or a driving simulator is not necessary. We can take advantage of many publicly-available, pre-recorded driving datasets when developing the algorithms.

![Open Loop Testing Diagram](image)

Figure 1.6: **Open-loop testing for the cognition module.**

### 1.1.4.2 Decision making/AI module

Decision making/AI is the second module in our system decomposition, and it is equally as important as the cognition module. When the vehicle is driving on the road, a considerable number of situations could arise, from normal ones such as overtaking and car-following, to urgent ones such as emergency braking and collision avoidance. In all these situations, the computer that drives the vehicle should be designed to make “good” decisions. Moreover, driving occurs in a dynamic and stochastic environment in which things are changing continuously, so the decision making/AI module needs to make adjustment towards the environment continuously. This task is a tremendous challenge since it is almost impossible to have a complete list of situations based on
which we can program the system. Another challenge comes from the noisy input. No cognition algorithm can generate 100% accurate results. With better sensors and more advanced algorithms, the error may be reduced, but cannot be eliminated. So, the decision making/AI module needs to be robust enough to tolerate noisy input.

There are two major types of approaches to design the decision making/AI module: 1) the physical rule-based methods and 2) the machine learning-based methods, both of which have advantages and disadvantages.

**Physical rule-based methods** use a list to enumerate all possible situations that the vehicle may encounter when driving and corresponding optimal decisions. The situations are derived from the cognition results according to a set of rules, which can be computed using well known physical laws. The advantage of such physical rule-based methods is obvious. Since we “hard code” the rules and decisions, we know exactly the mechanism of the system, so it is straightforward to track the system’s performance. Whenever the system runs into a problem, it is fairly easy to figure out what caused it. However, the drawback of such methods is also obvious: they are not flexible. The autonomous driving vehicle can only respond to situations enumerated in the list. Whenever an undefined situation is encountered, the system will fail miserably. Even in some known situations, the physical laws for the scenario may be too complicated to be used to hard code a decision. As it is almost impossible to compose a complete list for all the situations in driving, we cannot design a decision making/AI module purely based on this type of methods.

**Machine learning-based methods** learn a map from input cognition results to output decisions through training. To develop a decision making/AI module in this way, a large quantity of data must be collected for training. As all the physical laws and rules are implied in the training data rather than explicitly specified, the algorithm is required to be powerful enough in order to learn such an abstract relation. The virtue of machine learning-based methods is flexibility. It is possible to learn
complicated and highly abstract relations between the input and the output from abundant training data. Since we treat the machine learning algorithm as a black box, we no longer need to painstakingly hard code physical laws, as they are automatically learnt as certain inner representations of the algorithm. However, the disadvantage of such methods can be fatal. Machined learning-based methods are hard to analyze. It is difficult to track the behavior of an algorithm and pinpoint the cause of any problem, since the output is computed based on certain inner mechanisms that are not obvious to a human. Such methods are also less reliable compared to the physical rule-based ones. When testing in similar situations, the system may run correctly for 99% of tests, but fail for the other 1%, and it is difficult to find the reason.

Based on the above observations, to design a decision making/AI module for real applications, we cannot simply rely on either the physical rule-based methods or the machine learning-based methods. A combination of the two should be a good choice. For normal driving, the system could primarily depend on the machine learning-based part. For urgent situations when absolute reliability is required, the system should refer to the physical rule-based part.

1.1.4.3 Control module

As the third module in our system decomposition, control is almost a solved problem. This module executes the driving decisions by computing the steering angle and acceleration/brake level. There are a number of elegant algorithms in the control research field that fully consider the dynamics of the vehicle by compensating for any delay or disturbance. But in industry, the most effective and reliable control method is still the PID (Proportional, Integral, Derivative) controller. For our normal driving, a PID controller should be adequate. However, if we want the vehicle to have quick response in extreme conditions (e.g. race car driving, drift, emergency control), it is still necessary to take the dynamics of the vehicle’s mechanical system into
consideration, and implement a more advanced control algorithm. Since control is not the focus of this thesis, only the most simple PID controller is implemented in our experiments.

1.1.4.4 Variants on system architecture

Our cognition-decision making/AI-control decomposition for autonomous driving system is proposed based on the functionality of major components. However, with different approaches, it is possible to combine some of the modules together.

As a very powerful machine learning tool, deep convolutional neural network can not only map images to cognition results, but also map images directly to driving decisions. In this case, the input of the network are the images, while the output of the network are the driving decisions, and there are no intermediate cognition results. The network functions as a combination of both the cognition module and the decision making/AI module, as shown in Figure 1.7.

![Figure 1.7: Combining the cognition module and the decision making/AI module.](image)

If such an idea is pushed one step forward, we can even use a deep convolutional neural network to map the image directly to the steering angle and acceleration/brake level, as shown in Figure 1.8. In this system, the cognition, decision making/AI, and control modules are implemented altogether as a single network. The entire system is a black box with images as input and driving commands as output. There are no explicit intermediate steps.
Figure 1.8: **Combining all the three modules as an end-to-end system.**

The deep convolutional neural networks shown in Figure 1.7 and Figure 1.8 can be trained using either the regular supervised learning approach or the reinforcement learning approach (known as deep reinforcement learning). If the networks are trained with the supervised learning approach, the optimal decisions/driving commands are used as the ground truth labels. If the networks are trained using the reinforcement learning approach, the supervision is provided through a reward function defined on the state space of the environment. A detailed discussion on reinforcement learning is covered in Chapter 4. Such systems may not be able to handle very complicated driving scenarios (e.g. urban driving), as they are limited by the effectiveness of the network. However, they are still good candidates for highway driving.

Figure 1.9: **Combining the decision making/AI module and the control module.**

It is also possible to combine the decision making/AI module and the control module using a single machine learning algorithm. In this case, we create a generalized control module that is responsible for both tasks. The traditional driving controller (e.g. PID) is no longer needed. The algorithm takes the cognition results as input and
outputs driving commands directly, as shown in Figure 1.9. Similarly, this machine learning-based module can be trained using either the supervised learning approach or the reinforcement learning approach.

1.2 Thesis outline

This thesis introduces our research work on designing computer vision algorithms for autonomous driving, and the primary focus is on proposing a deep learning-based direct perception approach for street scene understanding (Chapter 5 and Chapter 6). By successfully applying the deep convolutional neural network developed in a virtual environment to the real world, we demonstrate that simulators and computer graphics models can serve as extremely useful tools for autonomous driving research.

The rest of this thesis is organized as follows: Chapter 2 provides background information and a literature review of computer vision algorithms for autonomous driving, deep learning and object detection. Chapter 3 describes the motion-based object recognition method that differentiates planar objects according to their speed. Chapter 4 introduces the reinforcement learning approaches that can be used to train AI agents for autonomous driving. As the key part of the thesis, Chapter 5 describes our DeepDriving approach [25] (Chen et al., 2015) which uses a direct perception ConvNet to drive a car autonomously in a highway environment. In Chapter 6, the DeepDriving system is extended by incorporating temporal information. In particular, the Long Short Term Memory (LSTM) unit and its influence on street scene understanding are studied. Chapter 7 concludes this thesis and outlines potential future work. Appendix A proposes a deep learning-based small object detection approach. The applicability of the state-of-the-art object detection algorithms to the small object detection task is studied.
Chapter 2

Literature Review

In this chapter, materials on autonomous driving systems, computer vision algorithms for autonomous driving, deep convolutional neural network, and object detection are reviewed.

2.1 Autonomous driving systems

The idea of autonomous driving dates back to the 1930s. Since then, people are always dreaming to build a computer system that is able to drive a vehicle as well as a human driver. Due to the limitation of algorithms and processing power, early experimental systems only have acceptable performance in simple driving scenarios. ALVINN is a seminal project that is conducted by CMU NavLab researchers in the 1980s [147, 148] (Pomerleau, 1989 & 1992). In this project, a simple neural network with only one hidden layer is used to map input images directly to steering angles. This system can successfully follow the road, but cannot handle more complicated situations such as driving in traffic. Early work also includes designing autonomous vehicle for parking [141] (Paromtchik & Laugier, 1996) and collision avoidance [13] (Aufrère et al., 2003).

Autonomous driving research took off with the starting of the DARPA Grand Challenge in 2004, which was created to spur the development of technologies need-
ed to power fully autonomous vehicles navigating through off-road course within a limited time. The first event was held in the Mojave Desert region of the United States, 2004. No team finished the whole course [32]. In the following year, 2005, the second DARPA Grand Challenge event was held, where 5 teams out of 23 finalists finished the 132-mile course [173, 21, 113, 26, 11, 30, 70, 139]. The robot “Stanley” from Stanford University was the winner of the challenge [173].

In 2007, the third DARPA Grand Challenge, known as the “Urban Challenge” was held at the site of George Air Force Base. The site contains a 60 mile urban region course, where the participants were required to follow traffic regulations similar to urban driving. Six teams successfully finished the entire course, with the winner being the robot “Boss” from Carnegie Mellon University [181, 130, 15, 116, 20, 127, 142, 100]. There was no further DARPA Grand Challenge after 2007. However, some other challenges on autonomous driving are held, e.g. Grand Cooperative Driving Challenge [56] in Europe.

Since 2007, the autonomous driving research field attracts lots of attention from the industry - Google, NVIDIA, BOSCH, Uber, Mobileye etc. all have established their autonomous driving groups. Research on active safety is conducted intensively in the recent years [14, 51, 172], and such research products are promising to be commercialized soon.

### 2.2 Computer vision for autonomous driving

Computer vision algorithms are foundations of the cognition module, which is crucial to the successful implementation of an autonomous driving system. As the focus of this thesis is on computer vision algorithms for autonomous driving, the work in this area is reviewed first.
2.2.1 Object recognition

Object recognition refers to finding objects of interest in an image or an image sequence. Object recognition is so important that it is not only the core perception task of autonomous driving, but also a key research subject in the entire computer vision research field. There are two types of approaches to recognize objects in street scene images: motion-based methods and appearance-based methods.

- **Motion-based methods**

  Motion-based methods group the objects into four distinct categories according to their relative speed to the traffic flow (assume the traffic flow speed is $v$):

  1. Objects being stationary relative to the host vehicle such as vehicles in the same lane.

  2. Objects moving towards the host vehicle at a speed of $2v$ such as opposing traffic.

  3. Objects moving towards the host vehicle at $v$ such as stationary objects, e.g. trees, buildings.

  4. Objects moving that fit into none of the above categories, such as pedestrians and cyclists.

  The most common approach to deal with motion is the optical flow algorithm [86] (Horn & Schunck, 1981), which produces a dense motion field across the image plane by computing the difference between two consecutive frames. Optical flow is useful in robotic applications for being able to detect moving objects [171] (Talukder et al., 2003), and some modifications are made to the original algorithm to allow large displacement between the two consecutive frames [22] (Brox & Malik, 2011). Optical flow can also be estimated together with a depth map by implementing the scene
flow algorithm [115, 185, 81]. Time to Contact (TTC) methods, which compute the time until an object crosses the image plane, provide a potential way to solve the object recognition problem based on the aforementioned principles. TTC methods are effective in helping autonomous navigation robot to avoid obstacles, and the corresponding algorithms are computationally efficient [135, 134, 154, 8]. There are many ways to compute TTC, and it also can be easily derived from the motion fields produced by optical flow [10](Ancona & Poggio, 1995). [84](Horn et al., 2007) and [85](Horn et al., 2009) propose an Image Brightness Derivatives (IBD)-based TTC method. The method categorizes the object’s movement into four cases, and the calculation is built on the “constant brightness assumption”, which states that the intensity of the object point will be the same in images sampled at time $t$ and $t + \delta t$. This is the same assumption required by the optical flow algorithm. Scale invariant ridge segments method estimates the TTC by detecting and optimizing the characteristic size of the object in each image of the sequence [7](Alenyà et al., 2009). Active contours affine scale method estimates the TTC by tracking an active contour, as the TTC is directly related to the scale change of the contour [7](Alenyà et al., 2009). In this thesis we only focus on the direct method for TTC that is proposed by [84](Horn et al., 2007) and [85](Horn et al., 2009). Motion-based detection can also be implemented on scene flow results by explicitly tracking feature points. Then, the tracked feature points are clustered into multiple groups according to the motion patterns, and each group represents a coherent object. For example, TriTrack [115](Lenz et al., 2011) implements this idea to perform moving object detection and tracking in urban environments.

Motion-based methods only recognize moving objects, so if a car is parked on the street, it will be considered as part of the ground by such methods. Moreover, motion-based methods cannot determine the category of an object (which is necessary
in many applications), but only its movement. Thus, to tell a moving object is a “car” or a “pedestrian”, we have to turn to the appearance-based methods.

- **Appearance-based methods**

The appearance-based methods localize the object and determine its category according to a unique pattern/template of that category. The pattern/template is usually generated on image features, for example, Histogram of Oriented Gradients (HOG) [117, 31, 5, 183]. As appearance of objects varies from different view points, explicit modeling of object components is considered [46, 67, 72]. In autonomous driving, a few key categories need to be recognized, such as vehicles [45, 35], lane markings [9], road areas [80, 123], traffic signs [16, 34], traffic lights [44, 83, 27], and pedestrians [38, 145]. To guarantee fast response to the environment, sometimes, the system also needs to track and predict the motion of detected objects. Among all the object categories, predicting the motion of pedestrians is very challenging, since compared with vehicles and other categories, human behaviors are more random and subject to quick changes [102, 156, 207].

Typical object detection algorithms output bounding boxes on detected objects (or splines on detected lane markings). However, these bounding boxes and splines are just coordinates on the image plane, and they are not the direct affordance information (e.g. distance measure in the real world) we use for driving. Thus, a conversion is necessary which may result in extra noise. For example, typical lane detection algorithms such as the one proposed by [9] (Aly, 2008) suffer from false detections. Structures with rigid boundaries, such as highway guardrails or asphalt surface cracks, can be mis-recognized as lane markings. Even with good lane detection results, critical information for car localization may be missing. For instance, given that only two lane markings are usually detected reliably, it can be difficult to determine if a car is driving on the left lane or the right lane of a two-lane road. So in this thesis, a deep
learning-based method is proposed which embraces both the low level appearance-based object recognition and the high level scene understanding at the same time. The method outputs affordance information for driving directly instead of any bounding boxes or splines on the image plane.

### 2.2.2 3D street scene reconstruction and vehicle localization

Street scene reconstruction is crucial for the autonomous driving systems which rely on pre-recording of the environment [125](Meillard et al., 2015). Algorithms such as Simultaneous Localization and Mapping (SLAM) are also powerful tools for localizing the autonomous driving vehicles in complex scenes [109](Lategahn et al., 2011) [161](Song & Chandraker, 2015). Generally, large scale city-level reconstruction can be achieved by using the structure from motion algorithm and tens of thousands of images from Internet [53](Furukawa et al., 2010). However, for autonomous driving, the input images are more structured since all of them come from consecutive frames (of a video) produced by the same camera [6, 146, 107, 131], and the reconstruction algorithms need to run in real-time. Variants of the reconstruction algorithms can take monocular images [136](Newcombe & Davison, 2010), stereo image pairs [61](Geiger et al., 2011), or panoramic images [126](Micusik & Kosecka, 2009) as input, which makes the applications very flexible.

In the standard pipeline of 3D reconstruction, the algorithms estimate the camera pose for each frame and the transformation matrices between consecutive frames. These camera poses and transformation matrices can be used to compute visual odometry - the trajectory of the camera (also the trajectory of the host vehicle) [103, 88, 110]. Visual odometry is demonstrated to be useful in localizing the host vehicle in urban area where GPS signal is often blocked. For example, [23](Brubaker et al., 2013) proposes a probabilistic model that matches a short piece of vehicle trajectory produced by visual odometry algorithm to a freely available road map which
consists of more than 2,150km of drivable roads. According to the authors, the proposed algorithm is able to localize the host vehicle within 3 meter accuracy after only a few seconds of driving on the map.

### 2.2.3 Scene understanding

Achieving autonomous driving in complex scenes such as urban environment requires higher level of “knowledge” beyond simple recognition of individual objects. Scene understanding is an area in computer vision that extracts such high-level information from images. For urban driving, scene understanding algorithm is usually built on the results of other low-level vision algorithms, such as object detection, vanishing points, optical flow, depth maps, and aims to draw a consistent world representation of the environment around the host vehicle \[57, 60, 58, 201\]. Due to the complexity, scene understanding for driving is still primitive and not well explored in autonomous driving research field, only a few papers are published on this subject. \[55\] (Geiger, 2013) makes an attempt to tackle the problem of 3D urban traffic scene understanding. The proposed system takes image sequence as input, and no GPS or annotated map are used. The system has prior knowledge on layouts of the the most common intersections and the frequently experienced traffic patterns in those intersections. When the host car approaches an intersection, the algorithm tries to estimate the specific layout and the traffic pattern based on the prior knowledge. The algorithm extract five categories of cues from the image sequence: tracklets (track traffic vehicles and their movement in the intersection), vanishing line (vanishing points of both crossing roads), semantic labels (segment the image into small patches, and assign a category label to each patch), scene flow (the host car’s egomotion and other objects’ velocity vector), and occupancy grid (calculated from depth map, gives the information of drivable area). A probabilistic generative model is proposed to process the cues and estimate the most possible scene. The work is still a preliminary experimental
system, which only takes vehicles into consideration, while pedestrian, cyclist, and many other elements are ignored.

2.2.4 Datasets for autonomous driving research

Datasets are crucial for developing computer vision algorithms. To facilitate autonomous driving research, a number of publicly available datasets are proposed. [69] (Geiger et al., 2013) describes the KITTI dataset, which contains recordings from multiple sensors (e.g. stereo camera, LiDAR, IMU, GPS) while the host car was driving around in an European city by a human driver. Part of the dataset is detailedly annotated [194] (Xie et al., 2015), which can be used for object recognition, optical flow, depth estimation, 3D reconstruction, and street scene understanding. [89] (Huang et al., 2010) introduces the data collected by MIT’s autonomous vehicle during the 2007 DARPA Urban Challenge. The Cityscapes Dataset [29] (Cordts et al., 2015) focuses on semantic understanding of urban street scenes, and it contains a diverse set of stereo video recordings from 50 cities. The German Traffic Sign Benchmark [87] (Houben et al., 2013) is presented to facilitate the development of robust and accurate traffic sign detectors.

2.3 Decision making for autonomous driving

Based on the cognition results, the decision making/AI module computes driving decisions for the host vehicle to execute. For example, path planning is among the tasks of decision making, where the objective is to negotiate what is expected to be an optimal trajectory ahead [90, 152, 52]. The decision making/AI module can be implemented through the physical rule-based approaches or the machine learning approaches. As urban driving scenes are complex, training decision making/AI module through a large amount of data is a common approach [200] (Ziebart et al.,
In such a decision making/AI module, a knowledge-engine as introduced by (Saxena et al., 2014) and (Wang et al., 2015) should be built to incorporate prior knowledge for driving. In this thesis, the applicability of reinforcement learning on decision making for driving is evaluated. Reinforcement learning is a very effective approach for AI research and robot design. In the training pipeline, reward of status rather than the actual label of optimal actions are used as supervision. (Abbeel & Ng, 2004) proposes an apprenticeship learning algorithm, which extends the ordinary reinforcement learning approach by learning the reward function from an expert’s demonstration. (Mnih et al., 2013) and (Mnih et al., 2015) design a vision-based AI agent that is capable of playing ATARI games even better than human players. They use a neural network to learn the Q-value of each state through the reinforcement learning approach. (Koutník et al., 2013) and (Koutník et al., 2013) train a large recurrent neural network to drive a car in a driving simulator using reinforcement learning. The reward in the training process is extracted from the game engine. The network maps the image directly to the steering angles, and the objective is to keep the car on track. The network takes edge detection results of the raw image as input instead of using the actual images, which limits the system to very simple driving task such as track following.

2.4 Deep learning and convolutional neural network (ConvNet)

Currently, deep learning, in particular, convolutional neural network, is very popular in computer vision research field. Unlike any specific computer vision algorithm, deep learning is a very general methodology that can be used for many vision tasks, e.g. object detection, semantic segmentation, scene understanding, stereo vision, etc. In most tasks, the deep learning-based algorithms have demonstrated superior
performance over their traditional counterparts. Furthermore, deep learning-based algorithms are capable of learning high-level and abstract relationship between the input and the output, which makes them a perfect solution for advanced and complex tasks such as scene understanding. The convolutional neural network was first applied for classifying low-resolution images of handwritten digits \cite{112}(LeCun et al., 1990) in 1990. It became popular after \cite{106}(Krizhevsky et al., 2012) won the Image Large Scale Visual Recognition Challenge (ILSVRC) in 2012. Before reviewing the research work, the basics of convolutional neural network are described here.

2.4.1 Basics of deep learning

Deep convolutional neural network (ConvNet) is a generalization of the traditional neural network that explicitly transform the input to develop the output. Unlike traditional neural networks whose input is a one dimensional array, the input to a ConvNet is an image, which has three dimensions: width, height and color channels (RGB), and the spatial information of neighboring pixels can be preserved. There are a few basic building blocks in ConvNet, and the key one is convolution.

- Convolution

The convolution operation takes a feature map as input, which is of width \(w\), height \(h\), and channel \(c\). An image can be considered as a special feature map with channel \(c = 3\) for red, green, and blue. A filter convolves with the input feature map in a sliding window fashion. The size of the step that the filter takes when it moves across the feature map plane is denoted as stride \(d\). In practice, the filter is usually a square with width \(k\), and it must have the same number of channels as the feature map. The convolution operation produces a new feature map of width \(w' = \lceil(w - k)/d\rceil + 1\), height \(h' = \lceil(h - k)/d\rceil + 1\), and channel 1. Each pixel in this new feature map corresponds to a \(k \times k \times c\) cube in the original input feature map.
The convolution operation is illustrated in Figure 2.1. In each convolutional layer of a ConvNet, there are usually multiple (denoted as $n$) filters of the same dimension, and each of them produces a $w' \times h' \times 1$ feature map. Then, these $n$ feature maps are concatenated across the channel dimension to form the final $w' \times h' \times n$ feature map as the output of this convolutional layer. And this $w' \times h' \times n$ feature map serves as input to the downstream convolutional layer. The weights of each filter in the convolutional layers are tunable parameters that are automatically learnt through training.

Figure 2.1: An example of convolution. The filter convolves with the input feature map and generates a new feature map.

- Pooling

Pooling is another important and unique operation in ConvNets, it allows small translation invariance and helps to reduce the size of the input feature map. Pooling only applies to each width*height ($w \times h$) plane of the input feature map, and the channel dimension $c$ is preserved. Two types of pooling are mostly implemented, the average pooling (Figure 2.2a) and the max pooling (Figure 2.2b). In a $k \times k$ average pooling, the resulting pixel is the average of all the $k^2$ pixels, while in a $k \times k$ max pooling, the resulting pixel is the maximum among the $k^2$ pixels. The $k \times k$ pooling window moves across the input feature map plane in a sliding window fashion, and
the size of the step that the pooling window takes is denoted as stride $d$. A new feature map of width $w' = \lceil (w - k)/d \rceil + 1$, height $h' = \lceil (h - k)/d \rceil + 1$, and channel $c' = c$ is generated by the pooling layer, as is shown in Figure 2.2c.

Figure 2.2: An example of pooling. (a) 2 * 2 average pooling. (b) 2 * 2 max pooling. (c) The pooling operation allows small translation invariance and helps to reduce the size of the input feature map.

- **Activation function**

  The activation function is very important to artificial neural networks since it introduces non-linearity to the system. Without it, the entire system remains linear and can be collapsed to a single layer. In traditional neural networks, sigmoid function:

  $$f(x) = \frac{1}{1 + e^{-x}} \quad (2.1)$$

  is used as the activation function. Initially, sigmoid function is also used by ConvNets. However, sigmoid function requires careful scaling of the weights to prevent large output values which would lead to near zero-gradients and slow learning. Thus, it is replaced by Rectified Linear Unit (ReLU) of the form:

  $$f(x) = \max(0, x) \quad (2.2)$$

  Although ReLU is such a simple function, it has much better properties than sigmoid function in training deep ConvNets.
• Fully connection and dropout

In ConvNets, there is also a fully connection operation, which is exactly the same as the dense feed-forward connection in traditional neural networks. Since in a fully connected layer, the output is densely connected to the input, spatial information preserved in the feature map is lost. So the output of a fully connected layer is just a one dimensional array. In practice, ConvNets are often of huge scales with tens of millions of parameters (and fully connected layers constitute the majority of them), so they are vulnerable to overfitting. To reduce the effect of overfitting, a special layer called dropout is usually inserted after fully connected layers. The dropout, whose input and output are both one dimensional arrays, is illustrated in Figure 2.3. In the training phase, during every forward and backward pass, each neuron is randomly deactivated (by setting the activation to zero) with probability 0.5. In the testing phase, the weights of all neurons are halved, and all of them are used for prediction.

![Figure 2.3: An example of dropout.](image)

• Training the ConvNets

The ConvNets are trained with the stochastic gradient descent algorithm, in which the gradient of loss function is back-propagated from the network’s output to its input. General rules for training traditional neural networks still apply to training the ConvNets. In practice, mini-batch training is implemented. In each forward and backward pass, a mini-batch of \( n \) images (rather than only 1 image) are processed together by the network, where the back-propagated gradient is the average of those
individual gradients produced by each of the $n$ images. Mini-batch training makes the gradient more stable and accelerates the convergence speed.

### 2.4.2 Deep learning-based research and applications

Although the most successful application of convolutional neural network is image classification [106](Krizhevsky et al., 2012), researchers soon discovered ways to apply it to other computer vision tasks, and most of them get superior performance over traditional approaches. For example, now deep learning is capable of detecting edges [19](Bertasius et al., 2015), predicting new views [48](Flynn et al., 2015), computing optical flow [17](Fischer et al., 2015) [186](Walker et al., 2015), estimating depth map [41](Eigen et al., 2014) [199](Zbontar & LeCun, 2015), predicting saliency map [140](Pan & Nieto, 2015), deblurring [157](Schuler et al., 2014), implementing semantic segmentation [76](Hariharan et al., 2014) [121](Long et al., 2015), and detecting objects. It is widely recognized that the deep features learnt by ConvNets are much more powerful than the traditional manually engineered features such as GIST [138](Oliva & Torralba, 2001), and the deep features are generally applicable to many different vision tasks. Thus, lots of work is conducted to learn deep features from large quantities of weakly supervised data [98](Joulin et al., 2015) [37](Doersch et al., 2015). Specifically, in driving related scenarios, [96](Jayaraman & Grauman, 2015) and [4](Agrawal et al., 2015) show that the deep features can be learnt from motion, and [50](Fragkiadaki et al., 2015) introduces an approach to segment moving objects with ConvNet.

To further improve the effectiveness of ConvNets, large amount of theoretical work is conducted. Many fundamental modifications are made to the network structure and training pipeline, which result in even more powerful deep learning models. For example, batch normalization [92](Ioffe & Szegedy, 2015) accelerates the training process by reducing internal covariate shift. [78](He et al., 2015) introduces the Parametric
Rectified Linear Unit (PReLU) layer that significantly reduces the classification error on ImageNet dataset. [79] (He et al., 2015) introduces the spatial pyramid pooling that drastically improves the mean average precision on object detection. [202] (Zhang et al., 2015), [111] (Lavin, 2015), and [92] (Iandola et al., 2015) study the approaches that could make the ConvNet training process faster and more efficient. Experiments discover that the deeper a ConvNet is, the more powerful it is. However, training a very deep ConvNet is not an easy task [69] (Glorot & Bengio, 2010). Some research work is done on the design of network structure to support training very deep networks [160, 170, 77, 164]. As ConvNets have superior performance over traditional computer vision algorithms in many tasks, researchers are curious about why they are so powerful. [17] (Bakry et al., 2015) and [18] (Bengio et al., 2015) try to answer this question by exploring the nature of ConvNet. Visualization is another effective way to partially understand the mechanism of ConvNet. Empirically, the visualization is generated from feature maps, deep representations, filters and reconstructions [40, 122, 159, 200, 119], and these work gives us insights into the deeper nature of the network.

Recurrent network is a special kind of neural network where the historical output is part of the current input. If unfolded through time, recurrent network has much deeper structure than the regular feed-forward neural networks. Due to the recurrent input, such networks can incorporate temporal information and memorize historical status, so they can achieve tasks such as [124] (Mao et al., 2014) and [182] (Vinyals et al., 2015) that cannot be achieved by regular feed-forward neural networks. Currently, combining memory modules [82] (Hochreiter & Schmidhuber, 1997) with ConvNets is a very popular direction in deep learning research [167] (Sukhbaatar et al., 2015). The applications of these memory networks are mostly in image captioning, video description [39] (Donahue et al., 2015), video labeling [197] (Ng et al., 2015), visual question & answer [196] (Yu et al., 2015), learning representations for videos [163] (Srivastava et
al., 2015), and building neural Turing machine [71](Graves et al., 2014) [198](Zaremba 
& Sutskever, 2015). In this thesis we create a new application for the memory module 
by using it to incorporate temporal information for street scene understanding.

2.4.3 Deep learning for driving

Observing the excellent performance of deep learning, we believe that autonomous 
driving research will also benefit from it. Actually, ConvNets were applied in 
autonomous off-road robots even before they became widely recognized in 2012. 
[133](Muller et al., 2005) proposes a ConvNet-based off-road driving robot DAVE 
that learns a mapping from images to a human driver’s steering angles. After 
training, the robot demonstrates the capability of obstacle avoidance. In 2009, 
[74](Hadsell et al., 2009) proposes an autonomous off-road driving robot with self-
supervised learning ability for long-range vision, which can overcome the myopia of 
typical stereo vision systems. In the system, a multi-layer convolutional network 
is used as an effective feature extractor, with which a classifier determines if an 
image segment represents a traversable area or not. The self supervisory labels for 
the learning process are automatically computed from the short-range stereo vision. 
Recently, [91](Huval et al., 2015) describes a ConvNet-based system that detects 
vehicles and lane markings for highway driving. The proposed system is developed 
using OverFeat, and runs in real time. The results further prove that deep learning 
holds promise for autonomous driving. Other work related to autonomous driving 
are: DeepFlow [188](Weinzaepfel et al., 2013) implements ConvNet to compute large 
displacement optical flow, where the proposed method achieves very good results for 
driving scene images from the KITTI dataset. [94](Jain et al., 2015) and [95](Jain 
et al., 2015) use recurrent networks to anticipate driving maneuvers (of a human 
driver) a few seconds before they occur, and alert the driver if a dangerous maneuver 
is going to be conducted.
2.5 Object detection

Object detection is one of the most important tasks for computer vision research. Since so many critical objects need to be detected while driving autonomously, it is also the most crucial element for computer vision-based autonomous driving system. In this section, a comprehensive review on this subject is made, and the emphasize is on deep learning-based object detection approaches.

As object detectors are primarily machine learning-based algorithms, datasets play a very important role. Pascal VOC [43] (Everingham et al., 2010) is the most widely used benchmark dataset for object detection. Different from the object-centered datasets such as Pascal, the Microsoft COCO [118] (Lin et al., 2014) is composed of images of complex everyday scenes with common objects in their natural context. The goal is to advance the object detection techniques by placing it in the context of the broader question of scene understanding. The Scene UNderstanding database (SUN) [192] (Xiao et al., 2014) consists of a variety of environmental scenes, where both scene and object labels are available. All the objects in SUN are placed in there natural scenes.

Before the deep learning era, the Deformable Part Model (DPM) [46] (Felzenszwalb et al., 2010) is a pretty successful algorithm. The method represents highly variable objects using mixtures of multi-scale deformable part models, specifically, a coarse root filter, several higher resolution part filters, and a spatial model for the location of each part relative to the root. The filters are developed using Histogram of Oriented Gradients (HOG) features.

Adopted by DPM, the sliding window approach is expensive since bounding boxes are proposed densely across the image plane. Thus region proposal methods, which only generate a small number of proposals per image while preserving the high recall for the target objects, becomes popular in the research field. Selective search [179] (Uijlings et al., 2013) is one of the commonly used region proposal methods for
object detection. By using multiple types of low-level cues and searching functions, it aims to capture all possible object locations. (Zitnick & Dollár, 2014) proposes a novel method for generating object bounding box proposals using edges. The authors argue that the number of contours fully covered by a bounding box indicates the likelihood of the box containing an object. Most object proposal approaches use bottom-up cues to rank proposals, however, (Kuo et al., 2015) argues that the “objectness” is actually a high-level construct, thus only relying on those low-level cues will not generate good proposals. The paper proposes a simple four layer ConvNet architecture to re-rank proposals from a bottom-up method.

(Girshick et al., 2014) introduces the widely used R-CNN pipeline for deep learning-based object detection. The proposed algorithm combines convolutional neural networks with bottom-up region proposals. In the testing phase, the method generates around 2000 category-independent region proposals for the input image using selective search, and then extracts a fixed-length feature vector from each proposal using ConvNet, and finally classifies each region with category-specific linear SVM. Fast R-CNN (Girshick, 2015) is a modification of the pipeline proposed by Girshick et al., 2014). The new framework trains networks using a multi-task loss in a single training stage, which both simplifies the learning process and improves the detection accuracy. While (Girshick, 2015) substantially improves the speed of the detection network in the R-CNN pipeline, the time-consuming selective search-based region proposal generation is intact. (Ren et al., 2015) proposes a Region Proposal Network (RPN) that shares convolutional layers with the detection network, thus enabling almost cost-free region proposals. The RPN is fully convolutional and it simultaneously predicts coordinates and objectness scores at each box proposal. (Erhan et al., 2014) proposes a ConvNet model that directly predicts a set of class-agnostic bounding boxes along with a single score for each box, corresponding to its likelihood of containing any object of interest. The class-agnostic bounding box
predictions are then fed into a second ConvNet which classifies each input region into different categories. Although DPM and ConvNet are typically considered as different approaches, [66](Girshick et al., 2014) provides a novel synthesis of the two ideas by formulating DPM as a ConvNet. [204](Zhang et al., 2015) introduces a Bayesian optimization-based algorithm that iteratively searches for better bounding boxes for object detection. [144](Pepik et al., 2015) shows the overall performance of object detection can be improved when image renderings for data augmentation are used.

There is a lot of research on methods to localize objects in images other than generating image specific region proposals. [114](Lenc & Vedaldi, 2015) develops and evaluates a ConvNet-based detector that uses a trivial region generation scheme that is constant for each image. The region proposals are computed from the statistics of Pascal VOC training data. The proposed pipeline results in a fast detector and has good performance on Pascal VOC dataset. [195](Yoo et al., 2015) casts object detection as an iterative classification problem. The proposed AttentionNet iteratively estimates quantized weak directions pointing to a target object that converge to an accurate object boundary box. [158](Sermanet et al., 2013) introduces an integrated framework for classification, localization and detection using ConvNet. The proposed network is implemented efficiently in a multi-scale sliding window fashion. [150](Redmon et al., 2015) treats object detection as a regression problem of spatially separated bounding boxes and associated class scores. The proposed ConvNet predicts bounding boxes and class scores from the input images in only one forward pass.

Generally, context, which implies the relation between an object and its environment (or other objects), is useful for improving the object detection performance in natural scenes [175](Torralba et al., 2003) [36](Divvala et al., 2009). Based on R-CNN, [68](Gkioxari et al., 2015) proposes a pipeline for action recognition using more than one regions. [68](Gidaris & Komodakis, 2015) proposes a multi-region
object detection system that can steer the ConvNet to focus on different regions of the object. [205](Zhu et al., 2015) uses both segmentation and context to improve object detection accuracy. [132](Mottaghi et al., 2014) studies the role of context in existing object detection approaches and further proposes a model that exploits both the local and global context. [73](Gupta et al., 2015) explores the influence of person context and local scene context on object detection. [28](Cinbis & Sclaroff, 2012) introduces an object detection method that is based on set representations of the contextual elements.
Chapter 3

Time to Contact: Recognizing Moving Objects Using Motion Patterns

3.1 Introduction

Recognizing objects is a crucial task for autonomous driving. Computer vision algorithms generally rely on two types of methods to recognize objects: 1) motion-based methods and 2) appearance-based methods. In particular, motion-based methods can group the objects into four categories according to their relative speed to the traffic flow (assume the traffic flow speed is $v$):

1. Objects being stationary relative to the host vehicle such as vehicles in the same lane.

2. Objects moving towards the host vehicle at a speed of $2v$ such as opposing traffic.

3. Objects moving towards the host vehicle at $v$ such as stationary objects.
4. Objects moving that fit into none of the above categories, such as pedestrians and cyclists.

Since driving fundamentally involves managing of the relative motion of objects, an algorithm that differentiates objects based on their motion patterns is investigated in this chapter. The Image Brightness Derivatives (IBD)-based Time to Contact (TTC) algorithm is introduced and its applicability to autonomous driving is investigated. The algorithm is formed to be simple, intuitive, and computationally efficient.

3.2 Basics of Time to Contact

This section introduces the procedure of deriving the IBD-based TTC method. All the following deduction are based on [84](Horn et al., 2007) and [85](Horn et al., 2009).

First of all, define a camera coordinate system where the center of projection (of the camera) is the origin, and a point in the system is represented by a coordinate \((X,Y,Z)\). As the name shows, Time to Contact (TTC) is the ratio of distance to velocity:

\[
T = -\frac{Z}{\frac{dZ}{dt}}
\]  

(3.1)

where \(Z\) is the distance between the center of projection and the object. Assume the “Constant Brightness Assumption” is satisfied, which is:

\[
\frac{d}{dt} E(x, y, t) = 0
\]

(3.2)

where \(E(x, y, t)\) is the brightness of the image at point \((x, y)\) and time \(t\). The above derivative can be expanded into:

\[
u E_x + v E_y + E_t = 0
\]

(3.3)
where \( u = dx/dt \) and \( v = dy/dt \) are the \( x \) and \( y \) components of the motion field in the image, while \( E_x, E_y, \) and \( E_t \) are the partial derivatives of the brightness w.r.t. \( x, y, \) and \( t. \)

According to the perspective projection, we have the following equations:

\[
\begin{align*}
x &= \frac{X}{f} - \frac{X}{Z} \frac{W}{Z} \\
y &= \frac{Y}{f} - \frac{Y}{Z} \frac{W}{Z} \\
\end{align*}
\] (3.4)

where \( f \) is the focal length of the camera. Differentiate the above equations w.r.t \( t, \) yields:

\[
\begin{align*}
u &= \frac{U}{f} - \frac{X}{f} \frac{W}{Z} \\
v &= \frac{V}{f} - \frac{Y}{f} \frac{W}{Z} \\
\end{align*}
\] (3.5)

where \((u,v) = (\dot{x}, \dot{y})\) and \((U,V,W) = (\dot{X}, \dot{Y}, \dot{Z}).\)

Substituting Equation 3.4 into Equation 3.5 yields:

\[
\begin{align*}
u &= \frac{U}{f} - \frac{x}{f} \frac{W}{Z} \\
v &= \frac{V}{f} - \frac{y}{f} \frac{W}{Z} \\
\end{align*}
\] (3.6)

or

\[
\begin{align*}
u &= \frac{1}{Z} (fU - xW) \\
v &= \frac{1}{Z} (fV - yW) \\
\end{align*}
\] (3.7)

Then according to the moving pattern of the planar object relative to the camera, the problem can be further divided into four cases \[85\] (Horn et al., 2009), which are shown in Figure 3.1.
• Case I: translational motion along the optical axis towards a planar object perpendicular to the optical axis.

This is a simplified case where $U = V = 0$. Substituting $u = -x(W/Z)$ and $v = -y(W/Z)$ into Equation 3.3 yields:

$$-\frac{W}{Z}(xE_x + yE_y) + E_t = 0 \quad (3.8)$$

or

$$CG + E_t = 0 \quad (3.9)$$

where $C = -W/Z$ and $G = (xE_x + yE_y)$.

In order to get the TTC, formulate a least squares method that minimizes:

$$\sum (CG + E_t)^2 \quad (3.10)$$

Differentiating it w.r.t. $C$ and setting the result equal to zero yields:

$$\sum (CG + E_t)G = 0 \quad (3.11)$$

or

$$C \sum G^2 = - \sum GE_t \quad (3.12)$$
Then \( C \), the inverse of TTC, is given by:

\[
C = -\sum \frac{GE_t}{\sum G^2}
\]  

(3.13)

- **Case II: arbitrary translational motion relative to a planar object perpendicular to the optical axis.**

Substituting Equation 3.7 into Equation 3.3 yields:

\[
AE_x + BE_y + CG + E_t = 0
\]

(3.14)

where \( A = f(U/Z), B = f(V/Z), \) and \( C = -W/Z \).

In order to get the TTC, formulate a least squares method that minimizes:

\[
\sum (AE_x + BE_y + CG + E_t)^2
\]

(3.15)

Differentiating it w.r.t. \( A, B, C \) and setting the results equal to zero yields:

\[
\sum (AE_x + BE_y + CG + E_t)E_x = 0
\]

\[
\sum (AE_x + BE_y + CG + E_t)E_y = 0
\]

(3.16)

\[
\sum (AE_x + BE_y + CG + E_t)G = 0
\]

or

\[
A \sum E_x^2 + B \sum E_xE_y + C \sum GE_x = - \sum E_tE_x
\]

\[
A \sum E_xE_y + B \sum E_y^2 + C \sum GE_y = - \sum E_tE_y
\]

(3.17)

\[
A \sum GE_x + B \sum GE_y + C \sum G^2 = - \sum GE_t
\]

Then solving the linear equations yields \( C \), the inverse of TTC.
Case III: translational motion along the optical axis relative to an arbitrary planar object.

Assume the plane equation is:

$$Z = Z_0 + pX + qY$$  \hspace{1cm} (3.18)

Substituting $X = (x/f)Z$ and $Y = (y/f)Z$, we obtain:

$$Z \left(1 - p\frac{x}{f} - q\frac{y}{f}\right) = Z_0$$  \hspace{1cm} (3.19)

Substituting it into Equation 3.9 yields:

$$-GW\frac{W}{Z_0} \left(1 - p\frac{x}{f} - q\frac{y}{f}\right) + E_t = 0$$  \hspace{1cm} (3.20)

or

$$G(C + Px + Qy) + E_t = 0$$  \hspace{1cm} (3.21)

where $P = (p/f)(W/Z_0)$, $Q = (q/f)(W/Z_0)$, and $C = -W/Z_0$. Since the translational motion is along the optical axis, the contact point on the plane is $(0, 0, Z_0)^T$.

In order to get the TTC, formulate a least squares method that minimizes:

$$\sum [G(C + Px + Qy) + E_t]^2$$  \hspace{1cm} (3.22)

Differentiating it w.r.t. $P$, $Q$, $C$ and setting the results equal to zero yields:

$$\sum [G(C + Px + Qy) + E_t]Gx = 0$$

$$\sum [G(C + Px + Qy) + E_t]Gy = 0$$  \hspace{1cm} (3.23)

$$\sum [G(C + Px + Qy) + E_t]G = 0$$
or

\[ P \sum G^2 x^2 + Q \sum G^2 xy + C \sum G^2 x = - \sum G x E_t \]

\[ P \sum G^2 xy + Q \sum G^2 y^2 + C \sum G^2 y = - \sum G y E_t \]  \hspace{1cm} (3.24)

\[ P \sum G^2 x + Q \sum G^2 y + C \sum G^2 = - \sum G E_t \]

Then solving the linear equations yields \( C \), the inverse of TTC.

- **Case IV: arbitrary translational motion relative to an arbitrary planar object.**

Substitute Equation 3.7 and Equation 3.18 into Equation 3.3 and following the previous definition of \( A, B, C, P, \) and \( Q \), we arrive at:

\[ C \left( 1 + x \frac{P}{C} + y \frac{Q}{C} \right) \left( G + E_x \frac{A}{C} + E_y \frac{B}{C} \right) + E_t = 0 \]  \hspace{1cm} (3.25)

In order to get the TTC, formulate a least squares method that minimizes:

\[ \sum \left[ C \left( 1 + x \frac{P}{C} + y \frac{Q}{C} \right) \left( G + E_x \frac{A}{C} + E_y \frac{B}{C} \right) + E_t \right]^2 \]  \hspace{1cm} (3.26)

or

\[ \sum [CFD + E_t]^2 \]  \hspace{1cm} (3.27)

where \( F = 1 + x \frac{P}{C} + y \frac{Q}{C} \) and \( D = G + E_x \frac{A}{C} + E_y \frac{B}{C} \).

Equation 3.26 is nonlinear and needs to be solved numerically. If \( P/C \) and \( Q/C \) are given, then \( F = 1 + x \frac{P}{C} + y \frac{Q}{C} \) is known. Equation 3.26 is linear in terms of \( A, B, \) and \( C \) as:

\[ \sum [F * (CG + E_x A + E_y B) + E_t]^2 \]  \hspace{1cm} (3.28)
Thus, parameters $A$, $B$, and $C$ can be solved based on the following linear equations:

\[
A \sum F^2E_x^2 + B \sum F^2E_yE_x + C \sum F^2GE_x = - \sum FE_xE_t \\
A \sum F^2E_xE_y + B \sum F^2E_y^2 + C \sum F^2GE_y = - \sum FE_yE_t \\
A \sum F^2GE_x + B \sum F^2GE_y + C \sum F^2G^2 = - \sum FGE_t
\]  

(3.29)

Conversely, if $A/C$ and $B/C$ are given, then $D = G + E_xA/C + E_yB/C$ is known. Equation 3.26 is linear in terms of $P$, $Q$, and $C$ as:

\[\sum [(C + xP + yQ) * D + E_t]^2\]  

(3.30)

Thus, parameters $P$, $Q$, and $C$ can be solved based on the following linear equations:

\[
P \sum D^2x^2 + Q \sum D^2xy + C \sum D^2x = - \sum xDE_t \\
P \sum D^2xy + Q \sum D^2y^2 + C \sum D^2y = - \sum yDE_t \\
P \sum D^2x + Q \sum D^2y + C \sum D^2 = - \sum DE_t
\]  

(3.31)

Therefore, given an initial guess of $P/C$ and $Q/C$, the algorithm first solves Equation 3.29 for $A$, $B$, and $C$. Then with the new estimates of $A/C$ and $B/C$, it updates $P$, $Q$, and $C$ using Equation 3.31. These two steps are implemented alternatively, and after a few iterations the algorithm will yield an approximation that is close enough to the actual solution.

In the above deductions, four methods are proposed corresponding to four cases, respectively. However, as Case I, II, III are all special cases for Case IV, the Case IV method can be applied to all of them. Similarly, the Case II method can be applied to both Case I and Case II, while the Case III method can be applied to both Case I and Case III.
3.3 Implementation

The algorithm introduced in the previous section (proposed by [84](Horn et al., 2007) and [85](Horn et al., 2009)) is designed for single planar object cases. Although according to the authors, the algorithm has good performance in such cases, it is not useful for real applications, since few street scenes contain only one object. So we propose a TTC map method to generalize the original algorithm. The idea of the TTC map method is straightforward: first, segment the image into a large number of super pixels (the segmentation should preserves the edges in the image), and estimate a TTC value for each super pixel using the standard IBD-based TTC method. Second, assume the TTC of each super pixel can be reliably computed, and super pixels belong to the same coherent object have roughly similar estimated TTC values. Then, the super pixels can be aggregated into different objects based on the ranges of these estimated TTC values that are close to each other.

To test the applicability of this straightforward IBD-based TTC method to autonomous driving, the experiment setting is made as simple as possible. We record a few sequences of images of moving planar objects, and the planar surfaces are chosen to be checkboards or those having rigid changing patterns, since the method only works on changing pixels. While capturing each frame, the distance between the object and the camera is also recorded, so the ground truth is known. The ground truth is then used to evaluate the accuracy of the TTC method. Our self-made equipments for carrying out the experiments are shown in Figure 3.2. In total, 20 different sequences of images are collected, and they are named as “Sequence-01” to “Sequence-20”.

48
Figure 3.2: **Self-made equipments for carrying out the experiments.**

As [84](Horn et al., 2007) and [85](Horn et al., 2009) suggested, the following pre-processing steps are implemented:

- Set a threshold on $E_t$ (e.g. $E_t = 0.1$) to exclude non-moving pixels;

- Use the method proposed by [86](Horn & Schunck, 1981) to calculate $E_x$, $E_y$, and $E_t$;

- Pre-process the images with a low-pass filter (e.g. block averaging kernel of the size $7 \times 7$);
• Implement sub-sampling (to 1/3 of the original image resolution) to reduce the size of the images before computing TTC, which accelerates the processing speed without losing the estimation accuracy.

The algorithm pipeline is described in Figure 3.3. At each time step, two consecutive frames from the image sequence are processed by the TTC method, and the estimated TTC value corresponding to that time step is produced. In all the experiments, time (e.g. time step, TTC) is discretized and measured by “frames”. The actual time can be computed as frames * interval.

![Figure 3.3: Pipeline of computing TTC and TTC map.](image-url)
3.4 Results and analysis

3.4.1 Time to Contact for single object

The standard TTC method is implemented on the collected image sequences that only contain one planar object. An example result on Sequence-02 is shown in Figure 3.4. In this sequence, the checkboard is perpendicular to the optical axis, and the translational motion is along the optical axis (Case I scenario). The checkboard moves at constant speed and the result is computed using the Case I method.

Figure 3.4: TTC result of Sequence-02 using the Case I method. In (b) the blue line is the estimated TTC, and the red line is the ground truth. In the experiment, time is measured by “frames”.

Further TTC results are computed using methods for Case I to Case IV respectively, and the results on Sequence-06 are shown in Figure 3.5. In this sequence, the checkboard is perpendicular to the optical axis, and the translational motion is along the optical axis (Case I scenario), but the speed of the checkboard changes occasionally. From Figure 3.5 we observe that all the four methods work reasonably well in this case.

However, in Figure 3.5b ~ Figure 3.5d spikes emerge at some time steps, while the probable cause is random noise and poor quality of pixel values. In order to
compute the TTC value at each time step, a least squares method is formulated by the proposed TTC method. If the least squares method is ill-posed due to bad data points, its solution will have huge error, which is indicated by the spike in the figure.

![Graphs showing TTC results for different methods](image)

(a) Case I method  (b) Case II method  
(c) Case III method  (d) Case IV method

Figure 3.5: **TTC results of Sequence-06 using the four different methods.** The blue line is the estimated TTC, and the red line is the ground truth. In the experiments, time is measured by “frames”.

### 3.4.2 Time to Contact map for multiple objects

The proposed TTC map method is implemented on the collected image sequences which contain multiple planar objects. The segmentation is generated by SLIC [184],
which segments the images along potential object edges. Figure 3.6 displays some SLIC segmentation results, in which the average size of the segments (e.g. $regionSize$) is a tunable parameter.

![Segmentation results](image)

(a) Segmentation with $regionSize = 30$
(b) Segmentation with $regionSize = 100$

Figure 3.6: **Examples of SLIC segmentation.**

The TTC map results of Sequence-03 (Case I scenario) are shown in Figure 3.7. To display the TTC map, a fixed time step is randomly chosen, and the estimated TTC value that belongs to each super pixel at that time step is projected on to the image plane (Figure 3.7b $\sim$ Figure 3.7d). The TTC values are color coded, the warmer the higher the TTC value, the colder the lower the TTC value. The symbol $NaN$ is assigned to TTC values of the background super pixels, which are stationary and non-changing (represented by dark blue color in the figures). However, in some super pixels that belong to the moving objects, due to the random noise or poor quality of pixel values, the least squares method formulated is ill-posed and no valid solution exists, so in that case $NaN$ is also assigned to those TTC values (e.g. the holes on the planner objects in Figure 3.7b $\sim$ Figure 3.7d).
Figure 3.7: **TTC map results (at a randomly chosen time step) of Sequence-03 using various methods.** (b) ∼ (d) The TTC value of each super pixel is color coded and indicated by the colorbar.

In our assumption of the TTC map method, we assume super pixels that belong to the same coherent object will have roughly similar estimated TTC values, and then, they can be aggregated into different objects based on the ranges of these estimated TTC values that are close to each other. Figure 3.7(b), which is computed using the Case I method, shows such a desired result. In this figure, super pixels that belong to the front object have very similar TTC values (e.g. indicated as yellow-green in the figure), while super pixels which belong to the rear object also have very similar TTC values (e.g. indicated as red in the figure). So the super pixels can be aggregated into two different objects based on their TTC values. We refer to such a result as a “consistent” result. However, Figure 3.7(c) and Figure 3.7(d), which are computed using the Case II method and the Case IV method respectively, show two undesired results.
In both TTC maps, super pixels that belong to the same object have a diverse range of TTC values (e.g. indicated as dark red and light blue for neighboring super pixels in the figures), so we are no longer able to aggregate super pixels based on their TTC values. We refer to such results as “inconsistent” results.

To improve the inconsistent TTC map results (of the Case II method and the Case IV method), ridge regression [83] (Hoerl & Kennard, 1970) is implemented. In traditional least squares method (the one used in the standard TTC pipelines), a formula such as $\|Ax-b\|^2$ is minimized. However, due to the random noise or poor quality of pixel values in a super pixel, the data points that are fitted to this model may be badly shaped. So $\|Ax-b\|^2 + \|\Gamma x\|^2$ is minimized instead in ridge regression, and the optimal solution to $x$ is of the form $\hat{x} = (A^TA + \Gamma^T\Gamma)^{-1}A^Tb$. Usually $\Gamma$ is chosen as the identity matrix $I$, resulting in minimizer $\hat{x} = (A^TA + kI)^{-1}A^Tb$, where $k$ is a coefficient. If we choose $k = 0$, then the ridge regression degenerates to the traditional least square method.

The TTC map results (of the Case II method and the Case IV method) produced by implementing ridge regression on the same image sequence are shown in Figure 3.8. From Figure 3.8a ~ Figure 3.8d we observe that the TTC map results of the Case II method are improved after applying ridge regression (since Figure 3.8b ~ Figure 3.8d are similar to Figure 3.7b, which is the desired result), and the results slightly differ when using different $k$ values. However, ridge regression cannot make substantial improvement on the results of the Case IV method (Figure 3.8d and Figure 3.8e), since the Case IV method includes a nonlinear optimization procedure. Further experiments on other settings and parameters reveal that the problem of inconsistent TTC map is probably caused by certain noise. When a region contains very few pixels (such as a super pixel), the Signal to Noise Ratio (SNR) tends to become very low. So it may not be possible to produce satisfactory TTC map results with super pixels that are very small.
(a) Case II method, without ridge regression
(b) Case II method, with ridge regression,  $k = 0.01$

(c) Case II method, with ridge regression,  $k = 0.1$
(d) Case II method, with ridge regression,  $k = 1$

(e) Case IV method, without ridge regression
(f) Case IV method, with ridge regression,  $k_1 = 0.1, k_2 = 0.01$

Figure 3.8: TTC map results (at a randomly chosen time step) of Sequence-03 using various methods and ridge regression. The TTC value of each superpixel is color-coded and indicated by the colorbar.
3.4.3 Quantitative assessment

To support the SNR argument, the following experiments are conducted: we manually extract the planar objects in all the images of two chosen sequences (Sequence-17 and Sequence-06) by labeling the four vertices of the object in each frame. Given the four vertices, any grid segmentation can be generated on top of the planer object, e.g. $2 \times 2$, $4 \times 4$, $8 \times 8$. A segmented region (e.g. upper left block of the $4 \times 4$ segmentation) throughout the sequence represents the same part of the planer object. For example, given the $4 \times 4$ segmentation, the planar object can be treated as 16 independent planar objects. Therefore, 16 TTC values can be computed independently which correspond to those 16 regions. However, all those TTC values are supposed to be close to each other, since physically the 16 regions all belong to the same object. To quantitatively analyze the estimation error of TTC, define the Mean Total Absolute Error (MTAE) as:

$$MTAE = \frac{1}{\text{regions}} \sum_{\text{regions}} \sum_{\text{frames}} | \text{TTC}_{\text{estimated}} - \text{TTC}_{\text{groundtruth}} |$$

(3.32)

The MTAE sums the absolute errors of TTC throughout the entire sequence, and then takes the mean across all the segmented regions. Thus, if there is no SNR issue, the MTAE should be independent of segmentation (e.g. MTAEs computed on the $1 \times 1$, $2 \times 2$, $4 \times 4$, $8 \times 8$ segmentations of the same sequence should have similar magnitudes). The experiments and quantitative results on Sequence-17 and Sequence-06 are illustrated as follows.
• **Sequence-17:** Case I scenario, the object has constant speed.

On this sequence, the procedure of labeling and segmenting is shown in Figure 3.9 and the example results of the 4 * 4 segmentation are illustrated in Figure 3.10 and Figure 3.11. The MTAEs of all the conducted experiments are presented in Table 3.1.

![An example image (far)](image1)
![An example image (close)](image2)

![An example 4 * 4 segmentation (far)](segmentation1)
![An example 4 * 4 segmentation (close)](segmentation2)

Figure 3.9: **Labeling and segmenting each image in Sequence-17.** (a) and (b) Label the vertices of the planar object. (c) and (d) Partition the planar object into 4 * 4 segmentation.
Figure 3.10: **TTC results of the $4 \times 4$ segmented regions in Sequence-17.** The Case II method with ridge regression $k = 0.1$ is implemented. The blue line is the estimated TTC, and the red line is the ground truth. In the experiment, time is measured by “frames”.

Figure 3.11: **TTC results of the $4 \times 4$ segmented regions in Sequence-17.** The Case IV method with ridge regression $k_1 = k_2 = 1$ is implemented. The blue line is the estimated TTC, and the red line is the ground truth. In the experiment, time is measured by “frames”.

59
<table>
<thead>
<tr>
<th>Segmentation</th>
<th>Parameter</th>
<th>Case I method</th>
<th>Case II method</th>
<th>Case IV method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 * 1</td>
<td>$k = 0$</td>
<td>$4.0855 \times 10^2$</td>
<td>$5.1907 \times 10^2$</td>
<td>$5.3407 \times 10^2$</td>
</tr>
<tr>
<td></td>
<td>$k = 1$</td>
<td></td>
<td>$5.1236 \times 10^2$</td>
<td></td>
</tr>
<tr>
<td>2 * 2</td>
<td>$k = 0$</td>
<td>$4.9228 \times 10^2$</td>
<td>$5.0615 \times 10^2$</td>
<td>$9.0930 \times 10^2$</td>
</tr>
<tr>
<td></td>
<td>$k = 0.01$</td>
<td>$9.3725 \times 10^2$</td>
<td>$1.5421 \times 10^3$</td>
<td>$1.0197 \times 10^4$</td>
</tr>
<tr>
<td></td>
<td>$k = 0.1$</td>
<td></td>
<td>$1.0497 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$k = 0.5$</td>
<td>$6.9169 \times 10^2$</td>
<td>$6.1224 \times 10^3$</td>
<td>$5.1279 \times 10^3$</td>
</tr>
<tr>
<td></td>
<td>$k = 1$</td>
<td>$6.7031 \times 10^2$</td>
<td>$3.9319 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$k = 5$</td>
<td></td>
<td>$4.1168 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>8 * 8</td>
<td>$k = 0$</td>
<td>$1.3567 \times 10^3$</td>
<td>$9.4773 \times 10^3$</td>
<td>$1.0059 \times 10^5$</td>
</tr>
<tr>
<td></td>
<td>$k = 0.01$</td>
<td></td>
<td>$2.1089 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$k = 0.1$</td>
<td>$1.1335 \times 10^3$</td>
<td>$2.5577 \times 10^4$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$k = 1$</td>
<td>$1.2496 \times 10^3$</td>
<td>$2.5680 \times 10^4$</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: **Mean Total Absolute Error (in frames) on Sequence-17.** The parameter $k$ is used in ridge regression. In both steps of the Case IV method, there is a $k_i$ for ridge regression, and we choose $k_1 = k_2 = k$. In the table, an empty spot refers to the result of the specific method and parameter is not computed (since not necessary).
• Sequence-06: Case I scenario, the object changes speed occasionally.

On this sequence, the procedure of labeling and segmenting is shown in Figure 3.12 and the example results of the 4\(\times\)4 segmentation are illustrated in Figure 3.13 and Figure 3.14. The MTAEs of all the conducted experiments are presented in Table 3.2.

Figure 3.12: Labeling and segmenting each image in Sequence-06. (a) and (b) Label the vertices of the planar object. (c) and (d) Partition the planar object into 4\(\times\)4 segmentation.
Figure 3.13: **TTC results of the 4 * 4 segmented regions in Sequence-06.** The Case II method without ridge regression is implemented. The blue line is the estimated TTC, and the red line is the ground truth. In the experiment, time is measured by “frames”.

Figure 3.14: **TTC results of the 4 * 4 segmented regions in Sequence-06.** The Case IV method with ridge regression $k_1 = k_2 = 1$ is implemented. The blue line is the estimated TTC, and the red line is the ground truth. In the experiment, time is measured by “frames”.

62
<table>
<thead>
<tr>
<th>Segmentation</th>
<th>Parameter</th>
<th>Case I method</th>
<th>Case II method</th>
<th>Case IV method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 * 1</td>
<td>k = 0</td>
<td>1.3710 * 10^3</td>
<td>1.3806 * 10^3</td>
<td>1.3023 * 10^3</td>
</tr>
<tr>
<td></td>
<td>k = 1</td>
<td></td>
<td>1.3801 * 10^3</td>
<td></td>
</tr>
<tr>
<td>2 * 2</td>
<td>k = 0</td>
<td>5.6108 * 10^3</td>
<td>1.6268 * 10^3</td>
<td>1.5303 * 10^3</td>
</tr>
<tr>
<td></td>
<td>k = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 * 4</td>
<td>k = 0</td>
<td>1.9168 * 10^4</td>
<td>2.3583 * 10^3</td>
<td>4.6742 * 10^3</td>
</tr>
<tr>
<td></td>
<td>k = 0.1</td>
<td>2.8791 * 10^3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>k = 1</td>
<td>2.9456 * 10^3</td>
<td></td>
<td>1.0675 * 10^4</td>
</tr>
<tr>
<td>8 * 8</td>
<td>k = 0</td>
<td>1.2700 * 10^4</td>
<td>4.7795 * 10^4</td>
<td>7.7241 * 10^4</td>
</tr>
<tr>
<td></td>
<td>k = 1</td>
<td>1.0355 * 10^4</td>
<td></td>
<td>5.0930 * 10^4</td>
</tr>
</tbody>
</table>

Table 3.2: **Mean Total Absolute Error (in frames) on Sequence-06.** The parameter $k$ is used in ridge regression. In both steps of the Case IV method, there is a $k_i$ for ridge regression, and we choose $k_1 = k_2 = k$. In the table, an empty spot refers to the result of the specific method and parameter is not computed (since not necessary).

From Figure 3.10, Figure 3.11, Figure 3.13 and Figure 3.14 we observe that, in some segmented regions, the estimated TTC follows the ground truth pretty well, while in some others, huge spikes emerge and totally ruin the estimation. Such a phenomenon tells us that although the 16 regions in the 4 * 4 segmentation are of the same size, the SNRs in various regions are far different. In Table 3.1 and Table 3.2, when looking at the quantitative results of any specific method (e.g. Case I, Case II, Case IV), it is obvious that the MTAEs of the 2 * 2 segmentation is much larger than the MTAEs of the 1 * 1 segmentation; the MTAEs of the 4 * 4 segmentation is much larger than the MTAEs of the 2 * 2 segmentation; and the MTAEs of the 8 * 8 segmentation is again much larger than the MTAEs of the 4 * 4 segmentation. Such ever-increasing MTAEs support the SNR argument, since if there is no such a SNR issue, the MTAEs should be independent of the segmentations and be close to each other. Therefore, it is discovered that, as the planar object is segmented from 1 * 1 to 8 * 8, the SNR of each region becomes lower and lower, thus the estimated TTC of
almost all the regions deteriorates enormously and the MTAEs get larger drastically. Furthermore, compared with the Case II method, the Case IV method is much more sensitive to segmentation.

3.4.4 Comparison with baseline

To answer the question of whether the method can be further improved, the IBD-based TTC method is compared with an optical flow-based baseline.

The optical flow implementation is provided by [120] (Liu, 2009). The algorithm generates motion fields in both the $x$ and $y$ axes, denoted as $v_x$ and $v_y$ respectively (equivalent to $u$ and $v$ defined before). TTC can be derived from these two motion fields using least squares method. According to Equation 3.7, we have:

$$v_x = \frac{1}{Z}(fU - xW) \quad (3.33)$$

$$v_y = \frac{1}{Z}(fV - yW) \quad (3.34)$$

by defining $\frac{1}{Z}fU \equiv A_1$, $\frac{1}{Z}fV \equiv A_2$ and $-\frac{1}{Z}W \equiv C$ (where $TTC = 1/C$), we can get the following equations:

$$v_x = A_1 + xC \quad (3.35)$$

$$v_y = A_2 + yC \quad (3.36)$$

In Equation 3.35 and Equation 3.36, $A_1$, $A_2$, and $C$ can be easily solved using least squares method, since $v_x$, $v_y$, $x$ and $y$ are known. So with each motion field, $v_x$ and $v_y$ respectively, a TTC value can be derived. We implement the baseline method and make the comparison on Sequence-17 and Sequence-06, the results are illustrated as follows.
- **Sequence-17**: Case I scenario, the object has constant speed.

Figure 3.15 ~ Figure 3.17 show the TTC results on Sequence-17 computed using the optical flow-based method. The corresponding MTAEs are presented in Table 3.3.

Figure 3.15: **Implementing optical flow on Sequence-17.** In (b) the optical flow is color coded, where hue indicates orientation and saturation indicates magnitude. In (c) and (d) the motion fields are color coded, the warmer the more positive the motion fields, the colder the more negative the motion fields.
Figure 3.16: **TTC results of the entire planar object in Sequence-17.** The TTC results are computed using the optical flow-based method (both from $v_x$ field (in green) and $v_y$ field (in blue) respectively) and the Case II method (in magenta). The ground truth is drawn in red. In the experiment, time is measured by “frames”.

Figure 3.17: **TTC results of the 4 × 4 segmented regions in Sequence-17.** The TTC results are computed using the optical flow-based method (both from $v_x$ field (in green) and $v_y$ field (in blue) respectively). The ground truth is drawn in red. In the experiment, time is measured by “frames”.

Table 3.3: Mean Total Absolute Error (in frames) of the IBD-based TTC method and the optical flow-based TTC method on Sequence-17.

<table>
<thead>
<tr>
<th>Segmentation</th>
<th>Case I method</th>
<th>Case II method</th>
<th>Case IV method</th>
<th>Optical flow $v_x$</th>
<th>Optical flow $v_y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 * 1</td>
<td>$4.0855 \times 10^2$</td>
<td>$5.1907 \times 10^2$</td>
<td>$5.3407 \times 10^2$</td>
<td>$9.5027 \times 10^2$</td>
<td>$8.4701 \times 10^2$</td>
</tr>
<tr>
<td>2 * 2</td>
<td>$4.9228 \times 10^2$</td>
<td>$5.0615 \times 10^2$</td>
<td>$9.0930 \times 10^2$</td>
<td>$1.1446 \times 10^3$</td>
<td>$7.7823 \times 10^2$</td>
</tr>
<tr>
<td>4 * 4</td>
<td>$9.3725 \times 10^2$</td>
<td>$1.5421 \times 10^3$</td>
<td>$1.0197 \times 10^4$</td>
<td>$2.8759 \times 10^3$</td>
<td>$1.1253 \times 10^3$</td>
</tr>
<tr>
<td>8 * 8</td>
<td>$1.3567 \times 10^3$</td>
<td>$9.4773 \times 10^3$</td>
<td>$1.0059 \times 10^5$</td>
<td>$9.5214 \times 10^3$</td>
<td>$1.0623 \times 10^4$</td>
</tr>
</tbody>
</table>

• Sequence-06: Case I scenario, the object changes speed occasionally.

Figure 3.18 ~ Figure 3.20 show the TTC results on Sequence-06 computed using the optical flow-based method. The corresponding MTAEs are presented in Table 3.4.

Figure 3.18: Implementing optical flow on Sequence-06. In (b) the optical flow is color coded, where hue indicates orientation and saturation indicates magnitude. In (c) and (d) the motion fields are color coded, the warmer the more positive the motion fields, the colder the more negative the motion fields.
Figure 3.19: **TTC results of the entire planar object in Sequence-06.** The TTC results are computed using the optical flow-based method (both from $v_x$ field (in green) and $v_y$ field (in blue) respectively) and the Case II method (in magenta). The ground truth is drawn in red. In the experiment, time is measured by “frames”.

Figure 3.20: **TTC results of the $4 \times 4$ segmented regions in Sequence-06.** The TTC results are computed using the optical flow-based method (both from $v_x$ field (in green) and $v_y$ field (in blue) respectively). The ground truth is drawn in red. In the experiment, time is measured by “frames”.

68
<table>
<thead>
<tr>
<th>Segmentation</th>
<th>Case I method</th>
<th>Case II method</th>
<th>Case IV method</th>
<th>Optical flow $v_x$</th>
<th>Optical flow $v_y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \times 1$</td>
<td>$1.3710 \times 10^3$</td>
<td>$1.3806 \times 10^3$</td>
<td>$1.3023 \times 10^3$</td>
<td>$4.3217 \times 10^3$</td>
<td>$3.5242 \times 10^3$</td>
</tr>
<tr>
<td>$2 \times 2$</td>
<td>$5.6108 \times 10^3$</td>
<td>$1.6268 \times 10^3$</td>
<td>$1.5303 \times 10^3$</td>
<td>$5.8673 \times 10^3$</td>
<td>$3.9938 \times 10^3$</td>
</tr>
<tr>
<td>$4 \times 4$</td>
<td>$1.9168 \times 10^4$</td>
<td>$2.3583 \times 10^3$</td>
<td>$4.6742 \times 10^3$</td>
<td>$1.3723 \times 10^4$</td>
<td>$5.3324 \times 10^3$</td>
</tr>
<tr>
<td>$8 \times 8$</td>
<td>$1.2700 \times 10^4$</td>
<td>$4.7795 \times 10^3$</td>
<td>$7.7241 \times 10^4$</td>
<td>$7.3067 \times 10^4$</td>
<td>$2.0539 \times 10^4$</td>
</tr>
</tbody>
</table>

Table 3.4: Mean Total Absolute Error (in frames) of the IBD-based TTC method and the optical flow-based TTC method on Sequence-06.

From Figure 3.16 Figure 3.17 Figure 3.19 and Figure 3.20 we observe that the general patterns of the TTC results generated by both methods are almost the same, while the TTC results of the entire planar object computed using the IBD-based method is slightly more accurate. This is reasonable since both methods assume that the brightness does not change over a short period of time. From the quantitative results in Table 3.3 and Table 3.4 we observe that the SNR issue also exists in the optical flow-based TTC results, since the MTAEs increase drastically from the $1 \times 1$ segmentation to the $8 \times 8$ segmentation. However, compared with the IBD-based TTC method (Case II and Case IV), the optical flow-based method is less sensitive to the SNR/segmentation. This is probably because that the optical flow algorithm uses more complex and advanced optimization techniques to compute the motion fields, which helps to refrain the noise.

In conclusion, we discover that the IBD-based TTC method has its own limitation - this method alone may not be able to produce reliable TTC map results, the reasons are:

1. The principle of the method is simple and intuitive, but it is derived for the continuous world. In digital images, the way of calculating the partial derivatives, e.g. $E_t$, $E_x$, and $E_y$, is just a discrete approximation of the continuous counterpart. Therefore, the equations derived at the beginning of this chapter may not be perfectly suitable for the discrete cases.
2. Signal to Noise Ratio (SNR) is critical. All images have noise. When the region contains a large number of pixels, the SNR is usually high enough to compute a relatively accurate TTC. However, when segmenting the image into small super pixels, in each super pixel, the noise may dominate the output.

3. The pipeline of the TTC method requires solving a least squares problem, and the way of formulating the least squares problem matters. For example, in our experiments, minimizing $\sum (AE_x + BE_y + CG + E_t)^2$ and minimizing $\sum (\frac{A}{C}E_x + \frac{B}{C}E_y + \frac{1}{C}E_t + G)^2$ result in different TTC values. When solving real problems, instead of minimizing an algebraic error, it is always more preferable to minimize a physical error that has actual meaning. Current least squares problem minimizes an algebraic error, which does not sound very reasonable.

3.5 Summary

In this chapter, the Image Brightness Derivatives-based Time to Contact method is implemented on planar objects that have different moving patterns. The results show that, when applied to image sequences that only contain a single object, the method works reasonably well. However, when we segment the image into super pixels and try to draw a TTC map, the results generated by the proposed method become inconsistent and unstable. This problem is especially severe in the Case IV method which is designed for the real world scenarios. Given inconsistent TTC map results, super pixels cannot be aggregated into objects that have different motion patterns. So for complicated applications of autonomous driving, motion-based algorithms such as [115] (Lenz et al., 2011) are more favourable.
Chapter 4

Training an Autonomous Driving AI Agent with Reinforcement Learning

4.1 Introduction

Decision making/AI is the second module in our decomposition of the autonomous driving system. This module serves as the brain of the system which determines how the vehicle reacts to the environment. Designing such a module lies in the field of artificial intelligence, and this chapter makes an exploration of this topic. This chapter introduces our experiments and findings in training an autonomous driving AI agent with the reinforcement learning approach. The AI agent designed in this chapter holds the functionality of both the decision making/AI module and the control module (Figure 1.9).

Reinforcement learning is a sub-field of artificial intelligence and machine learning. Unlike regular machine learning algorithms that learn directly from training data, the reinforcement learning trains the AI agent to pick the optimal actions through the
interaction with a virtual environment, so there is no “training set” for reinforcement learning. The AI agent explores different states in the virtual environment by taking a series of actions. Along with each state, a reward is specified by the environment (Figure 4.2), and the reward further determines the utility of the state. An optimal action is the one that leads to the state with the highest utility. The AI agent learns a mapping from a state to the corresponding optimal action through a trial-and-error process in the virtual environment. Reinforcement learning is a paradigm that fully relies on the virtual environment, since its training pipeline requires both the positive and the negative examples. For example, to learn “crashing” is bad, the AI agent needs to crash first, which is absolutely infeasible in reality.

Figure 4.1: An example of the reward function for driving. The reward values are manually designed to illustrate the idea.

4.2 Reinforcement learning for driving

4.2.1 Basics

The reinforcement learning [168, 169] is formulated with a few notations, they are:
• State $s$: a representation of the environment around the AI agent.

• Action $a$: the action can be taken by the AI agent at state $s$.

• Reward $R(s)$: the reward of state $s$.

• Utility $U(s)$ of the state $s$: cumulated future rewards, related to actions taken in the subsequent states.

• Transition probability $P(s'|s,a)$: given the AI agent is in state $s$ and taking action $a$, the probability of reaching state $s'$ in the next step.

• Policy $\pi(s)$: a table of state-action pairs, given state $s$, output an action $a$ that should be taken.

The utility of state $s$ obtained by executing policy $\pi$ starting from $s$ is given by:

$$U^\pi(s) = E\left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi \right], \ s_0 = s$$  \hspace{1cm} (4.1)

where $\gamma$ is a discount factor, and $s_0 \sim s_t$ refer to the states of time step $0 \sim t$. The optimal policy at state $s$ is given by:

$$\pi^*(s) = \arg \max_{\pi} U^\pi(s)$$  \hspace{1cm} (4.2)

Denote $U^{\pi^*}(s)$ as the (true) utility of state $s$, which is obtained by executing the optimal policy. The optimal policy chooses the actions that maximize the expected utility of the subsequent states:

$$\pi^*(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U^{\pi^*}(s')$$  \hspace{1cm} (4.3)

where $A(s)$ is the set of available actions to be taken at state $s$.  

The optimal policy is defined by the famous Bellman equation:

$$U^\pi^*(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U^\pi^*(s')$$  \hspace{1cm} (4.4)$$

The Bellman equation can be described as: the utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the AI agent chooses the optimal action. The following formula:

$$U^\pi^*(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi^*, s_0 = s \right]$$  \hspace{1cm} (4.5)$$

is the unique solution to the Bellman equation.

The standard method to solve reinforcement learning is the Q-learning algorithm, which is implemented iteratively to learn a Q-value table. The Q-value is defined as:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

$$= E_{s' \sim \epsilon} \left[ R(s) + \gamma \max_{a'} Q(s',a') | s,a \right]$$  \hspace{1cm} (4.6)$$

The relationship between the utility and the Q-value is:

$$U^\pi^*(s) = \max_a Q(s,a)$$  \hspace{1cm} (4.7)$$

And the optimal policy is given by:

$$\pi^*(s) = \arg \max_a Q(s,a)$$  \hspace{1cm} (4.8)$$

In this chapter, a modification is made to the standard Q-learning algorithm in our implementation. Inspired by [128](Mnih et al., 2013), instead of updating a Q-value table, a feed-forward neural network, called Q-net, is trained to approximate the Q-value table. The input of the network is a vector representing the state $s$ and...
a valid action \( a \), while the output is the corresponding Q-value.

\[
Q(s, a; \theta) \approx Q^*(s, a) \quad (4.9)
\]

where \( \theta \) is the parameter of the network, and \( Q^* \) is the Q-value table to be approximated.

The gradient descent algorithm is used to train the neural network. At each training iteration \( i \), the loss function is defined as:

\[
L_i(\theta_i) = E_{s,a \in A(s)} [(y_i - Q(s, a; \theta_i))^2] \quad (4.10)
\]

where:

\[
y_i = E_{s' \sim \epsilon} [R(s) + \gamma \max_{a'} Q(s', a'; \theta_i)]s, a
\]

(4.11)

\( Q(s, a; \theta_i) \) and \( Q(s', a'; \theta_i) \) are the network output (computed in the forward pass of the gradient descent) given corresponding input. Assuming the term \( y_i \) is fixed, and it is considered as a constant in the loss function, so only the derivative of the \( Q(s, a; \theta_i) \) term needs to be computed:

\[
\nabla_{\theta_i} L_i(\theta_i) = E_{s,a \in A(s); s' \sim \epsilon} \left[ \left( R(s) + \gamma \max_{a'} Q(s', a'; \theta_i) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right] \quad (4.12)
\]

Then back-propagate the gradient of the loss function on the parameter \( \theta \) as:

\[
\theta_{i+1} = \theta_i - \alpha \cdot \nabla_{\theta_i} L_i(\theta_i) \quad (4.13)
\]

where \( \alpha \) is the learning rate. Mini-batch stochastic gradient descent with batch size \( n \) is used to update the network weights.

In the above deduction, the expectation over state \( s \) and action \( a \) is required to compute the gradient. Since an explicit form for the distribution of state \( s \) is not
available, inspired by [128](Mnih et al., 2013), we maintain a large database $\mathcal{D}$ to store the historical values of state $s$, as well as any relevant information. At each training iteration, when the mini-batch stochastic gradient descent is being executed, the algorithm randomly selects $n$ samples from the historical database $\mathcal{D}$ to compose the mini-batch. We argue if the database is large enough, it is sufficient to approximate the distribution of state $s$. Moreover, the distribution of action $a$ is approximated by the $\epsilon$-greedy policy: the algorithm selects a “best” action (according to the Q-net’s output) with probability $1 - \epsilon$, and selects a random action with probability $\epsilon$, where $\epsilon$ is annealed linearly from 1 to 0.1 during training.

The schematic of the training pipeline is illustrated in Figure 4.2. It is an environment-in-loop training, in each iteration, the AI agent needs to interact with the simulator.

Figure 4.2: **Pipeline of training a Q-net for driving which takes a state vector as input.**

After the training is completed, the AI agent is deployed as shown in Figure 4.3.
4.2.2 AI agent for lane changing and velocity control

Since in reinforcement learning, the AI agent is trained by interacting with a virtual environment, a simple driving simulator is created to conduct the research. In the driving simulator, the host car can accelerate, slow down, and change lanes. The Intelligent Driver Model (IDM) car-following model [178] (Treiber et al., 2000) is implemented to approximate the behaviors of traffic cars around the host car, and we assume all the traffic cars have the same physical property for simplicity. Due to the implementation of the IDM, even if the host car slows down to a full stop, the following cars will not crash into it; they are able to slow down and stop too. Moreover, intersections with traffic lights are implemented in the simulator. Whenever a traffic light turns red, traffic cars beyond the dilemma zone will all slow down and make a full stop. Figure 4.4 shows two screenshots of the simulator. The visualization window of the driving simulator shows the top-down view of the road ahead. The host car is represented by the red solid box, while the traffic cars are represented by the yellow solid boxes. The traffic light status is displayed as a small square (either in green or in red) next to the intersection.

Figure 4.3: Testing the Q-net for autonomous driving.
The objective of the experiment is to train an AI agent that is able to drive autonomously in the simulator, avoiding collisions with traffic cars by making lane changing decision and implementing velocity control. There is no perception task in the experiment, since the state vector $s$ is directly extracted from the physics engine of the simulator. The state vector $s$ is designed to contain the following information:

1. The speed of the host car.

2. The lane that the host car is driving in.
3. The lane-wise time gap between the host car and the closest preceding car in each lane (spacing between them divided by their relative speed), illustrated in Figure 4.5a.

4. The lane-wise collision label, a set of labels indicating whether the host car is going through a collision when placed in each lane, illustrated in Figure 4.5b.

![Diagram](image)

Figure 4.5: **Illustration of the state** $s$. (b) The AI agent monitors an area of two car length for collision, if a traffic car’s rear bump is in this area, it has a collision with the host car. The host car is represented by the red solid box, while the traffic car is represented by the yellow solid box.

The red traffic light at an intersection is implemented as a fully stopped car directly in front the host car. The reward is defined as a linear function of the state vector $s$, $R(s) = w \cdot s$, where the weight vector $w$ is manually designed based on a large number of experiments.

The AI agent is implemented as a Q-net. Whenever a state vector $s$ and an valid action $a$ is given as input, the network outputs the corresponding Q-value, based on
which the optimal action for the current state is chosen. The AI agent is trained successfully. In testing, it demonstrates desired behaviors such as stopping at red traffic lights, avoiding collisions with any traffic cars by changing lanes or slowing down.

### 4.2.3 AI agent for steering control

In the previous section, the lane changing maneuver of the AI agent is implemented as the host car “jumping” instantly to an adjacent lane. Since only a small set of discrete states for various lanes is defined, there is no steering process in the simulator. In reality, the car needs to “steer” to the adjacent lanes, and this process usually takes a few seconds. To implement an AI agent for steering control, a new task - track following is defined. The track (or the center line) can be of any shape, and the AI agent is required to follow the track without deviating too much from it. Track following is important since we do the same thing in our daily driving. For example, the task of driving in a lane on a highway can be modeled as following the center line of the lane. Turning in an intersection is also following a desired turning trajectory in the driver’s mind.

The steering angle is controlled by a Q-net trained with the reinforcement learning approach. Again, there is no perception task in the experiment, and the state vector \( s \) is directly extracted from the physics engine of the simulator. The state vector \( s \) is designed to contain the following information:

1. The angle between the host car’s heading and the tangent of the track.
2. The shortest distance between the host car’s center and the track.

The reward is defined as a function of the state vector \( s \), \( R(s) = f(s) \), where \( f(\cdot) \) can be linear, square, square root, as well as other types. In our implementation, a piece-wise non-linear function \( f(\cdot) \) is chosen based on a large number of experiments.
In the testing, the speed of the host car is fixed and the AI agent controls the steering angle. Although the training is only done on a circular track (train the AI agent to follow the circle), the successfully trained AI agent can drive the host car on circuits of any shape, as is shown in Figure 4.6.

![Screenshots of the host car driven by the AI agent for steering control.](a) The training is only done on a circular track. (b) and (c) In testing, the AI agent can follow tracks of any shape. The black curve is the track to be followed, while the red curve is the trajectory of the host car.

A system which combines the lane-changing decision making, velocity control, and steering control is implemented. The controls are generated by three separated Q-nets in different frequencies, the velocity control and the steering control work at every time step (10 Hz), while the lane-changing decisions are made every 10 time steps (1 Hz).
Through the experiments the following findings are reached:

- Given an arbitrary problem, we can use detailed discrete state space to approximate the environment, even if the resulting state space contains millions of states, it is still finite. Thus the Q-learning algorithm is applicable.

- With a Q-net, which is a neural network, the input can be a real number, so the continuity of the real world is preserved.

- If the reward function is properly defined, we can use reinforcement learning to approximate any decision making process (e.g. driving behaviors). However, assigning reward to each state is a non-trivial task. We can easily fall into local minimums with non-optimal reward function, then the behavior of the AI agent will be awkward and unsatisfactory.

### 4.3 Apprenticeship learning for driving

Standard reinforcement learning requires a well defined reward function. To specify a reward function for highway driving, we need to assign a set of weights stating exactly how we would like to trade off different factors, such as maintaining safe distance from preceding vehicles, keeping away from the curb, staying far from any pedestrians, never speeding, driving in the middle lane, and so on. Despite being able to drive well, we are not confident in specifying a good reward function for the task of “driving well”. Actually, when teaching a young adult to drive, we always demonstrate driving to them, and have them learn from the demonstration, rather than telling them what the reward function is. The task of learning from an expert is called “apprenticeship learning”.

The concept of apprenticeship learning is proposed by [2](Abbeel & Ng, 2004). Here the pipeline of this algorithm is elaborated according to [2](Abbeel & Ng, 2004).
In a real problem, the reward is unknown, but \[2\] (Abbeel & Ng, 2004) assumes the reward is a linear function of feature $\phi$, where the feature is defined as $\phi : S \rightarrow [0, 1]^k$, a function of state $s$. So the reward can be written as:

$$R^*(s) = w^* \cdot \phi(s) \quad (4.14)$$

where $w^* \in R^k$ is the weight vector of the linear function.

Still take the highway driving as an example, the feature $\phi$ can be considered as an indicator function of the factors we care for driving (e.g. maintaining safe following distance, keeping away from the curb, staying far from any pedestrians, etc.), while the weight vector $w^*$ specifies our trade-off/preference over these factors.

The utility of a policy $\pi$ is defined as:

$$E_{s_0 \sim \varepsilon}[U^\pi(s_0)] = E_{s_0 \sim \varepsilon} \left[ E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi \right] \right]$$

$$= E \left[ \sum_{t=0}^{\infty} \gamma^t w \cdot \phi(s_t) | \pi \right]$$

$$= w \cdot E \left[ \sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi \right] \quad (4.15)$$

Define **feature expectation** of policy $\pi$ as:

$$\mu(\pi) = E \left[ \sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi \right] \in R^k \quad (4.16)$$

Then the definition of utility of a policy $\pi$ can be written as:

$$E_{s_0 \sim \varepsilon}[U^\pi(s_0)] = w \cdot \mu(\pi) \quad (4.17)$$

In apprenticeship learning, \[2\] (Abbeel & Ng, 2004) assumes the expert’s demonstration represents the optimal policy. Denote the feature expectation corresponding to
the optimal policy as:

\[ \mu_E = \mu(\pi_E) \]  

(4.18)

In practice, the expert’s feature expectation is sampled from the expert’s demonstration in the virtual environment:

\[ \hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \]  

(4.19)

where \( m \) represents a total of \( m \) trajectories are sampled.

The core idea of apprenticeship learning is to find a policy that is as close to the expert’s policy (which is implicitly defined by the expert’s demonstration) as possible. We mimic the expert’s performance by minimizing the difference between \( E_{s_0 \sim \epsilon}[U^{\pi_E}(s_0)] \) and \( E_{s_0 \sim \epsilon}[U^{\hat{\pi}}(s_0)] \), the utilities of both policies.

Given \( \|\mu(\hat{\pi}) - \mu_E\|_2 \leq \epsilon \), and assume \( \|w\|_2 \leq 1 \), then:

\[
|E_{s_0 \sim \epsilon}[U^{\pi_E}(s_0)] - E_{s_0 \sim \epsilon}[U^{\hat{\pi}}(s_0)]| \\
= |E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi_E] - E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\hat{\pi}]| \\
= |w^T \mu(\hat{\pi}) - w^T \mu_E| \\
\leq \|w\|_2 \|\mu(\hat{\pi}) - \mu_E\|_2 \\
\leq 1 \cdot \epsilon = \epsilon
\]  

(4.20)

Above deduction proves that we can apply the apprenticeship learning to find a good enough policy, and it also justifies that the task of searching for a good reward function can be converted to the tasks of searching for a good weight vector \( w \). The algorithm of apprenticeship learning is elaborated in Figure 4.7.
Randomly pick some policy \( \pi^{(0)} \), compute \( \mu^{(0)} = \mu(\pi^{(0)}) \)

**for** \( i = 1 : n \)

Compute \( t^{(i)} = \max_{w: \|w\|_2 \leq 1} \min_{j \in \{0\ldots(i-1)\}} w^T (\mu_E - \mu^{(j)}) \), and \( w = w^{(i)} \) that attains this maximum

**if** \( t^{(i)} \leq \epsilon \) **break**

Use Q-learning algorithm to compute the optimal policy \( \pi^{(i)} \) for the problem using rewards \( R = (w^{(i)})^T \phi \)

Compute (or estimate) \( \mu^{(i)} = \mu(\pi^{(i)}) \)

**end for**

**return** \( \{\pi^{(i)} : i = 0, \ldots, n\} \)

---

**Figure 4.7:** Apprenticeship learning algorithm.

![Diagram](image)

**Figure 4.8:** Illustration of the SVM problem.

According to [2](Abbeel & Ng, 2004), the 1st step in the for loop of the algorithm can be formulated as a SVM problem, and be rewritten as:
\[
\begin{align*}
\max_{t, w} & \quad t \\
\text{s.t.} \quad & \quad w^T \mu_E \geq w^T \mu^{(j)} + t, \quad j = 0, \ldots, (i - 1) \\
\|w\|_2 & \leq 1
\end{align*}
\] (4.21)

This means we are looking for a hyperplane represented by \( w \) that separates \( \mu_E \) and \( \mu^{(j)}, j \in \{0 \ldots (i - 1)\} \) as far as possible. The SVM problem is illustrated in Figure 4.8 and the pipeline of apprenticeship learning is shown in Figure 4.9.

![Diagram of pipeline of apprenticeship learning](image)

Figure 4.9: **Pipeline of apprenticeship learning.**

We reproduce the results of [2](Abbeel & Ng, 2004) by implementing a driving simulator similar to the one mentioned in their paper (Figure 4.10). In the simulator, there are three lanes on the road. The speed of the host car (red solid box) is fixed, and it is faster than any other traffic cars (yellow solid boxes) in the scene. The host car can drive in the three lanes or on the two shoulders (e.g. the left and the right shoulders of the road, not displayed in Figure 4.10). We demonstrate our driving behavior to the computer by manually drive in the simulator for several rounds, and let the computer record our performance. Many types of driving behavior are demonstrated, e.g. trying to avoid any collision; colliding as many cars as possible;
driving only on the right two lanes, etc. After the computer figuring out our driving preferences from the demonstrations, it is able to mimic all those behaviors.

Figure 4.10: **Simplified driving testbed for apprenticeship learning.** The host car is represented by the red solid box, while the traffic cars are represented by the yellow solid boxes.

The results of [2](Abbeel & Ng, 2004) is successfully reproduced. However, as we try to apply the apprenticeship learning to more realistic driving scenarios, some difficulties are encountered, and the problems come from the limitation of the algorithm. The apprenticeship learning proposed by [2](Abbeel & Ng, 2004) seems to be able to handle only simple scenarios and may not naturally be generalized to more complicated cases.

However, we still believe learning from an expert’s demonstration is a good direction for developing autonomous driving AI agent. In the following chapter, a more straightforward way to learn from the expert is evaluated: the network is trained by directly mimicking the expert’s action (in a supervised learning fashion) rather than by learning through reward and utility (in a reinforcement learning fashion).
Compared with reinforcement learning, such a direct mapping from input to action is much easier to implement, however, it has an obvious drawback. As the ground truth labels for the supervised learning are driving control commands only, the supervision is relatively weak, thus the algorithm may not be able to learning complex behaviors such as lane changing and overtaking (will be discussed in the next chapter). While in apprenticeship learning, since the supervision is provided through the reward function, and the reward is computed according to multiple factors, stronger supervision can be provided to the algorithm. However, since apprenticeship learning is not the major focus of this thesis, we do not explore further in this direction. We hope our work could raise attention in this area.

4.4 Deep reinforcement learning for driving

In previous sections, a feed-forward neural network Q-net is trained with the reinforcement learning approach, while the input to the Q-net is a state vector of cognition results (e.g. speed, time gap, collision label). As described in the Introduction, the cognition module, the decision making/AI module, and the control module can be jointly trained as a single deep convolutional neural network using deep reinforcement learning approach.

The deep reinforcement learning approach is proposed by [128] (Mnih et al., 2013), and the Q-net is implemented as a ConvNet which takes raw image sequence as input. The training pipeline is almost identical to the one introduced in Figure 4.2, while the only difference is now errors are back-propagated through a ConvNet rather than a traditional neural network. The training pipeline is described in Figure 4.11 and the detailed algorithm is elaborated in Figure 4.13. In the deep reinforcement learning approach proposed by [128] (Mnih et al., 2013), the state \( s_t \) is represented by the image sequence \( \{x_1, \cdots, x_t\} \), which contains the input frames of time step \( 1 \sim t \).
Figure 4.11: Pipeline of training a ConvNet-based Q-net for driving which takes an image sequence as input.

After the ConvNet is successfully trained, the AI agent is deployed as shown in Figure 4.12.

Figure 4.12: Testing the ConvNet-based Q-net for autonomous driving.
Initialize the database $\mathcal{D}$

Initialize the ConvNet-based $Q$-net with random weights

for $episode = 1 : N$

initialize sequence $s_1 = \{x_1\}$

for $t = 1 : T$

Compute reward $R(s_t)$ based on $s_t$

Select a random action $a_t$ with probability $\epsilon$

Otherwise select $a_t = \arg \max_a Q^*(s_t, a; \theta)$

Execute action $a_t$ in the virtual environment and receive image $x_{t+1}$

Set $s_{t+1} = \{s_t, x_{t+1}\}$

Store transition $(s_t, a_t, R(s_t), s_{t+1})$ in $\mathcal{D}$

Sample random mini-batch of transitions $(s_i, a_i, R(s_i), s_{i+1})$ from $\mathcal{D}$

Set $y_i = R(s_i) + \gamma \max_{a'} Q(s_{i+1}, a'; \theta)$

Perform a gradient descent step on $(y_i - Q(s_i, a_i; \theta))^2$

end for

end for

Figure 4.13: Deep reinforcement learning algorithm.

A platform is successfully established to implement the deep reinforcement learning approach. Since this is a proof-of-concept work, and it is not clear whether the idea works, we choose to start from the most simple case by using some ancient driving games of the MS-DOS operating system. The games only have very few colors and simple graphics, which can significantly reduce the complexity of the vision algorithm. We select five driving games whose display modes can be switched to first-person perspective, and Figure 4.14 shows two example screenshots of the games. All these MS-DOS driving games are executed in an open source emulator - DOSBox under Linux system. The source code of DOSBox is modified in order to allow rendered
images to be conveniently extracted from the memory space of the emulator. At the same time, simulated input (e.g. acceleration, brake, steering) can be sent into the emulator. Moreover, the emulator can be paused by an outside signal, so it is convenient to synchronize the emulator and the stand-alone deep reinforcement learning program. The implementation of deep reinforcement learning is built on Caffe [97](Jia et al., 2014), whose source code is also modified to be compatible with the DOSBox emulator. Information such as images and driving control commands are transferred between these two programs through memory sharing.

Figure 4.14: Screenshots of two selected driving games in DOSBox.

However, due to the problem of specifying the reward, training a ConvNet through deep reinforcement learning is unsuccessful. In [128](Mnih et al., 2013), when the authors applying deep reinforcement learning to playing the Atari games, the reward comes from the score generated by the game engines, so the authors do not need to design a reward function by themselves. However, in our driving games, score is not provided by the game engine. Furthermore, although the scenes in DOSBox driving games are primitive (Figure 4.14), they are still significantly more complicated than the scenes in the driving simulators proposed in Figure 4.4 and Figure 4.10. So neither manually designing a reward function nor directly applying the apprenticeship learning approach is feasible in our case. From the experiments, we learn that the
reward function is the key/bottleneck of implementing reinforcement learning for autonomous driving. An appropriate reward function possibly involves the following factors: 1) efficiency, e.g. minimum time to reach destination, 2) no accidents, 3) no violation of traffic regulations.

4.5 Summary

In this chapter, the applicability of reinforcement learning to autonomous driving is evaluated. By using a state vector (e.g. speed, lane, time gap, collision label) as the neural network’s input, and a properly defined reward function as supervision, we successfully train an AI agent to drive a car in a simple driving simulator. We further explore the possibility of applying the apprenticeship learning and deep reinforcement learning approaches to autonomous driving. However, these experiments are not successful. We discover that an appropriate reward function is crucial to reinforcement learning. Therefore, reinforcement learning will only be useful in autonomous driving if a good reward function of driving can be determined in certain convenient ways.
Chapter 5

DeepDriving: Direct Perception for Autonomous Driving

5.1 Introduction

In the past decade, significant progress has been made in autonomous driving. To date, most of these systems can be categorized into two major paradigms: mediated perception approaches and behavior reflex approaches.

Figure 5.1: Three paradigms for autonomous driving.
Mediated perception approaches \cite{Ullman1980} involve multiple sub-components for recognizing driving-relevant objects, such as lanes, traffic signs, traffic lights, cars, pedestrians, etc. \cite{Geiger2013}. The recognition results are then combined into a consistent world representation of the car’s immediate surroundings (Figure 5.1). To control the car, the decision making/AI module takes all of this information into account in computing each decision. Since only a small portion of the detected objects are indeed relevant to driving decisions, this level of total scene understanding may add unnecessary complexity to an already difficult task. Unlike other robotic tasks, which may require simultaneous control of multiple limbs and joints in order to maintain balance, driving a car only requires manipulating the direction and the speed. This final output space resides in a very low dimension, while mediated perception approaches compute a high-dimensional world representation, possibly including redundant information. Instead of detecting a bounding box of a car and then using the bounding box to estimate the distance to the car, why not simply predict the distance to the car directly? After all, the individual sub-tasks involved in mediated perception are themselves considered open research questions in computer vision. Although mediated perception encompasses the current state-of-the-art approaches for autonomous driving, most of these systems have to rely on laser range finders, GPS, radar and very accurate maps of the environment to reliably parse objects in a scene. Requiring solutions to many open challenges for general scene understanding in order to solve the simpler car-controlling problem unnecessarily increases the complexity and the cost of a system.

Behavior reflex approaches construct a direct mapping from the sensory input to a driving action. This idea dates back to the late 1980s when \cite{Pomerleau1989, Pomerleau1992} used a neural network to construct a direct mapping from an image to steering angles. To learn the model, a human drives the car along the road while the system records the images and steering angles as the training data. Although
this idea is very elegant, it can struggle to deal with traffic and complicated driving maneuvers for several reasons. First, with other cars on the road, even when the input images are similar, different human drivers may make completely different decisions, which results in an ill-posed problem that is confusing when training a regressor. For example, with a car directly ahead, one may choose to follow the car, to pass the car from the left, or to pass the car from the right. When all these scenarios exist in the training data, a machine learning model may have difficulty deciding what to do given almost the same images. Second, the decision making for behavior reflex is too low-level. The direct mapping cannot see a bigger picture of the situation. For example, from the model’s perspective, passing a car and switching back to a lane are just a sequence of very low-level decisions for turning the steering wheel slightly in one direction and then in the other direction for some period of time. This level of abstraction fails to capture what is really going on, and it increases the difficulty of the task unnecessarily. Finally, because the input to the model is the whole image, the learning algorithm must determine which parts of the image are relevant. However, the level of supervision to train a behavior reflex model, i.e. the steering angles, may be too weak to force the algorithm to learn this critical information.

We desire a representation that directly predicts the affordance for driving actions, instead of visually parsing the entire scene or blindly mapping an image to steering angles. This chapter proposes a direct perception approach \cite{gibson1979affordances} for autonomous driving - a third paradigm that falls in between mediated perception and behavior reflex. We propose to learn a mapping from an image to several basic affordance indicators of the road situation, including the angle of the car relative to the road, the distance to the lane markings, and the distance to cars in the current and adjacent lanes. With this compact but meaningful affordance representation as perception output, we demonstrate that a very simple controller can then make driving decisions at a high level and drive the car smoothly.
The model is built upon the state-of-the-art deep convolutional neural network (ConvNet) framework to automatically learn image features for estimating affordance related to autonomous driving. To build our training set, a human driver is asked to play a car racing video game TORCS for 12 hours while recording the screenshots and the corresponding labels. Together with the simple controller that we design, our model can make meaningful predictions for affordance indicators and autonomously drive a car on different tracks in the video game, under different traffic conditions and lane configurations. At the same time, it enjoys a much simpler structure than the typical mediated perception approach. Testing our system on car-mounted smartphone videos and the KITTI dataset [59](Geiger et al., 2013) demonstrates good real-world perception as well. Our direct perception approach provides a compact, task-specific affordance description for scene understanding in autonomous driving.

5.2 Learning affordance for driving perception through virtual environment

To efficiently implement and test our approach, we use the open source driving game TORCS (The Open Racing Car Simulator) [191], which is widely used for AI research. From the game engine, critical indicators for driving can be collected, e.g. speed of the host car, the host car’s relative position to the road’s center line, the distance to the preceding cars. In the training phase, a “label collecting car” is manually driven in the game to collect screenshots (first person driving view) and the corresponding ground truth values of the selected affordance indicators. This data is stored and used to train a model to estimate affordance in a supervised learning manner. In the testing phase, at each time step, the trained model takes a driving scene image from the game and estimates the affordance indicators for driving. A driving controller processes the indicators and computes the steering and acceleration/brake commands. The driving
commands are then sent back to the game to drive the host car. Ground truth labels are also collected during the testing phase to evaluate the system’s performance. In both the training and testing phase, traffic is configured by putting a number of pre-programmed AI cars on road.

5.2.1 Mapping from an image to affordance

The state-of-the-art deep learning ConvNet is used as our direct perception model to map an image to the affordance indicators. In this chapter, we focus on highway driving with multiple lanes. From an ego-centric perspective, the host car only needs to concern the traffic in its current lane and the two adjacent (left/right) lanes when making decisions. Therefore, only these three lanes need to be modeled. A single ConvNet is trained to handle three lane configurations together: a road of one lane, two lanes, or three lanes.

![Eight examples of driving scenarios from an ego-centric perspective.](image)

Figure 5.2: Eight examples of driving scenarios from an ego-centric perspective. The lanes monitored for making driving decisions are marked with light gray.
Shown in Figure 5.2 are the typical cases we are dealing with. Occasionally the car has to drive on lane markings, and in such situations only the lanes on each side of the lane marking need to be monitored, as shown in Figure 5.2g and 5.2h.

Figure 5.3: Illustration of our affordance representation. A lane changing maneuver needs to traverse the “in lane system” and the “on marking system”. (f) shows the designated overlapping area used to enable smooth transitions.

Highway driving actions can be categorized into two major types: 1) following the lane center line, and 2) changing lanes or slowing down to avoid collisions with
the preceding cars. To support these actions, our system is defined to have two sets of representations under two coordinate systems: “in lane system” and “on marking system”. To achieve two major functions, lane perception and car perception, three types of indicators are proposed to represent driving situations: heading angle, the distance to the nearby lane markings, and the distance to the preceding cars. In total, 13 affordance indicators are proposed as our driving scene representation, illustrated in Figure 5.3. A complete list of the affordance indicators is enumerated in Figure 5.4. They are the output of the ConvNet as our affordance estimation and the input of the driving controller.

The “in lane system” and “on marking system” are activated under different conditions. To have a smooth transition, we define an overlapping area, where both systems are active. The layout is shown in Figure 5.3f.

Except for angle, all the indicators may output an inactive state. The inactive state is represented by an extremely low/high value that is outside of the active range of the indicator (e.g. -9.5 for toMarking_LL ∈ [-7.5, -4.5], 7.0 for toMarking_R ∈ [3.0, 5.0], 75.0 for dist_MM ∈ [0, 60.0]). There are two cases in which an indicator will be inactive:

1. When the car is driving in either the “in lane system” or “on marking system” and the other system is deactivated, then all the indicators belonging to that system are inactive.

2. When the car is driving on boundary lanes (left most or right most lane), and there is either no left lane or no right lane, then the indicators corresponding to the non-existing adjacent lanes are inactive.

According to the indicators’ value and active/inactive state, the host car can be accurately localized on the road.
always:

1) angle $[-0.5, 0.5]$: angle between the car’s heading and the tangent of the road

“in lane system”, when driving in the lane:

2) toMarking_LL $[-7.5, -4.5]$: distance to the left lane marking of the left lane
3) toMarking_ML $[-3.5, -0.5]$: distance to the left lane marking of the current lane
4) toMarking_MR $[0.5, 3.5]$: distance to the right lane marking of the current lane
5) toMarking_RR $[4.5, 7.5]$: distance to the right lane marking of the right lane
6) dist_LL $[0, 60.0]$: distance to the preceding car in the left lane
7) dist_MM $[0, 60.0]$: distance to the preceding car in the current lane
8) dist_RR $[0, 60.0]$: distance to the preceding car in the right lane

“on marking system”, when driving on the lane marking:

9) toMarking_L $[-5.0, -3.0]$: distance to the left lane marking
10) toMarking_M $[-2.0, 2.0]$: distance to the central lane marking
11) toMarking_R $[3.0, 5.0]$: distance to the right lane marking
12) dist_L $[0, 60.0]$: distance to the preceding car in the left lane
13) dist_R $[0, 60.0]$: distance to the preceding car in the right lane

Figure 5.4: **Complete list of affordance indicators in our direct perception representation.** All the affordance indicators are measured in an ego-centric perspective, in which the center of the host car is the origin of the coordinate system. The active range of each affordance indicator is listed in the brackets (assume the lane width is 4 meters), where *angle* is in radians, the rest are in meters.

### 5.2.2 Mapping from affordance to action

A very simple strategy is implemented to drive the host car, which corresponds to a straightforward physical rule-based decision making/AI module. The strategy requires the host car to overtake the preceding cars from either the left lane or the right
lane whichever is available (Figure 5.5a and 5.5b, assume overtaking from the right is permitted by the local driving regulation). If no lane can be used for overtaking, the strategy requires the host car to slow down and follow the preceding car (Figure 5.5c). Smooth overtaking on sharp curves needs delicately designed controller parameters, so to reduce the complexity of the control module, a constraint is introduced in the strategy that only allows the host car to overtake from the inner lane of the curve (Figure 5.5d and 5.5e). In our implementation, sharp curves are indicated by the mean of the most recent $n$ historical steering angles being larger than a threshold.

![Figure 5.5: Illustration of the control strategy.](image)

(a) Overtake from left (b) Overtake from right (c) Slowdown, car-following (d) Overtaking constraint (e) Overtaking constraint

Figure 5.5: **Illustration of the control strategy.** Assume overtaking from the right is permitted by the local driving regulation. (d) and (e) explain the constraint for overtaking on sharp curves in order to reduce control complexity.

The steering control is computed using the car’s position and pose, and the goal is to minimize the gap between the car’s current position and the center line of the lane. Defining $dist_{center}$ as the distance to the center line of the lane, we have:

$$steerCmd = C_1 \times (angle - C_2 \times dist_{center})$$  \hspace{1cm} (5.1)$$

where $C_1$ is a coefficient that varies under different driving conditions, and $C_2$ is a weight to trade off the $angle$ term and the $dist_{center}$ term. When the car changes lanes, the center line switches from the current lane to the objective lane. The pseudocode describing the logic of the driving controller is listed in Figure 5.6.
while (in autonomous driving mode)

ConvNet outputs affordance indicators

check availability of both the left and right lanes

if (approaching the preceding car in the same lane)

if (left lane exists and available and lane changing allowable)

left lane changing decision made

else if (right lane exists and available and lane changing allowable)

right lane changing decision made

else

slow down decision made

end if

end if

if (normal driving)

center line = center line of current lane

else if (left/right lane changing)

center line = center line of objective lane

end if

compute steering command

compute desired speed

compute acceleration/brake command based on desired speed

end while

Figure 5.6: Controller logic. The control frequency is determined by the frequency of the while loop (e.g. 10Hz for the TORCS implementation).

At each time step, the system computes the desired speed. A controller makes the actual speed follow the desired speed by controlling the acceleration/brake. The desired speed has a baseline value (e.g. 72 km/h for the TORCS implementation),
while deviations are added to or deducted from this baseline value. For example, if the car is turning, a decrement is computed according to the most recent \( n \) historical steering angles and then deducted from the desired speed baseline. If there is a preceding car in close range and a slow down decision is made, the desired speed is also determined by the distance to the preceding car. To achieve car-following behavior in such situations, we implement the “optimal velocity car-following model” \[137\] (Newell, 1961) as:

\[
v(t) = v_{max} \left( 1 - \exp \left( -\frac{c}{v_{max}} \left( \text{dist}(t) - d \right) \right) \right)
\] (5.2)

where \( \text{dist}(t) \) is the distance to the preceding car, \( v_{max} \) is the largest allowable speed, \( c \) and \( d \) are coefficients to be calibrated. With this implementation, the host car can achieve stable and smooth car-following under a wide range of speeds and even make a full stop if necessary.

### 5.3 Implementation

Our direct perception ConvNet is built upon Caffe \[97\] (Jia et al., 2014), and the standard AlexNet architecture \[106\] (Krizhevsky et al., 2012) is used. There are five convolutional layers followed by four fully connected layers with output dimensions of 4096, 4096, 256, and 13, respectively. Euclidian loss is used as the loss function. Because the 13 affordance indicators have various ranges, they are normalized to the range of \([0.1, 0.9]\).

7 tracks and 22 traffic cars in TORCS, shown in Figure 5.7 and Figure 5.8, are selected to generate the training set. The original road surface textures in TORCS are replaced with over 30 customized asphalt textures that have various lane configurations and asphalt darkness levels. Different driving behaviors are also programmed for the traffic cars to create different traffic patterns. We manually drive a car on
each track multiple times to collect training data. While driving, the screenshots are simultaneously down-sampled to \(280 \times 210\) and stored in a database together with the ground truth labels. This data collection process can be easily automated by using an AI car. However, when driving manually, we can intentionally create extreme driving conditions (e.g. off the road, collide with other cars) to collect more effective training samples, which makes the ConvNet more powerful and significantly reduces the training time.

Figure 5.7: **Examples of the 7 tracks used for training.** Each track is customized to the configuration of one-lane, two-lane, and three-lane with multiple asphalt darkness levels. The rest of the tracks are used in the testing set.

Figure 5.8: **Examples of the 22 cars used in the training set.** The rest of the cars are used in the testing set.

In total, 484,815 images are collected for training. The training procedure is similar to training an AlexNet on ImageNet data. The differences are: the input image has a resolution of \(280 \times 210\) and is no longer a square image. Moreover, no crops or mirrored versions are used. The model is trained from scratch. The initial
learning rate is chosen to be 0.01, and each mini-batch consists of 64 images randomly selected from the training samples. The training process is terminated after 140,000 iterations.

In the testing phase, when our system drives a car in TORCS, the only information it accesses is the front facing image and the speed of the car. Right after the host car overtakes a car in its left/right lane, it cannot judge whether it is safe to move to that lane, simply because the system cannot see things behind. To solve this problem, we make an assumption that the host car is faster than the traffic. Therefore if sufficient time has passed since its overtaking (indicated by a timer), it is safe to change to that lane. The control frequency in our system for TORCS is 10Hz, which is sufficient for driving below 80 km/h. A schematic of the system is shown in Figure 5.9.

![Figure 5.9: System architecture. The ConvNet processes the TORCS image and estimates 13 indicators for driving. Based on the indicators and the current speed of the car, a controller computes the driving commands which will be sent back to TORCS to drive the host car in the game.](image)

### 5.4 TORCS evaluation

The proposed direct perception model is first evaluated on the TORCS driving game. Within the game, the ConvNet outputs are visualized and used by the controller to drive the host car. To measure the estimation accuracy of the affordance indicators,
we construct a testing set which consists of tracks and cars that are not included in
the training set.

In the aerial TORCS visualization (Figure 5.10a, right), the host car is treated as
the reference object, while the road environment and other objects move relatively to
this reference. As the host car’s vertical position is fixed, it moves horizontally (e.g.
lane changing) with a heading computed from $\text{angle}$. Traffic cars only move vertically
(e.g. being caught up and overtaken by the host car). The curvature of the road is
not visualized, so the road ahead is always represented as a straight line. Both the
estimation (empty box) and the ground truth (solid box) are displayed.

![Autonomous driving in TORCS](image1.png) ![Testing on real video](image2.png)

(a) Autonomous driving in TORCS (b) Testing on real video

Figure 5.10: **Testing the TORCS-based system.** The estimation is shown as an
empty box, while the ground truth is indicated by a solid box. For testing on real
videos, without the ground truth, only the estimation is shown.

### 5.4.1 Qualitative assessment

Our system can drive very well in TORCS without any collision. In some lane chang-
ing scenarios, the controller may slightly overshoot, but it quickly recovers to the
desired position of the objective lane’s center. As seen in the TORCS visualization,
the lane perception module is pretty accurate, and the car perception module is re-
liable up to 30 meters away. In the range of 30 meters to 60 meters, the ConvNet
output becomes noisier. In a $280 \times 210$ image, when the traffic car is over 30 meter
away, it actually appears as a very tiny spot, which makes it very challenging for the network to estimate the distance. However, because the speed of the host car does not exceed 80 km/h in our tests, empirical results show that reliable car perception within 30 meters can guarantee satisfactory control quality in the game.

Qualitative experiments demonstrate that our system can tolerate moderate error in the indicator estimation and maintain smooth driving. The car is a continuous system, and the controller is constantly correcting its position. Even with some scattered erroneous estimations, the car can still drive smoothly without any collision.

5.4.2 Comparison with baselines

To quantitatively evaluate the performance of the TORCS-based direct perception ConvNet, we compare it with three baseline methods. Our model is referred as “ConvNet full” in the following comparisons.

5.4.2.1 Behavior reflex ConvNet

The method directly maps an image to steering angle using a ConvNet. This model is trained on the driving game TORCS using two settings: 1) The training samples (over 60,000 pairs of image & ground truth steering angle) are all collected while driving on an empty track; the task is to follow the lane. 2) The training samples (over 80,000 pairs of image & ground truth steering angle) are collected while driving on a track with traffic; the task is to follow the lane, avoid collisions by switching lanes, and overtake slow preceding cars. In both settings, the speed of the host car is fixed. The ConvNet takes separated single image as input and directly outputs the steering angle corresponding to the image. Then, the steering angle is sent back to TORCS to drive the host car in the game. For 1), when testing on empty tracks that are different from the training track, the trained behavior reflex ConvNet can successfully follow the empty tracks. For 2), when testing on the same track where
the training set is collected, the trained ConvNet demonstrates some capability at avoiding collisions by turning left or right. However, the trajectory is erratic. The behavior is far different from a normal human driver and is unpredictable - the host car collides with the preceding cars frequently.

5.4.2.2 Mediated perception (lane detection)

We run the Caltech lane detector [21](Aly, 2008) on TORCS images, which detects lane markings (or curves that look similar to lane markings) on each image. Each detected lane marking is approximated by a spline and represented as the coordinates \( (x, y) \) of four anchor points of that spline. Because only two lane markings can be reliably detected for most TORCS images, we map the coordinates of spline anchor points of the top two detected lane markings to the lane-based affordance indicators. A system composed of 8 Support Vector Regression (SVR) and 6 Support Vector Classification (SVC) models is trained (by using libsvm [24](Chang & Lin, 2011)) to implement the mapping (a necessary step for mediated perception approaches). The input to each SVM and SVR is an one dimensional vector of length 16 (composed of 2 splines \( \times 4 \) anchor points \( \times 2 \) coordinates). The system layout is similar to the GIST-based system (next section) illustrated in Figure [5.11] but without car perception.

Because the Caltech lane detector is a relatively weak baseline, to make the task simpler, a special training set and a special testing set are created. Both the training set (2430 samples) and testing set (2533 samples) are collected from the same track (not among the 7 training tracks for ConvNet) without traffic, and in a finer image resolution of 640 \( \times 480 \). We discover that, even when trained and tested on the same track, the Caltech lane detector-based system still performs worse than our model. The error metric is defined as Mean Absolute Error (MAE) between the affordance estimations and the ground truth distances. A comparison of the errors of the two systems is shown in Table [5.1]
<table>
<thead>
<tr>
<th>Method</th>
<th>angle</th>
<th>to_LL</th>
<th>to_ML</th>
<th>to_MR</th>
<th>to_RR</th>
<th>to_L</th>
<th>to_M</th>
<th>to_R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech lane</td>
<td>0.048</td>
<td>1.673</td>
<td>1.179</td>
<td>1.084</td>
<td>1.220</td>
<td>1.113</td>
<td>1.060</td>
<td>0.895</td>
</tr>
<tr>
<td>ConvNet full</td>
<td>0.025</td>
<td>0.260</td>
<td>0.197</td>
<td>0.179</td>
<td>0.239</td>
<td>0.291</td>
<td>0.262</td>
<td>0.231</td>
</tr>
</tbody>
</table>

Table 5.1: **Mean Absolute Error on the testing set for the Caltech lane detector baseline.** *angle* is in radians, the rest are in meters.

While MAE is a good indicator of the average magnitude of the errors, Standard Deviation (STD) of the absolute errors provides extra information regarding the volatility of the errors. Given two systems of similar MAE, a smaller STD means that the system is making more consistent and predictable decisions. In terms of driving, a smaller STD is preferable. So we also compute the STD of absolute errors corresponding to Table 5.1 shown in Table 5.2.

<table>
<thead>
<tr>
<th>Method</th>
<th>angle</th>
<th>to_LL</th>
<th>to_ML</th>
<th>to_MR</th>
<th>to_RR</th>
<th>to_L</th>
<th>to_M</th>
<th>to_R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech lane</td>
<td>0.074</td>
<td>1.924</td>
<td>1.673</td>
<td>1.571</td>
<td>1.720</td>
<td>1.517</td>
<td>1.567</td>
<td>1.375</td>
</tr>
<tr>
<td>ConvNet full</td>
<td>0.043</td>
<td>0.718</td>
<td>0.532</td>
<td>0.401</td>
<td>0.521</td>
<td>0.694</td>
<td>0.665</td>
<td>0.517</td>
</tr>
</tbody>
</table>

Table 5.2: **Standard Deviation of absolute errors on the testing set for the Caltech lane detector baseline.** *angle* is in radians, the rest are in meters.

From Table 5.1 and Table 5.2, we observe that our ConvNet-based system largely outperforms the Caltech lane detector baseline in terms of both MAE and STD.

### 5.4.2.3 Direct perception with GIST

The hand-crafted GIST feature is compared with the deep feature learnt by the ConvNet’s convolutional layers in our model. A set of 13 SVR and 6 SVC models are trained to convert the GIST feature to the 13 affordance indicators defined in our system. The procedure is illustrated in Figure 5.11. The GIST feature partitions the image into 4 * 4 segments. Because the ground area represented by the lower 2 * 4 segments may be more relevant to driving, two different settings are tried in our experiments: 1) convert the whole GIST feature, and 2) convert the lower 2 * 4
segments of the GIST feature. These two baselines are referred to as “GIST whole” and “GIST half” respectively. The input to each SVM and SVR is the vectorized whole/half GIST feature.

Figure 5.11: **GIST baseline.** Procedure of mapping GIST feature to the 13 affordance indicators for driving using SVR and SVC.

Due to the constraints of libsvm, training with the full dataset of 484,815 samples is prohibitively expensive. We instead use a subset of the training set containing 86,564 samples for training. Samples in the sub training set are collected on two training tracks with two-lane configurations. To make a fair comparison, another ConvNet is trained on the same sub training set for 80,000 iterations (referred to as “ConvNet sub”). The testing set is collected by manually driving a car on three different testing tracks with two-lane configurations and traffic. It has 8,639 samples.

The MAE is shown in Table 5.3. The dist (car distance) errors are computed when the ground truth cars lie within [2, 50] meters ahead. Below two meters, cars in the adjacent lanes are not visually present in the image. The STD of absolute errors corresponding to Table 5.3 is shown in Table 5.4.
<table>
<thead>
<tr>
<th>Method</th>
<th>angle</th>
<th>dist_LL</th>
<th>dist_MM</th>
<th>dist_RR</th>
<th>dist_L</th>
<th>dist_M</th>
<th>dist_R</th>
<th>dist_L</th>
<th>dist_R</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIST whole</td>
<td>0.051</td>
<td>1.033</td>
<td>0.596</td>
<td>0.598</td>
<td>1.140</td>
<td>18.561</td>
<td>13.081</td>
<td>20.542</td>
<td>1.201</td>
</tr>
<tr>
<td>GIST half</td>
<td>0.055</td>
<td>1.052</td>
<td>0.547</td>
<td>0.544</td>
<td>1.238</td>
<td>17.643</td>
<td>12.749</td>
<td>22.229</td>
<td>1.156</td>
</tr>
<tr>
<td>ConvNet sub</td>
<td>0.043</td>
<td>0.253</td>
<td>0.180</td>
<td>0.193</td>
<td>0.289</td>
<td>6.168</td>
<td>8.608</td>
<td>9.839</td>
<td>0.345</td>
</tr>
<tr>
<td>ConvNet full</td>
<td>0.033</td>
<td>0.188</td>
<td>0.155</td>
<td>0.159</td>
<td>0.183</td>
<td>5.085</td>
<td>4.733</td>
<td>7.983</td>
<td>0.316</td>
</tr>
</tbody>
</table>

Table 5.3: **Mean Absolute Error on the testing set for the GIST baseline.** *angle* is in radians, the rest are in meters.

<table>
<thead>
<tr>
<th>Method</th>
<th>angle</th>
<th>dist_LL</th>
<th>dist_MM</th>
<th>dist_RR</th>
<th>dist_L</th>
<th>dist_M</th>
<th>dist_R</th>
<th>dist_L</th>
<th>dist_R</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIST whole</td>
<td>0.100</td>
<td>1.704</td>
<td>1.159</td>
<td>1.127</td>
<td>1.682</td>
<td>15.692</td>
<td>11.525</td>
<td>18.659</td>
<td>1.548</td>
</tr>
<tr>
<td>GIST half</td>
<td>0.106</td>
<td>1.701</td>
<td>1.068</td>
<td>1.033</td>
<td>1.716</td>
<td>16.492</td>
<td>12.204</td>
<td>21.730</td>
<td>1.528</td>
</tr>
<tr>
<td>ConvNet sub</td>
<td>0.105</td>
<td>0.592</td>
<td>0.486</td>
<td>0.561</td>
<td>0.705</td>
<td>10.678</td>
<td>11.965</td>
<td>14.438</td>
<td>0.711</td>
</tr>
<tr>
<td>ConvNet full</td>
<td>0.086</td>
<td>0.544</td>
<td>0.415</td>
<td>0.444</td>
<td>0.528</td>
<td>9.105</td>
<td>7.816</td>
<td>12.577</td>
<td>0.704</td>
</tr>
</tbody>
</table>

Table 5.4: **Standard Deviation of absolute errors on the testing set for the GIST baseline.** *angle* is in radians, the rest are in meters.

![Figure 5.12](image-url)

**Figure 5.12: Comparison of car perception accuracy between different models.** Legend: green, ConvNet full; dark blue, ConvNet sub; light blue, GIST half; red, GIST whole.
The $dist$ (car distance) errors in Table 5.3 and Table 5.4 are computed when the ground truth cars lie within $[2, 50]$ meters ahead. Generally, the cars that are close to the camera can be more accurately recognized than the cars that are far away. Therefore, when computing the $dist$ errors for a closer range, the values are very likely to decrease. When computing the $dist$ errors for a further range, the values are very likely to increase. To assess how the car perception accuracy changes with distance, the $dist$ errors for six different ranges are computed and shown in Figure 5.12. The chosen ranges are $[2,10]$, $[2,20]$, $[2,30]$, $[2,40]$, $[2,50]$, and $[2,60]$ meters.

Results in Table 5.3, Table 5.4, and Figure 5.12 show that the ConvNet-based system works considerably better than the GIST-based system. By comparing “ConvNet sub” and “ConvNet full”, it is clear that more training data is very helpful for increasing the accuracy of the ConvNet-based direct perception system.

5.5 Testing on real-world image data

5.5.1 Smartphone video

Our TORCS-based direct perception ConvNet is tested on real driving videos taken by a smartphone camera. Although trained and tested in two different domains, the virtual environment and the real world respectively, our system still demonstrates reasonably good performance. The lane perception module works particularly well. The algorithm is able to determine the correct lane configuration, localize the car in the correct lane, and recognize lane changing transitions. The car perception module is slightly noisier, probably because the computer graphics model of cars in TORCS look quite different from the real ones. A screenshot of the system running on real video is shown in Figure 5.10b. Since we do not have ground truth measurements, only the estimations are visualized.
The direct perception ConvNet is further tested on smartphone videos that are taken in several special driving scenarios. They are:

1) **Night driving**  (Figure 5.13a): surprisingly, the model still works in this scenario. Due to low illumination and bumping, the strong blur caused by camera vibration places enormous challenge to the recognition algorithm. However, our model works acceptably well, given it is only trained with synthetic data and no night driving image is included in the training set. The car perception is noisy since the diffraction of back lights make the view of cars in the video totally different from the graphics models in the training set. The lane perception is still accurate. A few lane changing maneuvers were performed in the video and the ConvNet can recognize all of them. This is also a proof that the ConvNet develops special features for recognizing lane markings. The result of this test enlightens us that night driving may actually be even easier than day driving, since things that are distractive to the vision algorithms
(e.g. trees, buildings, etc.) are blacked out at night, leaving only the vehicles, lane markings, and traffic signs.

2) Bi-directional single-lane road  (Figure 5.13b): due to the single-lane configuration, no lane changing were performed in the video. The road contains another lane for the opposing traffic, and the two lanes are separated by double solid yellow lines (no such a scenario in the training set). While driving in the lane, the ConvNet can still recognize the double solid yellow lines as the left boundary of the road and choose the correct one-lane configuration for visualization. However, while driving on the double solid yellow lines, the algorithm has difficulty in finding the appropriate configuration.

3) Multiple-lane road  (Figure 5.13c and 5.13d): according to the design of our system, the ConvNet only monitors three lanes, and we argue that any multiple-lane configurations can be handled by our model. When testing on this four-lane road, the algorithm correctly chooses the appropriate three-lane model to approximate the scene.

5.5.2 Car distance estimation on the KITTI dataset

To quantitatively analyze how the direct perception approach works on real images, a different ConvNet is trained on the KITTI dataset [59](Geiger et al., 2013). The task is estimating the distance to other cars ahead.

The KITTI dataset contains over 40,000 stereo image pairs taken by a car driving through European urban areas. Each stereo pair is accompanied by a Velodyne LiDAR 3D point cloud file. Around 12,000 stereo pairs come with official 3D labels for the positions of nearby cars, so the distance to other cars in the image can be easily extracted. The settings for the KITTI-based ConvNet are altered from the previous TORCS-based ConvNet. In most KITTI images, there is no lane marking
at all, so the cars cannot be localized by the lane in which they are driving. For each image, a 2D coordinate system is defined on the zero height plane: the origin is the center of the host car, the $y$ axis is along the host car’s heading, while the $x$ axis is pointing to the right of the host car (Figure 5.14a). The ConvNet is designed to estimate the coordinate $(x, y)$ of the cars “ahead” of the host car in this system.

There can be many cars in a typical KITTI image, but only those closest to the host car are critical for driving decisions. So it is not necessary to detect all the cars. The space in front of the host car is partitioned into three areas according to $x$ coordinate: 1) central area, $x \in [-1.6, 1.6]$ meters, where cars are directly in front of the host car. 2) left area, $x \in [-12, 1.6]$ meters, where cars are to the left of the host car. 3) right area, $x \in (1.6, 12]$ meters, where cars are to the right of the host car. We are not concerned with cars outside this range. The ConvNet is trained to estimate the coordinate $(x, y)$ of the closest car in each area (Figure 5.14a). Thus, this ConvNet has 6 outputs.

Due to the low resolution of input images, cars far away can hardly be discovered by the ConvNet. A two-ConvNet structure is adopted. The close range ConvNet covers 2 to 25 meters (in the $y$ coordinate) ahead, and its input is the entire KITTI image resized to $497 \times 150$ resolution. The far range ConvNet covers 15 to 55 meters ahead, and its input is a cropped KITTI image covering the central $497 \times 150$ area. The final distance estimation is a combination of the two ConvNets’ outputs. We build our training samples mostly from the KITTI officially labeled images, with some additional samples we labeled ourselves. The final number is around 14,000 stereo pairs. This is still insufficient to successfully train a ConvNet. The dataset is augmented by using both the left camera and right camera images, mirroring all the images, and adding some negative samples that do not contain any car. Our final training set contains 61,894 images. Both ConvNets are trained on this set for 50,000
iterations. We label another 2,200 images as our testing set, on which the numerical estimation error is computed.

Figure 5.14: **Car distance estimation on the KITTI dataset.** (a) The coordinate system is defined relative to the host car. the space is partitioned into three areas, and the objective is to estimate the coordinate of the closest car in each area. (b) Our direct perception approach is compared with the DPM-based mediated perception. The central crop of the KITTI image (indicated by the yellow box in the upper left image and shown in the lower left image) is sent to the far range ConvNet. The bounding boxes output by DPM are shown in red, as are its distance projections in the LiDAR visualization (right). The ConvNet outputs and the ground truth are represented by green and black boxes, respectively.

### 5.5.3 Comparison with DPM-based baseline

The performance of our KITTI-based ConvNet is compared with the state-of-the-art DPM car detector (a mediated perception approach). The DPM car detector is provided by [58](Geiger et al., 2013) and is optimized for the KITTI dataset. We run the detector on the full resolution images and convert the bounding boxes to distance measurements by projecting the central point of the lower edge to the ground plane (zero height) using the calibrated camera model. The projection is very accurate given that the ground plane is flat, which holds for most KITTI images. DPM can detect multiple cars in the image, and the closest ones (one on the host car’s left, one
on its right, and one directly in front of it) are selected to compute the estimation error. Since the images are taken while the host car is driving, many images contain closest cars that only partially appear in the left lower corner or right lower corner. DPM cannot detect these partial cars, while the ConvNet can better handle such situations. To make the comparison fair, we only count errors when the closest cars fully appear in the image. The errors are computed when the traffic cars show up within 50 meters ahead (in the $y$ coordinate). When there is no car present, the ground truth is set as 50 meters. Thus, if either model has a false positive, it will be penalized. A screenshot of the system is shown in Figure 5.14b. The MAE for the $y$ and $x$ coordinate, and the Euclidian distance $d$ between the estimation and the ground truth of the car position are shown in Table 5.5.

<table>
<thead>
<tr>
<th>Method</th>
<th>$y$</th>
<th>$x$</th>
<th>$d$</th>
<th>$y$ w/o FP</th>
<th>$x$ w/o FP</th>
<th>$d$ w/o FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet</td>
<td>5.832</td>
<td>1.565</td>
<td>6.299</td>
<td>4.332</td>
<td>1.097</td>
<td>4.669</td>
</tr>
<tr>
<td>DPM + Projection</td>
<td>5.824</td>
<td>1.502</td>
<td>6.271</td>
<td>5.000</td>
<td>1.214</td>
<td>5.331</td>
</tr>
</tbody>
</table>

Table 5.5: **Mean Absolute Error (in meters) on the KITTI testing set.** Errors are computed by both penalizing (column 1~3) and not penalizing false positives (column 4~6).

The STD of the absolute errors (in both $x$ and $y$ coordinate) and the STD of the Euclidian distance $d$ between the estimated car position and the ground truth car position on the KITTI testing set are shown in Table 5.6.

<table>
<thead>
<tr>
<th>Method</th>
<th>$y$</th>
<th>$x$</th>
<th>$d$</th>
<th>$y$ w/o FP</th>
<th>$x$ w/o FP</th>
<th>$d$ w/o FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM + Projection</td>
<td>9.542</td>
<td>2.927</td>
<td>9.910</td>
<td>8.344</td>
<td>2.542</td>
<td>8.664</td>
</tr>
</tbody>
</table>

Table 5.6: **Standard Deviation of absolute errors (in meters) on the KITTI testing set.** Errors are computed by both penalizing (column 1~3) and not penalizing false positives (column 4~6).
To demonstrate how the estimation accuracy changes with distance on the KITTI testing set, the errors for four different ranges are computed and shown in Figure 5.15. The chosen ranges are [2,20], [2,30], [2,40], and [2,50] meters (in the $y$ coordinate).

![Graphs showing MAE and STD for different ranges](image)

Figure 5.15: *Comparison of car distance estimation accuracy between different models.* Legend: green, ConvNet penalizing false positives; red, DPM + Projection penalizing false positives; dark blue, ConvNet not penalizing false positives; light blue, DPM + Projection not penalizing false positives.

From Table 5.5, we observe that our direct perception ConvNet has similar performance to the state-of-the-art mediated perception baseline. Due to the cluttered driving scene of the KITTI dataset, and the limited number of training samples, our ConvNet has slightly more false positives than the DPM baseline on some testing samples. If we do not penalize false positives, the ConvNet has much lower error than the DPM-based baseline, which means its direct distance estimations of true cars are more accurate than the DPM-based approach. From our experience, the false positive problem can be reduced by simply including more training samples. Table 5.6 tells us that our ConvNet model’s error has smaller standard deviations than the DPM-based
baseline when both penalizing and not penalizing the false positives. Note that the DPM baseline requires a flat ground plane for projection. If the host car is driving on some uneven road (e.g. hills), the projection will introduce a considerable amount of error. We also try building SVR regression models to map the DPM bounding box output to the distance measurements. But the regressors turn out to be far less accurate than the projection. From the comparisons, we can see that, while working on real-world images, our direct perception approach works as well as the state-of-the-art mediated perception approach and has a simpler structure.

5.6 Visualization

Figure 5.16: Activation patterns of neurons. The neurons’ activation patterns display strong correlations with the host car’s heading, the location of lane markings, and traffic cars.
To understand how the ConvNet neurons respond to the input images, the activation patterns are visualized. On an image dataset of 21,100 samples, for each of the 4,096 neurons in the first fully connected layer, we pick the top 100 images from the dataset that activate the neuron the most and average them to get an activation pattern for this neuron. In this way, we gain an idea of what this neuron learnt from training.

Figure 5.16 shows several randomly selected averaged images. We observe that the neurons' activation patterns have strong correlation with the host car’s heading, the location of the lane markings and the traffic cars. Thus we believe the ConvNet has developed task-specific features for driving.

Figure 5.17: Response map of our KITTI-based ConvNet. The level of the response is color coded, the warmer the color, the stronger the response. The ConvNet has strong responses over the locations of nearby cars.

Figure 5.18: Response map of our TORCS-based ConvNet. The level of the response is color coded, the warmer the color, the stronger the response. The ConvNet has strong responses over the locations of lane markings.
For a particular convolutional layer of the ConvNet, a response map can be generated by displaying the highest value among all the filter responses at each pixel. Because location information of objects in the original input image is preserved in the response map, we can learn where the salient regions of the image are for the ConvNet when making estimations for the affordance indicators. The response maps of the 4th convolutional layer of the close range ConvNet on a sample of KITTI testing images are shown in Figure 5.17. We observe that the ConvNet has strong responses over the locations of nearby cars, which indicates that it learns to “look” at these cars when estimating the distances. Some response maps of our TORCS-based ConvNet are shown in Figure 5.18. This ConvNet has very strong responses over the locations of lane markings.

5.7 Summary

In this chapter, we propose a novel approach for autonomous driving based on direct perception. Our representation leverages a deep ConvNet architecture to estimate the affordance for driving actions instead of parsing entire scenes (mediated perception approaches), or blindly mapping an image directly to driving commands (behavior reflex approaches). A large-scale synthetic dataset collected from a virtual environment is used to train the ConvNet. Experiments show that our approach can perform well in both the virtual environment and the real world.
Chapter 6

Extending DeepDriving with Temporal Information

6.1 Introduction

In the DeepDriving approach introduced in the previous chapter, the direct perception ConvNet takes separated frame as input at each time step, which means there is no temporal relation between its consecutive inputs or outputs. Although the system works well with such settings, ignoring the temporal information may reduce the robustness of the system. In this chapter, the direct perception ConvNet in DeepDriving is extended by adding a memory module - the Long Short Term Memory (LSTM) unit. Our experiments show that the newly proposed system has better performance than the original system which does not incorporate temporal information in the ConvNet.

6.1.1 The role of temporal information

As we know, driving fundamentally involves temporal information. Our driving maneuver at every moment is based on a series of previous observations and decisions.
For instance, a lane changing decision could be made due to a series of consistent observations - the host vehicle is approaching the preceding vehicle, and the adjacent lane is not occupied (thus the preceding vehicle can be overtaken). When executing the lane changing decision, the driver needs to consistently turn and hold the steering wheel to the left or right until the objective lane is reached. This process takes a few seconds, as the vehicle can not “jump” instantly to the objective lane. Above all, we are living in a continuous world, and our brain is evolved to process temporal information at all the time. Therefore, it is natural to design a computer system that is also able to take temporal information into account when making driving decisions.

In the previous chapter, the DeepDriving approach is introduced which estimates a set of affordance indicators purely based on separated frame at each time step. This means, the affordance indicators estimated on consecutive frames are independent. When designing the DeepDriving system, the affordance indicators are selected to be static and independent of motion. Thus, it is possible to derive all of them from a single image without using any temporal information. For instance, the distance to preceding cars can be estimated from the cars’ size in the image, while the distance to the lane markings can be derived from the perspective of the view and the location of the lane markings in the image. Such a design of the system is absolutely fine if the environment is perfect and no noise exists. However, since there is always unknown noise in images, omitting the smoothing effect of temporal information could limit the potential capacity of the system. In DeepDriving, as the affordance indicators are estimated separately, the network may produce non-smooth estimations of a same indicator across a sequence of consecutive frames. For example, assume the estimation of the distance to the preceding car at time step \( t - 1 \) is 40 meters, the estimation of the same indicator at time step \( t \) may suddenly become 25 meters in some occasions. This is impossible in reality since a car cannot move 15 meters within a 0.1 second interval, which is equivalent to 540 km/h or 335 mph (the fastest car in the virtual
environment drives at 80 km/h). When including observations of the past into the estimation pipeline, the network may be able to reduce such effects of noise and produce much smoother estimations. Thus, even though all the affordance indicators can be estimated from a single image, incorporating temporal information into the ConvNet may still improve the accuracy of the estimation.

However, although the direct perception ConvNet in the DeepDriving approach only uses separated frames, the system is not totally disregarding temporal information. The driving controller which processes those affordance indicators actually has strong temporal information involved. At each time step when the controller computes the steering angle and accelerate/brake level, it needs to take the previous decisions and the historical statues of the system into account. As temporal information is crucial to driving, the DeepDriving approach can work because of the temporal information involved in the controller. In this chapter, the objective is to extend the DeepDriving system by further incorporating temporal information into the affordance indicator estimation process.

There are two basic ways to incorporate temporal information into the direct perception ConvNet:

1. Generate motion feature by processing multiple consecutive frames in one forward pass. In this case, the temporal information incorporated is short term motion information (e.g. at the scale of 0.1 second), since the ConvNet can only process a few consecutive frames at one time due to the limitation of computational power.

2. Explicitly insert a “memory” structure into the ConvNet, which could memorize historical status of the network. In this case, the temporal information incorporated can be much longer (e.g. in the scale of a few seconds).
Both ways are complementary to each other. However, since long term dependency often involves in driving (e.g. lane changing, overtaking), we assume the “memory” might be slightly more important than the motion feature.

### 6.2 Short term motion information

Before making any big modification to the DeepDriving ConvNet structure, a straightforward way that might be possible to incorporate short term motion information is tested.

We concatenate multiple consecutive frames into the channel dimension (thus the channel dimension is treated as the time dimension in this case), and use the concatenated images as input to the network (Figure 6.11b). For example, two consecutive frames (the previous frame and the current frame) can be concatenated as a 6-channel input. The first 3 channels of the 6-channel input come from the previous frame (RGB), while the remaining 3 channels come from the current frame (RGB). In this way, modification is only made to the first convolutional layer, while the rest of the network structure remains unchanged. Although we hope the ConvNet could learn certain motion feature automatically, experiments show negative results - simply stacking multiple frames as input to a 2D convolutional neural network only produces worse results (e.g. much larger Mean Absolute Error) than DeepDriving.

As the ConvNet cannot learn motion features automatically, a further attempt is made by using the RGB images together with the corresponding optical flow results as input. The motion fields generated by the optical flow algorithm are explicitly provided to the network, and we hope the ConvNet could learn certain high-level motion features on top of them. Two different ways to encode the optical flow results are used:
1. Use the motion fields $v_x$ and $v_y$ as two additional input channels, shown in Figure 6.1.

2. Color code the motion fields $v_x$ and $v_y$ as a RGB image, and use that image as three additional input channels, shown in Figure 6.2.

![Figure 6.1: Concatenating the RGB image and the two raw motion fields as a 5-channel input. The first 3 channels of the 5-channel input come from the current frame (RGB), while the remaining 2 channels are the motion fields $v_x$ and $v_y$ (grayscale).](image1)

![Figure 6.2: Concatenating the RGB image and the color coded optical flow image as a 6-channel input. The first 3 channels of the 6-channel input come from the current frame (RGB), while the remaining 3 channels come from the color coded optical flow results (RGB).](image2)

Both settings are tested, but none of them produce satisfactory result. Instead of providing extra motion information, the additional optical flow-based input channels act as strong source of noise to the ConvNet. So the estimation accuracy is still worse than DeepDriving. See section 6.8 for more discussions.
6.3 Long Short Term Memory (LSTM)

To incorporate long term dependency, the network needs to have an actual “memory” module, such as recurrent structures, to store the historical information. Recurrent Neural Networks (RNN) address the temporal issue by allowing information to persist. RNNs are designed to connect previous information to the current task, so they may be able to generate estimation using both current and previous observations. However, the effectiveness of regular RNNs highly depends on the tasks, usually they cannot handle long term dependency very well.

To resolve the problem that regular RNN is facing, the Long Short Term Memory (LSTM) network is proposed by [82] (Hochreiter & Schmidhuber, 1997). LSTM is a special kind of RNN, and it is capable of learning long term dependency. There are many modified version of LSTM, but in this chapter, only the standard one is implemented. A decomposition of the LSTM unit is shown in Figure 6.3 where the LSTM is unfolded through time. Inside of a LSTM unit, there are four neural network layers that interact in a special way.

![Decomposition of LSTM unit](image)

Figure 6.3: Decomposition of LSTM unit.

The LSTM has a inner information flow, which is called the cell state $C_t$. Information can persist in the cell state through consecutive time steps, so this is where the memory or the temporal information is carried. At each time step, the LSTM
has mechanisms to forget part of its memory, add new information, and output its cell state. All of these are regulated by gate structures.

When $C_{t-1}$, the cell state of time step $t-1$, flows into the LSTM unit of time step $t$, the first task is to decide which part of the information should be thrown away. This is achieved by the “forget gate layer” (Equation 6.1). The input to the forget gate layer is a concatenation of $h_{t-1}$ and $X_t$, the LSTM’s output of time step $t-1$ and its input of time step $t$ respectively. This layer is a standard neural network layer, whose output $f_t$ is a vector of values in the range of $(0, 1)$. $f_t$ has the same dimension as $C_{t-1}$, and it is point-wisely multiplied with $C_{t-1}$. In this way, some parts of the cell state are removed.

$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f) \quad (6.1)$$

The next step is to add new information into the cell state, and this is achieved by the “input gate layer” (Equation 6.2). The input gate layer works exactly in the same way as the forget gate layer, and its output $i_t$ is point-wisely multiplied with $\hat{C}_t$, a candidate cell state input vector, which is also generated from the concatenation of $h_{t-1}$ and $X_t$ (Equation 6.3).

$$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i) \quad (6.2)$$

$$\hat{C}_t = \tanh(W_C \cdot [h_{t-1}, X_t] + b_C) \quad (6.3)$$

Then, $C_t$, the new cell state of time step $t$, is produced by adding the filtered old cell state and the new information, as:

$$C_t = f_t \ast C_{t-1} + i_t \ast \hat{C}_t \quad (6.4)$$

Finally, the last step is to generate the filtered output based on the updated cell state. The “output gate layer” (Equation 6.5) works in the same way as the forget gate
layer and the input gate layer. The cell state $C_t$ first goes through a $\tanh$ function, thus the output values are pushed to the range of $(-1, 1)$. Then, the resulting vector is point-wisely multiplied with the output $o_t$ of the output gate layer to generate the final output vector $h_t$ (Equation 6.6). In this way, we can decide which part of the cell state to output.

$$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o) \quad (6.5)$$

$$h_t = o_t * \tanh(C_t) \quad (6.6)$$

The weights of different layers in the LSTM unit (e.g. $W_f$, $b_f$, $W_i$, $b_i$, $W_o$, $b_o$, $W_C$, and $b_C$) are all developed though training. Thus, the network automatically learns to handle temporal information.

### 6.4 Network architecture

To take advantage of the LSTM unit, a proper way to insert it into the original DeepDriving ConvNet is required. In most applications of deep learning, the 4096D feature vector produced by the first fully connected layer (e.g. $fc6$ of AlexNet), is recognized as the deep feature vector, which encodes the information of the input image. In our implementation, the deep feature vector is chosen as the input $X_t$ to the LSTM unit. The output vector $h_t$ of the LSTM unit is further processed by a fully connected layer to produce the final affordance estimation. Figure 6.4 shows a straightforward implementation of the LSTM-based ConvNet, which is unfolded through time. In this network, only one layer of LSTM unit is used. At each time step, new observation of the input image is encoded as a deep feature vector and added to the LSTM’s cell state information flow. In this way, the final affordance estimation is built upon both the current and historical observations.
Figure 6.4: **Architecture of the network with one-layer LSTM.** This model is denoted by “1-L”.

Figure 6.5: **Architecture of the network with two-layer LSTM.** This model is denoted by “2-L”.

130
To check whether adding more layers of LSTM units could improve the performance, the architecture illustrated in Figure 6.5 is proposed. This network has two layers of LSTM units, while the output vector of the first LSTM layer serves as the input vector to the second LSTM layer. The structure is a simple cascade of multiple LSTM layers.

The network architecture is further tweaked by using the estimated affordance indicator of $t - 1$ as the third source of input to the LSTM unit of $t$. We hope this kind of direct feedback of the affordance estimation could further reduce the system noise. Similarly, such a feedback structure can be applied to both the one-layer LSTM network and the two-layer LSTM network, which are shown in Figure 6.6 and Figure 6.7, respectively.

Figure 6.6: Architecture of the network with one-layer LSTM and feedback. This model is denoted by “1-L FB”.

131
Figure 6.7: Architecture of the network with two-layer LSTM and feedback. This model is denoted by “2-L FB”.

Figure 6.8: Architecture of the network with two-layer LSTM, feedback, and skip structure. This model is denoted by “2-L Skip FB”.

132
Inspired by [39](Donahue et al., 2015), a skip-layer structure for the feedback network is proposed. The network contains two layers of LSTM units, the feedback of the estimated affordance indicators serves as the input to the first LSTM layer. However, instead of sending the deep feature vector to the first LSTM layer as illustrated in Figure 6.7, the deep feature vector skips the first layer and goes directly to the second LSTM layer. The architecture is shown in Figure 6.8.

The introduction of the affordance estimation feedback can be considered as an additional recurrent input to the network, which makes the training even more complicated. However, since a strong recurrent information flow already exists in the LSTM unit, the feedback is treated as a static input in our experiments, which means back-propagation through time algorithm is not applied to it during training.

To train the feedback network, there are two choices for generating the feedback input vector:

1. At each training step, the ground truth of affordance indicators of the previous step is used as the LSTM input.

2. At each training step, the actually network estimation output of the previous step is used as the LSTM input.

In order to search for the optimal structure for the new system, intensive experiments on the proposed architectures and settings are conducted. The details of the implementation is introduced as follows.

### 6.5 Implementation

The system is implemented based on a modified Caffe release [39](Donahue et al., 2015). The Caffe has a built-in LSTM layer, and the rest layers are identical to the Caffe framework used by DeepDriving.
The procedure of training the LSTM-based models is different from that of training the DeepDriving model. When training the DeepDriving model, because the affordance indicators are designed to be independent of temporal information, the algorithm shuffles the order of the input frames for each mini-batch by randomly sampling images from the entire training set. However, such an approach is not applicable to the LSTM-based model, since we need to keep the temporal order of the input image sequence. Defined by the authors of the Caffe release and adopted by our implementation, the input mini-batch of the LSTM layer has three dimensions: the first is the dimension of time, the second is the dimension of multiple independent sequences, and the third is the dimension of the deep feature vector (e.g. of length 4096). For example, assume in each mini-batch (of size \( N \times t \)), there are \( N \) independent sequences, and each sequence consists of \( t \) time steps, the mini-batch is ordered as \( \{ X_1^1, X_1^2, \cdots, X_1^N, X_2^1, X_2^2, \cdots, X_2^N, \cdots, X_t^1, X_t^2, \cdots, X_t^N \} \), where \( X_i^k = \{ x_i^k(1), x_i^k(2), x_i^k(3), x_i^k(4096) \} \) representing the deep feature vector of sequence \( k \) at time step \( i \). The data layer arranges the mini-batch of the input images in a same manner, specifically \( \{ I_1^1, I_1^2, \cdots, I_1^N, I_2^1, I_2^2, \cdots, I_2^N, \cdots, I_t^1, I_t^2, \cdots, I_t^N \} \), where \( I_i^k \) represents the input image of sequence \( k \) at time step \( i \).

Although the \( N \) independent sequences of a mini-batch only contain \( t \) time steps each, the temporal dependency of each sequence can go far beyond \( t \). A parameter is set up to determine the maximum duration that the temporal dependency could last. In our experiments, the maximum duration is chosen to be 20 seconds, which is equivalent to 200 frames as our system runs at 10Hz. During training, the algorithm is designed to terminate an input sequence when it reaches the maximum duration and initialize a new one. Correspondingly, the LSTM layer has an additional input which indicates the starting frame of each sequence. Whenever a new sequence starts, the cell state of the LSTM unit is cleared, so the memory regarding the previous sequence can be removed.
Detailed procedure of generating each mini-batch is described as follows: at the beginning of the training process, \( N \) frames are randomly selected from the entire training set as the starting frames of the \( N \) independent sequences. To fill the first mini-batch, frames from 1 to \( t \) of each sequence are picked up; to fill the second mini-batch, frames from \( t + 1 \) to \( 2t \) of each sequence are picked up. The same thing is implemented for the following mini-batches, until any sequence reaches the pre-set maximum duration (e.g. 200 frames), or hit the physical end of a clip (the dataset consists of many clips). Then, a new sequence is initialized by randomly sampling a frame as its starting frame.

To implement the affordance estimation feedback, a few new layers are proposed to store and extract the network’s previous output. A modified sigmoid layer copies the output of the network to a memory space. Then, at the next time step, a specially designed hist layer reads out the values from the memory space and use them as the input to the LSTM unit. Since the feedback is not treated as a recurrent input to the network, the hist layer works as an ordinary data layer, which does not require back-propagation during training.

In all of our LSTM units, the following settings are used: in each mini-batch, the number of independent sequences is \( N = 16 \), and the size of each sequence is \( t = 8 \). The maximum duration for each sequence is 200 frames. The length of the output vector is 2048.

The LSTM-based ConvNets of different architectures are trained on the same dataset, which is the one used for training the DeepDriving model. Because the LSTM-based models and the DeepDriving model share the same structure of convolutional layers, in order to accelerate the training process, the pre-trained DeepDriving model is used to initialize the LSTM-based models. When training the LSTM-based models, the weights of the convolutional layers are kept unchanged, while only the LSTM layers and the fully connected layers are updated. This procedure signifi-
cantly reduces the training time. For instance, training a full network from scratch takes 140,000 iterations, while with the initialization, the training only takes 20,000 iterations.

6.6 Results and analysis

The LSTM-based models are tested in our TORCS virtual environment, and the original DeepDriving model is used as a baseline. All the models can drive very well in the game. From the top-down view visualization, one can hardly discern the difference in performance between the DeepDriving model and the LSTM-based ones, since all of them have good driving quality. So the models are further evaluated quantitatively on the testing set. The testing set is the same one that is used for the GIST baseline comparison in the previous chapter. It is collected by manually driving a car on three different testing tracks with two-lane configurations and traffic in TORCS. It has 8,639 samples. Again, Mean Absolute Error (MAE) of affordance indicator estimation is used to evaluate the accuracy. The MAE and the Standard Deviation corresponding to each model are listed in Table 6.1 and Table 6.2 respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>angle</th>
<th>toM_LL</th>
<th>toM_ML</th>
<th>toM_MR</th>
<th>toM_RR</th>
<th>dist_LL</th>
<th>dist_MM</th>
<th>dist_RR</th>
<th>toM_L</th>
<th>toM_M</th>
<th>toM_R</th>
<th>dist_L</th>
<th>dist_R</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepDriving</td>
<td>0.033</td>
<td>0.188</td>
<td>0.155</td>
<td>0.159</td>
<td>0.183</td>
<td>5.085</td>
<td>4.738</td>
<td>7.983</td>
<td>0.316</td>
<td>0.308</td>
<td>0.294</td>
<td>8.910</td>
<td>10.861</td>
</tr>
<tr>
<td>1-L</td>
<td>0.035</td>
<td>0.197</td>
<td>0.143</td>
<td>0.135</td>
<td>0.184</td>
<td>4.751</td>
<td>4.570</td>
<td>7.811</td>
<td>0.249</td>
<td>0.272</td>
<td>0.264</td>
<td>7.543</td>
<td>10.971</td>
</tr>
<tr>
<td>2-L</td>
<td>0.038</td>
<td>0.195</td>
<td>0.153</td>
<td>0.145</td>
<td>0.196</td>
<td>5.213</td>
<td>4.829</td>
<td>7.653</td>
<td>0.250</td>
<td>0.288</td>
<td>0.285</td>
<td>7.324</td>
<td>10.692</td>
</tr>
<tr>
<td>1-L FB (GT)</td>
<td>0.036</td>
<td>0.193</td>
<td>0.125</td>
<td>0.125</td>
<td>0.176</td>
<td>5.271</td>
<td>4.238</td>
<td>10.360</td>
<td>0.266</td>
<td>0.238</td>
<td>0.229</td>
<td>7.831</td>
<td>9.660</td>
</tr>
<tr>
<td>1-L FB (OP)</td>
<td>0.036</td>
<td>0.212</td>
<td>0.139</td>
<td>0.132</td>
<td>0.163</td>
<td>5.004</td>
<td>5.089</td>
<td>8.977</td>
<td>0.244</td>
<td>0.283</td>
<td>0.285</td>
<td>7.542</td>
<td>12.879</td>
</tr>
<tr>
<td>2-L FB (GT)</td>
<td>0.040</td>
<td>0.176</td>
<td>0.127</td>
<td>0.124</td>
<td>0.177</td>
<td>5.129</td>
<td>4.683</td>
<td>10.016</td>
<td>0.253</td>
<td>0.280</td>
<td>0.275</td>
<td>7.652</td>
<td>14.088</td>
</tr>
<tr>
<td>2-L FB (OP)</td>
<td>0.045</td>
<td>0.213</td>
<td>0.140</td>
<td>0.144</td>
<td>0.206</td>
<td>5.166</td>
<td>4.948</td>
<td>9.933</td>
<td>0.295</td>
<td>0.303</td>
<td>0.296</td>
<td>8.311</td>
<td>13.170</td>
</tr>
<tr>
<td>2-L Skip FB (GT)</td>
<td>0.035</td>
<td>0.189</td>
<td>0.121</td>
<td>0.113</td>
<td>0.143</td>
<td>4.461</td>
<td>4.289</td>
<td>6.707</td>
<td>0.237</td>
<td>0.266</td>
<td>0.269</td>
<td>7.843</td>
<td>10.886</td>
</tr>
<tr>
<td>2-L Skip FB (OP)</td>
<td>0.040</td>
<td>0.225</td>
<td>0.145</td>
<td>0.133</td>
<td>0.181</td>
<td>5.800</td>
<td>5.044</td>
<td>8.667</td>
<td>0.235</td>
<td>0.289</td>
<td>0.294</td>
<td>8.860</td>
<td>11.504</td>
</tr>
</tbody>
</table>

Table 6.1: Mean Absolute Error of different models on the testing set. angle is in radians, the rest are in meters. “GT” refers to that the ground truth values are used as feedback for training, while “OP” refers to that the network’s actual outputs are used as feedback for training.
<table>
<thead>
<tr>
<th>Model</th>
<th>angle</th>
<th>toM_LL</th>
<th>toM_ML</th>
<th>toM_MR</th>
<th>toM_RR</th>
<th>dist_LL</th>
<th>dist_MM</th>
<th>dist_RR</th>
<th>L dist</th>
<th>R dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepDriving</td>
<td>0.086</td>
<td>0.544</td>
<td>0.415</td>
<td>0.444</td>
<td>0.528</td>
<td>9.105</td>
<td>7.816</td>
<td>12.577</td>
<td>0.704</td>
<td>0.719</td>
</tr>
<tr>
<td>1-L</td>
<td>0.089</td>
<td>0.601</td>
<td>0.440</td>
<td>0.419</td>
<td>0.511</td>
<td>7.876</td>
<td>7.629</td>
<td>10.782</td>
<td>0.584</td>
<td>0.731</td>
</tr>
<tr>
<td>2-L</td>
<td>0.094</td>
<td>0.567</td>
<td>0.408</td>
<td>0.414</td>
<td>0.510</td>
<td>7.877</td>
<td>7.951</td>
<td>10.723</td>
<td>0.555</td>
<td>0.736</td>
</tr>
<tr>
<td>1-L FB (GT)</td>
<td>0.081</td>
<td>0.536</td>
<td>0.372</td>
<td>0.363</td>
<td>0.463</td>
<td>8.387</td>
<td>7.118</td>
<td>11.600</td>
<td>0.595</td>
<td>0.630</td>
</tr>
<tr>
<td>1-L FB (OP)</td>
<td>0.097</td>
<td>0.585</td>
<td>0.242</td>
<td>0.236</td>
<td>0.420</td>
<td>7.867</td>
<td>8.059</td>
<td>12.357</td>
<td>0.597</td>
<td>0.778</td>
</tr>
<tr>
<td>2-L FB (GT)</td>
<td>0.092</td>
<td>0.523</td>
<td>0.363</td>
<td>0.333</td>
<td>0.425</td>
<td>8.064</td>
<td>7.918</td>
<td>12.556</td>
<td>0.572</td>
<td>0.721</td>
</tr>
<tr>
<td>2-L FB (OP)</td>
<td>0.097</td>
<td>0.559</td>
<td>0.381</td>
<td>0.403</td>
<td>0.519</td>
<td>7.736</td>
<td>7.818</td>
<td>12.167</td>
<td>0.582</td>
<td>0.747</td>
</tr>
<tr>
<td>2-L Skip FB (GT)</td>
<td>0.097</td>
<td>0.500</td>
<td>0.354</td>
<td>0.311</td>
<td>0.376</td>
<td>6.858</td>
<td>7.115</td>
<td>9.925</td>
<td>0.573</td>
<td>0.706</td>
</tr>
<tr>
<td>2-L Skip FB (OP)</td>
<td>0.090</td>
<td>0.581</td>
<td>0.419</td>
<td>0.387</td>
<td>0.492</td>
<td>8.899</td>
<td>7.901</td>
<td>12.205</td>
<td>0.556</td>
<td>0.764</td>
</tr>
</tbody>
</table>

Table 6.2: **Standard Deviation of different models on the testing set.** The angle is in radians, the rest are in meters. “GT” refers to that the ground truth values are used as feedback for training, while “OP” refers to that the network’s actual outputs are used as feedback for training.

<table>
<thead>
<tr>
<th>Model</th>
<th>is_in_lane</th>
<th>is_LL</th>
<th>is_RR</th>
<th>is_on_marking</th>
<th>is_L</th>
<th>is_R</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepDriving</td>
<td>201</td>
<td>114</td>
<td>182</td>
<td>374</td>
<td>285</td>
<td>343</td>
</tr>
<tr>
<td>1-L</td>
<td>218</td>
<td>92</td>
<td>119</td>
<td>318</td>
<td>270</td>
<td>290</td>
</tr>
<tr>
<td>2-L</td>
<td>221</td>
<td><strong>89</strong></td>
<td>123</td>
<td><strong>303</strong></td>
<td>258</td>
<td>286</td>
</tr>
<tr>
<td>1-L FB (GT)</td>
<td>200</td>
<td>95</td>
<td><strong>113</strong></td>
<td>344</td>
<td>253</td>
<td>311</td>
</tr>
<tr>
<td>1-L FB (OP)</td>
<td>211</td>
<td>94</td>
<td>132</td>
<td>334</td>
<td>255</td>
<td>298</td>
</tr>
<tr>
<td>2-L FB (GT)</td>
<td>178</td>
<td>90</td>
<td>119</td>
<td>334</td>
<td>254</td>
<td><strong>281</strong></td>
</tr>
<tr>
<td>2-L FB (OP)</td>
<td>202</td>
<td>108</td>
<td>136</td>
<td>307</td>
<td><strong>251</strong></td>
<td>323</td>
</tr>
<tr>
<td>2-L Skip FB (GT)</td>
<td><strong>177</strong></td>
<td>91</td>
<td>142</td>
<td>318</td>
<td>264</td>
<td>302</td>
</tr>
<tr>
<td>2-L Skip FB (OP)</td>
<td>207</td>
<td><strong>89</strong></td>
<td>130</td>
<td>339</td>
<td>281</td>
<td>304</td>
</tr>
</tbody>
</table>

Table 6.3: **Number of Misclassification of different models on the testing set.** “GT” refers to that the ground truth values are used as feedback for training, while “OP” refers to that the network’s actual outputs are used as feedback for training.

As described in the previous chapter, in addition to being used as regressors (e.g. affordance indicators), the outputs of the ConvNet are also used as classifiers (e.g. active/inactive state). These classifiers help to determine which system is the host car driving in (e.g. “in lane system” or “on marking system”, see Figure 5.3 for definition), and whether the left or right adjacent lane exists. So we further evaluate the error of the network by treating the outputs as classifiers. We count the number
of samples that are mistakenly classified by the network, and show the results in Table 6.3.

In Table 6.3, “is_in_lane” refers to that the “in lane system” (the host car is driving in the lane) should be active, while the network’s output shows the negative; corresponding to the “in lane system”, “is_LL” and “is_RR” refer to that the left adjacent lane and the right adjacent lane exist, while the network’s outputs show the negative, respectively. “is_on_marking”, “is_L”, and “is_R” have similar meanings corresponding to the “on marking system” (the host car is driving on the lane marking). The fewer misclassification a model makes, the better performance the model has.

Although there is some randomness when training the LSTM-based ConvNets, we still can discover several general patterns from the results. By comparing all the quantitative results, we have the following observations:

• Generally, adding the LSTM unit improves the accuracy.

• Simply stacking multiple LSTM layers (without using feedback) does not further improve the accuracy.

• Adding affordance estimation as feedback is possible to slightly further improve the accuracy. In terms of training options, the models trained by using ground truth affordance indicator as feedback outperforms the ones trained by using actual network output as feedback.

• Overall, the architecture illustrated in Figure 6.8 has the best performance.

To better interpret Table 6.1 and Table 6.3, and visualize the performance improvement of the LSTM-based models over the DeepDriving baseline, Figure 6.9 and Figure 6.10 are plotted. All the errors of the LSTM-based models are represented as relative values compared to their counterparts of the DeepDriving model. And all the error terms of the DeepDriving model are set to 100%.
Figure 6.9: Relative errors (%) of the affordance estimation of different LSTM-based models compared with the DeepDriving baseline. The error terms of the DeepDriving model are set to 100%. If any LSTM-based model outperforms the DeepDriving baseline on any indicator, the corresponding relative error is smaller than 100%.

Figure 6.10: Relative errors (%) of the classifiers of different LSTM-based models compared with the DeepDriving baseline. The error terms of the DeepDriving model are set to 100%. If any LSTM-based model outperforms the DeepDriving baseline on any indicator, the corresponding relative error is smaller than 100%.
From Figure 6.9 and Figure 6.10, we observe that adding the LSTM unit does improve the estimation accuracy on most affordance indicators. And the effectiveness of different network architectures varies a lot.

6.7 Further discussion

Inserting the LSTM unit into the ConvNet is only one of the possible ways to incorporate temporal information, specifically the long term dependency. In this proposed pipeline, deep feature vector generated by 2D convolution on single image (Figure 6.11a) is used as the LSTM input. In such a setting, the deep feature vector only contains spatial information regarding the input frame (e.g. distance to each object), while temporal information (e.g. velocity vector of each object) is not included. At each time step, the LSTM unit extracts useful information from the static deep feature vector and mix it into the cell state. Since the cell state carries the long term historical information, and the final estimation of the affordance indicators is based on the cell state, the proposed system can take advantage of temporal information even when it still uses single image as input of each time step.

In order to learn short term motion information, in section 6.2, the concatenations of multiple frames (short image sequence) are used as input to a 2D ConvNet (Figure 6.11b). Although temporal information is implied in the image sequence, the 2D ConvNet fails to extract motion deep feature vector from the input. The failure is due to the fundamental limitation of 2D convolution. In 2D convolution, the filter only slides across the 2D image plane, while omitting the third dimension - channel, which is used as the time dimension in this case. So when convolving a 2D filter with the concatenation of the image sequence, the size of the resulting feature map in the time dimension will always be 1 (see Chapter 2 for more details on convolution). This means the time dimension collapses after the first convolutional layer. Therefore, temporal information can hardly be preserved.
(a) Applying 2D convolution on a single image (a 3D cuboid with dimensions of height, width, and channel) produces 2D feature map.

(b) Applying 2D convolution on an image sequence (concatenating multiple frames as multiple channels, so still a 3D cuboid) still produces a 2D feature map, thus temporal information (motion features) is hardly preserved.

(c) Applying 3D convolution on an image sequence (a 4D cuboid with dimensions of height, width, time, and channel) produces a 3D feature map, thus temporal information (motion features) can be preserved.

Figure 6.11: 2D convolution vs. 3D convolution.
However, long term dependency (e.g. LSTM) and short term motion information (e.g. motion deep feature vector) are complementary, and including motion features could probably further improve the performance of the LSTM-based system. So it is promising to replace the current 2D convolution with 3D convolution, which is conducted on a sequence of consecutive frames (Figure 6.11c). In 3D convolution, the 3D filter slides across the entire volume, so the time dimension is preserved in the resulting feature map. The deep feature vector generated by a 3D ConvNet is possible to contain both the spatial information and the temporal information.

In training our LSTM-based ConvNets, the ground truth of the 13 affordance indicators are used as supervision, while all the affordance indicators are static spatial measures. Ideally, to help a LSTM-based and/or 3D convolution-based system better learn temporal features through training, extra temporal affordance indicators, such as velocity vectors of critical objects, should be explicitly provided as supervision as well. Since a virtual environment is used for developing the system, collecting such kind of training set is simple and straightforward.

6.8 Summary

In this chapter, we propose to incorporate temporal information into the DeepDriving approach. By inserting the LSTM unit into the feed-forward ConvNet and tweaking the network structure, we successfully design a model that produces more accurate affordance estimation compared to the original DeepDriving model. Quantitative results show that temporal information is effective in reducing errors from the estimation of the affordance indicators, which are physically continuous. So in future research on computer vision-based autonomous driving system, temporal information should always be considered, since it provides extra cues to solve the problem.
Chapter 7

Conclusions and Future Work

This thesis focuses on the application and extension of computer vision algorithms to address the autonomous driving problem. This is a complex problem that requires real-time robustness and reliability from cognition, artificial intelligence and control. Moreover, the development, testing and verification of the corresponding algorithms require data that properly depicts the vast situations and scenarios that are faced each and every second by today’s human drivers. Contained in this thesis are verified enhancements on a Time to Contact method for motion-based object recognition, a reinforcement learning method for driving decision making, and a deep learning-based direct perception method for driving scene understanding. Also developed is an efficient and robust approach to create data for training, testing and verification. It is hoped that this thesis will inspire others to build on the contributions of this and stimulate other efforts to accelerate the achievement of safe autonomous driving.

The fundamental contribution of the thesis is the combined use of a virtual environment to both efficiently create the training set (e.g. to train the algorithm) and conveniently build the close-loop feedback testbed (e.g. to evaluate the algorithm). The training set generated from the virtual environment contains pristine labels for interested objects as well as pristine values for measures of their spatial states (e.g.
position). Such a training set becomes the basis for calibrating machine learning algorithms that are capable of effectively and efficiently labeling interested objects and estimating their spatial states from real-world images. Other major contributions are:

1. Designed is a new driving scene understanding level that lies between complicated 3D semantic reconstruction and direct mapping from image to driving commands. Empirical results demonstrate that the proposed direct perception approach can accurately estimate many driving-relevant affordance indicators.

2. Developed is a new pipeline through which state-of-the-art deep learning algorithms can enable safe and efficient computer-vision based autonomous driving.

3. Proposed is a new way of incorporating temporal information that improves the understanding of driving scenes.

These contributions are combined into the DeepDriving approach which is deemed to be promising and viable to achieve autonomous driving. For future work, the DeepDriving approach can be advanced by incorporating the following extensions:

- **More encompassing virtual environment**

  A more encompassing virtual environment is crucial to develop good algorithms for autonomous driving. The virtual environment needs to have: 1) more realistic graphics models, and 2) more complex and dynamic virtual physical environment.

  Currently, the driving simulator being used is based on TORCS, which only has simplified racing driving scene. Although creating an urban driving environment in TORCS is very difficult, it is still possible to make a more realistic environment for highway driving. In our current setting, the lane marking models are very elementary - a lot of frequently seen lane markings are not included in the training set. A few more lane markings can be created in TORCS, as shown in Figure 7.1. Different lane markings lead to different driving behaviors. For example, with the broken and solid
yellow lines in Figure 7.1c overtaking from the opposing lane is prohibited, while with the solid and broken yellow lines in Figure 7.1d overtaking is permitted. Thus, diverse training samples for different situations need to be collected in the future.

(a) Double solid yellow lines, one-lane (b) Double solid yellow lines, two-lane (c) Broken and solid yellow lines (d) Solid and broken yellow lines

Figure 7.1: New lane marking models in TORCS.

To study urban driving, it is necessary to have a virtual environment that can simulate urban regions, including intersections, traffic signs, pedestrians, cyclists, etc. For example, Figure 7.2 shows the road texture that mimics the pavement markings in an intersection. Although the pavement markings look very similar to the real ones, other important elements are still missing in TORCS, such as pedestrians, traffic lights, traffic signs, and so on. Creating a new virtual environment that has those elements for urban driving is essential for future improvement.

Figure 7.2: Pavement markings at an intersection. Left: screenshot of TORCS, right: real image.

- 3D convolution for motion features

In the convolutional layers of our direct perception ConvNet, only 2D convolution on single image is conducted, which results in static deep feature vectors. Although
the LSTM unit is effective in incorporating long term dependency into the ConvNet, its input at each time step is still the static deep feature vector. Including short term motion information could probably further improve the performance of the system. So it is promising to replace the current 2D convolution with 3D convolution [177, 176, 162], which is conducted on a sequence of consecutive frames.

- **More advanced model for urban driving**

  In this thesis, to prove the new concept of direct perception, we choose highway driving as the scene for our model. Highway driving is much simpler than urban driving, while the latter is the ultimate goal for autonomous driving. After successfully demonstrating our approach on highway driving scenes, it is natural to set the next goal as to develop more advanced direct perception models for urban driving, which encompass the modules for recognizing traffic signs, traffic lights, pedestrians, etc.

- **Larger field of view**

  Our current system takes images with the aspect ratio of 4:3 as input, which has a relatively small field of view. In real life, when driving, we all prefer a large field of view, and this is also true for a computer. A future system can use panorama image [203, 193] or multiple images from various view points as input, since such settings can create a larger field of view for the cognition module.

- **Reducing the domain difference**

  In our DeepDriving approach, the training is done only in the virtual environment, while our goal is to apply the model to the real world. This domain difference issue needs to be considered. It is an important direction to study methods that can reduce the domain difference, such as domain adaptation, transfer learning, combining computer graphics models and real images for training.
Appendix A

Deep Learning-Based Small Object Detection

A.1 Introduction

We have witnessed several breakthroughs for object detection in the past decade, demonstrated by the ever-increasing performance improvement on PASCAL VOC. However, it is still a common belief among vision practitioners that it is very hard to make object detection work in the real world. Torralba and Efros [174] pointed out a major problem is the dataset bias on image appearance.

In this chapter, we focus on a related but slightly different types of bias: size selection bias. All the state-of-the-art object detectors focus on big objects, ignoring objects with smaller physical sizes or with fewer pixels (see Figure A.1). Almost all object categories in PASCAL VOC [43](Everingham et al., 2010) are very big (see Table A.2 and A.1). “R-CNN minus R” [114](Lenc & Vedaldi, 2015) pointed out that a constant set of region proposals can even work very well, which vividly demonstrates the problem. Quantitatively, there is roughly a linear correlation between average precision and median relative object area for each category (see Figure A.13b). Such
a correlation indicates the small object detection is indeed very challenging, but it also shows the huge potential for improvement in the object detector.

Figure A.1: Detecting small objects with low-res input is a must-have component of a robust vision pipeline.

It is true that one could argue we can always have a higher resolution image or take a closer snapshot of a small object in order to detect it. But this low-res input for small object is deeply embedded in the nature of visual perception, and a robust vision system should be able to deal with it. For example, the physical size of a typical desk and monitor is always many times bigger than a mouse. As a human, when we see a desk with a monitor and a mouse, we recognize all of them in one shot. We do not look particularly closer to the mouse to put a large image at the center of our retina. Detecting small objects with low-res input is a must-have component of a working vision pipeline.

However, there are several unique challenges for detecting small objects at a low resolution. First, there are a lot more possibilities for the locations of small objects. The precision requirement for accurate localization is much higher than typical PASCAL objects. Second, there are much less pixels for small objects, which means much weaker signal for the detector to utilize. Third, there is a void in the knowl-
edge. Practically, there is no benchmark for such tasks. In fact, we do not know how difficult this task is or how well an existing object detector works.

In this chapter, we study the applicability of the state-of-the-art object detection algorithms to the small object detection problem. First, we establish a benchmark for small object detection and evaluate baseline algorithms to lay the foundation for research. Second, we propose a region proposal network to learn objectness for small objects in order to obtain a small set of proposals while still keeping high recall. Third, we propose an automatic way to encode context from surrounding areas of the small object and show that context significantly improves the detection performance. We present our findings and summarize a few guidelines for vision practitioners. We hope to raise attention of the community and stimulate discussion regarding this very important, but largely neglected issue.

### A.1.1 Contributions

This chapter makes the following contributions: First, we propose a dataset containing diverse small objects for facilitating the study, and study of the applicability of state-of-the-art deep learning-based object detection algorithms for detecting small objects in the image. Second, from systematic experiment design and performance comparison, we conclude three good practices for applying deep learning-based object detector for detecting small objects:

1. Up-sampling the region proposals helps despite the distortion.
2. The deep feature map used for region proposal generation should be carefully chosen.
3. Large context regions helps.
A.2 Small object dataset

The PASCAL VOC [43](Everingham et al., 2010) is the most widely used benchmark dataset for object detection. It contains 20 object categories including “cow”, “vehicle”, and “dog”. The object instances in the PASCAL VOC are usually large. Many of them occupy a major portion of the image. Our focus is on the small object detection problem where the object instance should only occupy a small portion of the image. In this sense, directly using the PASCAL VOC dataset is inappropriate. For better representing the problem, we compose a new dataset by using a subset of images from the Scene UNderstanding database (SUN) [192](Xiao et al., 2014), which contains a large amount of small objects in a variety of scenes. We also borrow some filtered small object images (e.g. mouse) from the Microsoft COCO dataset [118](Lin et al., 2014), which is composed of images of common objects from everyday scenes. We call our dataset the small object dataset. We manually select ten small object categories where the largest physical dimension of instances in the categories are smaller than 30 centimeters. The selected object categories are “mouse”, “telephone”, “switch”, “outlet”, “clock”, “toilet paper”, “tissue box”, “faucet”, “plate”, and “jar”. A small object is not necessarily small in the image. For instance, the “tissue box” may occupy a large portion of an image. We use the ground truth bounding box locations in the SUN and COCO datasets to prune out big object instances and compose a dataset containing purely small objects with small bounding boxes.

The statistics of the small object dataset is shown in Table A.1. It contains about 8,393 object instances in 4,925 images. The “mouse” category has the largest number of object instances: 2,137 instances in 1,739 images. The “tissue box” category has the smallest number of instances: 103 instances in 100 images. All the object instances in our dataset are small. Median of relative areas (the ratio of the bounding box area over the entire image area) of all the object instances in the same category is between 0.08% to 0.58%. This corresponds to $16 \times 16$ to $42 \times 42$ pixel$^2$ areas in a VGA image.
As a comparison, median of relative areas of object categories in the PASCAL VOC dataset is between 1.38% to 46.40%, as shown in Table A.2. Even the smallest object category is much larger than the biggest object category in our dataset.

<table>
<thead>
<tr>
<th>Category</th>
<th>mouse</th>
<th>telephone</th>
<th>switch</th>
<th>outlet</th>
<th>clock</th>
<th>toilet paper</th>
<th>tissue box</th>
<th>faucet</th>
<th>plate</th>
<th>jar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>1739</td>
<td>345</td>
<td>425</td>
<td>916</td>
<td>746</td>
<td>157</td>
<td>100</td>
<td>1094</td>
<td>419</td>
<td>252</td>
</tr>
<tr>
<td>Number of instances</td>
<td>2137</td>
<td>363</td>
<td>487</td>
<td>1210</td>
<td>814</td>
<td>175</td>
<td>103</td>
<td>1388</td>
<td>1005</td>
<td>711</td>
</tr>
<tr>
<td>Median relative area</td>
<td>0.35</td>
<td>0.38</td>
<td>0.08</td>
<td>0.08</td>
<td>0.25</td>
<td>0.40</td>
<td>0.58</td>
<td>0.43</td>
<td>0.37</td>
<td>0.29</td>
</tr>
<tr>
<td>Median top-10% area</td>
<td>2.76</td>
<td>1.99</td>
<td>0.33</td>
<td>0.37</td>
<td>1.92</td>
<td>1.43</td>
<td>1.94</td>
<td>2.02</td>
<td>2.40</td>
<td>1.57</td>
</tr>
</tbody>
</table>

Table A.1: Statistics of our small object detection dataset.

<table>
<thead>
<tr>
<th>Category</th>
<th>cat</th>
<th>sofa</th>
<th>train</th>
<th>dog</th>
<th>table</th>
<th>bike</th>
<th>horse</th>
<th>bus</th>
<th>aero</th>
<th>person</th>
<th>bird</th>
<th>cow</th>
<th>chair</th>
<th>tv</th>
<th>boat</th>
<th>sheep</th>
<th>plant</th>
<th>car</th>
<th>bottle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median relative area</td>
<td>46.40</td>
<td>33.87</td>
<td>30.96</td>
<td>23.73</td>
<td>25.69</td>
<td>23.15</td>
<td>23.04</td>
<td>22.83</td>
<td>14.38</td>
<td>8.14</td>
<td>6.09</td>
<td>5.96</td>
<td>3.82</td>
<td>3.34</td>
<td>2.92</td>
<td>2.79</td>
<td>1.38</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Median relative area of the object categories in the PASCAL VOC dataset.

Our small object dataset is considered more challenging than the PASCAL VOC in at least two ways: First, the appearance cue available for distinguishing a small object from background clutters is much less due to the small size. Second, the number of bounding box hypotheses for a small object in an image is much larger than that for a big object in the PASCAL VOC.

This small object dataset is used as our foundation for studying the applicability of deep learning-based object detection algorithm for detecting small objects. During evaluation, the small object dataset is split into two subsets: one for training and the other for test. The number of object instances per category in the training set is roughly two times the corresponding number in the testing set. There are no common images between the two sets.

**Performance metric:** we use the standard performance metric for comparing various object detection algorithms. An object bounding box hypothesis is considered as a true detection if its overlap ratio with the ground truth bounding box is greater than 0.5, where the overlapping ratio is measured using the Intersection over Union.
(IoU) measure. The detection algorithm returns a confidence score for each object bounding box hypothesis. We vary the threshold and compute the precision recall curve for each object. We then use the average precision of the curve to report the performance of the detector for an object category. The performance of the detector for the entire dataset is measured using the mean Average Precision (mAP) score.

A.3 R-CNN style small object detection

Girshick et al. (2014) proposes the R-CNN algorithm, which combines convolutional neural networks with bottom-up region proposals for object detection. R-CNN has been established as the de facto algorithm for deep learning-based object detection. It significantly outperforms conventional approaches in the PASCAL VOC by capitalizing the following two insights: First, it uses object proposals rather than sliding windows. Before the R-CNN, most object detectors such as DPM adopt a image pyramid plus sliding window approach to generate potential object locations and handle various scales. In the R-CNN pipeline, a fixed number (e.g. 2000) of boxes are proposed per image which most likely contain the target objects. The problem of various scales is also handled automatically by the proposal generation. Fewer but better proposals contribute a lot to the good performance of R-CNN. Second, it leverages ImageNet (Russakovsky et al., 2015) pre-trained model. The R-CNN ConvNet is pre-trained on ImageNet, and then fine-tuned on the PASCAL VOC. The pre-training process is proven to be crucial to the performance. Without the pre-training process, the R-CNN cannot even work by only training on the PASCAL VOC.

Given the region proposals, training a R-CNN style object detection network generally composing two major steps: supervised pre-training and domain-specific fine-tuning. During supervised pre-training, ImageNet data is used to train the entire
ConvNet from scratch. In the domain-specific fine-tuning, the weights of the ConvNet for the target domain (e.g. PASCAL VOC) are initialized by the pre-trained model. Training images for the ConvNet are region proposal patches being resized and warped to the required resolution (e.g. $227 \times 227$). Both the positive and negative patches are sampled from the region proposals according to certain overlap thresholds.

We are interested in the small object detection problem. In the following sections, we investigate into various necessary changes for successfully implementing the R-CNN style algorithm for small object detection. We follow the same procedure to train our small object detection networks, but based on the nature of the problem, in the domain-specific fine-tuning stage, we only sample the negative patches from the region proposals. The positive patches are generated by randomly deviating from the ground truth box. We also try to balance the positive patches of each category by sampling complementary number of positive patches per category per instance.

Fast R-CNN [64] (Girshick, 2015) simplifies the R-CNN pipeline by proposing a ROIPooling layer that crops the proposals from the feature map instead of the input image. Although the Fast R-CNN reduces the time cost and further improves the performance on PASCAL VOC, the core idea of R-CNN is intact. As we do not have any knowledge about how the deep learning-based method works on small object, the original R-CNN pipeline provides a more convenient way to better understand the problem. For example, it is more convenient to visualize the neuron responses of the original R-CNN than the Fast R-CNN. By working with proposal patch input, analyzing the effects of up-sampling and context is also easier. Thus in this chapter, we choose to follow the original R-CNN, but we believe our method can be naturally extended to the Fast R-CNN pipeline.

Moreover, in our work, we do not implement bounding box regression. Although bounding box regression is proven as an effective way to increase the localization accuracy, we do not feel it is a major issue for small object detection. We believe
the challenges come from the region proposal generation and classification, while bounding box regression will be less useful on poor proposal and classification results. So in this chapter, we will only focus on generating better region proposals and searching for stronger classifiers.

A.3.1 Small proposal generation

Selective search and edge box are two popular choices for object proposal generation. They use mid-level image cues, such as segments and contours and are object category-agnostic. While the selective search and edge box work well for generating proposals for big objects in the PASCAL VOC. We empirically find them rendering unsatisfactory results for generating small object proposals even after an exhaustive search of the algorithm parameter space. With 2000 object proposals per image, the typical recall rate is lower than 60%, leading to poor performance for detecting small objects using R-CNN. Further investigation shows that both of the algorithms favor salient objects with closed contours and distinctive colors. Since the nature of the small objects are non-prominent, they are non-ideal for small object proposal generation.

The Region Proposal Network (RPN) [151] (Ren et al., 2015) is the current state-of-the-art method for proposal generation. It attaches nine anchor boxes - derived from three different aspect ratios at three different scales - to each spatial dimension of the feature map from the conv5_3 layer of the VGG16 network [160] (Simonyan & Zisserman, 2014) for region proposal classification and bounding box regression. The three aspect ratios used are 0.5 (landscape), 1 (square), and 2 (portrait), and the areas of the square shape bounding boxes at the three scales are $128^2$, $256^2$, and $512^2$ pixels$^2$, respectively. The RPN achieves good performance for big object proposal generation. But we find that directly applying the RPN to the small object proposal generation results in poor performance. Several modifications are necessary as described below.
We first notice that the RPN anchor boxes are too large. Even the smallest anchor box is much bigger than most instances in our small object dataset. Based on the statistics of the small object size in the dataset, we choose $16^2$, $40^2$, and $100^2$ pixel$^2$ for the square shape anchor box sizes. For the aspect ratios, we keep the original values used in the original paper. We further notice that the stride length of the $\text{conv5.}_3$ feature map, which is 16 pixels, is too large. It is larger than most of the “switch” and “outlet” objects in our dataset. The other candidate feature maps for attaching the anchor boxes are $\text{conv2.}_2$, $\text{conv3.}_3$ and $\text{conv4.}_3$. We empirically compare the performance and find that $\text{conv4.}_3$ renders the best performance for small object proposal generation. The $\text{conv4.}_3$ feature map has a theoretical receptive field of 92x92 pixel$^2$, which appears to be more appropriate than 196x196 pixel$^2$ from the $\text{conv5.}_3$ feature map. We also try multi-resolution structures for the RPN (Figure A.2b), but results show inferior performance over single resolution RPN (Figure A.2a) that use the $\text{conv4.}_3$ feature map. So in our following experiments, we only take the region proposals generated from the $\text{conv4.}_3$ feature map.

![Diagram](image.png)

Figure A.2: **Single resolution and multi-resolution structures for RPN.**

For benchmarking the progress of deep learning for small object detection, we also apply the Deformable Part Model (DPM) [46] (Felzenszwalb et al., 2010) detector to detect the small object. The DPM detector was the state-of-the-art algorithm on the PASCAL VOC dataset before the R-CNN algorithm. The DPM detector is based on the Histogram of Oriented Gradient (HOG) features and latent support vector machine. To accommodate the small object size, we down-sample the root and part
template sizes of the DPM detector by half. The DPM is a category-specific object detector. We train a DPM detector for each class.

**Evaluation:** in Table A.3, we compare the recall rate of the proposal generation methods for the small object detection problem. Specifically, we compare the recall performance of using the DPM detector, the original RPN, and the proposed modification of RPN. We vary the number of proposals per image and show the recall numbers. The DPM is category-specific. We use the top scored bounding boxes from all the classes for computing the recall rate. The effective number of bounding boxes are 10 times the number of the RPN. As discussed, the modified RPN renders the best recall performance. For 2000 proposals, the recall rate for the “tissue box” is about 97.6%. The recall rate for the “jar” is the worst. It is 85.2% with 2000 proposals. However, this is still much better than 54.6% achieved by the original RPN method. From the table, we also find that the original RPN algorithm renders worse performance than the DPM algorithm. The proposed modification of the RPN algorithm considers the nature of small object and largely improve the performance. Overall, the proposed modification achieves an average recall rate of 91.3%, which is relatively 19% better than the original RPN method.

<table>
<thead>
<tr>
<th>Method</th>
<th>mouse</th>
<th>telephone</th>
<th>switch</th>
<th>outlet</th>
<th>clock</th>
<th>toilet paper</th>
<th>tissue box</th>
<th>faucet</th>
<th>plate</th>
<th>jar</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM, 300 prop. per category</td>
<td>70.9</td>
<td>58.0</td>
<td>70.5</td>
<td>80.9</td>
<td>79.1</td>
<td>86.6</td>
<td>76.2</td>
<td>69.3</td>
<td>58.0</td>
<td>63.4</td>
<td>71.3</td>
</tr>
<tr>
<td>RPN original, 300 prop.</td>
<td>85.0</td>
<td>63.4</td>
<td>78.7</td>
<td>73.1</td>
<td>66.0</td>
<td>76.1</td>
<td>50.0</td>
<td>76.0</td>
<td>58.6</td>
<td>31.8</td>
<td>65.9</td>
</tr>
<tr>
<td>RPN modified, 300 prop.</td>
<td>88.4</td>
<td>82.4</td>
<td>80.9</td>
<td>83.1</td>
<td>86.9</td>
<td>83.6</td>
<td>88.1</td>
<td>86.4</td>
<td>71.9</td>
<td>58.9</td>
<td>81.1</td>
</tr>
<tr>
<td>DPM, 500 prop. per category</td>
<td>73.2</td>
<td>61.8</td>
<td>74.3</td>
<td>82.2</td>
<td>82.5</td>
<td>86.6</td>
<td>78.6</td>
<td>73.9</td>
<td>62.2</td>
<td>72.9</td>
<td>74.8</td>
</tr>
<tr>
<td>RPN original, 500 prop.</td>
<td>85.7</td>
<td>64.9</td>
<td>79.2</td>
<td>74.7</td>
<td>68.4</td>
<td>77.6</td>
<td>57.1</td>
<td>78.0</td>
<td>61.4</td>
<td>38.2</td>
<td>68.5</td>
</tr>
<tr>
<td>RPN modified, 500 prop.</td>
<td>89.9</td>
<td>86.3</td>
<td>82.0</td>
<td>84.2</td>
<td>88.9</td>
<td>91.0</td>
<td>90.5</td>
<td>89.8</td>
<td>76.4</td>
<td>67.1</td>
<td>84.6</td>
</tr>
<tr>
<td>DPM, 1000 prop. per category</td>
<td>76.5</td>
<td>67.2</td>
<td>78.7</td>
<td>84.2</td>
<td>86.9</td>
<td>89.6</td>
<td>81.0</td>
<td>79.7</td>
<td>68.3</td>
<td>81.7</td>
<td>79.4</td>
</tr>
<tr>
<td>RPN original, 1000 prop.</td>
<td>87.0</td>
<td>70.2</td>
<td>79.8</td>
<td>75.6</td>
<td>71.7</td>
<td>82.1</td>
<td>66.7</td>
<td>80.9</td>
<td>66.4</td>
<td>46.2</td>
<td>72.7</td>
</tr>
<tr>
<td>RPN modified, 1000 prop.</td>
<td>92.4</td>
<td>93.1</td>
<td>83.6</td>
<td>86.0</td>
<td>90.2</td>
<td>97.0</td>
<td>92.9</td>
<td>93.3</td>
<td>82.5</td>
<td>76.4</td>
<td>88.7</td>
</tr>
<tr>
<td>DPM, 2000 prop. per category</td>
<td>80.2</td>
<td>72.5</td>
<td>82.0</td>
<td>86.2</td>
<td>89.9</td>
<td>92.5</td>
<td>83.3</td>
<td>83.3</td>
<td>73.9</td>
<td>87.8</td>
<td>83.2</td>
</tr>
<tr>
<td>RPN original, 2000 prop.</td>
<td>87.7</td>
<td>76.6</td>
<td>80.3</td>
<td>76.0</td>
<td>75.1</td>
<td>89.6</td>
<td>76.2</td>
<td>84.0</td>
<td>69.4</td>
<td>54.6</td>
<td>76.9</td>
</tr>
<tr>
<td>RPN modified, 2000 prop.</td>
<td>94.1</td>
<td>94.7</td>
<td>85.3</td>
<td>87.1</td>
<td>90.9</td>
<td>97.0</td>
<td>97.6</td>
<td>95.3</td>
<td>86.1</td>
<td>85.2</td>
<td>91.3</td>
</tr>
</tbody>
</table>

Table A.3: Recall rate of the region proposal generation methods.
A.3.2 Up-sampling

The first question encountered as studying the applicability of the R-CNN style algorithms to the small object detection problem is whether to aggressively up-sample the image or not. Unlike the objects in the PASCAL VOC, the bounding boxes of the small objects in our dataset are very small. In Table A.4 we show the median bounding box size (square root of the box area) of the objects per category and the corresponding up-sampling ratios required to match the input size (227 × 227 in this case) of the deep convolutional neural networks. We find that, generally, 6 to 7 times up-sampling is required, which will introduce a large amount of up-sampling artifacts. One way to reduce the artifacts is to use low resolution small input patches with a ConvNet deviated from the standard pre-trained models. For example, we can exclude the pre-trained weights in the last few fully connected layers and only use the convolution layers. However, using small patches as input may create other disadvantages:

- The receptive field over small patch is larger than the same receptive field over large patch. This means given a small patch, the network can only look at the object in a coarse scale, thus possibly loses useful information regarding the parts of the object.

- Small input patch produces lower dimensional feature vector, thus the size of the vector may not be large enough to accommodate all the crucial information.

- Since all the fully connected layers need to be trained from scratch, we only utilize the partial strength of the pre-trained models.

To answer this question. We design an experiment comparing the two solutions using the following two networks.

1. Partial AlexNet [106] (Krizhevsky et al., 2012): Using conv1 to pool5 layers from the AlexNet. The object proposals are re-scaled to 67 × 67. The pool5 layer
produces a $1 \times 1 \times 256$ feature vector, which is used to get the final classification scores.

2. Full AlexNet: Using the entire AlexNet structure. The object proposals are up-sampled to $227 \times 227$ and contains a large amount of artifacts.

The results are shown in [A.4]. From the table, we found that although with the up-sampling artifacts. The full AlexNet still renders much better performance. So in our following experiments, we will only use the aggressively up-sampled proposal patches as input.

<table>
<thead>
<tr>
<th></th>
<th>mouse</th>
<th>telephone</th>
<th>switch</th>
<th>outlet</th>
<th>clock</th>
<th>toilet paper</th>
<th>tissue box</th>
<th>faucet</th>
<th>plate</th>
<th>jar</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial AlexNet</td>
<td>29.8</td>
<td>3.1</td>
<td>5.3</td>
<td>18.0</td>
<td>19.6</td>
<td>15.5</td>
<td>1.9</td>
<td>6.7</td>
<td>5.4</td>
<td>2.0</td>
<td>10.7</td>
</tr>
<tr>
<td>Full AlexNet</td>
<td>42.9</td>
<td>7.7</td>
<td>9.4</td>
<td>22.7</td>
<td>28.2</td>
<td>26.7</td>
<td>15.7</td>
<td>18.6</td>
<td>5.4</td>
<td>3.4</td>
<td>18.1</td>
</tr>
<tr>
<td>Median size</td>
<td>32.4</td>
<td>54.0</td>
<td>25.5</td>
<td>25.8</td>
<td>38.5</td>
<td>73.1</td>
<td>90.0</td>
<td>50.8</td>
<td>39.2</td>
<td>29.4</td>
<td>45.9</td>
</tr>
<tr>
<td>Up-sampling ratio</td>
<td>7.0</td>
<td>4.2</td>
<td>8.9</td>
<td>8.8</td>
<td>5.9</td>
<td>3.1</td>
<td>2.5</td>
<td>4.5</td>
<td>5.8</td>
<td>7.7</td>
<td>5.8</td>
</tr>
</tbody>
</table>

Table A.4: **Up-sampling effects.** Both networks are trained and tested with DPM proposals, 500 per image per category.

A.3.3 Context

Context is an important cue for object detection. We expect that it will be even more important for small object detection, since small objects are simple in shape and usually only cover a small image region. The feature extracted from the proposal region is less discriminative, so when only given the proposal region, it can be very difficult to recognize, even for human beings.

We propose two approaches to take advantage of the context information:

1. An simple end-to-end trainable ConvNet that takes both proposal region patches and context region patches as input.

2. A ConvNet-based co-occurrence model that leverages the detection results of big objects to help better localize the small objects. The spatial relation between
the big objects and the small objects are learnt as convolutional filters through training.

A.3.3.1 End-to-end trainable ContextNet

We first investigate into the end-to-end trainable method for incorporating context information. Based on the R-CNN algorithm, we propose a simple method that works quite well. When given an object proposal in an image, in addition to cropping the proposal region, we crop the corresponding context region enclosing the proposal region, with the center coinciding with the center of the proposal region. The context region is set to be several times larger than the proposal region. We then feed both regions into a neural network. The neural network consists of three sub-networks where the first one takes the proposal region as input, the second one takes the context region as input, and the last one takes the concatenation of the outputs of the others as input and computes the final classification score. We call this neural network ContextNet, and the structure is shown in Figure A.5.

![ContextNet diagram](image)

**Figure A.3: ContextNet: the neural network for integrating context information.** The two front-end sub-networks take proposal region patches and context region patches as input respectively, the back-end sub-network takes in the concatenation of the two feature vectors and computes the final classification score.

The two front-end sub-networks have identical structure. Each consists of a few convolutional layers followed by one fully connected layer, which are derived from the first six layers of the AlexNet (or the equivalent layers of VGG16). Input image regions to the two sub-networks are resized to $227 \times 227$ ($224 \times 224$ for VGG16)
patches. Each of the front-end sub-networks outputs a 4096 dimensional feature vector. The back-end sub-network consists of two fully connected layers and outputs the predicted object category label. During training, the front-end sub-networks are initialized using the ImageNet pre-trained model. However, as the training goes on, the weights of the two sub-networks evolve separately - the weights are not shared.

**Evaluation:** we evaluate the performance of the AlexNet-based ContextNet with two variants: the 3x and 7x models. The context region of the 3x model is three times larger than the proposal region in both height and width dimension. The 7x model is defined in a similar way and it uses a very larger context region. We also include the AlexNet R-CNN model as the baseline.

The performance is shown in Table A.5. We find that the neural network with context integration achieves better performance than the baseline model. The improvement with the 7x model is slightly better than that with the 3x model. Overall, the relative mAP improvement over the baseline are 7.9% and 9.8% for the 3x and 7x models, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>mouse</th>
<th>telephone</th>
<th>switch</th>
<th>outlet</th>
<th>clock</th>
<th>toilet paper</th>
<th>tissue box</th>
<th>faucet</th>
<th>plate</th>
<th>jar</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline AlexNet</td>
<td>48.2</td>
<td>10.6</td>
<td>8.9</td>
<td>21.4</td>
<td>32.3</td>
<td>34.1</td>
<td>23.0</td>
<td>25.1</td>
<td>6.7</td>
<td>3.6</td>
<td>21.4</td>
</tr>
<tr>
<td>ContextNet (AlexNet, 3x)</td>
<td>54.8</td>
<td>9.1</td>
<td>12.8</td>
<td>30.7</td>
<td>28.5</td>
<td>28.4</td>
<td>18.6</td>
<td>30.8</td>
<td>10.6</td>
<td>6.4</td>
<td>23.1</td>
</tr>
<tr>
<td>ContextNet (AlexNet, 7x)</td>
<td>56.4</td>
<td>12.2</td>
<td>12.9</td>
<td>26.3</td>
<td>32.7</td>
<td>34.0</td>
<td>18.7</td>
<td>26.8</td>
<td>9.9</td>
<td>4.6</td>
<td>23.5</td>
</tr>
</tbody>
</table>

Table A.5: **Results of ContextNet.** All the networks are trained (2000 per image) and tested (500 per image) with modified RPN proposals.

### A.3.3.2 Co-occurrence ConvNet

As we know, large objects are much easier to detect than small objects, and some small objects have strong co-occurrence spatial relation with big objects, thus we may use the more reliably detected big objects to improve the detection accuracy of small objects.

Figure A.4 shows an example. The initial detection of “mouse” contains a lot of false positives (shown in the lower left image with green boxes, the ground truth
“mouse” locations are represented as red boxes). By knowing the locations of “monitor” and “keyboard” (indicated as blue and yellow boxes, respectively), and the spatial relation between “mouse” and “monitor”, “mouse” and “keyboard” (upper left two heat maps), we can compute a corresponding probability map of finding a “mouse” in the input image (upper right image), and further use the probability map to remove false positives (lower right image). The spatial relation map represents the probability of having a small object at each location around the big object, where the location of the big object is fixed at the center of the map. For example, the mouse-keyboard spatial relation map indicates a mouse usually shows up on the right hand side of a keyboard, and the mouse-monitor spatial relation map indicates a mouse usually presents at lower right to a monitor.

Figure A.4: **Example of how co-occurrence improves the accuracy of small object detection.** By knowing the location of monitor and keyboard, many high score false positives can be eliminated since their locations are not likely to have a mouse.
Such co-occurrence spatial relation maps can be learnt as convolutional filters in a ConvNet. We implement the ConvNet that re-scores the output of a object detection network by rejecting high score false positives and boosting low score true positives. The schematic of the network is shown in Figure A.5. In the figure, we use “mouse”, “keyboard”, and “monitor” as an example.

Figure A.5: Co-occurrence ConvNet structure. The classification results of small object region proposals and the ground truth bounding boxes of the big objects are converted to input images of the co-occurrence ConvNet. Spatial relation filters convolve with the corresponding big object location maps, and the resulting feature maps concatenate with the small object score map. All these feature maps are used to produce a new score map for the small object region proposals.

We choose three small objects: “mouse”, “toilet paper”, “faucet” and five big objects: “monitor”, “keyboard”, “toilet”, “sink” and “night table” to test our co-occurrence approach. Positive spatial relation between small objects and big objects is learnt automatically through training, we do not manually pair any small objects
and big objects. In our experiments, we use the ground truth bounding boxes of big objects in both the training and testing phase.

The co-occurrence ConvNet has the following inputs:

- **Small object region proposal score map**: the location of each non-zero point is the center of each small object region proposal (independent of the box size). The value of the point is the classification score of that region proposal outputted by the object detection network.

- **Big object location map**: the location of each non-zero point is the center of each big object bounding box (independent of the box size), with the value being the detection score. In our current setting, we use ground truth bounding boxes, so the values of the non-zero points are always one.

To train a ConvNet for co-occurrence spatial relation, the filter size needs to be large enough to accommodate the space between objects. In our implementation, we choose $81 \times 81$. Generally, with only a few thousand training samples, it is impossible to train a ConvNet with such large filters. However, as the spatial relation is based on statistics of data, the filter does not need to have high resolution. Thus, we implement a grid filter, which partition the filter area into a $n \times n$ grid (e.g. $15 \times 15$ grid for the $81 \times 81$ filter). Within each block in the grid, the parameters are shared (updated together). So a $81 \times 81$ filter actually only has $15 \times 15 = 225$ trainable parameters, which can be successfully trained.

Objects in images have various scales, and in order to convolve all the images with only one filter, we need to explicitly handle scales. We select four different scales to cover the range of the object sizes, as illustrated in Figure A.6. The following pipeline is used to handle scales:
1. Re-scale the small object region proposals and the big object bounding boxes as if the original image is resized with the longest side to 240 (preserve the aspect ratio).

2. Partition the re-scaled region proposals and bounding boxes into four different scales, as listed in Table A.6.

3. For large scales (e.g. scale 2 to scale 4), further down-sample the region proposals and bounding boxes to $1/2x$, $1/4x$, and $1/8x$ respectively to match the size of the spatial relation filter.

Figure A.6: **Handling various scales.** The small object region proposals and the big object bounding boxes are partitioned into four scales according to the size. Large region proposals and bounding boxes are down-sampled to match the size of the spatial relation filter.
### Table A.6: Four scales of the object size. The size is counted using the longest edge of the re-scaled box, in pixel.

<table>
<thead>
<tr>
<th>Category</th>
<th>scale 1</th>
<th>scale 2</th>
<th>scale 3</th>
<th>scale 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>mouse</td>
<td>≤ 10</td>
<td>11 ~ 20</td>
<td>21 ~ 40</td>
<td>≥ 41</td>
</tr>
<tr>
<td>toilet paper</td>
<td>≤ 10</td>
<td>11 ~ 20</td>
<td>21 ~ 40</td>
<td>≥ 41</td>
</tr>
<tr>
<td>faucet</td>
<td>≤ 14</td>
<td>15 ~ 28</td>
<td>29 ~ 56</td>
<td>≥ 57</td>
</tr>
<tr>
<td>monitor</td>
<td>≤ 43</td>
<td>44 ~ 86</td>
<td>87 ~ 172</td>
<td>≥ 173</td>
</tr>
<tr>
<td>keyboard</td>
<td>≤ 34</td>
<td>35 ~ 68</td>
<td>69 ~ 136</td>
<td>≥ 137</td>
</tr>
<tr>
<td>toilet</td>
<td>≤ 45</td>
<td>46 ~ 90</td>
<td>91 ~ 180</td>
<td>≥ 181</td>
</tr>
<tr>
<td>sink</td>
<td>≤ 43</td>
<td>44 ~ 86</td>
<td>87 ~ 172</td>
<td>≥ 173</td>
</tr>
<tr>
<td>night table</td>
<td>≤ 40</td>
<td>41 ~ 80</td>
<td>81 ~ 160</td>
<td>≥ 161</td>
</tr>
</tbody>
</table>

The ground truth location of small objects are used as supervision to train the ConvNet. For each small object, we train a ConvNet to encode its co-occurrence spatial relation with all the five big objects. When we visualize the filters after sufficient training, we observe that the ConvNet does learn meaningful co-occurrence spatial relation between corresponding small objects and big objects, as shown in Figure A.7 to Figure A.10, where the ConvNet is based on the results of Full AlexNet (R-CNN). The filters of non-related objects (e.g. “mouse” and “toilet”, “faucet” and “keyboard”) are close to all-zero maps.

We train the co-occurrence ConvNet on top of the results of various object detection networks, and find the co-occurrence spatial relation is not very helpful in improving the average precision. In Table A.7, we list the average precision computed using $IoU \geq 0.5$ and $IoU \geq 0.3$ criteria. From the results we observe that the co-occurrence approach has no advantage over the simple end-to-end trainable ContextNet approach.
Figure A.7: **Spatial relation filter of “mouse” and “keyboard”**.

Figure A.8: **Spatial relation filter of “mouse” and “monitor”**.
Figure A.9: Spatial relation filter of “toilet paper” and “toilet”.

Figure A.10: Spatial relation filter of “faucet” and “sink”.
Table A.7: Original results of various object detection networks and re-scored results of the co-occurrence ConvNet. The average precision is computed using $\text{IoU} \geq 0.5$ (upper table) and $\text{IoU} \geq 0.3$ (lower table), respectively.

<table>
<thead>
<tr>
<th></th>
<th>mouse</th>
<th>toilet paper</th>
<th>faucet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial AlexNet</td>
<td>29.8</td>
<td>15.5</td>
<td>6.7</td>
</tr>
<tr>
<td>Partial AlexNet + co-occurrence</td>
<td>30.9</td>
<td>12.7</td>
<td>12.1</td>
</tr>
<tr>
<td>Full AlexNet (R-CNN)</td>
<td>42.9</td>
<td>26.7</td>
<td>18.6</td>
</tr>
<tr>
<td>Full AlexNet + co-occurrence</td>
<td>33.6</td>
<td>22.1</td>
<td>18.8</td>
</tr>
<tr>
<td>ContextNet (AlexNet, 7x)</td>
<td>48.4</td>
<td>30.4</td>
<td>20.5</td>
</tr>
<tr>
<td>ContextNet + co-occurrence</td>
<td>30.0</td>
<td>21.1</td>
<td>17.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>mouse</th>
<th>toilet paper</th>
<th>faucet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial AlexNet</td>
<td>36.7</td>
<td>28.7</td>
<td>18.4</td>
</tr>
<tr>
<td>Partial AlexNet + co-occurrence</td>
<td>43.9</td>
<td>32.4</td>
<td>29.2</td>
</tr>
<tr>
<td>Full AlexNet (R-CNN)</td>
<td>49.5</td>
<td>34.4</td>
<td>34.0</td>
</tr>
<tr>
<td>Full AlexNet + co-occurrence</td>
<td>49.8</td>
<td>39.9</td>
<td>36.4</td>
</tr>
<tr>
<td>ContextNet (AlexNet, 7x)</td>
<td>54.9</td>
<td>49.2</td>
<td>38.3</td>
</tr>
<tr>
<td>ContextNet + co-occurrence</td>
<td>47.5</td>
<td>42.8</td>
<td>41.0</td>
</tr>
</tbody>
</table>

We investigate the results and find the following two reasons that could possibly explain why the co-occurrence model degrades the performance:

1. The co-occurrence ConvNet re-scores lots of proposals covering the same ground truth to high score, such overlapping proposals may not be eliminated by non-maximum suppression, thus reduces the precision.

2. The co-occurrence ConvNet re-scores false positives that match the spatial relation (e.g. headset next to the keyboard) to high score.

The precision-recall curves of “mouse” and “faucet” are shown in Figure A.11. Based on these curves, we observe that the co-occurrence approach almost always deteriorate the performance in low recall - high precision range. In that range, the
object detection network usually has very reliable performance, so the co-occurrence ConvNet may mostly reject true positives and boost false positives. In contrast, using the context region as input to the object detection network is a more convenient and effective way. As we use a large context patch, the network automatically learns all kinds of context information (possibly also including co-occurrence) through training. While in the co-occurrence ConvNet setting, we need to manually select a list of big objects and constraint the context as co-occurrence spatial relation only. The finding is reasonable, since research has already demonstrated that deep features automatically learnt by ConvNet is superior than manually engineered ones.

![Graphs showing precision-recall curves for mouse and faucet](image)

(a) mouse  
(b) faucet

Figure A.11: **Precision-recall curves of the co-occurrence ConvNet.** The ConvNet is built on top of the Full AlexNet (R-CNN)'s results, and the curves are computed using $IoU \geq 0.3$.

## A.3.4 Further analysis

In Table A.8 we list the average precision of our R-CNN models on small object dataset, we also list the DPM as a baseline. Not surprising at all, DPM is significantly outperformed by all the deep learning-based models. And deeper network (VGG16) has superior performance over shallow network (AlexNet).
Table A.8: Results of DPM, AlexNet R-CNN, and VGG16 R-CNN. The AlexNet in row 2 is trained and tested with DPM proposals, 500 per image per category. The AlexNet in row 3 and the VGG16 in row 4 are trained (2000 per image) and tested (500 per image) with modified RPN proposals.

To demonstrate the influence of region proposal quality on the final average precision, we compare two AlexNet models: one using the DPM detection outputs as proposals, and the other use the modified RPN proposals. From Table A.3, we know the modified RPN proposals have much higher recall rate than the DPM proposals, and consequently, the AlexNet trained on modified RPN proposals performs much better (Table A.8).

Fewer proposals: in Table A.9, we show the average precision of the ContextNet using 7x context region on different number of proposals per image. We find it achieves higher average precision on a smaller number of proposals. Small object detection is very vulnerable to false positives. Using a smaller number of proposals help eliminate a large amount of potential false positives and improves the average precision. We achieve the best performance with 300 proposals per image.

Table A.9: Results of ContextNet. Both networks are trained (2000 per image) and tested (various) with modified RPN proposals.
Stronger pre-trained model: we also experiment with replacing the AlexNet with the VGG16 net to verify if the performance boost in the big object detection due to the stronger pre-trained model is also true for small object detection. The results are shown in Table A.9. From the table, we find that the stronger pre-trained model leads to improve performance for all the proposal numbers.

In Figure A.12 we show the detection results of the ContextNet (AlexNet, 7x) model on several images in the testing set. We use a fix threshold and show the output bounding boxes after non-maximum suppression. Since the target objects are too small for visualization, we put a zoom-in window to highlight the output bounding boxes. From the figure, one can see that the small object detector works well on many categories. It can detect object instances with very low resolution.

Figure A.12: Examples of the detection results on some testing images.
One of the major purposes of this chapter is to study the applicability of the state-of-the-art object detection algorithms to the small object detection problem. By summarizing the findings, we now can answer this question. Our answer is based on two observations: 1) before the R-CNN algorithm, the state-of-the-art object detector on PASCAL VOC was DPM. Since our work is a starting point of small object detection, we think it is comparable to DPM on PASCAL. Shown in Figure A.13a, the average precision of our best model, e.g. ContextNet (VGG16, 7x), on small object categories is distributed in the same range (indicated by the black dashed lines) as that of DPM on PASCAL. Numerically, on the small object dataset, our deep learning-based algorithm (mAP 28.3) has close performance to the DPM on PASCAL (mAP 33.7). 2) on PASCAL VOC, the R-CNN style algorithm improves the mAP of DPM from 33.7 to 70.4. While on the small object dataset, our best model improves the mAP of DPM from 8.2 to 28.3, which indicates the deep learning models are still very effective on small objects. Thus, we think they are applicable to small object detection problem.

Figure A.13: Comparison of methods on small objects and PASCAL. In both (a) and (b), a marker represents the mAP of a detector on an object category. Specifically, red represents Faster R-CNN on PASCAL objects, green represents our ContextNet on our small objects, light and dark blue represent DPM on PASCAL objects and our small objects, respectively.
A.4 Visualization

We visualize the neurons in our ContextNet (AlexNet, 7x) model to better understand what the network learns as learning to detect small objects. We plot the training patches that excite each neuron in the fc6 layer most for both the proposal and context front-end sub-networks.

![Visualization of neurons in fc6 layer](image)

Figure A.14: The proposal patches that have the largest excitation to the neurons in fc6 of proposal sub-network. Please refer to the main text for further discussions.

In Figure A.14, we display the top 20 image patches with the highest response to several neurons in the proposal sub-network. We find that the patches are dominated by mouse and round shape objects (e.g. row 1 to row 5). This partially explains why the network performs better for the “mouse” and “clock” categories. We also find the neurons in row 2 fire when seeing Apple mouses or similar shapes, while those in row 9 response to oval pattern. In row 10, we can see outlet patches are mixed with speaker and clock patches. The neurons in row 11 and row 12 correspond to a
monitor detector and a toilet detector. This is surprising since our dataset does not contain these two object category labels. The figure also suggests that there is not much high-level features to distinguish small objects. Hence, the network relies on basic shape patterns to detect small objects (e.g. row 6 to row 8).

![Image of object detectors](image_url)

**Figure 1:** PR-curve for different models

---

In Figure A.15, we display the top 8 image patches with the highest response to several neurons in the context sub-network. Since the 7x context region covers a large image area, the context patches fire for the same neuron have more diverse patterns. As expected, strong scene-specific patterns exist on many neurons. The neuron in row 1 looks at computers, and the neuron in row 2 is developed for bedroom scenes. The neurons in row 3, 4, and 5 respond to tables, toilets and sinks, respectively. The

![Image of context patches](image_url)

**Figure A.15:** The context patches that have the largest excitation to the neurons in \textit{fc6} of context sub-network. Please refer to the main text for further discussions.
neuron in row 6 activates on kitchen scene. These neurons provide context information to resolve the ambiguity in the proposal patches.

A.5 Summary

In this chapter, we study the applicability of the state-of-the-art deep learning-based object detection algorithms to the small object detection problem. We compose a small object dataset to facilitate the study. Through detailed experimental validation and analysis, we find that, with a carefully designed region proposal network and context modeling, the deep learning-based object detection algorithm achieves similar performance improvement over the conventional approach for small object detection as it does for big object detection.
Bibliography


[58] Andreas Geiger, Martin Lauer, Christian Wojek, Christoph Stiller, and Raquel Urtasun. 3d traffic scene understanding from movable platforms. *Pattern Analysis and Machine Intelligence (PAMI)*, 2013.


