Deep Learning for Autonomous and Driver Assistant Systems

by

Farzan Nowruzi

A thesis presented to the University of Ottawa in fulfillment of the thesis requirement for the degree of Doctor of Philosophy in Electrical and Computer Engineering

uOttawa

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The main goal of autonomous driving is the complete removal of human supervision from the work-flow of autonomous vehicles. This objective represents an opportunity for enhancing quality of life by reducing traffic, removing parking spaces in cities, increasing collective fuel efficiency, and reducing accidents.

As autonomous driving is progressively getting integrated into our daily lives, viable solutions are required for its challenges. Artificial intelligence is the main technology that provides intelligent agents with the capability to perceive visual information in a way similar or even superior to human agents. In recent years the deep learning methods showed their outstanding power in dealing with various data processing tasks.

Most of the open problems in autonomous driving are focused on the surrounding environment, and some are within the cabin. This dissertation presents solutions to selected problems in both domains using deep learning methods with various sensor modalities. We introduce a model that is able to extract the geometric relationship between two camera images. These results then allow us to proceed with the development of a model to solve geometric transformation in a sequence of point-cloud observations to address the odometry problem. Our proposed method is directly consuming the point-clouds in real-time. Further, we develop the first publicly available comprehensive Radar dataset and propose an open space segmentation model for this task. Lastly, we present a method that uses thermal imaging within the vehicle to count the number of passengers. The thermal images are hiding most of the visual features of passengers and better respect their privacy.

Acknowledgements

I would like to thank my supervisor, Prof. Robert Laganiere, without whose guidance this thesis would not have been completed. I would like to further extend my gratitude to Prof. Nathalie Japkowicz with whom I had a chance to collaborate for a limited amount of time.

This thesis would have been much harder to complete without support from friends and colleagues at the VIVA lab. I would like to thank Robson DeGrande, Wassim Al-Ahmar, Prince Kapoor and Dhanvin Kolhatkar.

I have learned a great bunch of practical information from my industrial experiences. I would like to thank Maxim Reznitski, Oleg Zhukov, Eric Law, Maximillian Muffert, Ramona Stefanescu, Teymur Sadikhov, Thomas Bock, Michael Maile, Luca Delgrossi and Axel Gern.

And finally, my gratitude goes to my siblings for supporting and guiding me through this process. And a huge thanks for my parents for their sacrifices that enabled me to pursue my academic career through various challenges. I hope this achievement makes them proud.

"It is paradoxical, yet true, to say, that the more we know, the more ignorant we become in the absolute sense, for it is only through enlightenment that we become conscious of our limitations. Precisely, one of the most gratifying results of intellectual evolution is the continuous opening up of new and greater prospects."

- Nikola Tesla

Contents

Abstract i

Acknowledgements ii

1 Introduction 1
- 1.1 Autonomous Driving . . . 2
  - 1.1.1 Sensors . . . 2
  - 1.1.2 System . . . 4
  - 1.1.3 Levels of Autonomy . . . 5
- 1.2 Open Problems in Autonomous Driving . . . 7
- 1.3 Contributions . . . 8
- 1.4 Outline . . . 9

2 Deep Learning Background 10
- 2.1 Artificial Neural Networks . . . 11
- 2.2 Objective Function . . . 14
- 2.3 Optimization . . . 15
- 2.4 Regularization . . . 16
- 2.5 Model Architectures . . . 16
  - 2.5.1 Convolutional Neural Network . . . 16
  - 2.5.2 Recurrent Neural Networks . . . 18
  - 2.5.3 Encoder-Decoder Networks . . . 18
  - 2.5.4 Generative Adversarial Networks . . . 20
- 2.6 Advanced Models for Practical Applications . . . 20
  - 2.6.1 Image-to-Image Matching . . . 20
  - 2.6.2 Point-Cloud Processing . . . 21
  - 2.6.3 Semantic Segmentation . . . 22
  - 2.6.4 Object Detection . . . 23
- 2.7 Libraries . . . 24

3 Image Homography Estimation 26
- 3.1 Introduction . . . 26
- 3.2 Related Work . . . 27

3.3 Hierarchical Convolutional Network Model . . . 29
3.3.1 Network Architecture . . . 30
3.3.2 Hierarchical Model . . . 31
3.4 Dataset . . . 32
3.5 Implementation . . . 32
3.6 Experiments . . . 33
3.6.1 Accuracy Results . . . 33
3.6.2 Single Module Comparison . . . 35
3.6.3 Occlusion Analysis . . . 36
3.6.4 Model Depth Evaluation . . . 36
3.6.5 Regularizers and Optimizers . . . 37
3.6.6 Optimal Number of Modules . . . 38
3.7 Conclusion . . . 38

4 Deep LiDAR Point-Cloud Odometry **40**
4.1 Introduction . . . 40
4.2 Problem Statement . . . 41
4.3 Related Work . . . 42
4.4 Data Pre-processing . . . 45
4.4.1 Label Extraction . . . 45
4.4.2 Point-Cloud Sampling . . . 45
4.4.3 Dataset Augmentation . . . 46
4.5 Model Architecture . . . 49
4.5.1 Proposed Core Model . . . 49
4.5.2 Hierarchical Registration Model . . . 50
4.5.3 Temporal Filtering . . . 51
4.5.4 Loss Function . . . 52
4.6 Experiments . . . 53
4.6.1 Hierarchical Model . . . 53
4.6.2 Temporal Model . . . 55
4.6.3 Comparison to the State-of-the-Art . . . 58
4.6.4 Loss comparison . . . 61
4.6.5 Input Dimensionality . . . 61
4.6.6 Sampling Comparison . . . 62
4.6.7 Label Representation . . . 62
4.7 Conclusion . . . 63

5 Radar based Open Space Segmentation **64**
5.1 Introduction . . . 64
5.2 Literature Review . . . 66

5.3 Radar Data . . . 68
5.3.1 Radar Processing Pipeline . . . 68
5.3.2 Annotation Challenge . . . 70
5.4 Proposed Models . . . 71
5.4.1 PolarNet . . . 71
5.4.2 FCN_tiny . . . 73
5.5 Experiments . . . 74
5.5.1 Input Modalities . . . 74
5.5.2 Loss Functions . . . 76
5.5.3 Computational Performance Analysis . . . 77
5.5.4 Feature Compression . . . 78
5.6 Conclusion . . . 79

6 In-Vehicle Occupancy Detection using Thermal Data 80
6.1 Introduction . . . 80
6.2 Related Work . . . 82
6.3 Dataset . . . 83
6.4 Proposed Method . . . 84
6.4.1 Augmentation and Preprocessing . . . 84
6.4.2 Core Model . . . 86
6.4.3 Classification Task . . . 87
6.4.4 Multi-Task Learning . . . 87
6.5 Experiments . . . 88
6.5.1 Counting Accuracy . . . 90
6.5.2 Precision-Recall . . . 90
6.5.3 Speed . . . 92
6.6 Conclusion . . . 93

7 Conclusion 94

List of Figures

1.1 Architecture of an autonomous driving vehicle. The four aspects of human error along with the topics tackled in this thesis are overlayed with the corresponding modules. 5
1.2 Different levels of autonomous driving [9]. 6

2.1 Relationship between artificial intelligence, machine learning, and deep learning. 10
2.2 An example of a fully connected neural network. 11
2.3 Visualizing a set of activation functions. 12
2.4 Difference between traditional [113, 50] vs deep learning methods. 13
2.5 Matrix representing an image, filter, and resulting matrix from convolution 17
2.6 An example of CNN architecture for image classification [98]. 17
2.7 RNN and unfolded RNN architecture. 18
2.8 Cell structure of LSTM and GRU. 19
2.9 Encoder-Decoder network architecture for semantic segmentation. Input image and output label are taken from the cityscapes dataset [48]. 19
2.10 An overview architecture of Generative Adversarial Networks. 20
2.11 Architecture of a convolutional siamese model. Feature extraction network CNN_s is applied independently on both of the inputs. The output features of each input are integrated using a merge function and processed through the matching network CNN_m. 21
2.12 Architecture of PointNet++ classification network [140]. Number of data points in the feature map is represented by N_i, C_i shows the number of channels, d is the input dimension, K is the number of nearest neighbours to be considered in the set abstraction, and k represents the number of classes. 22
2.13 Architecture of the Single Shot Detector (SSD) model [111]. Each successive feature map has a smaller size and a larger receptive field that enables them to detect larger objects. 23
2.14 The trend of using deep learning libraries [92]. 24

3.1 Sample output and the warping process. Source and target images are provided to the system. Due to shrinking error residual after each module, warped image is getting visually more similar to the target at each step. 27
3.2 Twin convolutional neural network architecture used as the core module. 29
3.3 Hierarchical Model Framework. Each of the convolutional homography estimation modules consists of twin convolutional neural networks that perform homography estimation on an input image pair. 30
3.4 Test Error. Corner pixel error comparison of various methods. 34
3.5 Training curves showing the learning process for implemented models in $log(L_2)$ scale. 35
3.6 Parametric Evaluation. Batch normalization is benchmarked against Weight normalization. Effectiveness of batch normalization is shown for homography estimation. Performance of Momentum optimizer and Adam optimizer are very similar. 37
3.7 Hierarchy size evaluation: training error vs. test error. 38

4.1 The additive nature of error in odometry. ϵ_{01} and ϵ_{23} are the same amount of estimation errors, while only occurring in different steps of the sequence. Earlier errors add up to larger trace disparities at the end of the sequence. 42
4.2 Visualization of the sequential local pose transformations (Red) and global poses (Blue). 46
4.3 Histograms of the consecutive transformation parameters before and after data augmentation. 47
4.4 Proposed core registration model architecture to process two consecutive frames and extract transformation parameters in between them. 50
4.5 Progressive Prediction. In the first iteration, $\Delta\theta$ and Δt is estimated using PCL_0 and PCL_1. The source point-cloud PCL_0 is transformed to $PCL_{0'}$. The residual transformation between PCL_1 and $X_{0'}$ is calculated and used to estimate the next iteration of the model. 50
4.6 Temporal model along with the three different input levels. FCLF is the natural input state of the temporal model. PREF and POSF require additional layers from the registration model to be included and re-learned inside the temporal model. 51
4.7 Estimated odometry trace for KITTI sequences using *8k4k_comp* model. Sequences 00-08 are used for training, and sequences 09-10 are used as test data for the model. 54
4.8 Comparison of errors on various iterations of the proposed model. 56

4.9 Progress of the distribution of label parameters through multiple iterations of training. 57
4.10 Comparison of the performance of the temporal model. 59
4.11 Comparative results over various iterations for *comp* vs *indiv* weighting scheme in loss function. ***comp*** uses one weight for rotation component and one weight for translation component in the loss function. ***indiv*** uses individual weights for each parameter in the pose. 60
4.12 Input point-cloud dimensionality analysis. ***8k4k*** indicates 8*k* non-ground and 4*k* ground points in the input cloud. ***4k2k*** uses 4*k* non-ground and 2*k* ground points in the input cloud. 61
4.13 Comparative results for various ablation studies. ***4k2k*** uses 4k non-ground and 2k ground points as its input. ***6k*** indicates that 6k points are globally sampled without any distinction between ground and non-ground points. ***comp*** uses one weight for the rotation component and one weight for the translation component in the loss function. ***indiv*** uses individual weights for each parameter in the pose. ***tmat*** indicates that the rotation matrix is used instead of normalized Euler angles to calculate the loss. . . 62

5.1 Radar data representation. (Left) SCA representation. Arranging data along antennas results in Sample-Chirp (SC) view, arranging along chirps results in Sample-Antenna (SA), and Chirp-Antenna (CA) is achieved by arranging data along sample dimension. (Right) Fourier transformation is applied along all three dimensions. Arranging values along azimuth results in Range-Doppler (RD), arrangement along Doppler results in Range-Azimuth (RA), and arrangement along range results in Doppler-Azimuth (DA). 69
5.2 Ground truth generation pipeline. (a) Camera frames are used to reconstruct the scene and extract odometry. (b) Cartesian DoA from a single frame. (c) Accumulated Cartesian DoA which is used to generate initial annotations. (d) Annotations are distributed to the corresponding frames, cropped for the radar's field of view, and are manually adjusted. 71
5.3 Detailed architecture of the PolarNet model. 72
5.4 Sample results of PolarNet. (a) camera image. (b) ground-truth in polar coordinates. (c) segmentation result. (b) segmentation result shown in Cartesian coordinates. 75
5.5 Encoder feature compression. Trade-off between reducing depth dimension of encoder's last layer versus the accuracy of the model. 78

6.1 Thermal camera and its location for data capture. 84
6.2 A sample of an annotated image. 85

6.3 Original image from dataset and its augmentations. 85
6.4 Proposed core model used as the backbone in all the learning tasks. 86
6.5 Stages of splitting the heatmap to connected regions and applying blob detection to get the enclosing bounding boxes. 88
6.6 Comparison of confusion matrices. Top row shows the detection models. Bottom row shows the classification models. 91
6.7 Precision/Recall comparison for IOU of 0.5. 91

List of Tables

3.1 Model names and parameters. 33
3.2 Test Results. Comparison of our method with a traditional feature-based (ORB key-points) robust estimation scheme and homographyNet, both reported in [55], against our proposed approach. 34
3.3 Effect of occlusion on the performance. The left column shows the datasets used for training. The pixel-wise corner accuracy of each test dataset is noted underneath the corresponding column header. 36

4.1 Sequence translation drift percentage t_{rel} and mean sequence rotation error r_{rel} for the lengths of $[100, 800]$m. Sequences 09 and 10 are used as the test set. 60

5.1 Details of the configuration used with radar. 68
5.2 mIoU reported for different model architectures. 76
5.3 mIoU for PolarNet with RA input and polar labels with different loss function 77
5.4 Computational complexity comparison for the tested methods. 77

6.1 Data distribution. Number of images per passenger count is shown in this table. 83
6.2 Comparison of counting accuracy. Counting based on classification and detection results are presented at the top and bottom of this figure, respectively. 90
6.3 Speed comparison for classification models 92
6.4 Speed comparison for detection models 93

List of Abbreviations

ACC	**A**daptive **C**ruise **C**ontrol
ADAS	**A**dvanced **D**riving **A**ssistance **S**ystems
ADC	**A**nalog to **D**igital **C**onverter
ANN	**A**rtificial **N**eural **N**etworks
API	**A**pplication **P**rogramming **I**nterface
ATE	**A**bsolute **T**rajectory **E**rror
BEV	**B**ird **E**ye **V**iew
BN	**B**atch **N**ormalization
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
DNN	**D**eep **N**eural **N**etwork
DoA	**D**irection **o**f **A**rrival
DOF	**D**egrees **O**f **F**reedom
FCN	**F**ully **C**onvolutional **N**etworks
FPPI	**F**alse **P**ositives **P**er **I**mage
GAN	**G**enerative **A**dversarial **N**etwork
GHz	**G**iga **H**ertz
GPS	**G**lobal **P**ositioning **S**ystem
GPU	**G**raphical **P**rocessing **U**nit
HOV	**H**igh **O**ccupancy **V**ehicle
ICP	**I**terative **C**losest **P**oint
IMU	**I**nertial **M**easurement **U**nit
INS	**I**nertial **N**avigation **S**ystem
IOU	**I**ntersection **O**ver **U**nion
LSTM	**L**ong **S**hort **T**erm **M**emory
NMS	**N**on **M**aximum **S**uppression
OHEM	**O**nline **H**ard **E**xample **M**ining
RA	**R**ange **A**zimuth
RAD	**R**ange **A**zimuth **D**oppler
RANSAC	**RAN**dom **SA**mple **C**onsensus
ReLU	**Re**ctified **L**inear **U**nit
RNN	**R**ecurrent **N**eural **N**etwork
ROI	**R**egion **O**f **I**nterest
RRE	**R**elative **R**otation **E**rror
RTE	**R**elative **T**ranslation **E**rror
SCA	**S**ample **C**hirp **A**ntenna
SGD	**S**tochastic **G**radient **D**escent
SIFT	**S**cale **I**nvariant **F**eature **T**ransform

Chapter 1

Introduction

One major benefit of self-driving vehicles is their potential in increasing safety. Every year approximately 1.25 million people lose their lives in traffic accidents [150, 71]. National Highway Traffic Safety Administration (NHTSA) estimates that 94% of crashes are attributed to human errors [51]. 2% are the result of a failure in vehicles, 2% are due to environmental conditions, and the rest are for unknown reasons. Autonomous driving aims at tackling the most significant issue of these 4 categories, the *human error*. Human errors can be classified into 5 categories [51].

- **Recognition errors** that consist of lack of attention, internal and external distractions, and poor surveillance of the surrounding environment. This factor accounts for 41% of total accidents.
- **Decision making errors** factors count for 33% of accidents caused by human error. Errors such as misjudgment, driving too fast, illegal maneuvers and false assumptions of others are included in this category.
- **Reflexive performance errors** accounts for 11% of crashes and is defined as overcompensation, poor directional control.
- **Non-performance errors** such as sleeping, cause 7% of the accidents.
- The rest of errors are categorized as **other errors** and are responsible for remaining 8% of the errors.

Impaired driving causes an increase in almost all categorized errors. The promise of self-driving cars is to replace human drivers by systems with extremely low error rates and great life-saving potential. [3] predicts that from 2035 to 2045, the broader usage of autonomous driving technology can save more than half a million lives only in the United States.

From an economic aspect, accidents cause a huge burden on society. In 2010, NHTSA [141, 176] reported that the economic cost directly related to automobile crashes for US residents was at $242 billion. The comprehensive economic costs that are indirectly

related to these crashes are reported to be $836 billion just in the US. A majority of these expenses could be saved and directed towards other venues by the removal of the human factor from transportation systems.

The major disruption that self-driving technology causes in the transportation industry is considered a common societal fear. Many jobs will be lost in the short term. However, such a change has already happened before when the robotic tools replaced factory workers. In the long term, it is only the nature of the jobs that will be changing rather than their quantity, due to such transformations.

Another aspect that self-driving vehicles will greatly improve is the quality of human life. Traffic jams in the inner cities will be dramatically reduced by eliminating parking space requirements and optimizing autonomous traffic flow. This reduction in traffic jams will free more than 250 million hours of commuting every year [3].

Further, this technology will open newer markets such as autonomous ride hailing, ride sharing, and in-vehicle services. Passenger Economy [3] is expected to jump from $800 billion in 2035 to $7 trillion in 2050.

1.1 Autonomous Driving

The first modern autonomous vehicles began when the Defense Advanced Research Projects Agency (DARPA) funded the ALV project [91] at Carnegie Mellon University in 1984. Later in 1987, Mercedes-Benz partnered with Bundeswehr University Munich to initiate the EUREKA Prometheus Project [187]. DARPA grand challenge initiated in 2004, resulted in major progress in this field.

1.1.1 Sensors

To achieve autonomous driving, researchers are relying on various sensors to percept the environment.

- **Cameras** are used to visually observe the environment that a human driver can see. Multiple variations of this module are used such as monocular, stereo, omnidirectional, thermal, and combinations of them. This sensor provides a rich representation of environments which it can be difficult to process in real-time.

- **LiDARs** are one of the best sensors for depth estimation in autonomous driving and driver assistance applications. They consist of several range measuring beams that together measure the depth at many locations in the scene. This provides information about the visible shape of the objects. The downsides of lidar are twofold. One is the low vertical resolution. There is a very high horizontal resolution but the vertical resolution is limited to the number of laser beams used.

Increasing the number of beams results in an increase in price of an already expensive sensor, and also increases the size of the sensor. Another problem with lidar is that it does not provide any visual features, but a point-cloud. Processing this point-cloud is a computationally expensive task to perform.

- **Radar** is very good in mid to long range distances for detecting moving objects and their velocities. The disadvantage of this module is that it is hard to identify stationary objects due to the Doppler effect, and it does not provide much information about the shape or appearance of the object. To address these issues, radar is being improved with the development of High Definition Imaging Radar.
- **Sonar** sensors are very accurate in short distances and are mainly used to detect near-by objects to the car. It is most useful in short-range parking scenarios.

It is obvious that a combination of these sensors would be required to have a fully functioning system. There are a couple of other information sources that are fused with the aforementioned sensors to reduce the complexity of the problem and provide a more robust solution.

- **Global Positioning System (GPS)** is used to locate the vehicle. The accuracy of this sensor is highly dependent on the number of connections to satellites at any time. Differential-GPS (D-GPS) is usually used to increase this accuracy and to reduce the effect of mirroring signal in urban canyons. However, its accuracy is still much lower than the *centimeter* level localization accuracy requirement [1]. An alternative is to fuse this source with perception sensors and maps to get a very high accuracy.
- **Maps** in autonomous driving refer to the previous precise measurements of the environment. Creating and maintaining an environment map is an exhaustive process. Any change in environment could make the map obsolete and will require the environment to be remapped. The ideal is to have vehicles update their maps on the fly as they drive. There are various maps and map layers proposed for the maps but there is no unified description. These layers may include navigation, planning, localization, and traffic layers. The nature of each layer is highly dependent on the used methods and sensors. For example, if cameras are used for localization, a visual landmark map is required. In the case of lidar localization, a depth map of the environment is needed. The combination of all these layers together creates the multi-layered autonomous driving map.
- **Vehicle odometry** sensor in the car provides information about the amount of movement of the car in between each observation. This sensor is prone to errors caused by tire slip and is only reliable for very small durations. However, it is used

in combination with Inertial Measurement Units (IMU) and perception sensors to precisely estimate the movement.

- **IMUs** are high quality sensors to detect motion. They consist of a combination of accelerometers and gyroscopes to report g-force and angular rate. Similar to the odometry module, they are used to calculate the motion and are less erroneous than the odometry module. However, they lose accuracy over longer durations of consecutive predictions due to the additive nature of localization error.

1.1.2 System

There are two approaches for designing an intelligent software system for self-driving cars [197]. A first approach is termed *mediated perception* and is very common in the industry and research. In this approach, each part of the system is treated independently. Landmarks and objects of intereset (e.g., cars, pedestrians, lanes, signs) are each recognized and are integrated in a later independent stage to carry out the driving task. The second approach is *end-to-end* learning. End-to-end models avoid generating mid-level results and directly translate the sensory input data to the driving instructions. The lack of subsystem wise mid-level representation that is understandable for experts, makes these systems harder to debug. Mediated perception approaches provide higher flexibility and independence in handling errors from each system component. In this thesis, we will focus on this class of solutions.

There are a number of key components in any autonomous driving system. Figure 1.1 provides a high-level overview of these components and the usual information flow between these components. A cycle starts with the arrival of sensor readings from a perception module. The perception module is responsible for extracting required information for other modules. The localization module takes landmarks, lanes and sensor/map specific features. This module estimates the exact location of the vehicle based on map data, and reports any mismatches to the mapping module. The mapping module collects the new local map information and later sends it to the back-end for collective map updates. Perception, localization and mapping modules aim at avoiding the recognition errors performed by human drivers and that account for 41% of accidents.

Planning module receives stationary and non-stationary objects, open space analysis, and lane information. This information is used to predict behavior of non-stationary objects first. Then, this data is processed and analyzed to come up with a short-term plan to avoid collision with other objects while staying on the drivable space given rules provided by the map. The navigation system inside this module provides long-term directions in order to reach the destination. This module is targeting the decision making errors made by human drivers that are the cause of 33% of crashes.

FIGURE 1.1: Architecture of an autonomous driving vehicle. The four aspects of human error along with the topics tackled in this thesis are overlayed with the corresponding modules.

The devised short-term plan is translated to driving commands in the control module. This module is responsible for realizing the plan given driving constraints such as maintaining a smooth driving situation. The reflexive performance errors of human drivers are solved by this module. Non-performance errors related to human drivers, such as sleeping, are completely eliminated by the autonomous nature of the system.

1.1.3 Levels of Autonomy

Development of autonomous driving technology is an evolutionary process. The Society of Automotive Engineers (SAE) has developed a standard to measure the progress of this technology designated as the *6 levels of autonomy* [9]. Figure 1.2 provides an overview of these levels.

At level 0 there is no autonomy and all the controls are assigned to the human driver. As we move higher in the hierarchy, the amount of human interactions decreases.

At level 1 there is a very limited contribution from the system. An example of this level is standard cruise control. Once the driver sets a value for the speed of the vehicle, the system only tries to keep the vehicle at that specific speed. At this level, the system is acting passively and can not react to environmental conditions. For example, if it is approaching another vehicle, then the driver has to take immediate actions. Level

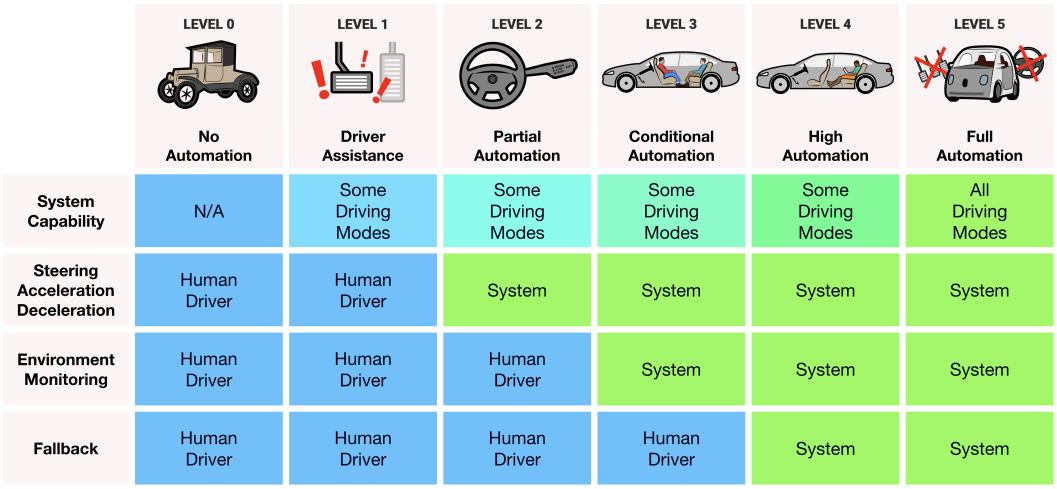


FIGURE 1.2: Different levels of autonomous driving [9].

2 aims at providing a solution to this issue. At this level, the vehicle is able to do limited observations and perform multiple autonomous tasks to adapt its driving behavior. Adaptive cruise control system is an instance of this class in which the vehicle can maintain a driver defined speed and slow down if it approaches the car ahead at less than a specific distance. Once the leading vehicle has moved, it can resume to the predefined speed. Level 3 is termed conditional automation. From this level, the system is taking the responsibility of monitoring the surrounding environment and performs most of the dynamic driving tasks. However, human drivers are still required to take control in case of system failures. Level 4 autonomy takes this burden off from the driver and provides a completely autonomous system. The only limitation at this level is that the system's ability to drive is constrained to known predefined environments. For example, an environment that has been mapped in advance for localization and for which the rule-set is completely defined for planning. Once the system leaves the known environment, the human driver has to take over. Finally in level 5, the system becomes capable of driving in unknown environments. In this case, human intervention is completely eliminated from the system in all driving modes and conditions.

At every level of autonomy, each system should satisfy the functional safety requirement ISO 26262 [89] to be in production. At the time of writing this thesis, Google Waymo [186] is the industry leader and is aiming to achieve level 4 autonomy. There are numerous universities, corporations, and start-up companies involved in the realization of the promise of autonomous cars, however no one has achieved a complete level 4 autonomy yet. This is mainly due to various challenges that a system could face in a dynamic environment. Here we are defining a subset of these problems from the perception perspective and we are seeking answers for them.

1.2 Open Problems in Autonomous Driving

In this section we discuss a few obstacles that need to be overcome in order to achieve autonomous driving. These problems mainly focus around issues that can be solved using machine learning and computer vision techniques.

In order to safely navigate with a self-driving car, the exact position of the vehicle should be localized within a few centimeters of accuracy. Localization error is a major problem in autonomous driving. A lateral localization error larger than 0.5 meter could put the vehicle in the wrong lane and cause irreparable damages. Localization is performed by calculating odometry and observation matching with the maps. The latter is generally more computationally expensive, but errors in both could cause system failures. To get the most accurate odometry, D-GPS, vehicle odometry, IMU, and sensor odometry readings are used. D-GPS is location sensitive and could be jammed. Due to the tire slip, vehicle odometry is less reliable than IMU. A constant error in acceleration measurement of IMU translates to a linear error in velocity, which results in a quadratic error in positioning. Sensor odometry as an independent source could be used to overcome this issue.

Autonomous cars operate in a highly dynamic environment with various forms of obstacles. All the objects around the vehicle such as pedestrians, cars, cyclists, and generally any object that could be an obstacle, need to be detected and localized with respect to the vehicle. Further, these objects have to be tracked and their trajectory must be predicted. This information is vital for the planning layer to make decisions in order to avoid collisions. Various sensors are used for this purpose. Cameras provide a rich and informative view of the environment that makes it easy to detect any object. Their shortcoming is in the localization accuracy after detection. Lidar sensors provide a high localization accuracy, but detecting objects in a point-cloud is generally a more challenging task. Radars are another low cost source of information for measuring the velocity of objects. However, they have a difficult time in detecting stationary objects. In practice, highest performance and reliability is always achieved by fusing multiple information sources. However, adding more sensors increases the production and maintenance costs of the autonomous vehicles.

Having the future of self-driving technology and smart cities in perspective, we address another issue that is of vital importance for the effectiveness of collective autonomous solutions. In order to balance the traffic load and serve passengers efficiently, available capacity in each autonomous vehicle has to be known. This entails developing systems that are able to count passengers occupying each vehicle. Seat pressure sensors are commonly used to count the number of seated passengers. Limitation of this model is that it can not identify if the seat is occupied by a person or just an object. An alternative approach is to use cameras in combination with computer vision techniques to

count the passengers. While devising such a solution, user-privacy measures have been considered. Visual spectrum camera images are the most common form of data being used in the industry. Replacing them with thermal sensors provides a better solution in terms of privacy, while providing the capability to be used for occupancy detection.

In order to have a fully functioning fleet of autonomous cars, we need to ensure all of these problems have reliable solutions. As it is apparent, perception is at the core of any solution for these tasks. In recent years, deep learning methods have shown state-of-the-art performance at tackling challenges in perception.

1.3 Contributions

In this thesis, we selected four problems in autonomous and assisted driving systems, and tackle them using computer vision and deep learning methods in each chapter. Our contributions are as follows:

- **Image based homography estimation [132].** We show that the standard frameworks of the convolutional networks are not suitable to perform precise homography calculation from images. We introduce a new way of using a hierarchy of small networks to provide a solution to this issue. Extensive experiments show the efficacy of the proposed model.

- **Lidar based odometry estimation.** We propose the usage of hierarchical deep models in this challenge. A novel architecture is developed and its performance is compared against the state-of-the-art. Further studies have been performed on various parameters related to the model and the dataset.

- **Radar based open space segmentation [131].** We collect a dataset of raw radar observations and publish it to enable the community to use it for the research in this field. We further apply deep learning models on various radar representations to perform open space segmentation. Finally, a novel open space segmentation network, PolarNet, is proposed that provides state-of-the-art performance while maintaining small computational complexity requirements. This research was informed by collaboration with Sensorcortek Inc. The author's role in this research involved participation in system design, data collection, ground truth generation and training various models. Moreover, PolarNet was designed, trained and evaluated by the author.

- **Thermal camera based passenger counting [130].** We propose a light-weight network to count the passengers in a vehicle that is able to achieve very high frame rates, making it suitable for low powered platforms. Further, we collect and annotate a dataset for this specific task, and share it with the research community.

1.4 Outline

Chapter 2 provides a background on deep learning and explains the building blocks of the methods that are used in later chapters. In **Chapter 3**, the homography estimation using convolutional networks is addressed. Inspired by the results of homography estimation, **Chapter 4** presents usage of hierarchical models to solve the lidar based odometry problem. In simplest form, odometry is the translation matrix of the sensor from one frame to the other. This is closely related to homography estimation but it is a more complicated task. The majority of the research since the advent of deep learning is focused on cameras and lidars. **Chapter 5** is focusing on radar, an outcast sensor in this era that is still relying on traditional processing methods, to perform open space segmentation. **Chapter 6** turns the focus to inside the vehicle in order to count the number of passengers occupying it using thermal camera images. It is crucial to know the passenger count for tasks such as automating HOV lane usage or route planning for autonomous cars in order to increase the efficiency of passenger delivery. And finally, we conclude this thesis in **Chapter 7**.

Chapter 2

Deep Learning Background

Self-driving cars need to be at least as intelligent as humans in the task of driving. Artificial Intelligence is the field that is responsible for introducing intelligence into computer systems. As we cannot manually design the rule set for all the circumstances in the dynamic driving environments, we rely on a smaller subset of artificial intelligence methods, termed Machine Learning. Machine learning methods are able to learn their own parameter space to define the rules and adapt themselves to the conditions. For example Support Vector Machines (SVM) [50] that is a prominent non-parametric machine learning model, learns a set of support vectors to classify the data. In the case of image data, visual features [113, 15, 103, 5, 152, 173] need to be extracted from the data in order to train a SVM. Traditionally, the methodology to extract these features was by manually hand-crafting them by the experts in the field. The number of support vectors to be learned grows with the number of samples presented to the SVM. This results in a slow down in learning, inference, and an increase in size of the model. These models enjoyed a great success dealing with smaller scale problems. However, adapting them to work with large amounts of data was not an easy task.

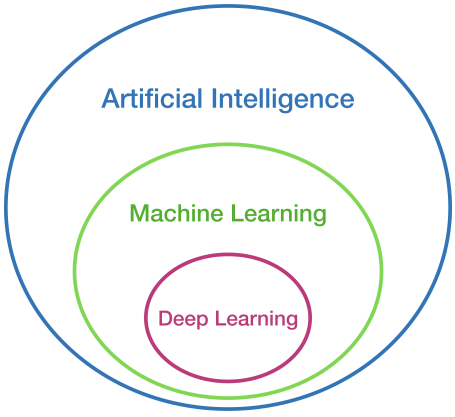


FIGURE 2.1: Relationship between artificial intelligence, machine learning, and deep learning.

Deep learning methods are a subset of machine learning algorithms that have shown

significant power at handling large amounts of data. Unlike traditional models, once a deep model is defined, its size becomes independent of the sample size used during the training. The disadvantage of deep models are their heavy processing power requirements. However, the highly parallelizable nature of them along with the recent advances in the Graphic Processing Unit (GPU) technology resulted in their applications on various problems. The introduction of convolutional layers [102] to extract image features from the image data made these models a very popular choice for perception tasks.

2.1 Artificial Neural Networks

To understand deep learning, first we need to understand Artificial Neural Networks (ANN) [19]. ANNs are a subset of machine learning methods and Deep Neural Networks are a variation of ANNs that consist of many layers. A simple neural network is shown in figure 2.2 with two hidden layers and one output layer. As all the neurons are connected to all the neurons in the next layer, this model is called Fully Connected Neural Network (FC-NN).

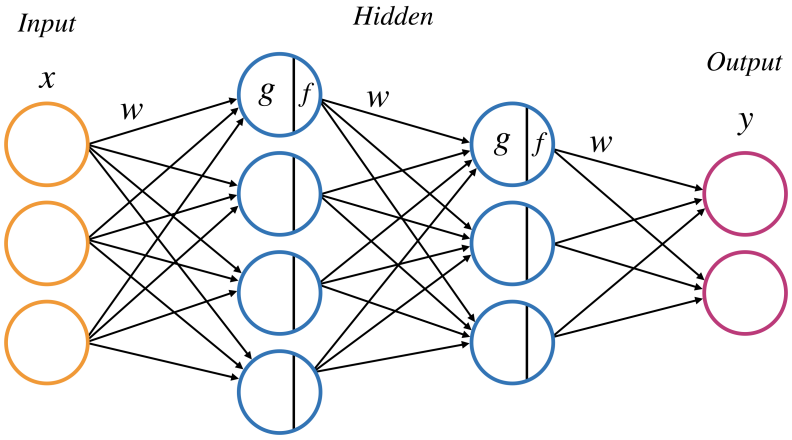


FIGURE 2.2: An example of a fully connected neural network.

At each layer, a set of transformations are applied to the inputs. In the forward pass process all the neurons of previous layers are weighted, and summed and a bias is added to them in order to produce the intermediate output.

$$g = \mathbf{w}^T \cdot \mathbf{x} + b \tag{2.1}$$

$\mathbf{x} = \{x_0, \ldots, x_N\}$ is the input vector to the neuron and $\mathbf{w} = \{w_0, \ldots, w_N\}$ defines the corresponding weight vector to the input. b is the added bias at the neuron. g is the intermediate output of the neuron that is only the linear transformations of the inputs.

The result of successive application of linear models can be recreated using a single linear transformation. Effectiveness of neural networks is induced from the non-linearity applied by their activation function to the intermediate outputs in each layer.

$$y = f(g) \tag{2.2}$$

f is the non-linear activation function and x_i is the calculated input for the next layer. Sigmoid and *tanh* functions are one of the earliest functions used as activations. Rectified Linear Unit (ReLU) is introduced in [126] that prepares the ground for modern deep learning models. ReLU has shown to have a better performance compared to previous functions and it is the most widely used activation function. Leaky-ReLU is introduced to deal with the dying ReLU problem, where negative values are always set to zero by ReLU [116]. This is achieved by using α parameter to weight the negative values. Exponential Linear Unit (ELU) [47] removes the bias shift from ReLU and centers the activations around zero to speed up the training. Gaussian Error Linear Unit (GELU) [81] is a non-monotonic function that weights the inputs using a zero mean normal distribution. [143] use an automatic search technique to discover scalar activation functions to replace ReLU without changing the network. The new activation function is termed SWISH that weights the inputs with a parameterized sigmoid function. Figure 2.3 shows these activation functions. [143, 136] provide a comprehensive review of these functions.

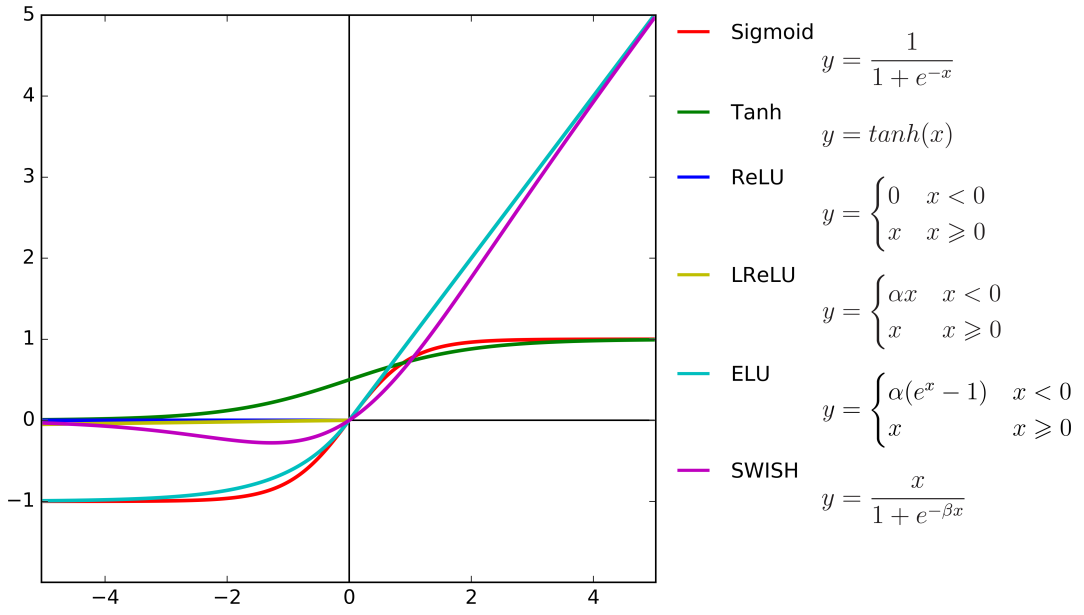


FIGURE 2.3: Visualizing a set of activation functions.

Training in ANN models is based on Back Propagation. The error between prediction made for inputs of the network and the actual target is calculated using an objective function. Utilizing the chain rule, partial derivatives of the error with respect to weights are computed. Derivatives are used in updating the weights of the network. The process of forward propagation and backpropagation is repeated until a convergence is achieved. This Converged model is considered a trained neural network. There are various strategies to perform back propagation. True gradient is achieved by finding error for all the samples. This is called *Batch Gradient Descent* and entails a heavy processing load. An alternative is to perform updates after error is calculated for each sample, *Stochastic Gradient Descent*. Instead of calculating gradients for each sample, we use a *mini-batch* of the samples at each cycle that is large enough to be representative of the underlying distribution of the dataset.

A major difference between a deep learning model and traditional models is in the way they extract features. Instead of hand-crafted features, deep models learn what to extract from data during training. By processing data at each layer, it creates feature maps using automated non-linear feature extractors and learns complex features as it goes deeper. The whole network learns a hierarchy of abstract representations and uses them for classification. This provides an ultimate power to adapt the system to the data.

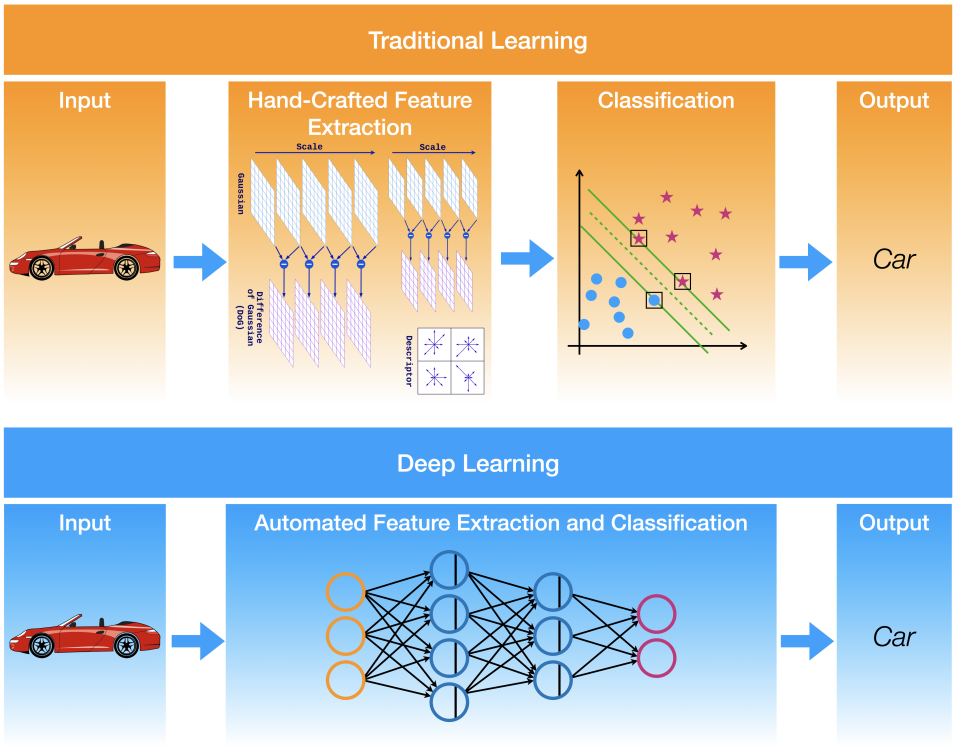


FIGURE 2.4: Difference between traditional [113, 50] vs deep learning methods.

Deep learning requires large amounts of data. In recent years, the reduction in data storage costs has enabled a drastic increase in availability of the large data. Deep models require many cycles to be trained. This requires a huge amount of processing power. Most of the calculations in a deep model can be parallelized. Graphical Processing Units (GPU) are designed for parallel processing and are the best choice for training deep neural networks.

In the recent years, modern deep learning models [98] have achieved a great success in many fields including image classification [164, 153, 170, 171], object detection [70, 145, 111], recognition [135, 159], scene segmentation [34], scene understanding [161], geometric analysis [132], natural language processing [120] and many others. In the field of computer vision, they have encountered significant success over traditional methods. One of the main differences between deep models and traditional methods is the automation of previously hand-crafted feature extraction that optimizes itself to specific data and tasks. This novelty led to an ever-growing amount of interest in using these methods in many research fields. During recent years, various architectures have been proposed to deal with specific problems, such as Convolutional Neural Networks (CNN) to extract image features, Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) to extract order and temporal information from the inputs [101, 72].

Besides the various models and architectures being proposed almost on a daily basis, there are novelties regarding the ways that a model could be trained. These modifications aim at improving the convergence and speed up the exhaustive learning process, which can be of crucial importance. Such methods include regularization methods such as Batch Normalization [88], Weight Normalization [154], or optimization methods like momentum optimizer [169], Adam optimizer [95].

At this section we are explaining key components for deep learning from a limited scope. Our goal is not to provide a comprehensive review of the field, but the basics and the interesting models that are related to this research. Readers are suggested to refer to [72] for a comprehensive review of deep learning.

2.2 Objective Function

Objective function is the most important factor in any learning system. An objective function is usually defined as the dissimilarity between target and predicted values, the *loss function*. Defining this function in a proper manner is vital for building a functioning model. The definition of objective function is completely problem dependent. Here, we are explaining a set basic loss functions that are being used in various areas.

L_2 loss defines the Euclidean distance between targets and predictions.

$$L_2(P,T) = \sqrt{\sum_i (p_i - t_i)^2} \quad , \quad \forall p_i \in P, \forall t_i \in T \tag{2.3}$$

p is the final predictions of the network and t is the target values. L_2 loss is commonly used in problems that deal with regression.

Soft-max Cross Entropy is designed for training classification models. Soft-max Cross Entropy loss is consisting of two parts. The soft-max part, also termed normalized exponential function, is taking logits and converting them in a way that they represent a probability distribution function [19].

$$f(P) = \frac{e^{p_i}}{\sum_j e^{p_j}} \quad , \quad \forall p_i \in P \tag{2.4}$$

Once soft-max is calculated, results are fed into a cross entropy function, along with target values, to calculate the loss. The discrete cross entropy is defined as,

$$CE(P,T) = -\sum_i t_i \log p_i \quad , \quad \forall p_i \in P, \forall t_i \in T \tag{2.5}$$

Focal Loss Recently, a variation of soft-max cross entropy loss is introduced [109]. Two parameters are introduced in cross entropy calculation. α for balancing cross entropy against class imbalance, and γ that is a focus parameter. Once these parameters are properly set, it can direct the focus of the network toward learning hard negatives and increase the average precision.

$$FL(P,T) = -\sum_i t_i \alpha (1 - p_i)^\gamma \log p_i \quad , \quad \forall p_i \in P, \forall t_i \in T \tag{2.6}$$

2.3 Optimization

Momentum optimizer is proposed to smooth oscillations and direct the Stochastic Gradient Descent (SGD) towards the minima locations in error space to speed up training. This is achieved by weighting gradients by a learning rate and adding a fraction of previous steps to it.

AdaGrad [59] is designed to deal with sparse data. It uses an adaptive learning rate that adapts based on the updated parameters. The main problem with AdaGrad is its monotonically decreasing learning rate. This results in the learning rate to shrink which at some point it becomes so small that the network stops learning.

AdaDelta [199] proposes to use a sliding window structure to deal with monotonically decreasing learning rate at AdaGrad.

Adam Optimizer [95] is similar to AdaDelta that in addition uses momentum changes in each parameter to direct the learning.

2.4 Regularization

Regularization is defined as any modification that reduces generalization error but does not result in a change in the training error [72].

Batch Normalization [88] is one of the most prominent regularization methods introduced in recent years. It takes the outputs of each layer and normalizes them along the batches. This way the problems caused by covariate shift in the data are eliminated. Further, this results in a faster convergence during training time.

Weight Normalization [154] is simply normalizing weight vectors at each layer instead of the output features. This enables the weight normalization to perform faster than the batch normalization.

Dropout [166] is introduced to target the over-fitting issue in training neural networks. The main idea of this method is to randomly drop weights from the network during training. At the test time this function is removed from the model. This prevents the network from becoming biased to specific features, hence resulting in a better performance during test time.

Early Stopping is another technique to deal with the over-fitting. During the training time the loss on the evaluation set is continuously monitored. Once the evaluation loss plateaued or started to constantly increase, training session is terminated.

2.5 Model Architectures

2.5.1 Convolutional Neural Network

Processing images with fully connected layers results in a huge set of weight parameters. In traditional feature extraction methods, convolution with predefined filters are

commonly used to extract features. Given an image I and a filter H, discrete convolution is defined as,

$$I_{xy} * H = \sum_u \sum_v I\left(x-u, y-v\right) H\left(u, v\right) \tag{2.7}$$

(x, y) are the all possible indexes over the image, and (u, v) are the size of the filter. Figure 2.5 shows the convolution results of a line detection filter on a sample image.

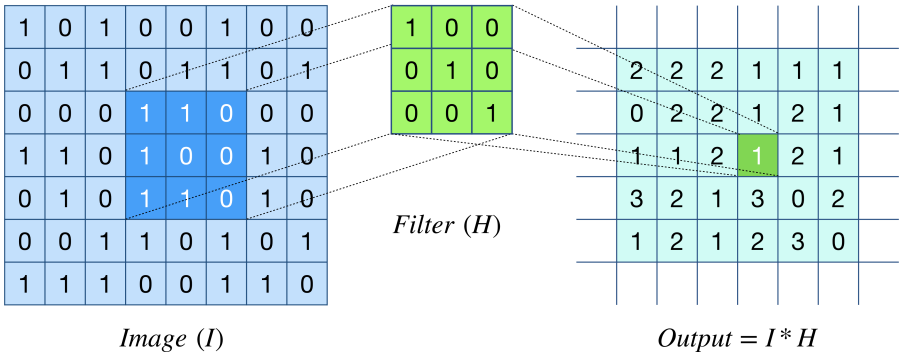


FIGURE 2.5: Matrix representing an image, filter, and resulting matrix from convolution

CNNs use a smaller set of weights as filters at each layer. These filters are learned during training. This is the ultimate superiority of deep learning compared to traditional feature extractors. Once a feature set is calculated the *pooling* function is called to reduce the dimensionality of the data that results in a faster execution performance. After each layer, a set of more complex features with smaller dimension are extracted. After a series of consecutive feature extraction layers, final features are used to get the predictions. A sample of a CNN architecture for a classification task is shown in Figure 2.6.

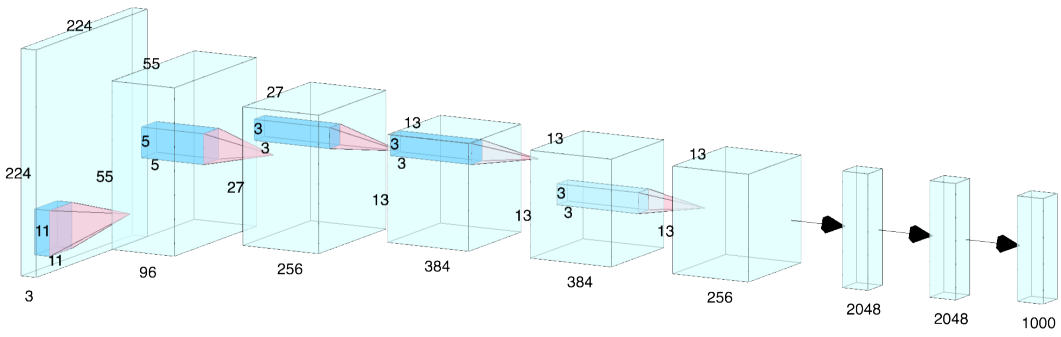


FIGURE 2.6: An example of CNN architecture for image classification [98].

CNN is used as the front end to extract features. Fully connected layers are used at final layers to produce prediction results. However, this also entails a huge parameter space that has to be learned. Fully Convolutional Networks (FCN) [170] aim to tackle this issue. Features at final layers are convolved with a series of filters with such stride values that result in final prediction to be in a vector form. This vector could then be used for regression or classification.

2.5.2 Recurrent Neural Networks

FC-NNs or CNNs are designed to process single images at a time and are unable to track between consecutive samples. Recurrent Neural Networks were developed to deal with sequential data processing. RNNs alter the structure of the cells in the network. Instead of simple multiplication and summation at each cell, a series of primitive calculations are used to introduce memory functionality. Figure 2.7 shows the unfolded structure of the RNN architecture.

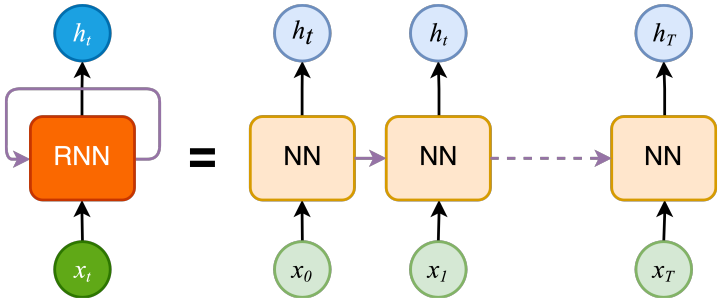


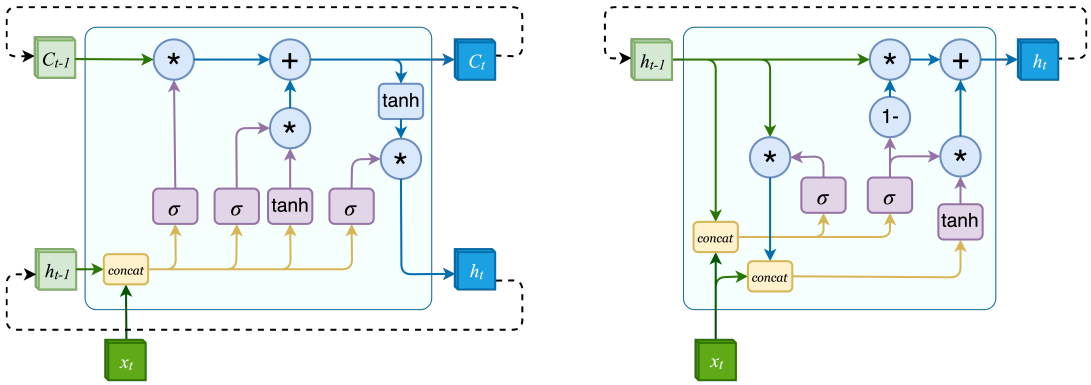
FIGURE 2.7: RNN and unfolded RNN architecture.

Original RNNs were suffering from vanishing and exploding gradient problems during training on longer sequences. To deal with this, Long Short Term Memory (LSTM) networks were introduced in [83]. LSTM is a variation of RNN in which each neuron consists of an input gate, an output gate, and a forget gate. There are various implementations and parameters that need to be carefully tuned while using LSTMs. A comprehensive analysis of these parameters are provided in [25] and various implementations of LSTM networks within different frameworks are compared in [24].

Gated Rectified Units (GRU) [40] have fewer parameters compared to LSTMs as they do not employ an output gate. From a performance point of view, they exhibit similar results in comparison to LSTMs [46].

2.5.3 Encoder-Decoder Networks

An encoder-decoder architecture is used in mapping data points from the input domain to the output domain. This is achieved by projecting input data to a latent space



(a) Long Short Term Memory (LSTM) cell (b) Gated Recurrent Unit (GRU) cell

FIGURE 2.8: Cell structure of LSTM and GRU.

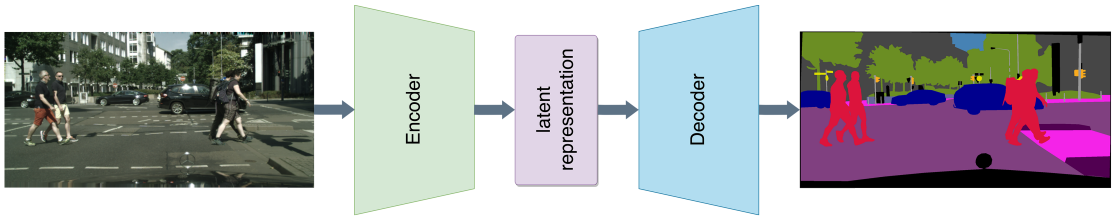


FIGURE 2.9: Encoder-Decoder network architecture for semantic segmentation. Input image and output label are taken from the cityscapes dataset [48].

representation using the encoder network. Using this representation, the decoder network produces the output in the intended domain. These models are commonly used in image-to-image translation [179, 12, 36, 151, 175, 203], compression [54, 38, 6] and natural language processing [21, 45].

A simple diagram of an encoder-decoder network for image segmentation is shown in Figure 2.9. The input to the system is an image and the output is the segmentation map of the scene. Encoder networks mainly rely on convolution operations to reduce feature map dimensions. To rescale the abstracted features back to the target domain, the decoder network employs methods such as bilinear upsampling, bicubic upsampling and transpose convolutions [162].

A variation of encoder-decoder models is termed Autoencoders [72]. The input and output data is set to be the same for this class of models. The latent space is limited to a specific size that usually is smaller than the input domain. The goal in this approach is to learn a latent representation of the input that can later be used by the decoder to regenerate the input data.

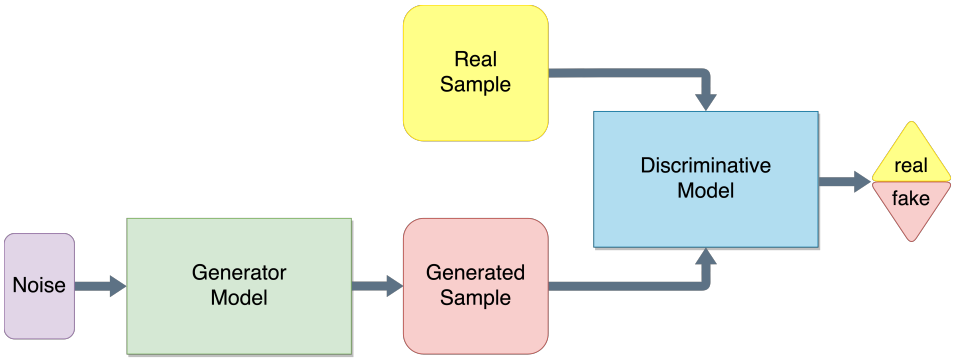


FIGURE 2.10: An overview architecture of Generative Adversarial Networks.

2.5.4 Generative Adversarial Networks

Conventional Generative Adversarial Networks (GANs) [73] consist of two networks. The generator network learns to generate samples in a targeted distribution using noise as the input. The discriminator network inputs samples from the outputs of the generator and also from the real samples of the targeted distribution. The goal of the discriminator network is to classify whether the input is fake and generated by the generator network, or if it is from the set of real samples. However, the generator network aims to capture the targeted distribution and produces samples that are difficult for the discriminator to classify. This results in a minimax game between the generator and the discriminator. Figure 2.10 shows a simple diagram for GANs. The training schedule for this family of models is divided into multiple cycles where the generator is learned and discriminator is frozen, and vice versa. There are multiple variations of GANs [142, 7, 93, 198] that have been used in various applications such as building images from text [193], visual saliency prediction [134], face aging [184], image-to-image translation [43, 181], 3D object generation [189], music generation [58] and many others [177]. Wang *et al*. [183] provide a comprehensive survey of GANs in computer vision.

2.6 Advanced Models for Practical Applications

In this section, some of the advanced architectures designed for specific applications that will be covered in the next chapters are introduced.

2.6.1 Image-to-Image Matching

Conventionally, image-to-image matching is done by extracting local features [113, 15, 5, 119] of both images and employing a matching function [8, 124, 65, 119, 67, 148] to calculate a matching score. This is useful in a range of applications such as stereo matching

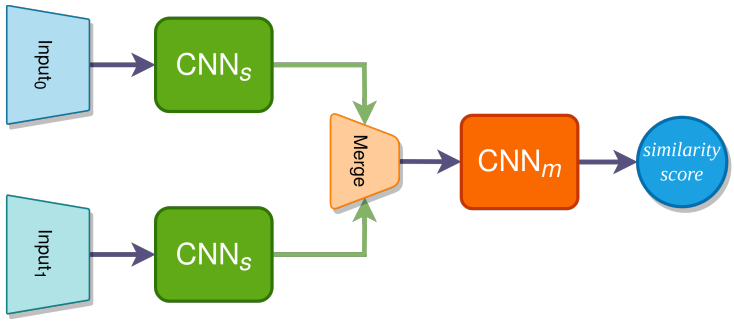


FIGURE 2.11: Architecture of a convolutional siamese model. Feature extraction network CNN_s is applied independently on both of the inputs. The output features of each input are integrated using a merge function and processed through the matching network CNN_m.

and depth estimation [53, 156, 68], flow estimation [112, 66, 118, 87], odometry [11, 96], localization [23, 155] and etc. Bromley *et al*. [26] introduces siamese networks that perform signature verification. In siamese networks, the two inputs are processed through the same network to extract features. A matching function or network is used at the end to extract the similarity score. Figure 2.11 shows a diagram of a convolutional siamese model. Fischer *et al*. [66] proposes an encoder-decoder model for flow estimation named FlowNet. The encoder consists of a siamese model with a correlation layer as the merge function. The decoder is based on successive upconvolutions. Flownet 2.0 [87] expands this idea with a hierarchy of FlowNet [66] models.

2.6.2 Point-Cloud Processing

Data from 3D perception sensors provide rich information regarding the scene geometry. 3D data in its rawest form is commonly represented in a point-cloud format. Meshes and volumetric grids are alternative representations to the point-cloud [192, 204, 182, 117, 191, 208]. These representations are used in order to avoid challenges in the processing of unordered point-cloud data. PointNet [138] provides a solution to this challenge. It uses MLPs to process each point and utilizes a symmetric aggregation function to build permutation invariant global features. PointNet++ [140] extends the PointNet by incorporating a hierarchy of set abstraction layers to capture the local features surrounding each point. The set abstraction consists of PointNet learning layers, farthest point sampling [60] as the sampling function, and k-nearest neigbour or ball query as the grouping method. The architecture of PointNet++ is shown in Figure 2.12. Given the success of the PointNet, many models have built based on this approach [205, 194, 112, 106, 85, 190]. A comprehensive study of deep point-cloud processing methods can be found in [75].

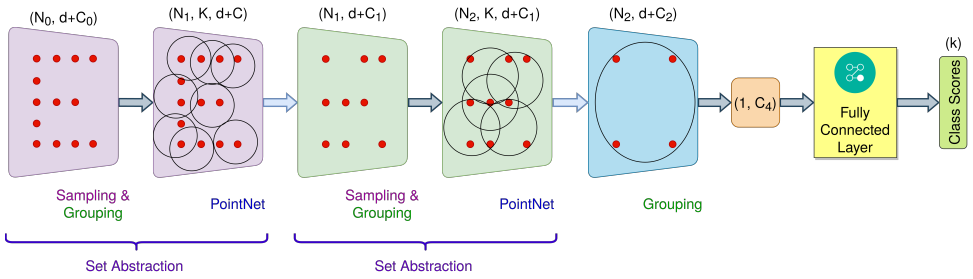


FIGURE 2.12: Architecture of PointNet++ classification network [140]. Number of data points in the feature map is represented by N_i, C_i shows the number of channels, d is the input dimension, K is the number of nearest neighbours to be considered in the set abstraction, and k represents the number of classes.

2.6.3 Semantic Segmentation

Semantic segmentation models are used to label every value in the input with a corresponding class label. Fully Convolutional Networks (FCN) [162] is the first deep learning model to tackle this problem and provide a viable solution. FCN replaces the fully connected layers of the classification networks [98, 164, 171] with convolutional layers, and instead of a classification result, it outputs a segmentation map using transposed convolutions. FCN is a popular approach in image segmentation, however it suffers from limited computational performance and fails to capture holistic features of the scene. Others [104, 160] have tried to integrate conditional random fields (CRFs) with deep models to improve the performance of segmentation. Noh *et al*. [129] use a complete encoder-decoder architecture to predict the segmentation map. SegNet [12] improves on this idea by integrating mid-level feature maps of the encoder with the feature maps of the decoder in a systematic form.

An alternative approach for segmentation is used in the DeepLab family of networks [33, 35, 36]. They propose the usage of Atrous convolutions (dilated convolution) to increase the receptive field of a convolution kernel by applying it on non-adjacent pixels on the input grid. Furthermore, they utilize multiple parallel Atrous convolutions in the last layer to capture local and global features, in what the authors call the Atrous Spatial Pyramid Pooling. In this way, the resolution of the input map is retained, providing a better segmentation performance at the cost of larger computational complexity.

Minaee *et al*. [121] provide an in depth analysis of the recent advances in deep image segmentation.

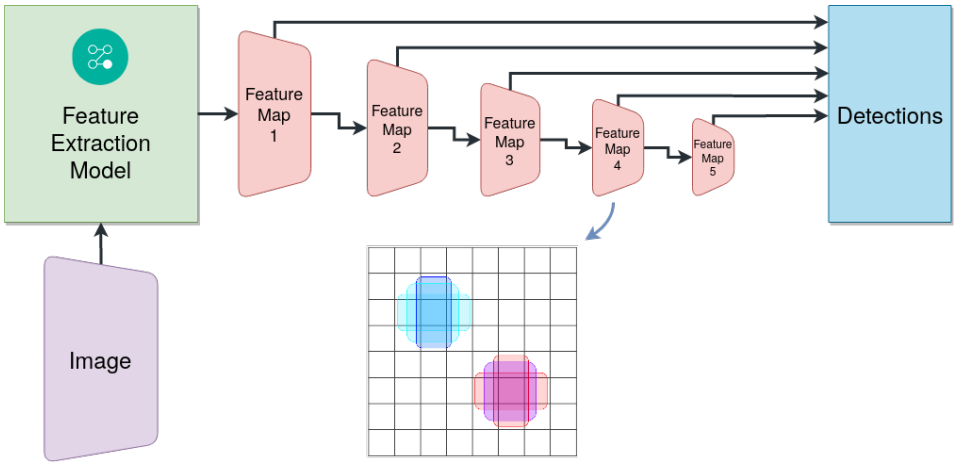


FIGURE 2.13: Architecture of the Single Shot Detector (SSD) model [111]. Each successive feature map has a smaller size and a larger receptive field that enables them to detect larger objects.

2.6.4 Object Detection

Girshick *et al*. [70] propose using Regions from CNN to detect objects. Each region is passed to a pretrained object classification network [98] to extract features. Extracted features are then used with a linear classifier to detect objects. This work is later expanded upon by Spatial Pyramid Pooling Networks [79], Fast RCNN [69] and Faster RCNN [147]. Faster RCNN introduced the Region Proposal Networks and was the first model to train the object detection model in an end-to-end manner. Feature Pyramid Networks (FPN) [108] expand on this idea and use features from various layers to enhance object detection performance. All of the mentioned models are considered to be two stage detection models.

You Only Look Once (YOLO) [144] was the first one stage detection model. It achieved realtime speeds while providing state-of-the-art results. In this approach, CNN is applied on the whole image. Single Shot Detector (SSD) [111] introduced the idea of multi-reference and multi-resolution detection. First, the image is passed through a feature extraction network [84, 164, 170]. Selected feature maps are passed through further convolutional layers to produce extra feature maps of different sizes. Figure 2.13 shows this model. Predefined anchor boxes with various aspect ratios are used to predict the correct bounding boxes on each of the extra feature maps. This ensures that objects of different sizes are detected using different layers. Recent proposals such as YOLO9000 [145], RetinaNet [109], and YOLOv3 [146] provide improvement in the detection performance and computational complexity requirements.

2.7 Libraries

There are various libraries that facilitate implementation of deep learning models. The most common libraries are listed in the Figure 2.14 based on their unique citations in arxiv[1] papers.

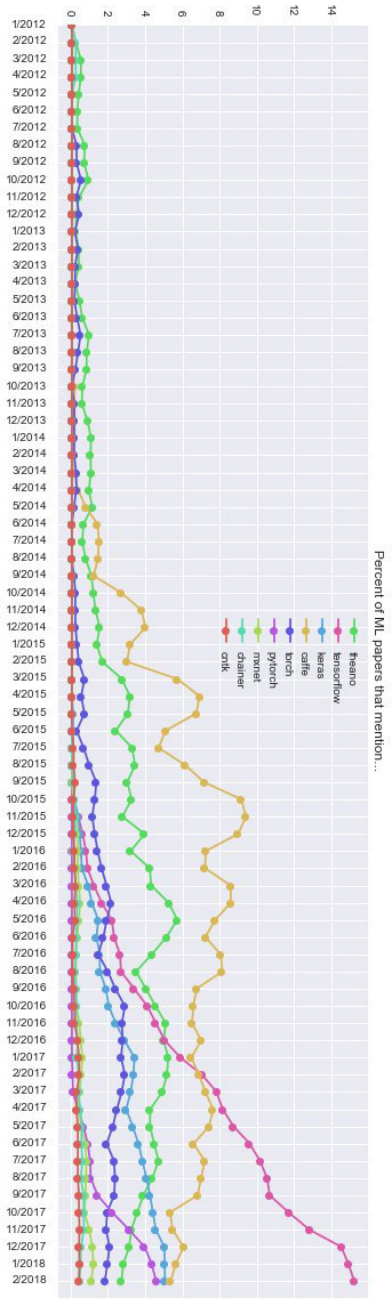


FIGURE 2.14: The trend of using deep learning libraries [92].

[1] arxiv.org

Caffe[2] is one of the early libraries for deep learning that is written in C++ and has a python interface. It is famous for its large model zoo.

PyTorch[3] is introduced by Facebook[4]. PyTorch is based on the Torch library that was originally written in the Lua programming language. Unpopularity of the Lua language kept many developers away from Torch. Later, Torch library was reimplemented with python and gave rise to PyTorch that is well received in the community.

Tensorflow[5]**[2]** is the most famous library for developing deep neural networks. It is created and maintained by Google [6]. Tensorflow provides a python based front end with a significant amount of documentation and various APIs. Apart from Google, there is a huge number of developers and researchers adding new functionalities to this open source library on a daily basis.

[2] caffe.berkeleyvision.org
[3] pytorch.org
[4] facebook.com
[5] tensorflow.org
[6] google.com

Chapter 3

Image Homography Estimation

In this chapter, we introduce a hierarchy of twin convolutional regression networks to estimate the homography between a pair of images. In this framework, networks are stacked sequentially in order to reduce error bounds of the estimate. We show that given the iterative nature of the framework, highly complicated models are not necessarily required, and high performance is achieved via hierarchical arrangement of simple models. Effectiveness of the proposed method is shown through experiments on the MSCOCO dataset.

3.1 Introduction

Using deep methods in autonomous robotics and vehicles is currently focused on sensor data processing [49, 12], planning [197], or end-to-end learning [20]. However, there are still many open problems such as odometry extraction, localization, mapping, or 3D model generation that have not been investigated thoroughly. Improving the effectiveness of deep methods on these tasks is essential.

All of these tasks at some point require a form of matching and regression. We therefore, study here the fundamental task of extracting deformation parameters from two observations. This is the building block of almost all tasks dealing with multiple observations and trying to build models from them. For example, when a camera calibration task is performed by using a planar rectangular grid in front of the camera, one must extract features from them. Then, using a regression module the homography is estimated and subsequently, extrinsic parameters between stereo cameras are calculated [185]. Similar ideas are applied for panoramic image generation [27]. To generate 3D models of the environment, all models use a similar framework to extract, localize and register features from various images to get relative poses information [4]. These methods later are used as building blocks in various fields, such as in autonomous driving, augmented reality, indoor robotic tasks and so on.

In this chapter, we propose the usage of a hierarchy of convolutional networks to estimate the homography between a pair of images. At every convolutional network

FIGURE 3.1: Sample output and the warping process. Source and target images are provided to the system. Due to shrinking error residual after each module, warped image is getting visually more similar to the target at each step.

module, features from each image are extracted independently, but using a shared set of kernels, also known as Siamese network model [26]. Later on in the process, they are merged together to estimate the homography. Sample results of this process are shown in Figure 3.1. Furthermore, we evaluate and compare effects of various training parameters in this context. We show that by only utilizing simple models, the proposed iterative framework can achieve state-of-the-art performance. The proposed method is throughly experimented using *MSCOCO* dataset [110].

3.2 Related Work

The basic approach to tackle a homography estimation is to use two sets of corresponding points in a Direct Linear Transform (DLT) method. However, finding the corresponding set of points from images is not always an easy task. In this regard, there has been a significant amount of research. Features such as SIFT [113] and ORB [152] are used to find the interest points, and employing a matching framework, point correspondences are achieved. Commonly, a Random Sample Consensus (RANSAC) [76] approach is applied on the correspondence set in order to avoid incorrect associations. And, after an iterative optimization process, the best estimate is chosen [27, 125].

One major problem with such methods is their requirements for the hand-crafted features and exhaustive matching step. Deep models automate feature extraction and provide much stronger features than conventional approaches. Their superiority has been shown many times in various tasks [12, 70, 164, 153]. Recently, there have been attempts to address the matching problem with similar models. Flownet [66] targets optical flow estimation by employing a parallel convolutional network model to extract features from each image independently. A correlation layer is used to locally match extracted features against each other and aggregate them with responses. The expanded

feature set is then used in further convolutional layers. Finally a refinement stage consisting of de-convolutions is used to map optical flow estimates back to the original image coordinates. Most recently, Flownet 2.0 [87] was introduced that is using Flownet models as building blocks to create a hierarchical framework to solve the same problem.

Similarly, [148] proposes an optical flow estimation method that employs a deep matching process with convolutions. From each image, patches of size 4×4 are extracted and described based on SIFT descriptors. Then, they are fed into a convolutional layer to produce correlation maps. This process is repeated and a pyramid of responses are created, which are used later to find local maximas and track them through the pyramids to get pixel correspondences.

Our proposed method is vastly inspired by the work of DeTone *et al*. [55], where a deep neural network is devised to tackle the homography estimation task. To be compatible and comparable, we strictly followed their guidelines when developing the model and its benchmark. As they proposed, a homography between two images is defined by relocation of a set of 4 points, also known as 4-point homography. Their model is based on the VGG's architecture [164] with 8 convolutional layers, a pooling layer after each 2 convolutions, and 2 fully connected layers with an L_2 loss function that results from the difference between predicted and true homography values. Their model starts with stacking up images in two channels and processing them together through the network. In contrast, the core of our model targets each input independently from the other. Motivation behind this approach is to have corresponding feature sets that will be merged in later stages, which has been shown in [26, 44] to be more suitable for tasks requiring feature matching.

Most of the existing deep models aim to solve the problem through a single, and in many cases, large model. Independent from how deep or wide the model is, an error margin will always be found, especially in tasks such as regression. This is due to the trade-off imposed on the model to fit the training data and also to be able to generalize. A well learned model should be able to minimize both errors in training and testing sets.

Based on concepts from iterative optimization approaches, boosting, or genetic models, we propose to stack the same model in a hierarchical manner such that the first model takes an image pair and produces an estimation, along with a new image pair. The newly generated image pair has a smaller homography residual, as the first model already estimated part of it. Next, a copy of the same model takes this data in and provides a better approximation with an even smaller error bound. Continuing this process will sequentially reduce error bounds until the error margin is so low that there is nothing else to correct. As we will discuss later, in this situation, the train and test errors start to diverge, which indicates the model has entered the over-fitting region. A similar idea was introduced by Carreira *et al.* in [30] for human pose estimation. In their work, the

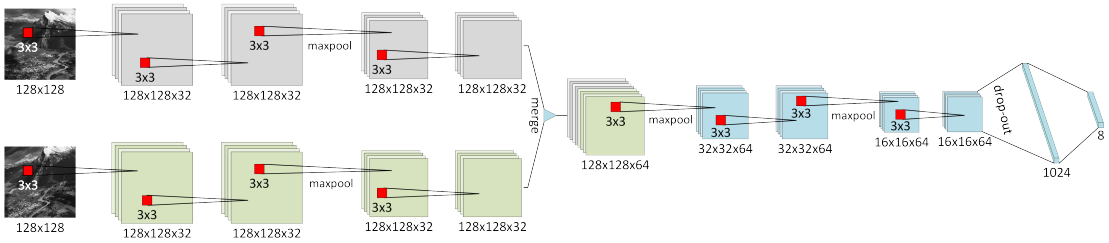


FIGURE 3.2: Twin convolutional neural network architecture used as the core module.

body part point coordinates are estimated using [171] and are iteratively updated to fit the human pose in a single image. In contrast to that work, we are introducing a visual warping step between each iteration to drive the model towards the target transformation, while using a smaller network. To the best of our knowledge, this framework has never been applied to the homography estimation task, which is a natural fit for the problem.

This framework produces mid-level results that are important in many real-world tasks. It provides a way to observe how the system evolves thus preventing false behavior of the components. In addition, coarse-to-fine results could be used by other modules, thus reducing system latency. Modularity of this framework is another benefit. On a case-by-case basis, each model could be retrained, or replaced with a larger or smaller model, to fix the problem specific tasks. Further, training a small network is much faster and less cumbersome than a large one. Based on the required error bounds various combinations of networks can be coupled in the framework. We believe this framework could be applied to any problem that includes regression and/or optimization. In Section 3.7, we will discuss how it can be adapted to the sensor odometry problem. Our model achieves real time performance on consumer available graphics cards that satisfies industrial requirements.

In essence, our proposed model takes the basics of the Siamese network model [44] that are shown to be more effective in extracting similarities between inputs using parallel layers with shared weights. We performed extensive testing and compared performance of multiple parameters for the model training.

3.3 Hierarchical Convolutional Network Model

In this section, we present our network architecture in detail and show how to calculate the four point homography estimate from an image pair. This network is shown in Figure 3.2.

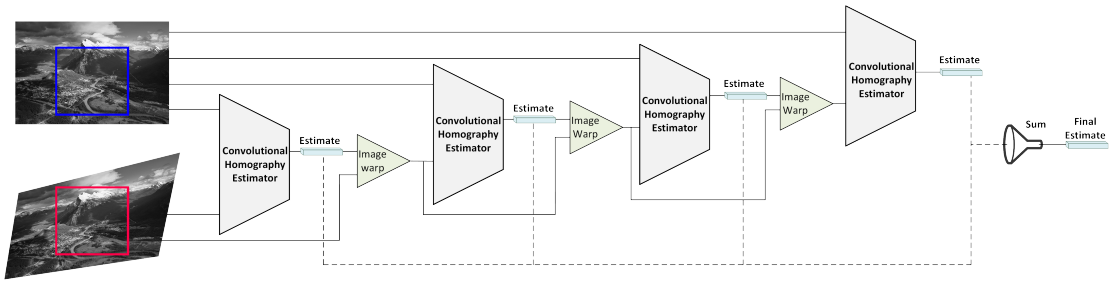


FIGURE 3.3: Hierarchical Model Framework. Each of the convolutional homography estimation modules consists of twin convolutional neural networks that perform homography estimation on an input image pair.

3.3.1 Network Architecture

Similar to [55], the main network in our method consists of 8 convolutional layers followed by two fully connected layers. In contrast to that model, the first 4 layers of our network are designed to process images in parallel. Input images are both normalized and each image is fed to one of the identical parallel layers. A filter with a kernel size of 3×3 is used to create features in 32 dimensions. At the end of parallel convolutional layers, two 32 dimensional feature vectors are concatenated along their third dimension, totaling in a 64 dimensional feature vector. Another 4 convolutional layers are applied on the merged feature map to conclude the convolutional layer. After each 2 convolutions a max pooling layer with strides of 2 is applied on feature output. To avoid over-fitting, the commonly used drop-out [166] scheme with a drop probability of 0.5 is employed prior to passing the output to the fully connected layer. The two fully connected layers have a dimensionality of 1024 and 8 respectively in which the latter is the flattened output homography estimate. Instances are processed in batches of size 64.

Rectified Linear Units (ReLU) are used in each neuron's fire module to add non-linearity to the feature outputs. Both the momentum and Adam optimizers have been tested for the proposed system. For momentum optimizer, as proposed in [55], a momentum value of 0.9 with piecewise constant learning rate of 0.005 with a decay factor of 0.1 is applied in the iterations 30000 and 55000. For Adam optimizer, we kept the learning rate and decay factor the same as for the momentum optimizer with the epsilon set to 0.1. The total number of epochs per model is set to 75000 which roughly equates to 10 rounds over the dataset.

To evaluate and train the network, an L_2-norm on the difference of the target H_i and estimate H_i^* is used as the loss function.

$$loss = \frac{1}{2} \sum ||H_i - H_i^*||^2 \tag{3.1}$$

3.3.2 Hierarchical Model

A hierarchical model arranges neural network modules in a stacked manner and successively reduces estimation error bounds. In each module, an approximate H_i^* of targeted homography H_i is generated. To prepare data for the next module, a new target homography H_i is calculated by simply subtracting the estimate from the processed module's target, and a new image pair (I_i^1, I_i^2) is generated by warping one image in the pair using the new target.

$$\begin{aligned} H_i &= H_{i-1} - H_{i-1}^* \\ (I_i^1, I_i^2) &= (I_{i-1}^1 * H_i, I_{i-1}^2) \end{aligned} \tag{3.2}$$

It is worth noting that the dynamic range of H_i at each iteration is smaller than the previous iteration. In other words, the new target is actually the error residual of estimates calculated in the module.

$$max(H_i) - min(H_i) < max(H_{i-1}) - min(H_{i-1}) \tag{3.3}$$

As explained, each module estimates the residual value of the overall homography. To calculate the final result that can directly transform one image to another, all 4-point estimates of the successive modules are added up together. Early modules tend to have larger values than later modules in the framework, since the residuals have lower dynamic ranges.

$$H^* = \sum_{i=0..n} H_{i-1}^* \tag{3.4}$$

Overall, this approach is similar to an iterative optimization scheme. Each module in this hierarchy is trained on the outputs of the previous module in order to produce a smaller error residual. This is in essence similar to Boosting methods [157] that divide feature space into chunks and assign them to various weak learners. After a learner is done, another weak learner starts learning based on the features assigned to it, and this process iteratively continues until all weak learners eventually create a strong learner. In contrast, our proposed framework does not divide the feature space. Instead, it trains the next model only on the residual of the previous module. Boosting also uses a weighting mechanism. In our model, the weighting scheme can be considered as an implicit function, as each module provides a smaller valued estimate. Figure 3.3 depicts the proposed hierarchical framework.

Warping with predicted homography values for each module results in a visually more similar patch pair. This can be visualized as a geometric morphing process that takes one image and successively makes it resemble the other, shown in Figure 3.1.

One relatively important benefit of the modular design of the hierarchy is that it is not restricted to the use of one specific network architecture. At each level, depending on the error boundaries, data statistics, accuracy and speed requirements, different networks could be used.

3.4 Dataset

We are using the Microsoft Common Objects in Context (*MSCOCO*) 2014 dataset [110]. First, all the images are converted to grayscale and are down-sampled to a resolution of 320×240. To prepare training and test samples, they are randomly divided into two groups [55]. A training set with 77870 and a test set of 5000 base images. Later, five samples from each base image is generated in order to increase the dataset size. To achieve this, five random rectangles of size 128×128, excluding a boundary region of 32 pixels, are chosen from each base image. A random perturbation in the range of 32 pixels is added to each corner point of the rectangles. This provides us with the target 4-point homography values. Target homography is used with the OpenCV library to warp original images. Finally, original corner point coordinates are used within the warped images to extract the warped patches. The pair of chosen patches along with the values of random perturbations (4-point homography) are fed as inputs to the system.

3.5 Implementation

We have realized a Tensorflow[1] [2] implementation of the proposed module. In addition, we have introduced a slightly different implementation of the weight normalization paradigm. To train we have used local machines and Google Cloud Machine Learning Engine [2], but all the tests were performed on a local machine equipped with an Intel Core-i7 CPU at 4.0Ghz, 32GB of memory, and an Nvidia Titan Xp graphics card with 12GB of available memory.

Our tests achieve an average processing speed of 3 milliseconds per network. When the same neural network model is used in each module, the overall computational time is given by,

$$d_e = (l_m + l_w) \times n \tag{3.5}$$

where d_e is the end-to-end delay, l_m the average latency at each module, l_w the overhead of warping to generate a new pair, and n the number of modules used in the framework. The real-time processing speed and architectural flexibility of the proposed model satisfies the requirements of most potential applications.

[1] tensorflow.org
[2] cloud.google.com/ml-engine

Model name	Parallel layer dimensions	Merged layer dimensions
64_dims	32×4	64×4
128_dims	64×4	128×4
256_dims deep	64×4	$128 \times 4 + 256 \times 4$

TABLE 3.1: Model names and parameters.

3.6 Experiments

In this section, we report training and test performance of our method under varying conditions. First, we show how our framework ranks against state-of-the-art approaches. Then, the results of single deep modules are compared. For this purpose, a series of models with different layer widths, as described in table 3.1, are implemented. The performance of the proposed model against occlusions is demonstrated in the third subsection. Then, the effects of various parameters are evaluated. And finally, we discuss how the optimal number of iterative modules is chosen.

3.6.1 Accuracy Results

First, we experimentally compared the corner error of our hierarchical convolutional network with two other approaches. The corner error is achieved by calculating L_2 distance between target and estimate corner locations and averaging them over 4 corners. The approaches used for comparison consist of a traditional one and a convolutional one. The selected traditional approach is based on ORB key-point matching followed by a robust RANSAC homography estimation scheme. The reference deep convolutional approach is the HomographyNet by [55].

Using our network, we report in Figure 3.4 the progressive improvement that results by iteratively stacking several networks, and the final results are shown in Table 3.2. This is compared with the other two approaches. HomographyNet-classification provides the worst results in this case. Its lower performance is attributed to the quantization performed on the range of values to extract bins for each value. Our hierarchical network achieves a drop in pixel error from 12 pixels to less than 4 pixels. The comparable convolutional HomographyNet only achieves an error of 9 pixels.

For the sake of simplicity, we kept the same model at each module of training. However, multiple combinations are also possible, which will be discussed in Section 3.7. As more models are stacked, higher accuracy is achieved. This is due to the fact that each model is trained to reduce error bounds of the previous model. There are downsides to having more models in the stack. One is the introduction of an end-to-end delay. However, the high speed of our network can largely compensate for the introduced delay.

Method	Pixel Error	Error reduction
ORB+RANSAC	11.7	—
HomographyNet[55],	9.2	21.37%
Proposed	**3.91**	**66.58**%

TABLE 3.2: Test Results. Comparison of our method with a traditional feature-based (ORB key-points) robust estimation scheme and homographyNet, both reported in [55], against our proposed approach.

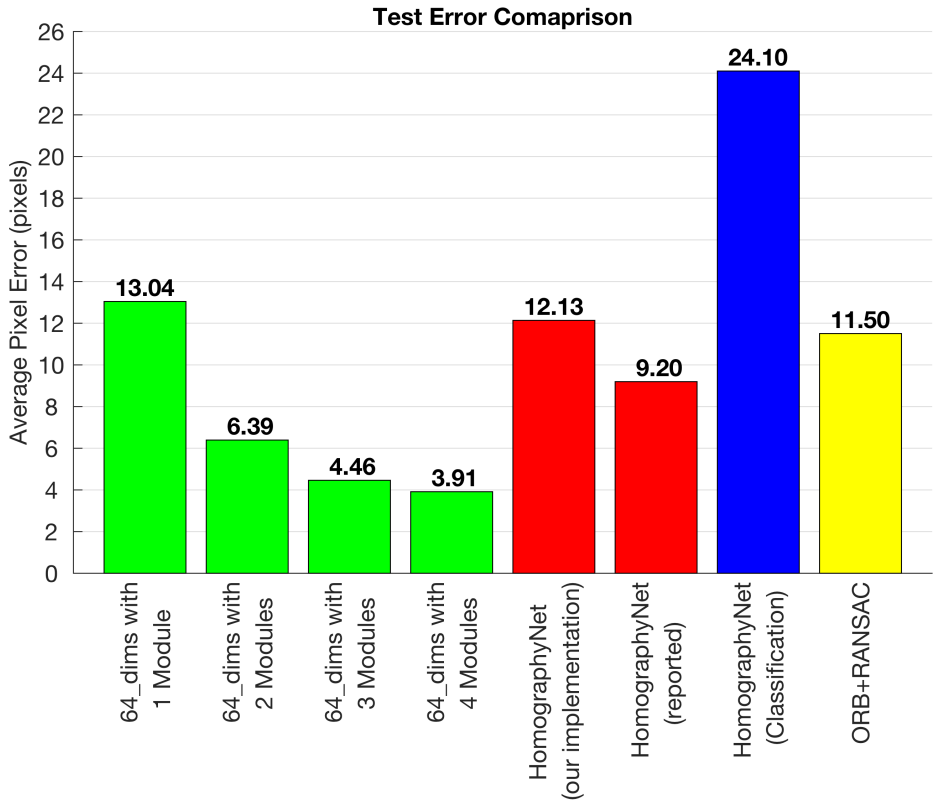


FIGURE 3.4: Test Error. Corner pixel error comparison of various methods.

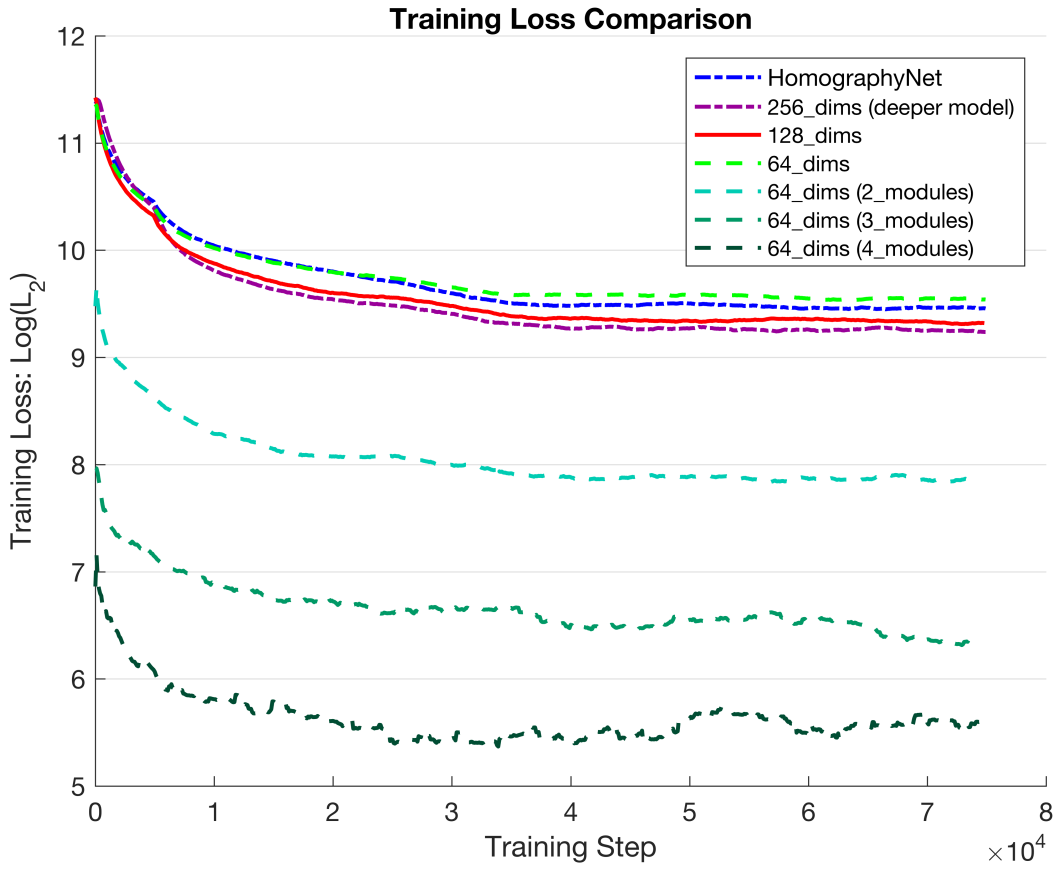


FIGURE 3.5: Training curves showing the learning process for implemented models in $log(L_2)$ scale.

3.6.2 Single Module Comparison

Another conducted experiment targets the width of the network. To address the learning process, we report the L_2 training loss curves using a *logaritmic* scale in Figure 3.5. As expected, the narrower the network, the lower the performance. Using wider network configurations only slightly increases the performance with an additional cost on memory requirement. We have also observed that any network narrower than *64_dims* provides notably worse results, and anything wider than *128_dims* produces a negligible increase in performance, while significantly increasing model size and training time. In fact, stacking a simple module such as *64_dims* twice in the hierarchy is capable of providing significantly better results than any of the other methods.

Train \ Test	No noise	6.25% noise	25% noise	mix of all
No noise	3.91	7.5	12.56	8.21
Mix of all	4.58	6.41	9.04	6.74

TABLE 3.3: Effect of occlusion on the performance. The left column shows the datasets used for training. The pixel-wise corner accuracy of each test dataset is noted underneath the corresponding column header.

3.6.3 Occlusion Analysis

In this section the performance of the proposed system is benchmarked against occlusion. To simulate occlusion, the *Caltech-101* [64] dataset is used. Each image in this dataset is resized to patches of size 32×32 and 64×64. Then, we randomly augment the *MSCOCO* dataset with generated occlusion patches from *Caltech-101* and create three datasets. The first dataset is only augmented with 32×32 patches representing an occlusion ratio of 6.25%. The second one is created using 64×64 patches resulting in an occlusion ratio of 25%. And the final dataset is a fair mix of no-occlusion, 6.25%, and 25% occluded images.

First, we test the model that has never encountered any disturbances during training against the noisy data. Second, we retrain the model with the occlusion dataset and test again for all the scenarios. The results of these tests are shown in Table 3.3. As expected, the model trained with noisy data has a higher generalization power overall.

3.6.4 Model Depth Evaluation

As shown in Figure 3.5, a deeper network increases the performance but only by a small margin, while significantly increasing the model complexity. This is not a satisfactory trade-off. The reason for this is the fact that extracted features are already capturing variations to the full extent as they can. While in the hierarchical usage, the warping function introduces a sort of geometric information that convolutional models have a hard time capturing from a single input set. The fact that more pooling layers are applied leads to the loss of spatial information.

To justify these claims, we have implemented a model with 4 extra convolutional layers. This results in a model with 4 twin layers, 8 convolutional layers with a pooling layer after each 2 convolutional layer, and 2 fully connected layers. A feature dimension size of 256 is assigned to the newly added convolutional layers. Results of this comparison are also shown in Figure 3.5, which confirms our hypothesis stating that more parameters and layers aren't assistive in this specific case. However, when using the second module in the hierarchy, our proposed framework clearly outperforms the competition.

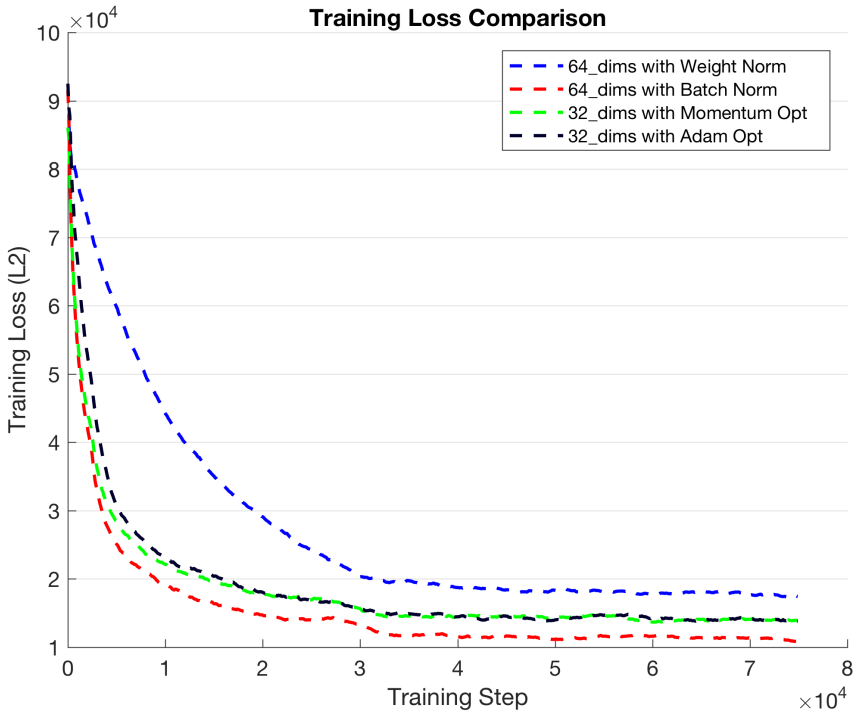


FIGURE 3.6: Parametric Evaluation. Batch normalization is benchmarked against Weight normalization. Effectiveness of batch normalization is shown for homography estimation. Performance of Momentum optimizer and Adam optimizer are very similar.

3.6.5 Regularizers and Optimizers

Batch normalization [88] aims at providing a faster learning convergence by reducing the effect of the covariate shift in the data using statistics from the batches. Weight normalization [154] has also a similar goal, however instead of normalizing the inputs of the layer, normalization is applied on the layer weights. We have compared these two approaches and our results show that batch normalization has better performance for this task. Weight normalization results in a slower convergence and under-fitting in the model. The comparative experiment is shown in Figure 3.6. Another important aspect to note is that having a regularization function is vital in the training phase and failing to use any exhibited devastating effects on the learning performance.

In another experiment, Adam and Momentum optimizers are compared against each other. We have concluded that both methods have a very similar performance in training and their differences are negligible. Results of this comparison are shown in Figure 3.6.

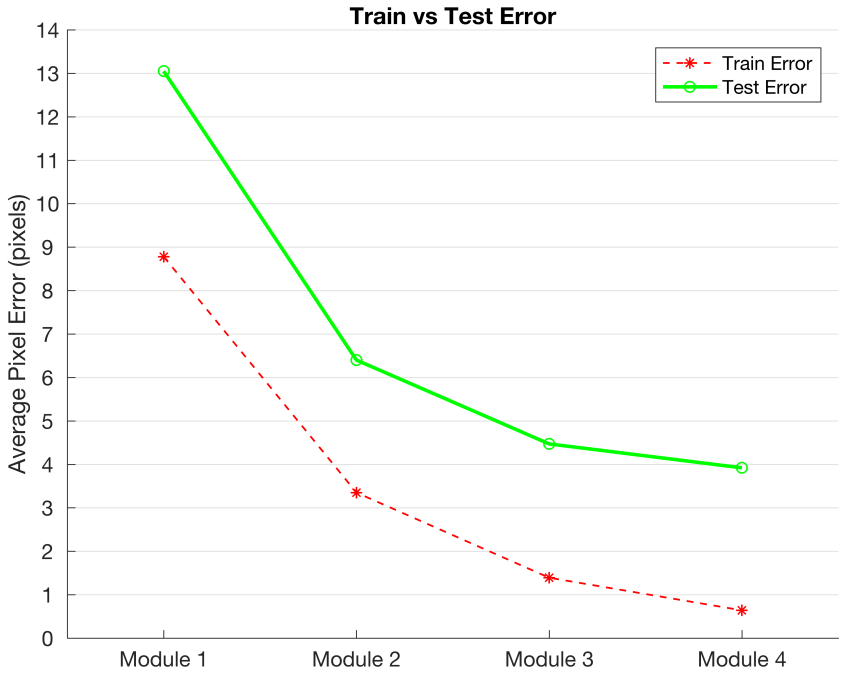


FIGURE 3.7: Hierarchy size evaluation: training error vs. test error.

3.6.6 Optimal Number of Modules

Finding the optimal number of modules to use depends on the complexity of the task and on the complexity of the module to be employed. As it is shown in Figure 3.7, for the 64_*dims* model, after the fourth module, the error residual is too shallow and it is very difficult for the model to learn. This is also shown in Figure 3.5 where the training loss for the fourth module has almost plateaued.

As in all machine learning approaches, a zero error value can not be achieved by chaining many modules together. This is due to the fact that after a few modules, the error residual gets very small and it becomes extensively difficult for it to learn. This results in a test error that starts to diverge from the training error, indicating that an over-fitting phenomenon is happening in the module.

3.7 Conclusion

In this chapter, we have proposed a hierarchical model to use convolutional neural networks in order to target homography estimation. We showed that with our simple hierarchical model, results are significantly better than the state-of-the-art at the time of

this publication. This chapter shows that deep neural networks are capable of effectively learning to regress homography from image pairs. A reduced parameter space entails faster and more manageable implementations. In our architecture, twin modules are unaware of each other until the merge function happens. Adding correlation features between parallel layers could potentially lead to better solutions. In addition to stacking, parallel modules could also be used in this hierarchy. The significance of our results is convincing enough to adapt this approach to further tasks such as odometry estimation, which are currently dominated by conventional approaches. Adapting our model to odometry estimation is ultimately straightforward. However, in order to handle the complexity of odometry, multiple measures must be taken. Augmenting the model to use semantic information could provide robustness against outliers such as dynamic objects. To capture a wider range of transformational cues, a better convolutional model needs to be designed. Finally, to handle the error propagation through consecutive frames, a sequential learning model must be utilized.

Chapter 4

Deep LiDAR Point-Cloud Odometry

In this chapter, we propose a deep model that learns to estimate odometry in driving scenarios using point-cloud data. The proposed deep registration model is used in an iterative order to extract frame-to-frame odometry estimations. Also, a local bundle adjustment variation of this model using LSTM layers is implemented. These two approaches are comprehensively evaluated and are compared against the state-of-the-art.

4.1 Introduction

Autonomous vehicles should be able to operate in known or unknown environments. In order to navigate in these environments, they have to be able to precisely localize themselves. A major issue in localization and mapping is caused by the tight coupling between these modules. We require highly precise maps for localization, and at the same time, accurate localization is required to create precise maps. This inter-dependency has raised interest in such methods that perform both tasks at the same time, termed Simultaneous Localization and Mapping (SLAM).

In an unknown environment, maps are not available to be used as a priori of the environment model. Localization module is needed to infer the position on its own. One way to address this challenge is to estimate the amount of movement in between two individual observations and incrementally calculate the location of the sensor in the coordinates frame of the first observation. This is termed *Odometry* based localization. In this regard, we propose to build a novel method to tackle this problem by relying on the rich sensory data from lidar in an unknown environment.

Various traditional approaches [200, 201, 107, 37, 16] are proposed to perform scan-to-scan matching to extract odometry data. The majority of these models rely on using Iterative Closest Point (ICP) and RANSAC [76] to extract the registration. The majority of odometry estimation approaches utilize a temporal filtering stage that is classified as the Filtering or the Bundle Adjustment.Filtering models summarize the observations in compact representations. This makes them lighter and faster than bundle adjustment

methods that maintain a much larger set of observations and constantly refine their past and current predictions.

Deep neural models have revolutionized many aspects of the computer vision field [164, 78, 172]. Point-cloud processing is one of the challenging scenarios that deep models have a harder time expanding. This is due to the complexity in the scale and the unordered nature of the information representation in point-clouds. [208, 138, 140, 112, 139, 117] have tackled this problem. Many of these approaches are designed to address the classification and segmentation tasks on point-clouds. A comprehensive review on these methods can be found in [75].

Designing a localization system that will function in unknown environments is a much more challenging problem. To tackle this challenge, we adapt the neural models designed to process point-clouds for segmentation and classification to this task. By utilizing their descriptive power, the features required to estimate the odometry are extracted. More specifically, the input point-cloud data is processed using Siamese PointNet++ layers [140]. It follows the same architecture as Flownet3D [112] in order to extract the correlation between feature maps. The point-clouds used in Flownet3D are captured from a single object and consist of fewer points. The point-clouds used for odometry include a much larger number of points, shifts, moving objects and drastic changes in the environment.

Flownet3D uses up-convolutions to extract the 3D flow between two point-clouds. Instead, we pass the features maps to fully connected layers to regress the rotation and translation parameters.

4.2 Problem Statement

There are few ways to address the odometry problem. One is the naive approach of processing two consecutive frames and estimating in-between transformation parameters. This family of methods is mainly used in registration problems. Filtering methods complement the model by summarizing the prior observations and using them in calculating the next odometry estimate. Another way to approach the odometry problem is to actively maintain a list of features from prior observations. With each new observation, the optimization process is repeated. The feature list and previous estimations are refined, and the new odometry parameters are estimated. This is referred to as bundle adjustment.

The error in odometry has an additive characteristic as shown in Figure 4.1. A small error at the beginning of the sequence has a much worse effect in comparison to a larger error at the end of the sequence. The main reason for better performance of the bundle adjustment techniques in comparison to the filtering approaches is due to the fact that

bundle adjustment methods can refine their previous estimates. However, they require much larger memory space and computational requirements.

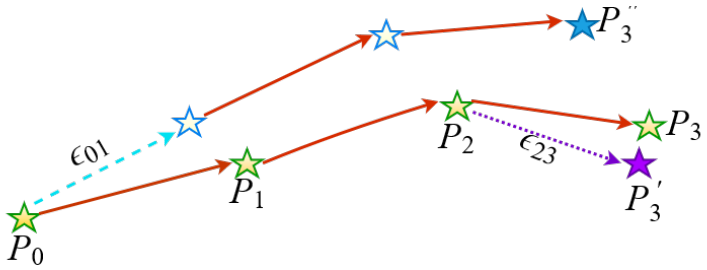


FIGURE 4.1: The additive nature of error in odometry. ϵ_{01} and ϵ_{23} are the same amount of estimation errors, while only occurring in different steps of the sequence. Earlier errors add up to larger trace disparities at the end of the sequence.

In this chapter, we utilize the naive approach to predict a transformation from one point-cloud to the other. While it is possible to perform bundle adjustment with deep neural models [174], using the naive approach with filtering has the benefit of limiting memory requirements.

Furthermore, there is a severe imbalance in the odometry data that requires additional attention during data pre-processing and model design that are explained in sections 4.4 and 5.4.1.

4.3 Related Work

Traditional visual localization methods such as LSD-SLAM [62], ORB-SLAM [125], DTAM [127], FAB-MAP 2.0 [52] mainly rely on local features such as SIFT [113], ORB [152], SURF [15] to detect the keypoints on camera images and track them through multiple frames. Their lidar based counterparts, LOAM [200], VLOAM [201], LOAM_LIVOX [107], SLOAM [37] utilize a similar processing framework with point-cloud data. The capability of these models are always limited by the repeatability of the hand-crafted features in consecutive frames.

One major issue in the odometry challenge is to solve the *data association* or *registration* problem. Descriptor extraction from detected key-points facilitates a way of achieving this. However, as the descriptors can be confused, a RANSAC [76] like system is used to rule out the outliers. After feature matching and outlier removal, the transformation is estimated. To generate a more robust estimation through odometry, a temporal smoothing technique is usually employed [167].

Zhang and Singh [200] introduce Lidar based Odometry and Mapping (LOAM), which is one of the most prominent works in this field. LOAM extracts key-points from

lidar point-clouds. These features are then used to build a voxel grid-based map. They dynamically switch between frame-to-model and frame-to-frame operation to simultaneously estimate odometry and build a map. Chen *et al*. [37] build on the idea of LOAM by replacing the key-points with semantic objects and, Lin and Zhang [107] introduce implementation optimization and adaption to solid-state lidars that have smaller field of view.

It is shown that the use of learned features provides better results in computer vision tasks in comparison to hand-crafted ones. Following the revolution of deep learning in image processing, Chen *et al*. [39] extract deep learned features from images that are later used to perform the place recognition task. One of the early networks for camera pose estimation is defined in [94] and is termed PoseNet. It estimates the camera re-localization parameters in 6 Degrees-of-Freedom (6-DoF) using a single image. To achieve this, it uses a deep convolutional neural network and trains to regress the pose in an end-to-end basis. Their method achieves translation accuracy of 2 meters and angular accuracy of 3 deg. In contrast with traditional methods that rely on Bag of Words (BoW) [65, 100], this method only requires the network weights and is highly scalable for place recognition. However, as the network weights represent a map, each new location will require a new training. Brahmbhatt *et al*. [23] increase the performance of the PoseNet by replacing the Euler angles with the log of unit-quaternions and incorporating odometry results from pre-existing methods. This parametrization of rotation only requires 3 values instead of 4 parameters of quaternions and it avoids over parameterization. Fusion of visual odometry estimates with estimated pose further reduces the final error. In contrary, Cho *et al*. [41] show that Euler angles are more stable than quaternion based loss in their study. Sattler *et al*. [155] provide a comparison of visual localization methods on multiple outdoor datasets with variable environmental conditions.

There is an increasing interest in recent years to solve the odometry problem with deep learning models. One of the first works that directly tackles the visual odometry challenge through an end-to-end approach is proposed by [180]. Odometry is estimated by utilizing a $9-$ *layered* convolutional neural network with two LSTM layers at the end. Yang *et al*. [195] use a monocular image to extract depth and then utilizes it in a classical state estimation framework of [61] to estimate the odometry. Chen *et al*. [32] combine the classical state estimation models with deep neural networks. An LSTM-based temporal model is used to update the prediction step parameters for the Kalman filter. This way, hand-crafted kinematics models are replaced with a learning one. In the update step, deep features extracted from a camera based encoder network are used as the observations to correct the latent system state.

Processing point-cloud data is very different than processing images. Special tricks are required to apply convolutional networks on this data. Li *et al*. [105] build depth

maps from lidar point-clouds and uses it to extract surface normals. Using multi-task learning, it tries to simultaneously estimate odometry and build attention masks for geometrically consistent locations through a Siamese model. Their model consumes the sequential lidar point-clouds and outputs the 6-DoF transformation parameters. Using their combined loss function, they achieve comparable results to traditional approaches. This method does not employ any temporal filtering strategies. Cho *et al*. [41] utilize two siamese networks; one with surface normals and the other with vertices. Features extracted from both Siamese branches are summed and passed to a odometry extraction network.

The deep neural models that take raw point-clouds as input are categorized in two groups: *Voxel-based* and *Point-based* methods. A voxel is a 3D partition in the 3D point-cloud space. The point-cloud is sub-sampled into multiple voxels and points inside each voxel are treated as a connected sub-group. The advantage of voxel representation is the inherent spatial coherence that it embodies. This makes the application of 3D filters on the data easier. However, the downside is caused by the ambiguity and challenges of the voxelization process. The size and the location of each voxel is a parameter that carries a huge effect on the performance of the system. Furthermore, the majority of the voxels in 3D space contain almost no observations, and only consume valuable compute power. Zhou and Tuzel [208] introduce VoxelNet that describes points in each voxel by analyzing their inter-point interactions through a set of point-wise and locally aggregated features. The objective of this method is to find 3D objects in the scene.

Unlike the volumetric models, point-based approaches directly consume the points in the cloud. These methods rely on local region extraction techniques and symmetric functions to describe the selected points in each region. Qi *et al*. [138] introduce the idea of approximating a symmetric function using a multi-layer perceptron in order to process unordered point-cloud data. Convolutional filters are used on these features to perform 3D shape classification and object part segmentation. This work is expanded by PointNet++ [140] to better extract features from local structures. This is achieved by the usage of iterative farthest point sampling and grouping of the unordered points to hierarchically reduce their number and only maintain a more abstract representation of the original set. Ben-Shabat *et al*. [17] expand on the idea of using symmetric functions and proposes 3D Modified Fisher vector representation for semantic understanding. They design a discrete grid structure and use it for continuous generalization with Fisher vectors.

A weakly supervised deep learning model architecture for point-cloud registration is proposed by [196]. Ground truth annotations are automatically generated by using GPS and Inertial Navigation System (INS) data. Similar to our work, they use PointNet [138] as their backbone in a Siamese architecture [26]. The employed attention module [128] targets selection of relevant features and discards the less relevant ones. The

extracted correspondence between them are later used to estimate the transformation parameters between point-clouds. Their model does not generalize as well as state-of-the-art on noisy point-clouds.

Any deep learning approach that uses the Siamese model for odometry relies on a matching layer that could be implemented in various forms. Revaud *et al*. [148] introduce a new layer that hierarchically extracts the image features for dense matching that is used for flow estimation. Flownet [66, 87] convolves features of one Siamese branch against the other and uses the results to explain the scene flow.

At the time of writing this chapter, there are not many datasets available for the localization of autonomous vehicles. The KITTI dataset [68] is the most famous autonomous driving dataset. It provides high resolution stereo images, lidar scans, D-GPS trace, IMU data, along with the calibration parameters for each sensor, and sensor-to-sensor. System observations are collected at 10 *fps* over a path of 39.2 *km* and are provided with annotations. We mainly rely on this dataset in our experiments. It is worth to mention that there were valuable recent additions [28, 168].

4.4 Data Pre-processing

4.4.1 Label Extraction

The KITTI dataset employs global pose coordinates on the local frame. Pose information is provided from the view of the first frame as the center of the coordinate system. These, however, are not suitable labels for the frame-to-frame odometry estimation task. Frame-to-frame pose transformations are achieved through the following formula:

$$\begin{aligned} X_{i+1} &= T_{i,i+1} \cdot X_i \\ T_{i,i+1} &= G_{0,i+1} \cdot G_{0,i}^{-1} \end{aligned} \tag{4.1}$$

$T_{i,i+1}$ is the local transformation between two coordinate centers X_i and X_{i+1}. $G_{0,i}$ and $G_{0,i+1}$ are the global transformations from the first frame (center of the global coordinate frame) to the frames i and $i+1$. This is also represented in Figure 4.2.

We find the minimum and maximum for each of these parameters and use it to normalize the labels in the range $[0, 1]$.

4.4.2 Point-Cloud Sampling

The point-clouds in the KITTI dataset consist of $100k$ points per frame. This is a huge set of points. Given that lidar observations become less reliable at longer distances, we remove any point farther than $50m$ from the center in the x and z dimensions. There is still a large number of points remaining after this step. To reduce these points, we use

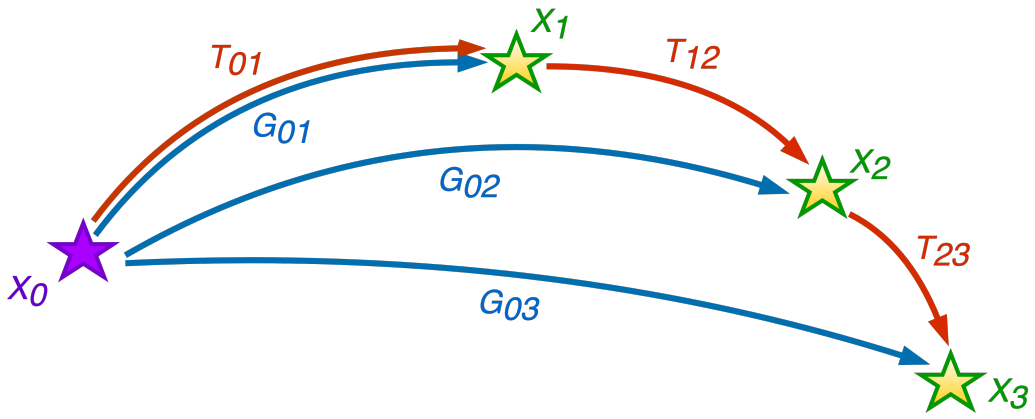


FIGURE 4.2: Visualization of the sequential local pose transformations (Red) and global poses (Blue).

the farthest point sampling strategy [60] to sample only $12k$ and $6k$ points that are later used in our experiments.

Point-clouds collected in driving environments usually contain a large amount of ground points. Ground points are not useful to extract motion information and are usually discarded as a pre-processing step [178, 56, 112]. However, ground as a flat surface can provide cues regarding the pitch and roll orientations of the vehicle, which is valuable for the estimation of 6-DoF odometry. This entails that sampling ground and non-ground points with different proportions can provide the needed information to balance the trade-off between the size of the data and the accuracy of the system. We choose a $75\% - 25\%$ sampling ratio for non-ground and ground points.

4.4.3 Dataset Augmentation

Before tackling the problem, the input data needs to be analyzed thoroughly. One of major issues with using deep learning models for odometry results from the nature of the labels that are highly imbalanced. The majority of roads are designed as straight lines to maximize the efficiency of the transportation system as a whole. Only a few turns are made in long drives from a start point to a destination. The amount of time spent for rapid acceleration and deceleration is usually much lower compared to cruising. This translates to a huge imbalance in the dataset. Most of the transformations between consecutive frames are centered around a specific average. There are a few outliers that define major changes in direction and speed. However, they constitute such a small ratio that presenting a network with this data does not result in good generalization for these outlier cases. Deep learning models are highly dependent on the datasets to be as complete as possible, which is not the case in these scenarios.

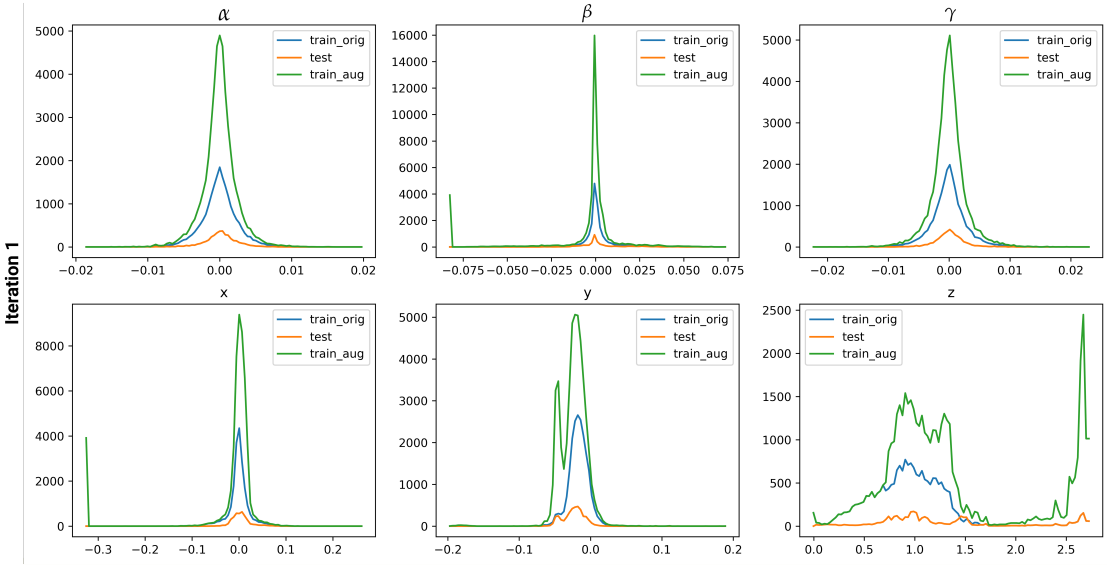


FIGURE 4.3: Histograms of the consecutive transformation parameters before and after data augmentation.

There are techniques on the model side as well as on the dataset side to address this issue. On the dataset side, one approach is to under-sample the abundant transformations to have a better distribution of data points. However, this will result in removal of dynamic noise information. Even though the transformations are still very similar, there are dynamic objects around the vehicle that provide valuable information to the model in understanding the objectives of motion. Another approach is to over-sample the minority labels. This way, more data points are generated. Models can use this to learn the dynamics of the environment while assigning the required attention to minority transformations.

In 6-DoF motion, there are three rotation and three translation parameters that need to be estimated. Rotation components include α, β, and γ as the pitch, yaw, and roll, and translation components consist of motion in x, y, and z axes. There are various methods that perform over-sampling - e.g. repetition, or Synthetic Minority Over-Sampling Technique [31].

Another alternative approach to augmenting data is to add skip steps. This way, it is possible to add the odometry between frames i and $i + N$ to the training set. This results in coverage of a larger range of motion. However, as this augmentation will result in a similar amount of imbalance, we decided not to include it in our augmentation framework.

In order to achieve a more balanced dataset, we use the repetition method based on the amount of divergence from the average odometry value. To augment the dataset, we first calculate the average μ and standard deviation σ for the rotation and translation

component norms individually. The reason for keeping this step separate is the fact that these values are in different ranges, and there is a correlation in between them. Usually, when the vehicle is moving fast, rotation components (especially the yaw component) are very small. When the vehicle is turning, the yaw component takes a larger value, but the translation components are smaller. To decide on which samples to use, we follow 3 rules:

- Translation: if $X_t < |\mu_t - c\sigma_t|$ then augment with X.
- Rotation: if $X_r < |\mu_r - c\sigma_r|$ then augment with X.
- If both translation and rotation rules are satisfied perform another augmentation run with X.

Rotation and translation parameters are respectively shown using subscripts r and t on the transformation X. c is the ratio that controls the distance threshold. In our experiments a is set to 1.

Once a point pair is chosen to be repeated, the number required copies is calculated using the following formula.

$$N_x = \lceil 2^{\frac{x-\mu}{c\cdot\sigma}} \cdot \frac{1}{D} \rceil \tag{4.2}$$

D is a divisor value that explicitly controls the magnitude of the repetitions and is set to 4. x is the rotation norm or the translation norm for the transformation X. μ is the average value and σ is the standard deviation that are acquired from the dataset. Finally, N_X represents the total number of repetitions for transformation X.

In this way, we add approximately 10000 samples for rotation, 6000 samples for translation, and 2000 samples for both.

In the majority of driving scenarios, the cars are in motion and are seldom stopped. To address this, we repeat the identity transformation with a random probability of 10% on the dataset. This results in the addition of approximately 2000 samples to the training set.

Figure 4.3 compares the distribution of data points before and after augmentation. The imbalance in specific components is not completely removed, but is less than in the original model. This is especially the case for yaw (β) and z components that have a major effect on the accuracy of odometry. It is worth mentioning that, as all these components are correlated to each other, completely removing the imbalance only using this data is an impossibly challenging task.

4.5 Model Architecture

4.5.1 Proposed Core Model

We rely on PointNet++ layers [140] to build our model. Similar to their model, we propose a Siamese network that is able to regress the transformation between two point-clouds. Instead of passing the whole point-cloud to a PointNet feature extraction layer, we divide the inputs into two groups; one for ground points, and the other for non-ground points. For ground points, we only use a single PointNet layer with a grouping distance threshold of 4 that outputs 400 points along with their descriptors. Descriptors are generated using 3 consecutive multi-layer perceptrons (MLP) of size $(64, 96, 128)$. For non-ground points, there are two layers that subsequently use grouping distances of 0.5 and 1. 1500 points are produced in the first layer and they are summarized to 800 points in the second layer. Both of these layers use 3 MLPs where the first one consists of $(64, 80, 96)$ and the second one has layers of size $(112, 128, 128)$. The distance metric used to group ground points is larger than the non-ground ones. This is due to the harsher sampling performed on the ground points that has resulted in larger distances between the points.

As the PointNet++ layers use farthest point sampling internally, we keep the ground and non-ground features separate for the feature extraction layer. The final outputs of ground and non-ground segments both have a dimensionality of 128. Both feature maps are concatenated along the points dimension building a feature map of size 1200×128. At this stage, features from each frame are passed to the flow embedding layer of [112]. We use the *cosine* distance metric to correlate the features of each frame to each other. For further future extraction in this layer, we use the MLP with $(128, 128, 128)$ width. Through some experimentation, we found using the nearest neighbor with $k = 10$ gave the best results at this step. The final output shape of this layer is 1200×128.

Once the embedding between two frames is calculated, we run another feature extraction layer with the radius of 1 and MLPs of size $128, 96, 64$. This layer is also responsible for reducing the number of feature points that results in a feature map of shape 300×64. All of the layers up to this stage include batch normalization [88] after each convolution.

The resulting 2D feature map is flattened and a drop-out layer with a keep probability of 0.6 is applied on it. In order to extract the rotation and translation parameters, we use two independent fully connected layers of width 128 and 3.

The final outputs of size 3 are concatenated to build the 6-DoF Euler transformation parameters. Figure 4.4 shows the architecture of this network.

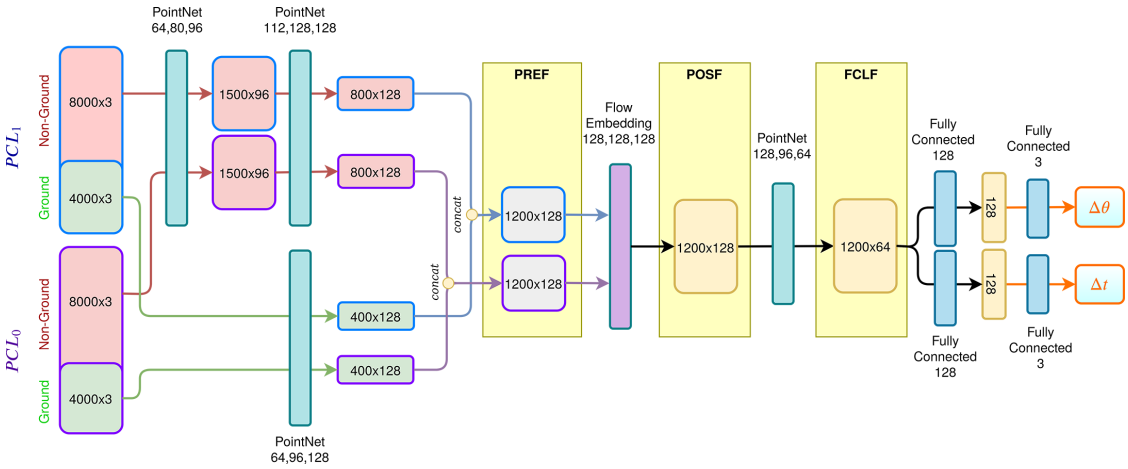


FIGURE 4.4: Proposed core registration model architecture to process two consecutive frames and extract transformation parameters in between them.

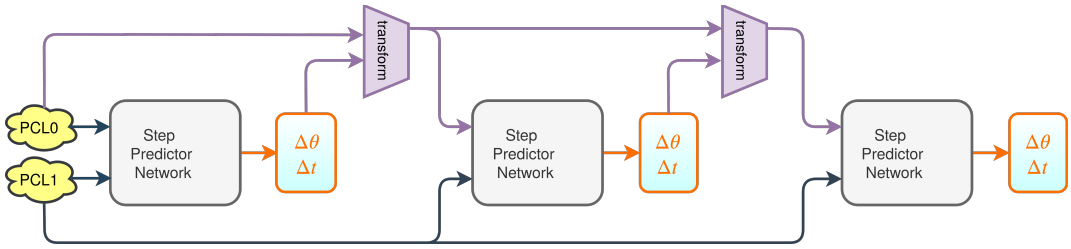


FIGURE 4.5: Progressive Prediction. In the first iteration, $\Delta\theta$ and Δt is estimated using PCL_0 and PCL_1. The source point-cloud PCL_0 is transformed to $PCL_{0'}$. The residual transformation between PCL_1 and $X_{0'}$ is calculated and used to estimate the next iteration of the model.

4.5.2 Hierarchical Registration Model

In traditional literature (optimization or RANSAC models) [53, 96, 11], an odometry prediction is refined quickly through multiple iterations. We argue that the same could be applied in the case of deep learning-based odometry estimation models. Inspired by the hierarchical homography network [132], we use multiple layers of the same network to train on the residuals of previous predictions. In this way, each network reduces the dynamic range of error from the previous models and successively achieves a better result. Figure 4.5 shows this process.

The higher accuracy comes with the cost of increased train and test times. The key element in this approach is to keep the computational complexity as low as possible for each module in order to satisfy real-time processing requirements for the odometry task.

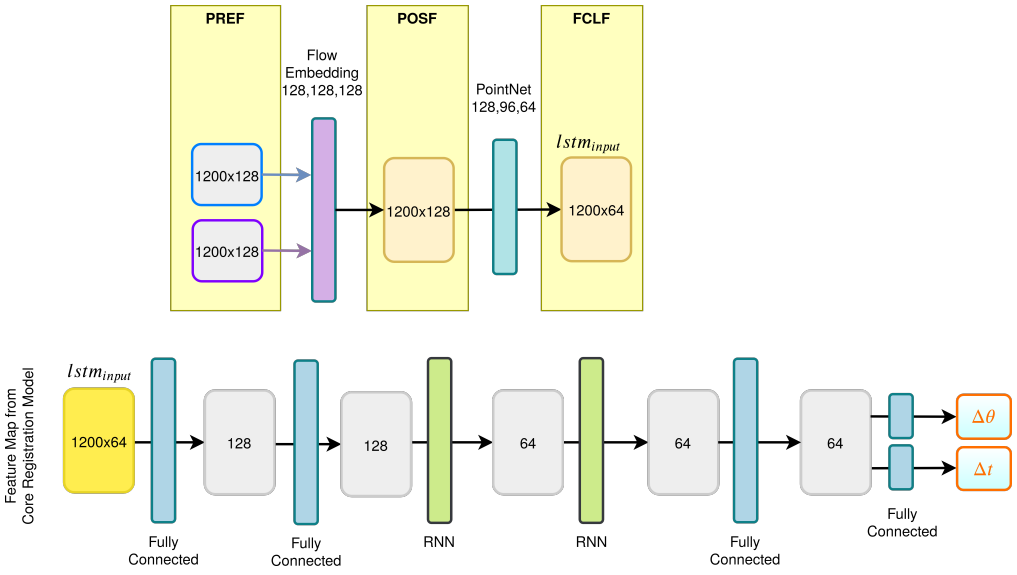


FIGURE 4.6: Temporal model along with the three different input levels. FCLF is the natural input state of the temporal model. PREF and POSF require additional layers from the registration model to be included and re-learned inside the temporal model.

4.5.3 Temporal Filtering

The goal of the odometry model is to smooth the effect of errors caused by the registration network. Our proposed model requires a large amount of memory due to its mid-level feature representation. Adding the memory requirements that the temporal model imposes, training quickly becomes a challenge for the system. Furthermore, training the core registration network is already a difficult task. Extra parametrization from the LSTM model makes an already difficult task, an even more challenging one.

To alleviate these issues we propose a two-stage training approach that breaks the initial feature extractor and the temporal filter into two disjoint models. Once the core registration network is trained, mid-level features are extracted and used as inputs to train the temporal model.

The base of our proposed temporal model consists of two fully connected layers of width 64 and 128. It is followed by two bidirectional LSTM with 64 hidden cells with a *soft_sign* activation function. The output of the LSTM is fed to a fully connected layer with a size of 64. Finally, in the output layer, we have two fully connected networks with 3 nodes each that estimate the rotation and translation, individually. We employ drop-out and batch-normalization after each layer. For the LSTM model, a temporal window of 5 frames is utilized. This is shown in Figure 4.6.

4.5.4 Loss Function

As explained in Section 4.4, the odometry labels suffer from data imbalance problem. The majority of the instances in the dataset follow a straight line, with only a few of them constituting the turns. To tackle these challenges, we incorporate measures in our loss function.

Using the naive *L2-norm* diminishes the effect of instances with larger errors, as most of the batches result in smaller errors. It is required to make the model more sensitive towards larger errors. This is achieved through the use of online-hard-example-mining (OHEM) loss [163]. In OHEM, only the top k samples with highest loss values are used to calculate the final loss. However, once this value is set it does not change during the training. This could result in a training session that has high fluctuations in loss values. To address this, we implement an adaptive version of OHEM loss, that increases the number of top k after certain epochs. The model initially focuses on the hardest examples, before its attention shifts towards all of the examples. This provides the hardest examples that are less frequent with a chance to drive the network towards the global minimum.

Another aspect of the loss function is to analyze the effects of errors in each component of the label. Rotation and translation components are separately extracted from the model. To enable the model's capability of adapting to each component, instead of using the naive approach, we employ a weighting mechanism. To introduce uncertainty in our loss function, we consider a normal distribution around the labels. The probability of a correct regression is shown in 4.3.

$$P(x|W) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{\|x - \hat{x}\|^2}{\sigma}) \tag{4.3}$$

Where W is our observation, $\hat{x}$ is the prediction. The log of this function is calculated, and it is rephrased as a minimization problem. In odometry, there are 6 parameters to be learnt that represent two components. We use the same weight for parameters related to each component. Hence, the equation is expanded to accommodate for this requirement.

$$-\log P(x_r, x_t|W) = \log \sigma_r + \log \sigma_t + \frac{1}{\sigma_r\sqrt{2\pi}}(\frac{\|x_r - \hat{x_r}\|^2}{\sigma_r}) + \frac{1}{\sigma_t\sqrt{2\pi}}(\frac{\|x_t - \hat{x_t}\|^2}{\sigma_t}) \tag{4.4}$$

x_t and x_r are predicted translation and rotation variables. In order to practically implement this loss function, we need to ensure that the learned confidence σ that represents the uncertainty of the task has a positive value. To achieve this, we replace the σ

with $\exp(w)$ that ensures a positive value for this parameter. Further, the constant coefficients of this equation are removed. This results in the following final loss function.

$$l_w = \exp(-w_r)\|x_r - \hat{x}_r\| + w_r + \exp(-w_t)\|x_t - \hat{x}_t\| + w_t \tag{4.5}$$

$\hat{x}_t$ and $\hat{x}_r$ define the ground-truth. w_r and w_t are the trainable weighting parameters, and l_w defines the final loss.

4.6 Experiments

Estimating 6-DoF odometry using point-clouds is a challenging task. All the points are down sampled as described in section 4.4. We first use the registration model to extract the transformation between two frames. This could be repeated for all the point-cloud pairs to generate the odometry trace. However, the additive nature of the noise quickly drifts the trace away from the ground truth as explained in Section 4.2. Due to this fact, there is no single metric that can explain all the various failures in the system. We evaluate various aspects of the model using KITTI odometry metrics:

- Absolute Trajectory Error (ATE) measures the difference between estimated and ground truth trajectory points. ATE is measured in meters scale.
- Sequence Translation Drift Percentage and Sequence Rotation error per 100 meters. For each sequence, the ratio of translation error to the total amount of translation is used to calculate the drift percentage.
- Relative Rotation Error (RRE) and Relative Translation Error (RTE). RRE is defined as the sum of the rotation error in consecutive frames and is measured in degrees. RTE is the sum of the translation error in consecutive frames and is measured in meters.

RTE and RRE are the most important metrics as they evaluate the effectiveness of the model for frame-to-frame estimation, independent of the trajectory. This is the main target that the model is trained for.

We follow the standard division of training and test data in the literature [41, 105, 200]. Sequences $00 - 08$ are used as the training set and sequences $09 - 10$ are used as the test data.

4.6.1 Hierarchical Model

In this section we evaluate the performance of the hierarchical model acronymed as *8k4k_comp*. Figure 4.7 shows the trajectories of various training iterations and Figure 4.8

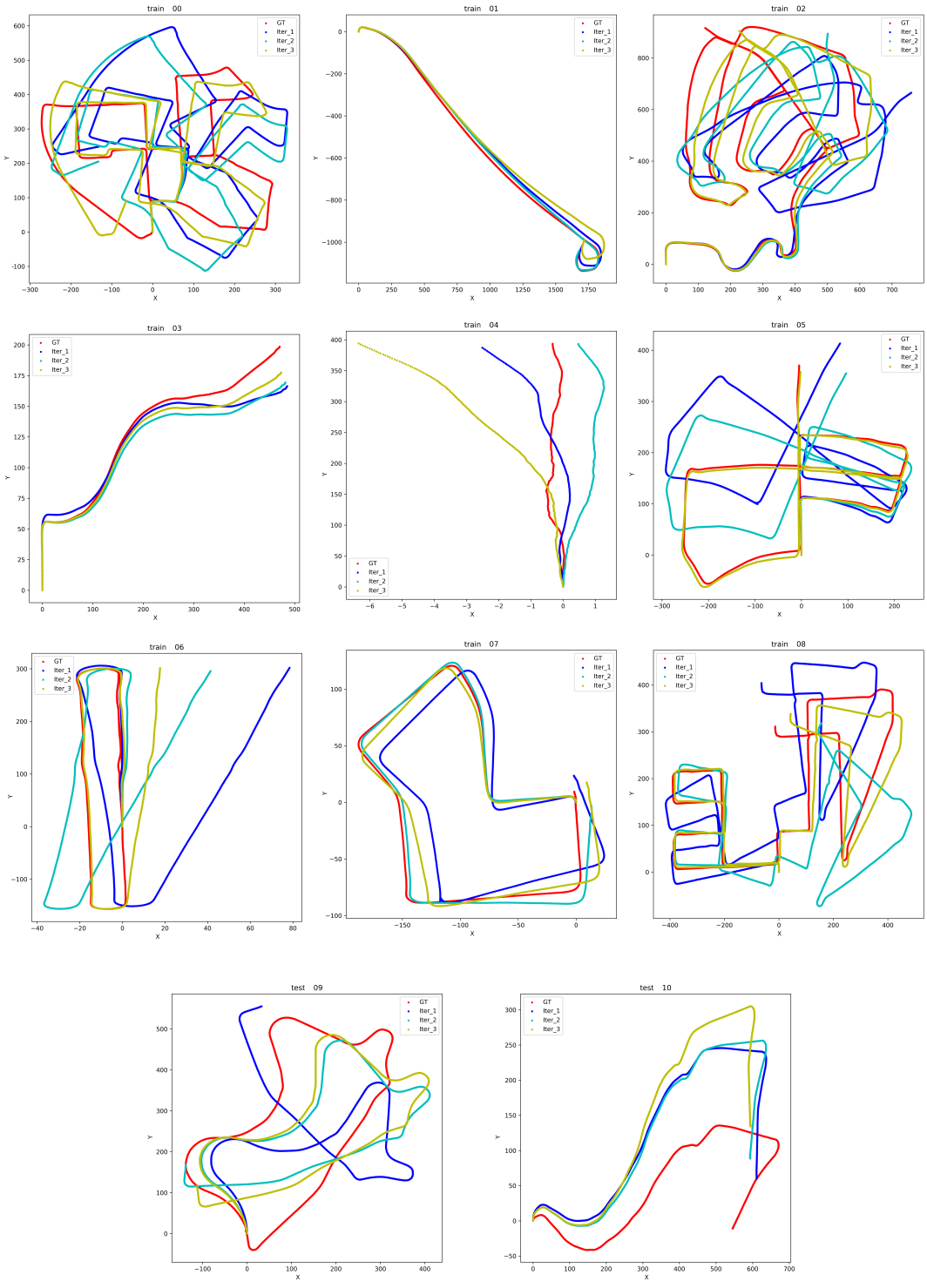


FIGURE 4.7: Estimated odometry trace for KITTI sequences using *8k4k_comp* model. Sequences 00-08 are used for training, and sequences 09-10 are used as test data for the model.

presents their error metrics. As expected, estimates of the second iteration are far superior to the first iteration. The third iteration reduces the error further but the reduction rate is smaller than the previous iteration. This is due to the fact that after a certain number of networks stacked, the learning process will plateau as the amount of information to learn is much smaller. This trend is clearly visible in RTE and RRE metrics.

Figure 4.9 shows the distribution of the labels through multiple stages of the training procedure. Every iteration results in a narrower distribution of individual parameters in each label meaning that the residual error is shrinking. This shows that the model is effective. The rate of this reduction is smaller in iteration 3, as the learning process for the model slows down.

4.6.2 Temporal Model

The complexity and the accuracy of the hierarchical model are increased by adding more iterations. Another way to increase the accuracy by keeping the complexity lower is to employ temporal features.

Training the frame-to-frame estimation model requires a significant amount of system resources. Adding LSTM on top of that model will dramatically increase these requirements. In such cases, it is common to use a pre-trained feature extraction network and provide mid-level features as inputs to a LSTM model. We extract three feature maps from our model in various layers, as below:

- Feature maps immediately before the flow embedding layer (PREF).
- Feature maps immediately after the flow embedding layer (POSF).
- Feature maps prior to the fully connected layers in the feature extraction network (FCLF).

Figures 4.6 and 4.4 show these feature maps along with the relationship between the frame-to-frame prediction network and the temporal model. PREF is the earliest level of features among the three. This model requires the largest amount of calculations as the flow embedding and the feature extraction layers from the original model have to be included next to the LSTM model. The advantage of this feature map is the inclusion of the matching layer in the LSTM model.

To reduce the complexity further, POSF features are used. This is achieved by employing features after the flow embedding layer. However, a point-cloud feature extractor needs to be present in the LSTM.

Finally, FCLF is the smallest feature map. Most of the flow matching and feature extraction work is completed and only the fully connected model is left for the LSTM model.

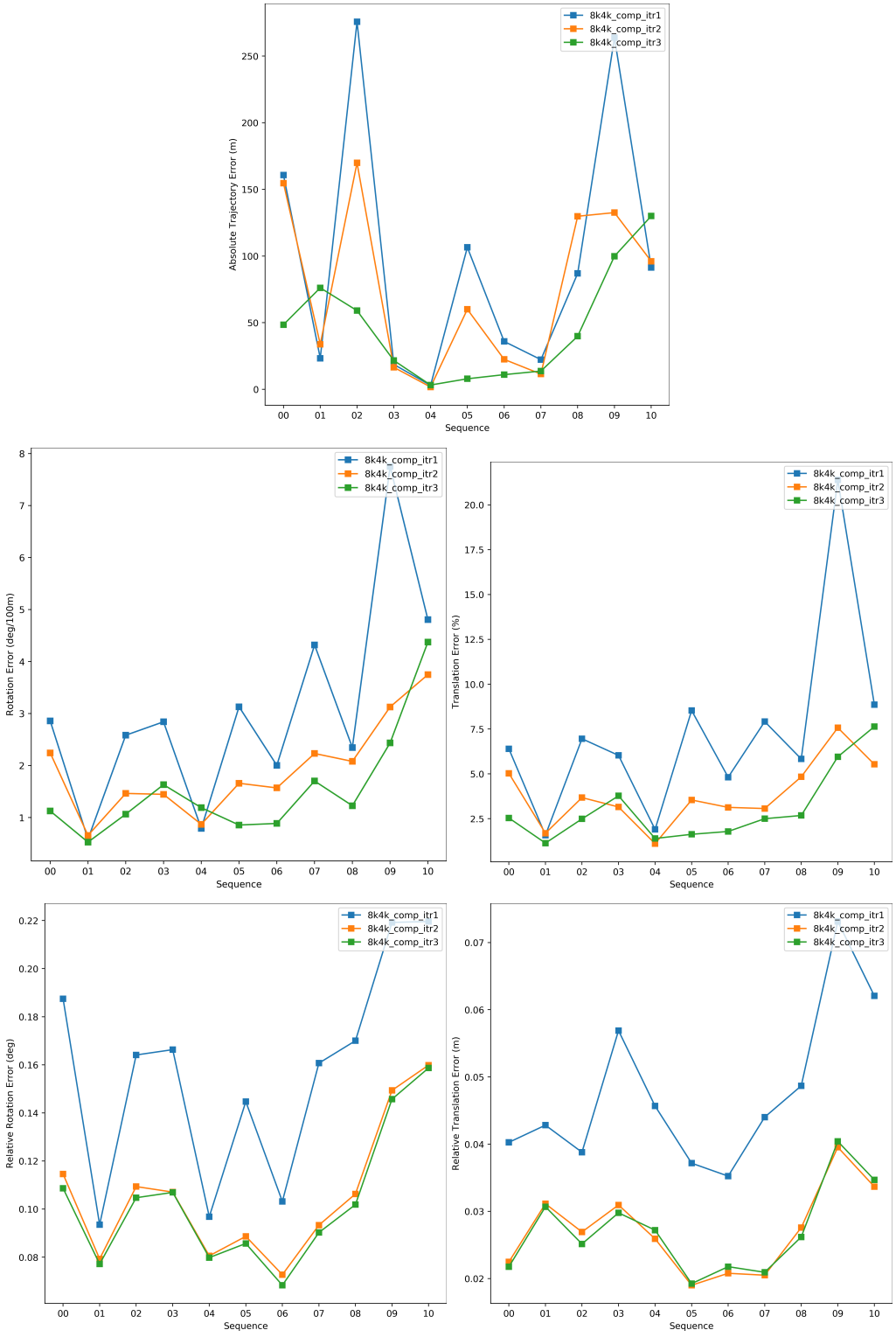


FIGURE 4.8: Comparison of errors on various iterations of the proposed model.

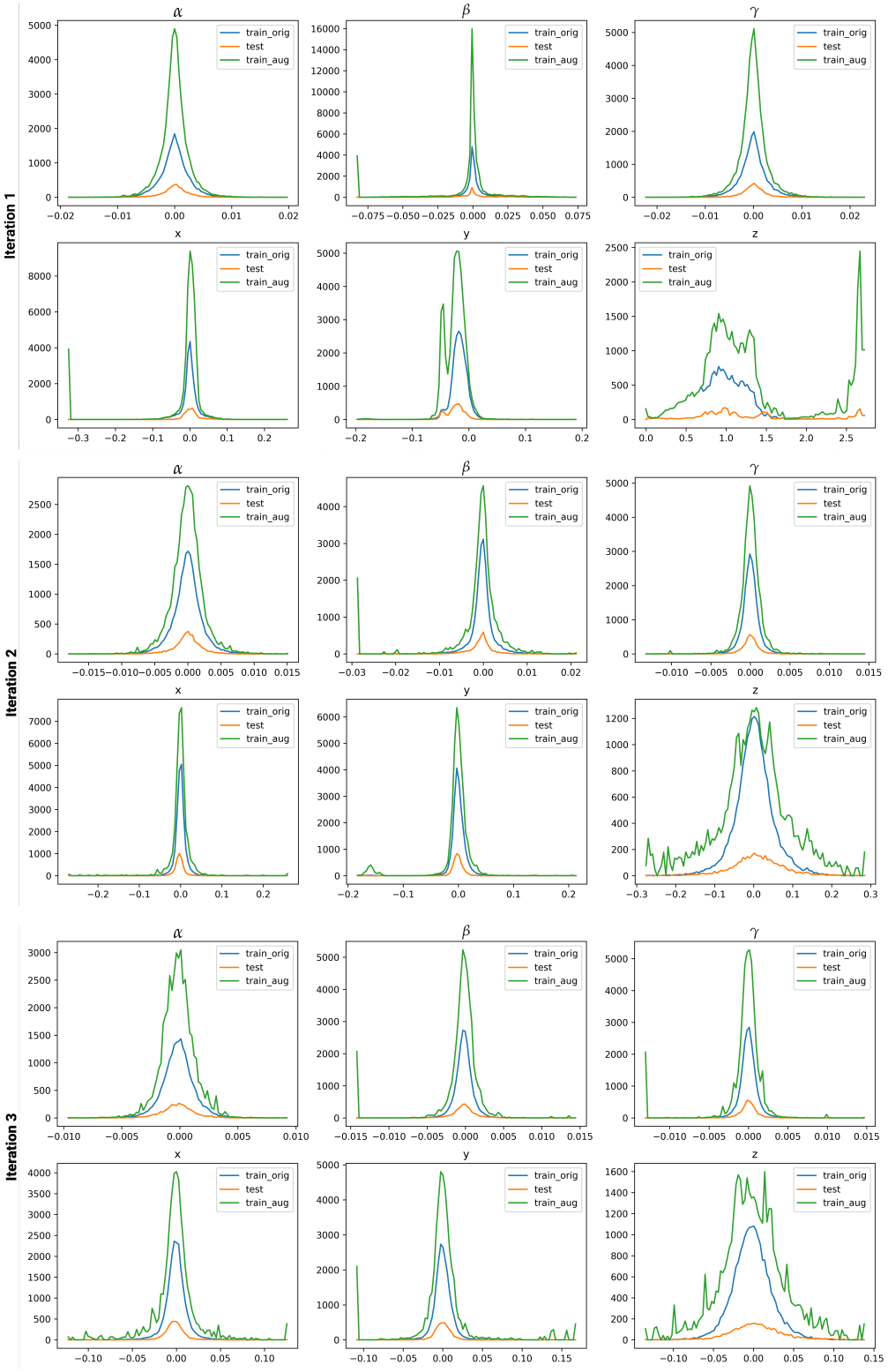


FIGURE 4.9: Progress of the distribution of label parameters through multiple iterations of training.

Once we employ the temporal features, the focus of errors will slide from RRE and RTE towards the other metrics, as they are more adequate for sequence-based analysis. PREF provides the best results for sequence analysis. This was expected as the matching layer gives more power to this model. However, while evaluating RRE and RTE, we observe that the FCLF features are outperforming both PREF and POSF features, especially in their relative rotation error. This is the objective that is used to train the frame-to-frame registration model. FCLF mimics the registration model rather than learning temporal features as the provided features are much less informative compared to PREF and POSF.

Figure 4.10 shows the results of these comparisons. It is seen that the usage of mid-level features is not as fruitful as using a second iteration. In some cases the results are worse than the original feature extraction model that is all attributed to dividing the training process into two parts.

4.6.3 Comparison to the State-of-the-Art

In this group of experiments, we compare the proposed model to the state-of-the-art. Table 4.1 compares various models in terms of sequence translation drift percentage and mean sequence rotation error for lengths of $[100, 800]$m. Please note that the results for LOAM [200], $\text{ICP}_{\text{reported}}$ are taken from [105].

LOAM [200] is one of the benchmarks in this field. It clearly outperforms all of the deep methods in the comparison. This is a new field for deep learning models and they have not matured enough to be competitive with such traditional models. One main reason for that is the feature extraction back-bone network. In our work, we relied on PointNet++ [140] features. Both LO-Net [105] and DeepLO [41] use 2D depth and surface models such as vertex and normal representations. They completely avoid the 3D feature extraction phase. This trend is also visible across the field. However, there is a growing interest regarding the 3D feature extraction methods [182, 192, 190] that can enhance the performance of deep odometry estimation with 3D point-clouds.

DeepLO [41] uses sequence 00-08 to train. It is clearly seen that their method is over-fitting to the training set. Our model is also suffering from such a phenomenon, but the scale of over-fitting is much lower. One reason for this is the size of our model compared to the size of the input dataset. Using PointNet++ layers results in large networks that require a large amount of data. LO-Net [105] addresses this problem by utilizing a 2D-based feature extraction network [207, 118]. We use an implementation of the ICP algorithm from the Open3D library[1]. In our implementation we use the sub-sampled point-clouds. This results in a significant drop in performance in comparison to the results of $\text{ICP}_{\text{reported}}$ [105]. The sub-sampling stage that is used to reduce the

[1] http://www.open3d.org/

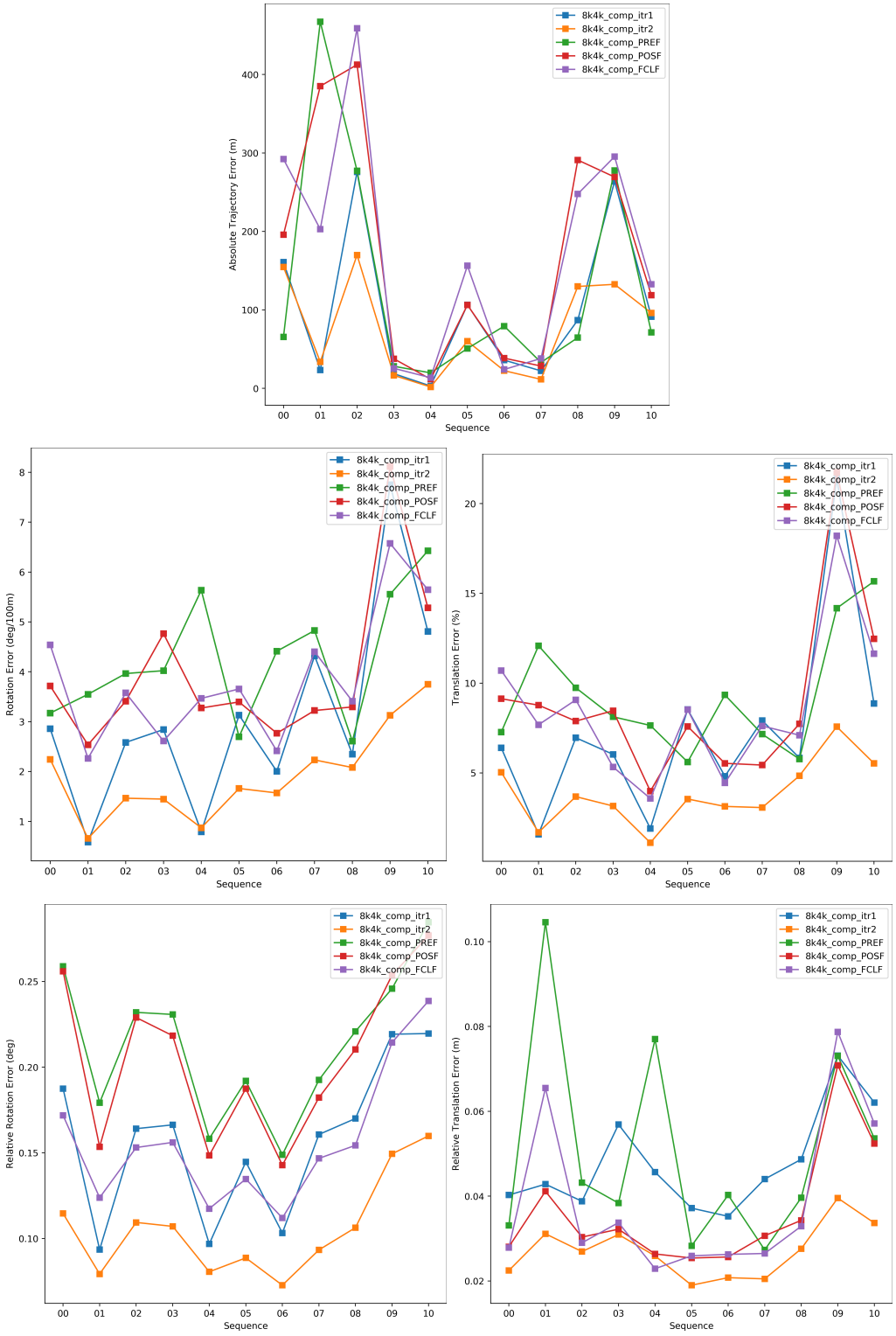


FIGURE 4.10: Comparison of the performance of the temporal model.

computational complexity is another aspect that affects the estimation performance of our model. The same points are not always chosen to represent the same static objects. This inherently adds noise to our dataset. We further explore using ICP as a final step on our estimates that significantly improves the performance. This is an expected outcome, as the complexity of the residual problem to solve for ICP at this stage is less than the original one. Hence, it can easily find the corresponding points and estimate better registration parameters.

	DeepLO[41]		LO-Net[105]		LOAM[200]		$\text{ICP}_{\text{reported}}$[105]		ICP		Ours		Ours+ICP	
Sequence	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
00	0.32	0.12	1.47	0.72	1.10 (0.78)	0.53	6.88	2.99	32.14	13.4	3.41	1.48	2.38	1.08
01	0.16	0.05	1.36	0.47	2.79 (1.43)	0.55	11.21	2.58	3.52	1.14	1.55	0.68	2.01	0.60
02	0.15	0.05	1.52	0.71	1.54 (0.92)	0.55	8.21	3.39	22.64	7.02	3.35	1.40	2.81	1.27
03	0.04	0.01	1.03	0.66	1.13 (0.86)	0.65	11.07	5.05	37.30	4.76	6.13	2.07	5.68	1.65
04	0.01	0.01	0.51	0.65	1.45 (0.71)	0.50	6.64	4.02	2.91	1.34	2.04	1.51	1.65	1.21
05	0.11	0.07	1.04	0.69	0.75 (0.57)	0.38	3.97	1.93	56.91	19.93	2.17	1.09	1.74	0.99
06	0.03	0.07	0.71	0.50	0.72 (0.65)	0.39	1.95	1.59	29.30	6.93	2.47	1.10	1.66	0.89
07	0.08	0.05	1.70	0.89	0.69 (0.63)	0.50	5.17	3.35	42.01	28.80	3.62	1.91	1.23	0.92
08	0.09	0.04	2.12	0.77	1.18 (1.12)	0.44	10.04	4.93	36.58	12.36	3.61	1.61	2.74	1.29
09*	13.35	4.45	1.37	0.58	1.20 (0.77)	0.48	6.93	2.89	36.54	12.82	8.26	3.11	2.69	1.57
10*	5.83	3.53	1.80	0.93	1.51 (0.79)	0.57	8.91	4.74	28.54	6.48	11.19	5.65	6.22	2.33

TABLE 4.1: Sequence translation drift percentage t_{rel} and mean sequence rotation error r_{rel} for the lengths of $[100, 800]$m. Sequences 09 and 10 are used as the test set.

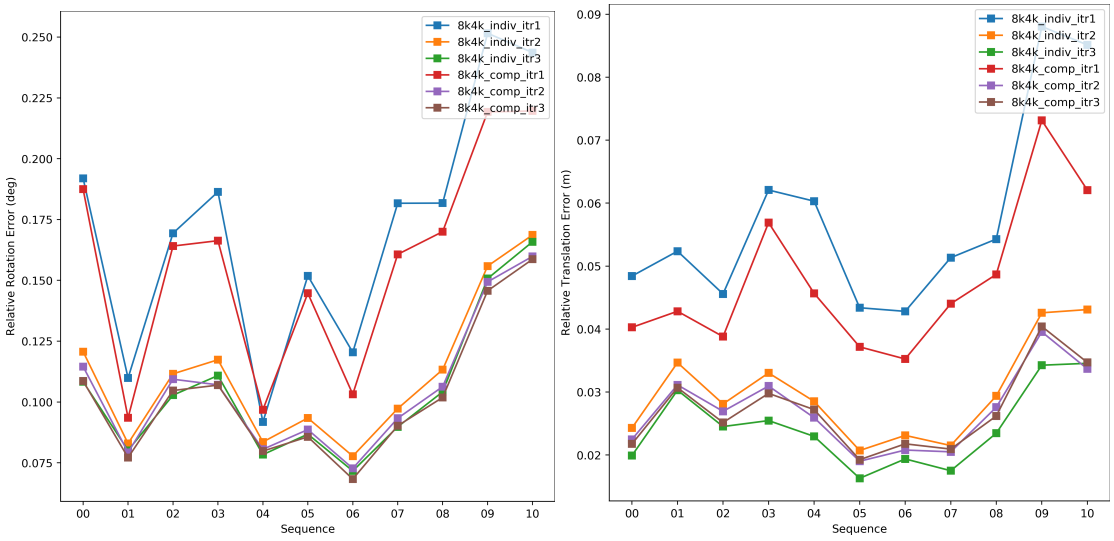


FIGURE 4.11: Comparative results over various iterations for *comp* vs *indiv* weighting scheme in loss function. ***comp*** uses one weight for rotation component and one weight for translation component in the loss function. ***indiv*** uses individual weights for each parameter in the pose.

4.6.4 Loss comparison

The proposed model employs weighted *l2_norm* loss on Euler angles and translation parameters. The loss function utilizing 2 weights for each rotation and translation components is indicated with *comp*, while the *indiv* represents the usage of individual weights for each transformation parameter.

Figure 4.11 shows comparative results of this experiment. We observe that in the first two iterations, component based weighting provides better results. However, in the third iteration, individual weighting achieves comparable results to the component based function. It is worth mentioning that the scale of reduction in error between iteration 2 and 3 is small, and the majority of the error reduction is achieved in the first 2 iterations.

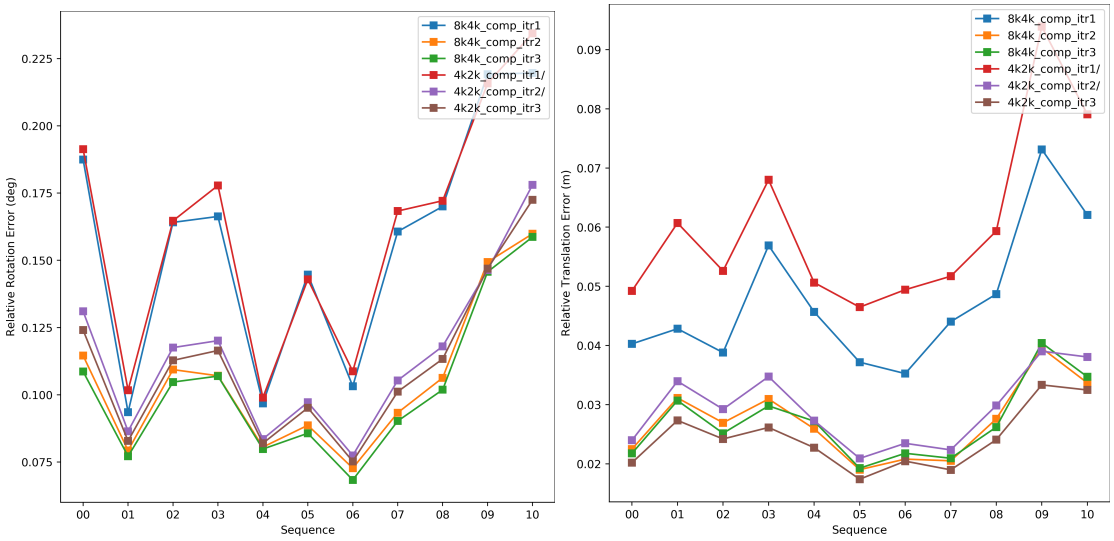


FIGURE 4.12: Input point-cloud dimensionality analysis. ***8k4k*** indicates 8*k* non-ground and 4*k* ground points in the input cloud. ***4k2k*** uses 4*k* non-ground and 2*k* ground points in the input cloud.

4.6.5 Input Dimensionality

We evaluate the performance of the models with 12*k* and 6*k* input points. *8k4k* represents the 8*k* and 4*k* division between non-ground and ground points. Similarly, *4k2k* corresponds to 4*k* ground and 2*k* non-ground point sampling. Extra points only help in providing better descriptors at the first layer where the first sampling function in the network is called. This results in better performance of the model, especially in the first iteration where the disparity between matching points in two frames is much larger. However, the difference diminishes in the second and third iterations. This entails that by employing a hierarchical model we could reduce the complexity of the input

point-cloud by using coarser 3D point-clouds. Results of this comparison are shown in Figure 4.12.

4.6.6 Sampling Comparison

To better understand the importance of separately sampling ground and non-ground points, we train the same model (*6k_comp*) with 6*k* globally sampled points from the point-cloud. We train this model in a hierarchical manner and compare it to the *4k2k_comp* model that employs 4*k* non-ground and 2*k* ground input points. As it is shown in Figure 4.13, sampling without distinction between ground and non-ground points results in far worse performance. This is due to the large number of ground points in the point-cloud that provide much less information regarding translation and orientation of the sensor in comparison to the non-ground points.

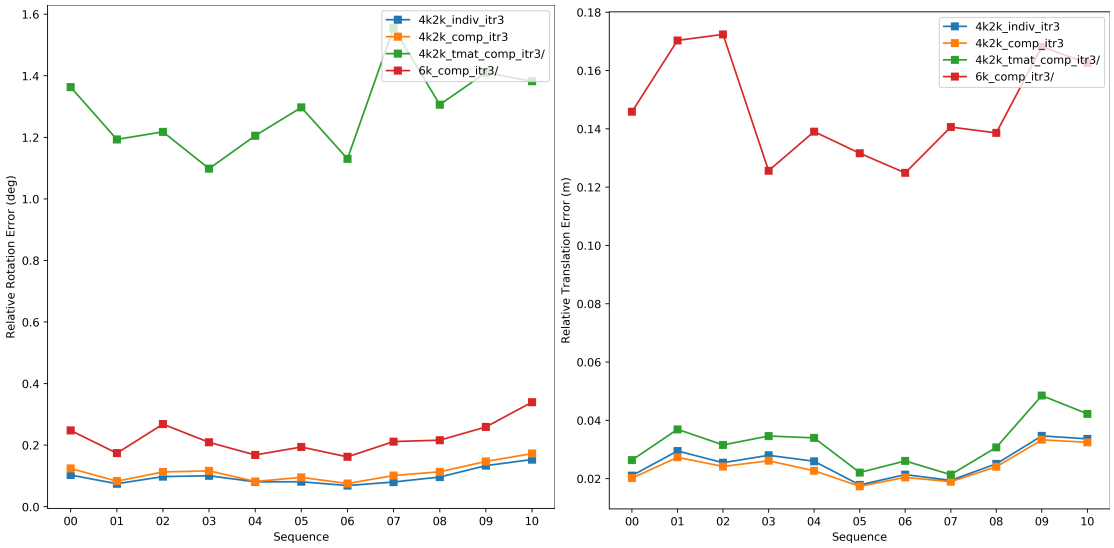


FIGURE 4.13: Comparative results for various ablation studies. ***4k2k*** uses 4k non-ground and 2k ground points as its input. ***6k*** indicates that 6k points are globally sampled without any distinction between ground and non-ground points. ***comp*** uses one weight for the rotation component and one weight for the translation component in the loss function. ***indiv*** uses individual weights for each parameter in the pose. ***tmat*** indicates that the rotation matrix is used instead of normalized Euler angles to calculate the loss.

4.6.7 Label Representation

Normalized Euler angles are used as the primary labels along with normalized translation parameters in our experiments. To validate our choice, we compare this label representation to the transformation matrix representation with 12 parameters (3×3 rotation

and 3 translation). We employ the component-based weighting on rotation and translation components of this representation. The trained model is shown as *4k2k_tmat_comp* in Figure 4.13. Results show that normalized Euler angles are a much better representation than the 3×3 rotation matrix, which is an over-parameterized representation of a rotation.

4.7 Conclusion

In this chapter, we have proposed a methodology to use deep neural networks to estimate odometry based on 3D point-clouds. We have proposed a data augmentation mechanism along with measures incorporated in the loss function to estimate the frame-to-frame transformation parameters. The proposed hierarchical model successfully reduces the error in consecutive iterations. Furthermore, we have evaluated the usage of pre-trained feature maps for training temporal models.

Our results are comparable to the state-of-the-art. We argue that the extracted features from the 3D point-clouds are not descriptive enough for this task. 3D point-cloud-based deep learning is still a new field and 3D deep feature extraction techniques have not matured as much as their 2D image-based counterparts. Employing novel 3D feature extraction models that require fewer parameters will enhance the performance of point-cloud-based odometry estimators. Similar to [207], using unsupervised learning models can enable usage of various datasets; an important bottleneck at the time of writing this thesis. Changing the backbone to a model with lower complexity can enable merging of the temporal model to the registration model. This will equip the deep model with the ability to learn temporal features more efficiently and provide it with the capability to smooth the errors in consecutive predictions, hence resulting in better estimates.

Chapter 5

Radar based Open Space Segmentation

Camera and lidar processing have been revolutionized with the rapid development of deep learning model architectures. Automotive radar is one of the crucial elements of automated driver assistance and autonomous driving systems. Radar still relies on traditional signal processing techniques, unlike camera and lidar based methods. We believe that this is the missing link to achieve the most robust perception system. Identifying drivable space and occupied space is the first step in any autonomous decision making task. Occupancy grid map representation of the environment is often used for this purpose. We propose PolarNet, a deep neural model to process radar information in polar domain for open space segmentation. We explore various input-output representations. Our experiments show that PolarNet is an effective way to process radar data that achieves state-of-the-art performance and processing speeds while maintaining a compact size.

In this chapter, we are proposing the use of radar with advanced deep segmentation models to identify open space in parking scenarios. We collect a raw radar dataset, extract multiple input representations and annotate it for open space. We propose PolarNet, a novel radar segmentation architecture. We further implement multiple well known models and evaluate their performance on various input representations and loss functions. Compared to the state-of-the-art, our proposed model achieves lower memory usage and faster processing time than larger models, and better performance compared to similarly sized models. These characteristics make PolarNet very well suited for embedded deployment.

5.1 Introduction

Autonomous driving can be approached from various sensor modalities such as camera, radar, sonar, or lidar. Each sensor provides a piece of valuable information. It is shown that the fusion of multiple sensors is required to cover all of the environmental

variations and achieve fully autonomous driving. Despite this observation, each individual sensor needs to be pushed to the edge of its capabilities to reduce the complexity of the fusion systems.

Radar sensors have been around in the automotive industry for a few decades already. The first systems were mostly used in the premium car segment for comfort applications, such as Adaptive Cruise Control (ACC). With the continuous improvement of radar technology, recent radar sensors are also used in safety applications, such as Autonomous Emergency Braking (AEB). Furthermore, radars are not affected by poor lightning and fog, unlike camera and lidar respectively. The new versions of radar will significantly increase the resolution of the observations, therefore enabling an even wider variety of applications.

Ultra-sonic sensors are very similar to radar, with a focus on close range applications only, such as automated parking detection. In this chapter, we propose a model to equip radar with the ability to replace ultra-sonic sensors. In this way, a single sensor can run in multiple modes to provide better value and reduce system complexity.

In recent years, deep learning models have achieved state-of-the-art performance on various applications [170, 202, 146, 109, 35, 77, 138]. However, these models are rarely used with automotive radar; traditional signal processing methods such as Constant False Alarm Rate (CFAR) [63, 122] are commonly used to process radar data. Radar is an alternative range measurement sensor that presents a less expensive solution than lidar at the cost of accuracy of measurements. Recent advances in radar technology introduced High Definition radars that address the issues with the quality of the depth data. The lack of publicly available radar datasets could be seen as the primary reason for the smaller amount of literature in this field. This issue is partially addressed by novel datasets with radar observations [13, 131] that have recently been introduced to enable the scientific community to expand the boundaries of knowledge in this field.

In this paper, we introduce a radar-camera dataset and a novel deep learning approach that takes benefit of the polar representation of radar observations and performs open space segmentation in parking lot scenarios. Our dataset, termed SCORP, includes raw radar observations and camera images taken in various parking lots, in addition to open-space annotations. On the other hand, our model relies on a series of *1D* and *2D* convolutions that reaches state-of-the-art performance. To address the requirements of the automotive industry, we ensure that the model is compact and fast to run on embedded platforms. Our contributions are listed as follows:

1. The first publicly available comprehensive dataset including raw radar echos along with ground-truth open space annotations.

2. A deep model named PolarNet, that takes radar frames in polar coordinate and generates an open-space segmentation mask in a parking-lot scenario

3. A comprehensive study of various radar modalities, models, and loss functions for the purpose of open area segmentation.

4. Detailed comparison of PolarNet to the segmentation state-of-the-art in terms of performance, speed, and size.

The literature review of deep segmentation model architectures, along with radar applications, are discussed in Section 5.2. A brief description of radar data can be found in Section 5.3. The compared model architectures are described in Section 5.4.1. In Section 5.5, various experiments, including the effect on model performance of using different input data representations, segmentation models, and loss functions are evaluated. Furthermore, the computational complexity requirements of each model are discussed in detail.

5.2 Literature Review

Many research works are proposed to address the challenge of labeling individual pixels in images for semantic segmentation. There are two category of methods in this subject. One uses an *encoder-decoder* architecture, and the other uses specialized convolutions to avoid decimating input map size. The complexity of the former approach is highly dependent on the models used for each component, while the latter suffers from large memory requirements caused by maintaining the large feature maps which help in generating fine segmentation masks.

Chen *et al*. [34] first proposed the idea of using a fully connected conditional random field as a decoder at the end of a deep convolutional model to extract quality segmentation masks. FCN [162] uses an encoder network, made up of convolutional layers only, to extract intermediate feature maps of the inputs. By employing skip-connections, upsampling, and transposed convolution operations on the decoder side, an output mask of the desired size is generated. Following up with the idea of Chen *et al*. [34], *DeconvNet* is introduced in [129], and utilizes a more specialized decoder architecture that consists of a series of decoupled unpooling and convolution layers. Badrinarayanan *et al*. [12] propose a similar architecture to DeconvNet, but with greatly reduced complexity and compares variations of the FCN, DeconvNet and SegNet models.

The *U-Net* [151] architecture iterates on the FCN model by using deeper decoder feature maps and extensive data augmentation. The *U-Net++* [209] architecture is a more general version of U-Net which significantly increases the number and the complexity of the skip connections between the encoder and the decoder.

Chen *et al*. [33] pioneered the use of atrous convolutions for segmentation to avoid subsampling of the input data, and instead enlarge the field of view of the convolution

kernel without using any pooling layers. Furthermore, multiple parallel atrous convolutions are utilized to segment objects at different scales.

Zhao *et al*. [206] introduce the idea of pyramid pooling to collect global and local information. Inception-ResNet [170] is used with [33] to build mid-level feature maps. By using parallel pooling layers, coarse to fine information is generated that is later used to extract the final segmentation result. Chen *et al*. [36] merge both categories of methods by using [33] as the encoder network and using a small decoder network to achieve better performance.

To accelerate scene segmentation, Badino *et al*. [10] consider the space in front of a vehicle free unless there is a vertical obstacle present. This resulted in the introduction of Stixel as a rectangular block on the image that identifies vertical surfaces. Hence, a compressed representation of the environment is achieved. Schneider *et al*. [158] add the semantic labels to each stixel. Inspired by these ideas, Stixelnet [104] segments an image for open space using stixel-like rectangular regions. Instead of a segmentation model, it relies on a classification network that predicts the junction point for ground and the obstacle. Later, a conditional random field is used to smooth the jittery predictions from the column-wise network. In this case, each stixel corresponds to a specific angle interval from the camera point of view. As we explain later, the polar representation of the radar is well suited for this family of architectures.

The majority of aforementioned methods rely on camera based inputs. It is common to use a pre-trained back-bone to build a new model as training a completely new architecture is still a challenging problem. However, in the case of radar input data, using a pre-trained model from other domains is not a promising approach. This is due to the fact that there are various radar representations, and that most radar processing currently uses traditional techniques [63, 122]. Sless *et al*. [165] propose an encoder-decoder architecture with a Bird's Eye View (BEV) input and a 3-class output: occupied, unoccupied or unobservable. Bauer *et al*. [14] propose a similar U-Net architecture for occupancy prediction. They formulate the classification problem as both a three class problem, as in [165], and as a four class problem. The four-class approach uses an *unknown* class to show the certainty of the predictions.

Another issue in the radar domain is the low number of publicly available datasets that contain some form of radar data. The NuScenes dataset [28] is a large-scale dataset developed for autonomous driving. However, the radar data is provided after processing with traditional models, rather than providing the raw signal data.

The *Oxford Radar RobotCar* dataset [13] is available for scene understanding analysis. The radar used for data collection offers much finer resolution and higher range than typical automotive radars. It provides a 360 degree azimuth-range representation of the received power reflection. It is worth noting that this is 2 dimensional representation. Raw radar data is not available in this dataset, but the radar modality provided by the

authors is less processed than that of the NuScenes dataset. The usage of specialized hardware uncommon in the automotive domain due to its physical characteristics is a major disadvantage for some applications.

5.3 Radar Data

To the best of our knowledge, there are no publicly available datasets for radar with accessible Analog-to-Digital Converter (ADC) signals and annotations. To overcome this problem, we collected our own dataset. We moved a car equipped with a side view radar and camera in a parking lot with the objective of identifying the drivable open spaces in the scene. A Linear Frequency Modulation Continuous Wave (LFMCW) radar with 77 Ghz frequency in Time Division Multiplexing-Multiple Input Multiple Output (MIMO) mode is utilized to collect the environment observations. The usage of the TDM-MIMO mode results in 8 virtual channels for the radar information: 2 Tx elements transmit sequentially and 4 Rx elements receive coherently. The resulting dataset is made up of 3913 frames, collected in 11 driving sequences.

To collect any radar data, the first step is to select a set of parameters for the signal. Table 5.1 shows the details of the configuration used to capture the data.

TABLE 5.1: Details of the configuration used with radar.

Frequency	77 *Ghz*
Maximum Range	15 *m*
Range resolution	0.12 *m*
Unambiguous Velocity	10.5 *m/s*
Velocity resolution	0.33 *m/s*
Field of View	90 deg

A camera is also used in our data capture to assist with the annotation of the data. We fix it to the same frame-rate as the radar and capture images of size 1280x960 pixels. The captured visual information by camera is later used to create the ground truth labels. All communication between various components and their synchronization is managed through the *Robot Operating System (ROS)* software.

5.3.1 Radar Processing Pipeline

To propose a new model, we first need to understand the data that will be used. Raw radar data consists of a series of echos received from each transmitter-receiver combination. The signals at each receiver element have the same amplitude but different phase values that represent angular spectrum once converted to frequency domain. This results in a four dimensional matrix of Samples, Chirps, Transmitters, and Receivers. It

is common that the last two dimensions are concatenated and stacked as one dimension resulting in a three dimensional representation of Samples, Chirps, and Antennas (SCA). SCA is the raw echo representation and is the earliest level to process the radar information. Although it includes all the information, its visualization does not provide much insight into the data. Further, all the information at this stage is in the time domain and there is no spatial coherence between the values. This entails that any model applied to this stage should explicitly or implicitly include layers to extract spatially coherent information.

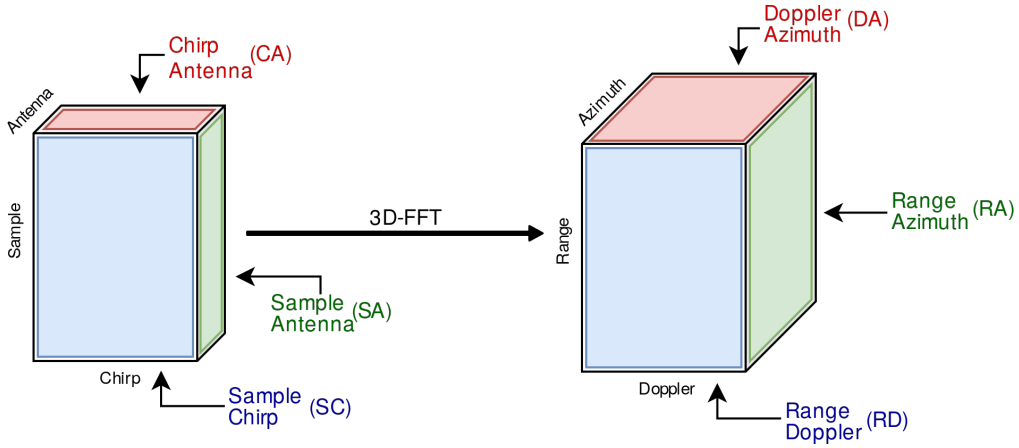


FIGURE 5.1: Radar data representation. (Left) SCA representation. Arranging data along antennas results in Sample-Chirp (SC) view, arranging along chirps results in Sample-Antenna (SA), and Chirp-Antenna (CA) is achieved by arranging data along sample dimension. (Right) Fourier transformation is applied along all three dimensions. Arranging values along azimuth results in Range-Doppler (RD), arrangement along Doppler results in Range-Azimuth (RA), and arrangement along range results in Doppler-Azimuth (DA).

To extract spatially coherent information, Fast Fourier Transforms (FFT) are applied along each dimension to achieve the Range, Doppler, and Azimuth (RDA) representation. The distance associated with an observation is represented in the *Range* dimension, the velocity is represented in the *Doppler* dimension, and the angle of arrival of the signal is represented in the *Azimuth* dimension. Note that prior to applying the FFT along the Antenna dimension, zero-padding is applied along the last dimension to achieve the target angle resolution.

$$RDA = \text{fft}_A\left(\texttt{zero\_pad}_\texttt{A}\left(\text{fft}_C\left(\text{fft}_S(SCA)\right)\right)\right) \tag{5.1}$$

Taking transpose of RDA results in RAD that is a 3D tensor. First two dimensions represent the spatial information. The convolutions in a deep model can be applied on Range and Azimuth dimensions and along the last dimension. This input is the polar

representation of radar frames for each individual Doppler channel that is generated from the third FFT along antenna dimension of SCA. RAD is commonly used by traditional methods to detect objects.

Range-Azimuth (RA) is a summarized spatial representation of the received signals that is achieved by applying logarithm on the summation of the Doppler values of each Range and Azimuth index. It represents the Bird-Eye-View (BEV) of the environment in polar coordinates. Radar data and conversion between the SCA and RAD is shown in Figure 5.1.

A polar to Cartesian transformation is regularly used on this representation to calculate the Direction of Arrival (DoA) point-cloud map in Cartesian coordinates. DoA is the Cartesian Bird-Eye-View (BEV) of the environment. The pixel values in the matrix represent the power received by the radar sensor at that location. The benefit of this modality is its metric coherence with convolutional kernels. This is important as the same convolutional filter at every location of this representation retains the same receptive field size.

In our dataset, we store SCA, RDA and DoA tensors to cover all the various mainstream levels of inputs to any system.

5.3.2 Annotation Challenge

Annotating radar data is an extremely challenging task as the echo-responses are not easily understandable for human. DoA point-cloud is an easier representation to understand, but the level of information is coarse, such that it is extremely difficult to use for annotation. To overcome this issue, we employ the sequence of monocular camera images collected in synchronization with radar. As the calibration of a single monocular camera to the radar sensor is prohibitively difficult, we rely on scene reconstruction techniques to extract odometry information. Once an odometry trace is calculated, we use this trace to accumulate the radar DoA's. The open source software of [123] is used to perform 3D reconstruction and extract the odometry trace.

Open space annotations from accumulated DoA are propagated to corresponding frames using the odometry and radar intrinsic parameters. This ground truth generation pipeline is shown in Figure 5.2.

As the annotations are propagated from 3D reconstruction, labels include values for locations that are not in the direct line of sight of the radar. Even though a radar can still detect free space in those regions, they are removed to be consistent with the single frame based annotations of [13].

Note that the inputs to our proposed models only include the radar data, and the output is generated as the occupancy grid map in Cartesian coordinates. Also, it is worth mentioning that the annotation task can be handled much more easily if a laser depth sensor such as Lidar would be used.

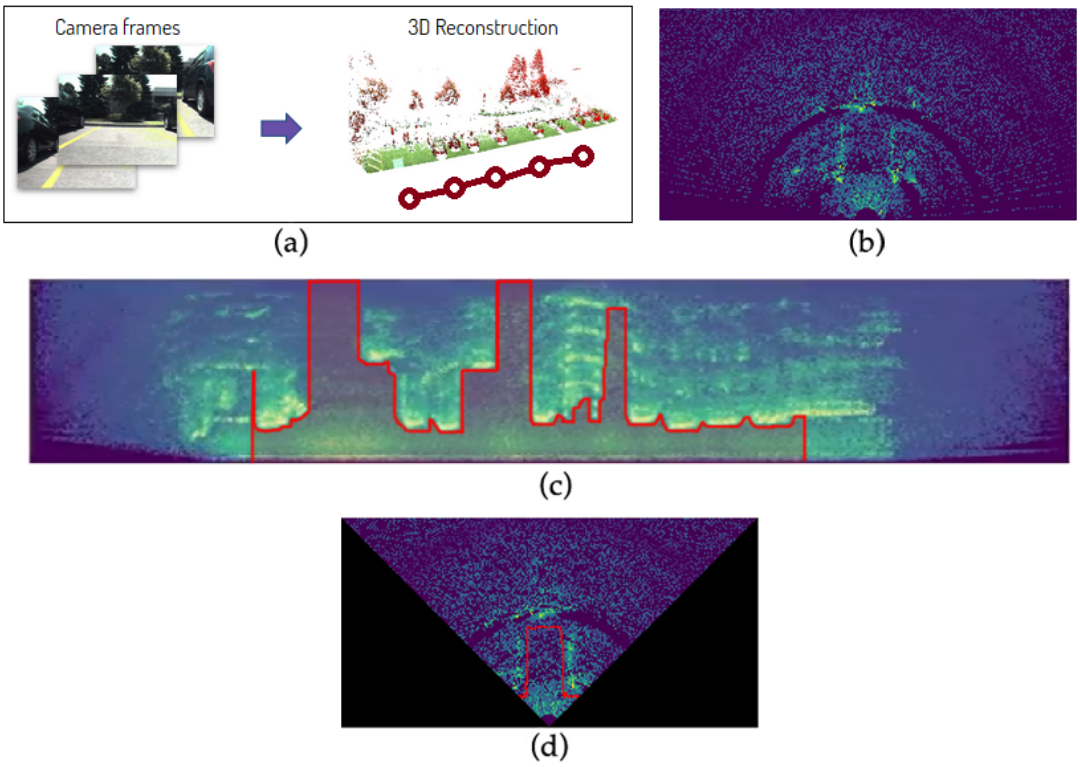


FIGURE 5.2: Ground truth generation pipeline. (a) Camera frames are used to reconstruct the scene and extract odometry. (b) Cartesian DoA from a single frame. (c) Accumulated Cartesian DoA which is used to generate initial annotations. (d) Annotations are distributed to the corresponding frames, cropped for the radar's field of view, and are manually adjusted.

The SCORP dataset is available www.sensorcortek.ai/publications.

5.4 Proposed Models

We propose two new models and compare them to two deep learning approaches inspired by recent work in the field of semantic segmentation . Our primary proposal is a novel approach that takes advantage of information representation in the polar domain to produce impressive results against the state-of-the-art. This approach is called *PolarNet*. Our second proposal is based on the *Fully Convolutional Networks* (FCN) [162] architecture and is referenced as *FCN_tiny*.

5.4.1 PolarNet

PolarNet is a convolutional neural network that is applied on the RA and RAD representations of radar observations. In this representation, the first dimension (rows) of the

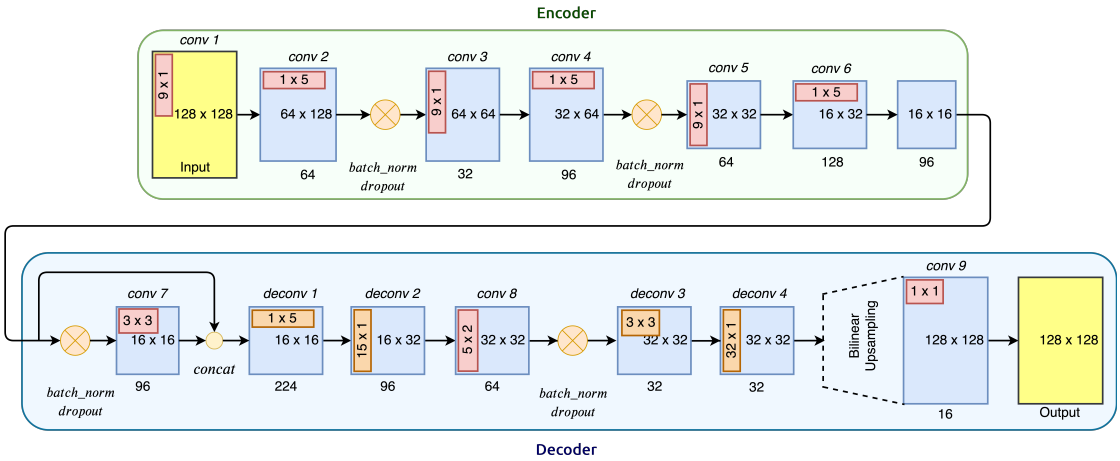


FIGURE 5.3: Detailed architecture of the PolarNet model.

input tensor renders the distance, while the second dimension (columns) corresponds to specific angles. Inspired by StixelNet [104] that uses Stixels, we propose applying one dimensional convolutions on each column of the input. In this way, we apply a filter on each angle and effectively try to identify where the open space ends. Furthermore, by using one-dimensional filters, we reduce the complexity of the network.

PolarNet is an encoder-decoder network with a smoothing layer at the end of the model. All of the layers use ReLu [126] as the activation, except for the final two layers that employ Sigmoid activations.

The encoder portion of the model consists of 6 layers. Three of these layers are column-wise convolutional filters that are designed to find the location where the ground meets the first obstacle. Another three layers, positioned after each column-wise layers, are designed as row-wise convolutions. These layers are used to pool the information from neighboring columns, and reduce the size and, consequently, the computational complexity of the model. Column-wise layers have a stride of 2 along the columns and row-wise layers use an stride of 2 along each row. After each column-wise and row-wise convolution tuple, a batch-normalization [88] and drop-out [166] layer is employed.

The decoder network consists of a series of convolutions and transposed convolutions. It starts with a two dimensional 3×3 convolution with a stride of 1. The output of this layer is concatenated with the output of layer 6 in encoder. Reversing the order in the encoder, one row-wise transposed convolution is applied on the feature map, followed by a column-wise transposed convolution. The former layer has a stride of 2 along the columns, while the latter uses the same stride size along the rows dimension. The output of this layer is fed to a two dimensional convolution layer with a kernel size of 5×2. This layer is responsible for integrating adjacent feature maps with the goal of reducing jittery artifacts in the final output. We apply the final batch-normalization

[88] and drop-out [166] layer. Another column-wise convolutional layer with a larger kernel of 32×1 is used at this stage, thereby forcing a larger receptive field over the feature map. A radar is capable of observing spaces behind various objects such as cars. Using this filter size, the information from an obstacle is propagated to those trailing areas. This helps in assigning opposite bits to the ground and non-ground locations; if the larger receptive fields are not used, the model will be confused by the conflicting nature of the information in the input and the mask requirements.

At each layer of the decoder, the depth of the filter map is consistently reduced. Prior to the final 1×1 convolution layer that pools all the information to produce the final result, the feature maps are upsampled to match the input shape through bilinear up-sampling. The complete architecture of the proposed model is shown in Figure 5.3.

The final output contains logits in the range [0,1]. Cells with a value lower than 0.5 is considered as occupied, while values higher than this threshold are considered as non-occupied. After this division, a softmax cross-entropy (SMCE) function is used to extract the final loss value. We use the $SMCE_{train}$ loss function as in [131]:

$$SMCE_{train} = \sum_{c}^{\text{class}} \left(\frac{1}{N_c} \sum_{i}^{N} (e^{-w_c} \times SMCE_i + |w_c|) \right) \tag{5.2}$$

where w_c represents the trainable weight for class c, N_c is the number of pixels of class c in a particular groundtruth mask, and $SMCE_i$ represents the softmax cross-entropy loss calculated for a pixel i. As such, this modification of SMCE uses trainable parameters to weight the SMCE loss on a per-class basis.

A rate of 0.5 is used as drop-out probability. Batch size is set at 64. RMSProp with initial learning rate of 0.1 and decay factor of 0.8 at every 3500 steps is chosen as the optimizer.

5.4.2 FCN_tiny

The *FCN_tiny* architecture is a smaller variation of the FCN architecture, which is a simple encoder-decoder method which uses transposed convolutions to upsample feature maps by a factor of 2. The encoder's output is upsampled and concatenated with lower level feature maps before being passed through a 3x3 convolutional layer, thereby recovering detailed spatial information. Two of these upsampling steps are used, generating output feature maps 8 times smaller than the network's input. These are then resized to the input size through bilinear interpolation. We use MobileNet-v2 [84] as the encoder for our experiments.

The *FCN_tiny* variant uses a number of feature maps in each encoder layer reduced by 75%, and a depth of 8 in the decoder feature maps. The resulting model has a much lower number of parameters than the original FCN model with a MobileNet-v2 backbone.

5.5 Experiments

We use 3193 frames for training, and 720 for evaluation (9 and 2 driving sequences, respectively) from our dataset. Mean Intersection-over-Union (mIoU), commonly used for semantic segmntation tasks [162, 33], is selected as the evaluation metric. mIoU is calculated as the average of the Intersection-over-Union (IoU) metric of each class.

To compare our proposed model, we have implemented two of the state-of-the-art segementation models, *DeepLabv3+* [36] and *Fully Convolutional Networks* [162].

The DeepLabv2 architecture for segmentation is implemented through two core concepts: reducing the output stride of the feature extractor while using atrous convolutions to generate larger feature maps, and performing atrous spatial pyramid pooling (ASPP) to cover a wider range of object sizes. The *DeepLabv3+* architecture iterates on the architecture of the ASPP, and appendes a small decoder to the DeepLabv2 encoder network, which upsamples the feature extractor's output and combines them with features from earlier layers. We implement a complete version of the DeepLab architecture, referred to as *DeepLabv3+* in our experiments. This version uses atrous convolutions of rates 2, 4 and 6 in its ASPP module. The model's output is then resized to the input size through bilinear interpolation.

The FCN architecture is explained in Section 5.4.2 along with its *FCN_tiny* variation. All of these three segmentation models are trained with MobileNet-v2 as their feature extractor.

For training these models, we used TensorFlow [1] as a framework to implement and test our models. We used *Trainable Softmax Cross-Entropy loss* as the loss function for each network, unless specified otherwise. This loss is based on Softmax Cross-Entropy loss, with the addition of trainable parameters used to weight each class during loss calculation, thereby avoiding the need to hand-pick appropriate weighting parameters. All training of our model architectures was undertaken on Nvidia Geforce RTX 2080 TI's.

Our experiments section is divided into four sections. First, we compare the effect of input representations on the performance of these models. Second, we evaluate the effectiveness of various loss functions on PolarNet. In the third experiment, we compare the computational performance and size of our model against the state-of-the-art on various platforms. Finally, we provide analysis regarding data compression capabilities of our proposed model.

5.5.1 Input Modalities

We used three distinct data representation, namely, RAD, RA, and DoA. RAD is a tensor of size $128 \times 128 \times 64$. Each range bin is approximately $11.17cm$ for a maximum range

[1] www.tensorflow.org

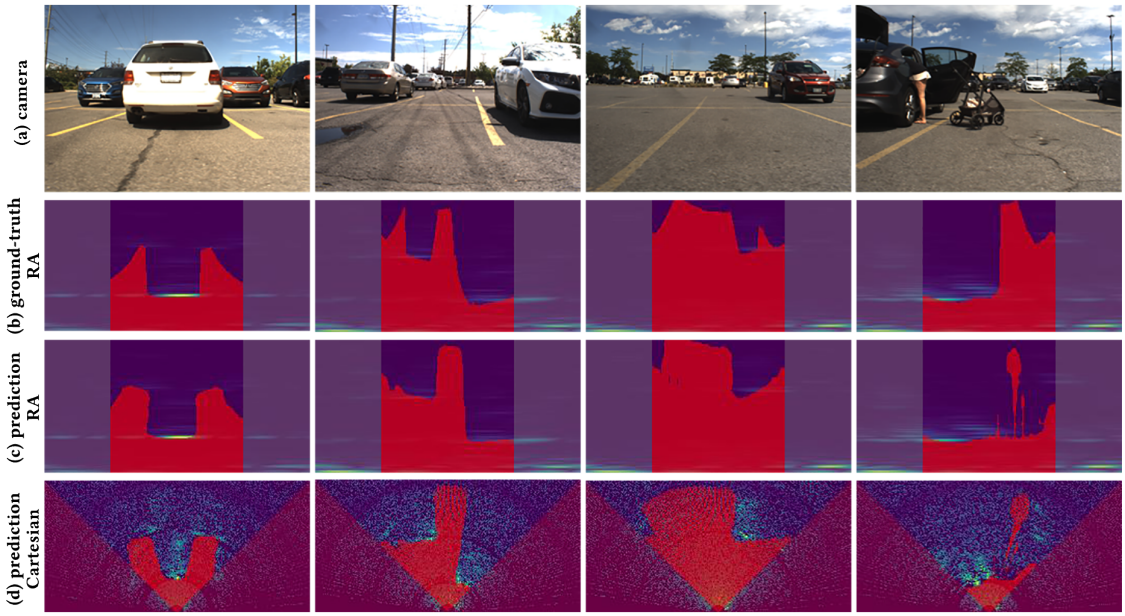


FIGURE 5.4: Sample results of PolarNet. (a) camera image. (b) ground-truth in polar coordinates. (c) segmentation result. (b) segmentation result shown in Cartesian coordinates.

of $15m$. Angle resolution is set to 0.70 degrees, covering an angle interval of -45 to $+45$ degrees. The third dimension represents the Doppler bins that cover a velocity range between -37.3 and $+37.3$ kmph. RA and DoA, both are tensors of size 128×128. The combination of these representations as input, polar and Cartesian outputs, along with the model architectures outlined in previous section, results in our list of experiments.

The predictions in polar coordinates can be simply converted to Cartesian system after inference. To isolate the effects resulting from having annotations in two different domains, and compare the polar inputs to Cartesian inputs fairly, we utilize two output representations:

- **Cartesian ground-truth:** As discussed in section 5.3.2, the open-space is annotated in a parking lot using DoA input Tensor. Then, the annotated points are used to generate a mask for open-space segmentation. However, we confined the field-of-view of radar to 90 deg. Cartesian ground truth can be used with all three input models.

- **Polar ground-truth:** After generating the Cartesian ground truth masks, the annotated points are transformed into the polar coordinate system. For training, we cropped the RA and RAD input tensors to match the selected field of view.

We conducted experiments where the input tensor is in one domain (i.e. polar), while the output mask is in another domain (i.e. Cartesian). We expect that the model

architecture should learn to adapt the transformation and generate comparable results. Table 5.2 shows the results of these experiments. As expected, having the input and output in the same domain in all cases resulted in better performance than learning the domain transformation internally.

We can further observe that using RAD as the input provides the best mIOU. This outcome is due to the descriptive information present in RAD that are manually summarized in RA and DoA representations. It is apparent that the model is extracting a better mid-level representation than the manual compression achieved through summation or coordinate transformation done by RA and DoA. RA beats the DoA in performance. This shows that a model using convolutional kernels defined with Cartesian coordinate in mind, is capable of adapting them to the polar usecase. From a sensor point of view, far points in the BEV map of DoA have much lower information density compared to the closer points. This imbalance is a reason for the lower performance of the DoA input.

DeepLabv3+ in almost every case is inferior to the similarly sized FCN model and the much smaller PolarNet. This shows that the atrous convolutions are not as effective in extracting useful features for this segmentation scenario.

Input	Label	PolarNet	FCN_tiny	FCN	DeepLabV3+
RAD	RA	83.79	83.14	85.16	83.58
RAD	DoA	N/A	77.95	79.71	77.48
RA	RA	84.20	84.18	83.98	83.01
RA	DoA	N/A	77.50	78.84	78.56
DoA	DoA	N/A	81.95	83.43	78.05

TABLE 5.2: mIoU reported for different model architectures.

PolarNet is designed to be applied on polar input representations only; hence, it is used with neither DoA inputs, nor with DoA masks. Our proposed PolarNet model is the best performer with RA inputs. While using RAD inputs drops PolarNet's performance, it still takes second place behind the much larger FCN model. From the experiments, we observe that larger models have an easier time managing the RAD inputs that contain 64 times more information than the RA. Inversely, smaller models such as *FCN_tiny* and PolarNet have a harder time handling the huge dimensionality increase of RAD inputs. Figure 5.4 shows a few samples of PolarNet's results.

5.5.2 Loss Functions

We compare three loss functions for this task: *trainable softmax cross-entropy loss ($SMCE_{train}$)*, *softmax cross-entropy (SMCE)* and *Lovasz loss* [18].

Softmax cross-entropy is commonly used as an extension of the typical classification task to the segmentation task; the loss is calculated independently on a per-pixel basis

and then averaged over the entire image. In contrast, Lovasz loss enables optimization of the mean IoU metric directly, thereby showing a direct link between the main metric for segmentation and the optimization of the loss function.

Using RA input and polar label modalities, we took the PolarNet architecture and experimented with the mentioned loss functions. Results are reported in Table 5.3.

Loss Function	mIoU
$SMCE_{train}$	84.20
SMCE	82.02
Lovasz	82.85

TABLE 5.3: mIoU for PolarNet with RA input and polar labels with different loss function

$SMCE_{train}$ achieved the best results in this experiment. Learning class weights within the network itself seems to show promising results as opposed to the more classical SMCE approach, which failed to generate accurate predictions. This is due to the necessity for precise class weighting to prevent the loss function from staying in a local minima. The Lovasz performs much better than the SMCE approach, but noticeably worse than $SMCE_{train}$; weighting the classes seems to improve the performance more significantly than optimizing with regards to the mIoU directly.

5.5.3 Computational Performance Analysis

Our final analysis concerns the computational complexity and the speed of our model architectures. It is crucial for the models to target embedded deployment and real-time performance on such systems. The details of our experiments for the proposed model architectures are shown in Table 5.4.

Input	RA				RAD			
Model	PolarNet	FCN_tiny	FCN	DeepLabv3+	PolarNet	FCN_tiny	FCN	DeepLabv3+
GPU (fps)	575.01	324.50	300.75	275.15	364.89	249.79	224.56	199.83
CPU (fps)	271.39	267.58	118.56	61.91	208.14	189.12	133.78	61.93
TX2 GPU (fps)	54.65	31.91	29.18	22.39	48.18	28.87	28.91	21.46
TX2 CPU (fps)	25.90	41.62	22.20	10.46	17.97	36.86	19.68	10.09
# parameters	562,472	210,279	2,933,449	3,223,865	598,758	214,817	2,951,593	3,242,009
Memory cost	29.85 *mb*	16.02 *mb*	59.64 *mb*	134.86 *mb*	39.79 *mb*	23.04 *mb*	70.81 *mb*	143.74 *mb*

TABLE 5.4: Computational complexity comparison for the tested methods.

DeepLabV3+ is the largest model with slowest performance in both RA and RAD input cases. Surprisingly, the performance of this model is also inferior to the other models. FCN is 90% of the size of DeepLabV3+ and almost 10% faster on GPU. However, on CPU it is almost twice as fast. This trend can be observed on the Jetson TX2 as well.

FCN is still a large model when considering a Radar on Chip (RoC) product. PolarNet and *FCN_tiny* address this challenge. PolarNet is one fifth of the size of DeepLabV3+. It runs more than 2.5 times faster on Jetson TX2 while providing the best mIoU on RA and second best on RAD.

The main advantage of *FCN_tiny* against PolarNet is its smaller parameter space. *FCN_tiny* performs faster on the Jetson TX2's CPU than PolarNet but PolarNet outperforms *FCN_tiny* on the Jetson TX2's GPU. This is due to the fact that PolarNet uses larger convolution kernels that are faster to calculate on GPU than on CPU because of their vectorized computation.

5.5.4 Feature Compression

In practical applications, there are three ways to run a model. The first is to have a powerful sensor to run the whole segmentation model that outputs the results. This is usually not the case in edge applications. The second approach is to stream the raw observations to a powerful central processing machine that can run the model. However, this imposes strict requirements on the final system's bandwidth, which is limited. The third approach is to compress the raw data prior to transfer. Data compression can be approached as the simple size reduction using compression techniques, or as a summarization that provides useful features to the model. To get the most out of our processing power, it is evident that the latter approach is desirable, and the encoder-decoder architecture of PolarNet is best suited for this.

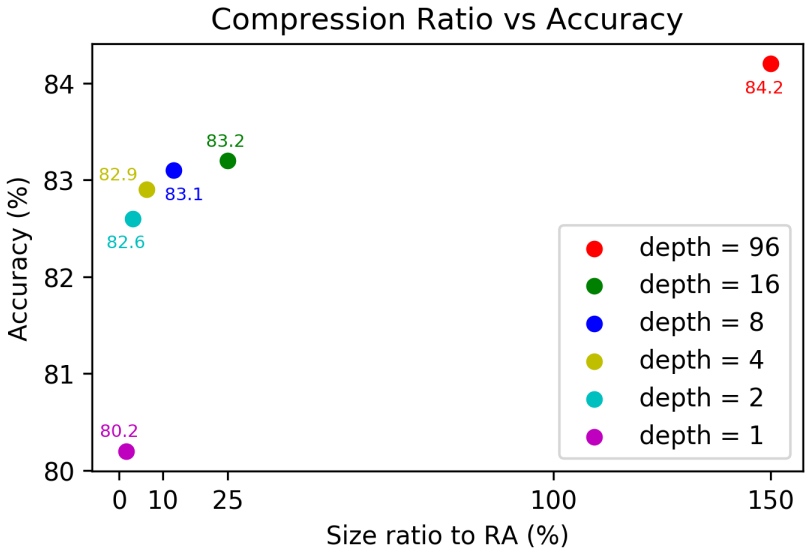


FIGURE 5.5: Encoder feature compression. Trade-off between reducing depth dimension of encoder's last layer versus the accuracy of the model.

To evaluate the compression ratio and descriptiveness of our encoder, the depth dimension of the *conv6* feature map is altered. Figure 5.5 shows the results of this experiment for various depth values. It is shown that with a small amount of sensor-side processing, we can achieve 75% reduction in the size of RA with a trade-off of only 1% in the accuracy of the system. Please note that all the other parameters of the network are identical.

5.6 Conclusion

In this chapter, we have evaluated the capabilities of an automotive radar and proposed a novel deep model to segment open spaces in parking scenarios using an automotive radar. Our model takes benefit of vectorized GPU operations and outperforms the state-of-the-art in terms of speed. Furthermore, PolarNet provides the best performance using Range-Azimuth as its input modality. These characteristics of the model make it the perfect candidate for RoC integration scenarios.

We have evaluated various representations of radar data as inputs to deep models, various models, the effect of the polar to Cartesian transformation, and the effect of three loss functions. Additionally, we have collected a dataset with ADC information. To the best of our knowledge, this is the first comprehensive Radar dataset that provides the ADC information and the toolkit to extract various representations from it. We hope this new dataset will enable an increasing number of researchers to easily access the radar data and further develop this field.

In this chapter, we only discussed single frame observations that do not employ any temporal information. In this way, the number of model parameters and computational requirements are kept as low as possible. However, we acknowledge that using temporal models would increase the performance of occupancy map predictions as the temporal information can be used to remove noise from predictions. We keep this aspect as a topic to address in our future research.

Chapter 6

In-Vehicle Occupancy Detection using Thermal Data

Counting people is a growing field of interest for researchers in recent years. In-vehicle passenger counting is an interesting problem in this domain that has several applications including High Occupancy Vehicle (HOV) lanes. In this chapter, we present a new in-vehicle thermal image dataset. We propose a tiny convolutional model to count on-board passengers and compare it to well known methods.

6.1 Introduction

HOV lanes are restricted traffic lanes that are reserved for vehicles with a certain number of passengers (passenger number varies between states and countries). HOV lanes are used to encourage carpooling and the use of public transportation. Current methods used to enforce HOV lanes include police officers being physically present and visually monitoring the HOV lanes, and penalizing the offenders. Some states encourage commuters to report HOV offending vehicles. It can hence be seen that counting the number of passengers in cars at all times is an important task in the process of enforcing HOV lanes.

Counting the number of humans present in a certain area has several applications, especially in urban environments. It can be used for congestion analysis at certain places (e.g., popularity of certain products in a supermarket, visitors of a sculpture in a museum, traffic through a certain entrance of a mall) [57]. Counting the number of passengers is an integral part of the enforcement of HOV lanes process. Automating the process of occupancy detection by installing road-side cameras on HOV lanes has not been very successful. This is mainly due to significant changes in the lighting and visibility conditions. In addition, an external camera will not be able to reliably detect all passengers, especially those in the back seats. A possible solution is to use seat sensors that determine whether or not there is a passenger sitting on a seat by measuring the

weight exerted on it. However, this approach is not feasible as there is no way to differentiate between an actual passenger and a heavy object placed on the seat. In addition, these sensors are usually installed for the front seats only.

Installing regular cameras inside a vehicle to detect the number of passengers at all times might be a feasible solution. However, this method raises privacy concerns as passengers will not be comfortable with cameras recording their activities at all times [114]. In addition, it is challenging for a visible camera to distinguish humans from the human-like dummies that have been often used to cheat the system. To address these problems, we propose the use of thermal sensors installed in vehicles to reliably detect the number of passengers using deep learning models without significantly compromising their privacy. Thermal data conceals most of the distinctive visual features, thereby the identities of passengers would remain anonymous. Further, the proposed model is capable of running on the edge-devices without the need to transmit any recordings to the server side. Our method could be used as part of a complete system where cars are registered in a database by their plate number. An embedded system would detect passengers using our model and update the number of passengers in real-time with the database accessible by authorities.

Deep learning models have dominated the field of computer vision and image processing since the introduction of AlexNet [98]. Deep models have been applied to many fields ever since including classification [98, 170, 171], recognition [135, 159], detection [145, 111], scene understanding [161], and geometric analysis [132]. In this work, we developed a neural network solution for passenger counting with thermal imaging, while taking into consideration its potential application on embedded and low-powered platforms. There are three main contributions of this chapter:

- We introduce a new, comprehensive, and annotated thermal image dataset of in-car environments.
- A custom convolutional neural model is designed that is able to surpass the classification performance of larger models, while keeping the model small enough to be suitable for embedded applications. We adapt our model to perform each of classification and detection tasks.
- We retrain the well known object detection methods on this dataset and provide a comprehensive evaluation of them on an embedded platform.

The rest of this chapter is organized as follows. Section 6.2 provides a literature review of related work. In section 6.3 we give detailed information related to our dataset. In section 6.4 we explain details of our algorithms and deep neural networks tested. Section 6.5 presents our experimentation results and conclusion is provided in section 6.6.

6.2 Related Work

Portmann *et al*. [137] detect people from aerial thermal images using a particle filter based detector and traditional local features. Authors show that the heat signature of the human body is unique in these environments. However, problems arise when the heat signature is close to the background temperature.

Olmeda *et al*. [133] provide a comprehensive study on pedestrian detection using local features and conventional machine learning methods. They found that thermal imaging is a reliable source of information, and it provides similar performance in comparison to visual images. Both of these methods are relying on the traditional features and learning methods that are widely outperformed by novel deep learning models.

Gundogdu *et al*. [74] use infrared thermal images to classify objects. They use a random forest with a depth of 2 and branching factor of 2. At each node, a Convolutional Neural Network (CNN) is trained to classify between four classes. Classification accuracy is increased by fusing these methods. However, this comes at the cost of having multiple CNN models and increasing the computational complexity of the model.

Konig *et al*. [97] propose a multi-spectral region proposal network based on Faster R-CNN [147] by fusing infrared and visual images. They show that adding BDT Classifier [202] improves the region proposal network performance even further. In contrast, we rely on training the model solely on thermal images to respect the privacy of the passengers.

Riggan *et al*. [149] learn to synthesize an image into visual domain from thermal images. It learns the mapping through a generative adversarial network [73] that includes local fiducial regions to provide more discriminative features in reconstructed images. Once the image is reconstructed, it is matched against a dataset of known visual images for recognition.

Herrmann *et al*. [82] propose using models that are previously trained on visual images and adapt them to the infrared domain. To achieve this, models are fine-tuned using infrared images. It is shown that simple preprocessing on infrared images boosts the overall performance significantly. Under this framework, inversion provides the largest benefit among different preprocessing methods. Similarly we have noticed the value of simple preprocessing techniques but with a different goal. Instead of trying to fit thermal data to exhibit a similar behavior as visual images, we extract a mask based on average human body heat signature and use it to guide the training of our model.

Laradji *et al*. [99] use deep convolutional models to detect roughly estimated regions to count individual objects in the image. Their method relies on three major stages. Feature extraction is performed using ResNet [78]. Neighborhood detection is done by utilizing specialized loss functions to encourage single blob detections. And finally, a line splits and watershed splitting methods are used to divide large blobs. In essence, this

#Passenger	0	1	2	3	4	5
#Images	65	314	402	276	253	3

TABLE 6.1: Data distribution. Number of images per passenger count is shown in this table.

method is very relevant to our detection approach. In our approach, we use gaussian-like density blobs that cover the bounding box region. Later, the generated blobs are post processed with a simple technique to generate bounding boxes. We are not using a specific loss for split learning, however the gaussian blobs provide the required basics internally to encourage easily separable density map generation. Further, unlike the work of Laradji *et al*. [99] our goal is to detect bounding boxes. And finally, our system works on thermal images rather than camera images.

6.3 Dataset

Our dataset was captured with a FLIR One Pro thermal imaging camera [1]. The camera can detect temperatures between $-20°C$ and $400°C$ with an accuracy of $\pm 3°C$. The raw recording is in $16-bit$ integer format. Dividing the raw thermal reading by 100 results in the *Kelvin* scale equivalent. The thermal image resolution is 640×480. Several locations for positioning the camera inside a vehicle were tested. The ideal location is found to be under the rear-view mirror (figure 6.1), so that it completely covers the area inside the vehicle with minimal obstruction especially to passengers in the backseat. An Android mobile application was developed using the FLIR One SDK to capture raw thermal data.

In total 1284 images were captured with the number of passengers varying between 0 and 5. Different vehicle types (SUV, Sedan, Hatchback) and passenger seating positions were adopted to provide a comprehensive dataset. Apart from changing cars and relocating passengers, air-conditioning in vehicles is used to introduce further environmental variations. Heating or cooling the vehicle interior causes a drastic shift in the distribution of the thermal values. The distribution of image classes is shown in Table 6.1.

Thermal images are manually labeled for visible portions of the passenger head and torso. The torso region is usually hidden behind the front seats. These annotations are stored in a *JSON* file that is provided with the dataset. Figure 6.2 shows a sample annotated image from our dataset.

The clothing of the passengers is a major source of variation in thermal images. They cover the body heat, and sometimes could reflect higher intensities depending on their

[1] www.flir.com

FIGURE 6.1: Thermal camera and its location for data capture.

material. The head is mostly visible and is not covered; this provides a better opportunity for detecting passengers. Therefore, the labels for the head are used in our experiments. To count passengers positioned in the vehicle, we simply count the number of detected heads.

The dataset is available to download through https://goo.gl/js1cLJ link.

6.4 Proposed Method

In this section, the data augmentation strategy and description of tested neural models are provided. We break down the proposed method in two settings and explain the parameters for each.

6.4.1 Augmentation and Preprocessing

Deep learning methods are extremely data hungry. In order to satisfy this requirement, we augment the dataset by applying a combination of rotation and scaling transformations on the raw data. See figure 6.3

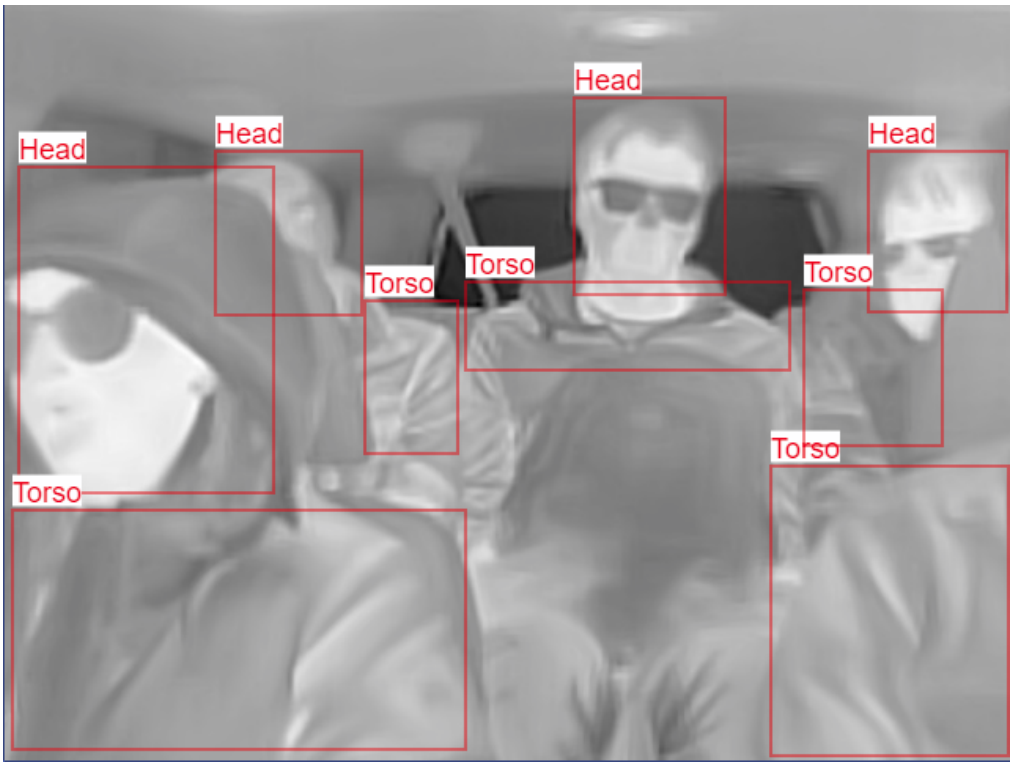


FIGURE 6.2: A sample of an annotated image.

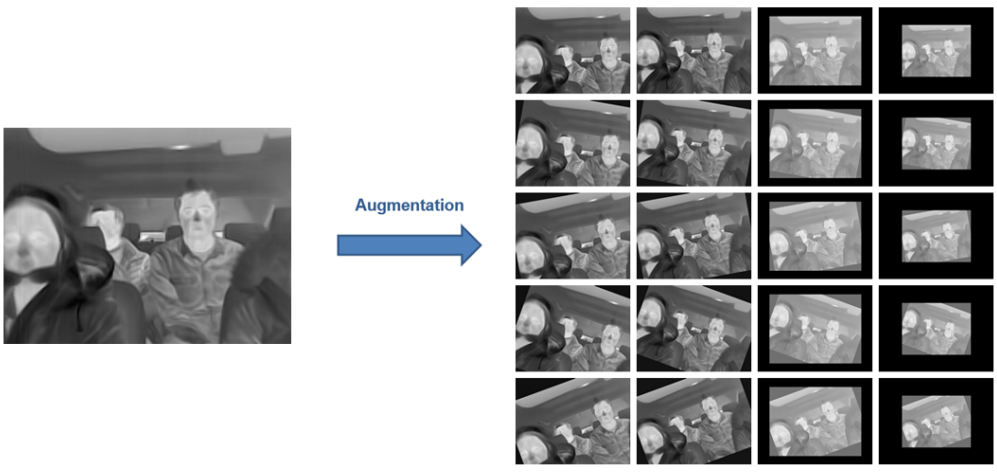


FIGURE 6.3: Original image from dataset and its augmentations.

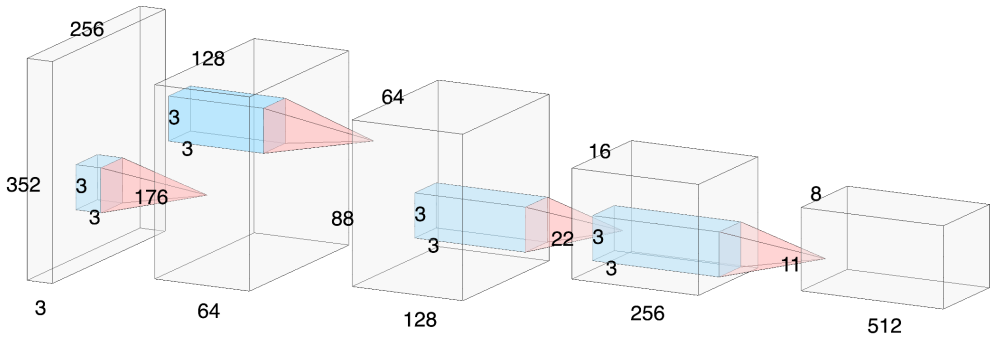


FIGURE 6.4: Proposed core model used as the backbone in all the learning tasks.

Scale. Each thermal image is rescaled with ratios of $[0.8, 1.0, 1.2, 1.4]$. These ratios are chosen to achieve invariance against variations that might rise from changing the size of the vehicle.

Rotation. Passengers in a car tilt or rotate their heads. In order to achieve invariance against this, all thermal images are transformed with rotation angles of $[-20, -10, 0, 10, 20]$ degrees.

In total, this process generates 20 augmented samples from each thermal image. This way, the final data set size is increased from 1284 to 25680 images which was found to be sufficient for training and testing neural models.

6.4.2 Core Model

We briefly introduce the neural networks and corresponding parameters that have been used for benchmarking on this dataset. Before processing images with these networks each image is resized to a smaller scale and ratios to match dimensionality of $90k$ pixels per image.

The core of the proposed method is a four layered convolutional model that is inspired by [98]. All the core convolutions are performed with 3×3 kernels and strides of 2, except the third layer that has the stride of 4 that drastically reduces the size of the feature map. This quick reduction in feature map size reduces the computational complexity of the model. First convolution consists of 64 kernels, and the number of kernels is doubled after every convolution. We propose two tasks on top of our core model. One is classification that is termed *C4S-C*, and the other is multi-task learning for classification and bounding box detection denoted as *C4S-CD*. The core model is shown in Figure 6.4, and all the models and definitions are provided on *github*[2].

[2] www.github.com/erlikn/sm_ps_ct.git

6.4.3 Classification Task

We redefine the counting problem as a classification problem and use the number of passengers as the class labels for each image. An input image of size 352×256 is passed to our proposed four layered convolutional network to extract the feature maps. At layer five, instead of flattening the 22×16 feature map and using a fully connected layer, we flatten the maps using a single branch factorized convolution [172] with one 1×1, two 3×3, and two consecutive groups of 1×3 row-wise and 3×1 column-wise convolutions. This further improves the performance compared to using a fully connected layer and reduces the model size by 20%. First convolution at this layer employs a 1×1 kernel with a stride of 1 followed by two 3×3 convolutions with a stride of 2. Finally, row and column-wise convolutions are applied. Drop-out [166] with a keep rate of 0.5 is applied on extracted feature maps from the core convolutions prior to passing them to the factorized convolution layer. The flattened image is passed to fully connected layers of size 256 and output layer of 6 representing the class labels. After calculating the features at each layer, Rectified Linear Units (ReLU) and batch normalization [88] are used. We have evaluated various loss functions including softmax cross-entropy, online hard example mining (OHEM) [163] with softmax cross-entropy, and focal loss functions [109]. We define the loss function by OHEM loss as it provides a few percentage points better classification accuracy than the regular softmax cross-entropy function. The concept of OHEM relies on taking the top n softmax cross-entropy loss from a batch of images. In other words, the hardest samples to classify are used when calculating the total loss for back-propagation. In our implementation, we set n to be half of the batch size. Momentum optimizer with the initial learning rate of 0.01, momentum value of 0.9 and decay factor of 0.1 after every 5000 steps is utilized as the optimizer for this model.

6.4.4 Multi-Task Learning

To perform bounding box detection, we use a similar method to [99]. Instead of using a single pixel label, we use a gaussian masking on coarser sized image to perform bounding box detection. The output of the core convolutional layers at one side are fed into a classification model similar to *C4S-C*, and on the other side is passed to deconvolutional layers. We call this model *C4S-CD*. The deconvolutions upsample the image by strides of 2. The first one has a feature map of size 512 and the last one produces the predicted heatmap. To calculate the loss, we resize the input image to the shape of 44×32 and create a mask that represents the head of each detected passenger using an intensity gaussian distribution with σ set at one fifth of the ground truth bounding box size. $L2$ function is used to calculate the loss for the predicted and target heatmaps.

Generating bounding boxes from the heatmap requires further attention. Close objects have a tendency to be merged with each other. Using gaussian masking instead of

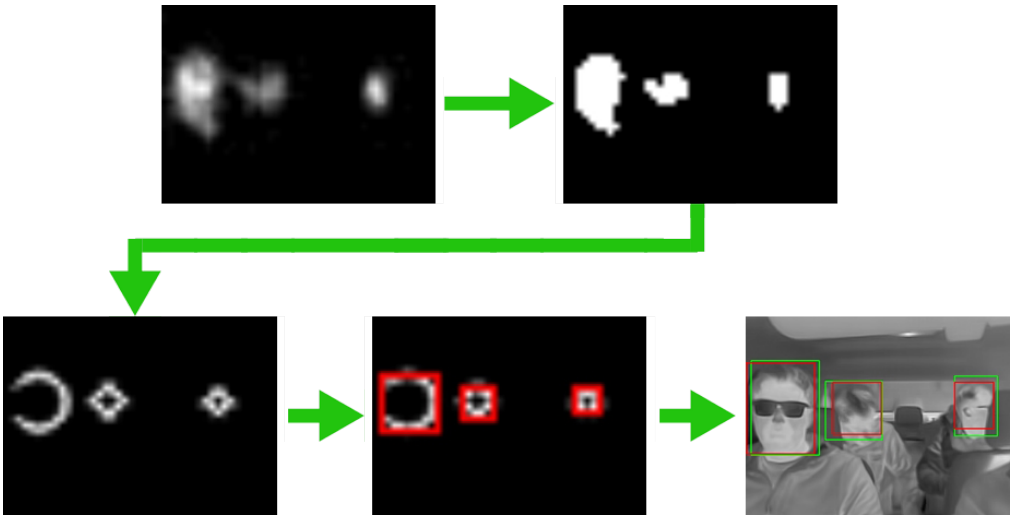


FIGURE 6.5: Stages of splitting the heatmap to connected regions and applying blob detection to get the enclosing bounding boxes.

the binary masking in the learning process has addressed this problem to some extent. The boundaries of the heads in the output heatmap contain smaller values compared to the center. We employ a simple thresholding mechanism to further split the connected regions of multiple targets. The thresholding value is set to 40 in our tests to create the binary masks. Once the regions are splitted we use the blob (binary large object) detection from the OpenCV library [22] to detect the connected regions in the binary heat map. Since our network also outputs the number of passengers n in the vehicle, we sort the detected blobs by size and take the n largest blobs as the predicted head locations. This way we eliminate noisy masked clouds from the data. Once we generate a list of accepted blobs, enclosing bounding boxes are generated for them. The boxes are then rescaled to the original image representing the locations of the detected heads. Figure 6.5 shows the various steps of generating bounding boxes from the output heat maps. Since our passenger counts are resulted from the classification branch of the *C4S-CD*, we only report precision/recall value for the heatmap based detections.

6.5 Experiments

We employ n-fold cross-validation to remove any bias towards any dataset distribution from experiments. To perform these experiments, four folds have been employed. In each fold, the dataset is divided into two mutually exclusive sets of train and test. For training, we have used 90% of the raw data and then augmented them with the methods proposed in section 6.4.1.

The remaining 10% of images are used as test images. We do not augment the test dataset. There are two reasons behind this decision. The first is that testing augmentations of the same image do not contribute much in reflecting the accuracy of detection models. An image and its augmentations most likely have the same detection results. Augmentation enriches the train set and makes it more robust, but it does not add variety to the testing set. The second reason is that we aim to get results on images that would reflect real situations.

In classification, we compare our model to Mobilenet. In the detection task, we use SSD [111] once on top of Mobilenet and once with Inception V2.

SSD-Inception. SSD [111] is another network that uses a single convolutional neural network for object detection. It is composed of the VGG classification network truncated before any classification layers and replaced by 6 convolutional layers. The 6 final layers and respective anchor box scales gradually decrease in size, allowing for the detection of objects of different scales. Szegedy *et al*. [171] introduced the inception model for classification. Instead of using one kernel of fixed size for a convolution, the inception model applies 3 filters of sizes 1×1, 3×3, and 5×5, in addition to a max-pooling operation. Later, this has been updated by factorization to provide faster computation [170]. Results of these operations are concatenated to form the final output of an inception layer. Since the inception model learns more parameters, we have decided to test its performance as the base layer for SSD.

SSD-MobileNet. Howard *et al*. [84] introduced Mobilenets, that are ideal for mobile and embedded systems applications. Instead of standard convolutions, Mobilenets use a combination of depthwise convolutions followed by 1×1 pointwise convolutions for optimization. Depthwise convolutions apply a single filter to each input channel. Pointwise convolutions are then used to combine the outputs of the depthwise convolution. This architecture separates the filtering and combining operations and drastically reduces the model size and computational requirements, while not significantly compromising the model accuracy. Mobilenet is composed of 19 layers (depthwise then pointwise convolutions) and a fully connected layer fed into a softmax layer for classification. While inception networks as the base for SSD generate more robust feature maps, this comes at a heavy size and computational cost. Since our goal is to have a model that is capable of running on limited power and memory devices, model size and computation requirements are vital. We tested the use of Mobilenet truncated before the classification layers as a base model for SSD.

To compare these models against each other, accuracy, speed, and precision-recall measures are used.

Classification		
Method	Counting Accuracy	Binary Accuracy
Mobilenet	84.54%	96.67%
C4S-C (SMCE)	89.70%	97.24%
C4S-C (OHEM)	**91.03**%	98.0%
Detection		
Method	Counting Accuracy	Binary Accuracy
SSD-MobileNet	92.56%	98.67%
SSD-Inception	**95.04**%	99%
SSD-C4S-C	84.73%	97%

TABLE 6.2: Comparison of counting accuracy. Counting based on classification and detection results are presented at the top and bottom of this figure, respectively.

6.5.1 Counting Accuracy

In the task of counting the number of passengers in a car, we define accuracy as the number of images for which the head count is correctly computed given the test dataset. At this point, the measure is agnostic of the passenger locations, and only the final count is valuable. Table 6.2 shows the best accuracies.

C4S-C provides a significantly better classification based counting accuracy than Mobilenet in both cases. OHEM loss introduces more weight on hard examples hence could generalize better on the test set. For HOV lanes, it is of utmost importance to detect if there are two or more passengers in the vehicle. To address this, we introduce the binary accuracy. Figure 6.6 presents the class confusion matrices for various approaches. Through confusion matrix analysis, it is concluded that the majority of misclassifications occur between adjacent numbered classes. Therefore the binary accuracy for all the models is usually larger than their exact counting accuracy.

6.5.2 Precision-Recall

SSD-MobileNet and SSD-Inception are designed to localize the objects in the image. We perform the object detection test in order to compare the effectiveness of the models in identifying the location of objects in the image. We can not solely rely on precision-recall or accuracy in order to identify the better performing method. We could have a method with high accuracy, but with a lower precision due to the fact that all the false positives happened in a few images. Or inversely, we could have a higher precision or recall but lower accuracy, due to the spread of false positives or false negatives between multiple images.

Figure 6.7 shows precision vs recall curve for the methods with confidence thresholds ranging from $[0.05, 0.9]$ and intersection of unions (IOU) threshold set to 0.5. We

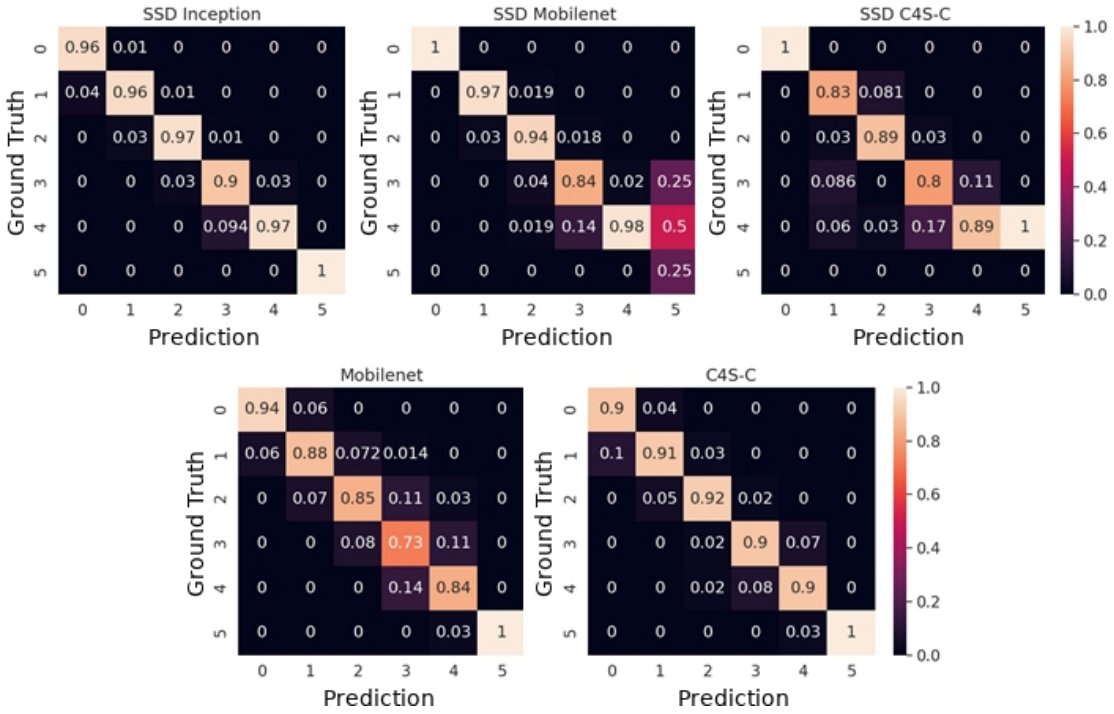


FIGURE 6.6: Comparison of confusion matrices. Top row shows the detection models. Bottom row shows the classification models.

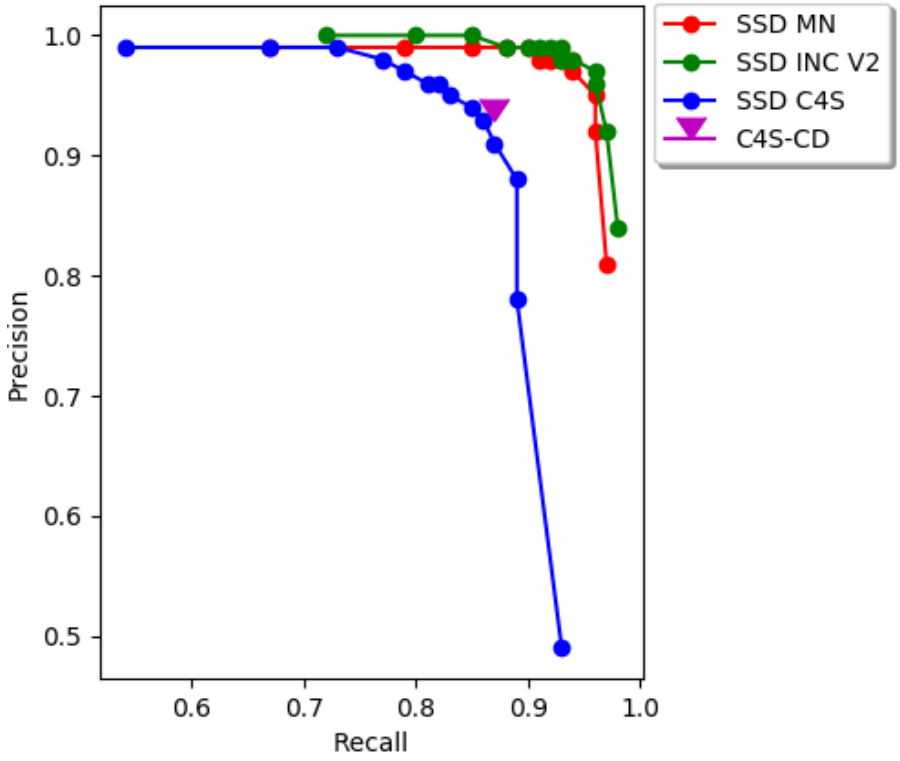


FIGURE 6.7: Precision/Recall comparison for IOU of 0.5.

	Mobilenet		**C4S-C**	
Jetson TX2	ms	FPS	ms	FPS
GPU	27	37	15.9	63
CPU	600	1.7	369	2.7

TABLE 6.3: Speed comparison for classification models

use the tensorflow object detection API [86] to compare our model performance with pre-existing methods in object detection using SSD. Huang *et al*. [86] show that Inception V2 [170] achieves the best performance in terms of mean average precision (mAP) compared to other deep CNN methods such as [78] when used as a backbone for SSD [111]. They also show that Mobilenet [84] does not lag too much behind Inception V2 in terms of mAP while having a lower processing time. This is also confirmed in our tests. Mobilenet with worse classification performance performs better in object detection task using SSD. There are better classification models such as InceptionResNet V2 [170] that perform poorly with SSD, however they provide good results with other detectors such as Faster RCNN [147]. This is due to the fact that they are tuned to function effectively with their respective detectors. Similarly, Mobilenet performs slightly worse and much slower with Faster RCNN compared to SSD. Our model is suffering from this issue. Specially, the rapid reduction of the feature map dimensions results in such features that limits the capabilities of SSD.

To alleviate this problem we use the *C4S-CD*. The deconvolution and blob detection modules add a negligible overhead to the system. However, it produces better results than SSD, it is still lagging behind the other methods.

6.5.3 Speed

Finally, to decide on which method is more appropriate for embedded platforms, we compare the execution performance. All of the methods are implemented in *Tensorflow* [2] and are benchmarked on a *Nvidia Jetson TX2* platform. The Jetson TX2 is one of the fastest and most power-efficient computing devices developed for AI embedded systems applications. Although it does not compare to the computing power of regular GPUs, it has a decent performance which makes it a better fit for embedded systems. Since this solution would be used in vehicles, the final goal is to embed the model on an edge device with much lower available power than Jetson TX2. However, Jetson TX2 would provide a better perspective over the performance of each of the models.

The proposed *C4S-C* outperforms all the others in this case with a whopping 63 frames per second processing speed on the Jetson TX2's GPU. Tables 6.3 and 6.4 show the speed comparison of classification and detection methods respectively.

	SSD Mobilenet V2		**SSD Inception V2**		**C4S-CD**	
Jetson TX2	ms	FPS	ms	FPS	ms	FPS
GPU	77	13	95	10.5	19.5	51.3
CPU	500	2	1210	0.8	373	2.7

TABLE 6.4: Speed comparison for detection models

The fast training and testing speed of our proposed network allowed us to perform a comprehensive network parameter tuning in order to choose the best fitting settings for the task in hand. This is an obvious advantage of small and fast models against the large ones.

6.6 Conclusion

In this chapter, we have introduced a new dataset for counting the number of passengers in the car that is collected using a thermal camera. We introduce a data augmentation model to increase the amount of data and build models that are robust against variations such as rotation and scale. We propose two models based on one core architecture for classification and detection tasks. The classification model outperforms the state-of-the-art with a comfortable margin with almost half the computational complexity. Further, we have compared the results of various object detection models, and proposed a new method based on blob detection to detect passengers. Our proposed models are designed with the main consideration to run on a low powered edge device. This results in having a very small and fast model that is comparable in performance to the state-of-the-art. One aspect that we have not explored in this chapter is the techniques to prune and quantize our models. This will further optimize our model to comfortably run on a low powered device.

Chapter 7

Conclusion

In this thesis, we explored the utilization of deep learning models in various autonomous driving applications. We chose problems to which deep models have not previously been applied or were unable to generalize. Over different chapters, we cover the sensor suite used in various autonomous and driver assistance systems.

In **Chapter 3**, we proposed a convolutional neural network for estimating camera based homography. The model uses two images as inputs to a siamese feature extraction network. Extracted features of the inputs are merged and, through further convolutions, the final homography is estimated. Using this estimate, one of the inputs is warped and used in the second iteration with the same model. We showed that deep neural networks are capable of providing superior estimates by extracting more descriptive image features in successive iterations. In the proposed architecture, twin models were unaware of each other until the merge function. This is a hard-coded matching layer. Switching to architectures that incorporate matching in multiple layers [29] could enhance the performance of estimations. Replacing the merge function with a correlation model [87] could result in more descriptive features for homography estimation. Furthermore, augmenting our proposal with semantic information could provide robustness against outliers such as dynamic objects. The scene semantics could be used to identify surface planes in the scene and estimate multi-plane homographies.

In **Chapter 4**, we used point net layers in our architecture to extract features from point-cloud observations collected via a lidar sensor in driving scenarios. These features are then matched through a correlation layer and more descriptive features are extracted. Finally, a fully connected layer is used to estimate the odometry parameters. Similar to Chapter 3, a hierarchy of this model is used to refine the estimates at each iteration. We proposed an augmentation model to handle the severe data imbalance. Moreover, we suggested using a simple thresholding technique to process ground and non-ground points separately. It was observed that the performance of feature extraction networks for point-clouds are not as mature as their image based counterparts. Switching the feature extraction backbone to a model with lower complexity and better descriptive power can provide better estimates [205, 194, 85]. This further enables

merging the temporal model to the registration model under one unified approach [115]. Employing unsupervised learning models [207] can enable usage of various large scale datasets without any labeling requirements. These aspects are open topics that will be explored later.

Chapter 5 introduces a new radar dataset and a novel deep model to perform open space segmentation on this data. PolarNet replaces the traditional approaches of finding obstacles in radar observations with a deep model solution. Range-Azimuth representation of the radar signal is processed through a series of one dimensional column-wise layers. These layers take advantage of polar representation of the data and apply spatially consistent filters. Later, row-wise and 2D kernels are used to reduce the map size while adding more descriptive power to the features. PolarNet achieves state-of-the-art performance while ensuring fast execution times on limited resources. The proposed model focuses on single frame open-space segmentation. However, radar data contains large amounts of noise that change erratically from frame-to-frame. Employing temporal models would substantially help the network in learning the characteristics of noise and improve the accuracy.

Thermal sensors are used in **Chapter 6** to collect a dataset of passengers in a vehicle, with the goal of counting the number of occupants. The proposed deep model is designed with embedded platforms in mind and is capable of achieving state-of-the-art performance. Furthermore, as a light-weight alternative to bounding box detection models, we use a segmentation head that produces a heatmap of the occupants' locations. The bounding boxes are localized using the classification result and a simple clustering method. The pruning and quantization techniques could be used to further reduce model size and achieve lower computational requirements [80, 42, 90].

As the no free lunch theorem [188] explains, there is no single method that can outperform every other method in all of the problems. Deep learning models are no exception to this theorem. The simple problem of counting sequential numbers in a series is an example that neural models have a hard time adapting to. Deep neural models can be crudely explained as an alternative method of implementing Support Vector Machines with advanced kernel tricks. These kernels are learned in the process of training. As such, neural models can incorporate drastic changes into their architecture and high-dimensional feature space to achieve an ultimate power against traditional methods. Further, they provide a sleeker approach to train their parameters that is well suited for parallel processing. However, this comes at the expense of immense data requirements.

Large data is continually becoming more available in all the various domains of autonomous driving. This ensures that, for a foreseeable future, deep models will be popular in this field.

Achieving the reality of autonomous vehicles is still far from reach. The high risk nature of the task requires extreme fault-tolerance and high safety standards that are yet

to be guaranteed. Pareto's principle entails that the final percentage points in reaching such safety standards will require much more time than what has already been spent in this field. Further, these technological advances will have a major effect on society and will require shifts in our life-style. To enable this transition, significant changes in regulations and city planning for the future has to be prepared as well. Needless to say, achieving this level of autonomy will change our lives forever, and this thesis is only a tiny step in this direction.

Bibliography

[1] *5G Automotive Vision*. https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Automotive-Vertical-Sectors.pdf. The 5G Public-Private Partnership (5GPP), 2015.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[3] *Accelerating the Future: The Economic Impact of the Emerging Passenger Economy*. https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/05/passenger-economy.pdf. Intel, 2017.

[4] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. "Building rome in a day". In: *Communications of the ACM*. Vol. 54. 10. ACM New York, NY, USA, 2011, pp. 105–112.

[5] Pablo F Alcantarilla. "Fast explicit diffusion for accelerated features in nonlinear scale spaces". In: *British Machine Vision Conference (BMVC)*. 2011.

[6] David Alexandre, Chih-Peng Chang, Wen-Hsiao Peng, and Hsueh-Ming Hang. "An Autoencoder-based Learned Image Compressor: Description of Challenge Proposal by NCTU." In: *CVPR Workshops*. 2018, pp. 2539–2542.

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875*. 2017.

[8] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions". In: *Journal of the ACM (JACM)*. Vol. 45. 6. ACM New York, NY, USA, 1998, pp. 891–923.

[9] *Automated Driving: Level of Driving Automation*. https://web.archive.org/web/20170903105244/https://www.sae.org/misc/pdfs/automated_driving.pdf. SAE International, 2014.

[10] Hernán Badino, Uwe Franke, and David Pfeiffer. "The stixel world-a compact medium level representation of the 3d-world". In: *Joint Pattern Recognition Symposium*. Springer. 2009, pp. 51–60.

[11] Hernán Badino, Akihiro Yamamoto, and Takeo Kanade. "Visual odometry by multi-frame feature integration". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013, pp. 222–229.

[12] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 39. Institute of Electrical and Electronics Engineers (IEEE), 2017, 2481–2495.

[13] Dan Barnes, Matthew Gadd, Paul Murcutt, Paul Newman, and Ingmar Posner. "The oxford radar robotcar dataset: A radar extension to the oxford robotcar dataset". In: *arXiv preprint arXiv:1909.01300*. 2019.

[14] Daniel Bauer, Lars Kuhnert, and Lutz Eckstein. "Deep, spatially coherent Inverse Sensor Models with Uncertainty Incorporation using the evidential Framework". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019. ISBN: 9781728105604.

[15] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *European conference on computer vision*. Springer. 2006.

[16] Jens Behley and Cyrill Stachniss. "Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments." In: *Robotics: Science and Systems*. 2018.

[17] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. "3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks". In: *IEEE Robotics and Automation Letters*. Vol. 3. 4. IEEE, 2018, pp. 3145–3152.

[18] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. "The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4413–4421.

[19] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

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

[21] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. "Generating sentences from a continuous space". In: *arXiv preprint arXiv:1511.06349*. 2015.

[22] Gary Bradski. “The OpenCV Library”. In: *Dr. Dobb's Journal of Software Tools*. 2000.

[23] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. “Geometry-Aware Learning of Maps for Camera Localization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2616–2625.

[24] Stefan Braun. “LSTM benchmarks for deep learning frameworks”. In: *arXiv preprint arXiv:1806.01818*. 2018.

[25] Thomas M Breuel. “Benchmarking of LSTM networks”. In: *arXiv preprint arXiv:1508.02774*. 2015.

[26] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. “Signature verification using a "siamese" time delay neural network”. In: *Advances in neural information processing systems*. 1994, pp. 737–744.

[27] Matthew Brown and David G Lowe. “Automatic panoramic image stitching using invariant features”. In: *International journal of computer vision*. Vol. 74. 1. Springer, 2007, pp. 59–73.

[28] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. “NuScenes: A multimodal dataset for autonomous driving”. In: *arXiv preprint arXiv:1903.11027*. 2019.

[29] Luca Caltagirone, Mauro Bellone, Lennart Svensson, and Mattias Wahde. “LIDAR–camera fusion for road detection using fully convolutional neural networks”. In: *Robotics and Autonomous Systems*. Vol. 111. Elsevier, 2019, pp. 125–131.

[30] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. “Human pose estimation with iterative error feedback”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4733–4742.

[31] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research*. Vol. 16. 2002, pp. 321–357.

[32] Changhao Chen, Chris Xiaoxuan Lu, Bing Wang, Niki Trigoni, and Andrew Markham. “DynaNet: Neural Kalman Dynamical Model for Motion Estimation and Prediction”. In: *arXiv preprint arXiv:1908.03918*. 2019.

[33] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 40. Institute of Electrical and Electronics Engineers (IEEE), 2018, 834–848.

[34] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Semantic image segmentation with deep convolutional nets and fully connected crfs". In: *arXiv preprint arXiv:1412.7062*. 2014.

[35] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587*. 2017.

[36] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2018, 833–851.

[37] Steven W Chen, Guilherme V Nardari, Elijah S Lee, Chao Qu, Xu Liu, Roseli Ap Francelin Romero, and Vijay Kumar. "SLOAM: Semantic Lidar Odometry and Mapping for Forest Inventory". In: *IEEE Robotics and Automation Letters*. Vol. 5. 2. IEEE, 2020, pp. 612–619.

[38] Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. "Variational lossy autoencoder". In: *arXiv preprint arXiv:1611.02731*. 2016.

[39] Zetao Chen, Adam Jacobson, Niko Sünderhauf, Ben Upcroft, Lingqiao Liu, Chunhua Shen, Ian Reid, and Michael Milford. "Deep learning features at scale for visual place recognition". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 3223–3230.

[40] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*. 2014.

[41] Younggun Cho, Giseop Kim, and Ayoung Kim. "DeepLO: Geometry-Aware Deep LiDAR Odometry". In: *arXiv preprint arXiv:1902.10562*. 2019.

[42] Yoojin Choi, Jihwan Choi, Mostafa El-Khamy, and Jungwon Lee. "Data-Free Network Quantization With Adversarial Knowledge Distillation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 710–711.

[43] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8789–8797.

[44] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 539–546.

[45] Jan Chorowski, Ron J Weiss, Samy Bengio, and Aäron van den Oord. "Unsupervised speech representation learning using wavenet autoencoders". In: *IEEE/ACM transactions on audio, speech, and language processing*. Vol. 27. 12. IEEE, 2019, pp. 2041–2053.

[46] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555*. 2014.

[47] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289*. 2015.

[48] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[49] Marius Cordts, Timo Rehfeld, Lukas Schneider, David Pfeiffer, Markus Enzweiler, Stefan Roth, Marc Pollefeys, and Uwe Franke. "The stixel world: A medium-level representation of traffic scenes". In: *Image and Vision Computing*. Vol. 68. Elsevier, 2017, pp. 40–52.

[50] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning*. Springer, 1995.

[51] *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812506. National Highway Traffic Safety Administration, 2018.

[52] Mark Cummins and Paul Newman. "Appearance-only SLAM at large scale with FAB-MAP 2.0". In: *The International Journal of Robotics Research*. SAGE Publications Sage UK: London, England, 2011, pp. 1100–1123.

[53] Igor Cvišić and Ivan Petrović. "Stereo odometry based on careful feature selection and tracking". In: *2015 European Conference on Mobile Robots (ECMR)*. IEEE. 2015, pp. 1–6.

[54] Shubham Dash, Giridharan Kumaravelu, Vijayakrishna Naganoor, Suraj Kiran Raman, Aditya Ramesh, and Honglak Lee. "CompressNet: Generative Compression at Extremely Low Bitrates". In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2020, pp. 2314–2322.

[55] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. "Deep image homography estimation". In: *arXiv preprint arXiv:1606.03798*. 2016.

[56] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. "Rigid scene flow for 3d lidar scans". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 1765–1770.

[57] Han Ding, Jinsong Han, Alex X Liu, Wei Xi, Jizhong Zhao, Panlong Yang, and Zhiping Jiang. "Counting human objects using backscattered radio frequency signals". In: *IEEE Transactions on Mobile Computing*. Vol. 18. 5. IEEE, 2018, pp. 1054–1067.

[58] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[59] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research*. 2011.

[60] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. "The farthest point strategy for progressive image sampling". In: *IEEE Transactions on Image Processing*. Vol. 6. 9. IEEE, 1997, pp. 1305–1315.

[61] Jakob Engel, Vladlen Koltun, and Daniel Cremers. "Direct sparse odometry". In: *IEEE transactions on pattern analysis and machine intelligence*. Vol. 40. 3. IEEE, 2017, pp. 611–625.

[62] Jakob Engel, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-scale direct monocular SLAM". In: *European Conference on Computer Vision*. Springer. 2014, pp. 834–849.

[63] Alfonso Farina and Francisco A Studer. "A review of CFAR detection techniques in radar systems". In: *Microwave Journal*. Vol. 29. 1986, p. 115.

[64] Li Fei-Fei, Rob Fergus, and Pietro Perona. "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories". In: *2004 conference on computer vision and pattern recognition workshop*. IEEE. 2004, pp. 178–178.

[65] Li Fei-Fei and Pietro Perona. "A bayesian hierarchical model for learning natural scene categories". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. IEEE. 2005, pp. 524–531.

[66] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick Van der Smagt, Daniel Cremers, and Thomas Brox. "Flownet: Learning optical flow with convolutional networks". In: *arXiv preprint arXiv:1504.06852*. 2015.

[67] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM*. 1981, pp. 381–395.

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

[69] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[70] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[71] *Global Status Report on Road Saftey*. http://apps.who.int/iris/bitstream/10665/189242/1/9789241565066_eng.pdf. World Health Organization, 2015.

[72] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[73] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[74] Erhan Gundogdu, Aykut Koc, and A Aydın Alatan. "Object classification in infrared images using deep representations". In: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2016, pp. 1066–1070.

[75] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. "Deep learning for 3d point clouds: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. IEEE, 2020.

[76] Richard Hartley and Andrew Zisserman. "Multiple view geometry in computer vision". In: Cambridge university press, 2003.

[77] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[78] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[79] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: *IEEE transactions on pattern analysis and machine intelligence*. Vol. 37. 9. IEEE, 2015, pp. 1904–1916.

[80] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. "Filter pruning via geometric median for deep convolutional neural networks acceleration". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4340–4349.

[81] Dan Hendrycks and Kevin Gimpel. "Bridging nonlinearities and stochastic regularizers with Gaussian error linear units". In: *arXiv preprint arXiv:1606.08415*. 2016.

[82] Christian Herrmann, Miriam Ruf, and Jürgen Beyerer. "CNN-based thermal infrared person detection by domain adaptation". In: *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*. Vol. 10643. International Society for Optics and Photonics. 2018, p. 1064308.

[83] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation*. 1997, pp. 1735–1780.

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

[85] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. "Pointwise convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 984–993.

[86] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. "Speed/accuracy trade-offs for modern convolutional object detectors". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7310–7311.

[87] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. "Flownet 2.0: Evolution of optical flow estimation with deep networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2462–2470.

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

[89] *ISO 26262: Road Vehicles - Functional Safety*. The International Organization for Standardization (ISO), 2011.

[90] Qing Jin, Linjie Yang, and Zhenyu Liao. "Adabits: Neural network quantization with adaptive bit-widths". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2146–2156.

[91] Takeo Kanade, Chuck Thorpe, and William Whittaker. "Autonomous Land Vehicle Project at CMU". In: *Proceedings of the 1986 ACM Fourteenth Annual Conference on Computer Science*. Cincinnati, Ohio, USA: ACM, 1986, pp. 71–80.

[92] Andrej Karpathy. *Unique mentions of deep learning frameworks in arxiv papers*. https://twitter.com/karpathy/status/972295865187512320.

[93] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[94] Alex Kendall, Matthew Grimes, and Roberto Cipolla. "Posenet: A convolutional network for real-time 6-dof camera relocalization". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2938–2946.

[95] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*. 2014.

[96] Bernd Kitt, Andreas Geiger, and Henning Lategahn. "Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme". In: *2010 ieee intelligent vehicles symposium*. IEEE. 2010, pp. 486–492.

[97] Daniel Konig, Michael Adam, Christian Jarvers, Georg Layher, Heiko Neumann, and Michael Teutsch. "Fully convolutional region proposal networks for multispectral person detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 49–56.

[98] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[99] Issam H Laradji, Negar Rostamzadeh, Pedro O Pinheiro, David Vazquez, and Mark Schmidt. "Where are the blobs: Counting by localization with point supervision". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 547–562.

[100] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. IEEE. 2006, pp. 2169–2178.

[101] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature*. Vol. 521. 7553. Nature Publishing Group, 2015, pp. 436–444.

[102] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[103] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. "BRISK: Binary robust invariant scalable keypoints". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011.

[104] Dan Levi, Noa Garnett, Ethan Fetaya, and Israel Herzlyia. "StixelNet: A Deep Convolutional Network for Obstacle Detection and Road Segmentation." In: *BMVC*. 2015, pp. 109–1.

[105] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. "LO-Net: Deep Real-time Lidar Odometry". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8473–8482.

[106] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. "Pointcnn: Convolution on x-transformed points". In: *Advances in neural information processing systems*. 2018, pp. 820–830.

[107] Jiarong Lin and Fu Zhang. "Loam_livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV". In: *arXiv preprint arXiv:1909.06700*. 2019.

[108] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[109] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[110] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[111] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. "SSD: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[112] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. "Flownet3d: Learning scene flow in 3d point clouds". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 529–537.

[113] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision*. Springer, 2004.

[114] Dawei Luo, Jianbo Lu, and Gang Guo. "An indirect occupancy detection and occupant counting system using motion sensors". In: 2017.

[115] Yecheng Lyu and Xinming Huang. "Road Segmentation Using CNN with GRU". In: *arXiv preprint arXiv:1804.05164*. 2018.

[116] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. Internation Conference on Machine Learning*. 2013.

[117] Daniel Maturana and Sebastian Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 922–928.

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

[119] Krystian Mikolajczyk and Cordelia Schmid. "A performance evaluation of local descriptors". In: *IEEE transactions on pattern analysis and machine intelligence*. IEEE, 2005, pp. 1615–1630.

[120] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*. 2013.

[121] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. "Image segmentation using deep learning: A survey". In: *arXiv preprint arXiv:2001.05566*. 2020.

[122] Gary Minkler and Jing Minkler. "CFAR: the principles of automatic radar detection in clutter". In: *NASA STI/Recon Technical Report A*. Vol. 90. 1990.

[123] Pierre Moulon, Pascal Monasse, and Renaud Marlet. "Adaptive structure from motion with a contrario model estimation". In: *Asian Conference on Computer Vision*. Springer. 2012, pp. 257–270.

[124] Marius Muja and David G Lowe. "Fast approximate nearest neighbors with automatic algorithm configuration." In: *VISAPP (1)*. Vol. 2. 331-340. 2009, p. 2.

[125] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system". In: *IEEE Transactions on Robotics*. IEEE, 2015, pp. 1147–1163.

[126] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010.

[127] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. "DTAM: Dense tracking and mapping in real-time". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2320–2327.

[128] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. "Large-scale image retrieval with attentive deep local features". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3456–3465.

[129] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. "Learning deconvolution network for semantic segmentation". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1520–1528.

[130] Farzan E Nowruzi, Wassim A El Ahmar, Robert Laganiere, and Amir H Ghods. "In-vehicle occupancy detection with convolutional networks on thermal images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 0–0.

[131] Farzan E Nowruzi, Dhanvin Kolhatkar, Prince Kapoor, Elnaz J. Heravi, Robert Laganiere, Julien Rebut, and Waqas Malik. "Deep Open Space Segmentation using Automotive Radar". In: *International Conference on Microwaves for Intelligent Mobility*. 2020.

[132] Farzan E Nowruzi, Robert Laganiere, and Nathalie Japkowicz. "Homography estimation from image pairs with hierarchical convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 913–920.

[133] Daniel Olmeda, Cristiano Premebida, Urbano Nunes, Jose Maria Armingol, and Arturo de la Escalera. "Pedestrian detection in far infrared images". In: *Integrated Computer-Aided Engineering*. Vol. 20. 4. IOS Press, 2013, pp. 347–360.

[134] Junting Pan, Cristian Canton Ferrer, Kevin McGuinness, Noel E O'Connor, Jordi Torres, Elisa Sayrol, and Xavier Giro-i Nieto. "Salgan: Visual saliency prediction with generative adversarial networks". In: *arXiv preprint arXiv:1701.01081*. 2017.

[135] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. "Deep face recognition". In: British Machine Vision Association, 2015.

[136] Dabal Pedamonti. "Comparison of non-linear activation functions for deep neural networks on MNIST classification task". In: *arXiv preprint arXiv:1804.02763*. 2018.

[137] Jan Portmann, Simon Lynen, Margarita Chli, and Roland Siegwart. "People detection and tracking from aerial thermal views". In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 1794–1800.

[138] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[139] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. "Volumetric and multi-view cnns for object classification on 3d data". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5648–5656.

[140] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems*. 2017, pp. 5099–5108.

[141] *Quick Facts 2016*. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812451. National Highway Traffic Safety Administration, 2018.

[142] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434*. 2015.

[143] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". In: *arXiv preprint arXiv:1710.05941*. 2017.

[144] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[145] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[146] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767*. 2018.

[147] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[148] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. "Deepmatching: Hierarchical deformable dense matching". In: *International Journal of Computer Vision*. Vol. 120. 3. Springer, 2016, pp. 300–323.

[149] Benjamin S Riggan, Nathaniel J Short, and Shuowen Hu. "Thermal to visible synthesis of face images using multiple regions". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 30–38.

[150] *Road traffic injuries*. http://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries. Workd Health Organization, 2018.

[151] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 2015, 234–241.

[152] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF". In: *Computer Vision (ICCV), 2011 IEEE international conference on*. 2011.

[153] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision*. Vol. 115. 3. Springer, 2015, pp. 211–252.

[154] Tim Salimans and Durk P Kingma. "Weight normalization: A simple reparameterization to accelerate training of deep neural networks". In: *Advances in neural information processing systems*. 2016, pp. 901–909.

[155] Torsten Sattler, Will Maddern, Carl Toft, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, et al. "Benchmarking 6dof outdoor visual localization in changing conditions". In: *Proc. CVPR*. Vol. 1. 2018.

[156] Ashutosh Saxena, Jamie Schulte, Andrew Y Ng, et al. "Depth Estimation Using Monocular and Stereo Cues." In: *IJCAI*. Vol. 7. 2007, pp. 2197–2203.

[157] Robert E Schapire. "The boosting approach to machine learning: An overview". In: *Nonlinear estimation and classification*. Springer, 2003, pp. 149–171.

[158] Lukas Schneider, Marius Cordts, Timo Rehfeld, David Pfeiffer, Markus Enzweiler, Uwe Franke, Marc Pollefeys, and Stefan Roth. "Semantic stixels: Depth is not enough". In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2016, pp. 110–117.

[159] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[160] Alexander G Schwing and Raquel Urtasun. "Fully connected deep structured networks". In: *arXiv preprint arXiv:1503.02351*. 2015.

[161] Jing Shao, Kai Kang, Chen Change Loy, and Xiaogang Wang. "Deeply learned attributes for crowded scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4657–4666.

[162] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 39. Institute of Electrical and Electronics Engineers (IEEE), 2017, 640–651.

[163] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. "Training region-based object detectors with online hard example mining". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 761–769.

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

[165] Liat Sless, Bat El Shlomo, Gilad Cohen, and Shaul Oron. "Road Scene Understanding by Occupancy Grid Learning from Sparse Radar Clusters using Semantic Segmentation". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019, pp. 0–0.

[166] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research*. Vol. 15. 1. JMLR. org, 2014, pp. 1929–1958.

[167] Hauke Strasdat, José MM Montiel, and Andrew J Davison. "Visual SLAM: why filter?" In: *Image and Vision Computing*. Elsevier, 2012, pp. 65–77.

[168] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. "Scalability in Perception for Autonomous Driving: An Open Dataset Benchmark". In: *arXiv preprint arXiv:1912.04838*. 2019.

[169] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*. 2013, pp. 1139–1147.

[170] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[171] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[172] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[173] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, 2010.

[174] Chengzhou Tang and Ping Tan. "Ba-net: Dense bundle adjustment network". In: *arXiv preprint arXiv:1806.04807*. 2018.

[175] Xin Tao, Hongyun Gao, Xiaoyong Shen, Jue Wang, and Jiaya Jia. "Scale-recurrent network for deep image deblurring". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8174–8182.

[176] *The Economic and Societal Impact Of Motor Vehicle Crashes*. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812013. National Highway Traffic Safety Administration, 2015.

[177] *The GAN Zoo*. https://github.com/hindupuravinash/the-gan-zoo.

[178] Arash K Ushani, Ryan W Wolcott, Jeffrey M Walls, and Ryan M Eustice. "A learning approach for real-time temporal scene flow estimation from lidar data". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5666–5673.

[179] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." In: *Journal of machine learning research*. Vol. 11. 12. 2010.

[180] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2043–2050.

[181] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. "Esrgan: Enhanced super-resolution generative adversarial networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 0–0.

[182] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. "Dynamic graph cnn for learning on point clouds". In: *ACM Transactions on Graphics (TOG)*. Vol. 38. 5. ACM New York, NY, USA, 2019, pp. 1–12.

[183] Zhengwei Wang, Qi She, and Tomas E Ward. "Generative adversarial networks in computer vision: A survey and taxonomy". In: *arXiv preprint arXiv:1906.01529*. 2019.

[184] Zongwei Wang, Xu Tang, Weixin Luo, and Shenghua Gao. "Face aging with identity-preserved conditional generative adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7939–7947.

[185] Michael Warren, David McKinnon, and Ben Upcroft. "Online calibration of stereo rigs for long-term autonomy". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 3692–3698.

[186] *Waymo Safety Report: On the Road to Fully Self-Driving*. https://storage.googleapis.com/sdc-prod/v1/safety-report/Safety%20Report%202018.pdf. Waymo, 2018.

[187] M Williams. "PROMETHEUS-The European research programme for optimising the road transport system in Europe". In: *Driver Information, IEE Colloquium on*. IET. 1988.

[188] David H Wolpert. "The lack of a priori distinctions between learning algorithms". In: *Neural computation*. Vol. 8. 7. MIT Press, 1996, pp. 1341–1390.

[189] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling". In: *Advances in neural information processing systems*. 2016, pp. 82–90.

[190] Wenxuan Wu, Zhongang Qi, and Li Fuxin. "Pointconv: Deep convolutional networks on 3d point clouds". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9621–9630.

[191] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3d shapenets: A deep representation for volumetric shapes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[192] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. "Grid-GCN for Fast and Scalable Point Cloud Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5661–5670.

[193] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. "Attngan: Fine-grained text to image generation with attentional generative adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1316–1324.

[194] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. "PointASNL: Robust Point Clouds Processing using Nonlocal Neural Networks with Adaptive Sampling". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5589–5598.

[195] Nan Yang, Rui Wang, Jorg Stuckler, and Daniel Cremers. "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 817–833.

[196] Zi Jian Yew and Gim Hee Lee. "3dfeat-net: Weakly supervised local 3d features for point cloud registration". In: *European Conference on Computer Vision*. Springer. 2018, pp. 630–646.

[197] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. "BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling". In: *arXiv preprint arXiv:1805.04687*. 2018.

[198] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. "Seqgan: Sequence generative adversarial nets with policy gradient". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[199] Matthew D Zeiler. "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701*. 2012.

[200] Ji Zhang and Sanjiv Singh. "LOAM: Lidar Odometry and Mapping in Real-time." In: *Robotics: Science and Systems*. Vol. 2. 9. 2014.

[201] Ji Zhang and Sanjiv Singh. "Visual-lidar odometry and mapping: Low-drift, robust, and fast". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 2174–2181.

[202] Liliang Zhang, Liang Lin, Xiaodan Liang, and Kaiming He. "Is faster R-CNN doing well for pedestrian detection?" In: *European conference on computer vision*. Springer. 2016, pp. 443–457.

[203] Xinyi Zhang, Fei Wang, Hang Dong, and Yu Guo. "A deep encoder-decoder networks for joint deblurring and super-resolution". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 1448–1452.

[204] Yingxue Zhang and Michael Rabbat. "A graph-cnn for 3d point cloud classification". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 6279–6283.

[205] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. "PointWeb: Enhancing local neighborhood features for point cloud processing". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5565–5573.

[206] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. "Pyramid Scene Parsing Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[207] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. "Unsupervised learning of depth and ego-motion from video". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1851–1858.

[208] Yin Zhou and Oncel Tuzel. "Voxelnet: End-to-end learning for point cloud based 3d object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4490–4499.

[209] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. "Unet++: A nested u-net architecture for medical image segmentation". In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, 2018, pp. 3–11.