Domain adaptation for cross-sensor 3D object detection on point-clouds

Vladislav Shlenskii

MASTER'S THESIS

Domain adaptation for cross-sensor 3D object detection on point-clouds

Master's Educational Program: Industrial and Applied Mathematics

Student: Vladislav Shlenskii

Research advisor: David Sierra-González, PhD

Research scientist, Chroma team, Centre Inria Grenoble

Research advisor: Özgür Erkent, PhD

Research scientist, Chroma team, Centre Inria Grenoble

Research advisor: Christian Laugier, PhD

Research Director, Chroma team, Centre Inria Grenoble

Grenoble 2020

Domain adaptation for cross-sensor 3D object detection on point-clouds

Vladislav Shlenskii

Submitted to the Université Grenoble Alpes

Abstract

The field of self-driving cars is developing tirelessly, attracting many technology companies to research this area: Google, Apple, Yandex, Tesla and many others. One of the most important and basic problems in the field of autonomous driving is 3D object detection the goal of which is to localize and classify objects utilizing data from different sensors.

To solve the 3D object detection problem we face two main issues: firstly, there is very little amount of labelled relatively to unlabelled data; and secondly, existing datasets are captured with platforms that have different sensor setups (usually LiDAR sensors with different resolution). Hence, there arises the idea to use a model, pre-trained on the available data set, in practical self-driving experiments, despite different sensor settings.

In this thesis we address, firstly, the problem of deep neural network's portability pre-trained on data from LiDAR with one setup to data from LiDAR with another setup; we do this on example of 3D object detection problem and establish that the problem does exist. Secondly, we conduct experiments on scene-wise cross-sensor domain adaptation altering the pre-trained model's weights in-place applying several domain adaptation approaches — adversarial training and maximum mean discrepancy.

Acknowledgments

I would like to express my warm gratitude to Christian Laugier, the research director of Chroma team in Centre Inria Grenoble I was honoured to have internship in for letting me work on this topic.

I would like also to sincerely thank my scientific advisors David Sierra-González and Özgür Erkent for their invaluable assistance throughout the entire period of work on this topic, for many fruitful discussions of research and practical ideas, for invaluable experience I luckily happened to achieve during this internship as well as their guidance on conducting experiments.

In addition I would like to express gratitude to Jérôme Lussereau whose help and patience in resolving hardware issues in post-first-wave-pandemic-period were highly helpful.

Also, I thank the engineers and everyone involved in the work of Grid5000 the high-performance Inria's cluster system, the hardest part of the calculations was done there.

This work is partially supported by the French National Research Agency in the framework of the "ANR 3IA MIAI@Grenoble Alpes".

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The field of self-driving cars is developing tirelessly, attracting many technology companies to research this area: Google, Apple, Yandex, Tesla and many others. Significant advances in autonomous driving are largely due to the developments that applied science and technology have received over the past ten years, be it architectures and methods of deep learning, software in the form of wonderful deep learning frameworks like Tensorflow [1] or PyTorch [2], powerful graphics or tensor accelerators, and LiDARs that scan the surrounding area with high precision.

One of the most important and basic problems in the field of autonomous driving is 3D object detection the goal of which is to localize and classify objects utilizing data from different sensors. A high-quality solution to this problem allows the car to better understand the road situation and improve the behavior and reaction of the car by decision-making systems.

At the same time, a large amount of labelled data is needed to get a robust model, ideally data is needed for every road situation. However, there is very little amount of it relatively to unlabelled data. This is where the idea of adaptation comes in — to use the labelled data information to introduce the model to the unlabelled data. For images, this adaptation has been successfully implemented since mid of the previous decade already, but this is not the case with point clouds, which are captured by LiDAR — one of the main sensors of an self-driving vehicles, especially for road situation as a whole.

There is a great amount of LiDAR sensors and all they produce data with slightly different distribution, which induces the need of domain adaptation for point clouds scene-wise. Additionally, the experimental vehicles of existing labelled datasets are equipped with different LiDAR configurations. This is how the problem of cross-sensor domain adaptation for autonomous vehicles arises.

1.1 Motivation

Importance of domain adaptation arises naturally as generalization error of supervised learning algorithms with insufficient amount of labelled data may be large, so that a model performs poorly while complementary manual labelling may be of high cost and labor resource, thus unsatisfactory. Domain adaptation addresses the problem of two domains describe closely related objects but under different distributions. Domain adaptation proposes the idea of knowledge transfer from the labelled source set (domain) of data to the unlabelled target set (domain) by exploration of domain-invariant features connecting different domains with discrepancy in distributions [3].

The problem of the availability of labeled 3D data and the complexity and high cost of manual labelling is

especially acute in the field of self-driving cars. For instance, turning back to an example of 3D vehicle detection, where at the stage of training a model it is necessary to know its exact location in the form of a 3D bounding box. To obtain an accurate location, it is necessary to conduct a thorough analysis of the data obtained: take into account the tilt of the car in several planes, for example, due to the slope of the road or non-parallel parking, overlap by other cars and objects, etc. The problem is compounded by the fact that the size of a road scene easily reaches a range of $(-70\text{ m}, 70\text{ m}) \times (-40\text{ m}, 40\text{ m}) \times (-1\text{ m}, 4\text{ m})$ in x-, y- and z-directions[1] correspondingly relative to the location of a LiDAR scanner and contains thousands of 3D points.

Example of 3D object detection inference for autonomous vehicles and underlying point cloud representing road situation is presented in Fig. 1.1.

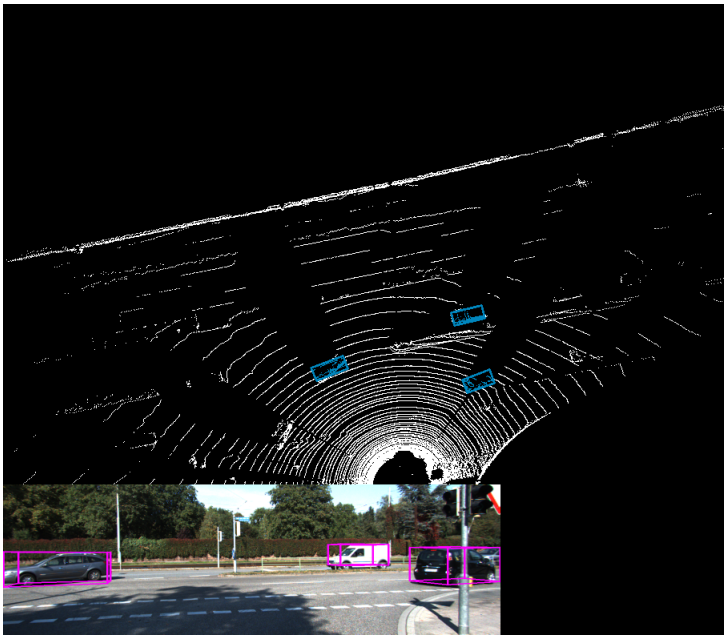


Figure 1.1: Example of 3D object detection inference and the corresponding point cloud, source [4]

With a neural network well-trained on a labeled dataset from a specific scanner, a manufacturing company will likely have to expand its fleet of self-driving cars by adapting to the specific LiDAR model used in training. On the one hand, good scanners that provide a more complete description of the surrounding space can be very expensive, for example, the cost of a sample with a resolution of 64 laser beams is $ 75,000[2]. On the other hand, the decrease in the cost of the used LiDARs leads to a decrease in scanning resolution, which entails a less comprehensive road situation description and drop in object detection accuracy[3] and *creates a problem of safe behavior of self-driving car on the road.*

A naïve way to solve the indicated contradiction between the cost of equipment and safety could be to train the model on the data obtained using the most modern LiDAR, and at the stage of mass use, install cheaper

[1] Negative values of z-direction are explained by the fact that LiDARs are installed on the roofs of cars, respectively, in this case, the roadbed is located lower relative to the sensor.

[2] According to information from 2018 based on analytic from http://www.woodsidecap.com/wp-content/uploads/2018/04/Yole_WCP-LiDAR-Report_April-2018-FINAL.pdf.

[3] We show this in our thesis in Chapter 5.

sensors on cars to collect the necessary information about the traffic situation[4]. This may be explained with effect known as covariance shift [5] between trained data and evaluation data, i.e. in discrepancy in train and evaluation distributions of data. In case of data collected by different LiDARs this takes place indeed, as each sensor has its own parameters which affects data distribution even though they depict information about equal entities — the road situation.

It is possible to go to two-dimensional data for detection, for example, to use monocular images or stereo-pairs of the surrounding environment, but this has noticeable disadvantages such as loss of scene depth and a shorter scene visibility range compared to LiDARs. All this leads to a significant deterioration in the quality of detection, especially in remote scenes. It is worth mentioning that there exist methods [6] fusing data both stereo-pairs and low-resolution LiDAR for depth correction that achieve remarkable results (achieving detection accuracy on par with sensors of high resolution) on 3D object detection benchmark [7], however, the problem of the high cost of additional data markup for better interaction of car in real-life situation remains in this case as well.

1.2 Goals

The main goal of this master thesis is to investigate the ways of bridging the gap between different LiDARs sensors for the task of 3D object detection scene-wise. We achieve the designated goal as follows:

- Exploring state-of-the-art multi-discipline approaches for point cloud processing, 3D object detection and domain adaptation.
- Justifying the necessity of road scene-wise[5] cross-sensor domain adaptation on example of 3D object detection validated on KITTI 3D object detection benchmark.
- Experimenting with different domain-adaptation architectures and approaches with focus on transfer learning and in-place adaptation of model to the new LiDAR sensor.

1.3 Thesis structure

Chapter 2 contains a brief introduction in point clouds, their properties and associated difficulties for deep-learning-based-approaches. Then we describe a key architecture for processing point clouds and task-specific model for 3D object detection that is used as the baseline model for cross-sensor domain adaptation. Finally, we overview different works on domain adaptation in general, applied to point clouds and to 3D object detection.

In Chapter 3 we formulate problems for two main experimental parts of this thesis in more detail and provide a description of the domain adaptation framework used for the second series of experiments.

[4] We show in Chapter 5 that this approach leads to detections with significant low accuracy.

[5] We use this term as an opposition to the "object-wise" domain adaptation on point clouds problem that considered widely in literature where the main goal is to operate between different representation of object, e.g. reconstruct the missing parts. "Object-wise" domain adaptation assumes the use of local a priori information to isolate common features of objects. At the same time, it is difficult to do the same for road scenes due to their huge variety and large size.

Chapter 4 describes in detail the experimental setup: benchmark, including dataset and evaluation metrics, networks' hyper-parameters and training procedure, utilized hardware and software.

Chapter 5 discusses the results of both experiments and provides future perspectives. The first experiment is intended to justify the necessity of domain adaptation on example of 3D object detection. Secondly, experimentation with variations of performing domain adaptation of model in-place w.r.t. 3D object detection task.

We finalize thesis with brief a summary in Chapter 6, followed by bibliography and appendices.

Chapter 2

Related Works and Theoretical Background

In this chapter, we highlight key concepts and important works with respect to, firstly, processing point cloud type of data and, secondly, domain adaptation in general.

2.1 Point clouds

2.1.1 Definition and properties

A point cloud is just a set of points in 3D Euclidean space, where each point is characterised by its coordinates $(x, y, z) \in \mathbb{R}^3$. The "point cloud" term is usually used to describe 3D representation of a real-world objects, e.g. their surface, geometry, location etc.

The most common way to obtain such set of points describing a real-world scene is to utilise a LiDAR[1] sensor[2]. Presently, including LiDARs for autonomous vehicles, lasers are used as the main source of light for such devices. Briefly, the principles of how LiDAR sensors operate are as follows: emitter outputs a laser beam into surroundings, when the beam reaches an object, it reflects from it and changes direction of motion to the opposite towards the LiDAR, where it is detected by a receiver. By the time difference between beam emitting and its detection, the distance between the sensor and the object is calculated. Having multiple beam emitters gives one multiple points describing the object's representation more fully. In addition to spatial coordinates, LiDAR may provide additional information about point, e.g. intensity of reflection.

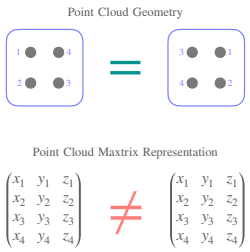
Point cloud type of data is stored as array of feature arrays, i.e. if there are n points and each point is described with m features, including three spatial coordinates, then the whole point cloud is stored as $\mathbb{R}^{n \times m}$ matrix; and this type of data has peculiarities that are challenging to neural networks and should be carefully taken into account. Listed in similar fashion to [8], the main challenges are as follows:

- **Unstructuredness:** Points are stored in the array and, thus, there is no possibility to reorder points in array and not to lose spatial interaction between points [9, p. 3]. It means that if two points are close to each other geometrically, it is not guaranteed they remain to be so after being placed in the array. This means the model should be capable of encapsulating local combinatory interactions among points.

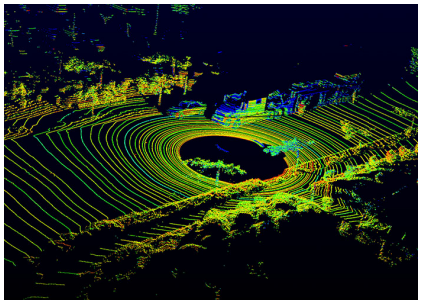
[1] Stands for "Light Detection and Ranging".

[2] Further, we will use "LiDAR sensor" and "LiDAR" interchangeably.

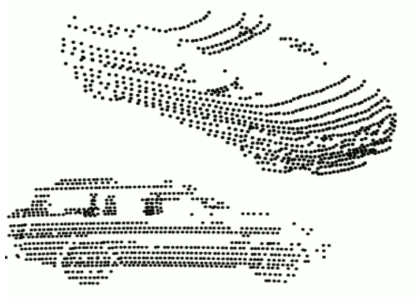
- **Invariance to permutations:** Points in array may be reordered in any of $n!$ variants. This means that there are $n!$ different $n \times m$ matrices describing the same object, hence, the model that takes a single point cloud should be invariant to $n!$ permutations as shown in Fig. 2.1a.
- **Varying number of points:** In image type of data, if one fixes a camera to shoot with, the number of pixels in output image is constant regardless of environment. However, even with fixed LiDAR sensor, the size of point cloud may vary significantly with change of environment.
- **Varying point density:** LiDARs for autonomous vehicles emit multiple laser beams at different angles collecting point 360° wide. Such sensor construction implies that objects closer to the sensor are described with more point than farther objects as shown in Fig. 2.1b.
- **Invariance to rigid transformations:** Being rotated, a 3D object remains the same, but the set of points describing it may change dramatically, see Fig. 2.1c. This means that learned object's representation should be invariant to such transformations.
- **Missing and noisy data:** In a real life environment objects interact with and moving relative to each other. This leads to object occlusions and incomplete point clouds. Also, imperfections of sensor and state of the environment may cause perturbations of point, i.e. point's real coordinates and ones registered with LiDAR differ from each other.



(a) Invariance to permutations



(b) Varying point density, source Arstechnica[3]



(c) Invariance to transformations, source ACFR dataset[4]

Figure 2.1: Demonstration of point cloud peculiarities

2.1.2 Deep Learning on Point Clouds

In relation to point clouds, just like for images and videos, all the same problems of computer vision are considered, as so: classification, semantic and instance segmentation or object detection. In all these areas was achieved significant success and it keeps moving forward.

[4]https://arstechnica.com/cars/2017/01/googles-waymo-invests-in-lidar-technology-cuts-costs-by-90-percent/
[4]http://its.acfr.usyd.edu.au/datasets/five-roundabouts-dataset/

2.1.2.1 PointNet: Raw Point Cloud Processing

There were several deep-learning-based approaches appeared nearly at the same time tailored for handling raw point clouds as an input. Those works may divided into two groups: deal with invariance property, for instance, [10, 9] regardless of point position in point cloud array or, like in [11], create more complex ordering on points than bijection from high m-dimensional space to a one-dimensional one, i.e. different ways of reordering points in the array.

Complex structuring in [11] is achieved with Kd-trees, the generalisation of a binary tree. Authors of [10] propose permutation invariant/equivariant layers for neural networks, whereas in [9] (and in its modification [12] presented right after the original paper) there was developed "end-to-end" permutation invariant architecture. PointNet [9] is of high importance and widespread usage in deep autonomous driving, in particular, it is used in a model that is considered as a baseline for this thesis, therefore, let us dwell into more details about it.

PointNet's architecture is as shown in Fig. 2.2, from which it is clear that it is suitable for classification and segmentation tasks, but what is more notable for us is the primary part of the model — up to global features extraction, i.e. some vector describing the point cloud as whole.

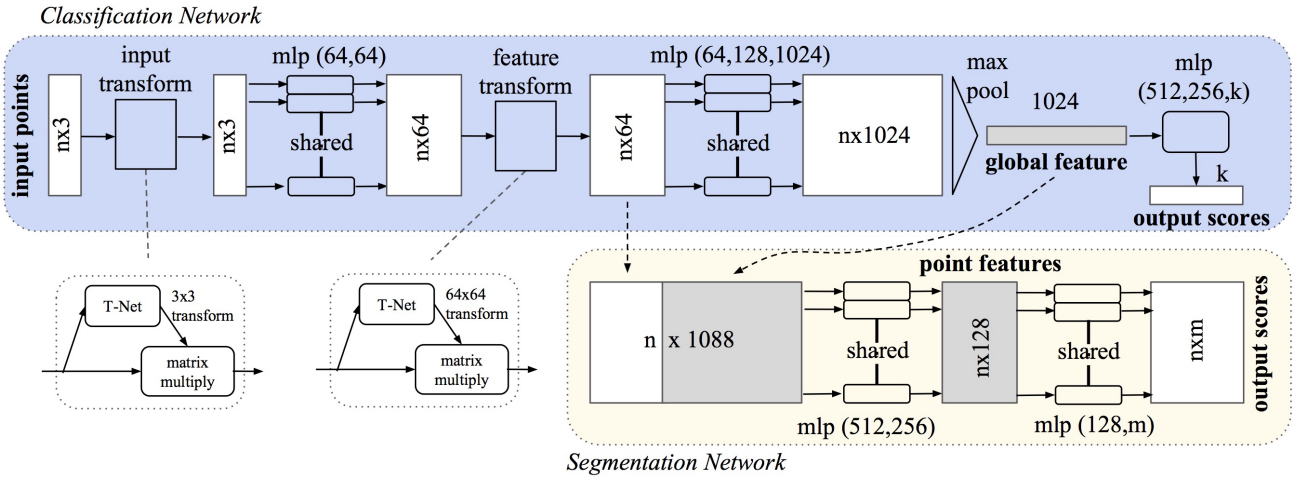


Figure 2.2: PointNet architecture, source [9]

Invariance to permutation of an input is proposed to be achieved with approximation of a general function $f\colon 2^{\mathbb{R}^N} \to \mathbb{R}$ defined on a point cloud by applying a symmetrical function $g\colon \mathbb{R}^{K^n} \to \mathbb{R}$ on transformed elements with some $h\colon \mathbb{R}^N \to \mathbb{R}^K$:

$$f\big(\{x_1, \ldots, x_n\}\big) \approx g\big(h(x_1), \ldots, h(x_n)\big). \tag{2.1}$$

In Fig. 2.2, a model's part from `'input transform'` to `'max-pool'`[5] excluding stands for a point-wise transformation g from Eq. (2.1) of an input. It is done by applying several shared MLPs[6], multi-layered perceptrons: single layer of MLP takes the same linear combination of point's features per each point in the whole cloud followed by batch normalization [13] and nonlinear function ReLU [14]. Whereas `'max-pool'`

[5] In Fig. 2.2 `'max-pool'` is schematically represented with a triangle.
[6] MLP stands for "Multi-Layer Perceptron".

operation represents a symmetric function h from Eq. (2.1). There are also supplementary networks called T-Net act as STN (Spatial Transformation Network) [15], and their aim is to predict a suitable affine transformation matrix for facilitating further feature extraction.

2.1.2.2 PointPillars: 3D Object Detection for Autonomous Vehicles

Models for 3D object detection may be roughly divided into groups w.r.t. data that is used for inference: it could be image-based models like [16], or LiDAR-only ones [17, 18] or both image- and LiDAR-based architectures [4]. In this thesis we stick to using LiDAR-only networks as we explore the adaptation in LiDAR domain. More precisely, we consider PointPillars architecture [18] to be a baseline for our experiments, so we will discuss the way it operates in more details.

PointPillars is an "end-to-end"[7] state-of-the-art algorithm for 3D object detection operating directly on point clouds. Compared to the preceding state-of-the-art model, the VoxelNet [17], PointPillars is more precise and is able to process single point cloud much faster as it uses 2D convolutions instead of 3D ones in VoxelNet, for the reason of 2D convolution require one magnitude less multiplications and summation operations. The whole PointPillars architecture is shown in Fig. 2.3.

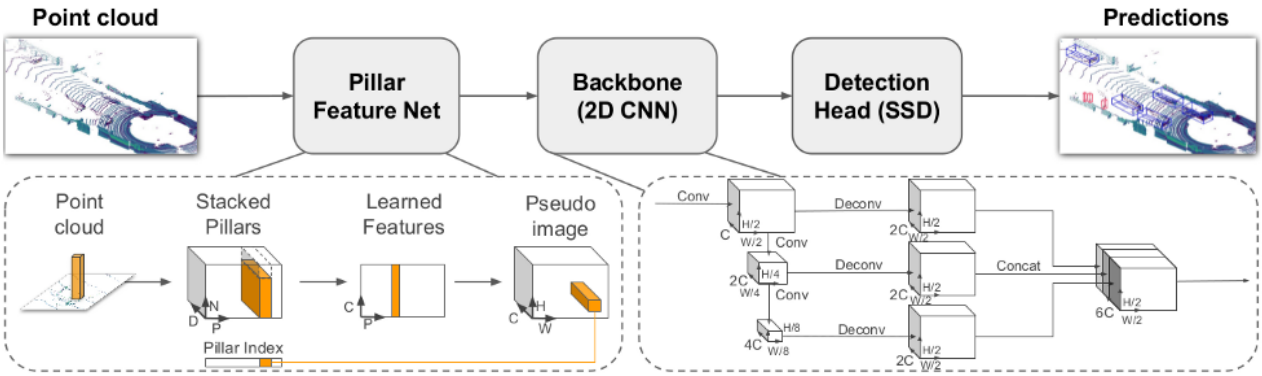


Figure 2.3: PointPillars architecture, source [18]

Pillar Feature Network Firstly, the whole point cloud is discretized into a grid over x-y plane creating a set of pillars. Then, additionally to raw characteristics like (x, y, z) Euclidean coordinates, each point is replenished with several more that reflect point's relationship with the pillar it belongs to: (x_c, y_c, z_c) — coordinate of mean point in the pillar — and (x_p, y_p) — its offset from the pillar centre at x-y plane. So, after such augmentation point is associated with $\mathbb{R}^D$-vector where $D \geq 8$.

As was mentioned in Section 2.1.1, point cloud has varying density, and after being quantized, the set of obtained pillars would be extremely sparse and may get hold of 97% [18, p. 3], so that using the whole pillar set is useless. To overcome natural sparsity of data authors use only nonempty pillars while taking in account fickle number of them in the point cloud as follows: fix P — number of nonempty pillars to be sampled from

[7] Here "end-to-end" means that particular problem is solved by a single model, bypassing intermediate steps, namely given a raw single input model processes it and outputs a solution.

each cloud; fix N — number of points to be sampled from each pillar, and if there are less points in pillar than N fill with the rest zero vectors. This procedure results an array of size (D, P, N) representing the initial cloud.

After is used a light version of PointNet [9] (one or several layers of MLP) to extract point features point-wise transforming (D, P, N)-sized array to (C, P, N)-sized one. The latter cloud representation becomes permutation invariant after applying max pool operation over point dimension resulting (C, P)-sized matrix and, consequently, is expanded back to pillar locations creating C-channelled pseudo-image of size (C, H, W).

Backbone Having 2D object instead of object from 3D world as an input leads to the ability to apply all the richness of effective methods for image processing. Authors of PointPillars use a similar backbone to previously mentioned VoxelNet [17]. It consists of two parts: top-down network that produces lower-resolution features and a down-up network performing upsampling lower-resolution features to unified size.

Top-down network is characterized as a series of blocks parametrized with (S, L, F) where S is a stride size [19, p. 12] measured relative to the original pseudo-image of size (C, H, W) the whole block operates at. L is a number of 3×3 2D convolutions that output F-channelled pseudo-image. The first convolution in a block operates at stride of size $\frac{S}{S_{\text{in}}}$ to ensure the whole block operates at stride of size S and other convolutions operate at stride equals to 1. Also, each convolution inside the block is also followed by BatchNorm and ReLU.

Each block in down-up network is described with $(S_{\text{in}}, S_{\text{out}}, F)$. First, lower-resolution features are up-sampled from initial stride S_{in} to S_{out} with 2D transposed convolution [19, p. 19] with F final features. Next, BatchNorm and ReLU are applied to every up-sampled pseudo-image.

Backbone's final output is formed from down-up network's outputs via concatenation over channel dimension.

Detection Head Finally, there is a '`Detection Head`'[8], which is a model called Single Shot Detector (SSD) [15] adopted for 3D object that outputs the final predictions.

2.2 Domain Adaptation

2.2.1 General Reasonings

The extensive prior work on domain adaptation problem was done for classical machine learning in [5]. After rapid rise of deep-learning-based models in early 2010-s given by the famous work [20], the research focus was shifted to transferring deep model representations obtained while training on one dataset to a different but similar one.

Deep-learning-based domain adaptation is usually hold on *low-level object representations*, typically obtained with ConvNets, as these dense features contain encoded description of an object as a whole and are smaller in size so working with them often is computationally easier than with the original input. There may be several ways to perform adaptation in practice.

[8] We will not go into details of detection methods themselves because this area lies out of scope of domain adaptation on point clouds.

The first one is the direct model transfer with the upcoming fine-tuning model's parameters, this is known as transfer learning [3, 21, 22, 23]. This means having a pre-trained task-specific model, for instance for semantic segmentation of road scene images under different weather conditions [21], one wants to use the same model for performing segmentation on sunny road scenes and the foggy ones, the latter case is more difficult due to blur and occlusions produced by fog. Even though the images have lots in common (roads, cars, pedestrians etc.), they are different, so the goal then may be, for instance, to preserve learned sunny road features and adapt model for fog.

Model modification may be held in-place by altering the last layers of the network [3] with MK-MMD [24] as distance discrepancy eliminator which computes the norm of difference between domains' means. One may also change the primary layers' weights, namely feature extractor, like in [22, 23]. The most common way to achieve it is known in literature as adversarial adaptation, and the idea is to add a discriminative classifier after feature extractor whose purpose is to learn to distinguish whether source or target features were given as an input, this in turn leads to the reconfiguration of the model's weights due to the fact that the additional goal of training now becomes to deceive the classifier.

The second way could be the direct adaptation [25] where a model takes source and target objects as an input and it is pushed to learn, firstly, to find regions that share common parts, i.e. table-legs, and, secondly, to make features representing these regions look alike.

In this thesis we focus on transfer learning, so let us formally define a problem to solve. But before that theoretical overview may not be completed without discussion of domain adaptation on point clouds and of autonomous vehicles applications.

2.2.2 Point Cloud and Autonomous Vehicles Application

Even though, to our best knowledge, there are no known literature that attempts to solve the problem that we state to examine, the literature overview would not be full without mentioning methods of domain adaptation on point clouds and w.r.t autonomous vehicles.

Point Cloud Deep domain adaptation on point clouds is a rather young trend of research largely spurred by the emergence of the PointNet architecture [9]. The key direction of known works is to perform adaptation object-wise [26, 25, 27] whereas we aim to do it scene-wise.

In [26], the authors solve the problem of unpaired object completion between source and target objects. Namely, source objects are "clean" (complete) point clouds of tables, target ones are deformed tables, i.e. with non-existing legs, this may happen due to occlusions in the moment of object's scanning. Idea is to, firstly, train point auto-encoders [28] on source and target domains, then in adversarial manner similar to [23] eliminate difference between source and target auto-encoded features, completion is done by applying decoder of source auto-encoder to target encoded features. Other authors focus on the classification problem, such as given a point cloud determine whether it belongs to one of categories, e.g. table [25]. The difficulty again is that the scans can be deformed or obtained with different LiDARs, adaptation is this case is done by finding similar regions in equally labelled objects with special network module and then mitigate domain shift in adversarial manner.

Autonomous Vehicles This area of research is even less explored than domain adaptation on point clouds with a single relevant paper [29] in which authors ameliorate image-based object detection on KITTI benchmark [7], namely predict object localization utilizing information from point cloud's projection on ground plane. Domain adaptation here is done between the real cloud's projections and projections from road simulator program with CycleGAN architecture [30] that proved itself well in image-to-image translation. This approach leads to better a model generalization of projection images, reinforced with any desired number of source images from simulator, and, consequently, increased object detection performance.

Our approach differs from [29] as well because we, firstly, aim to operate on point cloud directly, so less spatial information is lost in comparison to ground plane projections, and, secondly, aim to employ real-world scans only but not the synthetic ones.

Chapter 3

Methodology

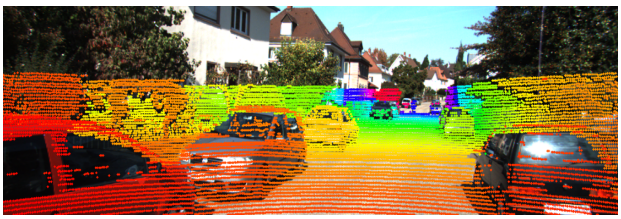
3.1 How transferable are features?

The main goal of this series of experiments is to obtain justification for necessity of domain adaptation w.r.t. autonomous vehicles field. Justification is done via answering the question "How transferable are features in deep learning for autonomous vehicles domain w.r.t different sensor configuration?".[1]

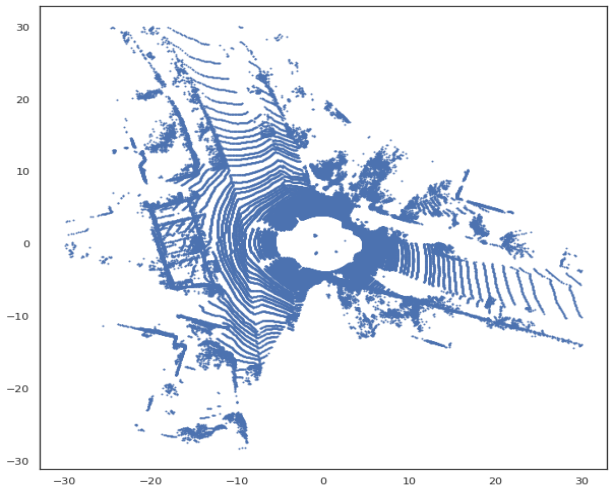
3.1.1 Experiment Description

LiDAR sensors differ in several aspects, but one of the most important characteristics is the number of laser beams it emits which may be understood as the resolution of captured of point cloud. Example of LiDAR scan of road scene with resolution of 64 beams is presented in Fig. 3.1.

(a) Road scene, source [7]



(b) Road scene with image-projected point cloud



(c) Road scene as point cloud ground-projection

Figure 3.1: Example of LiDAR sensor resolution, KITTI [7]

In this series of experiments we show by example of 3D object detection on road scenes problem that simple applying of pre-trained model on data captured with LiDAR of t-beamed resolution to evaluation on another set of data from s-beamed scanner leads in some cases to the drop in performance and in the other to redundant

[1] This is an homage to the classical work for domain adaptation on images [3].

complexity of the network (this, for instance, happens when the model is trained on data from higher resolution sensor and evaluated on lower resolution one).

To obtain more exhaustive number of train-evaluation scanner pairs we simulate them from the primary one by decreasing its resolution. As the primary scanner we take Velodyne HDL-64E LiDAR from 3D object detection KITTI benchmark [31] that has 64-beamed resolution. KITTI benchmark and performance metrics of 3D object detection for it are described in more details in Section 4.1.

Resolution of training and evaluation pairs we choose to be $[t, s] \in \{64, 32, 16, 4\} \times \{64, 32, 16, 4\}$ which are the most popular numbers of beams in LiDARs on the market.

Notation 1 *For brevity of narration we will further refer to conducted 3D object detection experiments on training/evaluation pairs with t-beamed and s-beamed resolution as $\mathrm{OD}_{t,s}$, to probability distributions of train data and evaluation data as $\mathbb{P}_t$ and $\mathbb{P}_s$ correspondingly, to the family of performance evaluation metrics, that are discussed in details in Section 4.1.1, as $\mathbb{M}_{t,s}$.*

Outcomes of this series of experiments is to compare performances of 3D object detection on different resolution pairs to justify necessity of domain adaptation and obtain pre-trained models on different sensor resolution that will be used further in experiments on domain adaptation.

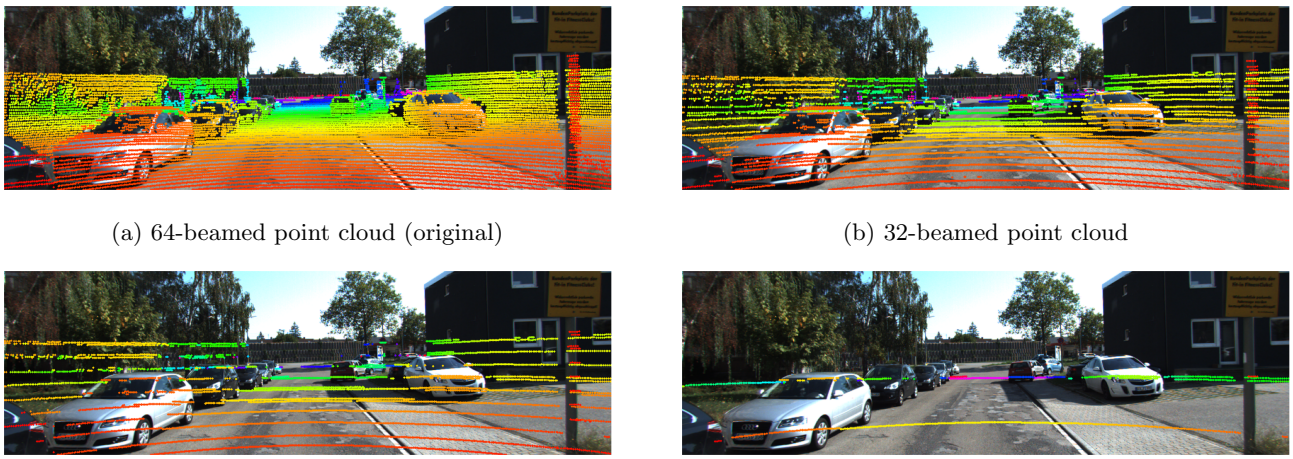
3.1.2 On Resolution Reduction

With knowing the 3D coordinates of point in space it is possible to find its elevation angle $\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ above the ground plane. This is done by a traversal from Euclidean coordinate system to a spherical coordinate system. Consider point's Euclidean coordinates are (x, y, z) and the corresponding polar coordinates are (r, θ, φ), where θ is an inclination angle. Then, the elevation angle may be derived from Euclidean coordinates as follows:

$$\alpha = \frac{\pi}{2} - \theta = \frac{\pi}{2} - \arccos\left(\frac{z}{x^2 + y^2 + z^2}\right). \tag{3.1}$$

According to technical specifications of the Velodyne HDL-64E LiDAR sensor [32], it has 64 laser beams which suggests that values of elevation angles should be settled down into 64 binned histogram. Thus, by thinning histogram and removing points corresponding to the deleted bins one may get a reduced point cloud representation that simulates a scan of a sensor with desired number of laser beams. For instance, to retrieve 32-beamed-like scan from 64-beamed one we remove every second bin, for 16-beamed-like it would be deleting every fourth beam etc.

Demonstration of produced 32-, 16- and 4-beamed scans outputted after applying histogram thinning with factors 2, 4 and 16 correspondingly is shown as image-projected point clouds in Fig. 3.2. Note that due to the presence of noise in the data, the actual elevation angle and the one recorded by the sensor may differ. Therefore, points actually having the same elevation angle can fall into different histogram bins. Thus, thinning the histogram may lead to artifacts in the point clouds, which is clearly seen in Fig. 3.2d where there are no four clearly fully filled "rings".



(a) 64-beamed point cloud (original)

(b) 32-beamed point cloud

(c) 16-beamed point cloud

(d) 4-beamed point cloud

Figure 3.2: Demonstration of resolution altering with image-projected point cloud, KITTI benchmark

3.2 Domain Adaptation Framework

In this section we properly define a domain adaptation task as an optimization problem, explain architecture details and experiment following [21].

3.2.1 Domain Adaptation Problem Formulation

General problem Consider a supervised learning dataset of n_s pairs $\mathcal{S} = \left\{(\mathbf{x}_s^i, \bar{y}_s^i)\right\}_{i=1}^{n_s}$ called "source domain" where $\mathbf{x}_s^i$ denotes a sample and $\bar{y}_s^i$ stands for manually assigned label for a sample $\mathbf{x}_s^i$, and another set of n_t samples $\mathcal{T} = \left\{\mathbf{x}_t^j\right\}_{j=1}^{n_t}$ standing for the "target domain". Also we define $\mathrm{X}_s = \{\mathbf{x}_s\}_{i=1}^{n_s}$ and $\mathrm{X}_t = \{\mathbf{x}_t\}_{j=1}^{n_t}$. Let probability distributions of data[2] of source domain be $p_s(\star)$ and of target domain be $p_t(\star)$ correspondingly.

Denote given model $M_{\hat{\gamma}}$, a deep neural network pre-trained on source domain $\mathcal{S}$ learned to estimate $\hat{y}_s^i$ given sample $\mathbf{x}_s^i$ via minimizing the source cost c_s:

$$c_s(M_{\hat{\gamma}}) = \mathbb{P}_{(\mathbf{x},\bar{y})\sim p_s}\left[M_{\hat{\gamma}}(\mathbf{x}) \neq \bar{y}\right]. \tag{3.2}$$

Then, the goal of domain adaptation is to provide a model M_γ so that the target cost c_t:

$$c_t(M_\gamma) = \mathbb{P}_{(\mathbf{x},y)\sim p_t}\left[M_\gamma(\mathbf{x}) \neq y\right], \tag{3.3}$$

and the source cost c_s minimize simultaneously.

Transfer learning We stick to perform domain adaptation through transfer learning with fine-tuning the primary part of model as this approach proved its efficiency both for images [21, 23] and point clouds [27], and mostly we rely on approach in [21] as a starting point.

[2]Or feature representation as well, for brevity of notation we use the same designation in both cases without additional clarification, the exact meaning may be easily divided from the context.

Under these assumptions we continue to expand Eq. (3.3). A deep network $M_{\hat{\gamma}}$ could be thought of as a composition of two networks $E_{\hat{\theta}}$ — initial layers — and $F_{\hat{\alpha}}$ — remaining layers — so that $M_{\hat{\gamma}}(\mathbf{x}) = F_{\hat{\alpha}}(E_{\hat{\theta}}(\mathbf{x}))$. As a result, we would like to come up with a model:

$$M_\gamma = F_{\hat{\alpha}}(E_\theta(\mathbf{x})) \wedge E^0_\theta = E_{\hat{\theta}}, \tag{3.4}$$

i.e. M_γ has the same architecture as $M_{\hat{\gamma}}$ and is equal to $M_{\hat{\gamma}}$ before domain adaptation procedure, where $\gamma, \hat{\gamma}, \theta, \hat{\theta}, \hat{\alpha}$ are networks parameters: hatted parameters are fixed and not changed during training, non-hatted parameters are learnable; and E^0_θ denotes weights of domain adapted model M_γ right before the domain adaptation learning.

We learn parameters θ so that the input features to $F_{\hat{\alpha}}$ network from the target domain would become similar w.r.t. distribution to the source domain. This changes the initial target cost from Eq. (3.3) into:

$$c_t\left(E_\theta(\mathbf{x}_t) = \mathbf{f}_t\right) = \mathbb{P}_{(\mathbf{f},y)\sim p_t}\left[F_{\hat{\alpha}}(\mathbf{f}) \neq y\right]. \tag{3.5}$$

Obtaining the desired model M_γ that suits Eq. (3.3) may be done in several ways. In this thesis we achieve it in similar fashion to [21] where the scheme was applied for domain adaptation on images.

Conceptually, domain adapted model M_γ is gained by altering weights of pre-trained model $M_{\hat{\gamma}}$, i.e. without changing architecture of the initial model at all. Weights modifications are done to fulfill the following requirements on the target model M_γ:

Proposition 1

(i) produced distribution of low-level features should be similar to source model's, i.e. closeness in distribution, this reflects sharing the same nature for target and source representations;

(ii) produced low-level features should lie closely to each other, i.e. closeness in norm, to preserve an ability for target model to operate on source domain (such manner of learning is called self-supervision).

Consider $\mathrm{D}(\star, \star)$ to be a function that measures distance between target distribution and $\mathrm{L}_{\mathrm{self-supervision}}(\star, \star)$ measures closeness in norm between the first term and the second term. Then, the optimal model parameter θ is found then as:

$$\tilde{\theta} = \inf_{\theta\in\Theta} \mathrm{D}\left(\mathbb{P}_{\hat{\mathbf{f}}_s\sim p_s}, \mathbb{P}_{E_\theta(\mathbf{x}_t)\sim p_t}\right) + \lambda_{\mathrm{ss}}\mathrm{L}_{\mathrm{self-supervision}}(\hat{\mathbf{f}}_s, E_\theta(\mathbf{x}_s)), \tag{3.6}$$

where λ_{ss} are the regularization parameters.

3.2.1.1 Handling distribution discrepancy

Minimizing discrepancy, the first term in Eq. (3.6), between source and target distributions of low-level features from Proposition 1 (i) may be done with adversarial training using Wasserstein distance [33] or using MKMMD [24].

1-Wasserstein distance between two distrubtions is defined as follows:

$$\begin{aligned} \mathrm{D}\left(\mathbb{P}_{\hat{\mathbf{f}}_s\sim p_s}, \mathbb{P}_{\mathbf{f}_t\sim p_t}\right) &= W_1\left(\mathbb{P}_{\hat{\mathbf{f}}_s\sim p_s}, \mathbb{P}_{E_\theta(\mathbf{x}_t)\sim p_t}\right) \\ &= \inf_{\mathbb{P}_{\hat{\mathbf{f}}_s\sim p_s, \mathbf{f}_t\sim p_t}} \mathbb{E}\left[\left\|\hat{\mathbf{f}}_s - \mathbf{f}_t\right\|\right] \qquad (3.7) \\ &= \inf_{\phi\in\Phi} \mathbb{E}\big(\mathrm{D}_\phi(\hat{\mathbf{f}}_s)\big) - \mathbb{E}\big(\mathrm{D}_\phi(\hat{\mathbf{f}}_t)\big). \qquad (3.8) \end{aligned}$$

For eliminating difficulties with convergence during solving optimization problem Eq. (3.8) pointed in [34], we use additional term called gradient penalty [34, p. 4, Algorithm 1]:

$$\mathrm{GP}(\mathrm{D}_\phi, \hat{\mathbf{f}}_s, \mathbf{f}_t) = \lambda_\mathrm{p}\left(\|\nabla_{\tilde{\mathbf{f}}}\mathrm{D}_\phi(\tilde{\mathbf{f}})\|_2 - 1\right)^2, \qquad (3.9)$$

where $\tilde{\mathbf{f}} = \varepsilon\hat{\mathbf{f}}_s + (1-\varepsilon)\mathbf{f}_t$ and $\varepsilon \sim \mathrm{Uniform}[0,1]$, and $\lambda_\mathrm{p} > 0$ is a gradient penalty.

When training adversarial training there is another hyper-parameter that affects convergence and final result — n_crit, the number of discriminator D_ϕ updates per single update of generator (in our notation it is E_θ).

MKMMD (Multi MMD, Multi Maximum Mean Discrepancy) is a convex combination of RKHS (Reproducing Hilbert Space $\mathcal{H}_\kappa$ with a positive-definite reproducing kernel κ) distances between mean low-level features of two distributions p_s and p_t. In a single element combination case is defined as follows:

$$\mathrm{MMD}_\kappa(p_s, p_t) = \mathbb{E}_{\hat{\mathbf{f}}_s, \hat{\mathbf{f}}_{s'}}\kappa(\hat{\mathbf{f}}_s, \hat{\mathbf{f}}_{s'}) + \mathbb{E}_{\mathbf{f}_t, \mathbf{f}_{t'}}\kappa(\mathbf{f}_t, \mathbf{f}_{t'}) - 2\mathbb{E}_{\hat{\mathbf{f}}_s, \mathbf{f}_{t'}}\kappa(\hat{\mathbf{f}}_s, \mathbf{f}_{t'}) \qquad (3.10)$$

where $\hat{\mathbf{f}}_s, \hat{\mathbf{f}}_{s'} \sim p_s$ and $\mathbf{f}_t, \mathbf{f}_{t'} \sim p_t$ are the features from the source and target domain respectively such that $\hat{\mathbf{f}}_s \neq \hat{\mathbf{f}}_{s'}$ and $\mathbf{f}_t \neq \mathbf{f}_{t'}$.

In our experiments we used the gaussian kernel:

$$\kappa(x, x') = \exp\left(-\gamma^{-2}\|x - x'\|^2\right), \; x, x' \in \mathbb{R}^d, \gamma \in \mathbb{R}. \qquad (3.11)$$

3.2.1.2 Self-supervision

Closseness in norm Proposition 1 (ii) is achieved with features self-supervision [35] is expressed as squared second of vector of difference between source domain features and target ones:

$$\mathrm{L}_{\mathrm{self-supervision}}(\hat{\mathbf{f}}_s, E_\theta(\mathbf{x}_s)) = \|\hat{\mathbf{f}}_s - \mathbf{f}_t\|_2^2. \qquad (3.12)$$

Combining all above stated, the training algorithm for domain adaptation is presented in pseudo-code in Algorithm 1 at listing Algorithm 1. And the whole non-task-specific self-supervised domain adaptation framework we stick to in this thesis is schematically shown in Fig. 3.3.

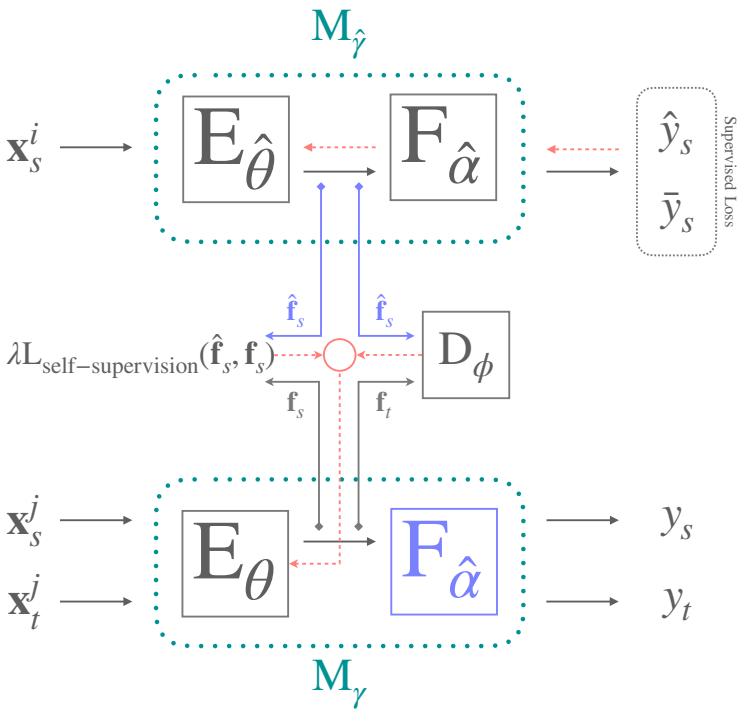


Figure 3.3: Adversarial domain adaptation framework. Green blocks represent 3D object detection models. Dotted red lines stand for back-propagation. Blue lines and frames indicate fixes parts without back-propagation. D_ϕ represents dicriminative classifier that is used in case of adversarial training, otherwise it means MKMMD term.

3.2.2 Architecture Details

As a baseline for 3D object detection task we take PointPillars architectures that was described in details in Section 2.1.2.2 because it performs greatly for announced task and is convenient for conducting experiments due to its relatively fast training time[3] in comparing to more complicated networks.

Encoder According PointPillars architecture Fig. 2.3 there may be extracted two non task-specific primary parts: one consists of `Pillar Feature Net` and the other one combines `Pillar Feature Net` and `Backbone (2D CNN)`. Indeed, the latter part of PointPillars is dedicated to object classification and localization and thus may not be considered as non task-specific. Thus, E_θ may be one of `Pillar Feature Net` or `Backbone (2D CNN)` which we refer to as `PostPFN` and `PostBackbone` correspondingly.

Adversarial training In the original paper of PointPillars size of pseudo-image produced in `Pillar Feature Net` was set to 496×432 pixels, so we use the following light-weight architecture for discriminative classifier D_ϕ: it consists of two convolutional layers with number of input channels equal to 64 and 8, each convolution is followed by instance normalization [36], classifier itself is two fully-connected layers with input feature sizes

[3]However, it is still approximately 17 hours while training on GTX 1080 GPU with 8 Gb of RAM.

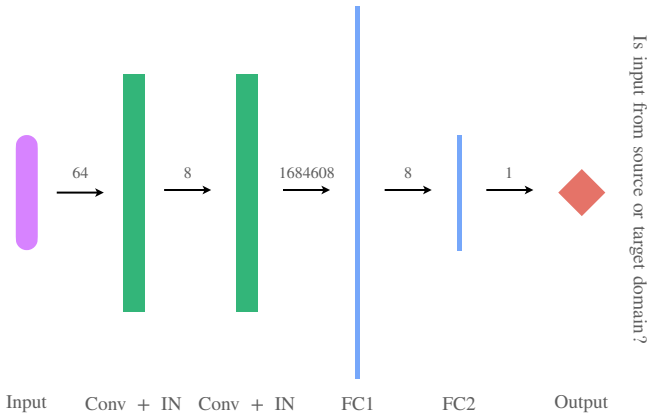


Figure 3.4: Discriminator for adversarial training. Here *Conv* stands for convolutional layer, *IN* — for instance normalization layer, *FC* — for fully connected layer.

equal to 1684608 and 8 correspondingly, the first fully-connected layer is followed by ReLU non linear function. Schematically the discriminator for adversarial training is shown at Fig. 3.4.

3.2.3 Experiment Description

As an experiment on domain adaptation, we have chosen the following problem statement: given a PointPillars model pre-trained on 64-beamed KITTI dataset (the source domain $\mathcal{S}$) perform its adaptation to 16-beamed variant of KITTI (the target domain $\mathcal{T}$). We chose this particular number because the Chroma team's autonomous vehicle fleet has a vehicle that captures 3D space descriptions using a 16-beamed LiDAR. However, to our best knowledge, there is not a 3D object detection dataset with 16-beamed resolution available, so we want to leverage KITTI to train our model.

Notation 2 *In addition to notation of 3D object detection experiments Notation 1 we will refer to the particular domain adaptation experiment settings as* $\mathrm{DA}_{64,16}$.

We evaluate success of domain adaptation by comparing 3D object detection performance of model with and without domain adaptation. Looking ahead to the results discussed in Section 5.1 the baseline model (i.e. the one gives the lower bound in performance) is $\mathrm{OD}_{64,16}$ and the reference model (that gives the upper bound in performance) is $\mathrm{OD}_{16,16}$. We compare domain adaptation performance for several models as listed below in Proposition 2.

Proposition 2

(i) `PostPFN + GP` *uses* `PostPFN` *as feature encoder and WGAN adversarial training with gradient penalty for adjusting source and domain distribution.*

(ii) `PostPFN + MKMMD`*, the same as Proposition 2 (i) but with MKMMD for handling distribution discrepancy.*

(iii) `Postbackbone + MKMMD`[4], *the same as Proposition 2 (i) but with* `Postbackbone` *encoder instead of* `PostPFN` *one.*

[4]Option `Postbackbone + GP` is not considered as it could not fit into a 8 Gb of GPU RAM memory.

Chapter 4

Experimental setup

4.1 3D Object Detection Dataset

As a dataset for this thesis we choose KITTI 3D Object Detection Dataset,[1] which comes as a part of 3D object detection benchmark [7] with the following information included:

- 7481 training images and 7518 test images.
- Transformation matrices between different cameras as well as between points' coordinates.
- Scene-synchronized point clouds obtained with Velodyne HDL-64E LiDAR sensor [32], comprising a total of 80,256 labeled objects.
- Objects' labels[2], s.t. each object is assosiated its vehicle type; its 3D coordinates (x, y, z); its dimensions (l, w, h) — length, width and height; coordinates of its 2D bounding box — a rectangle that wraps object's projection on the image; its truncation rate $t \in [0, 1]$ — a degree to which an object's 2D bounding box extends beyond the image boundary; and its an occulision state $\gamma \in \{0, 1, 2, 3\}$ — a degree to which an object overlaps with the others, categorical feature.

In this thesis we only evaluate performance on objects of type "Car".

4.1.1 On KITTI evaluation

4.1.1.1 KITTI Object's Difficulty Levels

With respect to object's properties to each of them are assigned difficulty levels "Easy", "Medium" and "Hard" for the more exhaustive understanding the performace of 3D object detection algorithm. These levels are defined in Table 4.1, namely this table sets the value boundaries for object properties, which determine the complexity of its detection.

4.1.1.2 Object Detection Metrics

We assess model's performance for 3D object detection in several official KITTI regimes: **bird's eye view (BEV)**, **3D**, **2D** (will be referred as `bbox`) and **average orientation similarity (AOS)**. The final metrics

[1] Here and furthermore, for a sake of brievity, we refer to a "KITTI 3D Object Detection Dataset" as "KITTI".

[2] The full object's description is more detailed and contains a few more parameters, here provided only those and only in such volume that is crucial for further narration.

Difficulty	Easy	Medium	Hard
Minimum bounding box height (px[3])	40	25	25
Maximum truncation (%)	15	30	50
Maximum occlusion level	0	1	2

Table 4.1: KITTI detection difficulty modes

of detection quality in each regime is calculated using mean average precision (mAP). **BEV** is dedicated to measure quality of object detection via juxtaposition of predicted and ground truth bounding boxes in the ground plane (top view), **3D** — speaks for itself, **2D** — in image plane and **AOS** operates in image plane as well but additionally takes in account direction of object's rotation.

For instance, **AOS** mean average precision is defined as follows [7]:

$$AOS@x = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} \max_{\tilde{r}:\, \tilde{r} \geq r} s(\tilde{r}), \tag{4.1}$$

where $r = \frac{TP}{TP+FN}$ is the PASCAL [37] object detection recall, i.e. how good the predicted positive cases are, and detected image-projected 2D bounding box is considered as the correct one if it overlaps with the ground truth bounding box by given threshold $x \in [0.5, 1]$. And $s(\star) \in [0, 1]$ given recall r is a normalized variant of cosine similarity:

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + \cos \Delta_\theta^{(i)}}{2} \delta_i, \tag{4.2}$$

where $D(r)$ stands for the set of all object detections at recall rate r and $\Delta_\theta^{(i)}$ is the difference between predicted orientation angle and ground truth orientation angle of i-th detection; coefficient δ_i penalizes multiple detections of the same object, namely, δ_i is set equal to 1 if i-th detection was assigned to a ground truth bounding box and $\delta_i = 0$ otherwise. In the very similar fashion mAP is defined for other regimes of KITTI evaluation.

4.1.2 Software and Hardware

The whole software for this project was written in Python of version 3.7 [38]. As a deep learning framework we used the PyTorch [2] of version 1.5. Other libraries including NumPy [39] 1.17.4 for point cloud processing, Matplotlib [40] 3.2.2 and OpenCV [41] 4.3.0, Hydra [42] 1.0.0rc2 for manipulation on various training configurations were widely used.

For conducting heavy experiments on feature transferability we used Grid5000 [43], the Inria's scientific cluster where user work spaces ran on Ubuntu 16.04 LTS and CUDA of version 10.2. Experiments on domain adaptation were carried out on the local machine under Ubuntu 18.04 LTS and CUDA of version 11.0. In both cases the GTX 1080 GPU with 8 Gb of RAM was used.

4.1.3 Training Details

The parameters of discriminator D_ϕ are initially set with random variable from normal distribution. Adam [44] was used for optimizing network parameters with constant learning rate $\alpha = 1 \cdot 10^{-5}$; coefficients for computing running averages of gradient and its square are $\beta_1 = 0.9$ and $\beta_2 = 0.99$ correspondingly; weight decay, or L2 penalty, was set equal to $5 \cdot 10^{-4}$. Loss multiplier for self-supervision $\lambda_{\text{ss}} = 1000$ and for the gradient penalty multiplier $\lambda_{\text{p}} = 10$. Also, in case of adversarial learning we set number of discriminator updates per single encoder update $n_{\text{crit}} = 1$. The batch size in case of WGAN equals to 4, and in case of MKMDD — to 2. Number of training adaptation iterations for WGAN and MKMMD were set correspondingly to batch sizes so that each variant used equal number of training samples with maximum quantity of 600 iterations.

Chapter 5

Results

5.1 How transferable are features?

Figures A.1, A.2, A.3 represent the heat maps with quality of 3D object detection for PoinPillars model w.r.t. AOS@70 metric for all training/evaluation resolution pairs discussed previously. Additionally, the list of all obtained metrics, based on which, in particular, the heat maps below are plotted, is exhaustively presented in Appendix B.

Note that we take AOS@70 for discussion just as an example, for other metrics or evaluation protocols (either AOS@50 or COCO) demonstrated trend is absolutely the same, and so we use general notation for metrics as proposed in Notation 1. Note that we also provide the heat maps and domain adaptation top-results for 3D@70 mAP (since this metric type is of great importance for 3D object detection) in Appendix C.1 and Appendix C.2 respectively.

In all situations performances on the side diagonal of the heat maps are increasing steadily, namely the following holds:

$$\mathbb{M}_{4,4} < \mathbb{M}_{16,16} < \mathbb{M}_{32,32} < \mathbb{M}_{64,64}. \tag{5.1}$$

This is rather expected as more thinned point cloud contains less fulled pillars and points inside them, so the sparsity of pseudo-image increases with decreasing resolution of LiDAR, i.e. higher resolution LiDAR produced more detailed description of road situation. And the difference in performances is significant even in "Easy" case, for instance, it is $\mathbb{M}_{64,64} - \mathbb{M}_{16,16} = 90,52 - 79,07 = 11,45$ of mAP AOS, and increases almost doubly for "Medium" and "Hard" reaching $\mathbb{M}_{64,64} - \mathbb{M}_{16,16} = 88,40 - 66,99 = 21,41$ and $\mathbb{M}_{64,64} - \mathbb{M}_{16,16} = 86,72 - 64,66 = 22,06$ respectively. These considerations prove the need of usage of high resolution LiDARs for uncompromising quality of 3D object detection.

Insufficiency of straightforward applying the pre-trained model on higher resolution to lower resolution data and thus the need for domain adaptation for 3D object detection is especially clear from the first two lines in "Medium" and "Hard" cases (refer to Figs. A.2 and A.3 correspondingly) as in these cases quantity of information about surroundings becomes crucial:

$$\mathbb{M}_{[64,32,16],4} < \mathbb{M}_{[64,32,16],16} < \mathbb{M}_{[64,32,16],32} < \mathbb{M}_{[64,32,16],64}. \tag{5.2}$$

Eq. (5.2) suggests that $\mathbb{P}_s$ with higher resolution s contains more meaningful information and, respectively, less noise for hardly distinguished objects in comparison to distributions $\mathbb{P}$ with lower resolution.

For the "Easy" mode (Fig. A.1), the behavior for the first two lines remains the same, but there is an interesting observation for 16-beamed resolution data:

$$\mathbb{M}_{[16,32]} > \mathbb{M}_{[16,16]} \wedge \mathbb{M}_{[16,32]} > \mathbb{M}_{[16,64]}. \tag{5.3}$$

These relations (5.3) may be described as follows: $\mathbb{P}_{32}$ is more richer in details about the road scene in terms of "easy" features and the drop in the performance may mean that $\mathbb{P}_{64}$ is more noisy than $\mathbb{P}_{32}$.

There is another interesting observation may be obtained if to look at the heat maps columns wise from the top to the corresponding side diagonal element, starting from the leftmost column:

$$\begin{aligned}
&\mathbb{M}_{64,4} < \mathbb{M}_{32,4} < \mathbb{M}_{16,4} < \mathbb{M}_{4,4}. \\
&\mathbb{M}_{64,16} < \mathbb{M}_{32,16} < \mathbb{M}_{16,16} \wedge \mathbb{M}_{16,16} > \mathbb{M}_{16,4}. \\
&\mathbb{M}_{64,32} < \mathbb{M}_{32,32} \wedge \mathbb{M}_{32,32} > \mathbb{M}_{32,16} \wedge \mathbb{M}_{32,32} > \mathbb{M}_{32,4}.
\end{aligned} \tag{5.4}$$

All in all, results from Eq. (5.4) may explained with covariance shift between the source $\mathbb{P}_s$ and the target $\mathbb{P}_t$ distributions when $s \neq t$, namely the model adjusted to the distribution of the training data $\mathbb{P}_s$ and performs in the best way when operating on its "native" distribution. The sequences of left inequality (i.e. ones with $<$ signs) intuitively may be explained expanding the covariance shift a little as follows: model trained on higher resolution fitted to data reach in detail and intermediate beams, and lower-beamed data happens to be too noisy for it.

Both observations Eq. (5.2) and Eq. (5.4) give upper and lower bounds on performance that successful domain adaptation may lie within. Consider referring to $\mathrm{OD}_{64,16}$ and $\mathrm{OD}_{16,16}$ training/evaluation pairs in all three difficulty cases. Indeed, $\mathrm{OD}_{64,16}$ represents performance on direct applying pre-trained model on 64-beamed data to 16-beamed data, and thus, it is the lower-bound of performance above which domain adaptation may be considered as successful. And $\mathrm{OD}_{16,16}$ represents performance of model trained and evaluated on the same 16-beamed point clouds, it is upper-bound because model trained and evaluated on data sharing the same distribution. The difference between upper and lower bounds tends to increase with increasing of KITTI scene difficulty as shown in Table 5.1. Hereinafter we refer to model that gives the lower bound on performance as "baseline" and the upper bound as "oracle".

	Easy	Medium	Hard
$\mathrm{AOS}_{64,16}$ (baseline)	64,66	52,51	49,33
$\mathrm{AOS}_{16,16}$ (oracle)	79,07	66,99	64,66
Baseline-oracle gap	14,41	14,48	15,33

Table 5.1: Increasing gap between lower and upper bounds w.r.t. KITTI difficulty level

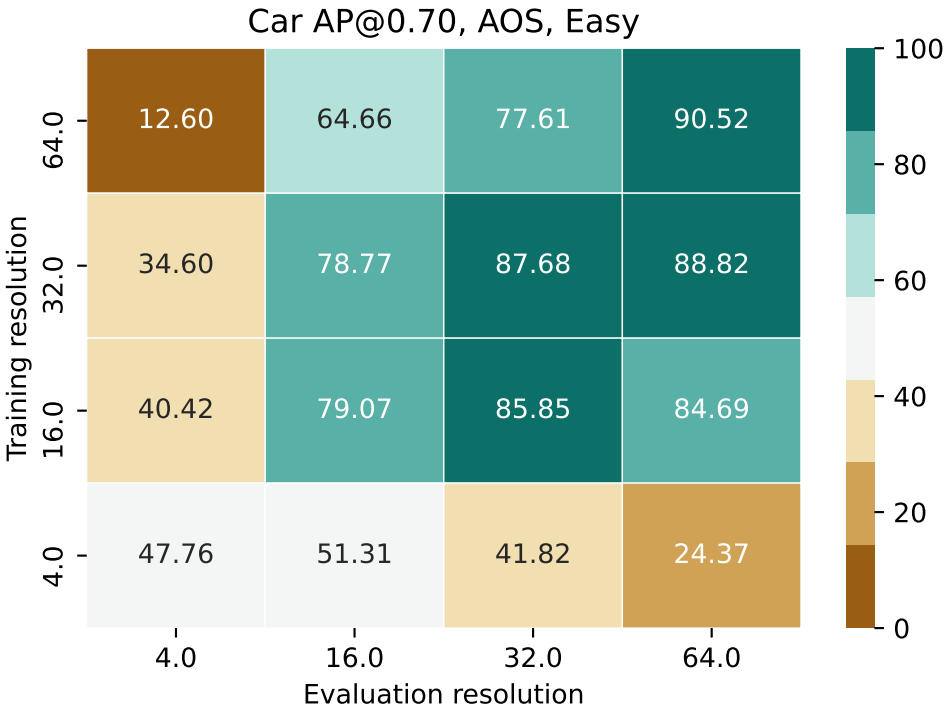


Figure 5.1: KITTI AOS@70, "Easy" mode

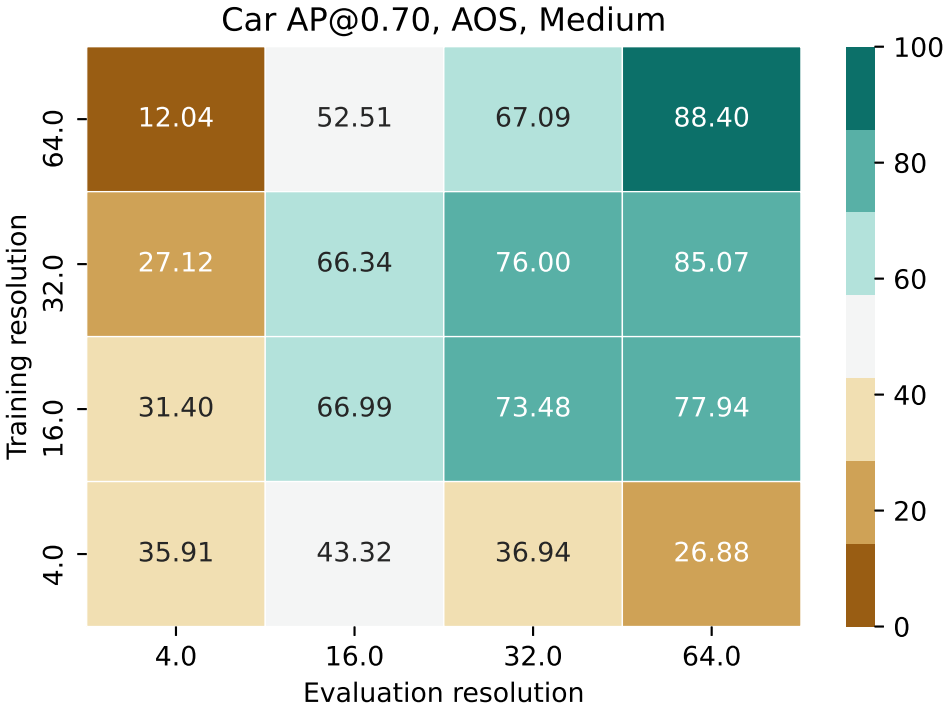


Figure 5.2: KITTI AOS@70, "Medium" mode

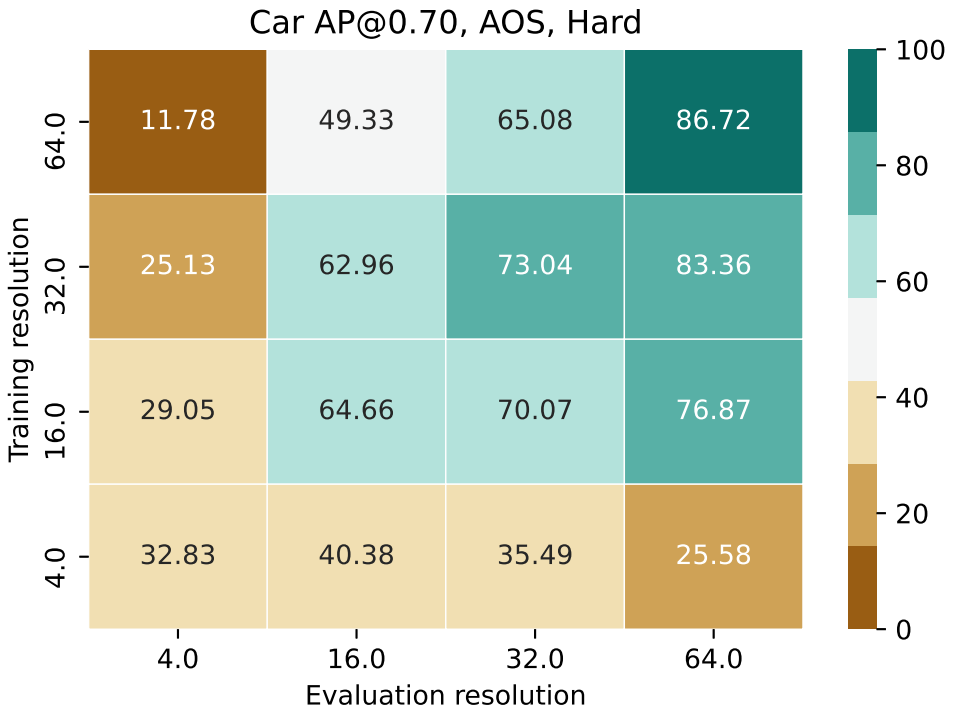


Figure 5.3: KITTI AOS@70, "Hard" mode

5.2 Domain Adaptation

In this section we examine, firstly, how quality of domain adaptation changes during training procedure by visualizing it in Fig. 5.4, Fig. 5.5 and Fig. 5.6 and, secondly, compare the best domain adaptation gap elimination between baseline and oracle performances: for AOS@70 results are shown in Section 5.2 whereas all other metrics are listed in Appendix B.

For that, during the models' training we evaluated their performance at the following checkpoints: 1, 5, 10, 20, 50, 100 and 150 training iterations, which is, taking into account the batch size, the same as checkpoints after adaptation on 4, 20, 40, 80, 200, 400 and 600 training samples (point clouds).

From the plots in Figs. 5.4, 5.5 and 5.6 it may be seen that `PostPFN + GP` shows the worst performance dropping it down rapidly, so that we may conclude that it does not produce domain adaptation at all. Almost the same holds for `PostPFN + MKMMD` which performs slightly better that `PostPFN + GP`, however, according to the best checkpoint results in Section 5.2 it does produce very weak domain adaptation to 16-beamed data. The reason for such a poor adaptation is that `PostPFN` feature extractor is too shallow and does not generalize the road scene much. Another notable detail that can be seen in all the graphs is that MKMMD degradation is flatter in comparison with GP setting, this may be explained with the fact that MKMMD tries to minimize distance between distributions' means in RKHS whereas Wasserstein distance modifies them as the whole, namely process of learning with WGAN is more fragile.

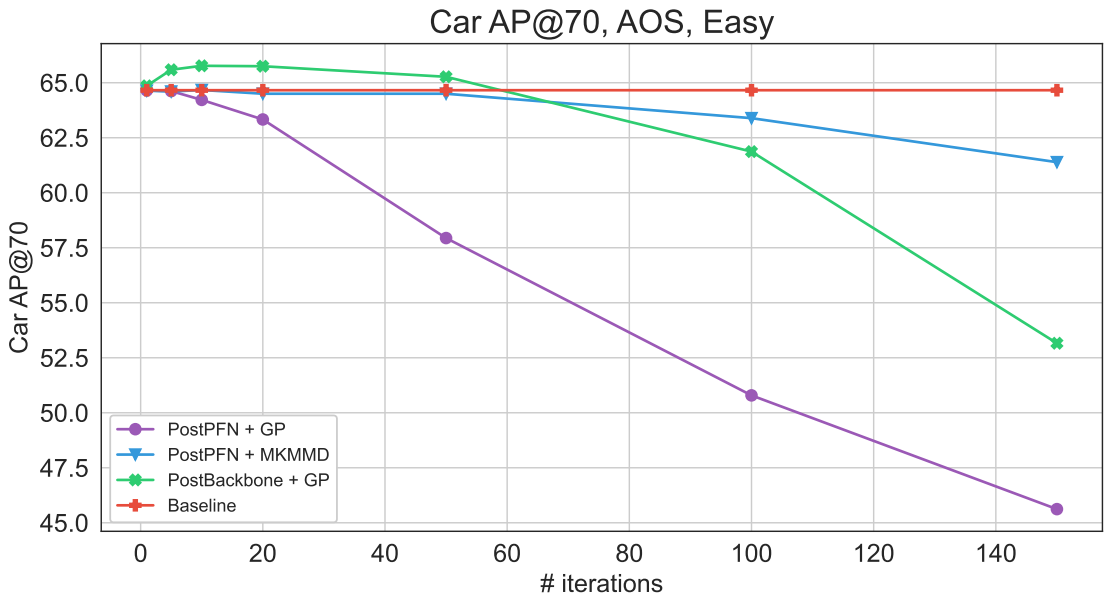


Figure 5.4: KITTI AOS evaluation per number of adaptation iterations, "Easy" mode

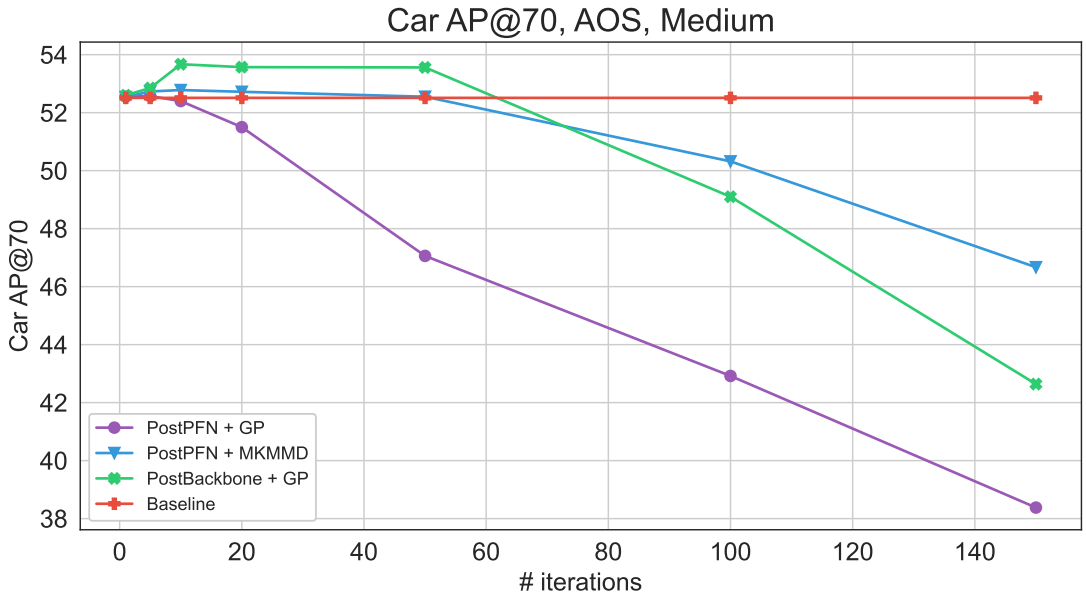


Figure 5.5: KITTI AOS evaluation per number of adaptation iterations, "Hard" mode

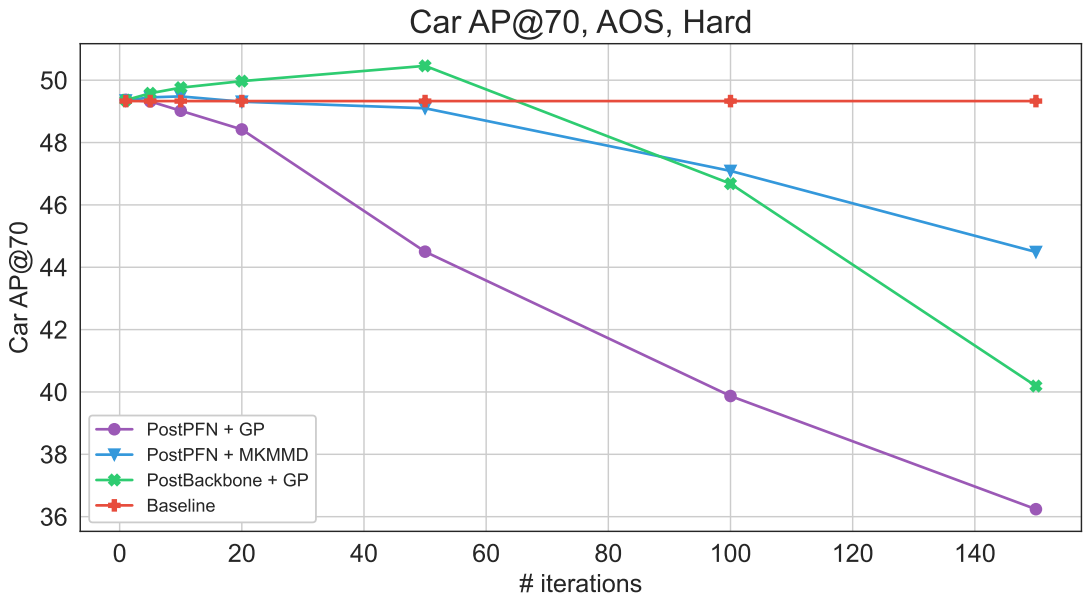


Figure 5.6: KITTI AOS evaluation per number of adaptation iterations, "Hard" mode

The most successful model for domain adaptation was `Postbackbone + GP` which achieved gain in performance at around 1.11 mAP AOS@70 in all difficulty regimes Section 5.2. It is interesting to notice the manner in which domain adaptation is held for this model. For the "Easy" mode Fig. 5.4 obtaining the best result is achieved rapidly after 5 training iterations with rather rapid decrease afterwards, for the "Medium" mode Fig. 5.5 it is the number of samples for getting maximum performance but decrease is much less rapid, for the "Hard" regime Fig. 5.6 it is steadily up to the 50-th iterations. This may be explained by density of point cloud that are away from sensor, and the "Hard" mode is characterized by highly sparse pillars, and that up to some point of training we indeed bring the useful information to model.

Model	Easy	Medium	Hard
Baseline	64.66	52.51	49.33
PostPFN + GP	64.65 (−0.01)	52.57 (+0.06)	49.37 (+0.04)
PostPFN + MKMMD	64.67 (+0.02)	52.78 (+0.27)	49.48 (+0.15)
PostBackbone + GP	**65.77** (+1.11)	**53.67** (+1.15)	**50.46** (+1.13)
Oracle	79.07	66.99	64.66

Table 5.2: Results of domain adaptation, AOS@70. Numbers in brackets demonstrate gain in performance w.r.t. baseline model. Numbers in bold stand for the best performance among domain adaptation models.

Chapter 6

Summary

In accordance with the purpose and objectives of the work, the following results were obtained:

- Carried out an analysis of state-of-the-art solutions for two topics: 3D object detection for autonomous vehicles and domain adaptation. As a result of which was investigated various problems formulations in this field of knowledge and methods for solving them. Moreover, an up-to-date research topic was chosen and formulated.
- Carried out exhaustive experiments to study the portability of model weights between different domains, which are produced by LiDAR sensors with different resolutions using PointPillars architecture on KITTI benchmark. With these experiments, we demonstrated the impossibility of directly using a trained model based on data from one sensor to data from another sensor without a significant loss of detection quality. And thus showed the importance of domain adaptation applicable to object detection on road scenes.
- For the first time in the literature conducted experiments on cross-sensor domain adaptation for the entire road scenes. We were in-place adapting pre-trained on 64-beamed LiDAR data to operate on 16-beamed LiDAR on example of PointPillars model.
- Domain adaptation was conducted on two different parts of PointPillars architecture: Pillar Feature Network and combination of Pillar Feature Network and 2D Backbone CNN. For eliminating distribution discrepancy between domain and source distributions we used adversarial training with Wasserstein GAN with gradient penalty (GP) and multi kernel maximum mean discrepancy (MKMMD).
- The best model `PostBackbone + GP` increased performance in every metric for KITTI benchmark in comparison to the baseline model, i.e. direct applying the pre-trained on source domain model to target domain. In particular, increase in **AOS** and **3D** reaches more 1.1 and 0.75 mAP points for every KITTI difficulty level correspondingly.

6.1 Future Work

There are several improvements that may be incorporated as:

- Instead of modifying the model for 3D object detection in-place, one may try changing the whole architecture incorporating specialized and deeper neural network module for domain adaptation in order to gather more meaningful point features.

- Divide domain adaptation process into three parts that roughly correspond to different KITTI difficulty regimes, namely carry out adaptation for different parts of the point cloud, depending on the distance from the scanner, separately.
- Experiment with adding noise to input point clouds or low-level features for a more robust adversarial training.

Bibliography

[1] Martıén Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[3] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *arXiv e-prints*, arXiv:1411.1792 (Nov. 2014), arXiv:1411.1792. arXiv: 1411.1792 [cs.LG].

[4] Xiaozhi Chen et al. "Multi-View 3D Object Detection Network for Autonomous Driving". In: *arXiv e-prints*, arXiv:1611.07759 (Nov. 2016), arXiv:1611.07759. arXiv: 1611.07759 [cs.CV].

[5] A. Gretton et al. "Covariate shift and local learning by distribution matching". In: *Dataset Shift in Machine Learning*. Cambridge, MA, USA: MIT Press, 2009, pp. 131–160.

[6] Yurong You et al. "Pseudo-LiDAR++: Accurate Depth for 3D Object Detection in Autonomous Driving". In: *arXiv e-prints*, arXiv:1906.06310 (June 2019), arXiv:1906.06310. arXiv: 1906.06310 [cs.CV].

[7] 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.

[8] Itzik Ben Shabat. *3D POINT CLOUD CLASSIFICATION USING DEEP LEARNING – RECENT WORKS*. URL: http://www.itzikbs.com/3d-point-cloud-classification-using-deep-learning.

[9] Charles R Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *arXiv preprint arXiv:1612.00593* (2016).

[10] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. "Deep Learning with Sets and Point Clouds". In: *arXiv e-prints*, arXiv:1611.04500 (Nov. 2016), arXiv:1611.04500. arXiv: 1611.04500 [stat.ML].

[11] Roman Klokov and Victor Lempitsky. "Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models". In: *arXiv e-prints*, arXiv:1704.01222 (Apr. 2017), arXiv:1704.01222. arXiv: 1704.01222 [cs.CV].

[12] Charles R. Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *arXiv e-prints*, arXiv:1706.02413 (June 2017), arXiv:1706.02413. arXiv: 1706.02413 [cs.CV].

[13] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv e-prints*, arXiv:1502.03167 (Feb. 2015), arXiv:1502.03167. arXiv: 1502.03167 [cs.LG].

[14] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[15] Max Jaderberg et al. "Spatial Transformer Networks". In: *arXiv e-prints*, arXiv:1506.02025 (June 2015), arXiv:1506.02025. arXiv: 1506.02025 [cs.CV].

[16] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *arXiv e-prints*, arXiv:1506.01497 (June 2015), arXiv:1506.01497. arXiv: 1506.01497 [cs.CV].

[17] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". In: *arXiv e-prints*, arXiv:1711.06396 (Nov. 2017), arXiv:1711.06396. arXiv: 1711.06396 [cs.CV].

[18] Alex H. Lang et al. "PointPillars: Fast Encoders for Object Detection from Point Clouds". In: *arXiv e-prints*, arXiv:1812.05784 (Dec. 2018), arXiv:1812.05784. arXiv: 1812.05784 [cs.LG].

[19] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv e-prints*, arXiv:1603.07285 (Mar. 2016), arXiv:1603.07285. arXiv: 1603.07285 [stat.ML].

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[21] Özgür Erkent and Christian Laugier. "Semantic Segmentation with Unsupervised Domain Adaptation Under Varying Weather Conditions for Autonomous Vehicles". In: *IEEE Robotics and Automation Letters* (Mar. 2020), pp. 1–8. DOI: 10.1109/LRA.2020.2978666. URL: https://hal.inria.fr/hal-02502457.

[22] Yaroslav Ganin and Victor Lempitsky. "Unsupervised Domain Adaptation by Backpropagation". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1180–1189. URL: http://proceedings.mlr.press/v37/ganin15.html.

[23] Eric Tzeng et al. "Adversarial Discriminative Domain Adaptation". In: *arXiv e-prints*, arXiv:1702.05464 (Feb. 2017), arXiv:1702.05464. arXiv: 1702.05464 [cs.CV].

[24] Arthur Gretton et al. "Optimal kernel choice for large-scale two-sample tests". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1205–1213. URL: http://papers.nips.cc/paper/4727-optimal-kernel-choice-for-large-scale-two-sample-tests.pdf.

[25] Can Qin et al. "PointDAN: A Multi-Scale 3D Domain Adaption Network for Point Cloud Representation". In: *arXiv e-prints*, arXiv:1911.02744 (Nov. 2019), arXiv:1911.02744. arXiv: 1911.02744 [cs.CV].

[26] Xuelin Chen, Baoquan Chen, and Niloy J. Mitra. "Unpaired Point Cloud Completion on Real Scans using Adversarial Training". In: *arXiv e-prints*, arXiv:1904.00069 (Mar. 2019), arXiv:1904.00069. arXiv: 1904.00069 [cs.CV].

[27] Jonathan Sauder and Bjarne Sievers. "Self-Supervised Deep Learning on Point Clouds by Reconstructing Space". In: *arXiv e-prints*, arXiv:1901.08396 (Jan. 2019), arXiv:1901.08396. arXiv: 1901.08396 [cs.LG].

[28] Panos Achlioptas et al. *Learning Representations and Generative Models for 3D Point Clouds*. 2018. URL: https://openreview.net/forum?id=BJInEZsTb.

[29] Khaled Saleh et al. "Domain Adaptation for Vehicle Detection from Bird's Eye View LiDAR Point Cloud Data". In: *arXiv e-prints*, arXiv:1905.08955 (May 2019), arXiv:1905.08955. arXiv: 1905.08955 [cs.CV].

[30] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". In: *arXiv e-prints*, arXiv:1703.10593 (Mar. 2017), arXiv:1703.10593. arXiv: 1703.10593 [cs.CV].

[31] Antoine Gauthier, Victor Talpaert, and Bruno Monsuez. *Point clouds density impact on performance in 3D object detection*. 2019.

[32] Velodyne Lidar ®. *HDL-64E High Definition Real-Time 3D Lidar*. URL: https://velodynelidar.com/products/hdl-64e/#downloads.

[33] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: *arXiv e-prints*, arXiv:1701.07875 (Jan. 2017), arXiv:1701.07875. arXiv: 1701.07875 [stat.ML].

[34] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *arXiv e-prints*, arXiv:1704.00028 (Mar. 2017), arXiv:1704.00028. arXiv: 1704.00028 [cs.LG].

[35] Ilya Tolstikhin et al. "Wasserstein Auto-Encoders". In: *arXiv e-prints*, arXiv:1711.01558 (Nov. 2017), arXiv:1711.01558. arXiv: 1711.01558 [stat.ML].

[36] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *arXiv e-prints*, arXiv:1607.08022 (July 2016), arXiv:1607.08022. arXiv: 1607.08022 [cs.CV].

[37] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *Int. J. Comput. Vision* 88.2 (June 2010), pp. 303–338. ISSN: 0920-5691. DOI: 10.1007/s11263-009-0275-4. URL: https://doi.org/10.1007/s11263-009-0275-4.

[38] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[39] S. van der Walt, S. C. Colbert, and G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science Engineering* 13.2 (2011), pp. 22–30.

[40] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[41] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[42] Omry Yadan. *Hydra - A framework for elegantly configuring complex applications*. Github. 2019. URL: https://github.com/facebookresearch/hydra.

[43] Daniel Balouek et al. “Adding Virtualization Capabilities to the Grid’5000 Testbed”. In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov et al. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1_1.

[44] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv e-prints*, arXiv:1412.6980 (Dec. 2014), arXiv:1412.6980. arXiv: 1412.6980 [cs.LG].

Appendix

A Domain Adaptation Algorithm

Algorithm 1: Self-Supervised Domain Adaptation

Input : $\mathrm{X}_s, \mathrm{X}_s, \mathrm{E}_{\hat{\theta}}, \mathrm{E}_\theta, \mathrm{D}_\phi, n_{\text{crit}}, \mathrm{D} \in \{\text{WGAN}, \text{MKMMD}\}$, n — mini-batch size, λ_{ss} — self-supervision regularization multiplier, λ_{p} — gradient penalty multiplier.

Initialize $\theta = \hat{\theta}$.

Output: Domain adapted model $\mathrm{E}_{\tilde{\theta}}$.

if $\mathrm{D} = \text{WGAN}$ **then**
  initialize parameters of D_ϕ.

else if $\mathrm{D} = \text{MKMMD}$ **then**
  provide positive-definite kernel $\kappa(\star, \star)$.

while *θ is not converged (to some optimal $\tilde{\theta}$)* **do**
  Sample $\{\mathbf{x}_s^i\}_{i=1}^n \subset \mathrm{X}_s$ and $\{\mathbf{x}_s^i\}_{i=1}^n \subset \mathrm{X}_t$.
  Compute $\hat{\mathbf{f}}_s^i = \mathrm{E}_{\hat{\theta}}(\mathbf{x}_s^i)$, $\mathbf{f}_t^i = \mathrm{E}_\theta(\mathbf{x}_t^i)$ and $\mathbf{f}_s^i = \mathrm{E}_\theta(\mathbf{x}_t^i)$.
  if $\mathrm{D} = \text{WGAN}$ **then**
    for $k \leftarrow 1$ **to** n_{crit} **do**
      Sample a random number $\varepsilon \sim \text{Uniform}[0, 1]$.
      Set $\tilde{\mathbf{f}} = \varepsilon \hat{\mathbf{f}}_s + (1 - \varepsilon)\mathbf{f}_t$.
      Update D_ϕ by ascending:

$$\frac{1}{n}\sum_{i=1}^{n} \mathrm{D}_\phi(\hat{\mathbf{f}}_s^i) - \mathrm{D}_\phi(\mathbf{f}_t^i) - \lambda_{\text{p}} \left(\left\| \nabla_{\tilde{\mathbf{f}}} \mathrm{D}(\tilde{\mathbf{f}}^i) \right\|_2 - 1 \right)^2 .$$

      Update E_θ by descending:

$$\frac{1}{n}\sum_{i=1}^{n} \mathrm{D}_\phi(\mathbf{f}_t^i) + \lambda_{\text{ss}} \mathrm{L}_{\text{semi-supervised}}(\mathbf{f}_s^i, \hat{\mathbf{f}}_t^i).$$

    end
  else if $\mathrm{D} = \text{MKMMD}$ **then**
    Update E_θ by descending:

$$\frac{1}{n(n-1)}\sum_{j \neq k}^{n} \kappa(\mathbf{f}_t^j, \mathbf{f}_t^k) + \frac{1}{n(n-1)}\sum_{j \neq k}^{n} \kappa(\mathbf{f}_s^j, \mathbf{f}_s^k) - \frac{1}{n^2}\sum_{j,k}^{n} \kappa(\mathbf{f}_t^k, \hat{\mathbf{f}}_s^j) + \lambda_{\text{ss}} \frac{1}{n}\sum_{i=1}^{n} \mathrm{L}_{\text{semi-supervised}}(\mathbf{f}_t^i, \hat{\mathbf{f}}_s^i).$$

end

B On Resolution Altering: Performance Results

In this appendix we list all the metrics for experiments discussed in Section 5.1.

B.1 KITTI Car@50

Table B.1: Car AP@0.50, 0.50, 0.50. Training resolution = 64

	Easy				Medium				Hard			
Evaluation layers	64	32	16	4	64	32	16	4	64	32	16	4
3D	90.77	82.98	74.59	17.67	89.87	76.80	63.15	16.50	89.11	74.13	59.58	15.32
AOS	90.52	77.61	64.66	12.60	88.40	67.09	52.51	12.04	86.72	65.08	49.33	11.78
BBOX	90.66	79.10	66.82	13.28	88.84	69.50	55.19	12.78	87.50	68.03	52.45	12.63
BEV	90.77	87.19	78.80	30.27	90.01	79.88	68.71	25.44	89.35	78.35	66.16	24.17

Table B.2: Car AP@0.50, 0.50, 0.50. Training resolution = 32

	Easy				Medium				Hard			
Evaluation layers	64	32	16	4	64	32	16	4	64	32	16	4
3D	89.30	90.00	82.06	48.35	88.00	80.34	72.55	37.97	87.49	79.43	70.33	34.69
AOS	88.82	87.68	78.77	34.60	85.07	76.00	66.34	27.12	83.36	73.04	62.96	25.13
BBOX	88.97	88.53	79.73	38.60	85.84	77.60	68.14	30.48	84.56	75.17	65.18	28.40
BEV	89.74	90.16	83.43	55.21	88.61	81.64	76.09	43.80	88.16	80.73	74.29	40.28

Table B.3: Car AP@0.50, 0.50, 0.50. Training resolution = 16

	Easy				Medium				Hard			
Evaluation layers	64	32	16	4	64	32	16	4	64	32	16	4
3D	86.72	89.27	86.39	54.59	85.61	79.58	74.91	43.30	85.08	78.85	72.78	39.95
AOS	84.69	85.85	79.07	40.42	77.94	73.48	66.99	31.40	76.87	70.07	64.66	29.05
BBOX	84.92	86.84	79.97	44.24	78.81	75.30	68.95	34.72	78.13	72.31	67.28	32.50
BEV	87.54	89.58	87.79	60.42	86.44	81.51	77.89	48.08	86.28	80.71	75.95	45.26

Table B.4: Car AP@0.50, 0.50, 0.50. Training resolution = 4

	Easy				Medium				Hard			
Evaluation layers	64	32	16	4	64	32	16	4	64	32	16	4
3D	41.07	68.16	74.49	60.50	49.66	60.76	64.24	46.83	47.11	57.33	60.15	42.94
AOS	24.37	41.82	51.31	47.76	26.88	36.94	43.32	35.91	25.58	35.49	40.38	32.83
BBOX	25.95	44.57	54.84	52.39	30.18	40.15	46.94	39.80	28.84	38.90	44.47	36.82
BEV	43.93	73.90	78.77	63.87	54.49	67.71	69.55	50.16	52.60	64.44	66.18	46.31

B.2 KITTI Car@70

Table B.5: Car AP@0.70, 0.70, 0.70. Training resolution = 64

	Easy				Medium				Hard			
Evaluation resolution	64	32	16	4	64	32	16	4	64	32	16	4
3D	79.61	58.25	42.93	2.00	74.52	47.29	35.00	4.67	69.29	45.02	31.80	4.70
AOS	90.52	77.61	64.66	12.60	88.40	67.09	52.51	12.04	86.72	65.08	49.33	11.78
BBOX	90.66	79.10	66.82	13.28	88.84	69.50	55.19	12.78	87.50	68.03	52.45	12.63
BEV	89.47	77.97	67.16	15.70	86.81	69.36	56.57	14.69	84.33	66.22	52.62	13.99

Table B.6: Car AP@0.70, 0.70, 0.70. Training resolution = 32

	Easy				Medium				Hard			
Evaluation resolution	64	32	16	4	64	32	16	4	64	32	16	4
3D	72.14	69.82	61.92	14.35	64.66	57.28	48.00	12.25	64.14	55.77	45.89	10.57
AOS	88.82	87.68	78.77	34.60	85.07	76.00	66.34	27.12	83.36	73.04	62.96	25.13
BBOX	88.97	88.53	79.73	38.60	85.84	77.60	68.14	30.48	84.56	75.17	65.18	28.40
BEV	87.95	83.33	78.38	38.92	84.84	74.74	66.76	30.87	83.12	70.13	62.94	28.04

Table B.7: Car AP@0.70, 0.70, 0.70. Training resolution = 16

	Easy				Medium				Hard			
Evaluation resolution	64	32	16	4	64	32	16	4	64	32	16	4
3D	60.95	69.69	64.08	21.76	58.83	56.50	49.29	18.46	55.86	54.97	47.74	16.61
AOS	84.69	85.85	79.07	40.42	77.94	73.48	66.99	31.40	76.87	70.07	64.66	29.05
BBOX	84.92	86.84	79.97	44.24	78.81	75.30	68.95	34.72	78.13	72.31	67.28	32.50
BEV	84.21	85.11	78.87	45.07	80.58	74.58	68.26	36.38	78.77	70.49	65.95	33.13

Table B.8: Car AP@0.70, 0.70, 0.70. Training resolution = 4

	Easy				Medium				Hard			
Evaluation resolution	64	32	16	4	64	32	16	4	64	32	16	4
3D	10.50	15.66	22.82	25.58	12.41	19.61	20.64	20.55	11.70	18.84	20.12	19.57
AOS	24.37	41.82	51.31	47.76	26.88	36.94	43.32	35.91	25.58	35.49	40.38	32.83
BBOX	25.95	44.57	54.84	52.39	30.18	40.15	46.94	39.80	28.84	38.90	44.47	36.82
BEV	37.67	64.42	69.31	49.21	46.14	57.71	59.53	39.24	43.88	53.54	55.12	35.66

B.3 KITTI Car COCO

Table B.9: Car coco AP@0.50:0.05:0.95. Training resolution = 64

	Easy				Medium				Hard			
Evaluation resolution	64	32	16	4	64	32	16	4	64	32	16	4
3D	55.35	43.51	34.71	5.94	52.14	38.39	28.91	6.26	50.84	36.69	26.63	6.00
AOS	69.03	56.35	48.67	12.32	65.48	49.38	40.42	11.32	64.40	47.86	38.16	11.04
BBOX	69.13	57.46	50.42	14.17	65.76	51.23	42.68	12.83	64.94	50.15	40.82	12.61
BEV	69.72	59.47	50.40	13.52	66.26	53.09	42.74	12.42	64.72	50.71	40.05	11.81

Table B.10: Car coco AP@0.50:0.05:0.95. Training resolution = 32

	Easy				Medium				Hard			
Evaluation resolution	64	32	16	4	64	32	16	4	64	32	16	4
3D	51.11	50.38	44.04	17.63	48.83	43.59	37.05	14.18	47.48	41.92	34.99	12.82
AOS	65.72	64.10	58.11	26.44	62.52	55.81	49.19	21.09	61.63	54.23	47.33	19.63
BBOX	65.83	64.67	58.84	29.96	63.05	56.96	50.55	24.17	62.49	55.84	49.11	22.77
BEV	67.12	64.29	58.19	29.86	64.45	56.81	50.62	24.01	63.86	55.27	48.72	21.74

Table B.11: Car coco AP@0.50:0.05:0.95. Training resolution = 16

	Easy				Medium				Hard			
Evaluation resolution	64	32	16	4	64	32	16	4	64	32	16	4
3D	45.04	49.15	46.25	21.81	43.68	42.76	38.28	17.68	42.89	41.08	37.07	16.20
AOS	58.24	61.75	59.46	31.44	56.23	54.93	49.93	24.98	55.45	52.92	48.42	23.39
BBOX	58.39	62.39	60.18	34.81	56.94	56.26	51.45	27.95	56.51	54.64	50.45	26.58
BEV	62.00	62.97	60.14	33.74	60.08	56.38	51.58	27.55	59.15	54.98	50.15	25.66

Table B.12: Car coco AP@0.50:0.05:0.95. Training resolution = 4

	Easy				Medium				Hard			
Evaluation resolution	64	32	16	4	64	32	16	4	64	32	16	4
3D	15.01	23.58	28.06	24.85	17.58	21.60	24.38	20.34	16.58	20.52	22.75	18.80
AOS	19.94	33.89	39.33	34.30	22.78	29.89	33.67	27.18	21.70	28.42	31.66	25.38
BBOX	21.06	36.20	41.83	37.94	25.45	32.74	36.62	30.34	24.48	31.46	34.97	28.74
BEV	26.39	45.84	50.08	37.17	32.75	41.81	44.03	29.70	31.28	39.36	41.47	27.43

C mAP 3D@70 Domain Adaptation Results

C.1 3D@70 mAP Heat Maps

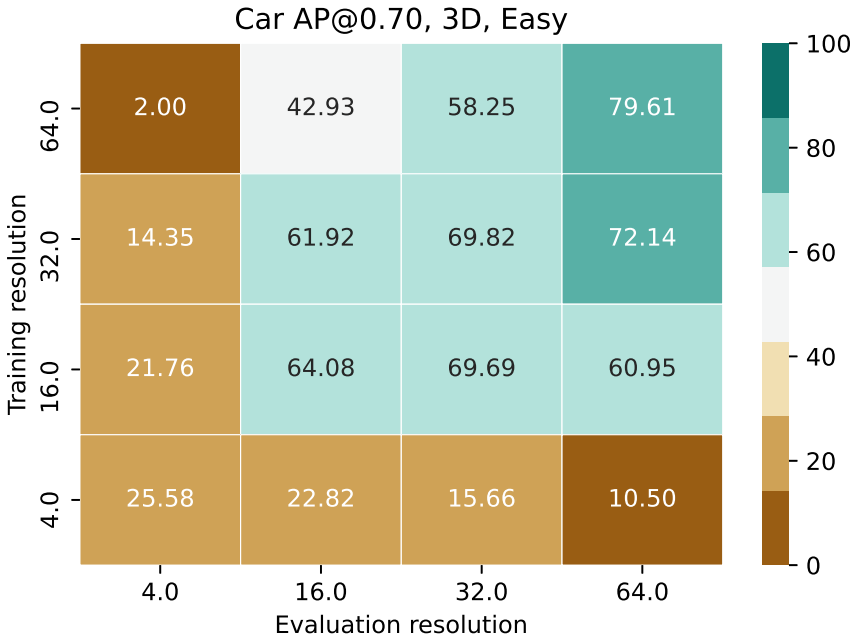


Figure A.1: KITTI mAP 3D@70, "Easy" mode

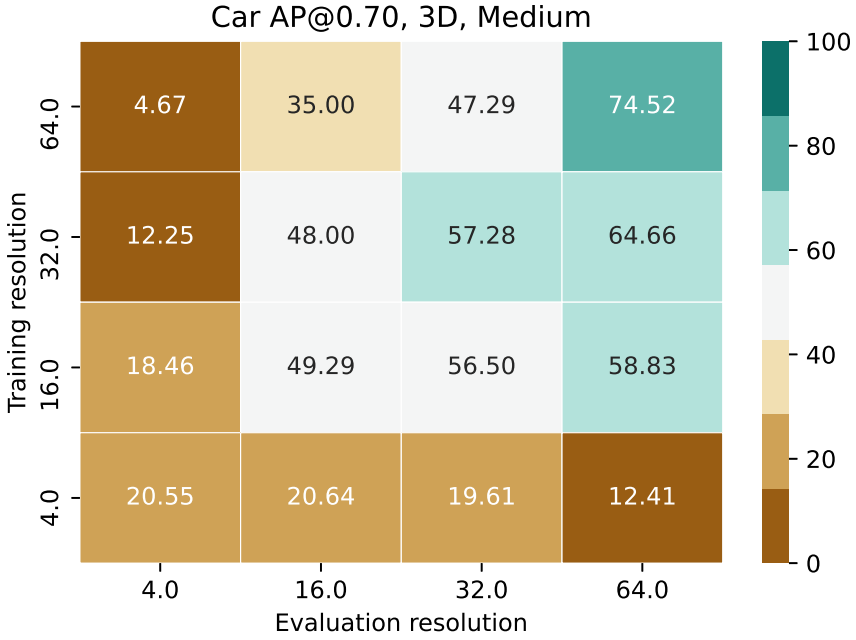


Figure A.2: KITTI mAP 3D@70, "Medium" mode

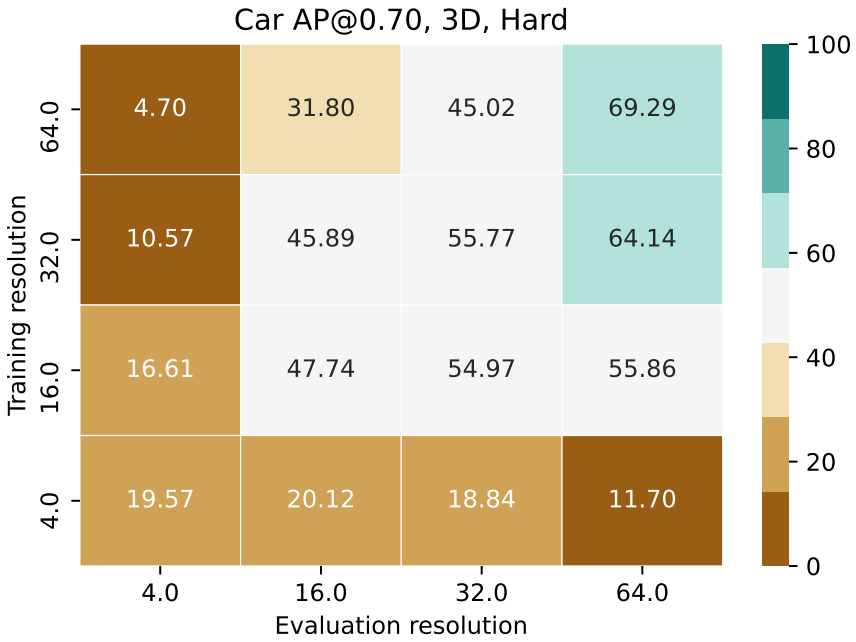


Figure A.3: KITTI mAP 3D@70, "Hard" mode

C.2 3D@70 mAP Top Results

Model	Easy	Medium	Hard
Baseline	42.93	35.00	31.80
PostPFN + GP	42.70 (−0.23)	34.82 (−0.18)	31.52 (−0.28)
PostPFN + MKMMD	42.87 (−0.06)	35.03 (+0.03)	31.89 (+0.09)
PostBackbone + GP	**43.69** (+0.75)	**35.79** (+0.79)	**32.65** (+0.75)
Oracle	64.08	49.29	47.74

Table C.1: Results of domain adaptation, 3D@70 mAP. Numbers in brackets demonstrate gain in performance w.r.t. baseline model. Numbers in bold stand for the best performance among domain adaptation models.

D Top Results for Domain Adaptation for Different KITTI Metrics

In the tables are presented top domain adaptation performances for each model per each metric type and difficulty level. The first number in tuple represents score, the second one — iteration number at which the corresponding score was achieved.

D.1 PostPFN + GP

Table D.1: PostPFN + GP, Car AP@0.70

	Easy	Medium	Hard
3D	(42.7, 1)	(34.82, 1)	(31.52, 1)
AOS	(64.65, 1)	(52.57, 5)	(49.37, 1)
BBOX	(66.82, 1)	(55.23, 1)	(52.49, 1)
BEV	(68.12, 50)	(56.96, 50)	(53.16, 50)

Table D.2: PostPFN + GP, Car AP@0.50

	Easy	Medium	Hard
3D	(74.54, 1)	(63.08, 1)	(59.56, 1)
AOS	(64.65, 1)	(52.57, 5)	(49.37, 1)
BBOX	(66.82, 1)	(55.23, 1)	(52.49, 1)
BEV	(80.45, 150)	(69.33, 150)	(66.71, 150)

Table D.3: PostPFN + GP, Car COCO

	Easy	Medium	Hard
3D	(34.83, 20)	(29.27, 20)	(27.12, 20)
AOS	(48.64, 1)	(40.4, 1)	(38.15, 1)
BBOX	(50.4, 1)	(42.65, 1)	(40.8, 1)
BEV	(52.16, 50)	(44.06, 150)	(41.27, 150)

D.2 PostPFN + MMD

Table D.4: PostPFN + MKMMD, Car AP@0.70

	Easy	Medium	Hard
3D	(42.7, 1)	(34.82, 1)	(31.52, 1)
AOS	(64.65, 1)	(52.57, 5)	(49.37, 1)
BBOX	(66.82, 1)	(55.23, 1)	(52.49, 1)
BEV	(68.12, 50)	(56.96, 50)	(53.16, 50)

Table D.5: PostPFN + MKMMD, Car AP@0.50

	Easy	Medium	Hard
3D	(74.54, 1)	(63.08, 1)	(59.56, 1)
AOS	(64.65, 1)	(52.57, 5)	(49.37, 1)
BBOX	(66.82, 1)	(55.23, 1)	(52.49, 1)
BEV	(80.45, 150)	(69.33, 150)	(66.71, 150)

Table D.6: PostPFN + MKMMD, Car COCO

	Easy	Medium	Hard
3D	(34.83, 20)	(29.27, 20)	(27.12, 20)
AOS	(48.64, 1)	(40.4, 1)	(38.15, 1)
BBOX	(50.4, 1)	(42.65, 1)	(40.8, 1)
BEV	(52.16, 50)	(44.06, 150)	(41.27, 150)

D.3 PostBackbone + GP

Table D.7: PostBackbone + GP, Car AP@0.70

	Easy	Medium	Hard
3D	(42.7, 1)	(34.82, 1)	(31.52, 1)
AOS	(64.65, 1)	(52.57, 5)	(49.37, 1)
BBOX	(66.82, 1)	(55.23, 1)	(52.49, 1)
BEV	(68.12, 50)	(56.96, 50)	(53.16, 50)

Table D.8: PostBackbone + GP, Car AP@0.50

	Easy	Medium	Hard
3D	(74.54, 1)	(63.08, 1)	(59.56, 1)
AOS	(64.65, 1)	(52.57, 5)	(49.37, 1)
BBOX	(66.82, 1)	(55.23, 1)	(52.49, 1)
BEV	(80.45, 150)	(69.33, 150)	(66.71, 150)

Table D.9: PostBackbone + GP, Car COCO

	Easy	Medium	Hard
3D	(34.83, 20)	(29.27, 20)	(27.12, 20)
AOS	(48.64, 1)	(40.4, 1)	(38.15, 1)
BBOX	(50.4, 1)	(42.65, 1)	(40.8, 1)
BEV	(52.16, 50)	(44.06, 150)	(41.27, 150)