Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2018

Automotive 3D Object Detection Without Target Domain Annotations

Erik Linder-Norén and Fredrik Gustafsson

Master of Science Thesis in Electrical Engineering

Automotive 3D Object Detection Without Target Domain Annotations

Erik Linder-Norén and Fredrik Gustafsson

LiTH-ISY-EX--18/5138--SE

Supervisors: **Gustav Häger**
ISY, Linköping University
Eskil Jörgensen, Amrit Krishnan
Zenuity AB

Examiner: **Michael Felsberg**
ISY, Linköping University

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

In this thesis we study a perception problem in the context of autonomous driving. Specifically, we study the computer vision problem of 3D object detection, in which objects should be detected from various sensor data and their position in the 3D world should be estimated. We also study the application of Generative Adversarial Networks in domain adaptation techniques, aiming to improve the 3D object detection model's ability to transfer between different domains.

The state-of-the-art Frustum-PointNet architecture for LiDAR-based 3D object detection was implemented and found to closely match its reported performance when trained and evaluated on the KITTI dataset. The architecture was also found to transfer reasonably well from the synthetic SYN dataset to KITTI, and is thus believed to be usable in a semi-automatic 3D bounding box annotation process. The Frustum-PointNet architecture was also extended to explicitly utilize image features, which surprisingly degraded its detection performance. Furthermore, an image-only 3D object detection model was designed and implemented, which was found to compare quite favourably with current state-of-the-art in terms of detection performance.

Additionally, the PixelDA approach was adopted and successfully applied to the MNIST to MNIST-M domain adaptation problem, which validated the idea that unsupervised domain adaptation using Generative Adversarial Networks can improve the performance of a task network for a dataset lacking ground truth annotations. Surprisingly, the approach did however not significantly improve upon the performance of the image-based 3D object detection models when trained on the SYN dataset and evaluated on KITTI.

Acknowledgments

We would like to thank our industrial supervisors Eskil Jörgensen and Amrit Krishnan for their excellent support. They made performing this work remotely not only possible but also thoroughly enjoyable. Furthermore, we thank our examiner Michael Felsberg and supervisor Gustav Häger for their invaluable input during the writing of this thesis. Finally, we would like to express our gratitude towards Zenuity AB for giving us the opportunity and excellent resources to perform this work. A special thanks to Erik Rosén for always showing great interest in our work and helping us setting up the project.

Linköping, May 2018
Erik Linder-Norén and Fredrik Gustafsson

Contents

1 Introduction

An autonomous system is commonly divided into three separate modules: perception, planning and control. The perception module is tasked with sensing and creating a model of the environment, the planning module utilizes this model to decide on which future actions to take, and finally the control module is tasked with actuating the system to follow this plan.

In this thesis we study a perception problem in the context of autonomous driving. Specifically, we study the computer vision problem of 3D object detection, in which objects should be detected from various sensor data and their position in the 3D world should be estimated. We also study the application of Generative Adversarial Networks in domain adaptation techniques, aiming to improve the 3D object detection model's ability to generalize between different domains.

In this chapter we introduce the studied problem in general, motivate why it is of interest and formulate a set of research questions which we aim to answer in this thesis.

1.1 Background

Zenuity has access to a number of automotive datasets, including the internal ZEN dataset, which contains sensor data from both cameras and LiDAR sensors. This dataset is used to train and evaluate machine learning models for various perception tasks in the context of autonomous driving.

One such perception task is that of 3D object detection (3DOD). Compared to 2D object detection (2DOD), where a model is trained to detect and draw bounding boxes around objects of interest in the image plane, 3DOD also requires estimation of an object's size, heading and position in the 3D world. Concretely, the goal of 3DOD is to place oriented 3D bounding boxes (rectangular cuboids) in 3D space, which tightly contain the objects of interest. An example of a de-

sired 3DOD model output is visualized in Figure 1.1, showing ground truth 3D bounding boxes both in the image plane and in LiDAR data.

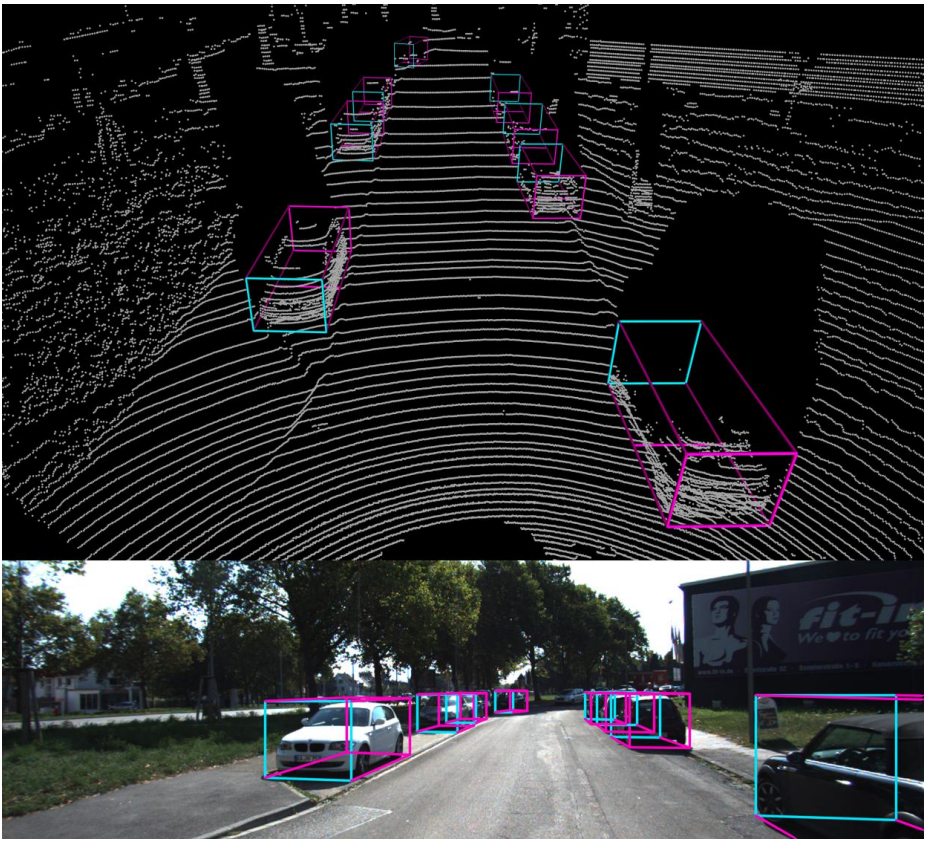

Figure 1.1: *Example desired output of a 3D object detection model, visualized in the image plane and in LiDAR data (upper part of the figure). The figure displays the ground truth 3D bounding boxes for the car object class in an example from the KITTI dataset [15]. The point cloud visualization was created using Open3D [64].*

Specifically, we wish to train a 3DOD model that only utilizes image data from a single forward-facing camera, as motivated by the low sensor cost. The model should perform well in real-world scenarios, as measured by its performance on the test subset of the ZEN dataset. Given annotated ground truth 3D bounding boxes on ZEN, such a model could be trained using supervised learning, as demonstrated in *e.g.* [43]. Annotating 3D bounding boxes is however a very time consuming and expensive process, and it would thus be highly beneficial if such a model could be trained without access to 3D ground truth. We will thus assume that ground truth 3D annotations not are available on ZEN.

There are however two other similar datasets available which do contain ground

truth 3D annotations, together with image data from a forward-facing camera and corresponding LiDAR point clouds. The first one is the publicly available KITTI dataset [15], and the second one is an internal synthetic dataset which we call SYN. The main question of interest in this thesis is thus how one can use KITTI, SYN and ZEN to train an image-only model for 3DOD that performs well on the test subset of ZEN.

One approach would be to just train an image-only model on KITTI and SYN, and then hope that it will generalize to perform well also on ZEN test. Image data in the KITTI and ZEN datasets is however collected using different camera sensors, and a model trained exclusively on one of the datasets is thus not believed to trivially also perform well on the other.

A second approach would be to again train an image-only model on KITTI and SYN, but this time also perform domain adaptation in some form. Domain adaptation is generally used to narrow the gap between a source domain and a target domain, and would in this case be used to improve the ability of the model trained on KITTI and SYN (source) to generalize to ZEN (target). A recent domain adaptation technique is to utilize Generative Adversarial Networks (GANs) [18] to essentially transform images from the source and target domains to more closely resemble each other. One can explicitly transform all source domain images to resemble the target domain, *e.g.* using the architecture presented in [66], and then train the model on the transformed source images. One can also follow the approach presented in [55], where the image encoder network is forced to learn an image embedding that is both domain invariant and suitable for the task in which the embedding is supposed to be utilized. In [55] the authors propose a network topology in which the encoder is tasked with producing an image embedding of the source image that will help the generator fool the discriminator into thinking that the translated source images produced by the generator are from the target domain. This embedding is also fed into a task network (in the paper the task network performs semantic segmentation), and the task network and the image encoder are then optimized to perform well given this task.

A different approach would be to train a LiDAR-only 3DOD model on KITTI and SYN, use this model to create 3D annotations on ZEN and then finally train an image-only model on ZEN using these 3D annotations. This approach seems promising since using LiDAR generally improves 3DOD performance significantly. The top 11 entries on the KITTI 3DOD leaderboard for cars [14] are for instance all using LiDAR. Since KITTI and ZEN were collected using similar LiDAR sensors, a LiDAR-only model is also believed to transfer relatively well between these two domains. To transfer from SYN to ZEN, *i.e.* from synthetic to real data, is still considered a nontrivial task. A LiDAR-only model is however believed to clearly outperform an image-only model in this regard since LiDAR provides a more general and geometric description of the vehicle surroundings.

Finally, one could also train an image-and-LiDAR model on KITTI and SYN and perform domain adaptation in some form, use this model to annotate ZEN and then train an image-only model on ZEN using these annotations. This approach seems promising since extending a LiDAR-only model to also utilize image information generally improves performance, as demonstrated in *e.g.* [63].

Theoretically, an image-and-LiDAR model should also have access to more information and thus provide the best possible performance. The question is however if adding image information, even when performing domain adaptation, will make the model more susceptible to the domain gap and thus less capable to transfer well between the domains.

1.2 Problem Formulation

In this thesis we will study the *"train 3DOD model utilizing LiDAR - use model to automatically annotate ZEN - train image-only 3DOD model on ZEN"* approach. Specifically, we will focus on the automatic annotation aspect. Our ultimate goal is thus to train a model utilizing LiDAR on KITTI and SYN that creates the best possible 3D bounding box annotations on ZEN.

Currently, we do however not have access to 3D ground truth annotations on the ZEN dataset, which makes it impossible to quantitatively evaluate these created 3D annotations. Instead of studying the transfer from KITTI and SYN to ZEN, we will thus instead study the transfer from SYN to KITTI.

Our specific goal is thus to, given SYN (images, LiDAR point clouds, 2D ground truth and 3D ground truth) and KITTI (images, LiDAR point clouds and 2D ground truth), train a 3D object detection model with maximum performance on KITTI. To achieve this, both a LiDAR-only model and an image-and-LiDAR model utilizing domain adaptation during training will be evaluated. The performance will be quantitatively measured by utilizing the 3D ground truth annotations on KITTI.

We aim to answer the following research questions in this thesis:

- By training a LiDAR-only model on SYN, what performance can be achieved on KITTI? How well does the model transfer from SYN to KITTI?
- If the LiDAR-only model is extended to also utilize image information, how is the performance on KITTI affected?
- If the extended image-and-LiDAR model is trained using domain adaptation, how is the performance on KITTI affected? How does the performance on KITTI compare with that of the LiDAR-only model?

Additionally, the thesis should result in a relatively complete literature review of related work in 3D object detection and domain adaptation techniques utilizing Generative Adversarial Networks.

1.3 Motivation

The general interest in 3DOD is motivated by the need for accurate 3D perception in subsequent areas of the autonomy stack. The output of 2DOD, a classified bounding box in the image plane, does in theory provide enough information to implement some of the features commonly found in advanced driving assistance

systems, such as *e.g.* automatic emergency breaking. Automatically performing more complex maneuvers, and ultimately obtaining a fully autonomous vehicle, does however require the system to plan and make decisions based on a 3D understanding of its environment. Since cameras are ubiquitous and significantly less expensive than LiDAR sensors, being able to create this 3D understanding with vision as the primary input modality would be highly beneficial from a financial point of view.

The specific problem studied in this thesis is motivated by the fact that if we can successfully train a model on SYN that transfers well to KITTI, this should also be doable from KITTI and SYN to ZEN. The method could thus be used to automatically annotate 3D bounding boxes on ZEN, or at least be used to generate proposal annotations on ZEN which then can then be fine-tuned by a human annotator. Either way, the method would dramatically decrease the annotation cost and thus enable training of an image-only 3DOD model with real-time performance on a large and diverse dataset, thus leading to good in-vehicle performance.

1.4 Delimitations

While a 3DOD model potentially could be trained taking data from various sensors as input, we are in this thesis only considering models utilizing LiDAR point clouds and image data from a single monocular camera.

Also, we are only considering models in which detection is performed independently on each individual set of sensor data. Extending the 3DOD model with a temporal component to perform joint detection and tracking is left as an interesting topic for future work.

Furthermore, while domain adaptation is a broad field with many different types of associated techniques, we are in this thesis only considering methods which explicitly utilize GANs.

To evaluate the quality of unsupervised image-to-image translations between a synthetic and real world street-view dataset we perform domain translation between the GTA 5 and Cityscapes dataset (Section 3.4). Because this involved performing image-to-image translations between two unpaired image datasets, pixelwise similarity measurements between generated images and target images were not possible, and because of time restrictions user studies in which the quality of generated images could be measured were not conducted.

1.5 Thesis Outline

The theoretical concepts that are of specific relevance for the thesis are covered in Chapter 2, together with a review of related work in 3D object detection and domain adaptation using GANs. Chapter 3 contains a detailed description of the methods which have been implemented and the datasets which have been utilized in the thesis. The results are presented in Chapter 4 and discussed in

more detail in Chapter 5, together with a discussion about possible future work. Finally, our conclusions are presented in Chapter 6.

The methods related to domain adaptation (Section 3.1-3.4) and 3D object detection (Section 3.5-3.8) were implemented in parallel by Erik Linder-Norén and Fredrik Gustafsson, respectively. Both authors then collaborated to implement the combined method described in Section 3.9.

2

Theory & Related Work

In this chapter we cover the theoretical concepts that are of specific interest for the thesis, and present a review of related work in 3D object detection and domain adaptation using GANs. The reader is assumed to be familiar with basic deep learning concepts. For a complete presentation of deep learning fundamentals and its application in computer vision, see *e.g.* [19].

Since we in this thesis aim to evaluate 3DOD models which take LiDAR data as input, the chapter begins with a brief overview of LiDAR sensors in Section 2.1, before a review of deep learning based methods for LiDAR data processing is presented in Section 2.2.

Most work on 3D object detection builds on certain ideas and techniques applied in 2D object detection, why an overview of this field is presented in Section 2.3, preceding the review of related work in 3D object detection found in Section 2.4.

An introduction to GANs is then found in Section 2.5, before a review of related work in domain adaptation utilizing GANs finally is presented in Section 2.6.

2.1 LiDAR Sensors

LiDAR is an acronym for *light detection and ranging* and the basic functionality of a LiDAR sensor is very similar to that of a radar or a sonar. The sensor contains two main elements: an emitter and a detector. The emitter repeatedly emits a laser light pulse which travels until it hits a target and is in part reflected back towards the emitter. This return light pulse is detected by the detector, and by measuring the time difference between the emitted and detected pulse the distance to the hit target is obtained.

LiDAR sensors commonly used in autonomous vehicle applications, such as

the Velodyne HDL-64E [25], typically contain multiple emitter-detector pairs mounted at slightly different vertical angles in a rotating housing, with each pair taking multiple range measurements in each revolution. This enables the sensor to measure the distance to thousands of points per second in a 360° horizontal field of view, resulting in a LiDAR point cloud describing the sensor's 3D environment.

The Velodyne HDL-64E has 64 emitter-detector pairs, also called channels, which gives it a 26.5° vertical field of view. It has a range of 120 m with a typical distance error of less than 3 cm, rotates at 5-15 Hz and outputs one million distance points per second. A point cloud captured by this sensor is shown together with an image of the corresponding scene in Figure 2.1.

Formally, a LiDAR point cloud is a set of n points $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^4$, where each point $p_i = (x_i, y_i, z_i, r_i) \in \mathbb{R}^4$ contains its 3D coordinates (x_i, y_i, z_i) together with the received reflectance value r_i.

2.2 Deep Learning on Point Clouds

As deep learning has become the prominent technique for image-based computer vision over the past few years, there has also been an increasing interest in applying learning-based methods to process geometric data such as point clouds. Until quite recently, the most common approach has been to preprocess the point clouds to transform them into a structure suited for existing deep learning algorithms, and then apply these on the transformed data.

One such preprocessing technique is to represent the point cloud as a collection of projected 2D image views, to which conventional CNNs can be applied. An example application of this approach is presented by Wu *et al.* in [62], where a network learns to segment vehicles, cyclists and pedestrians from a spherically projected front-view of the point cloud. Another example is presented by Caltagirone *et al.* in [6], where a fully convolutional network is applied to top-view projections of point clouds to segment the drivable road surface in street scenes. Another type of preprocessing is to discretize the point cloud into a volumetric 3D grid and then apply 3D convolutions, as demonstrated by Maturana and Scherer in [41]. In this work, a network learns to classify point cloud segments as either background or a specific object class.

Spatial information will however to some extent always be lost in such preprocessing, and in the case of 3D grid discretization the high computational cost of 3D convolution is a limiting factor. Motivated by these issues, learning-based architectures for processing of raw point clouds have recently been developed.

The pioneering architecture in this line of work is PointNet, which was introduced by Qi *et al.* in [48]. In this work, the authors demonstrated PointNet's applicability for the tasks of both classification and segmentation of point clouds. The architecture takes a raw point cloud $P = \{p_1, \dots, p_n\}$ as input, *i.e.* without discretization or image view projection, and can learn both a local feature vector for each point p_i and a global feature vector representing the entire point cloud P. A schematic overview of the basic PointNet architectures for both classification

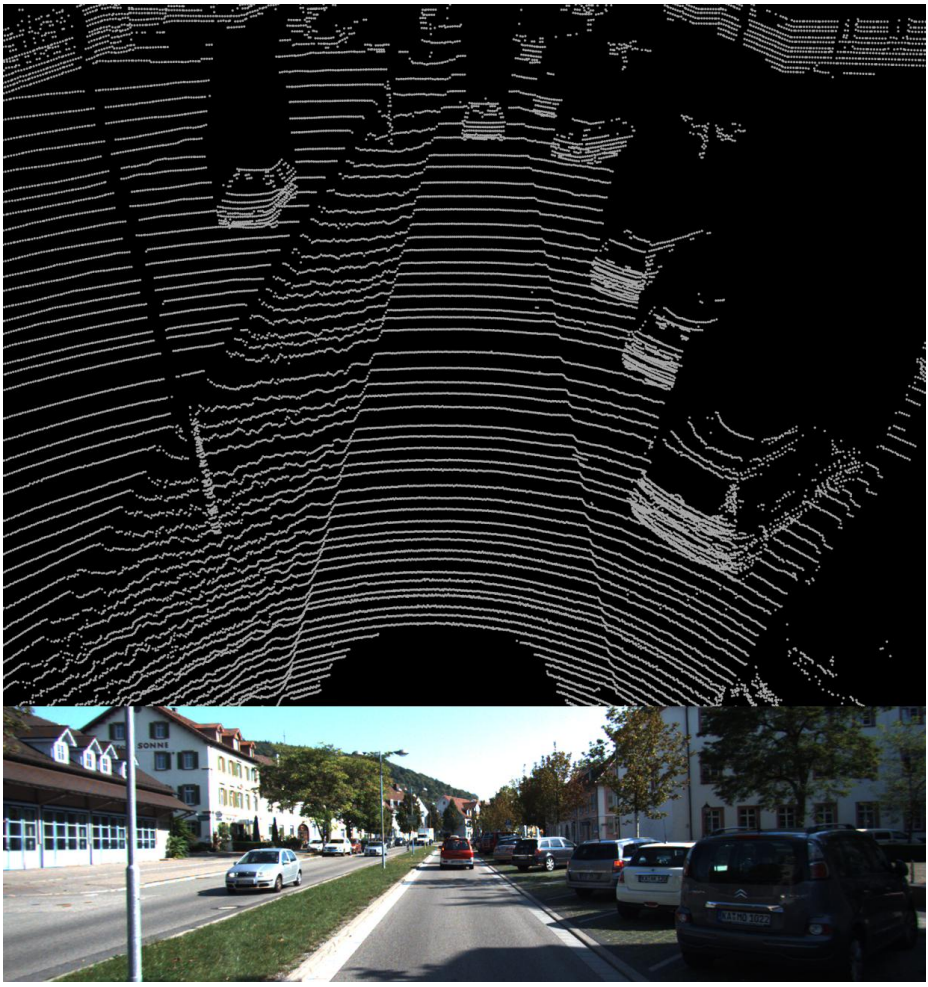

Figure 2.1: *Visualization of a LiDAR point cloud captured by the Velodyne HDL-64E, together with an image of the corresponding scene. Both the point cloud and the image is part of the KITTI dataset [15]. The point cloud visualization was created using Open3D [64].*

and segmentation is found in Figure 2.2.

In the PointNet classification network, a shared multi-layer perceptron (MLP) with batch normalization and ReLU activation function is applied to each input point $p_i = (x_i, y_i, z_i, r_i)$, to obtain a feature vector $f_i \in \mathbb{R}^{1024}$. Max pooling is then applied to the point-wise feature vectors $f_1, \ldots, f_n$ to obtain a global feature vector $f \in \mathbb{R}^{1024}$. This feature vector f is then finally fed through a small fully-connected network to output classification scores for k predefined object classes. Because max pooling is a symmetric function, it aggregates the point-wise local information in a way that makes the model invariant to the input point order.

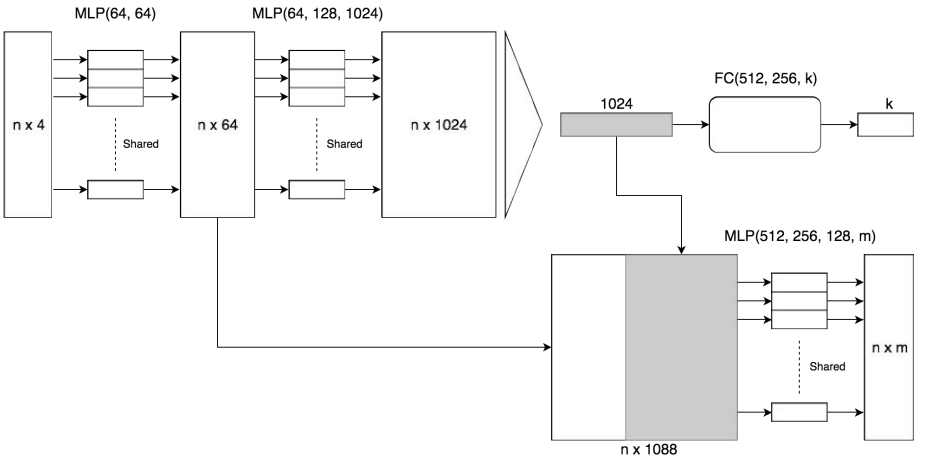

Figure 2.2: *A schematic overview of the basic PointNet architecture. The PointNet classification network takes n points as input and outputs classification scores for k predefined classes. The segmentation network is an extension to the classification network and outputs point-wise classification scores for m classes.*

The PointNet segmentation network is an extension to the classification version. In it, the global feature vector $f \in \mathbb{R}^{1024}$ is repeatedly concatenated with intermediate point-wise feature vectors $\tilde{f}_i \in \mathbb{R}^{64}$, to obtain point-wise feature vectors $g_i \in \mathbb{R}^{1088}$ containing both local and global information. Another shared MLP is then applied to each g_i to finally output point-wise classification scores for m predefined classes.

The authors prove that the PointNet network can approximate any continuous function operating on a set, and report that it according to visualization learns to summarize a point cloud by a sparse set of key points which roughly corresponds to the skeleton of objects. Quantitatively, the authors report state-of-the-art (SOTA) or close-to SOTA performance on datasets for both object classification and object part segmentation, while comparing favourably in terms of both model size (number of parameters) and computational complexity.

An extension of the PointNet architecture named PointNet++ was later presented by Qi *et al.* in [49]. In this approach, a hierarchical architecture is designed by repeatedly applying PointNet to a nested partitioning of the input point cloud. The set of points is first grouped into overlapping local regions according to a distance metric. Region-wise features are then extracted by a mini-PointNet shared between the local regions, capturing fine geometric structures. These region-wise features are then further grouped into larger regions, from which higher level features are extracted by another shared mini-PointNet. For the PointNet++ classification network, this process is repeated until a global feature vector representing the entire point cloud is obtained. Finally, this global feature vector is fed through fully-connected layers to output classification scores.

In the PointNet++ segmentation network, the intermediate region-wise features are instead upsampled by using interpolation and unit PointNets, which correspond to 1×1 convolutions in CNNs. This results in an architecture similar to that of the encoder-decoder architecture used for semantic segmentation of images [2].

The authors claim that the main improvement of PointNet++ is its ability to learn local features with increasing contextual scales. Since PointNet learns point-wise features which are aggregated to a global representation, the network by design does not capture local geometric structure. PointNet++ is specifically designed to mitigate this problem. The authors also report quite significant performance gains for both the classification and segmentation task, which results in new SOTA. The performance gains does however also come with increased computational complexity, as the PointNet++ inference time is more than three times that of PointNet.

A generalization of PointNet was recently presented by Wang *et al.* in [60]. The authors' main contribution is a novel operation named EdgeConv, which is designed to better capture local geometric structure among the points. By incorporating the EdgeConv module into the basic PointNet architectures, they obtain classification and segmentation networks with quite significantly improved performance.

With the EdgeConv module, the authors claim to address a key flaw of both PointNet and PointNet++: points are independently processed, neglecting the local geometric relationship among points. Instead of generating point-wise features solely from each point's previous embedding, EdgeConv utilizes edge features which capture the relationship between a point and its neighbors.

Specifically, a directed graph is created of the points, where each point is connected by a leaving edge to each of its k closest neighbor points. For each edge in the graph, an edge feature $h(p_i, p_j)$ is then computed, where p_i and p_j are the two points connected by the edge, and h is some parametric function. To apply the EdgeConv operation on a specific point p_i, an aggregation function (*e.g.* sum or max) is applied to all edge features $h(p_i, p_{j_1}), \ldots, h(p_i, p_{j_k})$ corresponding to that point, where $p_{j_1}, \ldots, p_{j_k}$ are the k closest neighbor points of p_i. These neighbors are not fixed but will change between network layers, since the directed graph is dynamically updated after each layer in the network. In later layers the k-nearest neighbor grouping can thus for instance correspond to a grouping of semantically similar points.

With an appropriate choice of $h(p_i, p_j)$ and the aggregation function, EdgeConv is a direct generalization of the standard convolution operation on images, with the neighbor points corresponding to the pixels surrounding the center pixel in an image patch. With $h(p_i, p_j) = h(p_i)$ the original PointNet architecture is also obtained, which thus can be considered a specific instance of the presented architecture.

Compared to PointNet++, the authors report improved performance for classification and identical performance for object part segmentation. In terms of computational complexity, the presented classification network is in inference almost four times slower than PointNet but nearly twice as fast as PointNet++.

2.3 2D Object Detection

A 2D object detection model takes an image as input, and should output a 2D bounding box together with a class label for all objects of interest in the image. The 2D bounding box is an axis-aligned rectangle, which ideally should be of minimum size while still containing all parts of the associated object in the image.

A 2D bounding box is parameterized as $(u_{min}, u_{max}, v_{min}, v_{max})$, where (u_{min}, v_{min}) are the pixel coordinates of the top-left bounding box corner, and (u_{max}, v_{max}) are the pixel coordinates of the bottom-right corner. The ground truth 2D bounding boxes for an example image in the KITTI dataset [15] are visualized in Figure 2.3, in which red bounding boxes correspond to the car object class.

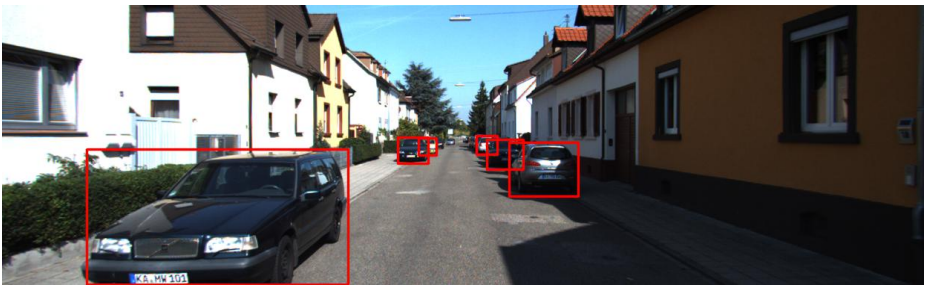

Figure 2.3: *Visualization of the ground truth 2D bounding boxes for an example image in the KITTI dataset [15]. Red bounding boxes correspond to the car object class.*

The modern 2D object detector was introduced by Girshick *et al.* in [17], where the authors presented the R-CNN architecture. R-CNN utilizes the selective search method presented by Uijlings *et al.* in [58] to extract object region proposals, *i.e.* candidate 2D bounding boxes. These region proposals should contain all objects of interest while filtering out the majority of background regions. The candidate regions are then fed to a detection stage where they are classified as either background or a specific object class. To do so, R-CNN independently feeds each candidate image region to an AlexNet CNN [29] to extract a feature vector, which is then fed to class-specific support vector machines (SVMs) to output class scores. Finally, non-maximum suppression (NMS) is used to filter redundant predicted bounding boxes in the image. In NMS, the bounding boxes are iterated in decreasing order of class score. For each bounding box, all lower-scoring bounding boxes with an intersection-over-union (IoU), also known as *Jaccard index*, greater than some threshold are then removed.

While using a CNN to independently extract features from each region proposal significantly improved detection performance compared to previous methods, it is also computationally inefficient. R-CNN was thus improved by Girshick in [16], where the Fast R-CNN architecture was introduced. Fast R-CNN utilizes the same selective search method to extract object region proposals, which are then fed to the detection stage together with the original image. The image is

processed by a CNN, extracting a feature map representing the entire image. Each region proposal is then projected onto this feature map, pooled into a fixed size and mapped to a region feature vector. This feature vector is finally fed to two fully-connected layers to output predicted class scores and regress a relative translation and size offset with respect to the proposal bounding box. Only one CNN forward pass per image is thus required, which significantly improves both training and inference time, while achieving comparable or improved detection performance. Compared to R-CNN, Fast R-CNN is also a less complex architecture. The Fast R-CNN detection stage consists of a single network, which also can be trained in a single stage using a multi-task loss.

Fast R-CNN does however still require the region proposals to be extracted by some external method, which becomes a computational bottleneck and prohibits full end-to-end training. This problem is addressed by Ren *et al.* in [51] by introducing the Faster R-CNN architecture. Faster R-CNN is a unified network utilizing the Fast R-CNN detection stage in combination with a novel region proposal network (RPN). In Faster R-CNN, each image is processed by a CNN to extract a global feature map, which is fed as input to both the RPN and the Fast R-CNN detection stage. The RPN is a fully convolutional network and outputs an array of shape $W \times H \times (4+2)k$. This output corresponds to 4 bounding box residuals and 2 object confidence scores for k anchor boxes, which are centered at an even grid of size $W \times H$ in the image. The k anchors are reference bounding boxes of different sizes and aspect ratios, which are chosen *a priori* such that a majority of all possible bounding boxes should have a close match in the anchor box set. For each of the k anchor boxes centered at each grid location, the RPN thus outputs a relative box translation and size offset together with two confidence scores, where the scores describe whether or not the translated and resized anchor box is likely to contain an object. The RPN output is thus WHk bounding boxes in the image, each with a corresponding object confidence score. The highest scoring bounding boxes are chosen as region proposals and fed to the Fast R-CNN detection network. Using the previously extracted image feature map, the region proposals are then classified and further refined by the network, as described in the previous paragraph. Faster R-CNN and recent extensions to this architecture, *e.g.* presented by Lin *et al.* in [36] and by He *et al.* in [21], are employed in a majority of current SOTA 2D detectors, for instance as measured by their performance on the COCO detection dataset [35].

All architectures described in this section are instances of what is normally referred to as two-stage detectors, in which a first stage generates region proposals which are then classified and refined in a second stage. Another approach is that of the single-stage detectors, as exemplified by the SSD architecture which was introduced by Liu *et al.* in [39].

In contrast to a typical two-stage detector, SSD directly outputs class scores and anchor offsets, and it does so by employing a technique similar to that of RPN. On a feature map of spatial size $W \times H$, convolutional filters are applied to output an array of size $W \times H \times (4+C)k$ where C is the number of object classes, including a background class. The output is thus WHk bounding boxes in the image, each with a corresponding score for all object classes. Such convolutional

filters are in SSD applied to several feature maps of different spatial sizes, allowing bounding box prediction at multiple scales. In a final step, NMS is applied to filter redundant bounding boxes.

Single-stage detectors were mainly designed with improved inference time in mind and can generally not compete with SOTA two-stage detectors in terms of detection accuracy. A single-stage detector designed for further improved computational efficiency was presented by Wu *et al.* in [61]. The presented architecture, named SqueezeDet, operates on a single scale image feature map and uses the light-weight SqueezeNet [23] CNN as its feature extractor. The improvement in terms of computational cost does however come at the expense of further decreased detection performance.

An improvement to single-shot detectors was however recently presented by Lin *et al.* in [37]. The authors find that the object-background class imbalance, *i.e.* the fact that only a small subset of anchor boxes are actually covering an object of interest in a standard input image, is the main cause for single-stage detectors' trailing detection accuracy. To address this problem, the authors introduce a novel loss term named the focal loss. The focal loss is a rescaled version of the standard cross entropy classification loss, reducing the loss for examples where the predicted probability for the ground truth object class is large. This decreases the loss contribution from the large number of easily classified background bounding boxes, and instead focuses training on difficult and currently misclassified examples. By applying the focal loss to a single-stage detector based on the feature pyramid network architecture in [36], the authors report SOTA performance on COCO [35] while comparing favourably in terms of inference time.

2.4 3D Object Detection

A 3D object detection model takes various sensor data as input, and should output a 3D bounding box together with a class label for all objects of interest in the sensors' field of view. A 3D bounding box is an oriented rectangular cuboid placed in 3D space, which ideally should be of minimum size while still containing all parts of the associated object.

In automotive applications, a 3D bounding box is commonly parameterized as $(x, y, z, h, w, l, \theta)$. (x, y, z) is the 3D coordinates of the bounding box center, (h, w, l) is respectively the height, width and length of the box, and θ is the yaw angle of the bounding box. The pitch and roll angles are assumed to be zero, or to be of negligible importance for the application. The ground truth 3D bounding boxes for the car object class in an example from the KITTI dataset [15] were shown in Figure 1.1, where the boxes are visualized both in the LiDAR point cloud and in the associated image.

The input sensor data to the model could potentially come from a combination of various sensors, such as *e.g.* monocular cameras, stereo cameras, sonars, radars and LiDARs. The literature is however almost entirely dominated by approaches trained and evaluated on the KITTI dataset [15], which was collected using a vehicle equipped with a forward-facing stereo camera rig and a Velodyne

HDL-64E LiDAR. The stereo cameras are not commonly utilized in related work on KITTI, most approaches instead take as input either only images from one of the cameras or only point clouds from the LiDAR, or fuses information from both modalities. Stereo cameras have however been extensively used by Daimler, see *e.g.* the work by Barrois and Wöhler in [3].

Early work related to 3D object detection on KITTI is that of Chen *et al.* in [8]. The presented method utilizes stereo imagery to generate object proposals in the form of 3D bounding boxes. These 3D proposals are then projected onto the image and scored to obtain region proposals, which are used in an extended Fast R-CNN detector to both perform 2D detection and estimate the object's yaw angle. The authors report SOTA performance for 2D detection and orientation estimation, but do not provide a quantitative evaluation of the proposal 3D bounding boxes.

Chen *et al.* extended this approach to obtain a monocular version in [9]. 3D proposals are generated by exhaustively placing 3D bounding boxes with typical sizes near an assumed orthogonal ground-plane, projecting these onto the image and scoring each region by utilizing semantic segmentation, instance segmentation, object shape and location priors. The top-scoring regions are then fed to the extended Fast R-CNN network from [8] to perform 2D detection and orientation estimation.

Another approach was later presented by Engelcke *et al.* in [12], where a LiDAR-only model is introduced. The LiDAR point cloud is discretized into a sparse 3D grid: for each grid cell that contains a non-zero number of points, a hand-crafted feature vector is extracted based on the statistics of the points in that cell. The discretized point cloud can then be quite efficiently processed by applying sparse 3D convolutions to this sparse grid, meaning that the filters are only applied to the non-empty grid cells. To detect objects, sliding-window search is performed with a fixed-size window of N different orientations. Each window is processed by a CNN performing binary classification, predicting whether or not the window contains an object. The model is evaluated by projecting the detected 3D bounding boxes onto the image plane and evaluating its 2D detection performance. The authors report SOTA results on KITTI among the LiDAR-only methods.

A similar architecture was presented by Li in [32] for the task of vehicle detection. The LiDAR point cloud is discretized into a 3D grid and then processed using 3D convolutions in a fully convolutional network. The network is essentially a 3D RPN and outputs object confidence scores together with 3D bounding box residuals. The model is evaluated by the projected bounding boxes' 2D detection performance and the author reports SOTA results for LiDAR-only methods, also outperforming the method by Engelcke *et al.* [12].

Chen *et al.* then introduced an architecture utilizing both monocular image and LiDAR information in [10]. The LiDAR point cloud is projected onto both a 2D top-view and a 2D front-view, from which feature maps are extracted using separate CNNs. In the feature extraction stage, a feature map is also extracted from the monocular image. The LiDAR top-view feature map is passed to an RPN to output proposal 3D bounding boxes. Each of these 3D proposals is pro-

jected onto the feature maps of all three views, and a fixed-size feature vector is extracted for each view by using pooling. The three feature vectors are then fused in a region-based fusion network, which finally outputs class scores and regresses 3D bounding box residuals. The feature vectors are fused by combining the three vectors by element-wise mean, feeding the combined vector through three separate fully-connected layers, and then once again combining the resulting vectors by element-wise mean. The authors evaluate the predicted 3D bounding boxes for the car class by the average precision metric (AP_{3D}). They follow [8] and split the KITTI dataset into a specific training and validation set, and report their obtained AP_{3D} score on the validation set. They report a significant performance improvement compared to previous methods for 3D detection, and also obtain close-to SOTA 2D detection performance by projecting the 3D bounding boxes onto the image plane.

A much simplified monocular image-only architecture was also presented by Mousavian *et al.* in [43]. In this approach a SOTA 2D object detector is used to generate 2D region proposals, from which corresponding 3D bounding boxes are estimated. Each image region proposal exceeding a certain confidence threshold is fed to a CNN, which outputs estimates for the associated 3D bounding box's dimensions (h, w, l) and heading angle θ. A novel classification-regression hybrid is utilized for the heading estimate, in which the angle is classified into discrete bins and residuals for each angle bin center is regressed. Given the estimated dimensions and heading angle, and utilizing the constraint that a projected 3D bounding box should fit tightly into the corresponding 2D bounding box, the center coordinates (x, y, z) and thus the complete 3D bounding box can then be estimated using an optimization-based method. The authors report somewhat improved 2D detection and orientation estimation performance compared to the significantly more complex and less general architecture by Chen *et al.* in [9].

Chabot *et al.* then presented the Deep MANTA architecture in [7], which is a monocular image-only architecture for joint 2D and 3D detection of vehicles. Deep MANTA consists of two main stages. The first stage is a CNN outputting 2D bounding boxes together with estimated vehicle parts pixel coordinates and 3D bounding box dimensions. This output is then fed to a second stage in which a 3D vehicle dataset is utilized to estimate the vehicle orientation and 3D location. The 3D vehicle dataset consists of M 3D models of different vehicle types, together with their associated 3D bounding box dimensions (h, w, l) and vehicle parts 3D coordinates. In the second stage, the estimated 3D bounding box dimensions are compared to the entries in the dataset to find the best-matching 3D vehicle model. The model's associated vehicle parts 3D coordinates are then matched to the estimated vehicle parts pixel coordinates, and using a pose estimation algorithm [31] an estimate of the full 3D bounding box is obtained. The authors report SOTA performance for 2D detection and orientation estimation on KITTI. For 3D localization accuracy, in which a 3D bounding box is regarded correct if the distance from its center to the ground truth bounding box center is less than 1 meter, the authors report improved performance compared to the monocular model by Chen *et al.* [9].

A much improved LiDAR-only architecture named VoxelNet was later pre-

sented by Zhou and Tuzel in [65]. The LiDAR point cloud is divided into equally spaced 3D voxels, and the points are grouped according to the voxel they reside in. The points in each non-empty voxel are then fed through a number of Voxel Feature Encoding (VFE) layers, which output a voxel-wise feature vector of fixed size. The VFE layer is essentially a small PointNet [48]. The output of this stage is thus a sparse 4D array, corresponding to a 3D grid in which only some voxels have an associated learned feature vector. The 4D array is fed through a number of 3D convolutional layers and then reshaped, resulting in a 3D feature map. This feature map is fed as input to an RPN, which outputs object confidence scores and 3D anchor box residuals. Separate networks are trained for detection of vehicles, pedestrians and cyclists, and the predicted 3D bounding boxes are evaluated by the AP_{3D} score. The authors report clear SOTA results for LiDAR-only methods, and even outperform the LiDAR-image fusion architecture by Chen *et al.* [10]. They also compare with a baseline architecture in which the point cloud instead is directly projected to a 2D top-view and report quite significant performance gains, especially for the pedestrian and cyclist classes.

Qi *et al.* presented in [47] the Frustum-PointNet architecture, which consists of three main stages: 3D frustum proposal, 3D instance segmentation and 3D bounding box estimation. Similarly to Mousavian *et al.* [43], a SOTA 2D object detector is first used to generate 2D region proposals. In the frustum proposal stage, each 2D region proposal is extruded to extract the corresponding 3D frustum proposal, containing all points in the LiDAR point cloud which lie inside the 2D region when projected onto the image plane. This frustum proposal point cloud is then fed to the instance segmentation stage, in which a PointNet [48] segmentation network performs binary classification of each point, predicting whether or not the point belongs to the detected object. All positively classified points are then finally fed to the bounding box estimation stage, in which another PointNet is used to estimate the 3D bounding box parameters. For the box center estimate, the network regresses residuals relative to the segmented point cloud centroid. For the bounding box dimensions and heading angle, a classification-regression hybrid inspired by Mousavian *et al.* [43] is utilized. The authors report SOTA 3D detection performance (as measured by AP_{3D} score) on KITTI for vehicles, pedestrians and cyclists, quite significantly outperforming VoxelNet [65] for all three object categories.

A quite similar approach, also utilizing the PointNet architecture [48], was independently presented by Xu *et al.* in [63]. In this work, the image-and-LiDAR architecture PointFusion was introduced. Just as in Frustum-PointNet [47], a SOTA 2D object detector is used to extract 2D region proposals which are extruded to the corresponding frustum point cloud. Each frustum is fed to a PointNet, extracting both point-wise feature vectors and a global LiDAR feature vector. Each 2D image region is also fed to a CNN that extracts an image feature vector. For each point in the frustum, its point-wise feature vector is concatenated with both the global LiDAR feature vector and the image feature vector. This concatenated vector is finally fed to a shared MLP, outputting 8×3 values for each point. The output corresponds to predicted (x, y, z) offsets relative the point for each of the eight 3D bounding box corners. The points in the frustum are thus used as dense

spatial anchors. The MLP also outputs a confidence score for each point, and in inference the bounding box corresponding to the highest-scoring point is chosen as the final prediction. The authors report AP_{3D} score for all three object categories on KITTI, but are quite significantly outperformed across the board by Frustum-PointNet. Their reported performance is comparable to that of Chen *et al.* in [10].

Finally, another fusion architecture named AVOD was introduced by Ku *et al.* in [30]. The LiDAR point cloud is projected onto a 2D top-view, from which a feature map is extracted by a CNN. A second CNN is used to extract a feature map also from the input monocular image. The two feature maps are shared by two sub-networks: an RPN and a second stage detection network. The architecture is thus similar to that of Chen *et al.* in [10], the key difference being that AVOD uses both image and LiDAR features also in the RPN. The reported 3D detection performance is a slight improvement compared to Chen *et al.* [10] and is comparable to that of VoxelNet [65] for cars, but somewhat lower for pedestrians and cyclists. The authors also find that utilizing both image and LiDAR features in the RPN, as compared to only using LiDAR features, has virtually no effect on the performance for cars, but a significant positive effect for pedestrians and cyclists.

2.5 Generative Adversarial Networks

In 2014 Ian Goodfellow *et al.* [18] presented a new method for estimating generative models via an adversarial process. It involves a generator network G and discriminator network D pitted against each other in a minimax game. The generator network is optimized to fool the discriminator network into predicting a high probability of its generated samples coming from the data distribution, and thereby training the generator to model the data distribution by generating data closely resembling data being drawn from the true distribution. The discriminator on the other hand is optimized to discriminate between samples from the model distribution and the data distribution. In their paper Goodfellow *et al.* use the analogy that the generator could be thought of as a counterfeiter, trying to produce fake money indistinguishable from real money. The discriminator could then be thought of as the police, trying to identify fake generated money from real money. As both networks are being optimized during this game they are driven to perform better given their task until the generated money will be indistinguishable from the real money.

In order to learn the generator's distribution p_g over the data x they define a prior $p_z(z)$ on input noise, and define a mapping $G(z;\theta_g)$, where G is a differentiable function represented as a multilayer perceptron with parameters θ_g. They also define a second multilayer perceptron $D(x;\theta_d)$, which maps the input vector x to a scalar output. The output scalar $D(x)$ represents the probability that the sample x came from the data distribution rather than p_g. D is then trained to maximize this probability for data coming from the data distribution and minimize the probability for samples coming from the model distribution, *i.e.* generated by G. The generator G is simultaneously optimized to fool the discriminator into

predicting a high probability of its generated samples coming from the true distribution, and thereby to minimize $log(1 - D(G(z)))$. This results in the following minimax game between D and G with objective function $V(D, G)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[log(D(x))] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \quad (2.1)$$

Radford *et al.* [50] built on top of the work of Goodfellow *et al.* and proposed to use convolutional neural network (CNN) representations for the generator and discriminator instead of multilayer perceptrons. CNNs had seen a huge adoption in supervised computer vision problems, but had not been applied as much to unsupervised learning tasks. By optimizing convolutional neural networks during the adversarial training process they showed that convolutional neural network layers in both the generator and discriminator learned low and high level feature representations of images from the data distribution. The authors then showed that this low dimensional feature representation was suitable for tasks such as classification problems. They named their architecture Deep Convolutional Generative Adversarial Network (DCGAN), and this representation of the generator and discriminator has been commonly adapted as generator and discriminator representations in later works on GANs.

2.5.1 Training Generative Adversarial Networks

Training generative adversarial networks involves finding the Nash equilibrium to a two-player game where the generator and discriminator competes against each other. Each of these networks are trained to minimize their respective cost functions. For the discriminator this means minimizing $J_d(\theta_d, \theta_d)$, and for the generator minimizing $J_g(\theta_d, \theta_d)$. The Nash equilibrium is a point such that J_d is minimized with respect to θ_d and where J_g is minimized with respect to θ_g. This occurs when neither the generator nor the discriminator has anything to gain by updating its weights with respect to the loss, determined by the opponent's configuration. The fact that Nash equilibrium occurs when the loss function of the discriminator and the generator are minimized with respect to their parameters seem to intuitively motivate the use of gradient descent techniques to find this point. However, as the loss functions are non-convex, the parameters are continuous and the parameter space is high-dimensional these algorithms have a high risk of not converging. As the networks are trained sequentially, updating the parameters of the discriminator θ_d to reduce J_d might increase J_g, and in turn updating θ_g might increase J_d. Because of this gradient descent can enter a stable orbit, instead of converging to the optimum [54]. Because of this, a lot of research has been done on finding methods to improve the stability of training GANs.

Wasserstein GAN, presented by Arjovsky *et al.* [1] has been shown to stabilize training by introducing a new objective function which aims at minimizing the Earth Mover (EM) distance, also called the Wasserstein distance. In general the goal of training a generative adversarial network is to model a probability distribution by training a generator and discriminator pair where the generator

converges to produce samples indistinguishable from samples drawn from the data distribution. This involves minimizing the distance between the model distribution p_g and the data distribution p_d. In their paper Arjovsky *et al.* show why minimizing the Wasserstein distance metric between p_g and p_d has several benefits compared to other common distance metrics such as Kullback-Leibler (KL) distance and Jensen-Shannon (JS) divergence. They show that out of EM, KL and JS the EM distance is the only distance metric with guarantees of continuity and differentiability, which are both coveted characteristics in a loss function. They also show that the discriminator loss correlates with sample quality, and in their method the discriminator is updated multiple times for every update to the generator [1].

2.5.2 Conditional Generative Adversarial Networks

In an unconditioned generative adversarial network, there is no control on modes of which the data is being generated. By conditioning the generator and discriminator on class labels, Mirza *et al.* [42] showed that it is possible to direct the data generating process and model multi-modal data distributions using the adversarial process presented by Goodfellow *et al.* The information on which the generator and discriminator are conditioned could be any kind of auxiliary information, such as class labels, an image, or text relevant to the data distribution which the generator learns to model. In their paper Mirza *et al.* conditions the generator and the discriminator on the auxiliary information y by feeding y to both the generator and the discriminator as an additional input layer. In the generator the prior $p_z(z)$ and y are combined as inputs, and in the discriminator x and y are combined as inputs to the discriminative function. This results in the following minimax game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[log(D(x|y))] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z|y)))] \quad (2.2)$$

This is a key contribution to generative adversarial networks for applications such as domain adaptation, where the generator is presented with images from one domain A and is tasked with translating these images to another domain B, where the translated images should be indistinguishable from images being sampled from domain B. One paper that builds on the idea of feeding auxiliary information to the generator is the paper on auxiliary classifier GANs by Odena *et al.* [45].

The authors of this paper demonstrated that by adding an auxiliary objective to the discriminator and let it not only optimize for discriminating between distributions, but also to classify the images it receives, the performance of the generated images is increased as the generator has to consider to not only produce images that look like they are sampled from the data distribution, but also generate images of the specific class that it is conditioned on. This is also work that many domain adaptation solutions using GANs build upon, as it is often a requirement that the images that are being translated from the source domain

to the target domain retain the semantics of the original image so that the annotation of the original image are still valid in its translated form. Similarly to the method by Odena *et al.* the generator needs to consider the semantics of it's input condition when generating a corresponding image.

2.6 Domain Adaptation

Domain adaptation is critical for success in supervised learning tasks on images from new unseen environments. It is often very costly to extract large diverse datasets with annotations, and lately multiple methods of performing unsupervised domain adaptation using generative adversarial networks have been presented as solutions to this problem. Domain adaptation is the process of narrowing the domain shift that occurs between images from different domains for the purpose of improving the performance of a task network with respect to the target domain. By optimizing the generator network to fool the discriminator network into predicting a high probability of its generated samples originating from the target distribution, and training the discriminator to discriminate between images from the model distribution and from the target distribution it's possible to train both networks sequentially and finding a translation from the source domain to the target domain in the mapping that the generator performs. Recent work has been presented where this translation is performed both on the image representation and on the feature representation.

2.6.1 Adaptation of Image Representation

In their paper Isola *et al.* [26] present a method that uses conditional adversarial networks as a way of performing image-to-image translations. They demonstrate how their method is able to synthesize images from label maps, reconstruct images from edge maps and colorize images, among other tasks. The authors use a generator architecture originally presented by Ronneberger et al. in their paper *U-Net: Convolutional Networks for Biomedical Image Segmentation* [53]. Many previous solutions to image-to-image translation problems use an encoder-decoder representation of the generator, where the generator consists of an encoder that extracts a low dimensional feature representation of the image, and a decoder that generates an image from the feature representation extracted by the encoder.

This architecture provides an embedding of the input image which captures high level features in the input image. The issue with this architecture is that as the decoder upsamples the embedding to reproduce an image of the original dimensions many of the low level features in the original input image are lost. To preserve these low level features the authors use a "U-Net" generator, which closely resembles the encoder-decoder architecture previously used for image-to-image problems. The novelty with this generator architecture is that it adds skip-connections between the layers in the encoder and the decoder. These connections distributes low level features captured during the downsampling phase with features in the upsampling phase, and generates images with more low level details.

The authors also make use of a PatchGAN discriminator proposed by Li *et al.* [33]. Instead of producing one scalar value indicating the validity of the whole image, the PatchGAN discriminator determines the validity of each patch in the image. The final layer in the discriminator consists of filters that are convolutionally propagated across each patch in the image and produces a final output of the dimension NxN, which represent the probability of each of the image patches coming from the data distribution. This forces the generator to produce images that look realistic on a lower detail level. In addition to the adversarial loss defined as $\mathcal{L}_{GAN} = \mathbb{E}_{x\sim p_{data}(x)}[log(D(x))] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))]$, the generator is also trained to minimize an $\mathcal{L}_1$ loss between the output of the generator and the original image, to further restrict the generator from producing images dissimilar to the original images. The total objective function of their solution is:

$$\min_G \max_D V(D, G) = \mathcal{L}_{GAN} + \lambda \mathop{\mathbb{E}}_{x,y,z}[\|y - G(x, z)\|_1] \tag{2.3}$$

A subsequent paper was published by Zhu *et al.* [66] on how to perform image-to-image translations between two domains without needing paired images from both domains. They called their method CycleGAN and their model topology consists of two generator and discriminator pairs. One generator G_{ST} translates images from the source domain to the target domain, and the other generator G_{TS} translates images from the target domain to the source domain. The discriminators are tasked with determining whether their input images originate from their respective domain or if they are translated image versions from the opposite domain. One of the major challenges with generative adversarial networks applied to domain adaptation problems is that for traditional GANs there is nothing that hinders the generators from making structural changes to their input images when translating them to the opposite domain. If the translated images are to be used for object detection or semantic segmentation tasks such changes could make the annotations of the original image invalid for the translated version of the image.

To further restrict the generators from making these changes the authors propose a cycle-consistency loss, meaning that images that are translated to the opposite domain and are then translated back to their original domain shall be identical to the original image, thus reducing the mapping from one domain to the other. To enforce this the authors apply a $\mathcal{L}_1$ loss between the reconstructed images and the original image. This will regularize the generator from making such changes to their input image that the original image can not be obtained by translating the image back to its original domain. To help stabilize training they apply a least-squares adversarial loss, which Mao et. al [40] presented as an alternative to the negative log-likelihood loss most commonly used by previous GAN varieties. This means that the generators are trained to minimize $\mathbb{E}_{x\sim p_{data}(x)}[(D(G(x)) - 1)^2]$ and the discriminators are trained to minimize $\mathbb{E}_{y\sim p_{data}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x\sim p_{data}(x)}[D(G(x))^2]$. To further help stabilize the model during training they adapt the strategy presented by Shrivastava *et al.* [57], by updating the discriminator networks based of a buffer of previously generated images instead of the latest generated images. In their experiments they use a

buffer of 50 of the previously generated images, and this buffer is updated after each iteration. Similarly to Isola *et al.* [26] they use PatchGAN discriminators that determine the probability of whether each patch in the image are from the target domain instead of estimating one probability for the image.

2.6.2 Adaptation of Feature Representation

One technique for learning a joint feature representation between multiple domains was presented by Liu *et al.* [38]. In their approach they train two generator-discriminator pairs, where the first layers in the generator networks are shared between the two generators. The same thing also applies to the discriminators, where the first layers are shared between them. Both generators take Gaussian noise z as input, and generator G_A is optimized to generate images that fool the discriminator D_A into predicting a high probability of the generated images coming from distribution A, and vice versa for G_B and D_B. Since the latent input to the generators is fed through layers that are shared between the two generators, both being optimized to generate images resembling their respective domains, the weights of these layers will be tuned to provide a domain agnostic representation of the images in both domain A and B. The same applies in the discriminators where D_A and D_B share the weights in the first layers and these are tuned to provide a domain agnostic image representation as the discriminators are optimized for their discriminative purposes.

Other works have been presented where this domain agnostic feature representation is further adapted for providing information suitable for a specific task, such as classification, object detection or semantic segmentation. In their work Sankaranarayanan *et al.* [55] present a method for extracting a domain agnostic feature representation from images in both the source domain and target domain by feeding images from both domains through an encoder network that learns an image embedding that is both domain invariant and suitable for semantic segmentation. This representation is further fed into a generator that is tasked with reconstructing this the original image from this feature representation. This image is further fed to the discriminator, where the generator wants to fool the discriminator into predicting a high probability of the image being sampled from the original domain. The encoder network in this topology is where the domain invariance is captured, as this network is optimized for task performance and for domain invariance, as the adversarial objective of the encoder is to have the generated images based of its feature representation fool the discriminator into labeling these images as coming from the opposite domain.

Similarly to Odena *et al.* the authors also introduce an auxiliary objective for the discriminator, where the discriminator is also optimized for performing semantic segmentation on the image. This will enforce that the encoder and generator preserves the semantic information present in the input image so that annotations are still valid in the translated version of the image. The feature representation that the encoder provides is also fed to a task network that performs semantic segmentation based of this representation. The encoder is then optimized to provide a representation that maximizes the performance of the task

network. After having optimized the encoder, the generator and the discriminator, the encoder will have learned an embedding for both domains that is domain agnostic and that is suitable for the task.

2.6.3 Simulatenous Domain Adaptation of Feature and Image Representation

As a method for generating feature representations and image representations that are both domain invariant Hoffman *et al.* [22] introduced a method they called Cycle-Consistent Adversarial Domain Adaptation (CyCADA). Their method is influenced by several of the papers previously described in section 2.7.1 and 2.7.2. In their framework the generators G_{ST} and G_{TS} are optimized to minimize a cycle-consistency loss, as introduced by Zhu *et al.* [66]. Cycle-consistency enforces that images that are translated by G_{ST} from the source domain to the target domain and then translated back to the source domain by G_{TS} should be identical to the original image. This will restrict the generators from making large changes to the original image during translation and preserve the semantic information present in the original image. However, it's common for generators in CycleGAN frameworks to *e.g.* add a tree line to where there was sky in the original image.

For tasks such as semantic segmentation and object detection these kinds of changes might make the annotations for the original image representation invalid in its translated form. To further restrict the generators from making these changes the authors argue that the predicted labels for the translated form of the image should be identical to the predicted labels for the original image. For a semantic segmentation task this involves adding a cross-entropy loss between the semantic segmentation prediction of the original image and the prediction of the translated image. They call this loss *semantic consistency loss*, and it is closely related to the content loss that is described by Gatys *et al.* in their paper *Image style transfer using convolutional neural networks* [13].

Images translated from the source domain to the target domain are also fed to a discriminator network D_T that tries to discriminate between images from the target domain and translated images from the source domain. The generator G_{ST} is optimized to fool the discriminator into predicting a high probability of the image being sampled from the target distribution. The translated image is further fed into an encoder f_T used for extracting a feature representation of images from the target domain, and this feature representation is fed to a second discriminator D_{feat} with the task of discriminating between whether feature representations are from images in the target domain or not. During the process the framework will optimize the generator G_{ST} to produce translated images that closely resemble images from the target domain, and also images with features that closely resemble features from images sampled from the target domains. This enforces domain invariance at both the image representation and the feature representation. Similarly to Sankaranarayanan *et al.* [55] the authors also apply a task network which performs semantic segmentation based of the features extracted by f_T. This will further force the generator to keep the semantic boundaries in

the original image during translation.

3
Methods

In this chapter we present a detailed description of the methods for domain adaptation and 3D object detection which have been implemented in the thesis. We describe the implemented model architectures, the utilized datasets and the process of training the models.

The chapter begins with a description of domain adaptation methods in Section 3.1-3.4, continues with all implemented 3D object detection methods in Section 3.5-3.8 and ends with a description of combined methods in Section 3.9.

3.1 Domain Adaptation - Datasets

We have evaluated our domain adaptation solution on three pairs of datasets. We performed unsupervised domain adaptation between the synthetic GTA 5 dataset and the real-world Cityscapes dataset to evaluate the level of quality of generated images by the CycleGAN model. To further validate that adding a task network to the image translation process helps the generator to preserve semantic information during translation that is important for the task, we then implemented an additional classification network and added it to the generator and discriminator architecture. This new model was evaluated on domain adaptation for the purpose of performing classification on translated images from the MNIST dataset made to resemble the modified MNIST images in the MNIST-M dataset. Finally we performed domain adaptation for the purpose of 3D object detection using Zenuity's SYN dataset as the source domain dataset and the KITTI dataset as the target domain.

3.1.1 Cityscapes

The Cityscapes dataset is a large-scale dataset that contains a diverse set of video sequences recorded in street scenes from 50 different cities in Europe. It is divided into one batch with 5,000 images accompanied with high-quality pixel-level annotations for semantic segmentation, and one batch with 20,000 images with more course-detailed annotations. The Cityscapes dataset is one of the largest real-world gathered datasets with ground-truth semantic segmentation annotations. It is commonly used to benchmark pixel-level and instance-level semantic segmentation models in the field of computer vision.

3.1.2 GTA 5

Recent advances in computer vision and deep learning have paved the way for high performing image recognition models powered by large datasets. However, for tasks such as semantic segmentation and 3D object detection gathering data is very costly as they require manual annotations of multiple objects per frame. To enable researchers to train models on huge datasets for these tasks Richter *et al.* presented a dataset extracted from the video game GTA 5 in their paper *Playing for Data: Ground Truth from Computer Games* [52]. They extracted 25,000 frames from the video game and implemented a solution that allowed for automatic extraction of semantic segmentation annotations.

For a dataset like Cityscapes, annotating each frame takes approximately 1 hour. The GTA 5 dataset has since then been commonly used as a benchmark for unsupervised domain adaptation models, and in many cases used together with Cityscapes for domain adaptation for the purpose of semantic segmentation with Cityscapes as the target domain. Since our objective involved performing domain adaptation between a synthetic dataset and a dataset extracted in the real world, evaluating models on the GTA 5 to Cityscapes domain adaptation problem seemed reasonable.

3.1.3 MNIST

The MNIST (Mixed National Institute of Standards and Technology) digit dataset is a dataset with images of handwritten digits. Each handwritten digit is associated with a label indicating what digit the image corresponds to. The dataset contains 60,000 training samples and 10,000 test samples. The MNIST dataset is commonly used to provide a performance metric for image classification models. Each image is grayscale and has a $28x28$ resolution. Because of the low resolution and the fact that it poses a relatively simple classification problem, it is also common to use the MNIST digit dataset as a proof of concept when developing new computer vision algorithms. Samples of images from the MNIST dataset are visualized in Figure 3.1.

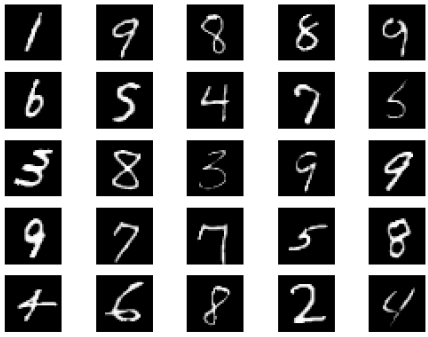

Figure 3.1: *Images from the MNIST dataset.*

3.1.4 MNIST-M

The MNIST-M dataset consists of modified images from the MNIST dataset. Each image in the MNIST dataset has been randomly blended with an image patch from the BSD300 dataset. The BSD300 dataset consists of natural images and was produced at Berkeley with the purpose of evaluating and developing semantic segmentation models. The images in MNIST-M which are produced as the results of this blending are RGB as opposed to grayscale (like the original MNIST images). The MNIST and MNIST-M datasets have been used to evaluate the performance of domain adaptation models, as it has been shown that classification models that are trained on MNIST and evaluated on MNIST-M perform poorly. Samples of images from the MNIST-M dataset are visualized in Figure 3.2.

Figure 3.2: *Images from the MNIST-M dataset.*

3.2 CycleGAN

CycleGAN has been shown to produce visually compelling results for image-to-image translation problems between two domains. By enforcing cycle-consistency during translation, and thus constrain the mapping from one domain to the other, it was a suitable choice to build on for our purposes. The performance of the task network, which is trained on translated images during the domain adaptation process, depends on that annotations of the original synthetic images remain valid after the images have been translated to the target domain. By both enforcing cycle-consistency and using the loss of the task network to restrict the generators from making semantic changes to the images we reasoned that this would be a suitable solution to investigate for our objective.

3.2.1 Model Architecture

CycleGAN consists of two generator- and two discriminator networks. The generators are optimized to provide an image-to-image translation between the two image domains, X_A and X_B. The discriminators are trained to differentiate between whether images which they receive as inputs are sampled from their respective domains or if the images have been translated from the opposite domain by a generator network. The discriminator D_A evaluates whether images are sampled from domain A and the discriminator D_B evaluates whether images are sampled from domain B.

The generators are trained to fool the discriminators into labeling the images that they have translated as coming from the distribution that they are trained to model. Generator G_{AB} translates images from domain A to domain B and the generator G_{BA} translates images from domain B to domain A. What sets the CycleGAN model apart from the traditional generator and discriminator setup in traditional Generative Adversarial Networks is that images that are translated to the opposite domain and then translated back to their original domain should be identical to their original form. By enforcing this circle translation to produce an identical image to the original, the translation mapping from one domain to the other is constrained from making changes to the original images so that the image can not be retained by re-translation. This prohibits the generators from making structural changes to the images during translation, and helps ensure that the semantics of the original image are preserved. See Figure 3.3 for a visualization of how the inputs are propagated through the CycleGAN architecture. To further restrict the mapping from one domain to the other we also applied the identity loss presented by the authors.

Two common representations for the generator networks are Residual Networks and U-Net architectures. Both network architectures have been developed to preserve low level details of the original input image during translation. We experimented with using both of these network architectures for the different domain adaptation problems we looked to solve. As for the discriminator representation we used the same architecture for every problem as this architecture provided a high classification performance irrespective of the datasets used. The

residual network, U-Net architecture and the discriminator architecture are described in the following sections.

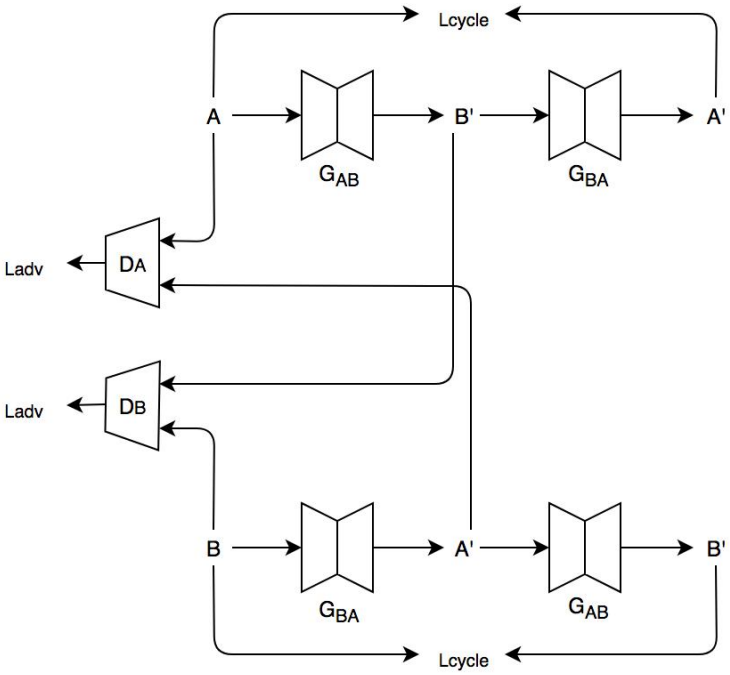

Figure 3.3: *Figure displaying how the inputs and outputs are propagated through the CycleGAN architecture.*

Generator Architecture: Residual Network

One challenge with training deep neural network is that as the network gets larger earlier layers in the network has an increasingly smaller affect on the output of the network. In turn, this results in that the gradient that is backpropagated and used to update the weights of these layers gets increasingly smaller. In their paper *Deep Residual Learning for Image Recognition* [20] He *et al.* presented the residual network as a solution to the vanishing gradient problem. This network architecture is composed of *residual blocks*, in which the input signal is propagated through one convolutional layer followed by a batch normalization layer and a rectified-linear unit followed by a second convolutional layer and a final batch normalization layer. The output is then added to the input of the block through a skip-connection. The architecture of a residual block can be seen in Figure 3.4.

By adding the input to the output of the block the convolutional layers in the residual block are trained to learn the residual to the input signal instead of a complete remapping of the input, and this allows for training of significantly

deeper neural networks than previous neural network architectures. The depth of the network is of central importance for visual recognition neural networks in computer vision, and the researchers were able to obtain a 28 % relative improvement on the COCO object detection dataset using this new network architecture. For the full architecture of the residual network image-to-image model used as generator representation, see Table 3.1.

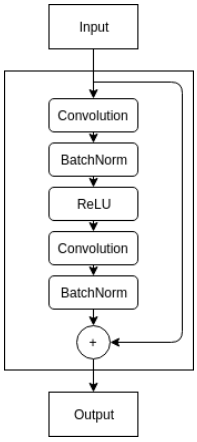

Figure 3.4: *Architecture of a residual block.*

Table 3.1: *Residual network generator architecture.*

	ResNet Architecture				
Block	Filters	Filter Size	Stride	Padding	Inst. Norm.
Conv+ReLU	64	7	1	1	Yes
Conv+ReLU	64	3	2	1	Yes
Conv+ReLU	128	3	2	1	Yes
9 × ResBlock	128	3	1	1	Yes
Conv+ReLU	128	3	0.5	1	Yes
Conv+ReLU	64	3	0.5	1	Yes
Conv+TanH	3	7	1	1	No

Generator Architecture: U-Net

The U-Net architecture was first presented in the paper *U-Net: Convolutional Networks for Biomedical Image Segmentation* by Ronneberger *et al.* [53]. Many of the previous image-to-image network architectures used the encoder-decoder architecture, where the image is first downsampled by the encoder part of the network into a low dimensional feature representation and then upsampled by the decoder network to produce the image output. This architecture allowed for

training of deep neural networks despite the high dimensionality of the input to the network. The problem with this architecture is that many of the low level features in the image are lost during the downsampling phase, and objects and structures in the input image are therefore hard to preserve during the translation for deep network architectures. The U-Net architecture has a high resemblance to the encoder-decoder architecture but adds additional skip-connections between the layers in the encoder network and decoder network. For a simplified visualization of the skip-connections between the encoder and decoder layers, see Figure 3.5.

Similar to the residual network, the U-Net architecture will because of the skip-connections only have to consider learning the residual to the input for the mapping to the target domain, and these skip-connections will also help preserve the low level information in the input during translation. In order to preserve variance in the output of the generator, and reduce the risk of the generator finding a many-to-one mapping from the source domain to the target domain, dropout is used in the U-Net architecture with a dropout rate of 0.5. It is common to add noise to the input of generators in generative adversarial networks, since a one-to-one or many-to-one mapping is often not sought after. See Table 3.2 for the full architecture.

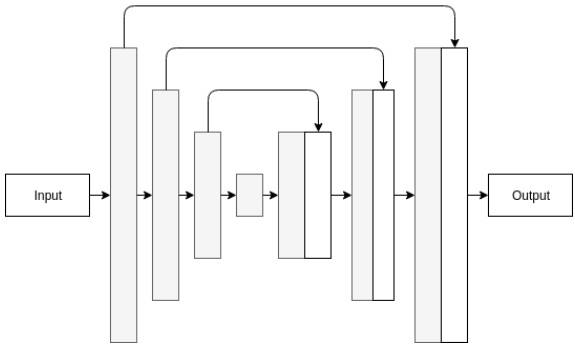

Figure 3.5: *Simplified architecture of a U-Net, where the skip-connections between layers are visualized.*

Discriminator Architecture

Similar to the authors we used a four layer convolutional neural network architecture for the discriminator. We use a PatchGAN discriminator that outputs a prediction of whether each patch in the image is sampled from the discriminator's distribution or if it has been generated.

In addition to the architecture described in the CycleGAN paper we implement a multi-scale discriminator. Instead of using one discriminator that determines the validity of the image in its original scale only, the multi-scale discriminator evaluates the validity of the image at several image scales. The multi-scale discriminator was first introduced by Wang *et al.* [59] and their implementation

Table 3.2: *The architecture of the U-Net generator.*

U-Net Architecture						
Block	Filters	Filter Size	Stride	Padding	Inst. Norm.	Dropout
Conv+LReLU	64	4	2	1	No	-
Conv+LReLU	128	4	2	1	Yes	-
Conv+LReLU	256	4	2	1	Yes	-
Conv+LReLU	512	4	2	1	Yes	0.5
Conv+LReLU	512	4	2	1	Yes	0.5
Conv+LReLU	512	4	2	1	Yes	0.5
Conv+LReLU	512	4	0.5	1	Yes	0.5
Conv+LReLU	512	4	0.5	1	Yes	0.5
Conv+LReLU	256	4	0.5	1	Yes	-
Conv+LReLU	128	4	0.5	1	Yes	-
Conv+LReLU	64	4	0.5	1	Yes	-
Conv+TanH	3	4	0.5	1	No	-

consisted of three separate discriminators with identical architectures that are fed as inputs the image in its original form, the image downsampled by a scale factor of two and the image downsampled by a factor of four. The discriminator that is fed the image of lowest spatial resolution will have the largest receptive field and will help guide the generator to produce images that are resembling the target domain at a high level, while the discriminator that is fed the image in its original form will be trained to evaluate the image at a courser level. We chose to implement two discriminator networks per multi-scale discriminator instead of three as in the original implementation.

The architecture of each of the two discriminators consists of four subsequent layers of strided convolutions with step size two, where the layers have 64, 128, 256 and 512 filters respectively. Each layer is followed by a leaky rectified-linear activation function. The final layer has a single-filter output that is the discriminator's prediction of the validity of the image at each patch of the image. For the full architecture of the multi-scale discriminator where both output layers are listed, see Table 3.3.

Table 3.3: *The architecture of the multi-scale CycleGAN discriminator.*

Discriminator Architecture					
Block	Filters	Filter Size	Stride	Padding	Inst. Norm.
Conv+LReLU	64	4	2	1	No
Conv+LReLU	128	4	2	1	Yes
Conv+LReLU	256	4	2	1	Yes
Conv+LReLU	512	4	2	1	Yes
Conv	1	4	1	1	No

3.2.2 Loss Functions

Adversarial Loss

Just as in the traditional Generative Adversarial Network setup, the generators in the CycleGAN architecture are trained to fool the discriminators. Each discriminator has the objective of differentiating between whether an input image is sampled from the discriminator's corresponding domain or if the image has been translated from the opposite domain through a generator network. Using a least squares loss has been shown to stabilize training for GANs [40], and to this end we opted for using this adversarial loss instead of the traditional cross-entropy loss. The adversarial loss of the domain discriminator D_B can be expressed as:

$$\begin{aligned}\mathcal{L}_{LSGAN}^{D_B}(G_{AB}, D_B, X_A, X_B) = &\frac{1}{2}\mathbb{E}_{x_B \sim p_{data}(X_B)}[(D_B(x_B) - 1)^2] \\ &+ \frac{1}{2}\mathbb{E}_{x_A \sim p_{data}(X_A)}[D_B(G_{AB}(x_A)^2]\end{aligned} \tag{3.1}$$

The discriminator is trained to label images that are sampled from its domain as 1 and images that are translated samples from the other domain as 0. The generator G_{AB} is trained to fool the discriminator D_B into labeling its translated images from domain A as having been sampled from domain B, *i.e.* it is trained to fool the discriminator into labeling its generated samples as 1. The adversarial loss for the generator G_{AB} can thus be expressed as:

$$\mathcal{L}_{LSGAN}(G_{AB}, D_B, X_A) = \frac{1}{2}\mathbb{E}_{x_A \sim p_{data}(X_A)}[(D_B(G_{AB}(x_A)) - 1)^2] \tag{3.2}$$

Cycle-consistency Loss

To constrain the mapping from one domain to the other the authors propose a cycle-consistency loss, which restricts the generators from making structural changes to the images during the domain translation. An image that is translated from one domain to the other and then translated back to its original domain shall exactly match the image in its original form. This loss is particularly important for us since we need the annotations of the original image to be valid in the image's translated form so that the task network can be trained on the translated image. This loss is calculated by a pixel-wise loss between the re-translated image and the original. We found that using the mean absolute error gave the best results. The cycle-consistency loss is defined as:

$$\begin{aligned}\mathcal{L}_{cyc}(G_{AB}, G_{BA}) = &\mathbb{E}_{x_A \sim p_{data}(X_A)}[\|G_{BA}(G_{AB}(x_A)) - x_A\|_1] \\ &+ \mathbb{E}_{x_B \sim p_{data}(X_B)}[\|G_{AB}(G_{BA}(x_B)) - x_B\|_1]\end{aligned} \tag{3.3}$$

Identity Loss

In many scenarios it is interesting to also preserve the color composition of the input image during translation. To this end the authors also propose the use of

an identity mapping loss. This loss enforces that *e.g.* an image from domain A that is fed through the generator network trained to translate images to domain A should remain unchanged. Without the use of this loss, the generators are free to change the tint and color of objects in the original image when there is no need to, by for example changing the color of a blue car to red. This loss is also helpful for preserving variation in the images from the source domain during translation. Generators could otherwise be predispositioned to for example change all colors of an object in the source domain image to the same color during translation, since this might have a higher chance of fooling the discriminator. The identity loss is defined as:

$$\begin{aligned}\mathcal{L}_{id}(G_{AB}, G_{BA}) = \mathbb{E}_{x_A \sim p_{data}(X_A)}[\|G_{BA}(x_A) - x_A\|_1] \\ + \mathbb{E}_{x_B \sim p_{data}(X_B)}[\|G_{AB}(x_B) - x_B\|_1]\end{aligned} \tag{3.4}$$

Total Loss

The full objective of the generators is thus to minimize the sum of these loss functions:

$$\begin{aligned}\mathcal{L}(G_{AB}, G_{BA}, D_A, D_B) = \mathcal{L}_{LSGAN}(G_{AB}, D_B, X_A) \\ + \mathcal{L}_{LSGAN}(G_{BA}, D_A, X_B) \\ + \lambda_{cyc}\mathcal{L}_{cyc}(G_{AB}, G_{BA}) \\ + \lambda_{id}\mathcal{L}_{id}(G_{AB}, G_{BA})\end{aligned} \tag{3.5}$$

The hyper parameters λ_{cyc} and λ_{id} determine the relative priority of the cycle-consistency loss and the identity mapping loss.

3.2.3 Training

We use a history of 50 generated samples to train the discriminators. Training the discriminators on batches of historical samples has been shown to reduce model oscillation during training. We also adopt the strategy of Mao *et al.* in [40] and train the networks to minimize the least squares error instead of the cross-entropy loss for the adversarial training. To update the weights of the neural networks we use the Adam optimizer [28]. The Adam optimizer keeps an adaptive learning rate by using a running average of the first and second order moments of the gradients to update the weights of the network layers. The Adam optimizer has two hyper parameters named β_1 and β_2 which control the decay of these running averages during training. The setting of these hyper parameters can be seen in Table 3.4. Similar to the authors we adapt a linear decay of the learning rate after 100 epochs.

3.3 Domain Adaptation - MNIST and MNIST-M

To validate that the approach of adding a task network to the traditional GAN architecture would force the generator to provide a domain agnostic and task suitable representation, we adopted the *PixelDA* approach described by Bousmalis

Table 3.4: *Hyper parameter settings for CycleGAN training.*

CycleGAN Parameters	
Parameter	Setting
Learning rate	0.0002
β_1	0.5
β_2	0.999
Buffer size	50
λ_{cyc}	10
λ_{id}	5

et al. in their paper *Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks* [4], in which a task network considers both images from the source domain and translated samples from the source domain made to resemble images from the target domain. The task network is trained jointly with the generator network during the adversarial process, and can after the training phase be evaluated on images from the target domain, which the generator has been trained to model during the adversarial process. The domain adaptation can be deemed successful if the task network performs better on images from the target domain than an identical task network that has been trained on images from the source domain.

We used MNIST as our source domain representation (X_A) and MNIST-M as the target domain representation (X_B). A task network that is trained during domain adaptation on images from MNIST that are translated to resemble MNIST-M was compared based on its ability to classify images from MNIST-M with an identical network that was trained on MNIST. As the generator is trained to both perform a translation of the image to the target domain, which is enforced by minimizing the adversarial loss, and generate an image where the annotations are still valid for the translated image, the task network receives input images that are resembling of the images from the target domain and where the semantics of the images are in line with the annotations.

3.3.1 Model Architecture

We adopted the PixelDA architecture presented by Bousmalis *et al.*, in which a generator network, G, translates images from the source domain to the target domain and is trained to fool a discriminator network, D, into labeling these images as having been sampled from the target domain. The translated images are also fed as input to a task network, T, that carries out its intended task based on these images. In our case we used the MNIST dataset as source domain and the MNIST-M dataset as target domain.

Translated images from the MNIST dataset are fed to the task network which is trained to classify these images based on their original ground truth annotations. The gradient of the classification loss is backpropagated to tune both the task network and the generator to provide a higher classification performance.

This joint optimization of both the task network and generator network will force the generator to preserve the semantics in the original image during translation so that the task network can provide an accurate classification of the translated image. We adopted the residual network representation for the generator, similar to the authors of PixelDA, and found that this gave better classification scores compared to the U-Net representation when we later evaluated the task network on images from the target domain. The architecture can be seen in Figure 3.6.

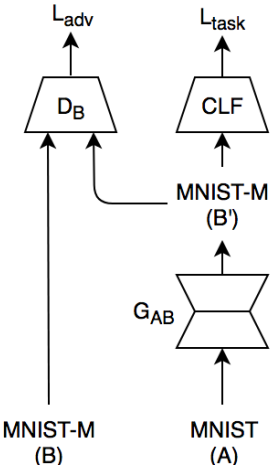

Figure 3.6: *Model architecture of PixelDA for MNIST to MNIST-M domain translation.*

3.3.2 Loss Functions

To train the classifier and generator to provide an accurate classification of the translated images we minimize the categorical-crossentropy loss function:

$$\mathcal{L}_t(G,T) = \mathbb{E}_{x_A,y_A,z}[-y_A^\intercal \log T(G(x_A)) - y_A^\intercal \log T(x_A)] \tag{3.6}$$

To make sure that samples that are generated closely resemble samples from the target domain we train the generator to minimize the least-square adversarial loss, similarly to the domain translation generators in the CycleGAN framework. The adversarial loss of the generator is defined as:

$$\mathcal{L}_{LSGAN}(G,D,X_A) = \frac{1}{2}\mathbb{E}_{x_A\sim p_{data}(X_A)}[(D(G(x_A)-1)^2] \tag{3.7}$$

The total loss of the generator and task network is:

$$\begin{aligned}\mathcal{L}(G,T,D) = \mathcal{L}_{LSGAN}(G,D,X_A) \\ + \lambda_t \mathcal{L}_t(G,T)\end{aligned} \tag{3.8}$$

Where λ_t controls the relative priority of the task loss. For our experiments we set it to the same value as in the PixelDA paper, which is 0.01. The least-squares adversarial loss of the discriminator network is defined as:

$$\begin{aligned}\mathcal{L}^{D}_{LSGAN}(G, D, X_A, X_B) = &\frac{1}{2}\mathbb{E}_{x_B \sim p_{data}(X_B)}[(D(x_B) - 1)^2] \\ &+ \frac{1}{2}\mathbb{E}_{x_A \sim p_{data}(X_A)}[(D(G(x_A))^2]\end{aligned} \tag{3.9}$$

3.4 Translating GTA 5 to Cityscapes

We implemented an unsupervised image-to-image translation method for translating images from the synthetic GTA 5 dataset and the Cityscapes dataset, as this closely resembled the problem that we were trying to solve. To this end we implemented the CycleGAN model presented by Isola *et al.* [66] in their paper *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* to perform unsupervised domain adaptation between these two datasets. CycleGAN has been shown to produce visually compelling results for similar domain adaptation problems. For a list of loss functions that the network were trained to minimize see section 3.2.2.

3.4.1 Model Architecture

We adopt the architecture and hyper parameters described in the original paper where CycleGAN was presented. We evaluate the generated images qualitatively for both the residual network generator representation and the U-Net generator representation. Images from both domains are scaled to 256×512 and standardized to the range $[-1, 1]$ before being fed as inputs to the model. In addition to the architecture described in the CycleGAN paper we add a second output of the discriminator, as described in Section 3.2.1.

3.5 3D Object Detection - Datasets

As stated in the problem formulation in Section 1.2, we utilize two different datasets for 3D object detection in this thesis: KITTI and SYN. These datasets are in this section more carefully introduced.

3.5.1 KITTI

The publicly available KITTI dataset [15] consists of 7481 training examples and 7518 testing examples extracted from a number of sequences, where each example contains two images from a forward-facing stereo camera rig and a LiDAR point cloud from a Velodyne HDL-64E [25]. In this thesis we do however only utilize one of the images together with the point cloud. The examples were collected in both urban areas and on highways in the German city Karlsruhe, and

were all captured in daylight and clear weather. The image and corresponding point cloud for a typical training example were shown in Figure 2.1.

Each training example is also annotated with both 2D and 3D ground truth bounding boxes for cars, pedestrians and cyclists which are visible in the images. 2D bounding boxes are parameterized as $(u_{min}, u_{max}, v_{min}, v_{max})$, where (u_{min}, v_{min}) are the pixel coordinates of the top-left box corner. 3D bounding boxes are parameterized as $(x, y, z, h, w, l, \theta)$, as described in the beginning of Section 2.4. All 2D bounding boxes for one training example were visualized in Figure 2.3, whereas 3D bounding boxes for another example were visualized in Figure 1.1.

Ground truth annotations for the test examples are however not publicly available. To evaluate a model's performance on the test set (*KITTI test*) one instead has to submit predicted 3D bounding boxes to the official 3D object detection benchmark leader board [14].

We followed [47, 63] and other previous works and split the training examples roughly in half into the training and validation sets presented by Chen *et al.* in [8]. The split was designed such that all examples from a given sequence exclusively belong to either the training or the validation set. This results in a training set (*KITTI train*) containing 3712 examples and a validation set (*KITTI val*) containing 3769 examples.

We also do a random 90 % - 10 % split of the training examples in order to obtain a larger training set, which is used to train models which will be evaluated on *KITTI test*. This results in a training set (*KITTI train random*) containing 6733 examples and a validation set (*KITTI val random*) containing 748 examples.

3.5.2 SYN

SYN is an internal dataset consisting of synthetic mono-camera images and LiDAR point clouds, together with annotated 2D and 3D ground truth bounding boxes for cars, pedestrians and cyclists. The examples are all captured in a simulated urban environment in daylight and clear weather conditions. Overall, SYN can roughly be viewed as a synthetic version of KITTI, and it also utilizes the same bounding box parameterizations. The image and corresponding LiDAR point cloud for a typical example from SYN are visualized in Figure 3.7.

To make the synthetic LiDAR point clouds more closely resemble those found in KITTI, we added Gaussian noise with a standard deviation of 0.02 to the (x, y, z) components of each point, and randomly selected a 25 % subset of all points.

SYN contains 25000 examples in total, which we randomly split into a training set (*SYN train*) containing 20000 examples and a validation set (*SYN val*) containing 5000 examples.

3.6 3D Object Detection - Frustum-PointNet

We chose the Frustum-PointNet architecture presented by Qi *et al.* in [47] as our main model for 3D object detection. Frustum-PointNet was chosen because of its

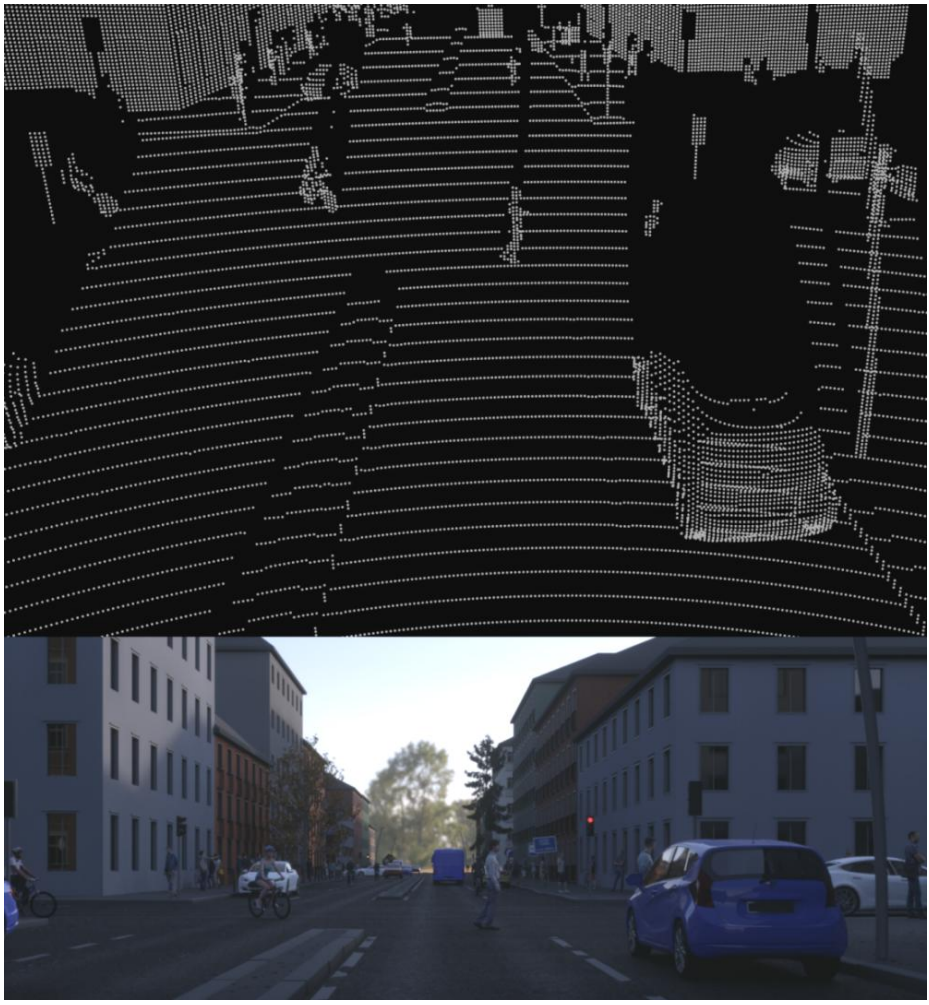

Figure 3.7: *Visualization of the image and corresponding LiDAR point cloud for a typical example from the SYN dataset. The point cloud visualization was created using Open3D [64].*

SOTA performance on KITTI (it was the top-performing entry for all three object classes on the 3DOD leader board [14] at the time) and the fact that it takes 2D bounding boxes as input, which was believed to be beneficial in our automatic annotation application where 2D ground truth is available.

The implemented Frustum-PointNet architecture is in this section described in detail, together with a description of how the model was trained.

3.6.1 Model Architecture

Frustum-PointNet takes a LiDAR point cloud and a 2D bounding box containing an object as input, and outputs an estimated 3D bounding box for the object. A schematic overview of the architecture is shown in Figure 3.8.

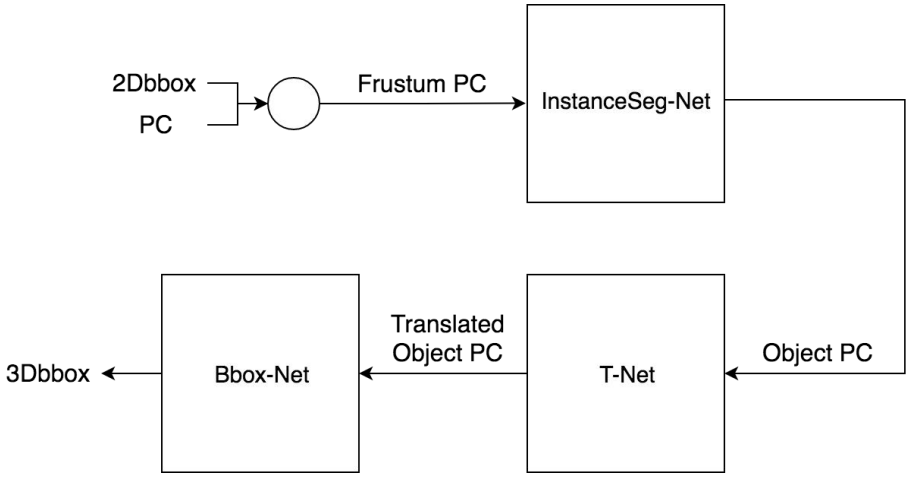

Figure 3.8: *Schematic overview of the Frustum-PointNet architecture for 3D object detection.*

The 2D bounding box is first geometrically extruded to extract the corresponding frustum point cloud, containing all points in the LiDAR point cloud which lie inside of the 2D box when projected onto the image plane. The frustum point cloud thus contains all points which potentially could belong to the object of interest.

The frustum point cloud is then fed to *InstanceSeg-Net* which performs binary classification of each point, predicting whether or not the point belongs to the object. All positively classified points are extracted to obtain the segmented object point cloud, which thus contains all points believed to belong to the object of interest.

This object point cloud is then fed to *T-Net* which outputs an estimate of the 3D bounding box center position (x, y, z), expressed as residuals relative to the object point cloud centroid.

Finally, the object point cloud is translated according to the estimated box center and fed to *Bbox-Net*, which outputs an estimate of the full set of 3D bounding box parameters $(x, y, z, h, w, l, \theta)$.

Qi *et al.* presented two different versions of Frustum-PointNet in [47]: version 1, in which the three subnetworks are based on the PointNet [48] architecture, and version 2 which is based on PointNet++ [49]. The difference in reported performance is quite significant in favor of version 2, but version 1 still corresponds to near SOTA performance and was believed to be more straightforward to implement. Our implementation is thus based on version 1.

To make the implementation slightly more straightforward we also chose to focus solely on the detection of cars. This choice does not have a big effect on the model architecture, but it does make the pre- and post-processing stages of the implementation somewhat less complicated.

Frustum Point Cloud Extraction

To extract a frustum point cloud from the input LiDAR point cloud $P = \{p_1, \ldots, p_N\}$ and 2D bounding box $(u_{min}, u_{max}, v_{min}, v_{max})$, each point $p_i = (x_i, y_i, z_i, r_i)$ is first projected onto the image plane to obtain its pixel coordinates $p_i^{img} = (u_i, v_i)$. All $N' \leq N$ points which now lie within the 2D bounding box, *i.e.* which fulfill

$$\begin{cases} u_{min} \leq u_i \leq u_{max} \\ v_{min} \leq v_i \leq v_{max} \end{cases} \tag{3.10}$$

are then extracted to obtain $P' = \{p_1, \ldots, p_{N'}\}$. To make each frustum point cloud contain the same number of points, we randomly sample n points from P' to finally obtain $P_{frustum} = \{p_1, \ldots, p_n\}$.

Before the frustum point cloud $P_{frustum}$ is fed to any of the subnetworks, it is also normalized by rotating its points such that the frustum point cloud center axis is orthogonal to the image plane. This is achieved by finding any point that projects onto the center pixel of the 2D bounding box, *e.g.* by using the pseudo inverse of the camera projection matrix, and computing its angle in the XZ camera coordinate plane. This normalizing frustum point cloud rotation is illustrated in Figure 3.9.

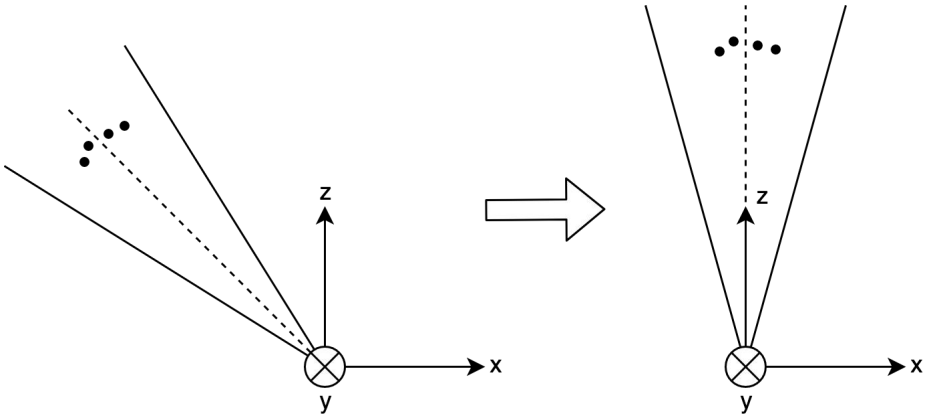

Figure 3.9: *Illustration of the frustum point cloud normalization in camera coordinates. The original frustum (left) is rotated such that its center axis is orthogonal to the image plane.*

InstanceSeg-Net

The subnetwork *InstanceSeg-Net* takes the normalized frustum point cloud $P_{frustum} = \{p_1, \ldots, p_n\} \subset \mathbb{R}^4$ as input and outputs two classification scores for each of the n input points. The scores correspond to the predicted probability that a given point belongs to the object of interest or not. *InstanceSeg-Net* is a slightly modified version of the PointNet [48] segmentation network with two output classes, as described in Section 2.2. The network architecture is schematically illustrated in Figure 3.10.

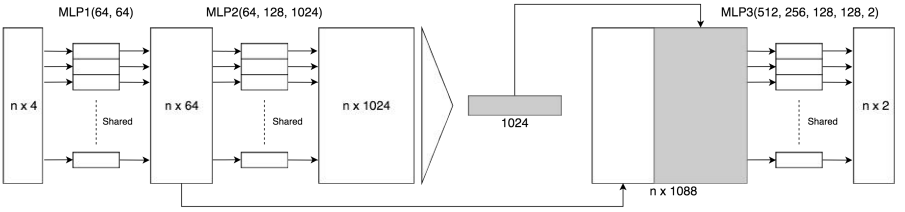

Figure 3.10: *Schematic illustration of the model architecture for the InstanceSeg-Net subnetwork.*

A shared multi-layer perceptron (MLP) with two layers and ReLU as the activation function is applied to each input point $p_i = (x_i, y_i, z_i, r_i)$, to obtain an intermediate feature vector $\tilde{f}_i \in \mathbb{R}^{64}$. A second shared MLP with three layers is then applied to each $\tilde{f}_i \in \mathbb{R}^{64}$ to obtain a feature vector $f_i \in \mathbb{R}^{1024}$. Max pooling is then applied to the feature vectors $f_1, \ldots, f_n$ to obtain a global feature vector $f \in \mathbb{R}^{1024}$. Then, f is repeatedly concatenated with each $\tilde{f}_i \in \mathbb{R}^{64}$ to obtain point-wise feature vectors $g_i \in \mathbb{R}^{1088}$ containing both local and global information. Finally, a third shared MLP with five layers is applied to each $g_i \in \mathbb{R}^{1088}$ to obtain two point-wise classification scores. The final layer of this third MLP has a softmax activation function instead of ReLU, in order to normalize the classification scores.

The three MLPs are specified in detail in the schematic illustration in Figure 3.10. For instance, $MLP2(64, 128, 1024)$ means that the first layer of the second MLP has 64 units, that the second layer has 128 units and the third layer has 1024 units.

Our implementation differs from the original one presented by Qi *et al.* in the removal of all batch normalization layers in the MLPs. We found that using batch normalization caused problems during training in the form of highly volatile performance on the validation set, and thus chose to remove it.

When training the model on *SYN train*, the architecture is also modified to only take the (x, y, z) components of each point p_i as input. This decision was made in order to improve the model's ability to generalize to KITTI, as the synthetic reflectance values in SYN are known to not be accurately modelled.

All $M \leq n$ points which are classified as belonging to the object of interest are extracted to obtain $P'_{frustum} = \{p_1, \ldots, p_M\}$. To make each object point cloud contain the same number of points, we randomly sample m points from $P'_{frustum}$

to obtain $P_{object} = \{p_1, \ldots, p_m\}$. Before the object point cloud P_{object} is fed to the next subnetwork, it is also normalized by subtracting its (x, y, z) centroid off of all its points.

T-Net

The *T-Net* subnetwork regresses residuals, relative to the object point cloud centroid, for the 3D bounding box center position (x, y, z). It takes the normalized object point cloud $P_{object} = \{p_1, \ldots, p_m\} \subset \mathbb{R}^4$ as input, but actually only uses the (x, y, z) components of each point p_i. *T-Net* is essentially a modified version of the PointNet [48] classification network, as described in Section 2.2. The network architecture is schematically illustrated in Figure 3.11.

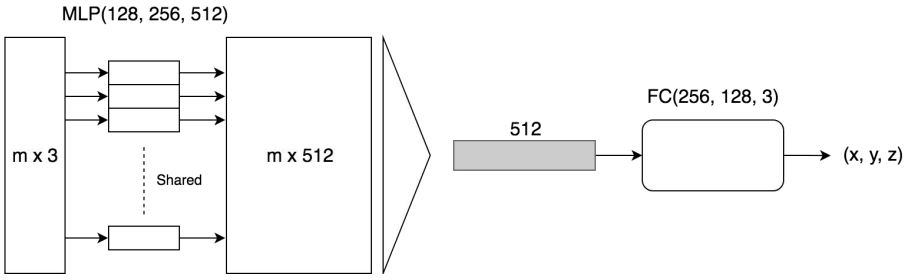

Figure 3.11: *Schematic illustration of the model architecture for the T-Net subnetwork.*

A shared MLP with three layers and ReLU is applied to each input point $\tilde{p}_i = (x_i, y_i, z_i)$, to obtain a feature vector $f_i \in \mathbb{R}^{512}$. Max pooling is then applied to the feature vectors $f_1, \ldots, f_m$ to obtain a global feature vector $f \in \mathbb{R}^{512}$. Finally, f is fed through a fully-connected network with three layers to regress (x, y, z) residuals. ReLU is applied to the output of the first two layers but not the output layer. $FC(256, 128, 3)$, as seen in Figure 3.11, means that the first layer has 256 units, that the second layer has 128 units and the output layer has 3 units. Just as for *InstanceSeg-Net*, our implementation differs from the original one in the removal of all batch normalization layers.

Before being fed to final subnetwork, the object point cloud P_{object} is translated by subtracting the regressed (x, y, z) residuals off of all its points.

Bbox-Net

The *Bbox-Net* subnetwork outputs an estimate for the full set of 3D bounding box parameters $(x, y, z, h, w, l, \theta)$. It takes the translated object point cloud $P_{object} = \{p_1, \ldots, p_m\} \subset \mathbb{R}^4$ as input, but only uses the (x, y, z) components of each point p_i. The *Bbox-Net* architecture is essentially identical to that of *T-Net*, and only differs in terms of number of layers and units. The network architecture is schematically illustrated in Figure 3.12.

As seen in Figure 3.12, *Bbox-Net* outputs $3 + 3 + 2N_H$ values. The first 3 values are regressed residuals for the 3D bounding box center position (x, y, z), relative

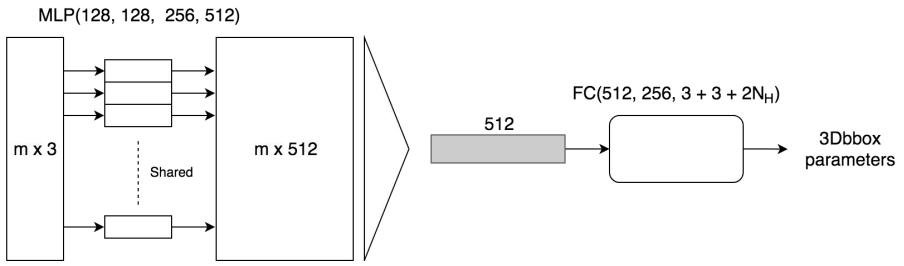

Figure 3.12: *Schematic illustration of the model architecture for the Bbox-Net subnetwork.*

to the center position estimated by *T-Net*. The next 3 values are regressed residuals for the 3D box size dimensions (h, w, l), relative to the mean ground truth box dimensions in the training set. The last $2N_H$ values correspond to the estimation of the yaw angle θ, for which a classification-regression hybrid is used.

The yaw angle range $[-\pi, \pi[$ is split into N_H equally sized angle bins. For each angle bin, *Bbox-Net* then outputs one classification score, corresponding to the predicted probability that the true yaw angle lies in this angle bin, and one regressed residual relative to the center angle of the bin. The actual θ estimate is thus obtained by finding the angle bin with the largest predicted probability, and adding the corresponding regressed residual to the center angle of that bin.

3.6.2 Implementation Details

The Frustum-PointNet model was entirely implemented in Python using the PyTorch framework [46]. The shared MLPs were implemented using PyTorch's 1D convolution module (nn.Conv1d) with the kernel size set to one.

We followed Qi *et al.* and set $n = 1024$, $m = 512$. We also chose to set $N_H = 4$ with $\{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}\}$ as the bin center angles, which correspond to the four most common ground truth yaw angles ($\frac{\pi}{2}$ corresponds to a car driving straight in the direction of travel of the ego-vehicle). This choice results in the following four angle bins:

$$\begin{cases} \theta \in bin_0 \iff \theta \in [-\frac{\pi}{4}, \frac{\pi}{4}[\\ \theta \in bin_1 \iff \theta \in [\frac{\pi}{4}, \frac{3\pi}{4}[\\ \theta \in bin_2 \iff \theta \in [\frac{3\pi}{4}, \pi[\cup [-\pi, -\frac{3\pi}{4}[\\ \theta \in bin_3 \iff \theta \in [-\frac{3\pi}{4}, -\frac{\pi}{4}[\end{cases} \tag{3.11}$$

We also follow the procedure of Qi *et al.* in order to obtain ground truth labels for the *InstanceSeg-Net* subnetwork: all points in the frustum point cloud $P_{frustum}$ which lie inside the ground truth 3D bounding box are considered to belong to the object.

Loss Function

The three subnetworks are jointly optimized using a multi-task loss function:

$$\begin{aligned}\mathcal{L} = \mathcal{L}_{InstanceSeg} + \lambda(\mathcal{L}_T + \mathcal{L}_{Bbox-c} \\ + \mathcal{L}_{Bbox-s} + \mathcal{L}^{reg}_{Bbox-\theta} + \mathcal{L}^{cls}_{Bbox-\theta} + \gamma\mathcal{L}_{corner})\end{aligned} \tag{3.12}$$

$\mathcal{L}_{InstanceSeg}$ is the negative log-likelihood loss applied to the output of *InstanceSeg-Net*, $\mathcal{L}_T$ is for the 3D box center regression in *T-Net* and $\mathcal{L}_{Bbox-c}$ for the center regression in *Bbox-Net*, $\mathcal{L}_{Bbox-s}$ is for the 3D box size regression, $\mathcal{L}^{reg}_{Bbox-\theta}$ is for regression of the outputted residual corresponding to the ground truth yaw angle bin, $\mathcal{L}^{cls}_{Bbox-\theta}$ is the negative log-likelihood loss applied to the heading classification output, and $\mathcal{L}_{corner}$ is the regularizing corner loss introduced by Qi *et al.* We again follow Qi *et al.* and set $\lambda = 1, \gamma = 10$.

The corner loss aims to jointly penalize the center, size and yaw angle estimates for optimal 3D bounding box estimation. It is the sum of the $L1$ distances between the eight corners of the ground truth 3D bounding box and the predicted one.

The Huber loss (also called the smooth $L1$ loss) is used for all regression tasks, and is defined according to:

$$\mathcal{L}_{Huber}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2}, & |y - \hat{y}| \geq 1 \end{cases} \tag{3.13}$$

Data Augmentation

To prevent overfitting during training, a number of data augmentation techniques are applied. Qi *et al.* used three different forms of augmentation: random perturbation of 2D bounding boxes, random flipping of frustum point clouds and random shifting.

Ground truth 2D bounding boxes are taken as input and are augmented by random translation and scaling before the frustum point cloud extraction. The 2D box center is translated by distances sampled from Uniform$[-0.1w, 0.1w]$ and Uniform$[-0.1h, 0.1h]$, where (w, h) is the original 2D box width and height. The width and height are then also independently rescaled by a factor sampled from Uniform$[0.9, 1.1]$.

The extracted frustum point cloud is then, after the normalizing rotation, flipped along the YZ camera coordinate plane with 0.5 probability. It is also shifted in the Z-axis direction by a distance sampled from Uniform$[-20, 20]$, in order to augment the depth of points.

We also apply a technique used by Zhou and Tuzel in [65] to augment the ground truth yaw angles: rotate the ground truth 3D bounding box (and the points in the frustum point cloud which lie within the box) around the camera coordinate Y-axis by an angle sampled from Uniform$[-\frac{\pi}{10}, \frac{\pi}{10}]$.

Training

We use the Adam optimizer, as described in Section 3.2.3, with a starting learning rate of 0.001 and a batch size of 32 frustum point clouds. The learning rate is halved every 100 epochs. The loss is computed on the entire validation set after each epoch, and the training is stopped when this validation loss stops to decrease.

3.7 3D Object Detection - Extended Frustum-PointNet

As having access to both LiDAR and image information intuitively should allow for improved 3D object detection performance, the Frustum-PointNet architecture was extended to also utilize image features. The implemented extension is in this section described in detail.

3.7.1 Model Architecture

Inspired by the image-only 3DOD architecture presented by Mousavian *et al.* in [43], we decided to utilize image features only for the estimation of the 3D bounding box size (h, w, l) and yaw angle θ.

The model architecture is thus very similar to that of the original Frustum-PointNet illustrated in Figure 3.8: the model input, the frustum point cloud extraction stage, *InstanceSeg-Net* and *T-Net* are all left unmodified. The only difference is in *Bbox-Net* where the global LiDAR feature vector $f \in \mathbb{R}^{512}$ is fused with an image feature vector $f_{img} \in \mathbb{R}^{512}$. The architecture of this modified *Bbox-Net* is schematically illustrated in Figure 3.13.

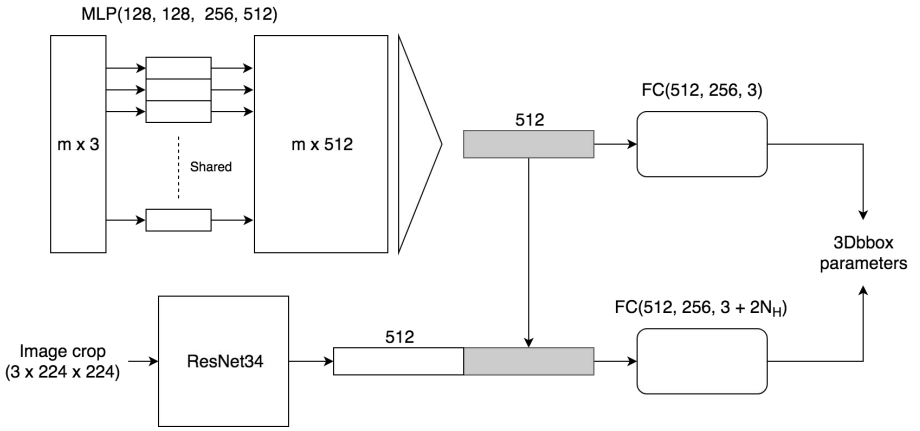

Figure 3.13: *Schematic illustration of the architecture for the extended Bbox-Net subnetwork.*

The LiDAR feature vector $f \in \mathbb{R}^{512}$ is fed to a fully-connected network with three layers to output regressed residuals for the 3D bounding box center position

(x, y, z), relative to the center position estimated by *T-Net*. It is also concatenated with $f_{img} \in \mathbb{R}^{512}$ and fed to a second fully-connected network with three layers to output the $3 + 2N_H$ values corresponding to the estimation of (h, w, l) and θ.

The image feature vector $f_{img} \in \mathbb{R}^{512}$ is obtained by cropping the image to the input 2D bounding box, resizing the image crop to 224×224 and feeding it to a ResNet [20] CNN with 34 layers.

3.7.2 Implementation Details

The extended Frustum-PointNet was also entirely implemented using PyTorch [46]. We utilized the ResNet-34 model found in the torchvision models subpackage and used the pre-trained version for initializing its weights. The pre-trained model expects inputs to be normalized, and the resized image crop was thus normalized with the mean vector $[0.485, 0.456, 0.406]^T$ and standard deviation $[0.229, 0.224, 0.225]^T$. To extract $f_{img} \in \mathbb{R}^{512}$ using ResNet-34, its classification head was removed.

Otherwise, the implementation details are basically identical to those of Frustum-PointNet in Section 3.6.2 in terms of both loss function, data augmentation and training. The only difference is that we do not augment the ground truth yaw angles, as there is no simple method to modify the input image crop accordingly.

3.8 3D Object Detection - Image-Only Model

To more conveniently be able to apply our domain adaptation techniques to the task of 3D object detection, we decided to also implement a simple image-only 3DOD model. The implemented architecture is in this section described in detail, together with a description of how the model was trained.

3.8.1 Model Architecture

The image-only model takes a mono-camera image and a 2D bounding box containing an object as input, and outputs an estimated 3D bounding box for the object. The architecture is inspired by the work of Mousavian *et al.* in [43] and Chabot *et al.* in [7], and utilizes both a deep network and geometric optimization. A schematic overview of the architecture is shown in Figure 3.14.

The input image is cropped to the 2D bounding box, resized to 224×224 and fed to a ResNet-34 [20] CNN that extracts a feature vector $f_{img} \in \mathbb{R}^{512}$. This feature vector is then fed to a fully-connected network with three layers to output $16 + 3 + 1$ values. ReLU is applied to the output of the first two layers but not the output layer.

The first 16 values are regressed residuals for the (u, v) pixel coordinates obtained by projecting the eight 3D bounding box corners onto the image plane. The residuals are relative to the 2D bounding box center pixel coordinates, and normalized by the 2D box height and width. The next 3 values are regressed residuals for the 3D box size dimensions (h, w, l), relative to the mean ground

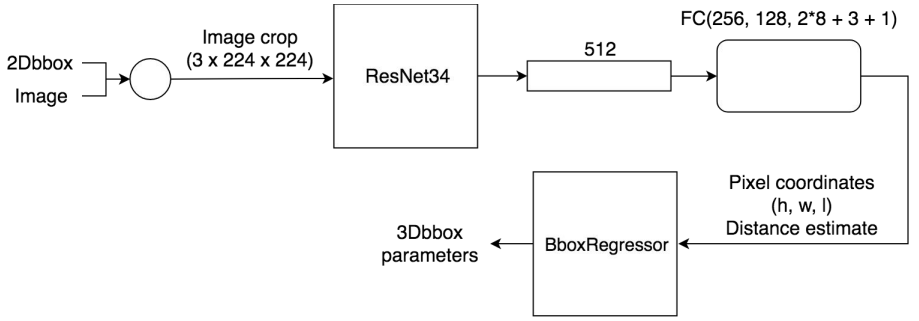

Figure 3.14: *Schematic overview of the image-only architecture for 3D object detection.*

truth box dimensions in the training set. The last value is a regressed residual for the distance from the camera to the 3D box center, relative to the mean ground truth box distance in the training set.

Given the regressed 3D box size, distance and corner pixel coordinates, the full set of 3D box parameters $(x, y, z, h, w, l, \theta)$ is geometrically estimated in *BboxRegressor* by minimizing the difference between the regressed pixel coordinates and the ones obtained by projecting the estimated 3D box onto the image plane. In this minimization, the regressed 3D box size and distance are used for both initialization and regularization.

3.8.2 Implementation Details

The image-only architecture was also entirely implemented in Python using PyTorch [46]. We used the same ResNet-34 model as in the extended Frustum-PointNet architecture, and the image crops were thus normalized as described in Section 3.7.2.

The network was trained using the Adam optimizer with the same batch size and learning rate scheme as for Frustum-PointNet, see Section 3.6.2.

BboxRegressor

An estimate of the 3D bounding box is obtained by setting $(x, y, z, h, w, l, \theta)$ as optimization variables and minimizing the objective function:

$$f(x, y, z, h, w, l, \theta) = f_p(x, y, z, h, w, l, \theta) + \alpha f_d(x, y, z) + \beta f_s(h, w, l) \tag{3.14}$$

f_p computes pixel coordinates for the 3D box corners and penalizes their difference with the regressed ones, f_d computes 3D box distance and penalizes the difference with the regressed one, and f_s penalizes difference with the regressed (h, w, l). We set $\alpha = 10$, $\beta = 100$, reflecting the fact that the regressed size estimate is believed to be more accurate than the distance estimate.

The nonlinear least-squares problem posed by (3.14) is solved utilizing the *least_squares* method of the optimization module in SciPy [27], which implements the Trust Region Reflective algorithm [5].

Loss Function

The ResNet CNN and the fully-connected network are jointly optimized using a multi-task loss function:

$$\mathcal{L} = \mathcal{L}_{pixel} + \lambda \mathcal{L}_{size} + \gamma \mathcal{L}_{dist} \tag{3.15}$$

$\mathcal{L}_{pixel}$ is for the pixel coordinates regression, $\mathcal{L}_{size}$ is for the 3D box size regression and $\mathcal{L}_{dist}$ for the distance regression. The Huber loss, as defined in Section 3.6.2, is used for all three task losses. In order to balance the task losses, we set $\lambda = 10, \gamma = 0.01$.

Data Augmentation

Ground truth 2D bounding boxes are taken as input and are augmented by random translation and scaling before the image crop. The 2D box center is translated by distances sampled from Uniform$[-0.1w, 0.1w]$ and Uniform$[-0.1h, 0.1h]$, where (w, h) is the original 2D box width and height. The width and height are then also independently rescaled by a factor sampled from Uniform$[0.9, 1.1]$. The 224×224 image crop is finally flipped with 0.5 probability.

3.9 Domain Adaptation - SYN and KITTI

As a final step we evaluated an adapted CycleGAN solution with an additional task network to perform simultaneous domain adaptation through image-to-image translation with CycleGAN and training of the 3D object detection task network on translated images from the source domain using their 3D ground truth annotations. The task network is finally evaluated on the target domain and compared to an identical model that has been trained solely on the source domain.

Images from both domains are scaled to 128×128 before being fed as inputs to the generator and discriminator. We adapt the training strategy of PixelDA and train the object detection network on both images translated from source to target domain and on images sampled from the source domain. Since domain adaptation is computationally costly and forces us to use small batches, training the object detection model on both samples from the source domain and translated images will help stabilize training and further regularize the generator. In addition to the loss functions listed in section 3.2.2, as a task loss for the image-only model we used the loss function listed in section 3.8.2. For the Extended Frustum Pointnet we use the loss function described in section 3.6.2.

3.9.1 Model Architecture

We use the image-only 3D object detection model described in section 3.8 and the Extended Frustum-PointNet presented in section 3.7 as task network for this problem. Crops of objects in each frame are first extracted by their 2D bounding box coordinates and scaled to a resolution of 128×128, as well as normalized to the range $[-1, 1]$. Crops from both domains are then fed through generator networks and translated to the opposite domain.

The translated images are further fed to the domain discriminators which determines whether these images are sampled from their respective domain or if they are translated images from the opposite domain.

Translated images from the source domain are fed together with the 3D bounding box coordinates, the size of the box and the distance to the box to the image-only 3D object detection model which trains a regression model based on the image and its annotations. We use as our regression loss a mean absolute error, and the gradient of this loss is propagated back through both the 3D object detection network and the source-to-target domain translation generator for weight updates. Images are then translated back to their original domains where the cycle-consistency loss is determined. We use the Adam optimizer described in section 3.2.3 to update the weights of the networks in this architecture.

4

Results

In this chapter the results of the different experiments and evaluations are presented. For most experiments both quantitative results as well as qualitative results are presented. However, in the case of performing domain adaptation between the GTA 5 dataset and the Cityscapes dataset, there are no quantitative results available, as a task network was not applied during this experiment.

4.1 Domain Adaptation - MNIST to MNIST-M

This section presents the quantitative and qualitative results of training a classification network during domain adaptation using the adversarial PixelDA framework presented by Bousmalis *et al.* in their paper *Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks* [4]. The aim of this experiment was to demonstrate that it was possible to improve the performance of a classifier on a dataset without access to annotations by training it on translated images from a separate dataset where annotations are available. To validate this idea we measured the classification accuracy of a classification network that was trained during domain adaptation and compared it to the classification network of a second identical classifier that was trained on untranslated images.

4.1.1 Quantitative Results

The classification network was trained for 200 epochs with a batch size of 32 in parallel with the unsupervised domain adaption. The classification network receives translated images from the MNIST dataset which have been translated to resemble the images in the MNIST-M dataset. Since all networks are trained from scratch the quality of these translated images will not be of high quality in the first couple of epochs. The generators, the discriminator and the classification

network was saved with a 5 epoch interval, and the final model was selected as the one that achieved the lowest validation error on an MNIST-M validation set.

The final model was further evaluated against the MNIST-M test set, and compared against a reference model. The reference model is a classification network with an identical network architecture to the classification network trained during domain adaptation, but trained on images directly from the MNIST dataset, and then tested on the MNIST-M dataset. We experimented with using both the U-Net architecture and the residual network architecture as representations for our generators, and found that the results differed by a surprising amount. The results of both the classification network trained during domain adaptation and the reference model can be seen in Table 4.1.

Table 4.1: *Measured classification accuracy of evaluated models on the MNIST-M dataset.*

Model	MNIST-M Classification Accuracy
Reference	55 %
PixelDA (U-Net)	91 %
PixelDA (ResNet)	95 %

The results of this experiment was encouraging and proved that given a suitable problem, domain adaptation can give a big boost to the performance of an image recognition solution where access to annotated data for an intended target domain is hard to come by, but where there exists a similar dataset with structural similarities to the target dataset and where annotations are available.

4.1.2 Qualitative Results

While training the PixelDA model translated images from the MNIST dataset are sampled with an interval of 400 batches. An example can be seen in Figure 4.1, which contains both translated digits from the MNIST domain and the images that the generator was conditioned on. Note that the fact that the background of these translated MNIST images do not share the same background pattern is a sign that the generators do not suffer from mode-collapse, which is a common problem when training GANs where the generators resort to producing the same kind of output every time, as they have found a pattern that manages to fool the discriminator. One technique that helped reduce this was adding dropout layers with a dropout rate set to 0.5 after several of the generator's layers. The fact that the task network achieved a 95 % classification accuracy on unseen images from the target domain was another strong indicator that the generators were able to closely model the target domain.

Figure 4.1: *Samples produced by the PixelDA generator during training. The top row shows images sampled from the MNIST domain. The second row displays the translated representations of these images produced by the generator. The third row displays examples of images from the MNIST-M domain.*

4.2 Translating GTA 5 to Cityscapes

This section presents the qualitative results of performing unsupervised domain adaptation between the synthetic GTA 5 dataset and the Cityscapes dataset. The purpose of this experiment was to evaluate what type of quality that could be expected for a domain adaptation problem similar to the main objective of the thesis by performing unsupervised domain adaptation with a CycleGAN. Because of time constraints we did not have time to carry out user studies for evaluating the quality of generated samples.

4.2.1 Qualitative Results

We used the same hyper parameter settings as in the paper in which CycleGAN was introduced, and adopted the techniques aimed at stabilizing the training presented in section 3.2.3. Samples produced in the early stages of training were encouraging as the generated images had preserved structures and semantics in their original forms.

Images from the GTA 5 dataset are significantly more vibrant than the ones in Cityscapes, and by examining images sampled from the generators it was apparent that the generated images were more in line with the lighting, contrast and color composition of the images in the target domain. Three examples of where the translation from GTA 5 to Cityscapes was deemed successful can be seen in

Figure 4.2. Examples of successful domain translations from Cityscapes to GTA 5 can be seen in Figure 4.4. As the networks kept training however, it was apparent that the generators started to pick up on additional peculiarities in the respective datasets. For example, in the Cityscapes dataset most of the images are collected from a dash camera on a Mercedes. This Mercedes has an ornament on the hood of the car, and as a result this ornament is present in essentially every image in the Cityscapes dataset.

As the CycleGAN reaches the later stages of training the domain discriminator of the Cityscapes domain have been presented with enough samples from the Cityscapes dataset to pick up on the fact that the presence of this hood ornament indicates a high probability of that the image have been sampled from the Cityscapes domain. And in turn the generator tasked with translating images from the GTA 5 domain to the Cityscapes domain starts to learn that by adding this hood ornament to images it has a higher probability of fooling the discriminator. As a result of this the Mercedes hood ornament becomes common in images that are produced by this generator. Cycle-consistency should help restrict the generators from making these semantic changes during translation. In this case however the Mercedes ornament never occurs in the GTA 5 domain, and as a result the generator translating images from the Cityscapes domain to the GTA 5 domain learns to remove the ornament during translation. Because of this the generator hallucinating these hood ornaments in its effort to produce images resembling those from Cityscapes is not penalized from adding these artifacts, as this fools the domain discriminator of the Cityscapes domain and the generator reconstructing the original GTA 5 image learns to remove the ornament artifacts during retranslation.

This ornament can be seen in the second column of the examples of unsuccessful domain adaptations in Figure 4.3. Examples of unsuccessful translations from the Cityscapes domain to the GTA 5 domain can be seen in Figure 4.5. Using a multi-scale discriminator, where the discriminator outputs probabilities of the validity of samples at different image scales has previously been shown to improve the visual quality and decrease the amount of artifacts in generated images. However, in our experiments we did not notice an obvious difference in the image quality of generated samples and the amount of artifacts present in these samples.

4.3 3D Object Detection - Evaluation Method

To evaluate the implemented 3D object detections models, we use 3D average precision (*3D AP*) as our main performance metric. *3D AP* is computed as the area under the precision-recall curve, where a predicted 3D bounding box is considered a true positive if its 3D IoU with the ground truth box exceeds a certain threshold. For cars, this threshold is set to 70 % in the official KITTI 3DOD benchmark [14], which is thus the default value used in this chapter. Since $\geq$ 70 % 3D IoU is a fairly strict requirement, a model can obtain a low score in terms of this metric while qualitatively still being relatively close to the ground truth, we will

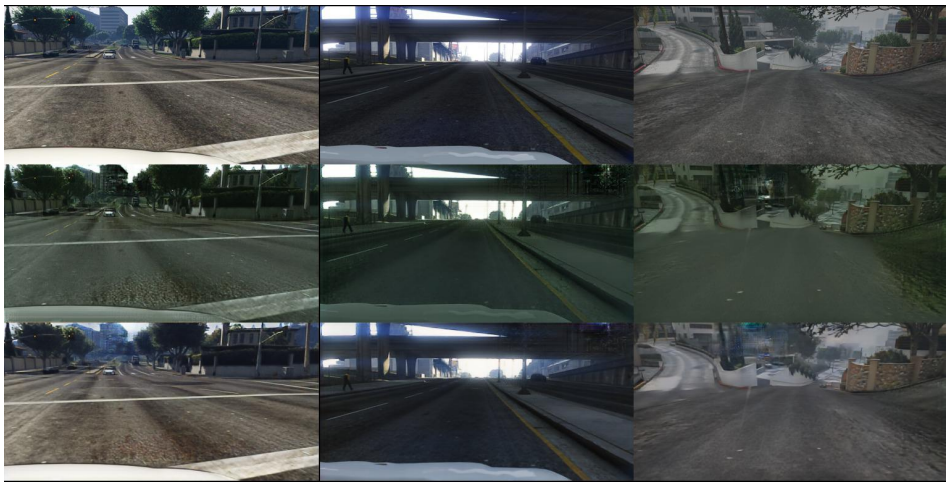

Figure 4.2: *Examples of successful domain translation from GTA 5 to Cityscapes. The top row shows images sampled from the GTA 5 domain. The second row displays the translated representations of these images produced by the generator. The third row displays the reconstructed image.*

Figure 4.3: *Examples of unsuccessful domain translation from GTA 5 to Cityscapes. The top row shows images sampled from the GTA 5 domain. The second row displays the translated representations of these images (where severe artifacts are present) produced by the generator. The third row displays the reconstructed image.*

Figure 4.4: *Examples of successful domain translation from Cityscapes to GTA 5. The top row shows images sampled from the Cityscapes domain. The second row displays the translated representations of these images produced by the generator. The third row displays the reconstructed image.*

Figure 4.5: *Examples of unsuccessful domain translation from Cityscapes to GTA 5. The top row shows images sampled from the Cityscapes domain. The second row displays the translated representations of these images (where severe artifacts are present) produced by the generator. The third row displays the reconstructed image.*

however also use a 50 % threshold to compare all our implemented models.

We also use top-view average precision (*Top-view AP*) as a secondary performance metric. *Top-view AP* is again computed as the area under the precision-recall curve, but in this case the IoU of the predicted 3D bounding box with the ground truth box is computed in top-view (bird's eye view), meaning that the box height and its location along the height axis are ignored.

Both metrics can be computed on *KITTI test* by submitting predicted 3D bounding boxes to the KITTI 3DOD benchmark server [14], which evaluates the metrics for three different object difficulty levels: easy, moderate and hard. A ground truth object's difficulty is determined by three in-image attributes, as specified in Table 4.2. Performance for the moderate level is used as the main metric.

Table 4.2: *Object attribute specifications for the three difficulty levels in the KITTI 3D object detection benchmark.*

Attribute	**Easy**	**Moderate**	**Hard**
Min. 2D box height	40 pixels	25 pixels	25 pixels
Max. occlusion level	Fully visible	Partly occluded	Difficult to see
Max. truncation	15 %	30 %	50 %

KITTI also provides a development kit containing C++ code for computing both *3D AP* and *Top-view AP*, for the three difficulty levels, on any set of images from the training set. This is used to evaluate the implemented models' performance on *KITTI val*.

When evaluating a model on *KITTI val*, we take 2D detections[1] provided by the Frustum-PointNet [47] authors as input 2D bounding boxes. For *KITTI test*, we instead take 2D detections from a simple internal Zenuity model as input. The 2D detection performance on *KITTI val* for both models is reported in Table 4.3. In both cases, the 2D detection confidence score is used also as the 3D detection confidence score.

Table 4.3: *2D detection performance for cars on KITTI val, as measured by 2D AP, for the 2D detections which are taken as input 2D bounding boxes when evaluating a 3DOD model.*

2D detector	**Easy**	**Moderate**	**Hard**
Frustum-PointNet (used on *KITTI val*)	96.48 %	90.31 %	87.63 %
Zenuity (used on *KITTI test*)	87.8 %	77.4 %	68.1 %

[1] `https://github.com/charlesq34/frustum-pointnets`

4.4 3D Object Detection - Frustum-PointNet

Our Frustum-PointNet implementation, as described in Section 3.6, was evaluated on *KITTI val* after being independently trained on both *KITTI train* and *SYN train*. Additionally, it was also trained on *KITTI train random* and evaluated on *KITTI test*. Quantitative results for all of these experiments are in this section presented, together with qualitative results on *KITTI test* in the form of images.

4.4.1 Quantitative Results

Performance on *KITTI val* in terms of *3D AP* and *Top-view AP* is presented in Table 4.4 and Table 4.5, respectively. The tables include performance with a 70 % threshold for both versions of the original Frustum-PointNet implementation, together with performance with both a 70 % and 50 % threshold for our implementation trained on both *KITTI train* and *SYN train*.

***Table 4.4: KITTI val, 3D AP**. Performance on KITTI val, in terms of the 3D AP metric, for the original and our Frustum-PointNet implementation.*

Method	Easy	Moderate	Hard
Frustum-PointNet (version 1) [47]	83.26 %	69.28 %	62.56 %
Frustum-PointNet (version 2) [47]	83.76 %	70.92 %	63.65 %
Our Frustum-PointNet - KITTI	78.01 %	65.22 %	59.06 %
Our Frustum-PointNet - KITTI (50 %)	93.73 %	87.96 %	79.04 %
Our Frustum-PointNet - SYN	10.23 %	7.96 %	7.23 %
Our Frustum-PointNet - SYN (50 %)	69.13 %	63.46 %	56.41 %

***Table 4.5: KITTI val, Top-view AP**. Performance on KITTI val, in terms of the Top-view AP metric, for the original and our Frustum-PointNet implementation.*

Method	Easy	Moderate	Hard
Frustum-PointNet (version 1) [47]	87.82 %	82.44 %	74.77 %
Frustum-PointNet (version 2) [47]	88.16 %	84.02 %	76.44 %
Our Frustum-PointNet - KITTI	85.30 %	79.89 %	72.38 %
Our Frustum-PointNet - KITTI (50 %)	94.13 %	88.50 %	85.65 %
Our Frustum-PointNet - SYN	30.43 %	28.54 %	23.43 %
Our Frustum-PointNet - SYN (50 %)	76.63 %	70.68 %	61.88 %

In Table 4.4 and Table 4.5, one can observe that our Frustum-PointNet implementation trained on *KITTI train* comes close to the original one in terms of both *3D AP* and *Top-view AP* performance. Further, our implementation trained on *SYN train* lags significantly when evaluated with a 70 % threshold, but is relatively close with the less strict 50 % threshold.

Performance on *KITTI test* in terms of *3D AP* and *Top-view AP* is presented in Table 4.6 and Table 4.7, respectively. The tables include both the original and our Frustum-PointNet implementation, together with a number of other SOTA models which all utilize LiDAR.

***Table 4.6: KITTI test, 3D AP**. Performance on KITTI test, in terms of the 3D AP metric, for our Frustum-PointNet implementation and a number of SOTA models.*

Method	Easy	Moderate	Hard
Frustum-PointNet (version 1) [47]	80.62 %	64.70 %	56.07 %
Frustum-PointNet (version 2) [47]	81.20 %	70.39 %	62.19 %
AVOD-FPN [30]	**81.94 %**	**71.88 %**	**66.38 %**
AVOD [30]	73.59 %	65.78 %	58.38 %
VoxelNet [65]	77.47 %	65.11 %	57.73 %
AVOD-SSD [30]	73.64 %	63.87 %	56.90 %
MV3D [10]	71.09 %	62.35 %	55.12 %
MV3D (LiDAR) [10]	66.77 %	52.73 %	51.31 %
F-PC_CNN [11]	60.06 %	48.07 %	45.22 %
Our Frustum-PointNet	59.35 %	56.81 %	50.51 %

***Table 4.7: KITTI test, Top-view AP**. Performance on KITTI test, in terms of the Top-view AP metric, for our Frustum-PointNet implementation and a number of SOTA models.*

Method	Easy	Moderate	Hard
Frustum-PointNet (version 1) [47]	87.28 %	77.09 %	67.90 %
Frustum-PointNet (version 2) [47]	88.70 %	84.00 %	75.33 %
AVOD-FPN [30]	88.53 %	83.79 %	**77.90 %**
AVOD [30]	86.80 %	**85.44 %**	77.73 %
VoxelNet [65]	**89.35 %**	79.26 %	77.39 %
AVOD-SSD [30]	86.14 %	77.66 %	75.68 %
MV3D [10]	86.02 %	76.90 %	68.49 %
MV3D (LiDAR) [10]	85.82 %	77.00 %	68.94 %
F-PC_CNN [11]	83.77 %	75.26 %	70.17 %
Our Frustum-PointNet	76.65 %	72.86 %	64.51 %

In Table 4.6 and Table 4.7, one can observe that the performance gap between our and the original Frustum-PointNet implementation is significantly bigger on *KITTI test*, likely due to the difference in input 2D detections. Also, the relative performance of our implementation compared to other models is somewhat worse in terms of *Top-view AP* than *3D AP*.

4.4.2 Qualitative Results

Figure 4.6 visualizes the result of running our Frustum-PointNet implementation, trained on *KITTI train random*, on example 5 in *KITTI test*. From top to bottom, Figure 4.6 shows the input 2D bounding boxes, the predicted 3D bounding boxes visualized in the image plane and the predicted 3D boxes visualized in the LiDAR point cloud. Red color marks the predicted front of the 3D box. Figure 4.7 shows the same visualizations but for example 22 in *KITTI test*.

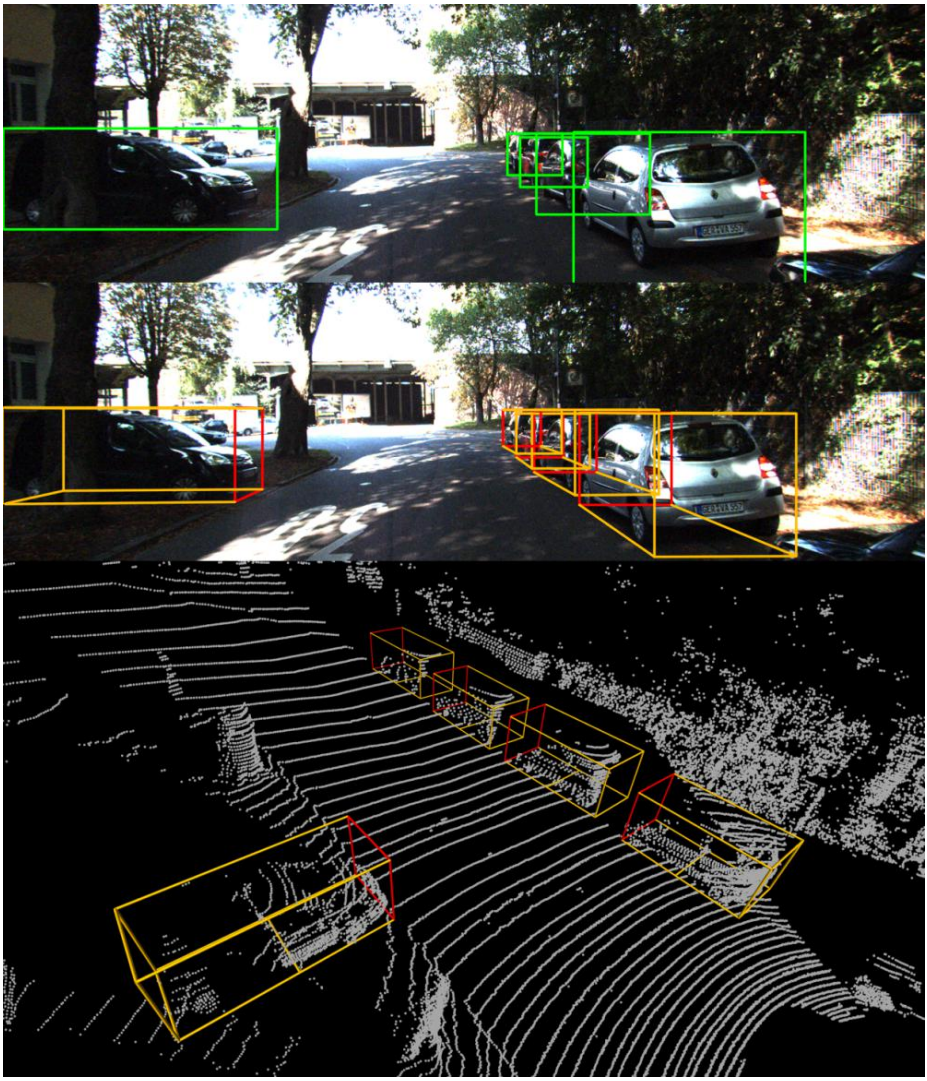

Figure 4.6: *Visualization of our Frustum-PointNet implementation, trained on KITTI train random, applied on example 5 in KITTI test.*

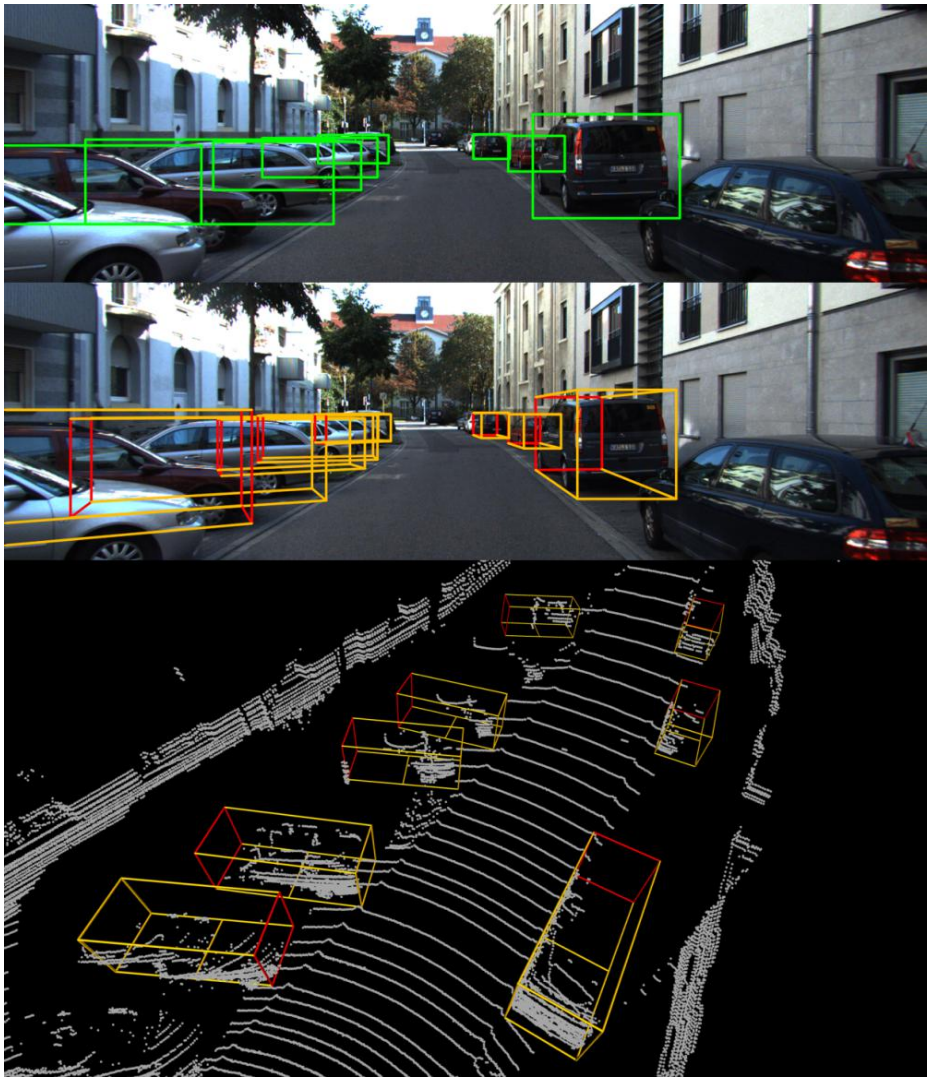

Figure 4.7: *Visualization of our Frustum-PointNet implementation, trained on KITTI train random, applied on example 22 in KITTI test.*

In Figure 4.6 and Figure 4.7, one can observe that the model usually is able to accurately predict the 3D location, size and yaw angle of the detected vehicles. The performance is however somewhat worse for the vehicles on the left side of the road in Figure 4.7, which are all heavily occluded and parked in a relatively uncommon angle.

Further qualitative results for the model trained on *KITTI train random* can be

found online[2], including video visualizations on several sequences in *KITTI test*.

Figure 4.8 instead visualizes the result of training our Frustum-PointNet implementation on *SYN train*, and running this on example 5 and 22 in *KITTI test*. For example 5, the predicted 3D boxes seem to be somewhat too large but overall relatively accurate. For example 22 however, the predictions for the vehicles on the left side of the road are completely off.

Further qualitative results, in the form of video visualizations on sequences from *KITTI test*, can for the model trained on *SYN train* be found online[3].

4.5 3D Object Detection - Extended Frustum-PointNet

The extended Frustum-PointNet implementation, as described in Section 3.7, was evaluated on both *KITTI val*, after being independently trained on *KITTI train* and *SYN train*, and *KITTI test*. In this section, we present quantitative results for all of these experiments, as well as some qualitative results on *KITTI test*.

4.5.1 Quantitative Results

Performance on KITTI val in terms of *3D AP* and *Top-view AP* is presented in Table 4.8 and Table 4.9, respectively. The tables include performance with both a 70 % and 50 % threshold. For easy comparison, the tables also include our Frustum-PointNet implementation. One can observe that the extended Frustum-PointNet implementation actually has worse performance metrics across the board.

***Table 4.8: KITTI val, 3D AP**. Performance on KITTI val, in terms of the 3D AP metric, for our Frustum-PointNet and extended Frustum-PointNet implementation.*

Method	Easy	Moderate	Hard
Our Frustum-PointNet - KITTI	78.01 %	65.22 %	59.06 %
Our Frustum-PointNet - KITTI (50 %)	93.73 %	87.96 %	79.04 %
Our Frustum-PointNet - SYN	10.23 %	7.96 %	7.23 %
Our Frustum-PointNet - SYN (50 %)	69.13 %	63.46 %	56.41 %
Our Extended - KITTI	68.72 %	57.21 %	50.57 %
Our Extended - KITTI (50 %)	88.90 %	87.39 %	78.53 %
Our Extended - SYN	5.67 %	4.72 %	3.90 %
Our Extended - SYN (50 %)	61.07 %	56.95 %	49.93 %

Performance on *KITTI test* in terms of *3D AP* and *Top-view AP* is presented in Table 4.10 and Table 4.11, respectively. For easy comparison, the tables also include the performance of our Frustum-PointNet implementation. Also in this case one can observe that there is a clear performance gap in favor of Frustum-PointNet.

[2] https://photos.app.goo.gl/QRgbU2r8JB5XABpw1

[3] https://photos.app.goo.gl/omucsrxlfhKsJNSG3

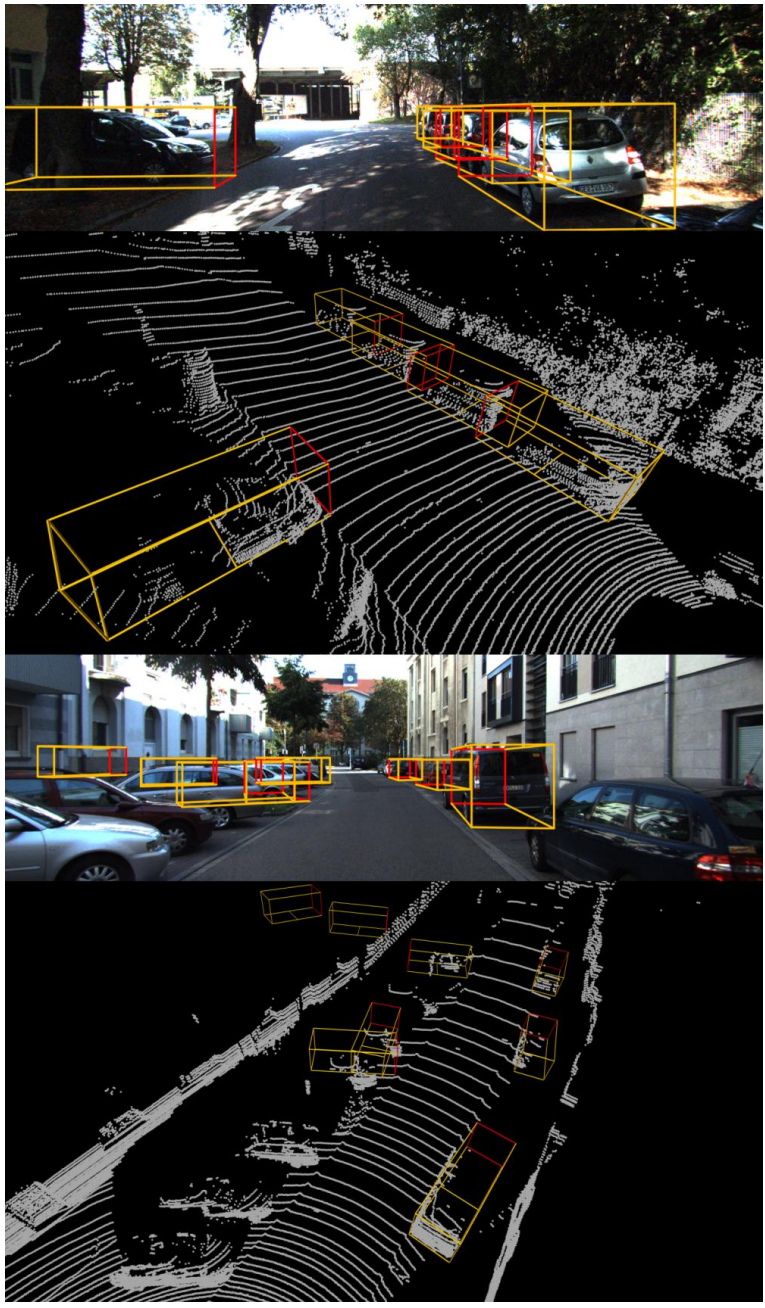

Figure 4.8: *Visualization of our Frustum-PointNet implementation, trained on SYN train, applied on example 5 (top) and 22 in KITTI test.*

***Table 4.9: KITTI val, Top-view AP**. Performance on KITTI val, in terms of the Top-view AP metric, for our Frustum-PointNet and extended Frustum-PointNet implementation.*

Method	Easy	Moderate	Hard
Our Frustum-PointNet - KITTI	85.30 %	79.89 %	72.38 %
Our Frustum-PointNet - KITTI (50 %)	94.13 %	88.50 %	85.65 %
Our Frustum-PointNet - SYN	30.43 %	28.54 %	23.43 %
Our Frustum-PointNet - SYN (50 %)	76.63 %	70.68 %	61.88 %
Our Extended - KITTI	81.18 %	72.78 %	64.47 %
Our Extended - KITTI (50 %)	89.27 %	88.25 %	85.36 %
Our Extended - SYN	17.27 %	17.19 %	14.62 %
Our Extended - SYN (50 %)	72.65 %	69.16 %	60.63 %

***Table 4.10: KITTI test, 3D AP**. Performance on KITTI test, in terms of the 3D AP metric, for our Frustum-PointNet and extended Frustum-PointNet implementation.*

Method	Easy	Moderate	Hard
Our Frustum-PointNet	59.35 %	56.81 %	50.51 %
Our Extended	54.05 %	48.13 %	42.27 %

4.5.2 Qualitative Results

Figure 4.9 visualizes the result of training our extended Frustum-PointNet implementation on *KITTI train random*, and running this on example 5 and 22 in *KITTI test*. If one compares this with Figure 4.6 and Figure 4.7, the extended Frustum-PointNet model seems to output virtually identical 3D boxes on example 5, but less accurately predict the 3D box yaw angles for the vehicles on the left side of the road in example 22. Video visualizations on several sequences from *KITTI test* can be found online[4].

4.6 3D Object Detection - Image-Only Model

Our image-only model, as described in Section 3.8, was evaluated on both *KITTI val*, after being independently trained on *KITTI train* and *SYN train*, and *KITTI test*. In this section, we present quantitative results for all of these experiments, as well as some qualitative results on *KITTI test*.

4.6.1 Quantitative Results

Performance on KITTI val in terms of *3D AP* and *Top-view AP* for our image-only model is presented in Table 4.12 and Table 4.13, respectively. The tables

[4] `https://photos.app.goo.gl/daH5v9jkG5zUmKfL2`

***Table 4.11: KITTI test, Top-view AP**. Performance on KITTI test, in terms of the Top-view AP metric, for our Frustum-PointNet and extended Frustum-PointNet implementation.*

Method	Easy	Moderate	Hard
Our Frustum-PointNet	76.65 %	72.86 %	64.51 %
Our Extended	71.91 %	69.70 %	55.91 %

include performance with both a 70 % and 50 % threshold. The tables also include previous works on image-only 3DOD by Chen *et al*. Compared to Table 4.4 and Table 4.5, one can observe that not utilizing LiDAR results in a significant drop in performance, especially for the 70 % threshold. It also becomes apparent that an image-only model trained on SYN is more or less completely incapable of generalizing to KITTI.

***Table 4.12: KITTI val, 3D AP**. Performance on KITTI val, in terms of the 3D AP metric, for our image-only model and previous works.*

Method	Easy	Moderate	Hard
Mono3D [9]	2.53 %	2.31 %	2.31 %
3DOP [8]	6.55 %	5.07 %	4.10 %
Our Image-Only - KITTI	10.13 %	8.32 %	8.20 %
Our Image-Only - KITTI (50 %)	40.31 %	30.77 %	26.55 %
Our Image-Only - SYN	0.06997 %	0.05798 %	0.05798 %
Our Image-Only - SYN (50 %)	0.3683 %	0.3422 %	0.3422 %

***Table 4.13: KITTI val, Top-view AP**. Performance on KITTI val, in terms of the Top-view AP metric, for our image-only model and previous works.*

Method	Easy	Moderate	Hard
Mono3D [9]	5.22 %	5.19 %	4.13 %
3DOP [8]	12.63 %	9.49 %	7.59 %
Our Image-Only - KITTI	15.64 %	12.90 %	12.30 %
Our Image-Only - KITTI (50 %)	45.46 %	33.83 %	31.79 %
Our Image-Only - SYN	0.1094 %	0.09735 %	0.09735 %
Our Image-Only - SYN (50 %)	0.4952 %	0.4901 %	0.4999 %

Performance on *KITTI test* in terms of *3D AP* and *Top-view AP* is presented in Table 4.14 and Table 4.15, respectively. The tables also include the only published image-only model on the KITTI 3DOD leaderboard [14]. One can observe that our model compares relatively well to the current SOTA, as there is currently no published image-only model with better reported performance. There are however unpublished image-only entries on the leaderboard which do outperform our model quite significantly.

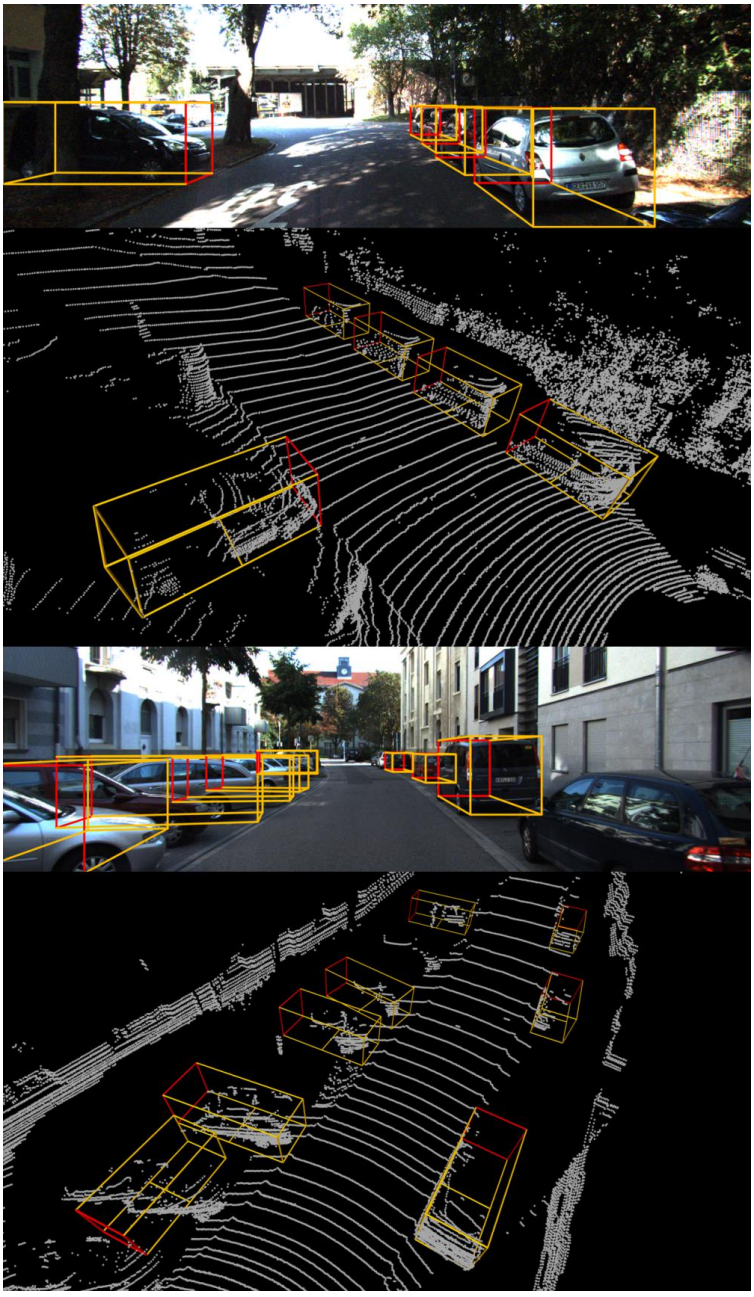

Figure 4.9: *Visualization of the extended Frustum-PointNet implementation, trained on KITTI train random, applied on example 5 (top) and 22 in KITTI test.*

***Table 4.14: KITTI test, 3D AP**. Performance on KITTI test, in terms of the 3D AP metric, for our image-only model and the only published image-only model on the KITTI 3DOD leaderboard.*

Method	Easy	Moderate	Hard
3D-SSMFCNN [44]	2.28 %	2.39 %	1.52 %
Our Image-only	6.76 %	6.45 %	4.87 %

***Table 4.15: KITTI test, Top-view AP**. Performance on KITTI test, in terms of the Top-view AP metric, for our image-only model and the only published image-only model on the KITTI 3DOD leaderboard.*

Method	Easy	Moderate	Hard
3D-SSMFCNN [44]	3.19 %	3.66 %	3.45 %
Our Image-Only	10.61 %	10.21 %	8.64 %

4.6.2 Qualitative Results

Figure 4.10 visualizes the result of training our image-only model on *KITTI train random*, and running this on example 5 and 22 in *KITTI test*. Except for one 3D box which is somewhat off-center, our model seems to output accurate 3D boxes on example 5. For example 22 however, the model clearly struggles to accurately estimate both 3D location and yaw angle. Video visualizations on several sequences from *KITTI test* can be found online[5].

4.7 Domain Adaptation - SYN to KITTI

This section presents the results of training a 3D object detection model during unsupervised domain adaptation between Zenuity's SYN dataset and the KITTI dataset using our domain adaptation model. To evaluate the 3D object detection model we used metrics described in Section 4.3. Both quantitative and qualitative evaluations have been made to evaluate whether it is helpful to train our 3D object detection model on images translated from the source domain to the target domain to help improve the model's performance of performing 3D object detection on images from the target domain.

The task network's 3D object detection performance on images from the target domain is then compared to the baseline approach, which involves training an identical task network on images from the source domain. We experimented with both performing domain adaptation for the image-only 3D object detection model and for the extended Frustum-PointNet implementation.

[5] `https://photos.app.goo.gl/KJAprIrIEF6o3D0B3`

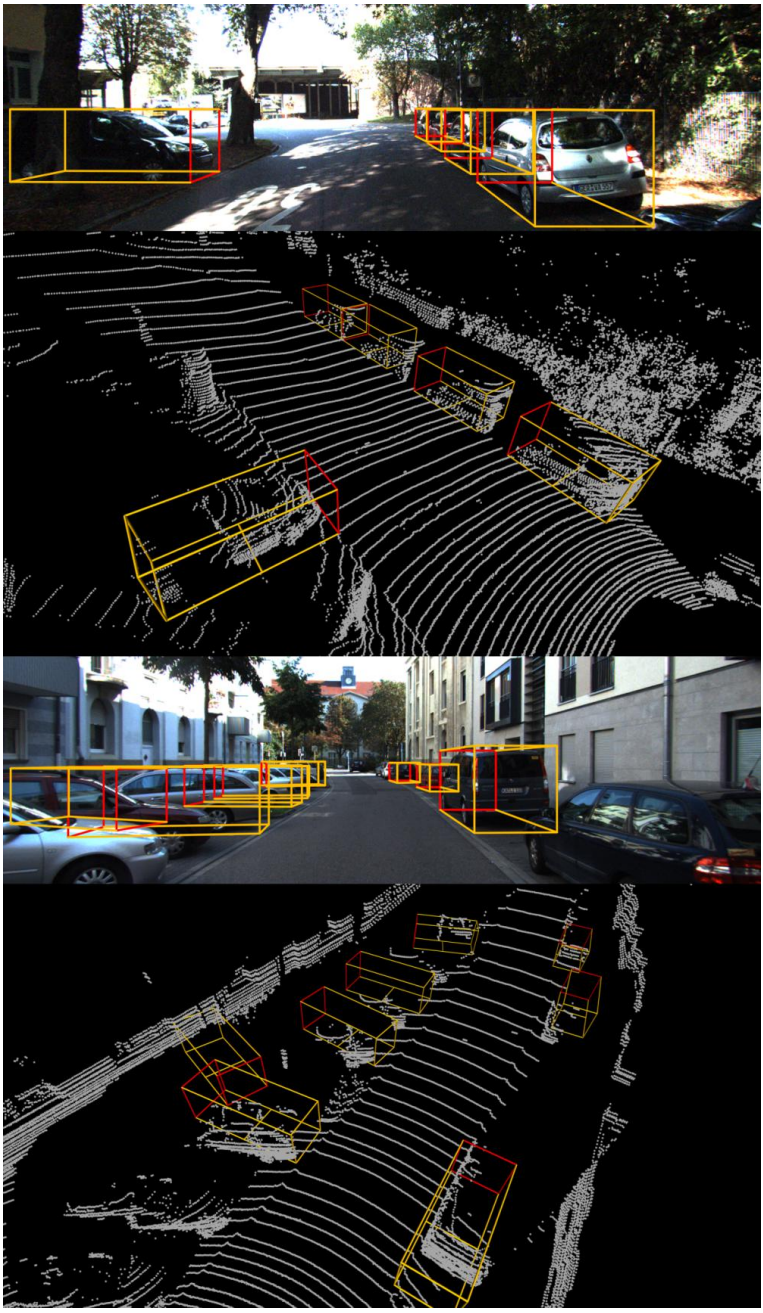

Figure 4.10: *Visualization of our image-only model, trained on KITTI train random, applied on example 5 (top) and 22 in KITTI test.*

4.7.1 Quantitative Results

After having trained the domain adaptation model for 200 epochs with a 5 epoch model checkpoint interval, we evaluate the performance of the task network on images from the target domain and compare it to the performance of an identical object detection model that was trained in parallel with the domain adaptation task network on images purely sampled from the source domain. The metrics used to evaluate and compare the two task networks are described in Section 4.3. For the image-only object detection model we did notice an increase in task performance by training the network during domain adaptation, but this was not the case with the Extended Frustum-PointNet.

Image-Only 3D Object Detection Task Network

Images from the source domain were translated to the target domain and fed as training data to the image-only 3D object detection model. After having been trained on translated samples from the SYN dataset the image-only 3D object detection model was evaluated on the KITTI validation set, and the 3D object detection performance can be seen in Table 4.16 and Table 4.17. For both the *3D AP* and *Top-view AP* metric we did see an improvement by training the model on translated images from SYN compared to images directly sampled from SYN. However, compared to an identical model trained on images directly sampled from the target domain the model did not perform close to on par.

***Table 4.16: KITTI val, 3D AP**. Performance on KITTI val, in terms of the 3D AP metric, for our image-only model and our image-only model trained during domain adaptation.*

Method	Easy	Moderate	Hard
Our Image-Only - SYN	0.06997 %	0.05798 %	0.05798 %
Our Image-Only - SYN (50 %)	0.3683 %	0.3422 %	0.3422 %
Our Image-Only - DA SYN	0.1007 %	0.1076 %	0.1076 %
Our Image-Only - DA SYN (50 %)	1.36 %	0.8004 %	0.8082 %

***Table 4.17: KITTI val, Top-view AP**. Performance on KITTI val, in terms of the Top-view AP metric, for our image-only model without domain adaptation and our image-only model trained during domain adaptation.*

Method	Easy	Moderate	Hard
Our Image-Only - SYN	0.1094 %	0.09735 %	0.09735 %
Our Image-Only - SYN (50 %)	0.4952 %	0.4901 %	0.4999 %
Our Image-Only - DA SYN	0.2690 %	0.2671 %	0.2692 %
Our Image-Only - DA SYN (50 %)	2.23 %	1.02 %	1.03 %

Extended Frustum-PointNet Task Network

We performed a similar domain adaptation as for the image-only 3D object detection model. Images were translated from the source domain to the target domain where they were used to train the extended Frustum-PointNet. The extended Frustum-PointNet trained during domain adaptation were further compared to an identical model trained on the source domain and evaluated on the target domain.

The 3D object detection performance after having been trained on translated images from the SYN dataset during domain adaptation when measured on the KITTI validation set can be seen in Table 4.18 and Table 4.19. The extended Frustum-PointNet trained during domain adaptation did not perform on par with a model trained directly on images sampled from the source domain. Domain adaptation did not result in better performance.

***Table 4.18: KITTI val, 3D AP**. Performance on KITTI val, in terms of the 3D AP metric, for our extended Frustum-PointNet implementation trained on SYN and trained during domain adaptation.*

Method	Easy	Moderate	Hard
Our Extended - SYN	5.67 %	4.72 %	3.90 %
Our Extended - SYN (50 %)	61.07 %	56.95 %	49.93 %
Our Extended - DA SYN	6.28 %	4.87 %	3.89 %
Our Extended - DA SYN (50 %)	55.26 %	53.06 %	44.82 %

***Table 4.19: KITTI val, Top-view AP**. Performance on KITTI val, in terms of the Top-view AP metric, for our extended Frustum-PointNet implementation trained on SYN and trained during domain adaptation.*

Method	Easy	Moderate	Hard
Our Extended - SYN	17.27 %	17.19 %	14.62 %
Our Extended - SYN (50 %)	72.65 %	69.16 %	60.63 %
Our Extended - DA SYN	18.13 %	16.88 %	13.97 %
Our Extended - DA SYN (50 %)	65.95 %	63.67 %	56.08 %

4.7.2 Qualitative Results

In the early stages of training the model performs well and the translation performance is gradually improved by inspecting translated image samples. Images in the KITTI domain are generally more vibrant and colorful than images in the SYN dataset, and the generator focuses on adapting image features such as the contrast of the image, lighting and colors.

We use the same hyper parameters as presented in the CycleGAN paper, in which the cycle-consistency loss is heavily weighted compared to the adversarial loss. It is important that images that are translated are structurally consistent

with the original images, and this weight will help constrain the mapping from introducing such changes.

By training on both adapted images and on images directly sampled from the source domain, the generator is further restricted from making significant changes to the the source domain images, since this would negatively affect the task network, which is also trained on images from the source domain. After a certain amount of epochs however, the model starts to generate artifacts in the translated images. Generated samples can be seen in Figure 4.11.

Figure 4.11: *Figure showing images sampled during domain adaptation of SYN to KITTI. The first three rows show images from SYN and the last three rows show images sampled from KITTI. The first row from each respective dataset shows original images sampled from each dataset and the second row shows the same image after translation. The third row from each dataset shows the images after reconstruction.*

5
Discussion

In this chapter we perform a detailed discussion of the methods and results previously presented. Additionally, the chapter ends with a discussion of possible areas of future work.

5.1 Domain Adaptation - MNIST to MNIST-M

The results of the PixelDA approach resulted in a significant performance increase in classifying digits from the MNIST-M dataset. Both quantitative and qualitative results were overall quite impressive. The background of images in MNIST-M are produced randomly, and the images that were generated when translating images from MNIST to MNIST-M had high variance in the type of background that was generated, indicating that the model had not simply found a one-to-one mapping from the source to the target domain.

In the paper the authors suggest that the task network should be trained both on images sampled from the source domain and images translated to the target domain. While this should reasonably stabilize the training of the task network, and possibly also help regularize the model as it sees data of higher visible variance, we did not observe a noticeable difference in the task network's final classification performance on the MNIST-M dataset. This could also be interpreted as the fact that the generator was able to closely model the target distribution.

Since we used two datasets with images of fairly low resolution (28×28) we had no problem training the model with 32 images per batch, which probably helped stabilize training somewhat. For images of higher resolution we would have to use smaller batches, and by then using images directly sampled from the source dataset we would be able to train the task network with twice as many samples per update. In this scenario using images from the source domain as well as translated images might have been more important.

The fact that the classification performance was higher when we used the residual network as generator representation was slightly surprising to us. The U-Net architecture is commonly recommended for image-to-image translation problems where the source domain and the target domain representation share a lot of structural similarity, as in this case. One reason that the U-Net might have been slightly worse in this case could be because of the background of the MNIST-M images being random, and information about what type of background should be generated during the translation phase can not be deciphered from the source domain image. In essence the information that is skipped between the encoder and the decoder in the U-Net architecture will not serve to help the decoder deduce what type of background is to be generated. Of course, the residual network also uses skip-connections, but in the case of the residual network the later layers will not have direct access to information from the earlier layers in the network, and the resulting translated image usually is more dissimilar to the input image than images that are translated via the U-Net architecture.

5.2 Translating GTA 5 to Cityscapes

In this section the results of the GTA 5 to Cityscapes domain adaptation experiment will be discussed. We adopted the neural network topology and training techniques described in the original CycleGAN paper to perform unsupervised domain adaptation between the synthetic GTA 5 dataset and the Cityscapes dataset.

As was presented in Section 4.2 we had mixed results for this translation problem. Some samples were quite impressive, and showed that the generators were able to pick up on the different idiosyncrasies of the two datasets, while restricting themselves from making other unwanted changes to the structures and semantics of the images. After a certain amount of training however, the quality of the results started to deteriorate as the generators started to introduce visual artifacts such as tree-like artifacts and even the Mercedes hood ornament in images translated from the GTA 5 dataset.

The Mercedes hood ornament in particular is an interesting case since it occurs in most of the images in the Cityscapes dataset, as the camera that has captured the data in this dataset is positioned on this car. The discriminator of the Cityscapes domain quickly learns that the presence of this Mercedes hood ornament is a strong indicator for that the image is sampled from the Cityscapes dataset, and as the networks keep training the generator that translates images to resemble those in the Cityscapes dataset starts to learn that by introducing these ornaments in images it is possible to fool the discriminator into labeling those as valid. On the other hand, in the GTA 5 dataset the hood ornament never occurs, so when the generator that translates images from the Cityscapes dataset to the GTA 5 dataset learns that it has a higher chance of fooling the GTA 5 discriminator by removing the hood ornament in the original Cityscapes dataset, it also enables the other generator to introduce these hood ornaments in images from the GTA 5 dataset when they are translated to the Cityscapes domain.

Since the generators are trained by cycle-consistency loss, which usually serves to reduce semantic changes to images during translation, it is in both generators' interest to make sure that the original image is reconstructed when translated back to the original domain. So the generators are not penalized in this case by the fact that the first generator introduces Mercedes hood ornaments with the purpose of fooling the discriminator of the Cityscapes dataset since the second generator has learned to remove these ornaments when translating images back to the GTA 5 dataset. Unfortunately this was not a problem we managed to mend.

If pixel-wise semantic labels would have been available for both datasets, it would be possible to further constrain the generators, as we could have filtered out each semantic region and have the discriminators evaluate each region independently by whether it is valid or generated. Such a solution was presented by Li *et al.* in their paper *Semantic-aware Grad-GAN for Virtual-to-Real Urban Scene Adaption* [34].

Despite there being artifacts in some samples, some of the results produced by the model were quite impressive and captured features in the target domain that are distinct for that dataset. One interesting example of this can be seen in the third column of Figure 4.5, where the generator has changed the color of the road lines from white to yellow, a feature that is common in the GTA 5 dataset but almost never occurs in the Cityscapes dataset.

5.3 3D Object Detection - Frustum-PointNet

As presented in Section 4.4, we were able to closely match the *KITTI val* performance of the original Frustum-PointNet implementation by Qi *et al.* when training the model on *KITTI train*. This demonstrates the robustness and usability of the Frustum-PointNet architecture.

The performance gap was significantly bigger on *KITTI test*, but this difference can to a large extent be explained by the difference in performance in the utilized 2D detectors, as demonstrated in Table 5.1. The table includes performance on *KITTI val* for two pairs of input 2D detections and their resulting 3D detections, and shows the significant effect 2DOD performance can have on 3DOD performance.

Table 5.1: *Performance on KITTI val for two pairs of input 2D detections and their resulting 3D detections.*

	Easy	**Moderate**	**Hard**
Input 2D detections (*2D AP*)	95.92 %	90.09 %	87.48 %
Resulting 3D detections (*3D AP*)	76.64 %	64.06 %	57.28 %
Input 2D detections (*2D AP*)	78.44 %	70.22 %	63.63 %
Resulting 3D detections (*3D AP*)	63.04 %	49.61 %	44.52 %

The qualitative results presented in Section 4.4 seem to suggest that our Frustum-PointNet implementation is able to accurately predict 3D bounding

boxes for most detected vehicles. Qualitatively, the predictions generally seem to be less accurate in cases where the detected vehicle has only a small number of associated LiDAR points: *e.g.* when the vehicle is heavily occluded by other objects or is far away from the ego-vehicle.

Another fundamental weakness of the Frustum-PointNet architecture is that it relies on a 2D detector to provide input 2D bounding boxes: no 2D detection means no 3D detection. This is not too much of a problem in our automatic annotation application, where 2D ground truth is assumed to be available, but it does limit the architecture's maximum performance and general usability. The architecture can however be extended to directly also take 3D region proposals as input, as demonstrated by Qi *et al.* in the appendix of their Frustum-PointNet paper [47].

Our Frustum-PointNet implementation trained on *SYN train* was found to lag significantly, in terms of performance on *KITTI val*, compared to the one trained on *KITTI train*. The performance gap was however significantly reduced when evaluated with a 50 % threshold, which seems to suggest that the predictions of the model trained on *SYN train* are actually reasonably close to the ground truth. Qualitatively, the main flaw seems to be that the predicted 3D boxes generally are too large, which of course has a negative effect on the performance metrics. This issue is actually quite expected, since the average ground truth vehicle size in *KITTI train* and *SYN train* differs quite substantially, as shown in Table 5.2.

Table 5.2: *Average ground truth vehicle size in KITTI train and SYN train.*

Dataset	Average height	Average width	Average length
KITTI train	1.5 m	1.6	3.9
SYN train	1.7 m	1.8 m	5.0 m

Overall then, our answer to the first research question posed in Section 1.2 is that a Frustum-PointNet model trained on *SYN train* transfers reasonably well to KITTI. We do not consider the obtained performance on *KITTI val* to be good enough for fully automatic annotation, but we do believe that the model could be successfully applied in a semi-automatic annotation process. Qualitatively, the predicted 3D boxes are definitely not perfect, but they are in most cases relatively close to the ground truth and should be easily improvable by a human annotator, given an appropriate annotation tool.

5.4 3D Object Detection - Extended Frustum-PointNet

As presented in Section 4.5, we were not able to improve 3D object detection performance by extending our Frustum-PointNet implementation to also utilize image features. In fact, we found that the extended architecture was clearly outperformed across both evaluated datasets and all metrics.

This was definitely a surprising result. Intuitively, having access to both LiDAR and image features should only improve performance, not degrade it. Qi *et*

al. also expressed a similar view in their Frustum-PointNet paper [47] when discussing future work. Additionally, adding image features to a LiDAR-only model was found to improve performance in previous work by both Chen *et al.* [10] and Xu *et al.* [63].

Qualitatively, the extended architecture definitely seems to improve the yaw angle estimate for vehicles far in-front of the ego-vehicle. This is most clearly observed by comparing the visualization video sequences on *KITTI test* for our Frustum-PointNet[1] and extended Frustum-PointNet[2] implementation: for the extended architecture, the predicted yaw angle is generally less volatile and more accurate. Distant vehicles do however correspond to only a small number of points in the LiDAR point cloud, which makes the estimation of 3D location more challenging and thus the expected 3D IoU with the ground truth box to remain quite small.

Adding image features might also have made the model more prone to overfit to the training data and its most common ground truth yaw angles and vehicle sizes. Comparing Figure 4.7 and Figure 4.9, the extended model does indeed seem less inclined to correctly output the relatively uncommon yaw angles of the cars parked on the left side of the road.

In Table 4.8 and Table 4.9, one can also observe that the relative drop in performance for the extended model is even more significant when trained on *SYN train*. Our best answer to the second research question posed in Section 1.2 is thus that adding image features to the model has a negative effect on its general performance, and especially on its ability to transfer from SYN to KITTI.

5.5 3D Object Detection - Image-Only Model

The quantitative results presented in Section 4.6 were quite competitive compared to SOTA for image-only 3DOD models, despite being far away from any of the best-performing LiDAR-based methods. The relatively good performance when evaluated with a 50 % threshold, together with most qualitative results, does however seem to suggest that the predicted 3D boxes not are too far away from ground truth in most cases.

Overall then, the concept of image-only 3DOD does actually seem reasonably promising. Not as the main perception system for a fully autonomous vehicle perhaps, but it could probably come to good use in inexpensive, vision-based systems for various active safety features.

The qualitative results also show the usefulness of having access to LiDAR point clouds when evaluating the performance of image-only 3DOD models. As seen for *e.g.* the van on the right side of the road in example 22 of Figure 4.10, the predicted 3D boxes usually seem accurate when projected onto the image plane, even in cases where the estimated 3D location is actually quite far from ground truth. Since a too small 3D box located too close to the camera could look identical to an accurately predicted box when projected onto the image, qualitatively

[1] `https://photos.app.goo.gl/QRgbU2r8JB5XABpw1`
[2] `https://photos.app.goo.gl/daH5v9jkG5zUmKfL2`

evaluating a 3DOD model solely by in-image visualizations is a fundamentally flawed concept.

The image-only model's virtually complete inability to naively transfer from SYN to KITTI was quite expected, and serves as a good example of the inherent difficulty associated with distinctly different domains.

5.6 Domain Adaptation - SYN to KITTI

In this section we will discuss the results of the experiment where we performed domain adaptation between SYN and KITTI for the purpose of 3D object detection. Despite the relatively high visual quality of the translated images, and their resemblance to images from the KITTI dataset, we were not able to gain a significant improvement upon the performance of the reference model neither for our image-only model nor for the extended Frustum-PointNet architecture. For the image-only model we did observe a slight improvement in terms of average precision, but the performance did not come close to a model trained on the target domain. For the extended Frustum-PointNet trained during domain adaptation we even observed a degraded performance compared to an identical model trained on the source domain.

This was a surprising result, as we expected that the difference between SYN and KITTI would be significant enough to leave room for improvement when performing domain adaptation, but this was not reflected in the results. We have a couple of theories why that might be the case. One problem with using image features in this case was that the two datasets were captured with different camera types, and the KITTI dataset had a significantly broader field of view. Since we scaled each image crop to a 128×128 resolution before feeding them to the task network, objects that occur in the left or right field of the image will have an altered aspect ratio after they are rescaled. Since KITTI has a broader field of view it will be more common that cars are rescaled to a resolution where the aspect ratio is severely altered, and if a model is trained purely on images from a separate dataset, where this does not occur to the same extent, the performance will of course be degraded when it is expected to carry out its task on objects that are not resembling those that the model has seen during training. A domain adaptation solution that has been trained to preserve structure can not be expected to fix this issue.

Our answer to the third research question posed in Section 1.2 is thus that performing domain adaptation does not have a significant effect on the extended Frustum-PointNet architecture's ability to transfer from SYN to KITTI. Its performance is also substantially worse than that of our Frustum-PointNet implementation.

5.7 Future Work

One major area suitable for future work is that of uncertainty estimation for the predicted 3D bounding boxes outputted by our 3D object detection models. Cur-

rently, we use the confidence score outputted by the 2D detector to obtain a confidence score also for the overall 3D detection, but we do not have any uncertainty measure for the estimated 3D box parameters $(x, y, z, h, w, l, \theta)$. For instance, the estimated parameters are likely more uncertain for a partially occluded vehicle located far away from the ego-vehicle than for a clearly visible vehicle right in-front. Also, for an image-only model the 3D location parameters (x, y, z) are probably associated with a larger uncertainty overall than the yaw angle θ. Currently however, our models are completely incapable of this type of reasoning, and it is impossible to know how certain any given 3D box prediction actually is. Promising recent work in this area is that of Ilg *et al.* in [24], where they present a model for optical flow estimation which also learns to output an estimated uncertainty of its prediction.

Our 3DOD models could potentially also be improved by taking inspiration from the work by Mousavian *et al.* in [43] and constrain the predicted 3D bounding boxes to not be too far away from the input 2D boxes when projected onto the image plane.

Another reasonably straightforward task for future work would be to extend our 3DOD models to full 360° field-of-view. This would require a vehicle equipped with a well-calibrated surround-vision camera system and a 2D detector specially trained for this input, but the actual 3DOD architectures would likely not need any significant modifications.

The 3D detections outputted by our 3DOD models could potentially also be taken as input to a recursive Bayesian filter in order to track objects in 3D. Promising recent work in this area includes that of Scheidegger *et al.* in [56], where they perform 3D multi-object tracking by combining a 2D detector, which also estimates the distance to detected objects, and a SOTA tracking filter.

Our image-only 3DOD architecture is a relatively straightforward combination of ideas from previous work and is likely improvable in terms of 3DOD performance. It is however almost certainly improvable in terms model inference time, and future work would thus first focus on implementing the technique from Fast R-CNN [16], *i.e.*, to only extract one feature map for the entire image instead of feeding each individual input 2D box through a CNN.

The most surprising result in this thesis was probably that extending the Frustum-PointNet architecture to also utilize image features actually degraded its 3DOD performance, even when both trained and evaluated on KITTI. Before one can conclude that it is impossible to achieve a performance gain by incorporating image features, further experiments and a more formal analysis of the difference in performance compared to the original architecture do however need to be performed.

In this thesis, we resorted to performing domain adaptation on the feature-level and pixel-level of images. One possibility for future work is however to also perform domain adaptation on some form of data representation for the LiDAR point clouds. The model only utilizing LiDAR proved to achieve the best performance when only training on data from the source domain (SYN) with the purpose of performing well on the target domain (KITTI), and by adapting the LiDAR data available in the source domain it could be possible to further improve

upon this performance.

6

Conclusion

In this thesis, we have studied the computer vision problem of 3D object detection and how Generative Adversarial Networks can be applied to perform domain adaptation for this task.

We implemented the Frustum-PointNet architecture for 3D object detection and closely matched its reported performance when trained and evaluated on the KITTI dataset. The architecture was found to transfer reasonably well from the synthetic SYN dataset to KITTI and is believed to be usable in a semi-automatic 3D bounding box annotation process. The Frustum-PointNet architecture was also extended to utilize image features, which surprisingly degraded its detection performance. Furthermore, we designed and implemented an image-only 3D object detection model which compared quite favourably with current state-of-the-art in terms of detection performance.

The PixelDA approach was adopted and successfully applied to the MNIST to MNIST-M domain translation problem, which validated the idea that unsupervised domain adaptation using generative adversarial networks can significantly improve the performance of a task network for a dataset lacking annotations. Although training the image-only 3D object detection model during domain adaptation did improve the performance of the model when evaluated on the target domain, it did not come close to the performance achieved when the model was trained directly on the target domain. Using the Extended Frustum-PointNet as task network resulted in worse performance when evaluated on the target domain than when it was trained directly on the source domain and tested on the target domain.

Finally, interesting areas for future work were presented, including extending the 3D object detection models to also output an uncertainty estimate and applying domain adaptation techniques on LiDAR point clouds.

Bibliography

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. Cited on pages 19 and 20.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. Cited on page 11.

[3] Björn Barrois and Christian Wöhler. 3d pose estimation of vehicles using stereo camera. In *Encyclopedia of Sustainability Science and Technology*, pages 10589–10612. Springer, 2012. Cited on page 15.

[4] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 95–104, 2017. Cited on pages 37 and 53.

[5] Mary Ann Branch, Thomas F Coleman, and Yuying Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21(1):1–23, 1999. Cited on page 51.

[6] Luca Caltagirone, Samuel Scheidegger, Lennart Svensson, and Mattias Wahde. Fast lidar-based road detection using fully convolutional neural networks. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 1019–1024. IEEE, 2017. Cited on page 8.

[7] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. In *Proceedings of*

the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2040–2049, 2017. Cited on pages 16 and 49.

[8] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *Advances in Neural Information Processing Systems*, pages 424–432, 2015. Cited on pages 15, 16, 40, and 67.

[9] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2147–2156, 2016. Cited on pages 15, 16, and 67.

[10] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2017. Cited on pages 15, 17, 18, 61, and 79.

[11] Xinxin Du, Marcelo H Ang Jr, Sertac Karaman, and Daniela Rus. A general pipeline for 3d detection of vehicles. *arXiv preprint arXiv:1803.00387*, 2018. Cited on page 61.

[12] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1355–1361. IEEE, 2017. Cited on page 15.

[13] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2016. Cited on page 24.

[14] Andreas Geiger. Kitti 3d object detection benchmark leader board, 2017. URL `http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d`. Accessed: 2018-02-04. Cited on pages 3, 40, 41, 56, 59, and 67.

[15] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. Cited on pages 2, 3, 9, 12, 14, and 39.

[16] Ross Girshick. Fast r-cnn. *IEEE International Conference on Computer Vision (ICCV)*, 2015. Cited on pages 12 and 81.

[17] 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*, pages 580–587, 2014. Cited on page 12.

[18] 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*, pages 2672–2680, 2014. Cited on pages 3 and 18.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. Cited on page 7.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. Cited on pages 31 and 49.

[21] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017. Cited on page 13.

[22] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213*, 2017. Cited on page 24.

[23] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. Cited on page 14.

[24] Eddy Ilg, Özgün Çiçek, Silvio Galesso, Aaron Klein, Osama Makansi, Frank Hutter, and Thomas Brox. Uncertainty estimates for optical flow with multi-hypotheses networks. *arXiv preprint arXiv:1802.07095*, 2018. Cited on page 81.

[25] Velodyne Acoustics Inc. *Velodyne's HDL-64E: A high definition LiDAR sensor fro 3-D applications (white paper)*. 2007. Cited on pages 8 and 39.

[26] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017. Cited on pages 21 and 23.

[27] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`. [Online; accessed 2018-04-23]. Cited on page 51.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14`. Cited on page 36.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. Cited on page 12.

[30] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation. *arXiv preprint arXiv:1712.02294*, 2017. Cited on pages 18 and 61.

[31] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009. Cited on page 16.

[32] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. Cited on page 15.

[33] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. page 17, 04 2016. Cited on page 22.

[34] Peilun Li, Xiaodan Liang, Daoyuan Jia, and Eric P Xing. Semantic-aware grad-gan for virtual-to-real urban scene adaption. *arXiv preprint arXiv:1801.01726*, 2018. Cited on page 77.

[35] 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*, pages 740–755. Springer, 2014. Cited on pages 13 and 14.

[36] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017. Cited on pages 13 and 14.

[37] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE International Conference on Computer Vision (ICCV)*, 2017. Cited on page 14.

[38] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 469–477. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6544-coupled-generative-adversarial-networks.pdf`. Cited on page 23.

[39] 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*, pages 21–37. Springer, 2016. Cited on page 13.

[40] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Least squares generative adversarial networks, 2016. URL `http://arxiv.org/abs/1611.04076`. cite arxiv:1611.04076. Cited on pages 22, 35, and 36.

[41] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015. Cited on page 8.

[42] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014. URL `http://arxiv.org/abs/1411.1784`. cite arxiv:1411.1784. Cited on page 20.

[43] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Košecká. 3d bounding box estimation using deep learning and geometry. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5632–5640. IEEE, 2017. Cited on pages 2, 16, 17, 48, 49, and 81.

[44] Libor Novak. Vehicle detection and pose estimation for autonomous driving. Master's thesis, Czech Technical University in Prague, 2017. Cited on page 69.

[45] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2642–2651, 2017. Cited on page 20.

[46] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. Cited on pages 46, 49, and 50.

[47] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. *arXiv preprint arXiv:1711.08488*, 2017. Cited on pages 17, 40, 42, 59, 60, 61, 78, and 79.

[48] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2017. Cited on pages 8, 17, 42, 44, and 45.

[49] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5105–5114, 2017. Cited on pages 10 and 42.

[50] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *International Conference on Learning Representations (ICLR)*, 2016. Cited on page 19.

[51] 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*, pages 91–99, 2015. Cited on page 13.

[52] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016. Cited on page 28.

[53] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. URL `http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a`. (available on arXiv:1505.04597 [cs.CV]). Cited on pages 21 and 32.

[54] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29*, pages 2234–2242. 2016. Cited on page 19.

[55] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser Nam Lim, and Rama Chellappa. Unsupervised domain adaptation for semantic segmentation with gans. *arXiv preprint arXiv:1711.06969*, 2017. Cited on pages 3, 23, and 24.

[56] Samuel Scheidegger, Joachim Benjaminsson, Emil Rosenberg, Amrit Krishnan, and Karl Granstrom. Mono-camera 3d multi-object tracking using deep learning detections and pmbm filtering. *arXiv preprint arXiv:1802.09975*, 2018. Cited on page 81.

[57] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. *CoRR*, abs/1612.07828, 2016. URL `http://dblp.uni-trier.de/db/journals/corr/corr1612.html#ShrivastavaPTSW16`. Cited on page 22.

[58] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013. Cited on page 12.

[59] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. *arXiv preprint arXiv:1711.11585*, 2017. Cited on page 33.

[60] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. Cited on page 11.

[61] Bichen Wu, Forrest Iandola, Peter H. Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time

object detection for autonomous driving. In *The IEEE Conference on Computer Vision and Pattern Recogsnition (CVPR) Workshops*, July 2017. Cited on page 14.

[62] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. *arXiv preprint arXiv:1710.07368*, 2017. Cited on page 8.

[63] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. *arXiv preprint arXiv:1711.10871*, 2017. Cited on pages 3, 17, 40, and 79.

[64] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. Cited on pages 2, 9, and 41.

[65] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *arXiv preprint arXiv:1711.06396*, 2017. Cited on pages 17, 18, 47, and 61.

[66] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *IEEE International Conference on Computer Vision (ICCV)*, 2017. Cited on pages 3, 22, 24, and 39.