Universitat Autònoma de Barcelona

Leveraging Synthetic Data to Create Autonomous Driving Perception Systems

Gabriel Villalonga Pineda

Leveraging Synthetic Data to Create Autonomous Driving Perception Systems

A dissertation submitted by **Gabriel Villalonga Pineda** at Universitat Autònoma de Barcelonato fulfil the degree of **Doctor of Philosophy**.

Bellaterra, December 15, 2020

Co-Directors	**Dr. Antonio López Peña** Dept. Ciències de la Computació & Centre de Visió per Computador Universitat Autònoma de Barcelona, Spain **Dr. Germán Ros** Intel Intelligent Systems Lab Santa Clara, CA, USA
Thesis committee	**Dr. Ernest Valveny** Dept. Ciències de la Computació & Centre de Visió per Computador Universitat Autònoma de Barcelona, Spain **Dr. Francesc Moreno** Institut de Robòtica i Informàtica Industrial CSIC & Universitat Politècnica de Catalunya **Dr. José Manuel Álvarez** AI-Infra, NVIDIA Santa Clara, CA, USA

This document was typeset by the author using LaTeX 2$_\varepsilon$.

ISBN: 978-84-122714-2-3

Printed by Ediciones Gráficas Rey, S.L.

Acknowledgements

With this thesis ends a stage of my life and a new one starts. When I look at my past self I see a young guy taking a journey which was insecure that he would be able to successfully end. Looking at it today, I am really grateful to have done it, which has changed me as a researcher and person. During this journey I have met a lot of people and I have had the support of a lot more, without whom I would have not been able to get where I am today. I need to thank so many people and I apologize if I forget someone.

To start I would like to thank those academically close, especially the ones who guided me through this journey. In order of appearance, I would first like to thank Dr. David Vazquez, for the faith you had in me since day one and for opening the world of research to me. Next, I would like to thank my co-director Dr. German Ros, you are an infinite source of knowledge and you were able to teach me so many things, most grateful for those which I would not have been aware of otherwise. Last but not least, I would like to thank my director Dr. Antonio Lopez, you helped me overcome many challenges and your guidance has been invaluable. I hope that I can become such an accomplished researcher as you some day. All three of you took an important role in my professional growth, without any of you this thesis would have not been the same.

I would also like to thank Dr. Peter Kontschieder for welcoming me and expanding my knowledge during my internship at Mapillary. I was able to meet great people such as Dr. Samuel Rota and Gerhard Neuhold, with whom I had invaluable discussions. I also extend my gratitude to Dr. Hamed Aghdam, Dr. Joost van de Weijer and Dr. Joan Serrat for the collaborations and for all that they taught me. Finally, Dr. Felipe Codevilla, thanks for the insightful conversations and I hope we work together again in the future.

I would also like to thank the people at the Computer Vision Center. Starting with the ADAS group where I had so many discussions about research with Diego Porres, Yi Xiao, Idoia Ruiz and Jose Luis Gomez. In particular, Jose Luis, with whom I am grateful to have been able to work with during so many projects. Other people that I was unlucky to spend too few time with, but I am anyways, grateful to share their internship with ADAS, Rohit Gajawada, Rishab Madan, Arjun Gupta, Motjaba Valipour, Sahil Gupta and Abishek Tandon. Also, I need to thank the SYNTHIA and CARLA teams, the work that you are doing for the group and the community is invaluable. Mainly Xisco Bosch and Marc Garcia, who had to suffer my constant need for data. In general, I would like to thank the whole CVC staff. In particular, administration for having my back with all the paperwork, the IT department, and

above all Joan Masoliver who had to deal with my server demands.

In addition, I would like to thank my closest friend circle. I would like to thank Smash Brothers and Vi Negre for helping me disconnect from work. Following with the closest people at CVC: Dani Hernandez, Mikel Menta and Laura Lopez. In special Marc Masana, who probably did not realize the importance he played through this journey, you were an anchor in the bad moments.

I leave my most inner circle to the end. I must thank my family which has always been there unconditionally without never expecting anything in return. You have always supported me in both good and bad moments. Without you, I would have not been able to to accomplish my goals. As someone said: "*La familia siempre estará allí solo porque es tu familia*", and I am very grateful for having you in my life.

Abstract

Manually annotating images to develop vision models has been a major bottleneck since computer vision and machine learning started to walk together. This has been more evident since computer vision falls on the shoulders of data-hungry deep learning techniques. When addressing on-board perception for autonomous driving, the curse of data annotation is exacerbated due to the use of additional sensors such as LiDAR. Therefore, any approach aiming at reducing such a time-consuming and costly work is of high interest for addressing autonomous driving and, in fact, for any application requiring some sort of artificial perception. In the last decade, it has been shown that leveraging from synthetic data is a paradigm worth to pursue in order to minimizing manual data annotation. The reason is that the automatic process of generating synthetic data can also produce different types of associated annotations (*e.g.* object bounding boxes for synthetic images and LiDAR pointclouds, pixel/point-wise semantic information, etc.). Directly using synthetic data for training deep perception models may not be the definitive solution in all circumstances since it can appear a synth-to-real domain shift. In this context, this work focuses on leveraging synthetic data to alleviate manual annotation for three perception tasks related to driving assistance and autonomous driving. In all cases, we assume the use of deep convolutional neural networks (CNNs) to develop our perception models.

The first task addresses traffic sign recognition (TSR), a kind of multi-class classification problem. We assume that the number of sign classes to be recognized must be suddenly increased without having annotated samples to perform the corresponding TSR CNN re-training. We show that leveraging synthetic samples of such new classes and transforming them by a generative adversarial network (GAN) trained on the known classes (*i.e.* without using samples from the new classes), it is possible to re-train the TSR CNN to properly classify all the signs for a $\sim 1/4$ ratio of new/known sign classes. The second task addresses on-board 2D object detection, focusing on vehicles and pedestrians. In this case, we assume that we receive a set of images without the annotations required to train an object detector, *i.e.* without object bounding boxes. Therefore, our goal is to self-annotate these images so that they can later be used to train the desired object detector. In order to reach this goal, we leverage from synthetic data and propose a semi-supervised learning approach based on the co-training idea. In fact, we use a GAN to reduce the synth-to-real domain shift before applying co-training. Our quantitative results show that co-training and GAN-based image-to-image translation complement each other up to allow the training of object detectors without manual annotation, and

still almost reaching the upper-bound performances of the detectors trained from human annotations. While in previous tasks we focus on vision-based perception, the third task we address focuses on LiDAR pointclouds. Our initial goal was to develop a 3D object detector trained on synthetic LiDAR-style pointclouds. While for images we may expect synth/real-to-real domain shift due to differences in their appearance (*e.g.* when source and target images come from different camera sensors), we did not expect so for LiDAR pointclouds since these active sensors factor out appearance and provide sampled shapes. However, in practice, we have seen that it can be domain shift even among real-world LiDAR pointclouds. Factors such as the sampling parameters of the LiDARs, the sensor suite configuration on-board the ego-vehicle, and the human annotation of 3D bounding boxes, do induce a domain shift. We show it through comprehensive experiments with different publicly available datasets and 3D detectors. This redirected our goal towards the design of a GAN for pointcloud-to-pointcloud translation, a relatively unexplored topic.

Finally, it is worth to mention that all the synthetic datasets used for these three tasks, have been designed and generated in the context of this PhD work and will be publicly released. Overall, we think this PhD presents several steps forward to encourage leveraging synthetic data for developing deep perception models in the field of driving assistance and autonomous driving.

Resumen

La anotación manual de imágenes para desarrollar sistemas basados en visión por computador ha sido uno de los puntos más problemáticos desde que se utiliza aprendizaje automático para ello. El problema se ha agravado desde la irrupción del aprendizaje profundo. Esta tesis se centra en el desarrollo de modelos profundos de percepción para conducción autónoma. Por tanto, no solo se requiere anotar imágenes, sino también nubes de puntos 3D que provienen de señores LiDAR, lo que resulta incluso más costoso. En la última década, se ha demostrado que merece la pena investigar el uso de datos sintéticos para minimizar esos costes. La razón es que los datos sintéticos pueden generarse con diferentes tipos de anotaciones asociadas de forma automática, p. ej., la localización de los objetos de interés, información semántica a nivel de pixeles/puntos, etc. Sin embargo, entrenar modelos profundos con datos sintéticos y posteriormente utilizarlos para operar en entornos reales puede dar lugar a problemas de adaptación de dominio, en otras palabras, la precisión de los modelos no es la esperada. En este contexto, esta tesis se centra en aprovechar los datos sintéticos para aliviar el coste de las anotaciones manuales en tres tareas de percepción relacionadas con la asistencia a la conducción y la conducción autónoma. En todo momento asumimos el uso de redes neuronales convolucionales para el desarrollo de nuestros modelos profundos de percepción.

La primera tarea plantea el reconocimiento de señales de tráfico, un problema de clasificación de imágenes. Asumimos que el número de clases de señales de tráfico a reconocer se debe incrementar sin haber podido anotar nuevas imágenes con las que realizar el correspondiente reentrenamiento. Demostramos que aprovechando los datos sintéticos de las nuevas clases y transformándolas con una red adversaria-generativa (GAN, de sus siglas en inglés) entrenada con las clases conocidas (sin usar muestras de las nuevas clases), es posible reentrenar la red neuronal para clasificar todas las señales en una proporción de $\sim 1/4$ entre clases nuevas y conocidas. La segunda tarea consiste en la detección de vehículos y peatones (objetos) en imágenes. En este caso, asumimos la recepción de un conjunto de imágenes sin anotar. El objetivo es anotar automáticamente esas imágenes para que así se puedan utilizar posteriormente en el entrenamiento del detector de objetos que deseemos. Para alcanzar este objetivo, partimos de datos sintéticos anotados y proponemos un método de aprendizaje semi-supervisado basado en la idea del co-aprendizaje. Además, utilizamos una GAN para reducir la distancia entre los dominios sintético y real antes de aplicar el co-aprendizaje. Nuestros resultados cuantitativos muestran que el procedimiento desarrollado permite ano-

tar el conjunto de imágenes de entrada con la precisión suficiente para entrenar detectores de objetos de forma efectiva; es decir, tan precisos como si las imágenes se hubiesen anotado manualmente. En la tercera tarea dejamos atrás el espacio 2D de las imágenes, y nos centramos en procesar nubes de puntos 3D provenientes de sensores LiDAR. Nuestro objetivo inicial era desarrollar un detector de objetos 3D (vehículos, peatones, ciclistas) entrenado en nubes de puntos sintéticos estilo LiDAR. En el caso de las imágenes cabía esperar el problema de cambio de dominio debido a las diferencias visuales entre las imágenes sintéticas y reales. Pero, a priori, no esperábamos lo mismo al trabajar con nubes de puntos LiDAR, ya que se trata de información geométrica proveniente del muestreo activo del mundo, sin que la apariencia visual influya. Sin embargo, en la práctica, hemos visto que también aparecen los problemas de adaptación de dominio. Factores como los parámetros de muestreo del LiDAR, la configuración de los sensores a bordo del vehículo autónomo, y la anotación manual de los objetos 3D, inducen diferencias de dominio. En la tesis demostramos esta observación mediante un exhaustivo conjunto de experimentos con diferentes bases de datos públicas y detectores 3D disponibles. Por tanto, en relación a la tercera tarea, el trabajo se ha centrado finalmente en el diseño de una GAN capaz de transformar nubes de puntos 3D para llevarlas de un dominio a otro, un tema relativamente inexplorado.

Finalmente, cabe mencionar que todos los conjuntos de datos sintéticos usados en estas tres tareas han sido diseñados y generados en el contexto de esta tesis doctoral y se harán públicos. En general, consideramos que esta tesis presenta un avance en el fomento de la utilización de datos sintéticos para el desarrollo de modelos profundos de percepción, esenciales en el campo de la conducción autónoma.

Resum

L'anotació manual d'imatges per desenvolupar sistemes basats en visió per computador ha estat un dels punts més problemàtics des que s'utilitza aprenentatge automàtic per a això. El problema s'ha agreujat des de la irrupció de l'aprenentatge profund. Aquesta tesi es centra en el desenvolupament de models profunds de percepció per a conducció autònoma. Per tant, no només es requereix anotar imatges, sinó també núvols de punts 3D que provenen de senyors LiDAR, el que resulta fins i tot més costós. En l'última dècada, s'ha demostrat que val la pena investigar l'ús de dades sintètiques per minimitzar aquests costos. La raó és que les dades sintètiques es poden generar amb diferents tipus d'anotacions associades de forma automàtica, p. ex., la localització dels objectes d'interès, informació semàntica a nivell de píxels/punts, etc. Tot i això, entrenar models profunds amb dades sintètiques i posteriorment utilitzar-los per operar en entorns reals pot donar lloc a problemes d'adaptació de domini, en altres paraules, la precisió dels models no és l'esperada. En aquest context, aquesta tesi es centra en aprofitar les dades sintètiques per alleujar el cost de les anotacions manuals en tres tasques de percepció relacionades amb l'assistència a la conducció i la conducció autònoma. En tot moment assumim l'ús de xarxes neuronals convolucionals per al desenvolupament dels nostres models profunds de percepció.

La primera tasca planteja el reconeixement de senyals de trànsit, un problema de classificació d'imatges. Assumim que el nombre de classes de senyals de trànsit a reconèixer s'ha d'incrementar sense haver pogut anotar noves imatges amb què realitzar el corresponent reentrenament. Demostrem que aprofitant les dades sintètiques de les noves classes i transformant-les amb una xarxa adversària-generativa (GAN, de les seves sigles en anglès) entrenada amb les classes conegudes (sense usar mostres de les noves classes), és possible reentrenar la xarxa neuronal per classificar tots els senyals en una proporció de $SIM1/4$ entre classes noves i conegudes. La segona tasca consisteix en la detecció de vehicles i vianants (objectes) en imatges. En aquest cas, assumim la recepció d'un conjunt d'imatges sense anotar. L'objectiu és anotar automàticament aquestes imatges perquè així es puguin utilitzar posteriorment en l'entrenament del detector d'objectes que desitgem. Per assolir aquest objectiu, vam partir de dades sintètiques anotades i proposem un mètode d'aprenentatge semi-supervisat basat en la idea del co-aprenentatge. A més, utilitzem una GAN per reduir la distància entre els dominis sintètic i real abans d'aplicar el co-aprenentatge. Els nostres resultats quantitatius mostren que el procediment desenvolupat permet anotar el conjunt d'imatges d'entrada amb la precisió suficient per entrenar detectors d'objectes de forma efectiva; és a dir,

tan precisos com si les imatges s'haguessin anotat manualment. A la tercera tasca deixem enrere l'espai 2D de les imatges, i ens centrem en processar núvols de punts 3D provinents de sensors LiDAR. El nostre objectiu inicial era desenvolupar un detector d'objectes 3D (vehicles, vianants, ciclistes) entrenat en núvols de punts sintètics estil LiDAR. En el cas de les imatges es podia esperar el problema de canvi de domini degut a les diferències visuals entre les imatges sintètiques i reals. Però, a priori, no esperàvem el mateix en treballar amb núvols de punts LiDAR, ja que es tracta d'informació geomètrica provinent del mostreig actiu del món, sense que l'aparença visual influeixi. No obstant això, a la pràctica, hem vist que també apareixen els problemes d'adaptació de domini. Factors com els paràmetres de mostreig del LiDAR, la configuració dels sensors a bord del vehicle autònom, i l'anotació manual dels objectes 3D, indueixen diferències de domini. A la tesi demostrem aquesta observació mitjançant un exhaustiu conjunt d'experiments amb diferents bases de dades públiques i detectors 3D disponibles. Per tant, en relació amb la tercera tasca, el treball s'ha centrat finalment en el disseny d'una GAN capaç de transformar núvols de punts 3D per portar-los d'un domini a un altre, un tema relativament inexplorat.

Finalment, cal esmentar que tots els conjunts de dades sintètiques usats en aquestes tres tasques han estat dissenyats i generats en el context d'aquesta tesi doctoral i es faran públics. En general, considerem que aquesta tesi presenta un avanç en el foment de la utilització de dades sintètiques per al desenvolupament de models profunds de percepció, essencials en el camp de la conducció autònoma.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Autonomous vehicles

Autonomous vehicles (AVs) are receiving increasing attention from the research community, the industry, and the authorities responsible for the mobility of citizens and goods. The EU consisders them as one of the top-10 technologies that will change our lives [167]. Well-coordinated fleets of AVs are seen as the solution to reduce the amount of individual infra-occupied and infra-used vehicles on the roads, so significantly reducing congestion and pollution, along saving energy. Furthermore, AVs have the advantage of not being distracted with non-driving tasks, not falling asleep, and respecting speed limits, thus, contributing to a safer driving. Note that distraction, tiredness, and not respecting speed limits, are major causes of traffic fatalities [165]. In fact, according to [15], AVs do not need to show a perfect performance in all circumstances for their deployment to be worth, since being able to perform slightly better than human drivers is enough to save hundreds of thousands of lives over 30 years.

AVs rely on sensors to capture the surrounding scene, since they are placed on different positions of the vehicle to avoid blind spots. This data consist of visual appearance (from regular cameras), distance and speed for moving traffic participants (from radars), distances and 3D shapes (from LiDARs), thermal information (from far infrared cameras), global localization (*e.g.* from GPS) or ego-vehicle motion information (from IMUs); additionally including vehicle-to-X (V2X) communications [134], *i.e.* the ability to communicate with other vehicles and/or traffic infrastructure to access useful information such as the presence of oncoming vehicles/pedestrians/etc. occluded behind the corner, or just for coordinating joint maneuvers. Once processed, all these data should make AVs able to take more accurate driving actions than humans. However, even for machines, driving is a highly complex task to be codified just as a set of hard-coded rules, so the artificial drivers rely on Artificial Intelligence (AI) to provide the expected behaviour [54, 69, 91, 182].

Accordingly, the mobility needs and the advances on AI encourage the development of AVs; however, we are still far from the performance of good human drivers.

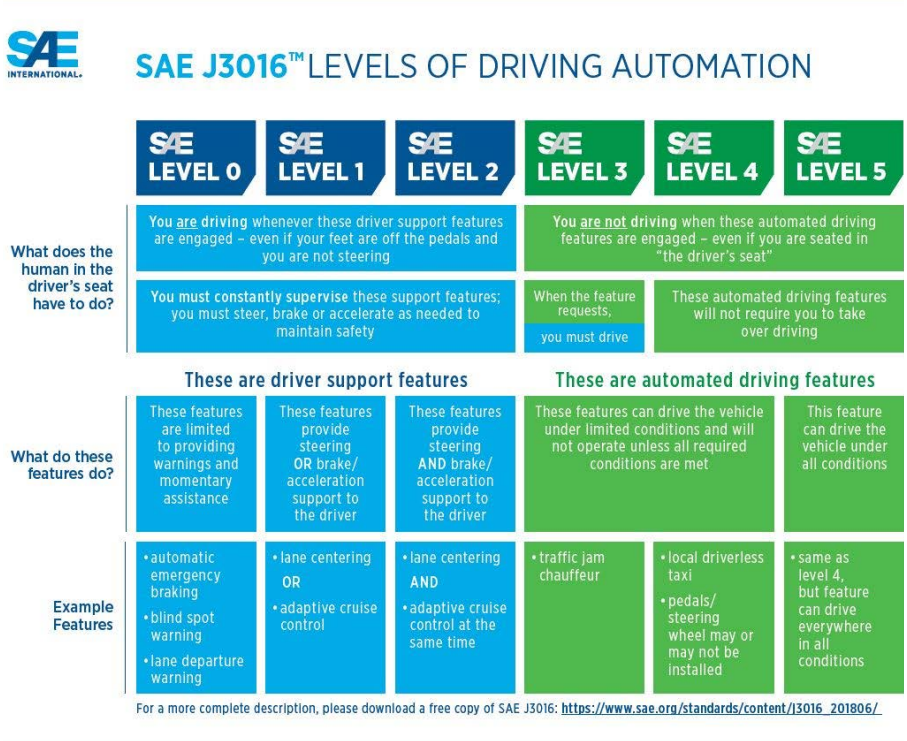

Figure 1.1 – SAE Levesl of driving automation. Source: <sae.org>.

The Society of Automotive Engineers (SAE) classifies the degrees of vehicle autonomy according to six levels, which can be seen in Fig. 1.1. Most of the automatized vehicles today are at some point between Level 2 and Level 3, so just crossing the border where the driving responsibility moves from the human to the AI; thus, we are still far from the goal of fully replacing human drivers in all situations, which corresponds to Level 5.

Most AI drivers behind AVs are based on a set of conceptual modules [91, 182] as those summarized in Fig. 1.2. The global planner acts at a high navigation level, defining which route the vehicle should follow to reach the desired destiny; which, in fact, nowadays is also used to support human driving. Once the route is defined, the scene understanding module identifies and localizes relevant traffic elements along this route, such as dynamic participants (*i.e.*, other vehicles, pedestrians, etc.) and static infrastructure (*i.e.*, free road surface, traffic signs, etc.). This information

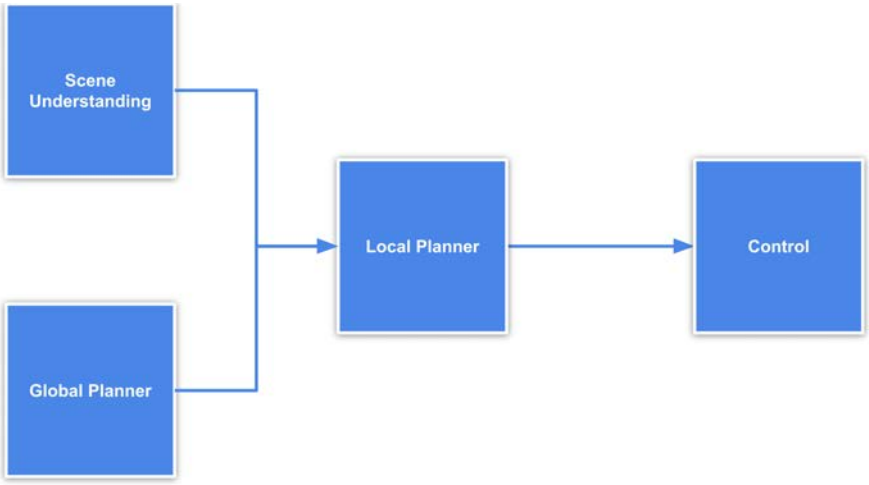

Figure 1.2 – Conceptualization of a classic autonomous driving pipeline.

is sent to the local planner to compute a safe maneuver keeping the desired route. Once the maneuver is planed, the control module executes the low level commands (*i.e.*, steering, acceleration/brake) to move the AV accordingly.

This PhD dissertation focuses on the scene understanding module, also known as *environmental perception* since the understanding is derived from the data provided by the on-board sensors.

1.2 Traffic scene understanding

There are different approaches to address traffic scene understanding. Roughly speaking, we can see approaches where predefined 3D semantic maps are assumed to exists, and others for which it is not the case [69, 80, 91, 182]. The former approaches assume that the AV localizes itself in a map in real-time, which allows it to access prior information about essential traffic infrastructure for driving, such as accurate localization of lanes, lane markings, sidewalks, intersections, traffic lights, as well as traffic sign enforcement. This information is then complemented by the on-board detection of dynamic traffic participants (called *objects* in the perception literature) such as vehicles, pedestrians, motorbikes and bicyclists, and, eventually, the state of some traffic infrastructure such as the one of traffic lights or changing traffic signs. In the map-free approaches, on-board object detection and recogni-

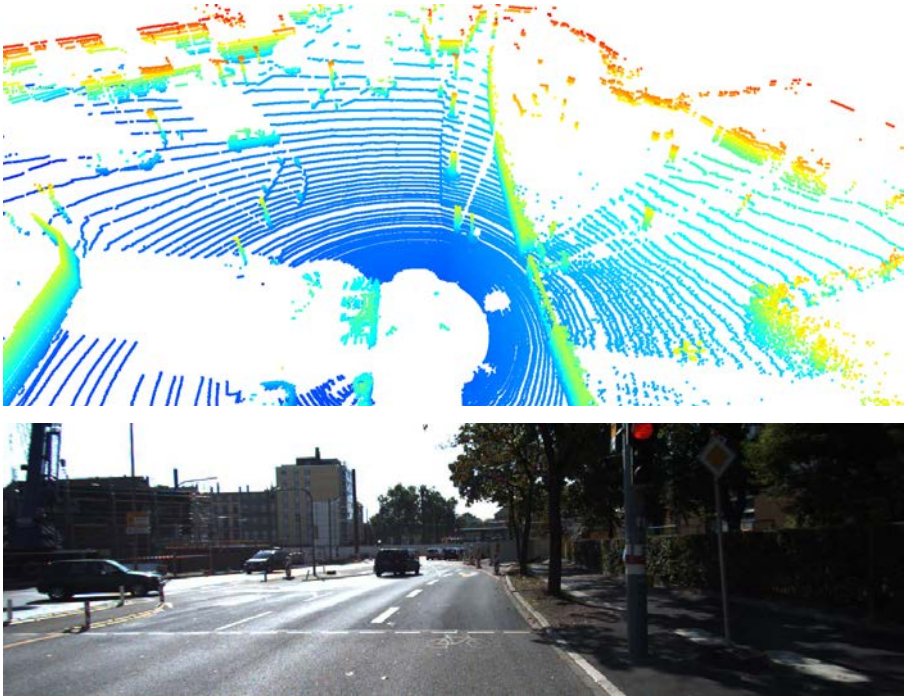

Figure 1.3 – Top: 3D pointcloud captured by a LiDAR sensor, color codifies height. Bottom: usual on-board RGB image from a forward facing camera behind the windshield. Both data samples correspond to the KITTI dataset [46].

tion tasks must be performed too; however, in addition, the traffic infrastructure potentially affecting the AV driving must be also identified, *i.e.* the lane limits, the state of some traffic elements (*e.g.*, traffic lights, certain traffic signs), etc. Obviously, these approaches require to solve more on-board tasks, but, on the other hand, relying on pre-recorder maps also resorts challenging problems such as their continuous updating at worldwide scale, the real-time access to them, and developing reliable 24/7 on-board and real-time vehicle localization algorithms.

In practical terms, traffic scene understanding is achieved when the state of all the dynamic objects (detection and motion/intention estimation) and static infrastructure (localization and semantic state recognition) affecting the AV is captured. In any case, *it is clear that on-board object detection and recognition is required as part of the perception behind any approach to traffic scene understanding.*

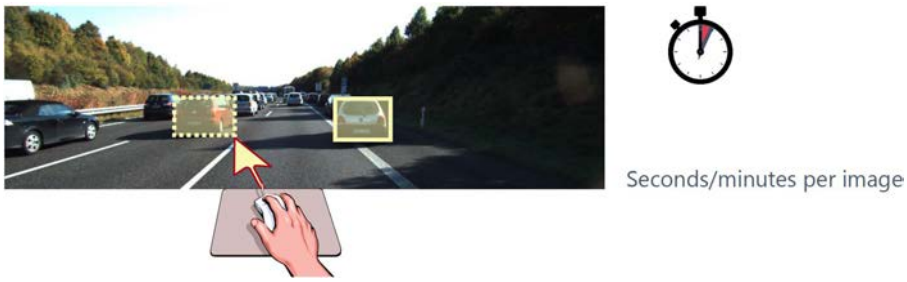

Figure 1.4 – Manual annotation of 2D bounding boxes framing vehicles.

The most common sensors used to address these tasks are cameras and LiDARs [69, 80, 182]. Fig. 1.3 shows a sample from each sensor. Cameras are required when the visual appearance is a must as for performing traffic sign recognition or detecting lane markings, while LiDAR sensors are specially accurate on providing distance to obstacles. In both cases, we can address the detection of objects such as vehicles and pedestrians, even following a multi-modal approach [7, 9, 31, 41, 79, 112, 115] provided the sensors are calibrated (synchronized and aligned). In fact, this PhD candidate has actively participated in such kind of approaches [52]. From this experience, the PhD candidate believes that Level 5 AVs will rely on both sensor technologies (most probably complemented with others such as radar, GNSS, and V2X), therefore, *in this PhD we focus on RGB images and LiDAR 3D pointclouds to develop models for on-board object detection and recognition.*

1.3 The need for annotated data

Nowadays, best models to perform on-board object detection and recognition rely on convolutional neural networks (CNNs), which are trained in a supervised manner. For instance, training a CNN for object detection requires bounding boxes framing the objects, which are usually provided by a costly and cumbersome process of manual annotation (see example in Fig. 1.4). Since, in addition, thousands or hundred of thousands of data samples (images or LiDAR 3D pointclouds) are required to train CNNs and, in fact, also for testing them, *data collection and annotation is a current practical bottleneck in the development of AI drivers.* In Table 1.1 we show a summary of the publicly available datasets for addressing autonomous driving perception, most of them released in last two years. It is worth to mention that just blindly summing up the data from all these datasets is not necessarily a good idea since the respective samples were captured from different

Table 1.1 – Publicly available datasets captured on-board vehicles. We consider only those providing detection bounding boxes with good quality. In the sensor column, C stands for camera and L for LiDAR.

Dataset	Images	Sensor	Annotations	Region	Images/second
KITTI [46]	7,4k	C & L	2D & 3D BBs	Germany	1
Waymo [142]	200M	C & L	2D & 3D BBs	US	10.000
Lyft [67]	30k	C & L	2D & 3D BBs	US	-
Tsinghua [187]	100k	C	2D BBs	China	1
Cityscapes [36]	5k	C	2D & 3D BBs	Germany	1
Nuscenes [21]	390k	C & L	2D & 3D BBs	Boston & Singapure	20
Apollo [160]	5.4k	C & L	2D & 3D BBs	China	2

sensor models and/or settings which introduces a *domain shift* [37, 159, 166].

Beyond trying to improve manual annotation processes, an alternative approach is to use synthetic datasets for training perception models while minimizing manual data annotation on real-world images. The key here is that synthetic data can come with all kinds of automatically generated ground truth (supervision), as can be seen in Fig. 1.5. This is a paradigm in which the CVC's group hosting the PhD candidate has been pioneer [38, 98, 123, 155, 171], and nowadays is a topic of worldwide research in itself since training on synthetic data and testing on real-world one challenges domain adaptation methods too [72, 96]. In fact, synthetic data can also be used to test vision algorithms [11, 38, 63, 100, 121, 126, 129, 145] and learning protocols [189]. The latter case, corresponding to a work on active learning where this PhD candidate contributed with the design of a proper synthetic dataset for the assessment of active learning algorithms.

Accordingly, *this PhD dissertation focuses on the use of synthetic data to train CNN-based models for on-board object detection and recognition tasks, aiming at reducing human annotation effort.*

1.4 PhD Objective and Outline

Once established human annotation as one of the bottlenecks for developing autonomous driving systems, we can state that **the overall goal of this PhD is to study methods for leveraging synthetic data to create autonomous driving perception systems**, under different practical settings. We are going to focus on object detection and recognition, since these are key functionalities from Level 1 to Level 5

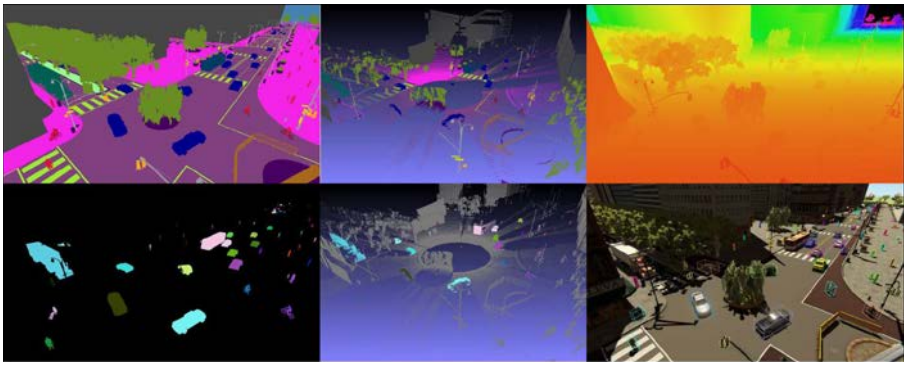

Figure 1.5 – Aerial view ground truth of the same synthetic scene based on the SYNTHIA dataset [123]. Left column: semantic segmentation (top) and instance segmentation (bottom) from a synthetic image. Mid column: analogous for LiDAR 3D pointclouds. Right column: color-coded depth (top) and 3D Bounding boxes (bottom) for vehicles/pedestrians/motorbikes/bicyclists shown on top of the synthetic image used in all these examples.

autonomy. Moreover, since cameras and LiDARs are going to be key in high levels of autonomy, we address work based on both. More specifically, in this PhD we address different research questions organized per chapter as follows.

Chapter 2. On-board vision systems may need to increase the number of classes that can be recognized in a relatively short period of time. For instance, a traffic sign recognition (TSR) system may suddenly be required to recognize new signs (classes). Since collecting and annotating samples of such new classes may need more time than we have, especially for uncommon signs, we propose to generate them from synthetic images. However, since the new version of the TSR system needs to perform on real-world images, we must expect a problem of domain shift. In this context, the chapter focuses on two arising questions: (Q1) Can we reduce this domain shift by applying an image-to-image translation based on a generative adversarial network (GAN) to the samples of the unknown classes, provided that such a GAN was trained only with samples of the known classes? note that TSR is a fine-grain classification task, so a priori it is unclear how GAN-based image-to-image translation can perform in totally unseen traffic signs; and (Q2) What are the overall classification results when training the TSR system using the real-world data of the known classes with the data generated for the new classes following this GAN-based proposal? In order to answer these questions, we present an extensive set of experiments. We also contribute with a synthetic dataset designed to support

this research.

Chapter 3. We assess co-training as a semi-supervised learning method for self-annotating objects in raw images, so reducing the human-annotation effort for developing deep object detectors; in particular, for vehicles and pedestrians. Our study pays special attention to a scenario involving domain shift; in particular, when we have automatically generated virtual-world images with object bounding boxes and we have real-world images which are not annotated. Accordingly, we also study the combination of co-training with GAN-based image-to-image translation. We also contribute with a synthetic dataset designed to support this research.

Chapter 4. In previous chapters, we address vision-based object recognition and detection problems. In this chapter, we focus on LiDAR-based object detection, also from the point of view of leveraging synthetic data to alleviate manual annotation. In this case, data refer to 3D pointclouds coming from LiDAR scans, thus manual annotation is especially cumbersome. Prior to start this research, we expected that the synth-to-real domain shift for LiDAR data would not be comparable to the case of images. The reason is that the same scene captured by different camera sensors and/or acquisition settings may look very different (*i.e.*, even producing domain shift among the cameras), while LiDARs provide accurate depth information which one may think to be more sensor/setting invariant. However, our experiments on LiDAR-based object detection soon revealed that it is not the case. When training CNNs for LiDAR-based object detection, there is a significant domain shift if the testing 3D pointclouds come from different LiDAR sensors and/or acquisition settings. In this context, there is a lack of GAN-based cloud-to-cloud translation operating on typical AV's data. This chapter focuses on filling in this gap. We also contribute with a synthetic dataset designed to support this research.

We have written these chapters to be self-contained, following the usual structure of a paper, *i.e.* abstract, introduction, related work, propose method, experimental work with associated discussion, and summarizing conclusions. Chapter 5 summarizes the main contributions of this PhD and draws lines of continuation. Finally, we have added Appendices A and B, the former providing details about the generation of the datasets generated for this PhD, the latter listing the publications done while working in this PhD.

2 Recognizing New Traffic Signs with Synthetic Data in the Loop

On-board vision systems may need to increase the number of classes that can be recognized in a relatively short period. For instance, a traffic sign recognition system may suddenly be required to recognize new signs. Since collecting and annotating samples of such new classes may need more time than we wish, especially for uncommon signs, we propose a method to generate these samples by combining synthetic images and Generative Adversarial Network (GAN) technology. In particular, the GAN is trained on synthetic and real-world samples from known classes to perform synthetic-to-real domain adaptation, but applied to synthetic samples of the new classes. Using the Tsinghua dataset with a synthetic counterpart, SYNTHIA-TS, we have run an extensive set of experiments. The results show that the proposed method is indeed effective, provided that we use a proper Convolutional Neural Network (CNN) to perform the traffic sign recognition (classification) task as well as a proper GAN to transform the synthetic images. Here, a ResNet101-based classifier and domain adaptation based on CycleGAN performed extremely well for a ratio $\sim 1/4$ for new/known classes; even for more challenging ratios such as $\sim 4/1$, the results are also very positive.

2.1 Introduction

On-board computer vision is fundamental to perceive the traffic environment around the ego-vehicle and, therefore, a crucial technology for assisting drivers or enabling fully autonomous vehicles (AVs). In the last decade, computer vision has been empowered by the use of Convolutional Neural Networks (CNNs), which allow to extract very detailed meaningful information from raw images. Over the last few years, we have witnessed unprecedented breakthroughs in tasks such as image level classification [78, 130], bounding box level 2D and 3D object detection [23, 92, 102, 118, 119], pixel-wise class and instance segmentation [12, 90, 95, 105, 151, 152, 180], skeleton-wise human pose estimation [22, 85, 141], even dense monocular depth estimation [6, 42, 45, 50, 56, 57, 113]; all of them, among others, essential visual

capabilities for high levels of driving automation or assistance.

Indeed, CNNs have become very accurate models provided there is sufficient data (in size and diversity) for their training [64, 140]; where *data* refer to both the raw images and the ground truth (GT) that we must associate to them as training supervision. In fact, since CNNs are data hungry and most GT comes from a (cumbersome and prone to errors) manual labelling process, providing GT at scale, reducing manual intervention, and/or reusing previous knowledge have become emerging topics for computer vision research in general, and for autonomous driving in particular. For instance, *active learning* techniques [1, 3, 125, 128] focus on automatically finding the a priori best training images for their posterior manual labelling, out of a large amount of unlabelled ones; *self-labeling* techniques [171, 173, 190] focus on progressively self-labelling images by refining increasingly more accurate models; *transfer learning* techniques [48, 108] focus on reusing existing models for re-training them to perform new tasks, so that only labels for such tasks are required; *domain adaptation* techniques [32, 66, 159, 183] focus on reusing exiting models in new domains, minimizing the labeling effort required in the new domains; while *self-supervision* [47, 74, 77, 174] focuses on learning visual models without manual labeling, with the support of auxiliary simple (pretext) tasks for which it is possible to automatically define self-labels. Overall, all these research topics are evidence that, due to CNNs, computer vision and its applications have become strongly data-dependent.

In this context, there can appear situations where a sudden lack of data can seriously limit the operational capabilities of a computer vision system. For instance, imagine an AV or a driver assistance system that must *detect* (*i.e.* localize within an image) and *recognize/classify* (*i.e.* assign a specific meaning) traffic signs by on-board computer vision. While detection itself can be very robust due to the stability of the shape of traffic signs across the world (triangles, rectangles, circles), it can happen that the vehicles of a fleet deployed in a new world area suddenly detect traffic signs that they cannot recognize. These vehicles can automatically broadcast the unknown traffic signs to the company for opening an incidence. If the company decides that it is required to actually recognize the new traffic signs, it may take too much time to collect sufficient samples and perform their proper labelling, specially if the samples must be acquired under a variety of environmental conditions and viewpoints, which can be challenging if such new traffic signs are scarce (*i.e.* they are rare events). While this is just an illustrative example, we can imagine analogous situations for robots manipulating goods, traffic surveillance systems separating vehicles according to their functionality, etc.; readers can imagine other examples where perception-based systems may have such a kind of *unknown class* problem, after all, both CNNs as well as traditional (shallow) vision models work under the *closed-world* assumption.

Following a new tendency during the last years [38, 44, 101, 121, 123, 129], in this chapter we propose to explore how synthetic data can help in such situations. In particular, assuming that we have a few samples of an unknown class (following with the example, these can be traffic sign images reported by the fleet), we propose a method consisting of the following steps: (1) to elaborate 3D graphical instances of the class and automatically place them all around in a virtual environment; (2) by varying the simulation conditions within the virtual environment, to automatically generate as many image samples as needed; (3) to domain-adapt these samples to close the visual gap with real-world images; (4) to retrain the corresponding CNNs for recognizing the new class. Then, the research question is how these results would differ from the results obtained by a counterpart procedure consisting on capturing samples of the new class with real-world cameras and labelling them by hand (*i.e.* instead of the steps (1)-(3)). Note that (3) must be a task-agnostic domain adaptation step. In particular, we propose to use a Generative Adversarial Network (GAN) [53], previously trained in a generic manner to transform our synthetic images to look like real-world images captured by the cameras of our computer vision system. We remark that such a GAN has never seen before samples of the unknown class. As a very important and challenging use case, we assess the success of our proposal by applying it to traffic sign recognition in the wild [187], which can be seen as a fine-grain image classification task. We will elaborate exhaustive experiments varying the known and unknown traffic signs according to different criteria, not only for training/retraining the classification CNNs but also the task-agnostic GAN. In order to support this experimental part, we have created a synthetic dataset of traffic signs using an evolution of our SYNTHIA environment[1] [123], which we call SYNTHIA-TS. Using the Tsinghua dataset [187] and SYNTHIA-TS, we have run an extensive set of experiments which show that the proposed method is indeed effective, provided we use a proper CNN to perform the traffic sign classification task as well as a proper GAN to transform the synthetic images. Here, a ResNet101-based classifier and a CycleGAN performed extremely well for a ratio $\sim 1/4$ for new/known classes, and even for more challenging ratios such as $\sim 4/1$ the results are also very positive. Therefore, synthesizing data following our proposal establishes a proper methodology to minimize the lack of real-world labelled data when a computer vision system must be retrained to recognize new classes in a relatively short term.

The rest of the chapter is organized as follows. Section 2.3 elaborates our proposal. Section 2.4 details the experimental protocol, the obtained results and the conclusions deduced from them. Finally, section 2.5 summarizes the work presented in this chapter and suggests future directions of research in line with our conclusions.

[1] We plan to publicly release this rendered data as well as its counterparts transformed by GANs

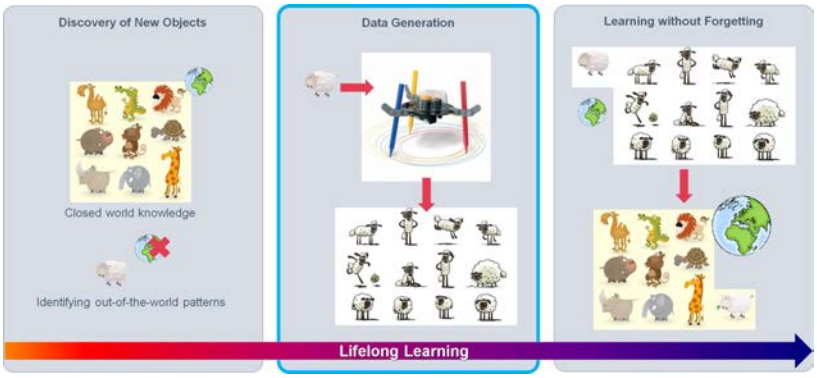

Figure 2.1 – Lifelong learning setting. First, unknown classes are identified. Then, if we want to consider them in the future, we must collect diverse samples of these classes for posterior model retraining. Finally, we retrain the models to recognize the new classes, without forgetting previous ones. In this chapter, we focus on the second step, assuming that rather than collecting the samples from the real world, we generate them by using a virtual world.

2.2 Related work

Learning to recognize new classes falls into the paradigm of *lifelong learning* [10, 33, 109], where a perception-based system has to continuously adapt to situations not previously experienced. We can think of three main components of such a learning capability (see Fig. 2.1). The first one consists in the ability to identify unknown classes, which is not trivial since CNNs tend to be overconfident about their classification decisions. In fact, this is an active research topic known as *out-of-distribution detection* (which includes *novelty* and *anomaly* detection; this chapter relates to the novelty case) [24,87,99,114]. The second component consists in a data generation protocol to collect training samples of these unknown classes provided the CNN needs to take them into account in the future. Finally, the third component consists in a procedure that allows the CNN to recognize the new classes, without deteriorating its accuracy identifying the classes for which it was initially/previously trained; this is known as learning without forgetting [5, 76, 86, 93] and still is an open and challenging research topic. In fact, [99] focuses on out-of-distribution for our use case, *i.e.* traffic sign recognition.

In this chapter, we focus on data generation. We require that we are given the unknown (novel) traffic signs in the form of a few on-board captured images. Also,

to avoid the forgetting problem when retraining the traffic sign recognition CNNs, we will just retrain using all the available data (known and synthesized); in this way, the chapter can really focus on assessing the usefulness of synthesizing samples of the unknown classes. Accordingly, the remaining of this section addresses the use of synthesized visual data for training computer vision models, as well as the use of GANs to perform task-agnostic domain adaptation.

Researchers such as Taylor *et al.* [145] pioneered the use of videogame data for testing vision-based tracking algorithms. Marin *et al.* [98] extended the use of this synthetic data to train object detectors performing in real images, while Vazquez *et al.* [156] raised the attention on the domain gap between virtual and real world images. From there, the use of synthetic visual data generated from virtual environments has kept growing. We find works using synthetic data for object detection/recognition [62, 110, 111, 172], object viewpoint recognition [139], re-identification [14], and human pose estimation [132]; building synthetic cities for autonomous driving tasks such as semantic segmentation [59, 123], place recognition [135], object tracking [44, 103], object detection [72, 147], stixels computation [63], and benchmarking different on-board computer vision tasks [121]; building indoor scenes for semantic segmentation [61], as well as normal and depth estimation [71]; generating GT for optical flow, scene flow and disparity [20, 100]; generating augmented reality images to support object detection [4]; simulating adverse atmospheric conditions such as rain or fog [11, 126]; even performing procedural generation of videos for human action recognition [137, 154]. Moreover, since robotics and autonomous driving rely on sensorimotor models worth to be trained and tested dynamically, in the last years the use of simulators has been intensified beyond datasets [27, 38, 127, 129].

In contrast to this literature, we can leverage from already annotated real-world images conveying a set of classes known by our current CNN-based classifier, but we have to assess the possibility of using automatically generated synthetic images as samples of classes that are unknown for our current CNN. Therefore, these synthetic images must be used to retrain the CNN to properly classify previous and new classes. We will evaluate two different settings. First, when the synthetic images are used as they come from the virtual environment. Second, when the synthetic images are transformed by a GAN to look like the real-world images, *i.e.* as a type of task-agnostic domain adaptation; where, following the domain adaptation terminology, the synthetic world acts as the *source* domain and the real world as the *target* domain.

GANs use a *generator* CNN to transform the appearance of source images to look like target images, and a *discriminator* CNN which aims at distinguishing between the transformed and the original images in the target domain. The generator-discriminator system is trained until the discriminator is not able to distinguish the

origin of the images, which is understood as the point when the source images are similar enough to the target ones. This application of GANs is known as *image-to-image translation.*

Isola *et al.* [107] proposed an encoder-decoder as generator architecture, and a patch-based (patchGAN) approach for the discriminator. Since this approach was only able to work with low resolution images, other approaches build upon this method to overcome this problem [161]. However, a relevant observation is that these proposals require pixel-level GT about how the generated images should look like, which is termed as supervised image-to-image translation. In order to avoid this kind of supervision, Taigman *et al.* [144] designed an encoder-decoder generator in such a way that the encoder features are indistinguishable for original and transformed images. In other words, for the GT it is only required to know if the images come from the source domain or the target one, which is always possible at training time. Liu *et al.* [89] also focused on generators' feature layers. Afterwards, other alternatives were proposed that did not require the mentioned supervision. Some approaches use an auxiliary task to define the loss between input and generated images; for instance, Bousmalis *et al.* [18] use image-level classification while Hoffman *et al.* [65] use semantic segmentation as auxiliary tasks. Other approaches focus on appearance of the input and generated images. Shrivastava *et al.* [133] proposed an identity loss between the input and generated images. One restriction of this approach is that source and target domain images have similar appearance. Zhu *et al.* [186] and Kim *et al.* [148] followed the cycle idea, *i.e.* from source images target-style ones are generate which, in turn, are the input to generate new source-style images. The source-to-target and the target-to-source are different generators. In each domain, we have different discriminators. The cycle idea is not only useful because it does not require image GT, but also because the input and transformed images can have a relatively different appearance, especially compared to the approach in [133]. In other words, in contrast to other GAN proposals, a GAN trained according to the cycle idea has the potential of properly transforming the appearance of source images showing content unseen during its training. Accordingly, in this chapter we follow the cycle idea. In particular, since [186] has publicly available code, called CycleGAN, we use it for the experiments in this chapter.

Finally, the work with the most similar goal to this chapter, has been recently presented by Beery *et al.* [16]. The addressed application is animal detection and classification from static cameras. The chapter evaluates the use of synthetic data for classifying animals for which it is difficult to have sufficient real-world image samples. Therefore, similarly to us, previous real-world image samples from known classes (animals) are leveraged for retraining their (animal) classifier together with the synthesized images containing the new class samples (they consider deers as a new class). In this chapter, rather than focusing on one new class at a time, we

evaluate also different balances between known and unknown classes. We also evaluate the difference between using the synthetic images as they come from the virtual environment, in contrast to transforming them via GANs. In both cases, since our application falls into fine-grain classification, we assess also the dependency on common visual cues between seen and unseen classes.

2.3 Method

2.3.1 Overall idea

Assume we need a classifier $\mathscr{C}$ such that, given an image (*e.g.* framing a traffic sign), it is able to assign to it a right label from a given set $\mathscr{K}$ of known labels/classes (*e.g.* traffic sign classes). Let $\mathscr{I}$ be a set of images collected for training such a $\mathscr{C}$. For supervised training, we need to assign one class to each image, which is usually done offline by human annotators. Let $\mathscr{I}_{\mathscr{K}}$ be the corresponding annotated set of images. Then, we can run a supervised machine learning algorithm that uses $\mathscr{I}_{\mathscr{K}}$ to generate a classifier $\mathscr{C}_{\mathscr{I}_{\mathscr{K}}}$, which will be used (at testing time) to support the addressed application (*e.g.* on-board traffic sign recognition). The problem arises when, during the execution of such application, we realise that there are classes of interest not included in $\mathscr{K}$ (*e.g.* after a warning from an out-of-distribution detection module also running as part of the application). Let's call $\mathscr{U}$ this set of new classes such that $\mathscr{K} \cap \mathscr{U} = \emptyset$. For training a new supervised classifier $\mathscr{C}_{\mathscr{I}_{\mathscr{K} \cup \mathscr{U}}}$, which takes into account all classes, we need to collect and manually annotate new images covering a sufficient amount of instances for each class in $\mathscr{U}$. This may be difficult to do as quickly as we could wish, since we may be facing unusual classes (*i.e.* for which it is difficult to find corresponding instances by just randomly roaming in the real world) and it will also be a latency due to the manual annotation of the found instances.

Accordingly, the alternative method that we want to explore, relies on automatically generating synthetic images to quickly obtain sufficient annotated instances of the new classes for training a new classifier. In addition, as we have already mentioned, training visual models using pure synthetic images can lead to a performance drop when performing in the real world. In order to reduce such a domain gap, GANs are a possible solution; *i.e.* by directly transforming the images of the source domain (*e.g.* synthetic world) to have a similar appearance to those of the target domain (*e.g.* real world). We follow this approach in this study.

Now, let $\widetilde{\mathscr{I}}_{\mathscr{U}}$ be a set of synthetically generated images automatically annotated according to the classes in $\mathscr{U}$, and $\mathscr{G}^{\widetilde{\mathscr{I}}_{\mathscr{U}}}$ the corresponding set of images transformed by a GAN. We aim to train a classifier $\mathscr{C}_{\{\mathscr{G}^{\widetilde{\mathscr{I}}_{\mathscr{U}}}, \mathscr{I}_{\mathscr{K}}\}}$ (ideally) performing in the real

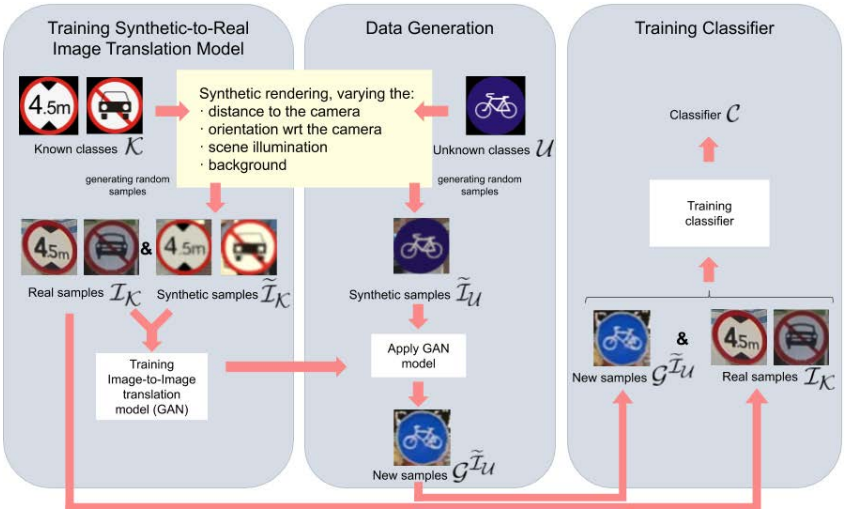

Figure 2.2 – Proposed method for retraining a classifier, $\mathscr{C}$, to keep detecting previously known classes ($\mathscr{K}$) for which we have labeled real-world and, in addition, new previously unknown classes ($\mathscr{U}$) for which we do not have real-world samples. The key idea is to have synthetic samples for both the known and new classes. The real-world samples of the known classes ($\mathscr{I}_\mathscr{K}$) and the synthetic ones ($\tilde{\mathscr{I}}_\mathscr{K}$) are used to train a GAN with the aim of performing synthetic-to-real domain adaptation. In particular, the synthetic samples of the new classes ($\tilde{\mathscr{I}}_\mathscr{U}$) are transformed ($\mathscr{G}^{\tilde{\mathscr{I}}_\mathscr{U}}$) by this GAN. Then, the real-world samples of the previously known classes and the transformed synthetic samples of the new classes, are used to train the desired classifier. The overall idea is illustrated for traffic sign recognition.

world as if $\mathscr{G}^{\tilde{\mathscr{I}}_\mathscr{U}}$ would consist of real-world images annotated by humans. Yet another question is how the GAN is trained. From the point of view of generating synthetic images, it is analogous to generate images for classes in $\mathscr{U}$ as for classes in $\mathscr{K}$. Therefore, we require that besides the set of real-world images $\mathscr{I}_\mathscr{K}$, we also have a set $\tilde{\mathscr{I}}_\mathscr{K}$ of (automatically) annotated synthetic images for the known classes; *i.e.* both sets cover the same classes but for each class it can be a different number of samples in each set. The GAN is trained to transform images from the set $\tilde{\mathscr{I}}_\mathscr{K}$ into the set $\mathscr{I}_\mathscr{K}$, but without assuming a one to one pairing of the images from both sets. In other words, the GAN will learn to perform domain-to-domain transformations, but not class-specific transformations between domains. Therefore, when we need to transform synthetically generated instances for a new previously unknown class

(*i.e.* in $\mathscr{U}$), we can apply the previous learned GAN even if it was not exposed to such class during training time and, in fact, it will not be exposed at this time due to the lack of real-world instances of such class. Figure 2.2 depicts the overall idea.

2.3.2 Data generation

We start by generating synthetic images with automatic GT for each unknown class. We require to have a real-world example showing the appearance of an instance of each unknown class (*i.e.* the example already used to decide that the class must be considered in future versions of the classifier). Then, a designer can create a textured 3D model of it. Such model can then be populated in a virtual environment that we have predefined. Next, we can capture as many images as we need containing instances of the new class along with automatic GT, which is done under predefined variations regarding the environmental and image capture conditions. For instance, for the traffic sign recognition study we address in this chapter, we perform the following steps: (1) we create a traffic sign 3D model for a given unknown sign; (2) we use the SYNTHIA environment [123] to populate the 3D model in locations predefined for traffic signs; (3) we automatically aim the camera that captures images towards these locations varying the capturing angle and distance between the camera and the traffic sign, as well as the scene illumination. This procedure ensures visual variability in the collected images due to the fact that environmental shadows influence the captures, as well as global illumination, resolution, etc. The same procedure can be used to capture synthetic images of known classes intended to be used in the training of the domain-adaptation GAN. In the case of the traffic signs, using the pixel-wise semantic segmentation GT provided by the virtual environment (SYNTHIA), we create corresponding 2D bounding boxes, which we crop to obtain the final synthetic image samples.

As we have mentioned before, synthetic images depicting instances of new classes must be still transformed by means of a GAN in order to alleviate domain shift effects. With this aim we have used the publicly available implementation of CycleGAN as detailed in [186]; which we train using images of known classes taken from the synthetic and real-world domains. The adversarial loss aiming at approaching the appearance of synthetic and real-world images is defined as follows:

$$\mathscr{L}_{adv}(G_{A\rightarrow B}, D_B, \mathscr{I}^A) = \mathbb{E}_{i\sim\mathscr{I}^A}[log(D_B(G_{A\rightarrow B}(i)))] \ , \tag{2.1}$$

where, A and B are different domains (synthetic or real in our case), $G_{A\rightarrow B}$ refers to the GAN generator from domain A to domain B, D_B refers to the GAN discriminator that distinguish between images really coming from domain B from those output by

$G_{A\to B}$, and $\mathcal{I}^A$ is a set of images from domain A. The GAN discriminator is trained according to the following loss:

$$\begin{aligned}\mathcal{L}_{disc}(G_{A\to B}, D_B, \mathcal{I}^A, \mathcal{I}^B) = \mathbb{E}_{i^B\sim\mathcal{I}^B}[log(D_B(i^B))] \\ + \mathbb{E}_{i^A\sim\mathcal{I}^A}[log(1 - D_B(G_{A\to B}(i^A)))] \ .\end{aligned} \tag{2.2}$$

In addition, CycleGAN uses additional losses to force that image appearance is transformed between domains without effecting the semantic content of the transformed images. In particular, the following cyclical reconstruction loss is used:

$$\mathcal{L}_{rec}(G_{A\to B}, G_{B\to A}, \mathcal{I}^A) = \mathbb{E}_{i\sim\mathcal{I}^A}[\|G_{B\to A}(G_{A\to B}(i)) - i\|_1] \ , \tag{2.3}$$

which is complemented (regularized) with an additional loss aiming at not only ensuring in-domain content reconstruction but also across-domain content similarity:

$$\mathcal{L}_{sim}(G_{A\to B}, \mathcal{I}^A) = \mathbb{E}_{i\sim\mathcal{I}^A}[\|G_{A\to B}(i) - i\|_1] \ . \tag{2.4}$$

Now, we can define the total loss function to train the GAN generator that transform images from a synthetic domain $\mathcal{S}$ to a real domain $\mathcal{R}$ as follows:

$$\begin{aligned}\mathcal{L}(G_{\mathcal{S}\to\mathcal{R}}, G_{\mathcal{R}\to\mathcal{S}}, D_{\mathcal{S}}, D_{\mathcal{R}}, \mathcal{I}^{\mathcal{S}}, \mathcal{I}^{\mathcal{R}}) = \mathcal{L}_{adv}(G_{\mathcal{S}\to\mathcal{R}}, D_{\mathcal{R}}, \mathcal{I}^{\mathcal{S}}) \\ + \mathcal{L}_{adv}(G_{\mathcal{R}\to\mathcal{S}}, D_{\mathcal{S}}, \mathcal{I}^{\mathcal{R}}) \\ + \mathcal{L}_{disc}(G_{\mathcal{S}\to\mathcal{R}}, D_{\mathcal{R}}, \mathcal{I}^{\mathcal{S}}, \mathcal{I}^{\mathcal{R}}) \\ + \mathcal{L}_{disc}(G_{\mathcal{R}\to\mathcal{S}}, D_{\mathcal{S}}, \mathcal{I}^{\mathcal{R}}, \mathcal{I}^{\mathcal{S}}) \\ + \mathcal{L}_{sim}(G_{\mathcal{S}\to\mathcal{R}}, \mathcal{I}^{\mathcal{S}}) \\ + \mathcal{L}_{sim}(G_{\mathcal{R}\to\mathcal{S}}, \mathcal{I}^{\mathcal{R}}) \\ + \mathcal{L}_{rec}(G_{\mathcal{S}\to\mathcal{R}}, G_{\mathcal{R}\to\mathcal{S}}, \mathcal{I}^{\mathcal{S}}) \\ + \mathcal{L}_{rec}(G_{\mathcal{R}\to\mathcal{S}}, G_{\mathcal{S}\to\mathcal{R}}, \mathcal{I}^{\mathcal{R}}) \ .\end{aligned} \tag{2.5}$$

At training time, we use $\mathcal{I}_{\mathcal{K}}$ and $\widetilde{\mathcal{I}}_{\mathcal{K}}$ as $\mathcal{I}^{\mathcal{R}}$ and $\mathcal{I}^{\mathcal{S}}$ image sets, respectively. Then, the learned generator $G_{\mathcal{S}\to\mathcal{R}}$ will be the CNN that we use to transform a set of synthetic images $\widetilde{\mathcal{I}}_{\mathcal{U}}$ into $\mathcal{G}^{\widetilde{\mathcal{I}}_{\mathcal{U}}}$.

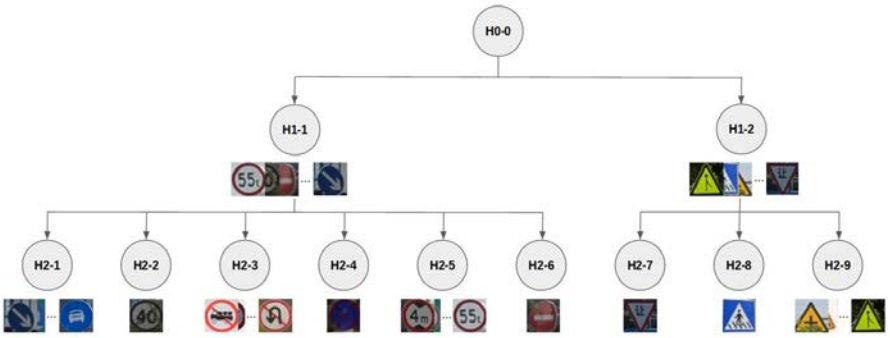

Figure 2.3 – Hierarchy of Tsinghua traffic signs.

2.4 Experimental results

The experiments have been designed to address two questions. Since we use synthetically generated instances of unknown classes to retrain the current classifier, we will have a domain shift problem. **(Q1)** *Can we reduce this domain shift by applying an image-to-image translation GAN to the samples of the unknown classes, provided such a GAN was trained only with samples of the known classes?*, and **(Q2)** *What are the overall classification results when training the classifier using the real-world data of the known classes with the data generated for the new classes following such GAN-based proposal?*

Note that question **Q1** focuses on classification results in terms of new classes in isolation, while **Q2** addresses the ultimate question, since we combine real-world samples form known classes with generated samples from unknown classes for training the all-classes classifier. In the following, section 2.4.1 introduces the synthetic and real-world datasets used in our experiments, and section 2.4.2 elaborates the designed experiments to answer these questions along with the obtained results and corresponding discussion.

2.4.1 Datasets

In order to perform our experiments, we need a dataset based on real-world images of traffic signs as well as another based on synthetic images. We have selected the widely used Tsinghua traffic sign dataset [187], and a synthetic analogous that we have created to perform the research in this chapter, which we call SYNTHIA-TS. We briefly describe them in the following.

Tsinghua is a dataset composed of outdoor scenes captured in China while driving a car in urban scenarios. Following the approach proposed in [187], we have cropped the traffic signs and removed all the classes with less than 100 samples. The resulting dataset is composed of 21,721 cropped images, representing 42 traffic sign classes. In terms of appearance, these classes can be hierarchically organized as shown in Fig. 2.3, where the first criterion to split the dataset is the external shape of the traffic signs, and the second is the textual/graphical content of the signs. Both shape and content define the semantics of each traffic sign, *i.e.* the class.

SYNTHIA-TS has been created by mimicking the 42 classes considered from the Tsinghua dataset, using one textured 3D model per each of those classes. Then, following the protocol explained in section 2.3.2, we have acquired traffic sign images within the SYNTHIA environment (refer to appendix A to see more details). The generated data are balanced for all image acquisition conditions and classes. We have generated 23,222 instances in total, covering the 42 classes. Since the SYNTHIA environment was previously created for multiple purposes, obtaining these instances from it took less than 2h using a desktop PC based on an INTEL Core i7 CPU and one NVIDIA Geforce GTX 1080 GPU.

2.4.2 Experiments: design, results, and discussion

We have not only considered the Tsinghua and SYNTHIA-TS datasets as a whole, *i.e.* H0-0 in terms of the hierarchy shown in Fig. 2.3; instead, in order to perform a finer grained analysis regarding questions **Q1** and **Q2**, we have also conducted experiments based on different nodes of such hierarchy, which we call *splits*. Accordingly, our setup assumes that we have an existing split s_1 of real-world annotated images for training, and that we want to learn also a new split s_2, for which we have no access to a proper amount of corresponding real-world images and, therefore, we have to synthesize them. It is understood that s_1 and s_2 have no intersection between classes. On the other hand, for the purpose of performing comparative evaluations in our experimental setting, in fact we do have access to the real-world annotated images of split s_2.

Table 2.1 – Splits of Fig. 2.3 used in appearance-driven experiments. Tsinghua samples are divided as training and testing tasks (details in main text). SYNTHIA-TS samples are used in training tasks only.

Split code	Num. classes	Samples Tsinghua	Samples SYN.-TS	Content
H0-0	42	21,721	23,222	All the traffic sign classes are considered
H1-1	35	20,256	19,507	Only circular traffic signs
H1-2	7	1,465	3,773	Only non-circular traffic signs
H2-1	6	3,713	3,248	Mandatory actions (circular, blue backg., white border).
H2-3	10	4,012	5,885	Prohibition actions (circular, white backg., red border & diag. line)
H2-5	16	7,316	8,896	Information (circular, white backg., with red border)
H2-9	5	989	2,713	Working areas (triangular, yellow backg., black border)

Table 2.2 – Basic notation for data subsets.

Notation	Description
$\mathscr{S}_s^T$	Synthetic training data from split s
$\mathscr{T}_s^T$	Tsinghua training data in split s
$\mathscr{G}^{s_2}{}_{s_1}$	Dataset $\mathscr{G}^{s_2}$ generated by applying a GAN trained on splits $\mathscr{S}_{s_1}^T$ & $\mathscr{T}_{s_1}^T$ to split $\mathscr{S}_{s_2}^T$
$\mathscr{T}_s^C$	Tsinghua testing (classification) data from split s

Table 2.3 – Lower and upper bounds for traffic sign classification on $\mathcal{T}^C_{H0-0}$. Average and standard deviation F1 score for five training-testing runs are shown. The lower bound corresponds to training only with synthetic data ($\mathcal{S}^T_{H0-0}$), while the upper bound corresponds to training with real data ($\mathcal{T}^T_{H0-0}$).

Training set	VGG16	ResNet101
$\mathcal{S}^T_{H0-0}$	36.05 ± 3.92	58.74 ± 2.04
$\mathcal{T}^T_{H0-0}$	97.59 ± 0.15	98.76 ± 0.14

Since, we will be referring to splits coming from synthetic and real-world data, the former sometimes transformed by a GAN, and the later sometimes used as training or testing data, we have defined the compact notation of Table 2.2, which will allow us to be precise and concise when describing the multiple experiments we are reporting in this section. Using this notation and given two splits s_1 and s_2, an example of an experiment for **Q2** would consist in using $\mathcal{T}^T_{s_1}$ and $\mathcal{S}^T_{s_2}$ to train a traffic sign classifier for the known classes in split s_1 together with the new classes in split s_2, which we would like to be accurate when testing in $\mathcal{T}^C_{s_1 \cup s_2}$, *i.e.* accurate for all classes. Alternatively, if we use a GAN to transform the synthetic images, then the training of the classifier would be done with $\mathcal{T}^T_{s_1}$ and $\mathcal{G}^{s_2}{}_{s_1}$. In fact, we transformed all the synthetic images at once, which took less than 1.5h using a desktop PC based on an INTEL Core i7 CPU and one NVIDIA GeForce TITAN X Pascal GPU.

As we can see in Fig. 2.3 we have 3 hierarchical levels: 1) the whole data, 2) two splits based on external shape, and 3) given a shape, different data based on content. Each considered split is defined on Table 2.1 specifying their features. We are not considering splits with only one class (*i.e.* H2-2, H2-4, H2-6, H2-7 and H2-8) since they would not allow to address **Q1** (for which at least two classes are needed). However, note that although these splits are not considered in isolation, their data are considered when working with a split corresponding to their parent nodes in the hierarchy.

Now, we start the experiments by establishing the upper and lower bounds of different traffic sign classifiers. In these experiments, we use the full Tsinghua and SYNTHIA-TS datasets. Therefore, in this case, we use the split H0-0 (Fig. 2.3) for Tsinghua, *i.e.* we use $\mathcal{T}^T_{H0-0}$ and $\mathcal{T}^C_{H0-0}$ for training and testing, respectively. Both sets have samples of all the traffic signs we are considering. More specifically, for each class, 60% of the samples are used for training tasks (CycleGAN and traffic sign classifiers) and the remaining 40% for testing traffic sign classifiers. The per-class training/testing sampling is performed randomly and once. Training on $\mathcal{S}^T_{H0-0}$ and testing on $\mathcal{T}^C_{H0-0}$ acts as lower bound, since we are using only synthetic

Table 2.4 – Experiments to support **Q1** (see main text). All tests are done in $\mathscr{T}^{C}_{s_u}$. Average and standard deviation of F1 score are reported since each experiment is performed five times. The column $\mathscr{G}^{s_u}{}_{s_k} - \mathscr{S}^{T}_{s_u}$ just stands for the subtraction of the means of the respective columns.

Known classes (s_k)	Unknown classes (s_u)	VGG16 $\mathscr{S}^{T}_{s_u}$	VGG16 $\mathscr{G}^{s_u}{}_{s_k}$	VGG16 $\mathscr{G}^{s_u}{}_{s_k}$ - $\mathscr{S}^{T}_{s_u}$	VGG16 $\mathscr{G}^{s_u}{}_{s_u}$	VGG16 $\mathscr{T}^{T}_{s_u}$
H1-2	H1-1	39.20 ± 2.99	49.88 ± 3.68	10.68	74.55 ± 2.67	97.44 ± 0.23
H1-1	H1-2	65.44 ± 4.39	74.65 ± 4.71	09.21	94.35 ± 0.73	94.23 ± 6.11
H2-3			65.45 ± 4.05	-04.84		
H2-5	H2-1	70.29 ± 5.31	65.91 ± 9.28	-04.38	90.81 ± 0.71	99.22 ± 0.28
H2-9			74.01 ± 7.71	03.72		
H2-1			39.10 ± 2.08	-02.93		
H2-5	H2-3	42.03 ± 4.44	62.87 ± 4.11	20.84	88.15 ± 2.37	97.29 ± 0.44
H2-9			55.92 ± 0.72	13.89		
H2-1			54.51 ± 1.33	01.43		
H2-3	H2-5	53.08 ± 7.75	60.85 ± 4.90	07.77	85.25 ± 1.48	97.35 ± 0.25
H2-9			61.59 ± 2.88	08.51		
H2-1			72.51 ± 5.98	02.19		
H2-3	H2-9	70.32 ± 5.90	80.35 ± 4.09	10.03	94.77 ± 1.03	91.81 ± 7.60
H2-5			76.82 ± 4.12	06.50		

Table 2.5 – Experiments to support **Q1** analogous to Table 2.4, but using ResNet101.

Known classes (s_k)	Unknown classes (s_u)	ResNet101 $\mathscr{S}^{T}_{s_u}$	ResNet101 $\mathscr{G}^{s_u}{}_{s_k}$	ResNet101 $\mathscr{G}^{s_u}{}_{s_k}$ - $\mathscr{S}^{T}_{s_u}$	ResNet101 $\mathscr{G}^{s_u}{}_{s_u}$	ResNet101 $\mathscr{T}^{T}_{s_u}$
H1-2	H1-1	58.44 ± 1.92	63.00 ± 2.15	04.56	84.35 ± 1.08	98.70 ± 0.24
H1-1	H1-2	79.20 ± 4.06	88.80 ± 0.52	09.60	92.06 ± 1.66	96.26 ± 0.83
H2-3			76.66 ± 1.59	10.27		
H2-5	H2-1	66.39 ± 2.05	72.65 ± 2.90	06.26	89.71 ± 0.88	99.42 ± 0.13
H2-9			78.47 ± 3.26	12.08		
H2-1			48.12 ± 4.61	-07.01		
H2-5	H2-3	55.13 ± 2.81	68.86 ± 2.83	13.73	87.99 ± 2.86	97.84 ± 0.42
H2-9			52.70 ± 2.82	-02.43		
H2-1			58.73 ± 3.16	-02.49		
H2-3	H2-5	61.22 ± 1.87	63.61 ± 3.37	02.39	82.52 ± 0.76	97.08 ± 0.26
H2-9			65.11 ± 2.09	03.89		
H2-1			67.62 ± 5.63	-02.83		
H2-3	H2-9	70.45 ± 4.78	82.05 ± 2.29	11.60	92.83 ± 0.91	90.18 ± 0.91
H2-5			83.42 ± 2.10	12.97		

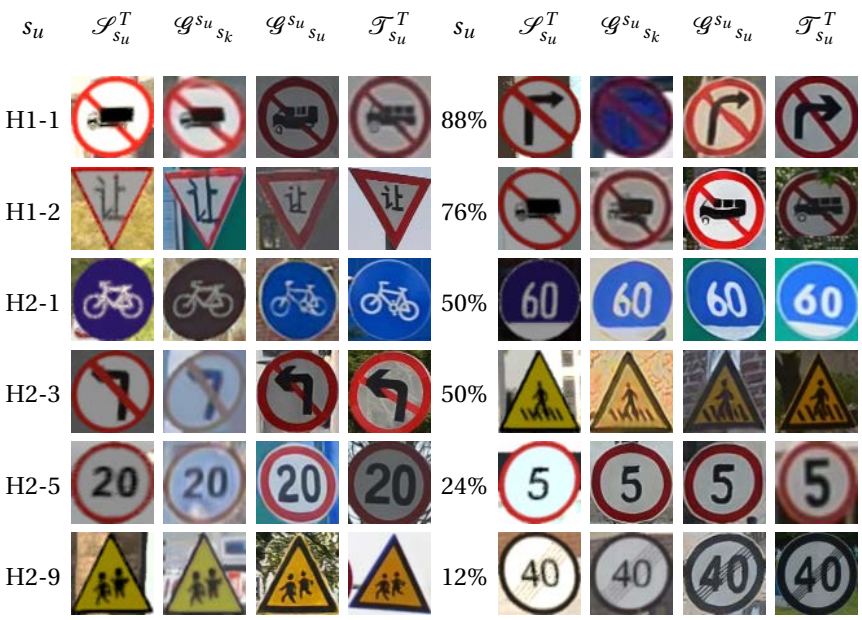

Figure 2.4 – Sample images. Left block (4 columns) and right block (4 columns) rely on different criteria to generate their splits. Left block: splits based on the hierarchy shown in Fig. 2.3. Right block: splits based on the balance between known and unknown (shown %) classes. Within each block, row-wise we show samples from the same class of the unknown split. Within each block, from the left to the right column, we have: a SYNTHIA-TS sample from the unknown class, a SYNTHIA-TS sample from an unknown class transformed by a CycleGAN trained on SYNTHIA-TS-to-Tsinghua samples from known classes, similar as the previous column but training a CycleGAN on the unknown classes, and a Tsinghua sample of an unknown class.

images (as they come from the virtual environment), therefore, we must expect a domain shift. Training on $\mathcal{T}^T_{H0-0}$ acts as upper bound, since we are using real-world images from the same distribution (camera and world area) than in the testing set. Table 2.3 shows these upper and lower bound results for the different architectures we have considered, namely VGG16 and ResNet101. Moreover, since during the training of CNNs there is certain amount of randomness (*e.g.* when sampling the datasets during a mini-batch), we repeat each training five times and report testing accuracy in terms of the mean and standard deviation of the F1 classification score

(*i.e.* F1 = (2TP)/(2TP+FN+FP)) computed on the respective classification results. These results show that: (1) we can achieve a high classification accuracy with the appropriate real-world data; (2) using the synthetic data for training produces a reasonable accuracy (far from random), but there is a dramatic domain shift with results dropping from 97.59% to 36.05% for VGG16, and from 98.76% to 58.74% for ResNet101.

Tables 2.4–2.5 report results to answer **Q1**. We consider paired splits, one is used as the set of known classes (s_k), the other as the set of unknown classes (s_u). These splits do not intersect, but their union does not necessarily corresponds to the full traffic sign hierarchy because only splits from Table 2.1 are considered. The pairs have been designed to force different global appearance between known and unknown classes. The $\mathscr{S}^T_{s_u}$ columns report the classification accuracy lower bound for each experiment, *i.e.* training a classifier for classes in s_u with samples in $\mathscr{S}^T_{s_u}$ but testing on the real-world data $\mathscr{T}^C_{s_u}$. Columns $\mathscr{T}^T_{s_u}$ act as upper bound, since training is done on real-world samples of s_u as if they were actually known. Columns $\mathscr{G}^{s_u}{}_{s_k}$ report the classification accuracies when training is done with the samples of $\mathscr{G}^{s_u}$, *i.e.* the samples of $\mathscr{S}^T_{s_u}$ transformed by a CycleGAN trained to perform image-to-image translation from $\mathscr{S}^T_{s_k}$ to $\mathscr{T}^T_{s_k}$. Therefore, the CycleGAN has not seen samples from classes in s_u at training time. Finally, we also include the case $\mathscr{G}^{s_u}{}_{s_u}$ where the CycleGAN has been trained using samples from the unknown set of classes. Obviously, this is not realistic in our application setting, however, it can be taken as an upper bound of the accuracy that would be possible to achieve by using CycleGAN to transform the synthetic images. Fig. 2.4 shows examples of the images involved in our experiments: synthetic, real, and transformed by different CycleGANs.

These results based on splits confirm the observations made for H0-0 according to Table 2.3; *i.e.* training and testing (for the unknown classes) with real-world data shows high classification accuracies, while training with the pure synthetic data and testing in the real-world data shows a significant drop of accuracy. Again, ResNet101 is more robust to domain shift than VGG16. We can see how the gap is larger as the number of classes based on synthetic data (unknown ones) increases. For instance, the gap for H1-1 is larger than for H1-2, both for VGG16 and ResNet101. Note that H1-1 contains 35 classes and H1-2 only 7 (see Table 2.1). If we analyse the splits of the next hierarchical level (H2-X), the same observations hold; note that H2-3 and H2-5 (10 and 16 classes, resp.) show a larger gap than H2-1 and H2-9 (6 and 5 classes, resp.), both for VGG16 and ResNet101.

On the other hand, CycleGAN indeed helps to significantly reduce domain shift. When using the H1-1 split as known classes to train the CycleGAN, and apply this GAN to the synthetic images of the unknown classes, *i.e.* those in H1-2 split, we see

~ 9 points of accuracy gain when testing in real-world images of H1-2 split (9.21 for VGG16 and 9.60 for ResNet101). Changing the roles of these splits, the gain is 10.68 for VGG16 and 4.56 for ResNet101. However, in the two situations (H1-1/H1-2 as known/unknown and viceversa) ResNet101 reports significantly higher accuracies (more than 10 points) after the GAN-based domain adaptation of the synthetic images. Also, for VGG16 and ResNet101, being H1-2 the split of unknown classes shows significantly higher accuracies (more than 20 points) than when it is H1-1, which is just a consequence of starting with similar accuracy differences before domain adaptation. Looking to the H2-X splits, we can see that the GAN-based domain adaptation reports significant higher accuracy in most of the experiments. In fact, it is more interesting to analyse when it is not the case. For instance, when split H2-1 is used to train the CycleGAN, we obtain either very low accuracy gains (*e.g.* for VGG16, 1.43 when the unknown classes are in H2-5 and 2.19 for H2-9) or even negative adaptation (*e.g.* -2.93 for H2-3 with VGG16, and for H2-3/5/9 with ResNet101). We think that, when using H2-1 to train the CycleGAN, the learned image-to-image transform is too biased towards a blue background, which is a color not present in the rest of the H2-X considered splits (in the role of unknown classes). When exchanging the roles between H2-1 and the rest of the considered H2-X splits, the conclusion is the same for VGG16. However, ResNet101 still is able to extract the most from the domain adapted images, showing significant accuracy gains with respect to using the synthetic images as they come directly from the virtual environment. Fig. 2.5 presents some visual hints. For instance, when split H2-1 is used to train the CycleGAN, this adds a bluish color to the transformed images, while the CycleGAN is trained with H2-9 split, the added color is yellowish. Being the former more marked than the latter, which may be the reason behind some of the previously mentioned cases of poor domain adaptation. We can see other effects, like blue background images going to black background. According to the reviewed results, ResNet101 seems more robust to this effect than VGG16 (see the case of s_u = H2-1 in Tables 2.4–2.5).

Tables 2.4–2.5 help to analyse results in scenarios where there are significant visual differences among the known/unknown classes. We are also interested on analysing different balances between known and unknown classes. Analogously to Table 2.2, we will define splits denoted by a percentage of classes, *e.g.* 100% would be H0-0. Each of these splits also have a complementary one with the remaining classes. When forming the new splits, in order to be sure that we do not degenerate in the previous hierarchy-based experiments, the classes are not sampled from H0-0, but they are proportionally and randomly sampled from all the H2-X splits. For example, 50% would consider half of H2-1, H2-3, H2-5 and H2-9 classes, added to H2-2, H2-6 and H2-8 which only have one class. Tables 2.6–2.7 present the corresponding results. We can see, how previous observations

Figure 2.5 – Samples based on H2-X splits (Fig. 2.3). Rows: splits with classes in the role of unknown. Left-to-right columns: samples from SYNTHIA-TS, SYNTHIA-TS samples transformed by a CycleGAN trained on SYNTHIA-TS and Tsinghua samples of classes in H2-1 split (which play the role of known classes here), analogous for H2-3 instead of H2-1, for H2-5 and H2-9, and samples from Tsinghua.

are confirmed, namely: (1) domain gap increases with the number of synthetic classes (the unknown ones) to be covered by the traffic sign classifier, but still the obtained accuracies are reasonable; (2) CycleGAN is able to dramatically reduce the domain shift for the unknown classes, recovering from ~ 10 to even ~ 30 points of accuracy; (3) ResNet101 is able to produce the best results before and after domain adaptation.

Overall, to already answer **Q1**, we see that using known classes to train a GAN-based transformation from synthetic to real-world domains, indeed helps to dramatically reduce the classification accuracy gap due to the domain shift for synthetically generated new classes. However, there are scenarios more favorable than others and still there is room for improvement.

Table 2.6 – Experiments to support **Q1** (see main text). All tests are done in $\mathcal{T}^{C}_{s_u}$. Average and standard deviation of F1 score are reported since each experiment is performed five times. The column $\mathcal{G}^{s_u}{}_{s_k} - \mathcal{S}^{T}_{s_u}$ just stands for the subtraction of the means of the respective columns.

Known (s_k)/Unknown (s_u) % classes ; Num. classes	VGG16 $\mathcal{S}^{T}_{s_u}$	VGG16 $\mathcal{G}^{s_u}{}_{s_k}$	VGG16 $\mathcal{G}^{s_u}{}_{s_k}$ - $\mathcal{S}^{T}_{s_u}$	VGG16 $\mathcal{G}^{s_u}{}_{s_u}$	VGG16 $\mathcal{T}^{T}_{s_u}$
~ 88%/12% ; 37/05	82.58 ± 3.62	91.84 ± 0.92	09.26	99.08 ± 0.42	99.85 ± 0.18
~ 76%/24% ; 32/10	78.72 ± 1.57	88.74 ± 0.86	10.02	95.00 ± 0.54	99.30 ± 0.20
~ 50%/50% ; 20/21	41.80 ± 2.85	74.11 ± 1.36	32.31	85.85 ± 1.15	97.75 ± 0.46
~ 24%/76% ; 10/32	43.81 ± 2.22	71.31 ± 2.64	27.50	82.62 ± 2.47	97.53 ± 0.30
~ 12%/88% ; 05/37	40.35 ± 2.69	50.24 ± 1.25	09.89	76.26 ± 3.63	94.93 ± 0.34

Table 2.7 – Experiments to support **Q1** analogous to Table 2.6, but using ResNet101.

Known (s_k)/Unknown (s_u) % classes ; Num. classes	ResNet101 $\mathcal{S}^{T}_{s_u}$	ResNet101 $\mathcal{G}^{s_u}{}_{s_k}$	ResNet101 $\mathcal{G}^{s_u}{}_{s_k}$ - $\mathcal{S}^{T}_{s_u}$	ResNet101 $\mathcal{G}^{s_u}{}_{s_u}$	ResNet101 $\mathcal{T}^{T}_{s_u}$
~ 88%/12% ; 37/05	97.54 ± 0.69	97.24 ± 2.27	09.70	99.47 ± 0.46	100.00 ± 0.00
~ 76%/24% ; 32/10	78.33 ± 3.08	88.12 ± 1.33	09.79	95.02 ± 0.84	99.69 ± 0.14
~ 50%/50% ; 20/21	63.78 ± 1.66	78.84 ± 2.24	15.06	87.11 ± 1.20	98.46 ± 0.24
~ 24%/76% ; 10/32	60.39 ± 3.27	77.68 ± 1.85	17.29	86.34 ± 1.05	98.75 ± 0.13
~ 12%/88% ; 05/37	60.17 ± 2.61	59.12 ± 2.61	-01.05	84.42 ± 0.76	96.22 ± 0.06

Table 2.8 – Experiments to support **Q2** (see main text), all done in $\mathcal{T}^{C}_{H0-0}$. Average and standard deviation of F1 score are reported since each experiment is performed five times. This is done for the all-classes classification problem, but we also show detailed results for known and unknown classes. Table 2.3 shows the lower and upper bounds for these experiments, *i.e.* training only on either SYNTHIA-TS or Tsinghua data. In terms of average F1, these bounds are 36.05 and 97.59, resp.

Unknown classes (s_u)	VGG16, $\mathcal{S}^{T}_{H0-0} + \mathcal{T}^{T}_{s_k}$ H0-0 (s_k / s_u)	VGG16, $\mathcal{G}^{H0-0}{}_{s_k} + \mathcal{T}^{T}_{s_k}$ H0-0 (s_k / s_u)
H1-1	37.12 ± 2.47 (59.83 ± 8.55/32.58 ± 2.90)	59.54 ± 3.60 (84.97 ± 2.89/54.45 ± 3.82)
H1-2	88.81 ± 1.70 (95.92 ± 0.74/53.22 ± 9.91)	92.88 ± 1.42 (96.77 ± 0.62/73.46 ± 6.21)
12%	87.53 ± 1.55 (92.50 ± 0.81/50.69 ± 7.88)	93.79 ± 1.16 (94.73 ± 1.16/86.83 ± 2.11)
24%	78.17 ± 1.81 (88.21 ± 0.68/46.05 ± 5.79)	89.91 ± 2.19 (92.84 ± 1.17/80.53 ± 5.79)
50%	56.52 ± 3.84 (75.85 ± 1.50/35.26 ± 6.60)	74.96 ± 1.40 (85.01 ± 1.26/63.90 ± 2.73)
76%	30.69 ± 2.13 (49.09 ± 1.67/24.94 ± 3.10)	67.45 ± 1.41 (74.04 ± 2.71/65.39 ± 1.28)
88%	31.46 ± 3.18 (33.34 ± 3.66/31.23 ± 3.27)	53.62 ± 2.26 (64.44 ± 4.74/50.70 ± 2.46)

First, the used CNN matters. Here Resnet101 shows significantly better classification accuracies than VGG16, *i.e.* ResNet101 is more robust to this kind of known/unknown class setting. We can see this by looking at Tables 2.4–2.5. Note how, for H2-X splits, when the split of classes used to train CycleGAN is the same split as used to train the traffic sign classifier ($\mathcal{G}^{s_u}{}_{s_u}$ columns), then the classification accuracies of VGG16 and ResNet101 are similar, VGG16 even outperforms ResNet101 several times. A similar effect can be appreciated in Tables 2.6–2.7. Hence, ResNet101 seems to be more robust to image imperfections introduced by CycleGAN. In this favorable, but unrealistic, setting the domain adapted images show less artifacts (see Figures 2.4 and 2.5).

Second, the most adverse scenario indeed is when known and unknown classes show very different appearance combined with a low known/unknown class ratio. In a reasonable case, as for $\sim$ (88%/12%) splits of randomly selected known/unknown classes, using ResNet101, we can see how we obtain a classification accuracy of 97.24 in average (Table 2.7), where training with real-world data reaches 100.00. In the most imbalanced case, $\sim$ (12%/88%), ResNet101 reaches a classification accuracy of 77.85, still far from the 98.82 when training with real-world images. Note that in this scenario there is room to improve GAN-based image-to-image translation since even using the classification classes to train the CycleGAN, the obtained accuracy is 84.93, still far from 98.82.

Although in this chapter these are just intermediate results in our way to address **Q2**, this analysis is already useful if our goal is to perform transfer learning for the

Table 2.9 – Experiments to support **Q2** analogous to Table 2.8, but using ResNet101. In this case, the lower and upper bounds are 58.74 and 98.76, respectively.

Unknown classes (s_u)	ResNet101, $\mathscr{S}^T_{H0-0} + \mathscr{T}^T_{s_k}$ H0-0 (s_k / s_u)	ResNet101, $\mathscr{G}^{H0-0}{}_{s_k} + \mathscr{T}^T_{s_k}$ H0-0 (s_k / s_u)
H1-1	65.67 ± 0.66 (95.42 ± 0.80/59.72 ± 0.68)	70.65 ± 1.60 (98.53 ± 0.38/65.08 ± 1.97)
H1-2	95.55 ± 0.39 (98.37 ± 0.27/81.45 ± 3.30)	97.57 ± 0.31 (98.77 ± 0.11/91.55 ± 1.33)
12%	90.05 ± 1.67 (94.11 ± 0.62/60.00 ± 9.71)	95.54 ± 0.82 (96.44 ± 0.47/88.95 ± 3.49)
24%	82.70 ± 1.22 (91.02 ± 0.63/56.07 ± 3.15)	92.57 ± 0.36 (95.00 ± 0.50/84.78 ± 0.77)
50%	68.03 ± 2.03 (81.92 ± 0.95/52.74 ± 3.32)	81.08 ± 0.57 (87.72 ± 0.60/73.78 ± 0.85)
76%	49.23 ± 1.17 (50.94 ± 1.66/48.70 ± 1.06)	71.94 ± 1.76 (71.50 ± 2.38/72.08 ± 1.75)
88%	52.95 ± 2.31 (43.77 ± 1.99/54.06 ± 2.53)	56.56 ± 2.56 (56.83 ± 1.93/55.43 ± 2.70)

traffic sign classifier; *i.e.* if we want to train a classifier that only needs to operate in a new set of traffic signs for which we have not enough samples, and we want to leverage knowledge from the known classes even if these are not going to be used anymore for classification. For instance, in particular environments with specialized traffic signs, like in some closed infrastructures or industrial facilities. In fact, the vision system does not need to be on-board a vehicle, it can even be on a humanoid or any other robotic platform. However, a probable requisite in this case would be to use the same camera sensor model to classify new classes than the used for collecting the real-world images involved in the training of the GAN-based domain adapter.

Finally, in order to address **Q2**, we have performed the experiments shown in Tables 2.8–2.9. In these tables, columns $\mathscr{S}^T_{H0-0} + \mathscr{T}^T_{s_k}$ refer to training jointly with synthetic and real-world data. $\mathscr{S}^T_{H0-0}$ is the full SYNTHIA-TS set, *i.e.* covering known (k) and unknown (u) classes. Therefore, we use all the synthetic data available for the classes we want to classify. On the other hand, $\mathscr{T}^T_{s_k}$ refers to the training set of all known real-world classes; in these experiments $k \cup u$ are the 42 considered classes of split H0-0. Therefore, $\mathscr{T}^T_{s_k} + \mathscr{T}^T_{s_u}$ equals the full Tsinghua training set ($\mathscr{T}^T_{H0-0}$). Columns $\mathscr{G}^{H0-0}{}_{s_k} + \mathscr{T}^T_{s_k}$ are analogous to previous ones, but in this case rather than using the synthetic data as gathered from the virtual environment, we use it transformed by a CycleGAN. These GANs are trained only using the known classes in each experiment, *i.e.* in this case $\mathscr{T}^T_{s_k}$ and $\mathscr{S}^T_{s_k}$. Accordingly, $\mathscr{G}^{H0-0}{}_{s_k}$ is composed by $\mathscr{G}^{s_k}{}_{s_k}$, used as pure data augmentation, as well as $\mathscr{G}^{s_u}{}_{s_k}$, which are really needed since we assume that we do not have real-world training samples for the split s_u.

We see that the observations done in the context of **Q1** apply here too: (1) domain gap increases with the number of synthetic classes (the unknown ones); (2)

CycleGAN is able to significantly reduce the domain shift; (3) ResNet101 performs better than VGG16 before and after domain adaptation, even when using the full Tsinghua training set they report similar upper bounds (97.59 for VGG16, 98.76 for ResNet101). In the case of ResNet101, the best cases almost reach the upper bound: (1) when the known classes are those in split H1-1 (~ 83% of classes), and the unknown in split H1-2 (~ 17% of classes), we obtain 97.57 which is very close to 98.76; (2) when classes are directly selected randomly on the H2-X hierarchy level, the case of 12% of unknown classes reaches 95.54 which is also a very high accuracy; even the 24% case still reports 92.57. Moreover, in all these cases, the accuracy on the known classes keeps over 95 and over 84 for the unknown ones (91.55 for H1-2, 88.95 for the 12%, 84.78 for the 24%). Overall, we can conclude that with ResNet101 the proposed method works well when the ratio of unknown/known classes is of ~ 1/4. In order to reach the upper bound, we can investigate if we can still improve CycleGAN, but in this ~ 1/4 regime, the *last mile* can be probably covered by adding a small amount of real-world collected and annotated samples from the unknown classes. As the vision system keeps performing in the real-world, the samples falling in the new classes can be kept, then we can replace the synthesized and transformed samples by these self-annotated ones in a future retraining of the classifier. In fact, this self-annotation cycles can be also a good approach for more challenging ratios of unknown/known classes; note that for the 76% of unknown classes the results are over 70, and over 50 for the 88%.

2.5 Conclusions

There are situations where a computer vision system may need to recognise new (previously unknown) classes, but the lack of samples from such classes (*i.e.* raw images with annotations) may seriously delay this possibility. In this chapter, we have explored how to address this situation by using synthetic data and leveraging samples from the classes already known to the system. Since, there is a domain shift between synthetic and real worlds, addressing the problem involves incorporating some kind of domain adaptation. To solve the lack of data problem, we have proposed to learn a GAN using (the already available) samples from the known classes, and apply it to adapt synthetic samples from the new classes. As a proof of concept, we have focused on traffic sign recognition. We have used the publicly available Tsinghua dataset and we have created a synthetic dataset (SYNTHIA-TS), which we will also publicly release, for designing the experiments presented in this chapter. In particular, the experiments have been designed to address two questions. First, we addressed the intermediate question *can we reduce the synthetic-to-real domain shift by applying an image-to-image translation GAN to the unknown classes,*

provided such a GAN was trained only with the known classes? After an extensive set of experiments and results that we have presented and discussed, the answer was positive, which leads us to the main question, namely, *what are the overall classification results when training the classifier using the real-world data of the known classes with the data generated for the new classes following our proposal?* Again, the obtained results allow us to conclude that the proposed method is indeed effective provided we use a proper CNN to perform the classifications task as well as a proper GAN to transform the synthetic images. Here, a ResNet101-based classifier and domain adaptation based on CycleGAN performed extremely well for ratios of unknown/known classes of even $\sim 1/4$. For more challenging ratios such as $\sim 4/1$ the results are also very positive. As a matter of fact, instead of focusing on improving the components of the presented method, as future work we plan to augment this method with complementary techniques such as self-annotation, *i.e.* using the classifier generated with our current method to self-annotate real-wold samples of the new classes for a posterior retraining/fine-tuning of the classifier.

3 Co-training for On-board Deep Object Detection

Providing ground truth supervision to train visual models has been a bottleneck over the years, exacerbated by domain shifts which degenerate the performance of such models. This was the case when visual tasks relied on handcrafted features and shallow machine learning and, despite its unprecedented performance gains, the problem remains open within the deep learning paradigm due to its data-hungry nature. Best performing deep vision-based object detectors are trained in a supervised manner by relying on human-labeled bounding boxes which localize class instances (*i.e.* objects) within the training images. Thus, object detection is one of such tasks for which human labeling is a major bottleneck. In this chapter, we assess co-training as a semi-supervised learning method for self-labeling objects in unlabeled images, so reducing the human-labeling effort for developing deep object detectors. Our study pays special attention to a scenario involving domain shift; in particular, when we have automatically generated virtual-world images with object bounding boxes and we have real-world images which are unlabeled. Moreover, we are particularly interested in using co-training for deep object detection in the context of driver assistance systems and/or self-driving vehicles. Thus, using well-established datasets and protocols for object detection in these application contexts, we will show how co-training is a paradigm worth to pursue for alleviating object labeling, working both alone and together with task-agnostic domain adaptation.

3.1 Introduction

Since more than two decades ago, machine learning (ML) has been the enabling technology to solve computer vision tasks. In the last decade, traditional ML, *i.e.* based on relatively shallow classifiers and hand-crafted features, has given way to deep learning (DL). Thanks to DL models based on convolutional neural networks (CNNs), DL approaches significantly outperformed traditional ML in all kinds of computer vision tasks, such as image classification, object detection, semantic

segmentation, etc. A major key for the usefulness of DL models is to train them in a supervised way. In other words, the raw data (still images and videos) need to be supplemented by ground truth; nowadays, usually collected via crowd-sourced labeling. In practice, due to the data-hungry nature of CNNs, data labeling is considered a major bottleneck. Therefore, approaches to minimize labeling effort are of high interest; or put in another way, approaches to automatically leverage the large amounts of available unlabeled raw data must be pursued. Thus, we can find different families of algorithms, or paradigms, which address human-labeling minimization under different working assumptions. We continue this introduction by reviewing them in subsection 3.1.1, while subsection 3.1.2 highlights how domain shifts exacerbate human labeling requirements. With the human-labeling bottleneck problem introduced, subsection 3.1.3 summarizes our focus when addressing such a problem. In particular, we outline our application context and the proposed approach for reducing human annotation effort in such a context. Finally, subsection 3.1.4 highlights the main contributions presented in this chapter as well as the organization of the rest of its contents.

3.1.1 Paradigms to minimize human labeling

For instance, *active learning* (AL) approaches [1, 125, 128] assume an initial model and an unlabeled dataset to be labeled by a human (oracle) following an interactive procedure. In particular, the current model processes the unlabeled data providing so-called *pseudo-labels* (at image, object, or pixel level); then, these results are inspected, either automatically or directly by the oracle, to select the next data to be labeled. Afterwards, the model is trained again, and the process repeated until fulfilling a stop criterion. AL assumes that with less labeling budget than the required to have the oracle labeling data at random, one can train models that perform at least similarly. In practice, it is usually hard to clearly outperform the oracle's random labeling strategy. In contrast to AL, other approaches do not assume human oracles in the labeling loop. For instance, this is the case of semi-supervised learning (SSL) algorithms [25, 149, 153], which assume the availability of a large number or raw unlabeled data together with a relatively small number of labeled data. Then, a model must be trained using the unlabeled and labeled data (without human intervention), with the goal of being more accurate than if only the labeled data were used. The so-called *self-training* and *co-training* algorithms, which are of our main interest in this chapter, fall in the SSL paradigm [153]. In AL and SSL, the most common idea is to efficiently label the unlabeled data, either via an oracle (AL) or automatically (SSL). *Self-supervised learning* (SfSL) follows a different approach which consists in providing supervision in the form of additional (relatively simple) tasks, known as pretext tasks, for which automatic ground truth

can be easily generated (*e.g.* solving jigsaw puzzles [47, 74, 77]).

3.1.2 The domain adaptation problem

Within these approaches, the most classical assumption is that the labeled and the unlabeled data are drawn from the same distribution and one aims at using the unlabeled data (via annotations or pretext tasks) to solve the same tasks for which the labeled data were annotated. However, in practice, we may require to solve new tasks, leading to *transfer learning* (TL) [164, 188], or solving the same tasks in a new domain, leading to *domain adaptation* (DA) [37, 159, 166]. Beyond specific techniques to tackle TL or DA, we can leverage solutions/ideas from AL, SSL, or SfSL. For instance, AL [155], SSL [75, 190], and SfSL [174] algorithms have been used to address DA problems. In this chapter, we use SSL (comparing self-training and co-training) to address DA too.

3.1.3 The focus of this work

From the application viewpoint, in this chapter, we focus on vision-based on-board deep object detection. Note that this is a very relevant visual task for driving, since detecting objects (*e.g.* vehicles, pedestrians, etc.) along the route of a vehicle is a key functionality for perception-decision-action pipelines in the context of both advanced driver assistance systems (ADAS) and self-driving vehicles. Moreover, nowadays, most accurate vision-based models to detect such objects are based on deep CNN architectures [88, 92, 118, 120]. In addition, in this application context, it is possible to acquire innumerable quantities of raw images, for instance, from cameras installed in fleets of cars. Thus, methods to minimize the effort of manually labeling them are of great relevance.

Among the SSL approaches, co-training [17, 58] has been explored for deep image classification [29, 117] with promising results. However, up to the best of our knowledge, it remains unexplored for deep object detection. Note that, while image classification aims at assigning image-level class/attribute labels, object detection is more challenging since it requires to localize and classify objects in images, *i.e.* placing a bounding box (BB) per object, together with the class label assigned to it. Moreover, object detection from on-board images, *i.e.* detection of vehicles, pedestrians, etc., is especially difficult since, to the inherent intra-class differences (*e.g.* due to vehicle models, pedestrian clothes, etc.), acquisition conditions add a vast variability because the objects appear in a large range of distances to the camera (resolution and focus variations), under different illumination conditions (from strong shadows to direct sunlight), and they usually move (blur, occlusion, view angle, and pose variations).

Originally, co-training relies on the *agreement* of two trained models performing predictions from different *features* (views) of the data [17]. These predictions are taken as pseudo-labels to improve the models incrementally. Later, prediction *disagreement* was shown to improve co-training results in applications related to natural language processing [58, 150]. Thus, in this chapter, we propose a co-training algorithm inspired by prediction *disagreement*. On the other hand, since co-training was proposed as a SSL method aiming at avoiding the drift problem of self-training [17], which incrementally re-trains a single model from its previously most confident predictions, we also elaborate a strong self-training baseline for our deep object detection problem.

We assess the effectiveness of the two self-labeling methods, *i.e.* self-training and co-training, in two different practical situations. First, following most classical SSL, we will assume access to a small dataset of labeled images, $\mathscr{X}^l$, together with a larger dataset of unlabeled images $\mathscr{X}^u$; where here the labels are object BBs with class labels. Second, we will address a relevant setting that resorts DA. In particular, we will assume access to a dataset of virtual-world images with automatically generated object labels. Therefore, eventually, $\mathscr{X}^l$ can be larger than $\mathscr{X}^u$, since labeling these images does not require human intervention; however, $\mathscr{X}^u$ is composed of real-world images, thus, between $\mathscr{X}^l$ and $\mathscr{X}^u$ there is a domain shift [155]. In this case, we also compare self-labeling techniques with task-agnostic DA, in particular, using GAN-based image-to-image translation [186]. As especially relevant cases, we will focus on on-board detection of vehicles and pedestrians using a deep CNN architecture. Our experiments will show how, indeed, our co-training algorithm is a good SSL alternative for on-board deep object detection. Co-training clearly boosts detection accuracy in regimes where the size of $\mathscr{X}^l$ is just the 5%/10% of the labeled data used to train an upper-bound object detector. Moreover, under the presence of domain shift, we will see how image-to-image GAN-based translation and co-training complement each other, allowing to reach almost upper-bound performances without human labeling.

3.1.4 Contributions and organization

Hence, the main contributions of this chapter are: (1) proposing a co-training algorithm for deep object detection; (2) designing this algorithm to allow addressing domain shift via GAN-based image-to-image translation; (3) showing its effectiveness by developing a strong self-training baseline and relying on publicly available evaluation standards and on-board image datasets. Alongside, we will also contribute with the public release of the virtual-world dataset we generated for our experiments. To show these achievements we organize the rest of the chapter as follows. Section 3.3 details our self-training and co-training algorithms. Section 3.4

draws our experimental setting, and discuss the obtained results. Finally, Section 3.5 summarizes the presented work, suggesting lines of continuation.

3.2 Semi-supervised learning

Our main goal is to train a vision-based deep object detector without relying on human labeling. Following our work line [96], we propose to leverage labeled images from virtual worlds and unlabeled ones from the real world, in such a way that we can automatically label the real-world images by progressively re-training a deep object detector that substitutes human annotators. Note that the labeling step is needed due to the domain shift between virtual and real-world images, otherwise, we could just use the virtual-world images to train object detectors and, afterwards, deploy them in the real world expecting a reliable performance. In particular, we explore the combination of the co-training idea for self-labeling objects and GAN-based image-to-image translation in the role of task-agnostic DA. On the other hand, our proposal is agnostic to the object detection architecture supporting self-labeling.

Accordingly, in the remaining of this section, we first review related works on self-labeling (subsection 3.2.1), including self-training and co-training; next, we review related works on DA (subsection 3.2.2). In both sections, we relate our proposal to the reviewed works.

3.2.1 Self-labeling

Self-labeling algorithms are examples of SSL *wrappers* [153], which work as meta-learners for supervised ML algorithms. The starting point consists of a labeled dataset, $\mathscr{X}^l$, and an unlabeled dataset, $\mathscr{X}^u$, where it is supposed that the cardinality of $\mathscr{X}^u$ is significantly larger than the cardinality of $\mathscr{X}^l$, and both datasets are drawn from the same domain, $\mathscr{D}$. The goal is to learn a predictive model, ϕ, whose accuracy would be relatively poor by only using $\mathscr{X}^l$ but becomes significantly higher by also leveraging $\mathscr{X}^u$. Briefly, self-labeling is an incremental process where ϕ is first trained with $\mathscr{X}^l$; then $\mathscr{X}^u$ is processed using ϕ in a way that the predictions are taken as a new pseudo-labeled dataset $\mathscr{X}^{\hat{l}}$, which in turn is used to retrain ϕ. The pseudo-labeling/retraining cycle is repeated until reaching some stop criterion, when ϕ is expected to be more accurate than at the beginning of the process. The main differences between self-labeling algorithms arise from how $\mathscr{X}^{\hat{l}}$ is formed and used to retrain ϕ in each cycle.

Self-training In self-training, introduced by Yarowsky [179] for word sense disambiguation, $\mathscr{X}^{\hat{l}}$ is formed by collecting the most confident predictions of ϕ in $\mathscr{X}^u$, updating $\mathscr{X}^u$ to contain only the remaining unlabeled data. Then, ϕ is retrained using supervision from $\mathscr{X}^l \cup \mathscr{X}^{\hat{l}}$, *i.e.* pseudo-labels are taken as ground truth. Before the DL era, Rosenberg et al. [124] already showed promising results when applying self-training to eye detection in face images. More recently, Jeong et al. [70], proposed an alternative to collect $\mathscr{X}^{\hat{l}}$ for deep object detection, which consists of adding a consistency loss for training ϕ as well as eliminating predominant backgrounds. If I_i^u is an unlabeled image, the consistence loss is based on the idea that $\phi(\mathrm{I}_i^u)$ and $\phi((\mathrm{I}_i^u)^{\leftarrow})$, where $()^{\leftarrow}$ stands for horizontal mirroring, must provide corresponding detections. Experiments are conducted on PASCAL VOC and MS-COCO datasets, and results are on pair with other state-of-the-art methods combining AL and self-training [158]. The reader is referred to [106] for a review on loss-based SSL methods for deep image classification. On the other hand, note that this SSL variations are not agnostic to ϕ, since its training loss is modified. This is also the case in [94], where, in the context of deep image classification, the activation functions composing ϕ must be replaced by Hermite polynomials. In this chapter, we use a self-training strategy as SSL baseline for on-board deep object detection, thus, keeping agnosticism regarding ϕ.

Co-training Co-training was introduced by Blum and Mitchell [17] in the context of web-page classification, as alternative to the self-training of Yarowsky [179], in particular, to avoid model drift. In this case, two models, ϕ_1 and ϕ_2, are trained on different conditionally independent features of the data, called *views*, assuming that each view is sufficiently good to learn an accurate model. Each model is trained by following the same idea as with self-training, but in each cycle the data samples self-labeled by ϕ_1 and ϕ_2 are aggregated together. Soon, co-training was shown to outperform other state-of-the-art methods including self-training [104], and the conditional independence assumption was shown not to be essential in practice [13, 162, 163]. Later, in the context of sentence segmentation, Guz et al. [58] introduced the *disagreement* idea for co-training, which was refined by Tur [150] to jointly tackle DA in the context of natural language processing. In this case, samples self-labeled with high confidence by ϕ_i but with low confidence by ϕ_j, $i, j \in \{1,2\}, i \neq j$, are considered as part of the new pseudo-labeled data in each cycle. In fact, disagreement-based SSL became a subject of study on its own at that time [185].

Before the irruption of DL, Levin et al. [84] applied co-training to detect vehicles in video-surveillance images, so removing background and using different training data to generate different views. More recently, Qiao et al. [117] used a co-training

setting for deep image classification, based on several views. Each view corresponds to a different CNN, ϕ_i, trained by including samples generated to be mutually adversarial. The idea is to use different training data for each ϕ_i to prevent them to prematurely collapse in the same network. This implies to link the training of the ϕ_i's at the level of the loss function. In this chapter, we force the use of different training data for each ϕ_i, without linking their training at the level of loss functions, again, keeping co-training agnostic to the used ϕ_i. Moreover, we address objected detection, which involves not only predicting class instances as in image classification, but also localizing them within the images, so that the background becomes a large source of potential false positives.

Finally, to avoid confusion, it is worth to mention the so-called *co-teaching*, recently introduced by Han et al. [60], and its variant *co-teaching+* introduced by Yu et al. [181]. These algorithms are designed to address situations with noisy labels on $\mathscr{X}^l$, both demonstrated on deep image classification. Indeed, these algorithms stem ideas from co-training (the classical one from [60], the disagreement-based one from [181]), however, reproducing the words of Han et al. [60], *co-training is designed for SSL, and co-teaching is for learning with noisy (ground truth) labels (LNL); as LNL is not a special case of SSL, we cannot simply translate co-training from one problem setting to another problem setting.*

3.2.2 Domain adaptation

ML algorithms assume that training and testing data are drawn from the same domain, $\mathscr{D}$. When this is not the case, the trained models suffer from *domain shift*. In other words, we have data, $\mathscr{X}_S$, drawn from a *source domain*, $\mathscr{D}_S$, as well as data, $\mathscr{X}_T$, drawn from a *target domain*, $\mathscr{D}_T, \mathscr{D}_T \neq \mathscr{D}_S$. We can assume that $\mathscr{X}_S = \mathscr{X}_S^{tr} \cup \mathscr{X}_S^{tt}, \mathscr{X}_S^{tr} \cap \mathscr{X}_S^{tt} = \emptyset$, where $\mathscr{X}_S^{tr}$ is used to train some predictive model ϕ_S. It turns out that the prediction accuracy of ϕ_S in $\mathscr{X}_S^{tt}$ is much higher than in $\mathscr{X}_T$, a phenomenon known as domain shift. Addressing this problem is the goal of DA techniques, under the assumption that there is some (unknown) correlation between $\mathscr{D}_S$ and $\mathscr{D}_T$, since DA is not possible otherwise.

The core idea is to use $\mathscr{X}_T$ to obtain a new model, ϕ_T, being clearly more accurate than ϕ_S in $\mathscr{D}_T$. While doing this, the human-based labeling effort in $\mathscr{X}_T$ must be minimized. *Supervised DA* (SDA) assumes access to a relatively small set of labeled target-domain data $\mathscr{X}_T^{l,tr} \subset \mathscr{X}_T$. If we do not have access to $\mathscr{X}_S^{l,tr}$, then the challenge is to leverage from ϕ_S and $\mathscr{X}_T^{l,tr}$ to obtain ϕ_T. Otherwise, we can combine $\mathscr{X}_S^{l,tr}$ and $\mathscr{X}_T^{l,tr}$ to train ϕ_T. In *unsupervised DA* (UDA), $\mathscr{X}_T$ is unlabeled; thus, we address the more challenging situations of using $\mathscr{X}_T$ with either ϕ_S or $\mathscr{X}_S^{l,tr}$ to train ϕ_T. For a review of the DA corpus we advise the reader to consult [37, 159, 166].

In this chapter, we assume an UDA setting. Moreover, our source data come from a virtual world with automatically generated labels, so we have $\mathscr{X}_S^{l,tr}$. Since we aim at assessing self-labeling by co-training to address the UDA problem, we have $\mathscr{X}^l = \mathscr{X}_S^{l,tr}$, $\mathscr{X}^u = \mathscr{X}_T$, thus, $\mathscr{X}^{\hat{l}} \subseteq \mathscr{X}_T$.

Following this line of work, Kim et al. [75] addressed USD for deep object detection by proposing a combination of a weak self-training and a special treatment of backgrounds via a loss component used during the training of the object detector, with PASCAL VOC as $\mathscr{X}_S$ and art-style datasets (Clipart1K, Watercolor2K, Comic2K) as $\mathscr{X}_T$. Zou et al. [191] also use a self-training strategy for UDA in the context of deep image classification and semantic segmentation (including a virtual-to-real setting), where the core idea is to perform confidence and model regularization of the trained classifiers. Since in our experimental setting we will use Faster R-CNN to obtain ϕ, it is also worth to mention the work by Chen et al. [32], where an UDA method was specifically designed for Faster R-CNN and demonstrated on car detection under a virtual-to-real setting too. Since Faster R-CNN is a two-stage classifier, the proposed UDA involves an image-level adaptation for the region proposal stage, and an instance-level adaptation for the BB prediction stage. Finally, focusing on traditional ML and *Amazon reviews* datasets, Chen et al. [28] showed that co-training is a promising algorithm for UDA, providing better performance than self-training.

Accordingly, beyond self-training-style and model-specific strategies for UDA, in this chapter, we are interested in assessing co-training as a meta-learning UDA strategy. To the best of our knowledge, this is an under-explored and relevant setting. Moreover, even we are going to use Faster R-CNN because its outstanding accuracy, our proposal neither is specific for it, nor requires modifying its losses. In other words, our co-training-based UDA works at the training-data level. This allows to complement it with other UDA working at the same level. In particular, we combine it with GAN-based image-to-image translation, since such task-agnostic approach can transform $\mathscr{X}_S^{l,tr}$ to be more similar to $\mathscr{X}_T$ before starting co-training. By using the CycleGAN implementation of Zhu et al. [186], we will see how, indeed, GAN-based image-to-image translation combined with co-training outperforms the use of each method in isolation.

3.3 Methods

In this section, we detail our self-training and co-training meta-learning proposals as Algorithms 1 and 2, respectively. In addition, Figures 3.1 and 3.2 provide a visual representation of these algorithms, highlighting their main components with the corresponding data flow. Our main interest is to assess the performance of co-

input : Labeled images: $\mathscr{X}^l = \{< \mathrm{I}_i^l, \mathscr{Y}_i^l >\}$
Unlabeled images: $\mathscr{X}^u = \{\mathrm{I}_i^u\}$
Object detection architecture: Φ
Φ Training hyper-par.: $\mathscr{H}_\Phi$
Slf-tr. hyp.-p.: $\mathscr{H}_{sl} = \{T, N, n, \mathscr{H}_{stp}[, \mathscr{H}_{seq}]\}$
output : New labeled images: $\mathscr{X}^{\hat{l}} = \{< \mathrm{I}_i^{\hat{l}}, \mathscr{Y}_i^{\hat{l}} >\}$

$< \mathscr{X}^{\hat{l}}, k > \leftarrow < \emptyset, 0 >$;
$\phi \leftarrow \mathrm{TrainDetector}(\Phi, \mathscr{H}_\Phi, \mathscr{X}^l, \mathscr{X}^{\hat{l}})$;
$\mathscr{X}^{\hat{l}}_{new} \leftarrow \mathrm{RunDetector}(\phi, \mathscr{X}^u, T)$;
repeat
  $\mathscr{X}^{\hat{l}}_{old} \leftarrow \mathscr{X}^{\hat{l}}_{new}$;
  $\mathscr{X}^{\hat{l}}_{\uparrow} \leftarrow \mathrm{Select}(\uparrow, n, \mathrm{Rand}(\mathscr{X}^{\hat{l}}_{new}, N[, \mathscr{H}_{seq}, k]))$;
  $\mathscr{X}^{\hat{l}} \leftarrow \mathrm{Fuse}(\mathscr{X}^{\hat{l}}, \mathscr{X}^{\hat{l}}_{\uparrow})$;
  $\phi \leftarrow \mathrm{TrainDetector}(\Phi, \mathscr{H}_\Phi, \mathscr{X}^l, \mathscr{X}^{\hat{l}})$;
  $\mathscr{X}^{\hat{l}}_{new} \leftarrow \mathrm{RunDetector}(\phi, \mathscr{X}^u, T)$;
until *Stop?*$(\mathscr{H}_{stp}, \mathscr{X}^{\hat{l}}_{old}, \mathscr{X}^{\hat{l}}_{new}, k{+}{+})$;
$\mathscr{X}^{\hat{l}} \leftarrow \mathscr{X}^{\hat{l}}_{new}$;
return $\mathscr{X}^{\hat{l}}$;

Algorithm 1: Self-labeling by self-training (see main text).

training in vision-based object detection, but we need also to develop a strong self-training baseline. Since they share functional components, we first introduce those and then detail how they are used for both self-training and co-training. Finally, we will see how, depending on the input data, these SSL algorithms can be used even in a context where there is a domain shift between already existing labeled images and the images to be labeled automatically. To refer to both, self-training and co-training indistinctly, we will use the term *self-labeling* in the rest of this section.

3.3.1 Self-labeling functional components

Input and output parameters Since we follow a SSL setting [153], we assume access to a set of images (*e.g.* acquired on-board a car) with each object of interest (*e.g.* vehicles and pedestrians) labeled by a BB and a class label, as well as to a set of unlabeled images. The former is denoted as $\mathscr{X}^l = \{< \mathrm{I}_i^l, \mathscr{Y}_i^l >\}$, where I_i^l is a

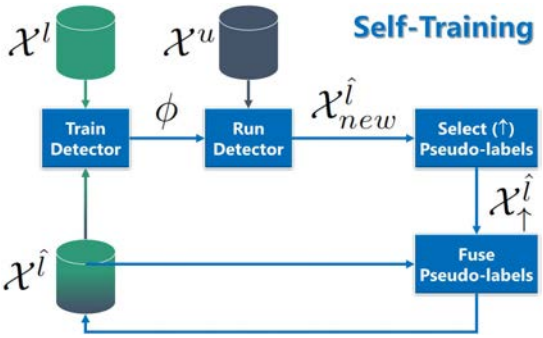

Figure 3.1 – Self-training main components. We use the same notation as in Algorithm 1. The data in green is labeled, the one in dark grey is unlabeled, the transition of both colors represents pseudo-labeled data. The blue boxes correspond to the main components involved in self-training according to Algorithm 1.

particular image of the labeled set, being $\mathscr{Y}_i^l$ its corresponding labeling information; *i.e.*, for each object of interest in I_i^l, $\mathscr{Y}_i^l$ includes its BB and its class label. The unlabeled set is $\mathscr{X}^u = \{\mathrm{I}_i^u\}$, where I_i^u is a particular unlabeled image. We assume also a given object detection architecture to perform self-labeling, denoted as Φ, with corresponding training hyper-parameters denoted as $\mathscr{H}_\Phi$. After self-labeling, we expect to obtain a new set of automatically labeled images, which we denote as $\mathscr{X}^{\hat{l}} = \{< \mathrm{I}_i^{\hat{l}}, \mathscr{Y}_i^{\hat{l}} >\}$, where $\mathrm{I}_i^{\hat{l}} \in \mathscr{X}^u$ and has $\mathscr{Y}_i^{\hat{l}}$ as so-called pseudo-labels, which in this case consists of a BB and a class label for each detected object in $\mathrm{I}_i^{\hat{l}}$. The variables introduced so far are generic in SSL, *i.e.* they are not specific of our proposals. We denote as $\mathscr{H}_{sl}$ the specific set of hyper-parameters we require for self-labeling. In fact, $\mathscr{H}_{sl}$ includes $\{T, N, n, \mathscr{H}_{stp}, \mathscr{H}_{seq}\}$ as common parameters for both self-training and co-training, but the latter requires an additional parameter m that we will explain in the context of co-training. If K is the number of classes to be considered, then $T = \{t_1, \ldots, t_K\}$ is a set of per-class detection thresholds, normally used by object detectors to ensure a minimum per-class confidence to accept potential detections. Since self-labeling must perform object detection on unlabeled images, these thresholds are needed. During a self-labeling cycle, N self-labeled images are randomly selected, from which n are kept to retrain the object detector. Self-training and co-training use different criteria to select such n self-labeled images. $\mathscr{H}_{stp}$ consists of the parameters required to evaluate whether self-labeling should stop or not. Finally, $\mathscr{H}_{seq}$ is an optional set of parameters, only required if $\mathscr{X}^u$ consists of sequences rather than isolated images. Note that, in this

input : Labeled images: $\mathscr{X}^l = \{< \mathrm{I}_i^l, \mathscr{Y}_i^l >\}$
Unlabeled images: $\mathscr{X}^u = \{\mathrm{I}_i^u\}$
Object detection architecture: Φ
Φ Training hyper-par.: $\mathscr{H}_\Phi$
Co-tr. hyp.-p.: $\mathscr{H}_{sl} = \{T, N, n, m, \mathscr{H}_{stp}[, \mathscr{H}_{seq}]\}$
output : New labeled images: $\mathscr{X}^{\hat{l}} = \{< \mathrm{I}_i^{\hat{l}}, \mathscr{Y}_i^{\hat{l}} >\}$
$< \mathscr{X}_1^{\hat{l}}, \mathscr{X}_2^{\hat{l}}, k > \leftarrow < \emptyset, \emptyset, 0 >$;
$< \mathscr{X}_1^l, \mathscr{X}_2^l > \leftarrow < \mathscr{X}^l, (\mathscr{X}^l)^{\curvearrowleft} >$;
$\phi_1 \leftarrow \mathrm{TrainDetector}(\Phi, \mathscr{H}_\Phi, \mathscr{X}_1^l, \mathscr{X}_1^{\hat{l}})$;
$\phi_2 \leftarrow \mathrm{TrainDetector}(\Phi, \mathscr{H}_\Phi, \mathscr{X}_2^l, \mathscr{X}_2^{\hat{l}})$;
$\mathscr{X}_{1,new}^{\hat{l}} \leftarrow \mathrm{RunDetector}(\phi_1, \mathscr{X}^u, T)$;
$\mathscr{X}_{2,new}^{\hat{l}} \leftarrow \mathrm{RunDetector}(\phi_2, \mathscr{X}^u, T)$;
repeat
  $\mathscr{X}_{old}^{\hat{l}} \leftarrow \mathscr{X}_{1,new}^{\hat{l}}$;
  $\mathscr{X}_{1,\uparrow}^{\hat{l}} \leftarrow \mathrm{Select}(\uparrow, m, \mathrm{Rand}(\mathscr{X}_{1,new}^{\hat{l}}, N[, \mathscr{H}_{seq}, k]))$;
  $\mathscr{X}_{2,\uparrow}^{\hat{l}} \leftarrow \mathrm{Select}(\uparrow, m, \mathrm{Rand}(\mathscr{X}_{2,new}^{\hat{l}}, N[, \mathscr{H}_{seq}, k]))$;
  $\mathscr{X}_{1,\downarrow}^{\hat{l}} \leftarrow \mathrm{Select}(\downarrow, n, \mathrm{RunDetector}(\phi_1, \mathscr{X}_{2,\uparrow}^{\hat{l}}, T))$;
  $\mathscr{X}_{2,\downarrow}^{\hat{l}} \leftarrow \mathrm{Select}(\downarrow, n, \mathrm{RunDetector}(\phi_2, \mathscr{X}_{1,\uparrow}^{\hat{l}}, T))$;
  $\mathscr{X}_1^{\hat{l}} \leftarrow \mathrm{Fuse}(\mathscr{X}_1^{\hat{l}}, \mathscr{X}_{1,\downarrow}^{\hat{l}})$;
  $\mathscr{X}_2^{\hat{l}} \leftarrow \mathrm{Fuse}(\mathscr{X}_2^{\hat{l}}, \mathscr{X}_{2,\downarrow}^{\hat{l}})$;
  $\phi_1 \leftarrow \mathrm{TrainDetector}(\Phi, \mathscr{H}_\Phi, \mathscr{X}_1^l, \mathscr{X}_1^{\hat{l}})$;
  $\phi_2 \leftarrow \mathrm{TrainDetector}(\Phi, \mathscr{H}_\Phi, \mathscr{X}_2^l, \mathscr{X}_2^{\hat{l}})$;
  $\mathscr{X}_{1,new}^{\hat{l}} \leftarrow \mathrm{RunDetector}(\phi_1, \mathscr{X}^u, T)$;
  $\mathscr{X}_{2,new}^{\hat{l}} \leftarrow \mathrm{RunDetector}(\phi_2, \mathscr{X}^u, T)$;
until *Stop?*$(\mathscr{H}_{stp}, \mathscr{X}_{old}^{\hat{l}}, \mathscr{X}_{1,new}^{\hat{l}}, k{+}{+})$;
$\mathscr{X}^{\hat{l}} \leftarrow \mathscr{X}_{1,new}^{\hat{l}}$;
return $\mathscr{X}^{\hat{l}}$;

Algorithm 2: Self-labeling by co-training (see main text).

case, we can easily avoid to introduce too similar training samples coming from the same object instances in consecutive frames, which has shown to be useful in AL approaches [3].

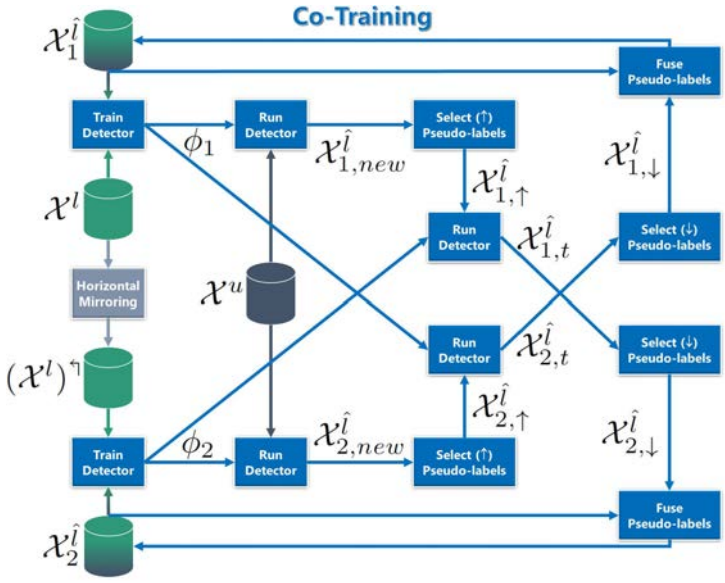

Figure 3.2 – Co-training main components. We use the same notation as in Algorithm 2, but we have introduced two dummy variables for the sake of clarity ($\mathscr{X}^{\hat{l}}_{1,t}, \mathscr{X}^{\hat{l}}_{2,t}$). The data in green is labeled, the one in dark grey is unlabeled, the transition of both colors represents pseudo-labeled data. The blue boxes correspond to the main components involved in co-training according to Algorithm 2. The light grey bounding box is executed just once.

TrainDetector$(\Phi, \mathscr{H}_\Phi, \mathscr{X}^l, \mathscr{X}^{\hat{l}}) : \phi$ This function returns an object detector, ϕ, by training a CNN architecture Φ (*e.g.* Faster R-CNN [120]) according to the training hyper-parameters $\mathscr{H}_\Phi$ (*e.g.* optimizer, learning rate, mini-batch size, training iterations, etc.). Note that this is just the standard manner of training ϕ, but as part of our self-labeling procedures, we control the provided training data. In particular, we use labels (in $\mathscr{X}^l$) and pseudo-labels (in $\mathscr{X}^{\hat{l}}$) indistinctly. However, we only consider background samples based on $\mathscr{X}^l$, since, during self-labeling, $\mathscr{X}^{\hat{l}}$ can contain false negatives which could be erroneously taken as hard negatives when training ϕ. Despite this control over the training data, note that we neither require custom modifications of the loss function used to train ϕ, nor architectural modifications of Φ.

RunDetector$(\phi, \mathscr{X}^u, T): \mathscr{X}^{\hat{l}}$ This function returns a set of self-labeled images, $\mathscr{X}^{\hat{l}}$, obtained by running the object detector ϕ on the unlabeled set of images $\mathscr{X}^u$; in other words, the pseudo-labels correspond to the object detections. Each detection D consists of a BB B and a class label c; moreover, being a detection implies that the confidence $\phi(D)$ fulfils the condition $\phi(D) \geq t_c, t_c \in T$.

Rand$(\mathscr{X}^{\hat{l}}, N[, \mathscr{H}_{seq}, k]): \mathscr{X}_s^{\hat{l}}$ This function returns a set $\mathscr{X}_s^{\hat{l}}$ of N self-labeled images randomly chosen from $\mathscr{X}^{\hat{l}}$. $\mathscr{H}_{seq}$ and k are optional parameters. If they are provided, it means that $\mathscr{X}^{\hat{l}}$ consists of sequences and we want to ensure not to return self-labeled consecutive frames, since they may contain very similar samples coming from the same object instances and we want to favor variability in the samples. Moreover, in this way we can minimize the inclusion of spurious false positives that may persist for several frames. In this case, $\mathscr{H}_{seq} = \{\Delta t_1, \Delta t_2\}$; where Δt_1 is the minimum distance between frames with pseudo-labels generated in the current cycle, k, and Δt_2 is the minimum distance between frames with pseudo-labels generated in cycle k and frames with pseudo-labels generated in previous cycles ($< k$). Δt_1 condition is applied first, then Δt_2 one. The final random selection is performed on the frames remaining after applying these distance conditions.

Select$(s, n, \mathscr{X}^{\hat{l}}): \mathscr{X}_s^{\hat{l}}$ This function returns a sub-set $\mathscr{X}_s^{\hat{l}} \subset \mathscr{X}^{\hat{l}}$ of m self-labeled images, selected according to a criterion s. If $s = \uparrow$, the top n most confident images are selected; while for $s = \downarrow$, the n least confident images are selected. We resume the confidence of an image from the confidences of its detections. For the sake of simplicity, since $\mathscr{X}^{\hat{l}} = \{< \mathrm{I}_i^{\hat{l}}, \mathscr{Y}_i^{\hat{l}} >\}$, we can assume that $\mathscr{Y}_i^{\hat{l}}$ not only contains the BB and class label for each detected object in $\mathrm{I}_i^{\hat{l}}$, but also the corresponding detection confidence. Then, the confidence assigned to $\mathrm{I}_i^{\hat{l}}$ is the average of the confidences in $\mathscr{Y}_i^{\hat{l}}$.

Fuse$(\mathscr{X}_{old}^{\hat{l}}, \mathscr{X}_{new}^{\hat{l}}): \mathscr{X}^{\hat{l}}$ The goal of this function is to avoid redundant detections between the self-labeled sets $\mathscr{X}_{old}^{\hat{l}}$ and $\mathscr{X}_{new}^{\hat{l}}$ and preserving different ones, producing a new set of self-labeled images, $\mathscr{X}^{\hat{l}}$. Thus, on the one hand, $\mathscr{X}^{\hat{l}}$ will contain all the self-labeled images in $\mathscr{X}_{old}^{\hat{l}} \cup \mathscr{X}_{new}^{\hat{l}} - \mathscr{X}_{old}^{\hat{l}} \cap \mathscr{X}_{new}^{\hat{l}}$; on the other hand, for those images in $\mathscr{X}_{old}^{\hat{l}} \cap \mathscr{X}_{new}^{\hat{l}}$, only the detections in $\mathscr{X}_{new}^{\hat{l}}$ are kept, so the corresponding information is added to $\mathscr{X}^{\hat{l}}$.

Stop?$(\mathscr{H}_{stp}, \mathscr{X}^{\hat{l}}_{old}, \mathscr{X}^{\hat{l}}_{new}, k)$: **Boolean** This function decides whether to stop self-labeling or not. The decision relies on two conditions. The first one is if a minimum number of self-labeling cycles, K_{min}, have been performed, where current number is k. The second condition relies on the similarity of $\mathscr{X}^{\hat{l}}_{old}$ and $\mathscr{X}^{\hat{l}}_{new}$, computed as the mAP (mean average precision) [46] by using $\mathscr{X}^{\hat{l}}_{old}$ in the role of ground truth and $\mathscr{X}^{\hat{l}}_{new}$ in the role of results to be evaluated. We compute the absolute value difference between the mAP at current cycle and previous one, keeping track of these differences in a sliding window fashion. If these differences are below a threshold, $T_{\Delta_{mAP}}$, for more than a given number of consecutive cycles, ΔK, self-labeling will stop. Therefore, $\mathscr{H}_{stp} = \{K_{min}, T_{\Delta_{mAP}}, \Delta K\}$.

3.3.2 Self-training

Algorithm 1 describes self-training based on the functional components introduced in Section 3.3.1. In the following, we highlight several points of this algorithm.

As Select(,,) is run in the loop, it implies that in each cycle the n most confident self-labeled images are kept for retraining ϕ. Moreover, potentially redundant object detections arising from different cycles are resumed by running the Fuse(,) function. As is called Fuse(,) in the loop, when a previous detection and a new one overlap enough, the new one is kept since it is based on the last trained version of ϕ, which is expected to be more accurate than previous ones. When calling Select(,,), not all the available self-labeled images are considered, but just a random set of them of size N; which are chosen taking into account the selection conditions introduced in case of working with sequences. This prevents the same highly accurate detections to be systematically selected across cycles, which would prevent the injection of variability when retraining ϕ. The random selection over the entire $\mathscr{X}^u$ was also proposed in the original co-training algorithm [17]. As we have mentioned before, when running TrainDetector(,,,), background samples are only collected from $\mathscr{X}^l$ to avoid introducing false positives. Moreover, we set $\mathscr{H}_\Phi$ to ensure that all the training samples are visited at least once (mini-batch size × number of iterations ≥ number of training images). Therefore, we can think of $\mathscr{X}^l \cup \mathscr{X}^{\hat{l}}$ as a mixing of data, where $\mathscr{X}^l$ acts as a regularizing factor during training, which aims to prevent ϕ to become an irrecoverably bad detector. Thinking in a virtual-to-real UDA setting, we note that from traditional ML algorithms to modern deep CNNs, mixing virtual and real data has been shown to be systematically beneficial across different computer vision tasks [68, 123, 138, 155]. Finally, we can see that to run the stopping criterion we rely on the full self-labeled data available at the beginning and end of each cycle, which results from applying the last available version of ϕ to the full $\mathscr{X}^u$ set.

3.3.3 Co-training

Algorithm 2 describes our co-training proposal based on the functional components introduced in Section 3.3.1. In the following, we highlight several points of this algorithm.

Our co-training strategy tries to make ϕ_1 and ϕ_2 different by training them on different data. Note how each ϕ_i is trained on $\mathscr{X}_i^l$ and $\mathscr{X}_i^{\hat{l}}$, $i \in \{1,2\}$, at each cycle. The $\mathscr{X}_i^l$'s do not change across cycles and we have $\mathscr{X}_1^l = \mathscr{X}^l$ and $\mathscr{X}_2^l = (\mathscr{X}^l)^{\curvearrowleft}$, thus, one of the labeled sets is a horizontal mirroring of the other. Moreover, the $\mathscr{X}_i^{\hat{l}}$'s are expected to be different from each other and changing from cycle to cycle. On the other hand, even technically not using the same exact data to train ϕ_1 and ϕ_2, there will be a cycle when they converge and are not able to improve each other. This is what we check to stop, *i.e.* as for self-training, but focusing on the performance of ϕ_1 since it is the detector using $\mathscr{X}^l$ (the original labeled dataset).

Another essential question is how each $\mathscr{X}_i^{\hat{l}}$ is obtained. As we have mentioned before, we follow the idea of disagreement. Thus, from the random set of images self-labeled by ϕ_i, *i.e.* $\mathscr{X}_{i,new}^{\hat{l}}$, $i \in \{1,2\}$, we set to $\mathscr{X}_{i,\uparrow}^{\hat{l}}$ the m with overall higher confident pseudo-labels; then, the images in $\mathscr{X}_{i,\uparrow}^{\hat{l}}$ are processed by ϕ_j, $j \in \{1,2\}$, $i \neq j$, and the n with the overall lower confident pseudo-labels are set to $\mathscr{X}_{j,\downarrow}^{\hat{l}}$. Finally, each $\mathscr{X}_{i,\downarrow}^{\hat{l}}$ is fused with the $\mathscr{X}_i^{\hat{l}}$ accumulated from previous cycles to update $\mathscr{X}_i^{\hat{l}}$. Therefore, in each cycle, each ϕ_i is retrained with the images containing the less confident pseudo-labels for current ϕ_i, selected among the images containing the most confident pseudo-labels for current ϕ_j. In this way, the detectors trust each other, and use the samples that are more difficult for them for improving.

Note also that the most costly processing in self-training and co-training cycles is the TrainDetector(,,,) procedure. It is called once per cycle for self-training, and twice for co-training. However, in the latter case the two executions can run totally in parallel, therefore, with the proper hardware resources, self-training and co-training cycles can be performed in a very similar time.

3.3.4 Self-labeling for UDA

Once introduced our self-training and co-training algorithms, we see how using them for UDA is just a matter of providing the proper input parameters; *i.e.*, $\mathscr{X}^l = \mathscr{X}_S^{l,tr}$, $\mathscr{X}^u = \mathscr{X}_T$, being $\mathscr{X}_S^{l,tr}$ the source-domain labeled dataset, and $\mathscr{X}_T$ the target-domain unlabeled one. We will draw the former from a virtual world, and the latter from the real world.

Figure 3.3 – Top images: virtual-world patches showing 3D BBs framing vehicles and pedestrians. Bottom image: projecting 3D BBs as 2D BBs for different views of a pickup, with instance segmentation as visual reference.

3.4 Experiments

3.4.1 Experimental setup

As real-world datasets we use KITTI [46] and Waymo [142], in the following denoted as $\mathcal{K}$,$\mathcal{W}$, respectively. The former is a well-established standard born in the academia; the latter is a new contribution from the industry. To work with $\mathcal{K}$, we follow the splits introduced in [170], which are based on isolated images, and guarantee that training and testing images are not highly correlated; *i.e.*, training and testing images are not just different frames sampled from the same sequences. On the other hand, $\mathcal{W}$ dataset contains video sequences. Following the advice for its use, we have randomly selected part of the sequences for training and the rest for testing. Taking $\mathcal{K}$ as reference, we focus on daytime and non-adverse weather conditions in all the cases. In addition, we have accommodated $\mathcal{W}$ images to the resolution of $\mathcal{K}$ ones, *i.e.* 1240×375 pixels, by just cropping away their upper part (which mainly shows sky) and keeping the vertically centered 1240 columns. Following KITTI benchmark moderate settings, we set the minimum BB height to detect objects to 25 pixels and, analogously, 50 pixels for $\mathcal{W}$. Moreover, since $\mathcal{W}$ contains 3D BBs, as for the virtual-world objects (Figure 3.3), we obtain 2D BBs from them. In other words, the resulting 2D BBs account for occluded object areas, which is not the case for the 2D BBs directly available with $\mathcal{W}$.

In order to perform our experiments, as set of virtual-world images ($\mathcal{V}$), we have used a dataset that we generated internally (refer to appendix A to see more details) around two years ago as a complement for our former SYNTHIA dataset [123]. We did not have the opportunity to release it at that moment, but it will be publicly available to complement this chapter. We generated this data following KITTI parameters: same image resolution, daytime and non-adverse weather, and isolated images. As in [123], we did not focus on obtaining photo-realistic images. However, for generating $\mathcal{V}$, we included standard visual post-effects such as anti-aliasing, ambient occlusion, depth of field, eye adaptation, blooming and chromatic aberration. Later, we will see how the domain shift between $\mathcal{V}$ and $\mathcal{K}$/$\mathcal{W}$, is similar to the one between $\mathcal{K}$ and $\mathcal{W}$; thus, being $\mathcal{V}$ a proper virtual-world dataset for our study. Figure 3.3 shows virtual-world images with 3D BBs framing vehicles and pedestrians, illustrating also how the 3D BBs are projected as 2D BBs covering occluded object areas.

For each dataset, Table 3.1 summarizes the number of main/sections4/images/ frames and objects (vehicles and pedestrians) used for training and testing our object detectors.

Table 3.1 – Datasets ($\mathcal{X}$): train ($\mathcal{X}^{tr}$) and test ($\mathcal{X}^{tt}$) information, $\mathcal{X} = \mathcal{X}^{tr} \cup \mathcal{X}^{tt}, \mathcal{X}^{tr} \cap \mathcal{X}^{tt} = \emptyset$. We show the number of main/sections4/images/frames (sequences), vehicle BBs, pedestrian BBs, and whether the datasets consists of video sequences or not.

	$\mathcal{X}^{tr}$			$\mathcal{X}^{tt}$			
Dataset ($\mathcal{X}$)	Images (seq.)	Vehicles	Pedestrians	Images (seq.)	Vehicles	Pedestrians	Seq.?
VIRTUAL ($\mathcal{V}$)	19,791	43,326	44,863				No
KITTI ($\mathcal{K}$)	3,682	14,941	3,154	3,799	18,194	1,333	No
WAYMO ($\mathcal{W}$)	9,873 (50)	64,446	9,918	4,161 (21)	24,600	3,068	Yes

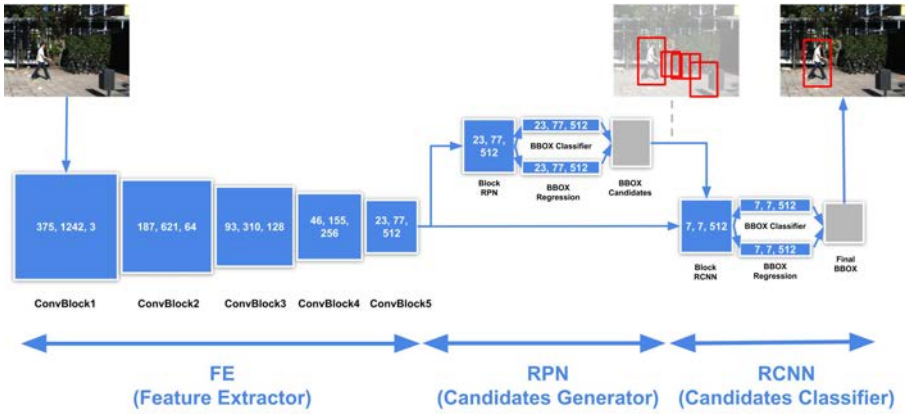

Figure 3.4 – Main components of Faster R-CNN: feature extractor (FE), region proposal network (RPN), and region-based CNN (RCNN). Their responsibilities are outlined in parenthesis and elaborated in the main text. We use VGG16 as FE. Blue boxes are blocks of neural network layers with input dimensions indicated as <height, width, channels>. Grey boxes are algorithmic steps to return BBs, candidates (RPN) or detections (RCNN).

Since our proposal does not assume a particular object detection architecture, for performing the experiments we have chosen Faster R-CNN [120] because it provides a competitive detection accuracy and is very well known by computer vision practitioners [88]. For the sake of completeness, Figure 3.4 shows the main components of Faster R-CNN, namely, the Feature Extractor (FE), the region proposal network (RPN), and the region-based CNN (RCNN). We use the implementation available in the *Detectron* framework [49], with VGG16 as FE. The RPN component generates object candidates from the so-called bottleneck of FE (*i.e.* ConvBlock5). Conceptually, these candidates can be understood as BBs which can potentially contain objects of interest. Using these candidates and the same bottleneck, the RCNN component performs the final object classification and BB regression (a refinement of the BBs proposed by RPN). For comprehensive details the reader can refer to [49]. At training time, we always initialize VGG16 (FE) with ImageNet weights since it is a recommended best practice. The weights of the RPN and RCNN components are randomly initialized. We run each Faster R-CNN training for 40,000 iterations using SGD optimizer. The learning rate starts at 0.001 and we have used a decay of 0.1 at iterations 30,000 and 35,000. Each iteration uses a mini batch of two images. In terms of Algorithms 1 and 2, these settings correspond to $\mathscr{H}_{\Phi}$.

As GAN-based image-to-image translation method we use the CycleGAN implementation in [186]. A GAN training runs for 40 epochs using a weight of 1.0 for the identity mapping loss. In order to be able to use full resolution images, the training of the GAN has been done patch-wise, with patch sizes of 300 × 300 pixels. The rest of parameters have been set as recommended in [186]. We train a $\mathcal{V} \rightarrow \mathcal{K}$ transforming GAN, $\mathcal{G}_{\mathcal{K}}$, and another for $\mathcal{V} \rightarrow \mathcal{W}$, $\mathcal{G}_{\mathcal{W}}$. Accordingly, $\mathcal{V}_{\mathcal{G}_{\mathcal{K}}} = \mathcal{G}_{\mathcal{K}}(\mathcal{V})$ denotes the set of virtual-world images transformed by $\mathcal{G}_{\mathcal{K}}$ and, analogously, we have $\mathcal{V}_{\mathcal{G}_{\mathcal{W}}} = \mathcal{G}_{\mathcal{W}}(\mathcal{V})$. We will use the notation $\mathcal{V}_{\mathcal{G}}$ to refer to any of these sets. The ground truth present in $\mathcal{V}$ is used as ground truth for $\mathcal{V}_{\mathcal{G}}$. For qualitative examples of image-to-image transformations, we refer to Figures 3.5 and 3.6, using $\mathcal{G}_{\mathcal{K}}$ and $\mathcal{G}_{\mathcal{W}}$, respectively.

Table 3.2 summarizes the rest of hyper-parameters used to perform our experiments. Note that, regarding our self-labeling approaches, we use the same values for $\mathcal{K}$ and $\mathcal{W}$, with the only exception that for $\mathcal{W}$ we also consider that it is composed of video sequences. Moreover, we also use the same values for hyper-parameters shared by self-training and co-training. We relied on the meaning of the hyper-parameters to set them with reasonable values. Then, during the experiments of the 5% with self/co-training (see Table 3.3) we did visual inspection of the self-labeled images to ensure the methods were working well. In this process, we noted that, since the confidence thresholds (T) were already avoiding too erroneous self-labeled images, for co-training it was better to send all the images self-labeled by the detector ϕ_i to detector ϕ_j, $i, j \in \{1,2\}, i \neq j$, so that ϕ_j can select the n most difficult for it among them. This is equivalent to set $m = \infty$ in Algorithm 2 (so nullifying the parameter).

Algorithms 1 and 2 return self-labeled images, *i.e.* $\mathcal{X}^{\hat{l}}$. For our experiments, we use $\mathcal{X}^{\hat{l}}$ together with the input labeled set, $\mathcal{X}^{l}$, to train a final Faster R-CNN object detector out of the self-labeling cycles but using the same training settings. This detector is the one used for testing. Finally, to measure object detection performance, we follow the KITTI benchmark mean average precision (mAP) metric [46].

3.4.2 Results

We start by assessing the self-labeling algorithms in a pure SSL setting, working only with either the $\mathcal{K}$ or $\mathcal{W}$ dataset. Table 3.3 shows the results when using the 100% of the respective real-world training data, *i.e.* $\mathcal{X}^{l} = \mathcal{X}^{tr} \in \{\mathcal{K}^{tr}, \mathcal{W}^{tr}\}$, as well as when randomly selecting the 5% or the 10% of $\mathcal{X}^{tr}$ as $\mathcal{X}^{l}$ set, meaning that these subsets are created once and frozen for all the experiments. Table 3.4 shows the number of object instances induced by the random selection in each case. In Table 3.3, the

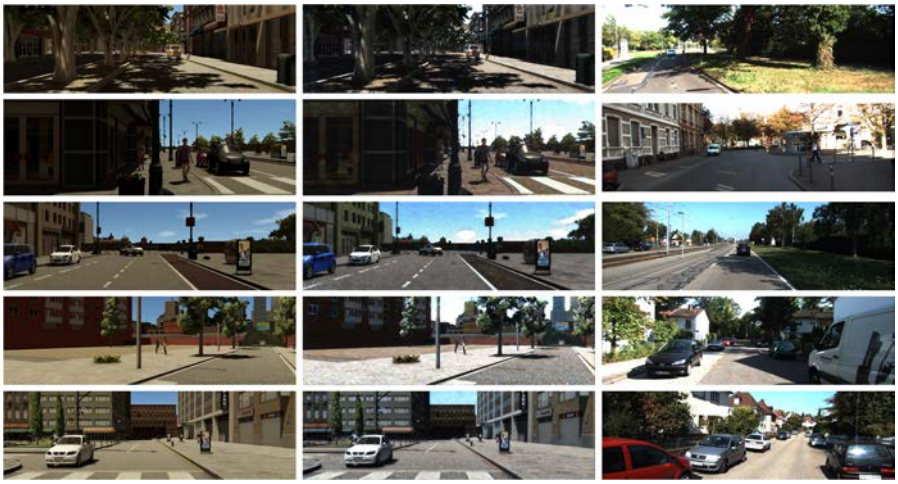

Figure 3.5 – From left to right: images from $\mathcal{V}$, corresponding images in $\mathcal{V}_{\mathcal{G}_\mathcal{K}}$ (*i.e.* processed by the $\mathcal{V} \rightarrow \mathcal{K}$ GAN), and images from $\mathcal{K}$. Last column is just a visual reference since, obviously, there is no a one-to-one correspondence between $\mathcal{V}$ and $\mathcal{K}$.

Figure 3.6 – From left to right: images from $\mathcal{V}$, corresponding images in $\mathcal{V}_{\mathcal{G}_\mathcal{W}}$ (*i.e.* processed by the $\mathcal{V} \rightarrow \mathcal{W}$ GAN), and images from $\mathcal{W}$. For visual comparison, the two top rows of this figure and those in Figure 3.5, start with the same images in $\mathcal{V}$.

Table 3.2 – Self-training and co-training hyper-parameters as defined in Algorithms 1 and 2. We use the same values for both, as well as to work with KITTI ($\mathscr{K}$) and Waymo ($\mathscr{W}$) datasets, except for $\mathscr{H}_{seq}$ which only applies to $\mathscr{W}$. N, n, m, Δt_1, and Δt_2 are set in number-of-images units, K_{min} and ΔK in number-of-cycles, $T_{\Delta_{mAP}}$ runs in [0..100]. We use the same confidence detection threshold for vehicles and pedestrians, which runs in [0..1]. (*) Only used in co-training, however, for $m = \infty$, it has no effect.

				$\mathscr{H}_{stp}$			$\mathscr{H}_{seq}$	
T	N	n	$m^\star$	K_{min}	$T_{\Delta_{mAP}}$	ΔK	Δt_1	Δt_2
{0.8,0.8}	2,000	100	∞	20	2.0	5	5	10

Table 3.3 – SSL results for $\mathscr{K}$ and $\mathscr{W}$. We assess vehicle (V) and pedestrian (P) detection, according to the mAP metric. From $\mathscr{X}^{tr} \in \{\mathscr{K}^{tr}, \mathscr{W}^{tr}\}$, we preserve the labeling information for a randomly chosen p% of its images, while it is ignored for the rest. We report results for p=100 (all labels are used), p=5 and p=10. Slf-T and Co-T stand for self-training and co-training, resp., which refers to how images were self-labeled from the respective unlabeled training sets.

	$\mathscr{X}^{tt} = \mathscr{K}^{tt}$			$\mathscr{X}^{tt} = \mathscr{W}^{tt}$		
Training set	V	P	V&P	V	P	V&P
100% Labeled (upper bound)	81.02	65.40	73.21	61.71	57.74	59.73
5% Labeled (lower bound)	59.81	36.49	48.15	51.69	41.92	46.81
5% Labeled + Slf-T	68.94	40.99	54.97	**54.26**	55.38	54.82
5% Labeled + Co-T	**68.99**	**55.07**	**62.03**	54.00	**56.34**	**55.17**
10% Labeled (lower bound)	72.13	49.03	60.58	49.53	49.83	49.68
10% Labeled + Slf-T	**76.32**	56.05	**66.19**	54.88	57.40	56.14
10% Labeled + Co-T	73.08	**58.53**	65.81	**56.15**	**60.20**	**58.18**

Table 3.4 – Number of self-labeled vehicles and pedestrians applying self-training and co-training, for the SSL (5% & 10%) and UDA (Source & ASource) settings, for KITTI ($\mathscr{K}$) and Waymo ($\mathscr{W}$). In parenthesis we indicate the percentage of false positives. The top block corresponds to ground truth labels in the full training sets, and the percentages used for SSL. After removing false positives, in each block of rows, the corresponding Δ_X shows how many more objects are labeled by co-training compared to self-training.

	$\mathscr{K}$		$\mathscr{W}$	
Training set	V	P	V	P
100 % Labeled	14,941	3,154	64,446	9,918
5 % Labeled	647	83	3,520	472
10 % Labeled	1,590	367	6,521	959
5% Labeled + Slf-T	9,376 (0.8%)	837 (0.5%)	52,150 (1.7%)	5,554 (3.1%)
5% Labeled + Co-T	10,960 (1.8%)	1,591 (6.2%)	56,102 (4.0%)	6,948 (10.9%)
$\Delta_{5\%}$	+1,462	+660	+2,594	+809
10% Labeled + Slf-T	10,536 (0.7%)	1,409 (2.3%)	54,698 (1.6%)	6,519 (2.7%)
10% Labeled + Co-T	11,309 (1.7%)	1,552 (3.9%)	56,686 (3.1%)	7,384 (6.0%)
$\Delta_{10\%}$	+654	+115	+1,106	+598
Slf-T + Source	12,686 (4.6%)	1,378 (8.9%)	29,036 (23.8%)	837 (5.1%)
Co-T + Source	14,537 (6.5%)	1,964 (14.0%)	48,653 (30.4%)	3,481 (12.9%)
Δ_{Source}	+1,490	+434	+1,1737	+2,238
Slf-T + ASource	10,389 (1.7%)	1,268 (3.5%)	41,173 (24.7%)	1,928 (13.5%)
Co-T + ASource	12,864 (3.1%)	1,532 (5.2%)	53,955 (28.8%)	3,553 (16.0%)
$\Delta_{ASource}$	+2,253	+229	+7,413	+1,317

100% case shows the upper-bound performance, and the 5% and 10% act as lower bounds.

We can see that, in all the cases, the self-labeling methods outperform the lower bounds. For vehicles (V), self-training and co-training perform similarly for the 5% lower bound, while for the 10% self-training performs better than co-training in $\mathscr{K}$ but worse in $\mathscr{W}$. For pedestrians (P), co-training always performs better. Looking at the vehicle-pedestrian combined (V&P) performance, we see significant improvements over the lower bounds. Interestingly, for the 10% setting, co-training even outperforms the upper bound for pedestrians in $\mathscr{W}$. In fact, in this case, the corresponding V&P performance is just 1.55 points below the upper bound. The same setting in $\mathscr{K}$, improves 5.3 points over the lower bound, but is 7.4 points below the upper bound. These experiments show that our self-labeling

Table 3.5 – UDA results for $\mathcal{V} \rightarrow \{\mathcal{K}, \mathcal{W}\}$, *i.e.* virtual to real. ASource (*adapted source*) refers to $\mathcal{V}_{\mathcal{G}} \in \{\mathcal{V}_{\mathcal{G}_\mathcal{K}}, \mathcal{V}_{\mathcal{G}_\mathcal{W}}\}$. $\mathcal{X}^{l,tr}$ refers to the fully labeled target-domain training set. $\mathcal{X}^{\hat{l},tr}$ consists of the same images as $\mathcal{X}^{l,tr}$, but self-labeled by either self-taining (Slf-T) or co-training (Co-T). Just as reference, we also show the domain shift between $\mathcal{K}$ and $\mathcal{W}$. According to these results, as upper bound for $\mathcal{K}$ we take the detector based on $\mathcal{X}^{l,tr}\&\mathcal{V}_{\mathcal{G}}$, while for $\mathcal{W}$ it is the detector based on $\mathcal{X}^{l,tr}$. We refer to the main text for more details.

	$\mathcal{X}^{tt} = \mathcal{K}^{tt}$			$\mathcal{X}^{tt} = \mathcal{W}^{tt}$		
Training set	V	P	V&P	V	P	V&P
Source ($\mathcal{K}$)	-	-	-	37.79	49.80	43.80
Source ($\mathcal{W}$)	45.57	44.11	44.84	-	-	-
Source ($\mathcal{V}$) (lower bound)	62.27	61.28	64.28	38.88	53.37	46.13
ASource ($\mathcal{V}_{\mathcal{G}}$)	75.52	61.94	68.73	52.42	53.08	52.75
Target ($\mathcal{X}^{l,tr}$)	81.02	65.40	73.21	†61.71	†**57.74**	†**59.73**
Target + Source ($\mathcal{X}^{l,tr}\&\mathcal{V}$)	81.68	**68.07**	74.88	60.65	54.23	57.44
Target + ASource ($\mathcal{X}^{l,tr}\&\mathcal{V}_{\mathcal{G}}$)	†**84.08**	†66.00	†**75.04**	**62.02**	51.55	56.79
Slf-T + Source ($\mathcal{X}^{\hat{l},tr}\&\mathcal{V}$)	70.25	67.59	68.92	48.50	44.63	46.57
Co-T + Source ($\mathcal{X}^{\hat{l},tr}\&\mathcal{V}$)	**73.53**	**69.50**	**71.52**	**48.56**	**56.33**	**52.45**
Slf-T + ASource ($\mathcal{X}^{\hat{l},tr}\&\mathcal{V}_{\mathcal{G}}$)	79.62	65.87	72.75	59.19	52.48	55.84
Co-T + ASource ($\mathcal{X}^{\hat{l},tr}\&\mathcal{V}_{\mathcal{G}}$)	**79.99**	**69.01**	**74.50**	**59.99**	**55.39**	**57.69**
Δ(Co-T + ASource) vs Source	+17.72	+07.73	+10.22	+21.11	+02.02	+11.56
Δ(Co-T + ASource) vs ASource	+04.47	+07.07	+05.77	+07.57	+02.31	+04.94
Δ(Co-T + ASource) vs upp.-b.†	-04.09	+03.01	-00.54	-01.72	-02.35	-02.04
Δ(Co-T + Source) vs Source	+11.26	+08.22	+07.22	+09.68	+02.96	+06.32
ΔASource vs Source	+13.25	+00.66	+04.45	+13.54	-00.29	+06.62

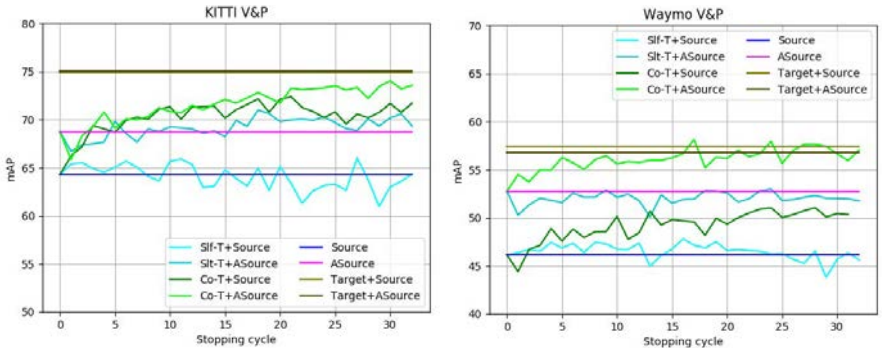

Figure 3.7 – Eventual detection performance (mAP) of self-training and co-training as a function of the stopping cycle, in the UDA setting. Upper and lower bounds are included as visual reference. We refer to the main text for more details.

algorithms, especially co-training, are performing the task of SSL reasonably well, which encourages to address the UDA challenge with them.

Table 3.5 shows the UDA results for $\mathcal{V} \rightarrow \mathcal{K}$ and $\mathcal{V} \rightarrow \mathcal{W}$, thus, training with $\mathcal{V}$ acts as lower bound. Just as reference, we also show the results of training on $\mathcal{K}$ and testing on $\mathcal{W}$, and vice versa. The former case shows a similar domain shift as when training on $\mathcal{V}$. The latter case shows a significant lower shift from $\mathcal{V}$ to $\mathcal{K}$, than from $\mathcal{W}$ to $\mathcal{K}$. Thus, we think that $\mathcal{V}$ offers a realistic use case to assess virtual-to-real UDA. The Δ_X rows at the bottom block of Table 3.5 summarize numerically the main insights.

A first observation is that, by combining GAN-based image-to-image translation and co-training, we obtain significant performance improvements over the lower bounds in all cases; in terms of V&P, 10.22 points for $\mathcal{K}$ and 11.56 for $\mathcal{W}$. In fact, the gap to reach upper-bound performances, is relatively small compared to the improvement over the lower bounds; in terms of V&P, such gap is of 0.54 points for $\mathcal{K}$ and of 2.04 for $\mathcal{W}$. Note that for $\mathcal{K}$, the upper bound comes from the $\mathcal{X}^{l,tr}$&$\mathcal{V}_{\mathcal{G}}$ setting (*i.e.* training on the full labeled training set of $\mathcal{K}$ plus $\mathcal{V}_{\mathcal{G}_{\mathcal{K}}}$); while for $\mathcal{W}$, the upper bound comes from the $\mathcal{X}^{l,tr}$ setting (*i.e.* training on the full labeled training set of $\mathcal{W}$). Thus, without any manual training data labeling, we are almost reaching upper-bound performance.

Figure 3.8 – Examples of Self/Co-training + ASource. Red BBs are from the ground truth of $\mathscr{X}^{tr}$, and green BBs are predicted. Each block of two columns with the same underlying image compares self-training (left column of the block) and co-training (right column of the block). Top row corresponds to detections in Cycle 0, when, in these examples, the only available training data is $\mathscr{V}_{\mathscr{X}}$ (so it is the same for self-training and co-training). The following rows, top to bottom, correspond to detections from cycles 1, 10, and 20, respectively, when self-labeled images are incrementally added to the training set.

Figure 3.9 – Analogous to Figure 3.8 for $\mathscr{W}^{tr}$ and $\mathcal{V}_{\mathscr{G}_{\mathscr{W}}}$.

A second observation is that, indeed, co-training brings additional improvements on top of image-to-image translation; in terms of V&P, 5.77 points for $\mathcal{K}$ and 4.94 for $\mathcal{W}$. Note that, when applying them separately to the virtual-world images (source domain), both co-training and CycleGAN (ASource) show similar performance improvements for $\mathcal{W}$ but co-training outperforms CycleGAN in $\mathcal{K}$; in terms of V&P, co-training improves 7.22 points for $\mathcal{K}$ and 6.32 for $\mathcal{W}$, while CycleGAN improves 4.45 and 6.62 points, respectively. Interestingly, CycleGAN performs better than co-training for vehicles and it is the opposite for pedestrians. In any case, using both together outperforms them in isolation.

A third observation is that co-training consistently outperforms self-training. Moreover, in Figure 3.7 we can see that this is also consistent along self-labeling cycles (we show the UDA setting). It plots curves illustrating how the self-training and co-training strategies would perform as a function of the stopping cycle. We collect the respective self-labeled images at different cycles (x-axis) and train an object detector as these cycles were determined as the stopping ones. Then, we assess the performance of the detector in the corresponding testing set (either from $\mathcal{K}$ or $\mathcal{W}$), so collecting the corresponding mAP (y-axis) per each considered cycle. We see how self-training curves oscillate more around the respective lower bounds, while co-training ones keep improving performance as we train more cycles. Moreover, in Table 3.4, we can see how co-training systematically self-labels more correct object instances than self-training (Δ_X rows show the increment for SSL and UDA settings).

Figures 3.8 and 3.9 show qualitative examples of how self-training and co-training progress in $\mathcal{K}^{tr}$ and $\mathcal{W}^{tr}$ images, respectively; in this case, starting with $\mathcal{V}_{\mathcal{G}_\mathcal{K}}$ and $\mathcal{V}_{\mathcal{G}_\mathcal{W}}$ as corresponding initial labeled training sets. Both self-labeling strategies improve over the starting point, since they can correct BB localization errors, remove false positives, and recovering from false negatives. In some cases, self-training and co-training show the same final right result, but co-training reaches it in earlier cycles, while in other cases co-training shows better final results.

As summarized in Table 3.6, with additional experiments, we have further analyzed co-training results. We repeat the training and evaluation of all the detectors developed for both the SSL and UDA settings, considering two variants in the respective training sets. In (/FP) we remove the false positives from the self-labeled data. In (/FP+BB), in addition, for those self-labeled instances that are true positives, we replace the BBs predicted during self-labeling by the corresponding BB ground truth. In this way, we can incrementally analyze the effect of false positives and BB adjustment accuracy.

Table 3.6 shows how the impact of having false positives as training data is not as strong as one may think a priori. For instance, in Table 3.4 we can see that co-training with CycleGAN (Co-T + ASource) has output a 28.8% of false vehicles in

Table 3.6 – Results for two new settings: (/FP) assuming we remove the self-labeled false positives; (/FP+BB) assuming that, in addition, for the self-labeled instances, we change the predicted BB by the corresponding one in the ground truth. The Δ_X rows show differences between these variants and the respective original one (*i.e.* neither removing the FP nor adjusting the BBs). The bottom block of rows remarks the differences between the best self-labeling (Co-T+ASource, including /FB and /FB+BB cases) and the upper bound.

	$\mathscr{X}^{tt} = \mathscr{K}^{tt}$			$\mathscr{X}^{tt} = \mathscr{W}^{tt}$		
Training set	V	P	V&P	V	P	V&P
Upper bound (UB)	†84.08	†66.00	†75.04	†61.71	†57.74	†59.73
5% Labeled + Co-T	68.99	55.07	62.03	54.00	56.34	55.17
5% Labeled + Co-T/FP	68.37	55.34	61.86	53.94	55.23	54.59
5% Labeled + Co-T/FP+BB	**82.98**	**59.24**	**71.11**	**62.72**	**58.54**	**60.63**
$\Delta_{5\%,\ FP}$	-0.62	+0.27	-0.17	-0.06	-1.11	-0.58
$\Delta_{5\%,\ FP+BB}$	+13.99	+4.17	+9.08	+8.72	+2.20	+5.46
10% Labeled + Co-T	73.08	58.53	65.81	56.15	60.20	58.18
10% Labeled + Co-T/FP	72.94	58.68	65.81	56.62	57.74	57.18
10% Labeled + Co-T/FP+BB	**83.16**	**61.29**	**72.23**	**63.17**	**60.13**	**61.65**
$\Delta_{10\%/FP}$	-0.14	+0.15	0.00	+0.47	-2.46	-1.00
$\Delta_{10\%/FP+BB}$	+10.08	+2.76	+6.42	+7.02	-0.07	+3.47
Co-T + Source	73.53	**69.50**	71.52	48.56	56.33	52.45
Co-T + Source/FP	71.64	**69.50**	70.57	51,06	**56.94**	54.00
Co-T + Source/FP+BB	**86.03**	68.97	**77.50**	**59.21**	56.59	**57.90**
$\Delta_{\text{Co-T + Source/FP}}$	-1.89	0.00	-0.95	+2.50	+0.61	+1.55
$\Delta_{\text{Co-T + Source/FP+BB}}$	+12.50	-0.53	+5.98	+10.65	+0.26	+5.45
Co-T + ASource	79.99	**69.01**	74.50	59.99	55.39	57.69
Co-T + ASource/FP	80,16	67.79	73.98	60.70	56,87	58,79
Co-T + ASource/FP+BB	**86.26**	66.69	**76.48**	**63.98**	**58.55**	**61.27**
$\Delta_{\text{Co-T + ASource/FP}}$	+0.17	-1.22	-0.52	+0.71	+1.48	+1.10
$\Delta_{FP+FP+BB}$	+6.27	-2.32	+1.98	+3.99	+3.16	+3.58
$\Delta_{\text{(Co-T + ASource) vs UB}}$	-4,09	+3,01	-0,54	-1,72	-2,35	-2,04
$\Delta_{\text{(Co-T + ASource/FP) vs UB}}$	-3.92	+1.79	-1.06	-1.01	-0.87	-0.94
$\Delta_{\text{(Co-T + ASource/FP+BB) vs UB}}$	+2.18	-0.69	+1.44	+2.27	+0.81	+1.54

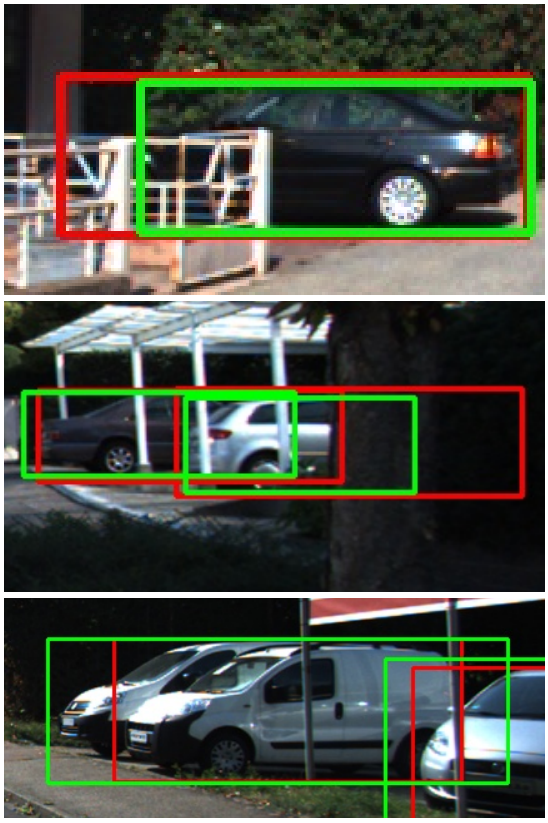

Figure 3.10 – Examples of misalignment between ground truth BBs (red) and self-labeled ones (green). Occlusion is the underlying problem, giving rise to shorter BBs (top and middle) or BBs fusing several instances in one (bottom).

$\mathscr{W}$ and a 3.1% in $\mathscr{K}$, a large difference; however, Table 3.6 indicates that removing them results in a vehicle detection improvement of just 0.71 points in the former case, and 0.17 in the latter. This may be linked with the fact that SGD optimization in deep neural networks (DNNs) tend to prioritize learning patterns instead of noise [8].

In general, as Table 3.6 shows, a better BB adjustment has a higher impact than removing false positives, especially for vehicles. In fact, for the higher performing detector, *i.e.* co-training with CycleGAN, this adjustment would allow to even outperform the upper bound. This improvement is mainly coming from the detection

of vehicles. For instance, Figure 3.10 shows examples of the typical BB misalignment we have found, mainly due to occluded vehicle instances. Note also that the (/FP+BB) results indicate that non-self-labeled objects (false negatives) do not cause any loss of performance, unless they would result in better BB adjustments.

We finish by showing qualitative results on $\mathcal{K}^{tt}$ and $\mathcal{W}^{tt}$ (Figure 3.11), for different object detectors. Comparing ground truth BBs and detections, we appreciate how it is confirmed what is expected from the quantitative results, *i.e.* co-training combined with GAN-based image-to-image translation is providing the most accurate results.

3.5 Conclusion

Motivated by the burden of manual data labeling when addressing vision-based object detection, we have explored co-training as SSL strategy for self-labeling objects in images. Moreover, we have focused on the challenging scenario where the initial labeled set is generated automatically in a virtual world; thus, co-training must actually perform UDA. We have proposed a specific co-training algorithm which is agnostic to the particular object detector used for self-labeling. We have devised a comprehensive set of experiments addressing the challenging task of on-board vehicle and pedestrian detection, using de facto standards such as KITTI and Waymo datasets, together with a virtual-world dataset introduced in this chapter. Our qualitative results allow us to conclude that co-training and GAN-based image-to-image translation complement each other up to allow the training of object detectors without manual labeling, while still reaching almost the same performance as by totally relying on human labeling for obtaining upper-bound performances. These results show that the self-labeled objects are sufficient to train a well-performing object detector, but also that improving BB adjustment is convenient to improve its performance. Accordingly, our future work will address this point, for instance, by developing a multi-modal co-training which jointly explores RGB images (as in this chapter) as well as depth information based on monocular depth estimation [51], where object borders may be better localized.

Figure 3.11 – Results for $\mathcal{K}^{tt}$ (top 'Source → Co-T+Asource' block) and $\mathcal{W}^{tt}$ (bottom 'Source → Co-T+Asource' block). Red BBs are the ground truth, and green ones are the detections done by the detector indicated at the first column of each row.

4 Lidar-based 3D object detection

In order to explore on-board perception beyond vision, we decided to work on 3D object detection based on LiDAR pointclouds. In line with previous chapters, we aim at leveraging synthetic data with automatically generated ground truth to develop such detectors. While for images we may expect synth/real-to-real domain shift due to differences in their appearance, we did not expect so for LiDAR pointclouds since these active sensors factor out appearance and provide sampled shapes. However, after an extensive set of experiments that we present and discuss in this chapter, we conclude that there is domain shift among pointclouds coming from different datasets, *i.e.* not only in the synth-to-real case but also in the real-to-real setting. Factors such as the sampling parameters of the LiDARs and the sensor suite configuration on-board the ego-vehicle, do induce a domain shift. This redirected our initial goal towards the design of a GAN for pointcloud-to-pointcloud translation, a relatively unexplored topic. The first decision was to chose a proper pointcloud representation. After assessing different options, we have used a voxel grid representation, which allows us to process pointclouds of an arbitrary number of points, in contrast to previous related works. Then, we have developed a GAN able to translate between two pointcloud domains which were forced to have shifts in terms of shape, rigid motion, and presence/absence of noisy points.

4.1 Introduction

After addressing traffic sign recognition, a multi-class image classification problem, and vision-based 2D object detection, in both cases focusing on methods to leverage synthetic data for training the respective deep models, a natural step forward is to address LiDAR-based 3D object detection, in this case leveraging synthetic pointclouds for training. We started this work by reviewing the literature to know the publicly available real-world LiDAR-based pointcloud datasets with 3D BB annotations (we found KITTI, Waymo, and Lyft), as well as to know the best

performing detectors. This review of related works is summarized in Sect. 4.2. We will see that a key question is how to represent pointclouds to train deep models. Basically, there are three main approaches and we have selected a top-performing detector representing each one (Frustrum PointNet, PointPillars, PointRCNN). We also developed a dataset of synthetic pointclouds with automatically generated 3D BBs (SYNTHIA-3D), aiming at training the selected detectors with it to latter perform in the real-world datasets. After doing this, we found that there is a domain shift between our synthetic pointclouds and the real-world ones. Then, as sanity check, we performed the same kind of experiments crossing all real-world datasets and, indeed, we found that there is also a real-to-real domain shift problem for LiDAR-based pointclouds. The comprehensive list of experiments is presented in Sect. 4.3. In fact, we will see that the cross-domain drop of accuracy of all the 3D object detectors is relatively large. We did not expect so for LiDAR pointclouds since these active sensors factor out appearance and provide sampled shapes. In the same section, we also show that even compensating for dataset differences on 3D BB margins, which may be due to different criteria during the human annotation process, domain shift still persists. We think it may be because of factors such as the sampling parameters of the LiDARs and the sensor suite configuration on-board the ego-vehicle. After obtaining these results, we searched the literature for domain adaptation methods applied to pointclouds and found that this is still a relatively unexplored topic. We added the related works to Sect. 4.2 too.

Since in previous chapters GANs were helping to perform domain adaptation for vision-based perception tasks, we have decided to use GANs for pointcloud-to-pointcloud translation between a source and a target domain. We present our proposal on Sect. 4.4. The first decision was to chose a proper pointcloud representation. We have used a voxel grid representation, which allows us to process pointclouds of an arbitrary number of points, in contrast to previous related works. Then, we have developed a GAN that takes this representation as input. Our GAN is based on an encoder common to target and source domains and two domain-specific decoders, so not being restricted by cycle or reconstruction losses. The only regularization enforcement comes from autoencoder losses. In order to evaluate such a GAN, we have designed a set of experiments based on synthetic pointclouds. They have shown that the GAN is able to translate between two domains which were forced to have shifts in terms of shape, rigid motion, and presence/absence of noisy points.

Overall, as we will summarize and remark in Sect. 4.5, this chapter shows that domain adaptation is needed for LiDAR-based pointclouds, and that GAN-based pointcloud-to-pointcloud translation is a promising line of work to address the problem.

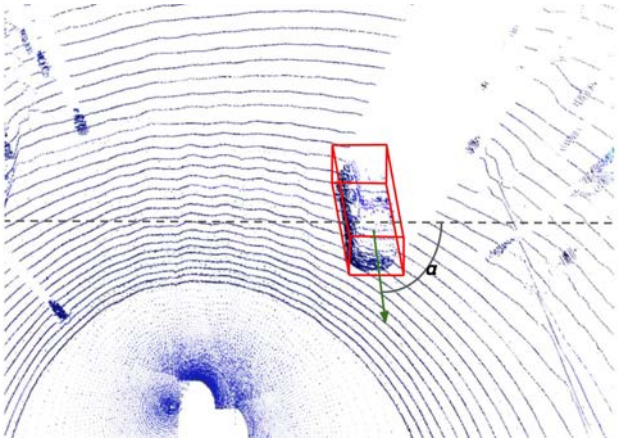

Figure 4.1 – 3D detection case. The position, size and orientation is provided. The position and size is estimated using a bounding box format with its centroid and box size (red lines). The orientation is computed using the angle of the car direction respect to the x-axis of the pointcloud (green arrow).

4.2 Related work

In order to assess how synthetic pointclouds can help to address on-board perception tasks, we have divided the related work in two sections. Section 4.2.1 summarizes works addressing pointcloud-based 3D object detection. Section 4.2.2 focuses on works addressing the domain shift problem for pointclouds.

4.2.1 Working with pointclouds

In the context of AVs, the amount of publicly available LiDAR-based pointclouds with annotations is significantly fewer than for images, where we have a bunch of datasets for classification, detection and segmentation tasks. KITTI [46] has been the first publicly available dataset providing LiDAR-based pointclouds with BB annotations for 3D object detection, and being key for developing monocular depth estimation algorithms. There are other datasets which provide depth information by other methods, as the Cityscapes dataset [36] which relies on a stereo rig, or as the SYNTHIA dataset [123] which uses the z-buffer of a virtual world. In the last two years, different companies developing AVs have started to release on-board data from LiDAR sensors, as illustrate Lyft [67] and Waymo [142] datasets, both including different kinds of annotations. Thanks to these datasets and deep learning, LiDAR-

based classification [26, 116, 169], object detection [82, 115, 131, 175], and semantic segmentation [26, 35, 146], are emerging topics of research with promising results for on-board perception.

Independently of the addressed perception task, processing LiDAR pointclouds is highly demanding in terms of memory, especially compared to images. To solve this problem different representation of the pointclouds have been proposed. A common approach is the use of a *voxel grid*, which was first applied to a classification task by Wu *et al.* [169]. In this approach, the original pointcloud is transformed on a grid of voxels. Each voxel contains the information of a set of 3D points from the original pointcloud, in particular, those falling within a neighbourhood of a given size. Later, this approach has been applied to other tasks such as object detection [184]. Another approach consists in projecting the pointcloud to a 2D space as on a bird-eye view [176, 177] or fusing multiple views [30]. Charles *et al.* [26, 116] propose to sub-sample a random subset of 3D points to be processed by 3D convolutions. Since this approach disregards the 3D topology, different alternatives to 3D convolutions have been proposed, such as KPConv [146] or the Minkowski engine [35]. Following the voxel grid idea, Riegler *et al.* [122] propose to use a octree representation which is an approach leveraging from the voxel grid idea and the direct use of the raw 3D points; in particular, larger voxels are used at 3D areas with existing points, while void voxels are removed. Finally, the implicit representation is commonly used for some tasks [55], however, since it involves processing one 3D point at a time, it turns to be too slow for training CNN-based models.

Focusing on LiDAR-based 3D object detection, we can find works which assume the support of 2D images, works that rely on a voxel grid representation of the pointcloud, and works directly using the pointcloud representation. The most relevant are briefly summarized in the following.

Works based on 2D images. In [30, 79] the key is to fuse image content with LiDAR-based pointclouds projected to the image space. In these works 3D BBs are directly predicted. In [81], image-based 2D object detections are generated first, and then 3D BBs adjusted to them with the support of a corresponding depth image (coming from an indoors active sensor). Following this line, Qui *et al.* [115] proposed a 2D-3D pipeline, first 2D object detection is performed on the images, so that the corresponding 2D BBs delimit a frustrum on the LiDAR-based pointcloud, then for each 2D detection the 3D BB fitting is performed within its frustrum. Obviously, all these approaches assume that LiDAR (or any other active sensor) and camera sensors are calibrated.

Works based on the voxel grid representation. In [39, 157], this representation is used to extract features consisting on different statistics of the 3D points falling in each voxel. In contrast to these works where the voxel features are post-processed in a more ad hoc fashion, in [184] a fully 3D CNN is used to directly process the voxel

features. Yan *et al.* [175] include sparse 3D convolutions to increase the detection speed and accuracy. Lang *et al.* [82] change the 3D convolutions by 2D convolutions working with so-called *pillars*.

Works based on the 3D pointcloud representation. PointNet [26], and its enhanced version PointNet++ [116], work directly on the pointcloud to learn a feature vector for each point and a global feature for the whole pointcloud. The global feature vector is learnt from all point feature vectors, then each of those is augmented by its concatenation with the global feature vector. These augmented point feature vectors are used to obtain the 3D bounding boxes. Shi *et al.* [131] propose the PointRCNN inspired in the two-stage image-based object detectors, such as Faster R-CNN [119], where there is a object candidate proposal and a further regression to improve the proposals and classify them. In this case, the role of backbone network is based on the local-global feature concatenation of PointNet.

In order to perform 3D object detection, for the sake of completeness, in this chapter we have selected one CNN architecture from each of these approaches. In particular, we use Frustrum PointNet [115], PointPillars [82], and PointRCNN [131].

4.2.2 Domain shift in pointclouds

The domain shift problem has been explored for a long time in the context of vision-based tasks [37, 159, 166]. As we have mentioned in previous chapters, since it is a task-agnostic approach, image-to-image translation based on Generative Adversarial Networks (GANs) [53] emerged as a very promising paradigm to alleviate visual domain shift. In this context, most GANs use a generator based on an autoencoder trained with some loss which aims at preserving the relevant content of the input images after its transformation [18, 65, 89, 133, 144, 148, 186]. Since in previous chapters we saw that GANs were able to reduce the domain shift between synthetic and real-world images in our context of application, we have decide to follow the same approach to alleviate the domain shift between LiDAR-based pointclouds.

When trying to translate LiDAR-based pointclouds with GANs, we find similar memory drawbacks as when addressing 3D object detection. Obviously, the choice of the pointcloud representation may have an important impact on the pointcloud-to-pointcloud translation. The approaches based on voxel grids [19, 34, 169] require a proper adjustment of the grid resolution. Approaches based on the octree idea, *e.g.* Octnet [122], include multiples scales to increase the grid resolution using less memory but at the expense of increasing computation time. There are approaches based on 2D projections (*e.g.* bird-eye view, etc.) [73, 136] to reduce memory requirements but losing information. Yet other approaches work directly on 3D points [2, 43, 143] having the problem of scalability due to their computational complexity, thus, they are restricted to generate (translate) a fixed amount of 3D points. In fact, to avoid

this restriction further proposals translate one 3D point at a time [178].

Indeed, the translation results of these proposals are very promising, however, they are designed to perform on isolated 3D shapes and relatively small indoor 3D spaces. In contrast, in the context of AVs, we cannot assume LiDAR-based pointclouds containing isolated shapes (*e.g.* in addition to more or less density on traffic participants, the road surface is always present) and we must work on relative large outdoor scenarios. Wu *et al.* [168] start working on domain adaptation for outdoor scenes captured by LiDAR, but for task-specific (pointcloud segmentation) CNNs. In [97], a task-agnostic CNN-based pointcloud-to-pointcloud synth-to-real translation is proposed (using CARLA and KITTI), although not relying on the GAN idea. The underlying idea is that the domain shift is due to the inability to simulate the right behaviour of LiDAR rays in all circumstances. Thus, this is an specific issue arising in simulation (the effect can be even different for different simulators), but not necessarily in real-to-real domain shift. Even if we are specially interested in synth-to-real domain shift, we bet for GANs because, in practice, there is also a real-to-real domain shift. Moreover, after performing a number of exploratory experiments, we decided to rely on the voxel grid representation to perform GAN-based pointcloud-to-pointcloud translation.

4.3 Cross-domain LiDAR-based 3D object detection

As we have shown in previous chapters, synthetic images and unlabeled real-world ones can be used to train CNN-based visual models that must perform in the real-world. Although synthetic and real-world domains suffer from domain shift, it can be reduced by using different approaches as GAN-based image-to-image translation or semi-supervised learning. In this section, we want to assess if this domain gap also exists between pointclouds from different domains, being of especial interest the synth-to-real case. We focus on 3D object detection. To perform a proper analysis we have selected different publicly available datasets and detectors, which will be explained along next subsections. In fact, we will see that, indeed, the domain shift does exist.

4.3.1 Data

We use three real-world datasets acquired on-board and one synthetic dataset. In all cases, we have used three classes to evaluate 3D object detection performance: car, pedestrian and cyclist. The sensor specifications and datasets statistics are detailed in the following. We remark that, since one of the CNN models under evaluation involves a stage of 2D object detection on images paired with LiDAR samples, these

Table 4.1 – Dataset annotation statistics.

Dataset	Car	Pedestrian	Cyclist	images
KITTI	28,742	4,487	1,627	7,481
Waymo	156,408	66,825	2,257	19,395
Lyft	153,292	5,833	5,063	18,553
SYNTHIA-3D	84,328	64,826	103,501	37,369

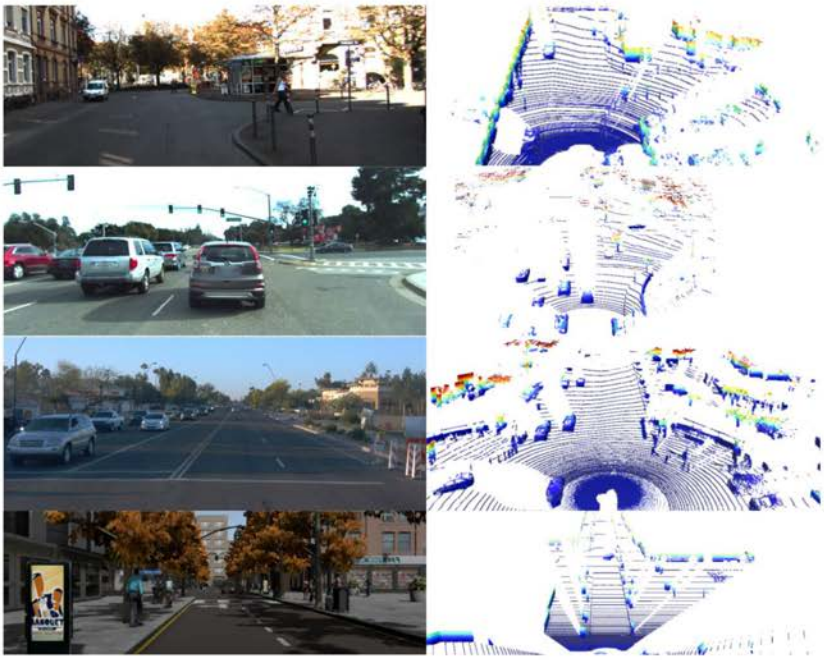

Figure 4.2 – Corresponding image (left) and LiDAR pointcloud (right) samples. From top to bottom: KITTI, Lyft, Waymo and SYNTHIA-3D.

dataset statistics refer to LiDAR samples having corresponding images.

KITTI. The KITTI dataset contains samples around Germany using different sensors. In this chapter, we use only the data from the LiDAR sensor on top of the vehicle. The sensor used is a velodyne of 64 beams with 26.9° vertical field of view (FOV) ([−24.9°,+2.0°]) and 360° horizontal FOV with 0.08° angular resolution (azimuth). The maximum data acquisition distance is 120 meters, although only the labels closer than 80 meters are considered. This dataset has 7,481 samples, split in 3,711 samples for training and 3,759 for validation.

Lyft. The Lyft dataset contains samples from Palo Alto, CA, United States. It consists of paired image and LiDAR samples, but in this chapter we use onlye the LiDAR ones. The data were acquired by three LiDAR sensors, one placed on top of the vehicle, and the other two on each bumper side, being the resulting pointclouds put in correspondence. The LiDAR sensor on the top scans the scenario with 64 beams, the others with 40. This is done for a 360° horizontal FOV with 0.2° azimuth resolution. The resulting dataset consists of 18,553 samples, 16,553 for training and 2,553 for validation.

Waymo. The waymo dataset consists of scenes from suburban and urban areas around different cities of the United States. The vehicle is equipped with five LiDAR sensors: one on top of the vehicle, three on the frontal bumper and one on the rear part of the vehicle. The sensor on top has a vertical FOV of 20°([−17.6°,+2.4°]) with a range of 75 meters; the other four LiDAR sensors are using a vertical FOV of 120°([−90°,+30°]) on a 20 meters range. The resulting dataset consists of 19,395 samples, 14,441 for training and 4,954 for validation.

SYNTHIA-3D. The SYNTHIA-3D dataset is sampled from the virtual world introduced in [123], thus, inspired on cities and urban areas. The 3D data are retrieved from depth images (z-buffer associated to the RGB images), using a sampling algorithm which partially mimics the LiDAR parameters used in the KITTI dataset, further explained in appendix A. To increase the realism of the simulated LiDAR we have removed the points from car's glasses as it is the case on real sensors. We have generated 33,600 samples for training on this synthetic data.

Overall, all the datasets are recorded on urban and non-urban areas but, as we can see in Table 4.1, the distribution of classes is different among them. For instance, cars receive the 93% of the annotations in Lyft, and cyclist just the 1% in Waymo. Furthermore, due to setting differences in the respective LiDAR suites, as can be seen in Fig. 4.2, the datasets have different pointcloud distributions.

4.3.2 Methods

In this chapter, we consider a representative CNN architecture from each of the three different approaches introduced in Sect. 3.2. In particular, among the methods

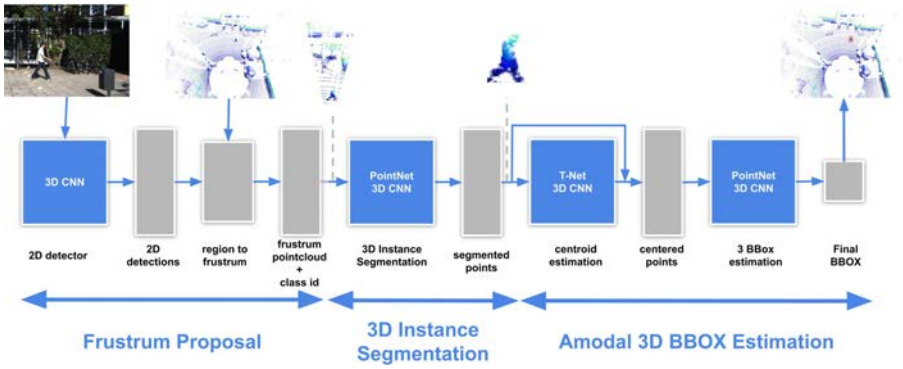

Figure 4.3 – Frustrum PointNet pipeline. The pipeline is divided in three stages. (1) Frustrum proposal is on charge of generating the 2D BBs, *i.e.* object candidates. (2) The 3D instance segmentation, which classifies the points within each candidate frustrum as object or background. (3) The amodal 3D BBOX estimation of the center of each object and the eight points defining its 3D BB. Blue boxes refer to trained model parameters.

requiring 2D images we use Frustrum PointNet [115], among the methods inspired on the voxel grid representation we use PointPillars [82], and from the methods directly working with the pointcloud representation we use PointRCNN [131]. We summarize Frustrum PointNet, PointPillars, and PointRCNN in the following.

Frustrum PointNet. Frustrum PointNet is based on a three-stage framework as can be seen in Fig. 4.3. The first stage uses a 2D object detector to predict 2D BBs, *i.e.* object candidates. However, in this chapter, since we are only interested in 3D BB prediction in pointclouds, instead of true 2D detections we use the corresponding ground truth; in this way, we remove errors due to 2D object detection from our analysis. Therefore, in our analysis, each object frustum is right, and the 3D object detection focuses on two stages. First, the points within each candidate frustrum are classified as object or background and, then, from the points classified as object, the center of each object and the eight points defining its 3D BB are estimated. For training will all datasets, we have used the same parameters as in [115], *i.e.* the *Frustrum_v1* parameter setting, and we have trained for the three selected classes (car/pedestrian/cyclist) using a single model.

PointPillars. As can be seen in Fig. 4.4, this CNN relies on two main stages. The first, *i.e.* the pillar feature network, takes the pointcloud aiming at producing a set of features in the form of a pseudo-image. The second, *i.e.* the 3D detection network, aims at detecting the objects from the pseudo-image. For the first stage,

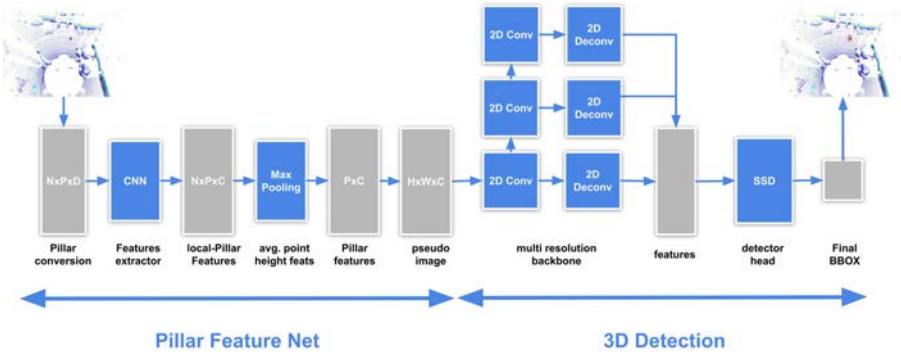

Figure 4.4 – PointPillars pipeline. It consists of two stages. (1) The Pillar Feature Net converts the pointcloud into pillars which are processed to obtain features. These are finally rearranged in the form of a pseudo-image. N is the number of points in each pillar, P is the number of pillars, D is the input data dimension for each point in each pillar, and C is the feature length of the learnt pillar. H and W set the dimensions of the pillar grid. (2) The 3D detection stage processes the pseudo-image using a multi-resolution network with a SSD head [92] for generating the detections. Blue boxes refer to trained model parameters.

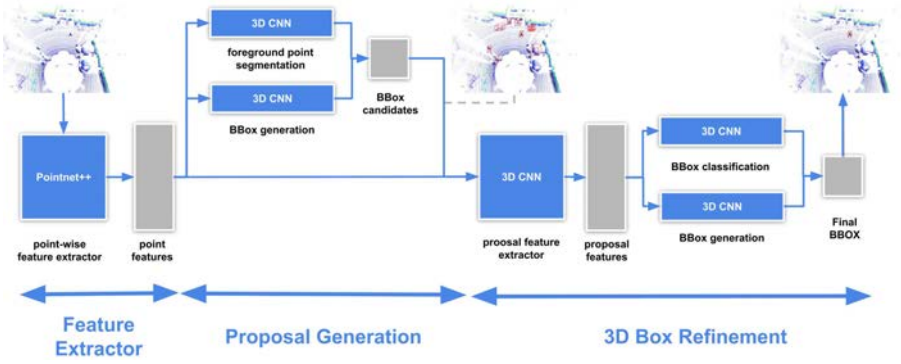

Figure 4.5 – PointRCNN pipeline. It uses a PointNet network style [26] to extract features, *i.e.* per-point features, further processed by a two-stage object detector, *i.e.* with a generation of candidates followed by a classification which determines the class of the candidates and refines the 3D BBs. Blue boxes refer to trained model parameters.

Table 4.2 – Difficulty levels in KITTI benchmark (KB) and our modification (OB) to accommodate Waymo and Lift datasets. K: KITTI, W: Waymo, L: Lyft.

Difficulty (KB/OB)	Height (K/W/L)	Occlusion	Truncation
(Easy/Easy)	> (40/120/50) pixels	fully visible	0.15%
(Moderate/Hard)	> (25/70/31) pixels	half visible	0.30%
(Hard/Hard)	> (25/70/31) pixels	hard visible	0.50%

a 2D grid perpendicular to the z-axis pointcloud is assumed. For each cell in the grid, a so-called *pillar* of N points is created. These N points are randomly chosen among those with the same (x, y) coordinate than the corresponding cell, zero padding is applied when there are less than N points available. This information is transformed into a N × P × D, where P is the number of pillars (*i.e.* the number of cells of the grid), and D accounts for information appended to the points of the pillars (*e.g.* mean point of the pillar, distance to this mean point, etc.). After a set of steps based on this representation, we obtain a pseudo-image of features. In the second stage, this pseudo-image is processed by using a multi-resolution network with a SSD head [92] which provides the final 3D object detections.

PointRCNN. As summarized in Fig. 4.5, the PointRCNN framework works directly on the pointcloud. It uses a PointNet network style [26] to extract features, *i.e.* per-point features, further processed by a two-stage object detector, *i.e.* with a generation of candidates followed by a classification which determines the class of the candidates and refines the 3D BBs.

4.3.3 Results

We provide results for the three selected 3D object detectors and each dataset. We use the metrics of the KITTI benchmark to report object detection [46] and orientation scores [40]. The metrics are reported for the three considered classes (car, pedestrian and cyclist) and for the two levels of difficulty (easy, hard) based on the three levels defined in the KITTI benchmark (easy, moderate and hard). In practice, since the pointclouds are registered with corresponding images, in order to assign these difficulty levels, it is is considered the projection of the 3D object BBs (ground truth) in the respective images. In particular, the levels are defined by the height and truncation of the projected object BBs, and the occlusion level assigned to the objects (which is a subjective ground truth label). Table 4.2 shows these values, which are the same for all datasets but the height parameter (which needs to accommodate the different resolution of the original camera sensors).

Table 4.3 – mAP scores for all training-testing domain combinations and all 3D object detectors. E and H refer to easy and hard difficulty levels, resp. FP, PP, and PRCNN refer to Frustrum PointNet, PointPillars, and PointRCCN, resp. Following KITTI benchmark, the mean IoU to accept a car detection is 70%, while for pedestrians and cyclists it is 50%. Bold highlights the cases where training and testing datasets are from the same domain.

			Car		Pedestrian		Cyclist	
Testing	Training	Method	E	H	E	H	E	H
KITTI	KITTI	FP	**74.09%**	**63.76%**	**61.21%**	**51.57%**	**83.80%**	**67.23%**
		PP	**86.47%**	**73.11%**	**62.83%**	**49.79%**	**79.68%**	**55.14%**
		PRCNN	**84.34%**	**74.00%**	**61.01%**	**46.19%**	**84.86%**	**57.51%**
	Lyft	FP	44.43%	31.89%	64.02%	51.08%	64.87%	54.33%
		PP	47.92%	33.15%	43.72%	35.22%	56.04%	40.97%
		PRCNN	30.47%	28.02%	31.43%	25.27%	57.17%	34.75%
	Waymo	FP	33.10%	19.17%	46.05%	38.27%	67.05%	57.11%
		PP	67.13%	47.99%	37.91%	29.51%	65.04%	46.59%
		PRCNN	05.53%	06.38%	56.59%	47.20%	60.05%	43.97%
	SYNTHIA-3D	FP	57.59%	37.10%	18.63%	17.30%	78.96%	63.51%
		PP	67.13%	47.99%	37.91%	29.51%	65.04%	46.59%
		PRCNN	28.21%	23.43%	08.31%	06.64%	62.32%	46.69%
Lyft	KITTI	FP	9.15%	9.37%	36.02%	30.27%	19.32%	14.44%
		PP	16.80%	13.11%	21.24%	15.30%	11.67%	8.54%
		PRCNN	36.77%	30.61%	19.73%	14.25%	22.44%	15.67%
	Lyft	FP	**66.83%**	**59.56%**	**68.57%**	**64.83%**	**51.82%**	**52.89%**
		PP	**54.85%**	**45.29%**	**29.54%**	**22.06%**	**31.05%**	**23.56%**
		PRCNN	**74.90%**	**65.98%**	**67.66%**	**58.30%**	**54.94%**	**47.02%**
	Waymo	FP	34.35%	28.62%	44.97%	44.21%	32.52%	25.75%
		PP	45.15%	39.12%	34.83%	25.13%	22.43%	16.69%
		PRCNN	63.30%	54.33%	37.12%	31.14%	36.01%	26.17%
	SYNTHIA-3D	FP	16.78%	12.93%	12.63%	10.61%	28.48%	19.94%
		PP	28.23%	21.60%	17.39%	12.49%	11.56%	7.08%
		PRCNN	55.25%	42.14%	6.05%	5.40%	34.23%	23.94%
Waymo	KITTI	FP	0.73%	0.96%	28.96%	22.88%	32.22%	25.74%
		PP	4.36%	5.20%	9.79%	5.31%	4.79%	1.33%
		PRCNN	7.24%	6.87%	15.97%	12.84%	31.02%	29.20%
	Lyft	FP	18.97%	18.55%	48.38%	45.81%	40.67%	42.28%
		PP	51.42%	37.40%	20.72%	16.28%	33.53%	19.48%
		PRCNN	42.44%	38.12%	21.41%	19.11%	35.61%	31.45%
	Waymo	FP	**37.28%**	**35.21%**	**72.16%**	**68.79%**	**48.44%**	**42.86%**
		PP	**66.44%**	**56.54%**	**43.08%**	**34.06%**	**26.91%**	**22.28%**
		PRCNN	**84.95%**	**77.22%**	**72.87%**	**61.38%**	**40.70%**	**36.78%**
	SYNTHIA-3D	FP	1.43%	1.38%	2.20%	1.54%	11.01%	16.97%
		PP	21.25%	16.01%	18.11%	9.06%	17.33%	2.78%
		PRCNN	45.05%	37.62%	01.02%	01.54%	32.69%	32.95%

Table 4.4 – Comparing the mAP scores of Table 4.3. Note that each cell in this table has a corresponding one in Table 4.3. Therefore, looking at Table 4.3, we compare the mAP score in such corresponding cell with its counterpart free of domain shift (*i.e.*, same training and testing domains). A negative sign indicates domain shift, and the value is the amount (in mAP units).

			Car		Pedestrian		Cyclist	
Testing	Training	Method	E	H	E	H	E	H
KITTI	Lyft	FP	-29.66	-31.87	2.81	-0.49	-18.93	-12.90
		PP	-38.55	-39.96	-19.11	-14.57	-23.64	-14.17
		PRCNN	-53.87	-45.98	-29.58	-20.92	-27.69	-22.76
	Waymo	FP	-40.99	-44.59	-15.16	-13.30	-16.75	-10.12
		PP	-19.34	-25.12	-24.92	-20.28	-14.64	-8.55
		PRCNN	-78.81	-67.62	-4.42	1.01	-24.81	-13.54
	SYNTHIA-3D	FP	-16.50	-26.66	-42.58	-34.27	-4.84	-3.72
		PP	-19.34	-25.12	-24.92	-20.28	-14.64	-8.55
		PRCNN	-56.13	-50.57	-52.70	-39.55	-22.54	-10.82
Lyft	KITTI	FP	-57.68	-50.19	-32.55	-34.56	-32.50	-38.45
		PP	-38.05	-32.18	-8.30	-6.76	-19.38	-15.02
		PRCNN	-38.13	-35.37	-47.93	-44.05	-32.50	-31.35
	Waymo	FP	-32.48	-30.94	-23.60	-20.62	-19.30	-27.14
		PP	-9.70	-6.17	5.29	3.07	-8.62	-6.87
		PRCNN	-11.60	-11.65	-30.54	-27.16	-18.93	-20.85
	SYNTHIA-3D	FP	-50.05	-46.63	-55.94	-54.22	-23.34	-32.95
		PP	-26.62	-23.69	-12.15	-9.57	-19.49	-16.48
		PRCNN	-19.65	-23.84	-61.61	-52.90	-20.71	-23.08
Waymo	KITTI	FP	-36.55	-34.25	-43.20	-45.91	-16.22	-17.12
		PP	-62.08	-51.34	-33.29	-28.75	-22.12	-20.95
		PRCNN	-77.71	-70.35	-56.90	-48.54	-9.68	-7.58
	Lyft	FP	-18.31	-16.66	-23.78	-22.98	-7.77	-0.58
		PP	-15.02	-19.14	-22.36	-17.78	6.62	-2.80
		PRCNN	-42.51	-39.10	-51.46	-42.27	-5.09	-5.33
	SYNTHIA-3D	FP	-35.85	-33.83	-69.96	-67.25	-37.43	-25.89
		PP	-45.19	-40.53	-24.97	-25.00	-9.58	-19.50
		PRCNN	-39.90	-39.60	-71.85	-59.84	-8.01	-3.83

While for KITTI the three occlusion levels are given, for Waymo and Lift it is only possible to distinguish from full or partial visibility, therefore, we have simplified the difficulty levels to two, easy (fully visible) and hard (otherwise).

Table 4.3 presents the results of training and testing with different datasets using the three 3D object detectors that we have selected. First we focus on the cases where the training and testing datasets correspond to the same domain, which have been highlighted in bold. Thus, we start by focusing on the differences between the 3D object detectors. We can see that for KITTI there is not a dominant detector which performs the best for all difficulty levels on the three classes. When looking at Waymo and Lift datasets, which are more challenging than KITTI, we observe that PointRCNN clearly outperforms PointPillars in all difficulty levels and classes. PointRCNN also outperforms Frustrum PointNet for cars, but not for pedestrians and cyclists. However, we remind that for our analysis, we assume perfect 2D detections (ground truth) to define the frustrum of each object.

Let's focus now in the case were training and testing datasets come from different domains. The detailed results can be seen in Table 4.3. However, in order to determine if there is domain shift, we introduce Table 4.4, where we compare the cross-domain results with their single-domain counterparts. Almost all values are negative in these comparative table, indicating a generalized problem of large domain shifts, both among real-world datasets and in the synth-to-real case.

Since manually annotating 3D BB is not trivial, we have assessed if such domain shifts could be due to human annotation instead of coming from actual pointcloud differences. In order to do so, we have devised the following protocol. Let $\mathscr{D}$ be the detector we are using (Frustrum PointNet/PointPillars/PointRCNN), $\mathscr{S}$ the source domain we are considering, and $\mathscr{T}$ the target domain. Here we assume different source and target domains. Then, $\mathscr{D}$ has been trained with the training set of $\mathscr{S}$, and tested in the validation set of $\mathscr{T}$. In fact, these are the results reported in Table 4.3. Now, we apply $\mathscr{D}$ to 100 pointcloud samples randomly selected from the training set of $\mathscr{T}$ (note, not from the validation set), and keep the right detections (true positives). What we do now is to explore how much we must modify the 3D BBs of these detections to have a larger mean IoU score computed on the corresponding 3D BBs of the GT. Since we look for differences among datasets due to the manual annotation process, we apply the same perturbations to all these detections. We do this by applying an independent per-axis (length/height/width) enlargement to all the detections, *i.e.* to all the corresponding 3D BBs. In particular, we consider enlargements running on $[-60,\dots,0,\dots 60]$ cm, with steps of 5 cm, which are added symmetrically to each axis (half left and half right, top/down, forward/backward). Then, the best enlargement for each axis is kept, where *best enlargement* refers to the one giving rise to the higher mean IoU score. These enlargements are reported in Table 4.5. Intuitively, these values can help to adjust the 3D BB margins between

Table 4.5 – Estimated 3D BB margin differences (in cm) among datasets due to the annotation process. L: length, W: width, H: height. As expected, we obtained 0 cm as result for training and testing sets being from the same domain.

			Car			Pedestrian			Cyclist		
Testing	Training	Method	L	W	H	L	W	H	L	W	H
KITTI	Lyft	FP	-25	-20	0	5	-5	5	-15	-10	-5
		PP	-30	-20	0	10	-20	10	0	-20	0
		PRCNN	-40	-20	0	10	-10	10	-30	-5	0
	Waymo	FP	-30	-25	-5	-15	-15	5	-10	-20	5
		PP	-30	-30	-10	-10	-10	10	0	-20	10
		PRCNN	-45	-35	0	-15	-20	0	-10	-20	5
	SYNTHIA-3D	FP	-20	-5	0	20	5	0	15	-15	-5
		PP	-20	-1	0	20	-10	0	-20	0	-10
		PRCNN	-25	-35	0	25	5	10	0	-15	0
Waymo	KITTI	FP	40	40	5	5	25	-5	-10	40	-15
		PP	40	40	10	10	30	-10	0	20	0
		PRCNN	35	40	0	10	35	-5	-20	30	-5
	Lyft	FP	40	35	5	10	20	-10	5	15	0
		PP	0	30	0	20	20	-10	0	10	0
		PRCNN	5	20	5	20	20	5	0	5	5
	SYNTHIA-3D	FP	40	35	5	30	30	-10	20	15	-10
		PP	-10	30	0	20	10	-10	-10	50	0
		PRCNN	10	-5	0	25	20	10	0	15	-20
Lyft	KITTI	FP	40	25	5	-5	20	0	15	35	-5
		PP	3	2	1	10	20	10	0	20	0
		PRCNN	35	25	0	15	20	10	-10	35	0
	Waymo	FP	35	0	5	-15	0	0	5	10	5
		PP	10	-10	0	0	-10	0	10	10	10
		PRCNN	20	-20	0	-15	-15	5	-10	-20	5
	SYNTHIA-3D	FP	40	15	0	25	15	5	5	25	0
		PP	-10	10	0	20	-10	0	10	20	0
		PRCNN	5	-15	0	35	25	10	10	15	5

Table 4.6 – mAP scores for all training-testing domain combinations and all 3D object detectors, applying the margin corrections of Table 4.5 for the cross-domain cases. E and H refer to easy and hard difficulty levels, resp. FP, PP, and PRCNN refer to Frustrum PointNet, PointPillars, and PointRCCN, resp. Following KITTI benchmark, the mean IoU to accept a car detection is 70%, while for pedestrians and cyclists it is 50%.

			Car		Pedestrian		Cyclist	
Testing	Training	Method	E	H	E	H	E	H
KITTI	KITTI	FP	**74.09%**	**63.76%**	**61.21%**	**51.57%**	**83.80%**	**67.23%**
		PP	**86.47%**	**73.11%**	**62.83%**	**49.79%**	**79.68%**	**55.14%**
		PRCNN	**84.34%**	**74.00%**	**61.01%**	**46.19%**	**84.86%**	**57.51%**
	Lyft	FP	49.77%	35.69%	52.59%	44.69%	74.10%	57.83%
		PP	72.19%	55.59%	43.72%	35.22%	67.01%	48.15%
		PRCNN	50.05%	45.74%	31.46%	25.14%	62.50%	38.37%
	Waymo	FP	58.97%	41.46%	31.07%	30.05%	79.60%	70.72%
		PP	76.32%	58.47%	53.31%	45.55%	66.18%	46.83%
		PRCNN	71.53%	63.64%	65.38%	53.34%	77.56%	55.86%
	SYNTHIA-3D	FP	53.90%	34.59%	52.58%	44.45%	68.08%	50.70%
		PP	75.34%	54.79%	44.10%	33.44%	69.42%	48.34%
		PRCNN	67.54%	51.55%	44.72%	32.60%	73.04%	53.57%
Lyft	KITTI	FP	35.74%	30.82%	39.35%	31.56%	30.64%	23.43%
		PP	41.79%	33.67%	22.02%	15.84%	16.80%	12.16%
		PRCNN	63.19%	52.53%	22.68%	15.83%	30.53%	22.17%
	Lyft	FP	**66.83%**	**59.56%**	**68.57%**	**64.83%**	**51.82%**	**52.89%**
		PP	**54.85%**	**45.29%**	**29.54%**	**22.06%**	**31.05%**	**23.56%**
		PRCNN	**74.90%**	**65.98%**	**67.66%**	**58.30%**	**54.94%**	**47.02%**
	Waymo	FP	14.86%	10.10%	45.81%	38.36%	16.76%	10.92%
		PP	49.36%	42.78%	35.18%	25.34%	22.39%	16.98%
		PRCNN	70.01%	58.93%	38.13%	32.15%	33.37%	23.66%
	SYNTHIA-3D	FP	19.59%	13.38%	29.82%	17.75%	21.84%	13.08%
		PP	25.73%	20.08%	19.14%	13.60%	15.54%	9.47%
		PRCNN	59.46%	48.05%	43.34%	34.77%	36.36%	25.18%
Waymo	KITTI	FP	27.37%	20.50%	37.65%	32.86%	38.91%	32.66%
		PP	42.28%	30.67%	35.49%	24.55%	6.04%	1.84%
		PRCNN	55.96%	48.15%	44.91%	35.27%	31.02%	34.57%
	Lyft	FP	48.21%	43.48%	56.62%	54.25%	38.01%	41.52%
		PP	49.29%	38.01%	26.62%	20.59%	33.53%	18.08%
		PRCNN	54.24%	46.71%	29.93%	25.38%	35.61%	32.10%
	Waymo	FP	**37.28%**	**35.21%**	**72.16%**	**68.79%**	**48.44%**	**42.86%**
		PP	**66.44%**	**56.54%**	**43.08%**	**34.06%**	**26.91%**	**22.28%**
		PRCNN	**84.95%**	**77.22%**	**72.87%**	**61.38%**	**40.70%**	**36.78%**
	SYNTHIA-3D	FP	07.06%	04.99%	16.40%	08.41%	03.81%	11.40%
		PP	28.35%	20.89%	23.02%	16.69%	17.33%	3.93%
		PRCNN	47.78%	38.50%	27.39%	21.21%	32.69%	32.84%

Table 4.7 – Comparing the mAP scores of Table 4.6. Note that each cell in this table has a corresponding one in Table 4.6. Therefore, looking at Table 4.6, we compare the mAP score in such corresponding cell with its counterpart free of domain shift (*i.e.*, same training and testing domains). A negative sign indicates domain shift, and the value is the amount (in mAP units).

			Car		Pedestrian		Cyclist	
Testing	Training	Method	E	H	E	H	E	H
KITTI	Lyft	FP	-24.32	-28.07	-8.62	-6.88	-9.70	-9.40
		PP	-14.28	-17.52	-19.11	-14.57	-12.67	-6.99
		PRCNN	-34.29	-28.26	-29.55	-21.05	-22.36	-19.14
	Waymo	FP	-15.12	-22.30	-30.14	-21.52	-4.20	3.49
		PP	-10.15	-14.64	-9.52	-4.24	-13.50	-8.31
		PRCNN	-12.81	-10.36	4.37	7.15	-7.30	-1.65
	SYNTHIA-3D	FP	-20.19	-29.17	-8.63	-7.12	-15.72	-16.53
		PP	-11.13	-18.32	-18.73	-16.35	-10.26	-6.80
		PRCNN	-16.80	-22.45	-16.29	-13.59	-11.82	-3.94
Lyft	KITTI	FP	-31.09	-28.74	-29.22	-33.27	-21.18	-29.46
		PP	-13.06	-11.62	-7.52	-6.22	-14.25	-11.40
		PRCNN	-11.71	-13.45	-44.98	-42.47	-24.41	-24.85
	Waymo	FP	-51.97	-49.46	-22.76	-26.47	-35.06	-41.97
		PP	-5.49	-2.51	5.64	3.28	-8.66	-6.58
		PRCNN	-4.89	-7.05	-29.53	-26.15	-21.57	-23.36
	SYNTHIA-3D	FP	-47.24	-46.18	-38.75	-47.08	-29.98	-39.81
		PP	-29.12	-25.21	-10.40	-8.46	-15.51	-14.09
		PRCNN	-15.44	-17.93	-24.32	-23.53	-18.58	-21.84
Waymo	KITTI	FP	-9.91	-14.71	-34.51	-35.93	-9.53	-10.20
		PP	-24.16	-25.87	-7.59	-9.51	-20.87	-20.44
		PRCNN	-28.99	-29.07	-27.96	-26.11	-9.68	-2.21
	Lyft	FP	10.93	8.27	-15.54	-14.54	-10.43	-1.34
		PP	-17.15	-18.53	-16.46	-13.47	6.62	-4.20
		PRCNN	-30.71	-30.51	-42.94	-36.00	-5.09	-4.68
	SYNTHIA-3D	FP	-30.22	-30.22	-55.76	-60.38	-44.63	-31.46
		PP	-38.09	-35.65	-20.06	-17.37	-9.58	-18.35
		PRCNN	-37.17	-38.72	-45.48	-40.17	-8.01	-3.94

Table 4.8 – Comparing the mAP scores of Table 4.6 with Table 4.3 (in mAP units). A negative sign indicates that margin corrections according to Table 4.5 did not help.

			Car		Pedestrian		Cyclist	
Testing	Training	Method	E	H	E	H	E	H
KITTI	Lyft	FP	5.34	3.80	-11.43	-6.39	9.23	3.50
		PP	24.27	22.44	0.00	0.00	10.97	7.18
		PRCNN	19.58	17.72	0.03	-0.13	5.33	3.62
	Waymo	FP	25.87	22.29	-14.98	-8.22	12.55	13.61
		PP	9.19	10.48	15.40	16.04	1.14	0.24
		PRCNN	66.00	57.26	8.79	6.14	17.51	11.89
	SYNTHIA-3D	FP	-3.69	-2.51	33.95	27.15	-10.88	-12.81
		PP	8.21	6.80	6.19	3.93	4.38	1.75
		PRCNN	39.33	28.12	36.41	25.96	10.72	6.88
Lyft	KITTI	FP	26.59	21.45	3.33	1.29	11.32	8.99
		PP	24.99	20.56	0.78	0.54	5.13	3.62
		PRCNN	26.42	21.92	2.95	1.58	8.09	6.50
	Waymo	FP	-19.49	-18.52	0.84	-5.85	-15.76	-14.83
		PP	4.21	3.66	0.35	0.21	-0.04	0.29
		PRCNN	6.71	4.60	1.01	1.01	-2.64	-2.51
	SYNTHIA-3D	FP	2.81	0.45	17.19	7.14	-6.64	-6.86
		PP	-2.50	-1.52	1.75	1.11	3.98	2.39
		PRCNN	4.21	5.91	37.29	29.37	2.13	1.24
Waymo	KITTI	FP	26.64	19.54	8.69	9.98	6.69	6.92
		PP	37.92	25.47	25.70	19.24	1.25	0.51
		PRCNN	48.72	41.28	28.94	22.43	0.00	5.37
	Lyft	FP	29.24	24.93	8.24	8.44	-2.66	-0.76
		PP	-2.13	0.61	5.90	4.31	0.00	-1.40
		PRCNN	11.80	8.59	8.52	6.27	0.00	0.65
	SYNTHIA-3D	FP	5.63	3.61	14.20	6.87	-7.20	-5.57
		PP	7.10	4.88	4.91	7.63	0.00	1.15
		PRCNN	2.73	0.88	26.37	19.67	0.00	-0.11

source and target datasets. This kind of differences may be due to the instructions followed by the human annotators. As a sanity check, we also considered the case $\mathscr{S} = \mathscr{T}$, and the found margins are zero as we should expect.

Now, we present new 3D object detection results in Table 4.6. The difference between these results and those in Table 4.3 is that, instead of directly evaluating the 3D BBs provided by each detector, we applied the corresponding margins of Table 4.5 to these 3D BBs. Then, these corrected BBs are submitted for evaluation against the 3D BBs of the GT. Again, to simplify the assessment of domain shift, we present Table 4.7 as summary for the results in Table 4.6.

Looking at Table 4.4 and Table 4.7 we can confirm that there is domain shift among the considered datasets, at least to affect 3D object detection. However, indeed, compensating for the different annotation criteria alleviate the accuracy drops significantly, which can be appreciated in a more straightforward way in Table 4.8. Therefore, we can take Table 4.6 as the final 3D object detection results, and Table 4.7 as final results regarding domain shift assessment. Among the 3D object detectors, Frustrum PointNet is suffering the most from domain shift. Comparing PointPillars and PointRCNN, we can see that depending on the source-target domain pairs and the particular class, they outperform each other, without a clear best. Overall, what matters for us is that these results indicate that domain shift must be addressed for LiDAR-based pointclouds. From these results, we also believe that the synth-to-real domain shift is not worse than the real-to-real cases.

4.4 GAN-based pointcloud-to-pointcloud translation

Encouraged by the results obtained in previous chapters, in order to reduce pointcloud domain shift, we have followed an approach based on GANs. Designing and training GANs is not trivial even for 2D data such as images. Therefore, it may be even more difficult for 3D pointclouds. Accordingly, in this chapter we have started by the basics. In particular, we want to assess if via GANs we can learn to perform certain basic pointcloud transformations, namely, rigid motion, shape change, removal and addition of noise. In order to properly assess the capabilities of GANs to learn such transformations, we use synthetic pointclouds since we can easily force all of them and we can generate the pointcloud as we wish.

4.4.1 Synthetic data

The synthetic dataset is generated by placing objects of predefined shape on a ground plane (Fig. 4.6), randomly varying certain object parameters such as size, point density (sampling), location, and surrounding noisy points. This kind of

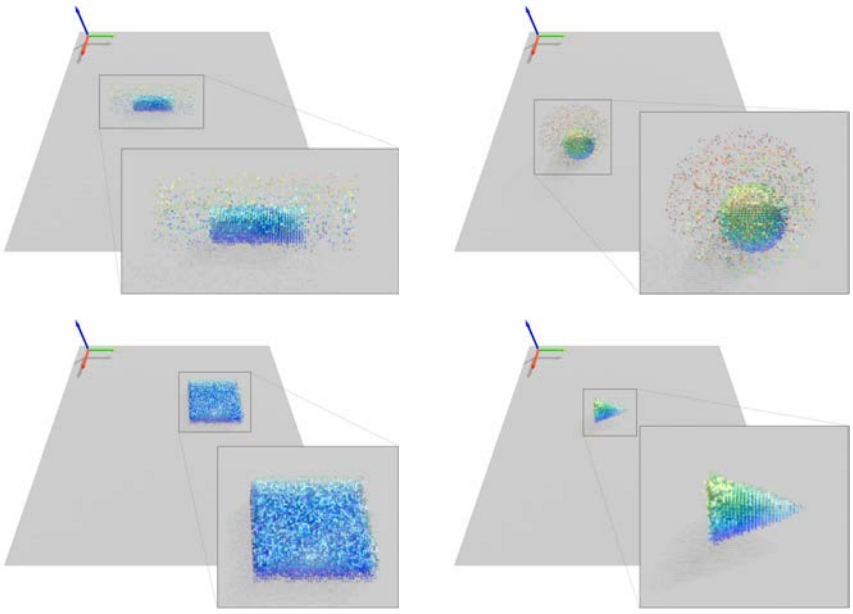

Figure 4.6 – Synthetic pointcloud samples over the considered ground plane. The height of the points is color-coded for the sake of a better visualization.

variations may produce domain shift, and arise because differences in the scanning strategies and physical placement of on-board LiDAR suites. The ground plane covers 70×70 meters as the usual forward-facing view of LiDAR-based pointclouds captured on-board (view of 70 meters forward and 35 meters at each side of the car). Objects can be framed by 3D BBs ranging from $6 \times 6 \times 1$ (width/length/height) to $40 \times 40 \times 3$ meters.

Shape changes. We consider four types of object according to their shape, namely, cuboids, spheres, cylinders and cones. Each object can be of different sizes and sampled according to a desired point density. Taking this into account, we have generated two pointcloud datasets, one based on cuboids and the other in the curved shapes (spheres, cylinders, cones). No matter the dataset, objects are generated according to a random size, point density, and location. In all cases constrained to the above mentioned ground plane limits and range of object sizes. Each dataset contains 10,000 samples. We consider that there is a domain shift between these datasets due to the different shapes that appear in each one.

Rigid motion. Each object can be placed in any location over the finite ground plane area, by defining the coordinates of its center of masses. According to this parameter, we generate two pointcloud datasets. One with objects on the ground plane, and another with objects *flying* at a fixed height. We consider that there is a domain shift between these datasets due to a rigid motion. Each dataset contains 10,000 samples, where objects have been randomly chosen among the predefined ones.

Noisy points. LiDAR pointclouds can have noisy points, for instance, when the LiDAR beams intersect difficult materials such as glasses. We simulate this noise by adding points around the generated objects. The amount of added points ranges from the 5% to the 10%, following a uniform probability. These points are also uniformly distributed around the objects. Again, we generate two different datasets, one with noise the other without noise. Each one containing 10,000 samples. Thus, in this case the domain shift comes from the presence of noise.

4.4.2 Proposed GAN

We assume a source domain A and a target domain B. We are interested in finding a function f able to transform samples from A to B. To find this function we make use of GANs [53], where f will play the role of the GAN's generator. As we have mentioned before, for training such a GAN, it is important to define a proper representation of the pointcloud.

In order to be able to work with pointclouds composed by any number of points, we use the voxel grid representation. Each voxel corresponds to $0.30 \times 0.30 \times 0.33$ meters of the pointcloud space, and the value of the voxel is -1 if there are no points falling in its corresponding space, and 1 otherwise. Then, for the synthetic pointcloud samples that we have generated, this implies that we have a grid of $200 \times 200 \times 30$ voxels. As is common practice, we process each plane individually, *i.e.* using 2D convolutions, at the input of the GAN's generator and discriminator.

The GAN's generator is composed by one common encoder (E), one decoder (X_D) for each domain D, and a translator (T) to transform the bottleneck features from the source domain to the target domain, as seen in Fig. 4.7. The encoder has three convolutional blocks, where each block is composed of 2D convolutions, batch normalization and leaky ReLU. The last block does not use batch normalization. The decoder is built as the encoder but changing convolutions for deconvolutions. To improve details on generated samples there are skip connections on the two first convolution layers. The translator network works from the bottleneck and each skip connection layer. Each translator consists of three ResNet blocks (Fig. 4.8).

Once we have defined the elements of our framework, we can define the losses used for training the GAN. The training procedure uses the adversarial losse defined

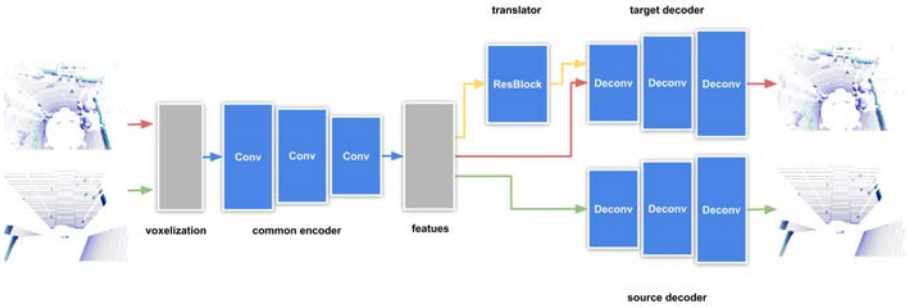

Figure 4.7 – GAN for pointcloud-to-pointcloud translation. Blue arrows show the common domain forward path, red arrows the target domain forward path, green arrows the source domain forward path, and yellow arrows the translation path from source domain samples.

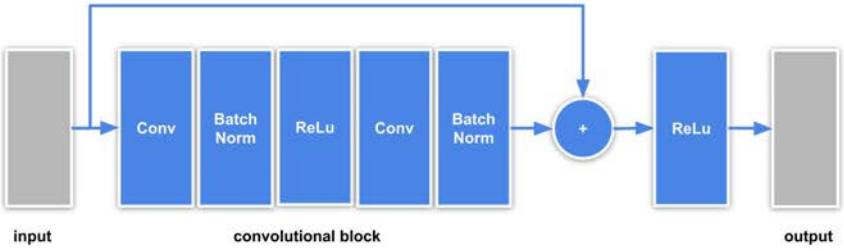

Figure 4.8 – Architecture based on ResNet blocks.

in [53]:

$$\mathscr{L}_{adv}(G_{A\to B}, D_B, \mathscr{I}^A) = \mathbb{E}_{i\sim\mathscr{I}^A}[log(D_B(G_{A\to B}(i)))] \ , \tag{4.1}$$

where $G_{A\to B}(i) = X_B(T(E(A)))$ for a generation from domain A to domain B. Among that loss, we have added regularization losses working on any domain D as:

$$\mathscr{L}_{reg}(E, X_D, \mathscr{I}^D) = \mathbb{E}_{i\sim\mathscr{I}^D}[\|X_D(E(i)) - i\|_1] \ , \tag{4.2}$$

note that there is no regularization on the translator network T, which is only trained using the previous adversarial loss. The regularizer term only works on the autoencoder (encoder+decoder) part using the same decoder domain. On the other hand, we have the losses for training the discriminator network defined as:

$$\begin{aligned}\mathscr{L}_{disc}(G_{A\to B}, D_B, \mathscr{I}^A, \mathscr{I}^B) = \mathbb{E}_{i^B\sim\mathscr{I}^B}[log(D_B(i^B))] \\ + \mathbb{E}_{i^A\sim\mathscr{I}^A}[log(1 - D_B(G_{A\to B}(i^A)))] \ .\end{aligned} \tag{4.3}$$

Using the previous defined losses, we can define our total loss as:

$$\begin{aligned}\mathscr{L}(G_{\mathscr{S}\to B}, E, X_A, X_B, D_B, \mathscr{I}^A, \mathscr{I}^B) = \mathscr{L}_{adv}(G_{A\to B}, D_B, \mathscr{I}^A) \\ + \mathscr{L}_{disc}(G_{\mathscr{S}\to B}, D_B, \mathscr{I}^A, \mathscr{I}^B) \\ + \mathscr{L}_{rec}(E, X_A, \mathscr{I}^A) + \mathscr{L}_{rec}(E, X_B, \mathscr{I}^B) \ ,\end{aligned} \tag{4.4}$$

where we want to transform samples from domain A to domain B.

4.4.3 Results

In order to evaluate the performance of this pointcloud-to-pointcloud translation GAN, we consider four situations: (1) shape change, from curved-shape objects (spheres, cylinders, cones) to cuboid objects; (2) rigid motion, from objects on the ground plane to flying objects; (3)-(4) noisy points, from situations without noise to situations with noise and vice versa. As we have described before, we have synthetic datasets with 10,000 pointclouds in each domain. For training the GANs, we always take the 50% of the source-domain dataset for training and the 50% for testing.

For each experiment we provide quantitative and qualitative results. In order to evaluate GAN-based image-to-image translation, it is common practice to use realism metrics (*e.g.* PIF, Inception Score, etc.) that could be adapted to work on pointclouds. However, it has been shown [83] that high values on these metrics do not imply that the transformed images are better than the original images to train for specific visual tasks. For these experiments, since we know how the source

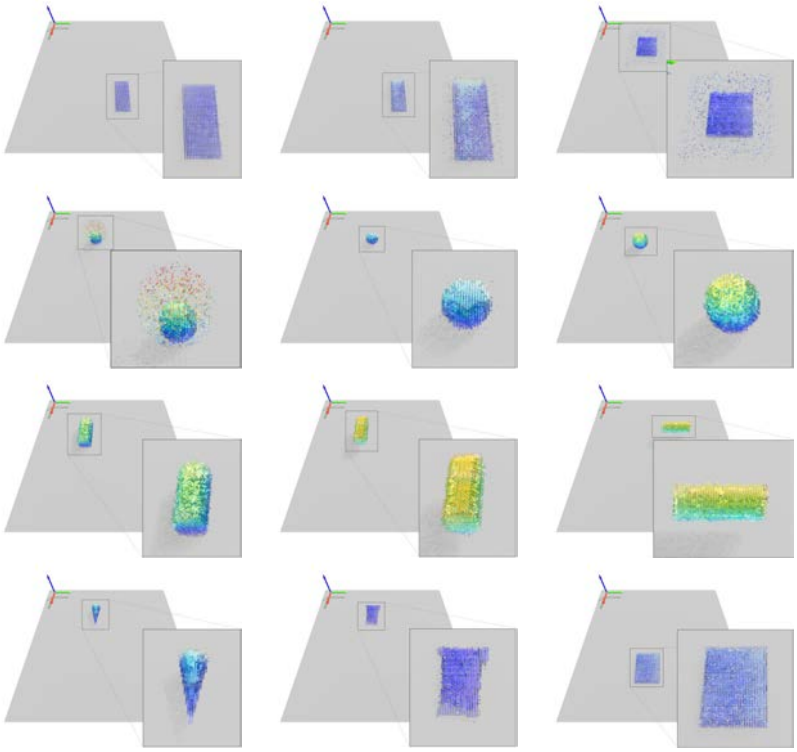

Figure 4.9 – Qualitative results. Column-wise: source pointcloud (left), translated pointcloud (center), and target-domain pointcloud (right). The later is added just for visual comparison with the translated pointcloud. From top row to bottom: adding noisy points, removing noise points, rigid motion towards a higher plane (note that height is color-coded), shape change (from a cone to a cuboid).

Table 4.9 – Results

Transformation	NI	$NCD_{\text{width}}, NCD_{\text{length}}, NCD_{\text{height}}$	KP
Shapes	86.07%	2.07%, 5.89%, 21.08%	65.43%
Rigid Motion	81.76%	4.32%, 2.78%, 9.25%	78.87%
Adding noisy points	99.84%	0.46%, 0.19%, 2.58%	81.94%
Removing noisy points	76.70%	1.79%, 1.54%, 12.33%	58.15%

and target domain pointclouds were generated, for each source-domain pointcloud under validation we can compute on-the-fly a corresponding target-domain surface as ground truth, *i.e.* to act as container of the translated pointcloud if the GAN performs perfectly. Therefore, we can measure the accuracy of the translation by using a normalized intersection, NI, defined as follows:

$$NI = 100 \times \frac{\text{Intersection}}{2\text{Prediction} - \text{Intersection}} \ , \tag{4.5}$$

where Prediction is the amount of points of the translated pointcloud, and Intersection the amount of points of the translated pointcloud falling inside the volume delimited by the corresponding ground truth surface. Note that $NI = 100$ for Prediction = Intersection, *i.e.* for an ideal translation; while $NI = 0$ when there is no intersection. An additional metric we use to evaluate how close is the translated pointcloud to the target-domain ground truth, is the difference between the corresponding centroids. In particular, we provide a per-axis normalize centroid difference, NCD_d, $d \in \{\text{width, length, height}\}$, where the per-axis normalization is with respect to the size of the corresponding side of the source-domain pointcloud 3D BB. On the other hand, it is not only important to measure where the translated points are located in the target-domain space, but also measuring if there is a loss of points, which it is not desired since in these experiments we use the same point density for source and target domains; thus, we defined a kept points (KP) metric as:

$$KP = 100 \times \frac{\min(\text{Prediction}, \text{Source})}{\max(\text{Prediction}, \text{Source})} \ , \tag{4.6}$$

where Source is the number of points in the source pointcloud.

During training, we use the same hyper-parameters for all experiments. We train the GANs for 100,000 iterations with batch size of 1, and using the Adam optimizer with a 0.0002 learning rate. We use the GAN model obtained at the last iteration. In Fig. 4.9 we can see examples of original and translated pointclouds, we can see how the trained GANs are able to add the desired effect to the source-domain pointclouds so that they resemble more target-domain counterparts, *i.e.* adding and removing noisy points, performing a rigid motion, and changing shape (*e.g.* from a cone to a cuboid). In Table 4.9 we present the quantitative results. Overall, NI values are high, and NCD_d values are low. This means that globally the GAN is able to perform the desire translation. Looking at the KP metric, we see that point density is kept well for rigid motion and adding noisy points. However, it is lower than we would like when removing noisy points and changing shapes. In the former case, it means that object points are also removed, not only noisy ones. In the latter

case, it is a bit more expected since we are changing shapes.

4.5 Conclusions

With the recent interest in LiDAR-based pointclouds to train deep perception models for AVs, we were motivated to perform such training by leveraging from synthetic pointclouds with automatically generated ground truth. Since LiDAR is a kind of active sensor which factors out appearance and provides geometry in the form of pointclouds, in contrast to previous chapters, we did not expect a serious issue regarding potential domain shifts. However, our comprehensive experiments on on-board 3D object detection, show that there is domain shift between publicly available LiDAR-based pointclouds, so in real-to-real settings, and also in the synth-to-real setting. In order to reach these conclusions we have relied on three real-wold datasets and a synthetic one. We have generated our own dataset of synthetic pointclouds, which we plan to put publicly available too. We have also used three essentially different 3D object detectors. Intuitively, the shift among the different domains, comes from different LIDAR acquisition settings in terms of scanning resolution and placement of the sensor suite on-board the ego-vehicle. Even more, our experiments show that if we not only have 3D BBs from the source domain but also from the target domain, then domain differences on human annotation must be compensated to avoid artificial domain shifts not really due to the pointclouds themselves.

Since in previous chapters GANs were helping to perform domain adaptation for vision-based perception tasks, we have decided to use GANs for pointcloud-to-pointcloud translation between a source and a target domain. The first decision was to chose a proper pointcloud representation. We have used a voxel grid representation, which allows us to process pointclouds of an arbitrary number of points, in contrast to previous related works. Then, we have designed and trained a GAN working with this representation as input. In order to evaluate such a GAN, we have designed a set of experiments based on synthetic pointclouds. They have shown that the GAN is able to translate between two domains which were forced to have shifts in terms of shape, rigid motion, and presence/absence of noisy points. Overall, we have shown that domain adaptation is also needed for LiDAR-based pointclouds. We have shown that GAN-based pointcloud-to-pointcloud translation is a promising way to reduce domain shift. Next steps will focus on extensive experiments with real-world LiDAR data. We have to remark that, according to our experience, it is important to design and understand the pointcloud-to-pointcloud GANs using fundamental examples as we have done in this chapter, since developing such GANS directly on real-world uncontrolled LiDAR data is extremely expensive in compu-

tational terms. In fact, although for the sake of simplicity we have not mentioned a large part of the work done related to this topic, we must say that until we did not follow such approach we were not able to decide which is the best pointcloud representation to work with GANs.

5 Conclusions and Future work

Autonomous vehicles (AVs) are considered to be one of the top technologies that will change our life for good. One can see already some use cases of AVs safely driving in certain areas under the supervision of a human expert. AVs make use of a on-board suite of sensors to understand the scene around them. These sensors can capture appearance (cameras), distances (LiDAR), speed (radar), ego-vehicle motion (IMU), etc. The vast amount of data provided by these sensors are used by powerful artificial intelligence (AI) to learn how to drive. The need for these data keeps increasing with new AIs based on deep convolutional neural networks (CNNs), which are highly data-hungry. The cost of obtaining the proper data to develop such CNNs does not reside only on their acquisition from the raw sensors, but there is also a cumbersome human labor on providing supervision (ground truth) by annotating object locations, semantic categories, pedestrian/cyclist/driver intentions, etc. Therefore, approaches to reduce the data annotation burden are essential for scaling the development of deep perception models for AVs. In the last decade, we have seen how leveraging from synthetic data with automatic annotations can reduce the amount of required human annotated data provided the synth-to-real domain shift is well addressed. In fact, domain shift also appears in real-to-real domain cases. Overall, this relevant context motivated the research presented in this thesis.

The first topic we have addressed is motivated by the observation that AVs need to be able to adapt to a constantly changing world, or new environments. In particular, when addressing traffic sign recognition (TSR), a kind of multi-class classification problem, it may happen that the number of sign classes to be recognized must be suddenly increased without having annotated samples to perform the corresponding TSR CNN re-training. We address this problem in Chapter 2. The idea is to leverage synthetic samples of the new traffic sign classes, however, this induces a domain shift problem. Since generative adversarial networks (GANs) can perform image-to-image translation, we decided to rely on such technology to reduce the domain shift. The advantage is that such a GAN is task-agnostic,so not depending on the specific CNN used to perform the TSR itself. However, the

challenge comes from the fact that TSR is sensitive to image details, but such a GAN must be trained on known classes while latter it must perform a synth-to-real translation on samples of unknown classes. The main tasks we performed in this research are the following:

1. The generation of a synthetic traffic sign dataset with compatibility with the Tsinghua dataset, a main reference to address traffic sign detection and recognition. We call this dataset SYNTHIA-TS and we plan to put it publicly available.

2. The design of an exhaustive set of experiments covering different ways of balancing known and unknown traffic sign classes, varying the relative amounts and considering both random balances and semantic-based ones (using a hierarchy of traffic signs classes that we defined).

3. The training and validation of different CNN architectures for TSR with known and unknown classes by using synthesized samples for unknown classes, both as they come from a virtual environment and as translated by a CycleGAN trained on samples from the known classes.

The obtained results show that some TSR CNNs are more robust than others to synth-to-real domain shift after applying the GAN, in particular, ResNet architecture outperforms VGG in this setting. A ResNet101-based TSR classifier and image translation based on CycleGAN performed extremely well for a $\sim 1/4$ ratio of unknown/known classes; even for more challenging ratios such as $\sim 4/1$, the results are very positive.

Motivated by these results, in Chapter 3 we addressed a more challenging task, namely, on-board vision-based 2D object detection. We explore if by leveraging from synthetic data, we can significantly reduce the human annotation effort of the 2D bounding boxes (BBs) required to train 2D object detectors. We consider the detection of vehicles and pedestrians. Thus, the idea is to automatically provide 2D BBs for such classes given an unannotated set of images. We propose a semi-supervised learning (SSL) pipeline based on the co-training idea. In short, co-training is an iterative approach where new data are progressively self-annotated by a coordinated action of two different models (here two deep 2D object detectors), which are eventually improved by retraining with the new self-annotations, which in turn should allow to self-annotate more data (false negatives in current iteration) in the next iteration. In our pipeline, the initial models, *i.e.* those in the first iteration, are trained on synthetic data. Therefore, since we aim at self-annotating a real-world set of images, there is an inherent domain shift problem. Accordingly, we have also used CycleGAN to reduce this shift and so starting with more accurate initial

models. In fact, this kind of synthesized data are used in all iterations as regularizing mechanism during the training of the deep detectors; *i.e.*, at the mini-batch level, synthesized (original or translated by the GAN) and real-world images are combined. Overall, the designed pipeline has the goal of self-annotating a set of real-world images, which can be latter used to train any 2D object detector. Therefore, the pipeline is agnostic to the CNN architecture used for 2D object self-annotation during co-training; in other words, such architecture is taken as a black box. The main tasks we performed in this research are the following:

1. The generation of a synthetic dataset of on-board images with 2D BBs for vehicles and pedestrians, which we plan to put publicly available. We use the publicly available KITTI and Waymo datasets as real-world ones.

2. The design of an exhaustive set of experiments covering different settings regarding the annotated and unannotated data as input to the SSL pipeline. We cover the case where we start with a small amount of annotated real-world images and many unannotated ones (no domain shift), the case where no real-world images are annotated but we have the mentioned synthetic dataset, and the analogous case but where the original synthetic data are previously translated by a CycleGAN to approach the appearance of the real-world (target) domain images. The experimental setting also assesses the relevance of false positives and BB adjustment when training a deep 2D object detector with the obtained self-annotations.

3. The design and implementation of a co-training pipeline for deep 2D object detection, as well as a self-training pipeline acting as SSL baseline. Moreover, we trained the needed CycleGANs to alleviate domain shift.

The obtained results confirm that our proposed co-training pipeline allows to reach almost upper-bound results (*i.e.* when the training data are fully human-annotated) provided we use the synthetic data translated by a CycleGAN. In other words, almost upper-bound results are obtained without human annotation. Interestingly, co-training and GAN-based translation provide complementary contributions on the obtained accuracy. According to our experiments, the remaining issue to fully reach the upper-bound performance is to obtain a better 2D BB adjustment, which may be due to difficult cases such as objects with heavy occlusion. This seems to be more important than the small amount of false positives output by our co-training pipeline.

After addressing TSR, a multi-class image classification problem, and vision-based 2D object detection, in both cases focusing on methods to leverage synthetic data for training the respective deep models, a natural step forward is to address

LiDAR-based 3D object detection, in this case leveraging synthetic pointclouds for training. We elaborate this work in Chapter 4. The main tasks we performed in this research are the following:

1. The generation of a synthetic dataset consisting of on-board pointclouds with 3D BBs for vehicles, pedestrians, and cyclists, which we plan to release publicly. We use the publicly available KITTI, Waymo, and Lift datasets as real-world ones.

2. A comprehensive set of experiments with cross-domain training and testing runs, considering three different 3D object detectors.

3. The design and evaluation of a pointcloud-to-pointcloud translation GAN, for which we use a voxel grid representation of pointclouds as input.

In contrast to previous chapters, we did not expect a serious issue regarding potential domain shifts. However, the mentioned set of experiments show that there is domain shift between publicly available LiDAR-based pointclouds, so in real-to-real settings, and also in the synth-to-real setting. Intuitively, the shift among the different domains comes from different LiDAR acquisition settings in terms of scanning resolution and placement of the sensor suite on-board the ego-vehicle. Even more, our experiments show that, if we not only have 3D BBs from the source domain, but also from the target domain, then domain differences on human annotation must be compensated to avoid artificial domain shifts not really due to the pointclouds themselves. Accordingly, since in previous chapters GANs helped to perform domain adaptation, we decided to design a new GAN for pointcloud-to-pointcloud translation between a source and a target domain. We determined that, in order to do so, using a voxel grid representation as input to the GAN, allows us to process pointclouds of an arbitrary number of points, in contrast to what we found in the state of the art. Extensive experiments with synthesized pointclouds have shown that our GAN is able to translate between two pointcloud domains which were forced to have shifts in terms of contained shapes, rigid motion, and presence/absence of noisy points.

Overall, in this thesis, we have shown how synthetic data can help to alleviate the human annotation of on-board raw sensor data. As a result of this research, several ideas for improvement arise which we think are worth to pursue as future work. We think that our SSL proposal, based on GANs and co-training, should be tested on a multi-modal setting. At the moment, the models cooperating during co-training are applied to images. However, for the success of co-training it is critical to keep these models essentially different (so avoiding drifting). We think that these differences could be higher by using a multi-modal approach, *e.g.* one model

performing on RGB images and another on corresponding depth images (from monocular depth estimation or from a registered LiDAR). In addition, it is worth to assess this approach on other perception tasks such as semantic or instance segmentation. We would like also to assess the performance of our pointcloud-to-pointcloud translation GAN beyond synthetic experiments, which certainly will require a proper computational setup since voxel grid resolution will need to be increased. Finally, in line with our work on TSR, we woul like to apply the idea to indoor environments where humanoid robots must recognize different types of domestic objects, and some of them can be totally unknown at a given moment.

A Appendix

A.1 The SYNTHIA Dataset Reloaded

SYNTHIA is nowadays a widely used public dataset for studying Autonomous Driving in the context of vision-based perception tasks. SYNTHIA's images were originally released with pixel-wise ground truth for semantic and instance segmentation, as well as depth maps. Here, we present an upgrade of SYNTHIA (SYNTHIA:Reloaded), with more variability thanks to new scenarios and assets; new data modalities such as LIDAR; ground truth for new problems, such as 3D and 2D object detection of pedestrians, cyclists, vehicles and traffic signs/lights; This new version has been used to generate the data in chapters 2, 3 and 4.

By varying different parameters of the Unity-based SYNTHIA framework, the new datasets include random shots and sequences. In the next sections we outline both the main parameters involved in the configuration of the SYNTHIA's virtual world, as well as the raw data and GT that it can provide. The specifications used to generate each dataset is specified in their respective chapters.

A.1.1 Virtual world configuration

Fig. A.1 shows the different areas of the current SYNTHIA world (town and NY[1] style areas, and highway). Table A.1 summarizes the main parameters we can set for collecting data.

The *data acquisition mode* indicates that we can either capture data as random shots or as driving sequences. In the former case, we can acquire data anywhere in the virtual world, using a random sensor pose (avoiding useless shots such as looking at the sky, or from inside a building). We can even randomize the pose and location of the dynamic objects (pedestrians, vehicles, etc.) and traffic signs/lights to obtain many instances of them without actually driving. In the latter case, the *player* car as well as the rest of traffic participants are moving while respecting traffic rules. In both cases, data acquisition is determined by the sensor suite that

[1] New York city

Table A.1 – Main configuration parameters in SYNTHIA.

Data acquisition mode		Driv. Seq. / Random shots
Sensors	Camera	focal length, image resolution, fps, 6D pose
	LIDAR	vertical & horizontal FoV, vertical & horizontal resolution, rpm, max. depth, 6D pose
Scene	Textures	selection of road, sidewalk, and vegetation textures
	Ambient	brightness, shadow strength, IDs of time, weather, & season
	Area	Town/NY/Highway
Objects	Parked	number of: more than 2-wheel vehicles, bikes, motorbikes
	Moving	number of: vehicles, cyclists, bikers, pedestrians, wheelchairs

we define. In the case of driving sequences, sensors are attached to the player car as in the example of Fig. A.2.

At the moment we can generate data by approximating two types of sensors for perception, camera for textures and image processing, and LIDAR for depth estimation. As can be seen in Table A.1, we can place the cameras as wished and define their main parameters (focal length and resolution). Camera acquisition is directly leveraged from Unity®Pro framework, which allows us to set a desired fps value. A real LIDAR throws a vertical ray perpendicular to the XZ plane, rotating the ray at different times on this plane until completing 360° with a vertical FoV. The space between points follow a vertical and horizontal resolution. Due to the ray rotation among the time, if there are moving objects, motion blur appears. To simulate this sensor we rely on a rotating camera with the same vertical FoV, although the depth buffer of the camera is considered instead of the RGB image. The acquired depth buffers are sampled according to the established horizontal and vertical resolution. To simulate a real LIDAR we would need as many camera captions as rays needed to complete 360° according to the horizontal resolution, which it is computationally impractical. So, we have added a horizontal FoV parameter to define how many degrees we capture at the same time with the camera. As lower the value more realistic LIDAR we can obtain.

There are several synthetic data nowadays where they try to complement the real data by applying a high degree of variability and keeping the world as fidel as

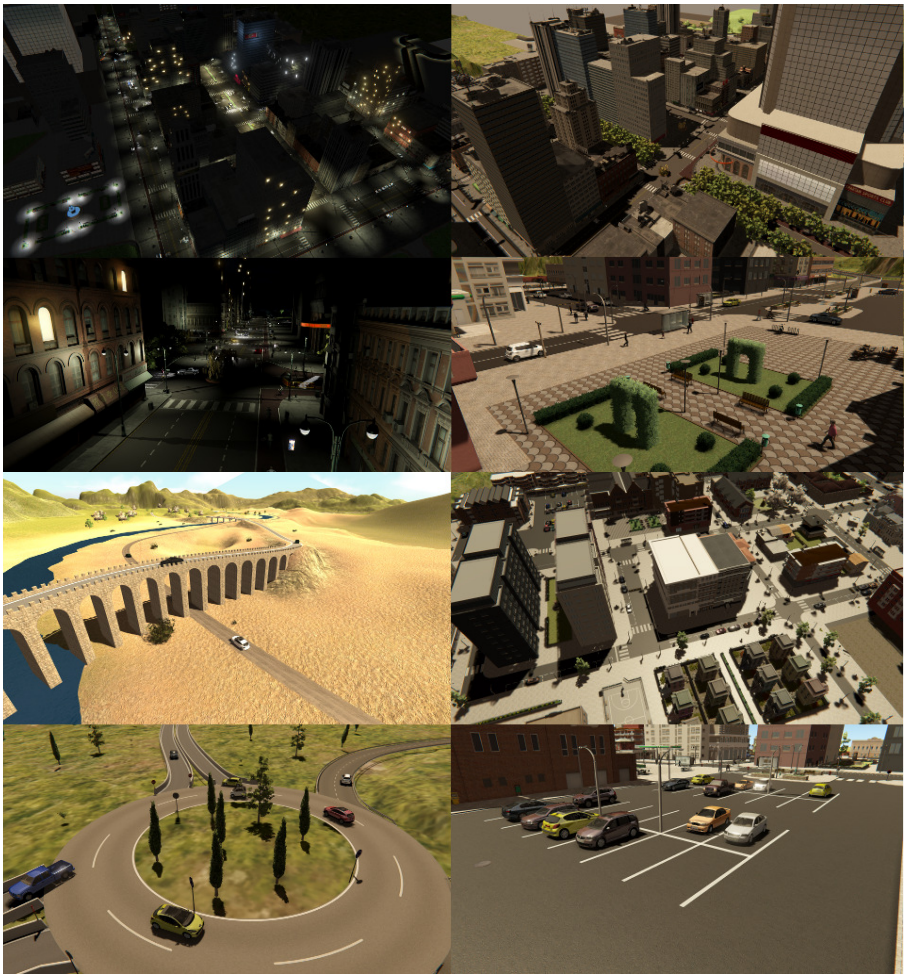

Figure A.1 – SYNTHIA views. Left column, top to bottom: NY and Town styles during night, bridge area and highway area. Right column, top to bottom: aerial view from NY style area, park area, aerial view from tonw style area and parking example. Regarding areas, the town covers approx. $370 \times 260m$, NY $480 \times 370m$ and the highway $1630 \times 1050m$.

possible to the real world and sensors. On the next points we talk about the fidelity and variability included in the SYNTHIA dataset.

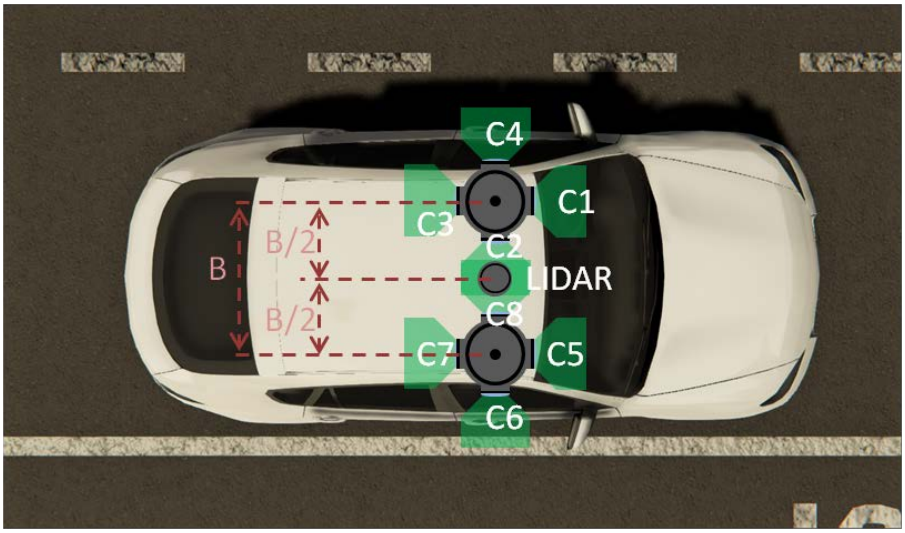

Figure A.2 – Example of sensor suite; C# stands for *camera #*. When recording with a mono camera B is always 0.

Fidelity. The virtual world is captured by camera and LIDAR sensors. The camera sensor follow a perfect pinhole camera where does not exist noise, blurring or other effects created by current cameras. This can be a problem when we train to learn patterns from synthetic data and applied on real data, where all type of noise can be observed in the images. To decrease the effect of this situation we have included some post processing features (see Table A.2) to improve the realism from the synthetic data. The listed features focus mainly on improving the light transfer between different materials textures (*i.e.* metallic cars, rough buildings, etc) and color relationship between different objects. We have included different blurring and noise effects to simulate the effects of real camera sensors based on motion and perception. Although it is an important forward step among realism we still are missing features based on physics models created by camera sensors as rolling shutter. On our future work we plan to work on physic based models among exploring this type of features for LIDAR sensors.

Variability. For generating the virtual-world content we can also select different textures for roads, sidewalks and vegetation; as well as global scene brightness, strength of the shadows, daytime/nighttime, weather conditions (*e.g.* clear/rainy), and season. We can also select the number of traffic participants to be spawned, either to stay parked (vehicles/bikes/motorbikes) or to move (vehicles, pedestrians,

Table A.2 – Feature effects included into the SYNTHIA:Reloaded model to improve realism.

Feature	Definition
Antialiasing	Definition
Ambient occlusion	Shadows created by close objects
Depth of Field	Blurring effect at far objects
Motion Blur	Blurring effects caused by camera or moving objects
Eye Adaptation	Colors perceived are modified based on the minimum and maximum presented on the scene
Bloom	Propagate illumination to places closes to high illumination peaks
Chromatic Aberration	Small noise created at borders objects by camera sensors

bike riders and cyclists). Fig. A.3 shows several examples. Table A.3 compares variability factors (dynamic objects – multi-view) for different publicly available datasets. Thorough the experiments we show how this variability is able to improve the performance of SYNTHIA over other datasets as GTA-V which has more realism, but they contain less variability and flexibility to generate diverse data.

A.1.2 Groundtruth

Table A.3 shows the GT types required for covering research activity on most key computer vision tasks, indicating also which publicly available datasets include them. Fig. A.4 shows visual examples of such GT for the case of the new SYNTHIA dataset.

In addition to the GT already present in previous SYNTHIA dataset, *i.e.* pixel-wise depth, semantic class, instance ID, and camera pose; the new SYNTHIA dataset also includes 3D oriented BBs for camera and LIDAR data (not only for dynamic objects as the rest of datasets, but also for traffic signs/lights), pixel-wise optical flow, LIDAR-like point clouds with semantic class and instance ID per point. Note also that in contrast with other datasets SYNTHIA has pixel-wise GT for lane-makings (can be specially interesting along the highway) and horizontal signs. It is worth to mention that, up to the best of our knowledge, such a semantic and instance LIDAR information is not provided by other publicly available datasets.

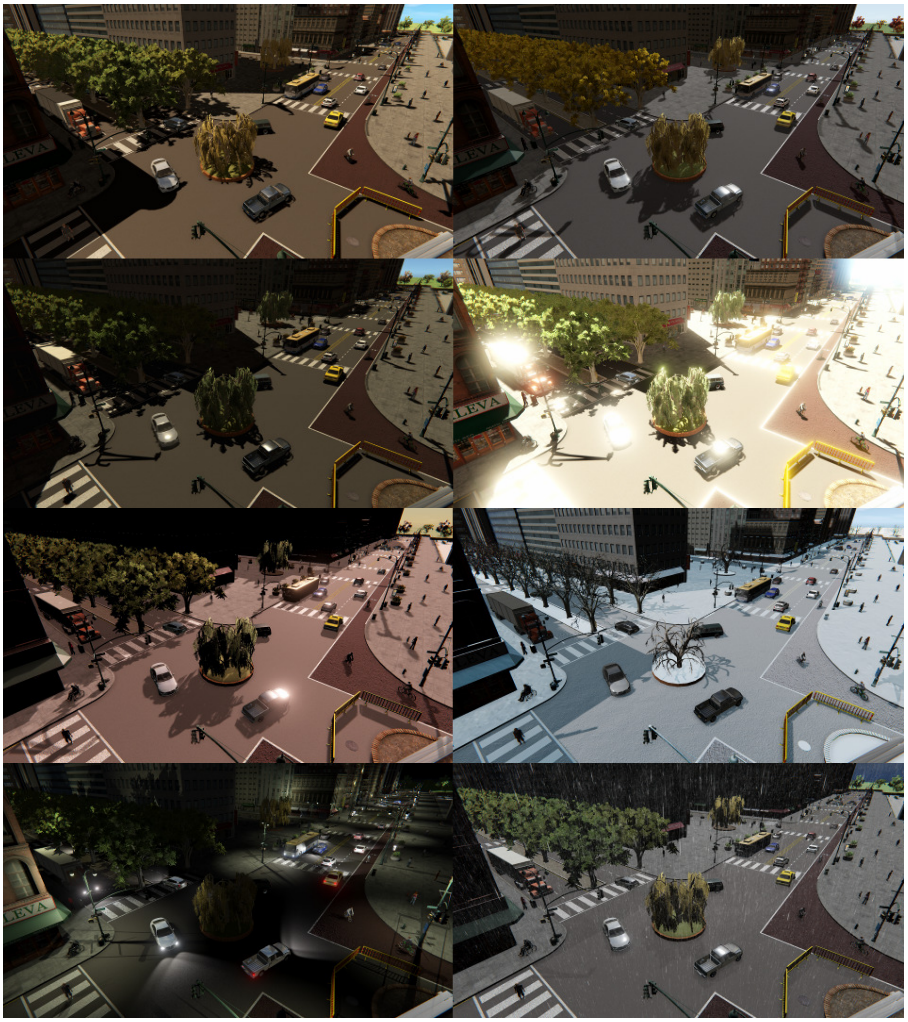

Figure A.3 – Aerial view of the same intersection under different ambient conditions. Top, from left to right: summer, fall, sunset and winter. Bottom, left to right: heavy shadows, bright illumination, night and rain.

Table A.3 – GT and variability factors provided in different publicly available datasets of synthetic images. BBs stands for 3D bounding boxes with instance ID (in the new SYNTHIA they are provided not only for images, but also for LIDAR data), CK means CamVid/KITTI compatible semantic classes, while CS refers to Cityscapes. Sem LIDAR refers to the point cloud with associated class label and instance label per point. V stands for Vehicles, and VRUs for vulnerable road users (pedestrians and cyclists). In the case of SYNTHIA, Multi-view includes stereo rigs as well as a 360° view based on several cameras.

	VDrift [59]	SceneFlow [101]	Virtual KITTI [44]	GTA-V [121]	SYNTHIA [123]	SYNTHIA:Reloaded
BBs	✗	✗	✓	✓	✗	✓
Depth	✓	✓	✓	✗	✓	✓
Sem. classes	None	None	CS	CK&CS	CK	CK&CS
Instance segm.	✓	✓	✓	✓	✓	✓
Camera pose	✓	✓	✓	✓	✓	✓
Sem. LIDAR	✗	✗	✗	✗	✗	✓
Dyn. objects	V	V	V	V&VRUs	V&VRUs	V& VRUs
Weather	Sunny	Sunny	Variable	Variable	Variable	Variable
Variable Illum.	✗	✗	✗	✗	✓	✓
Seasons	Fixed	Fixed	Fixed	Fixed	4	4
Multi-view	✗	✗	✗	✗	✓	✓

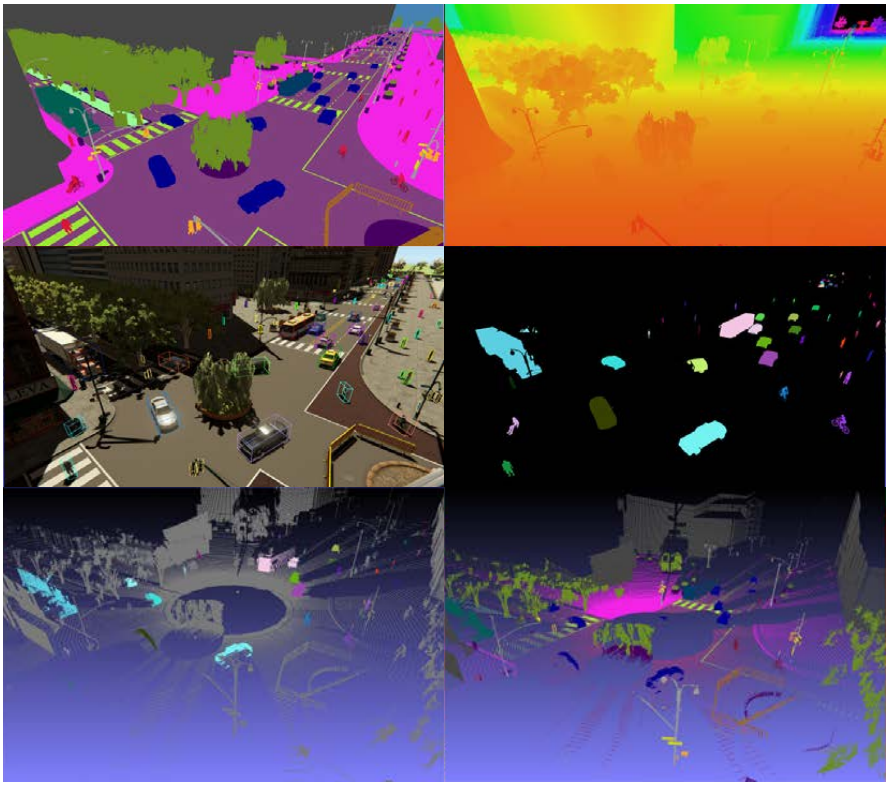

Figure A.4 – Aerial view grountruth of the same scene. Left, from top to bottom: Semantic segmentation, 3D Bounding boxes and instance LIDAR. Right, from top to bottom: Depth, instance segmentation and semantic LIDAR.

Obviously, as in previous version we have vehicles, pedestrians and cyclists, and we have added wheelchairs. Overall, we have also improved the visual quality. Moreover, when we use a suite of sensors, like in Fig. A.2, camera and LIDAR raw data can be spatially registered by the provided camera and LIDAR meta-information, and timestamps are used for temporal alignment. Odometry information also comes with timestamps. Thus, we follow a data acquisition protocol which is common to autonomous driving research.

Since in chapter 3 we also use 2D object detection, it is worth to explain how 2D BBs are obtained from the GT. There are two straight forward approaches. On the one hand, one can use the pixel-wise instance IDs to build the BBs; however, this

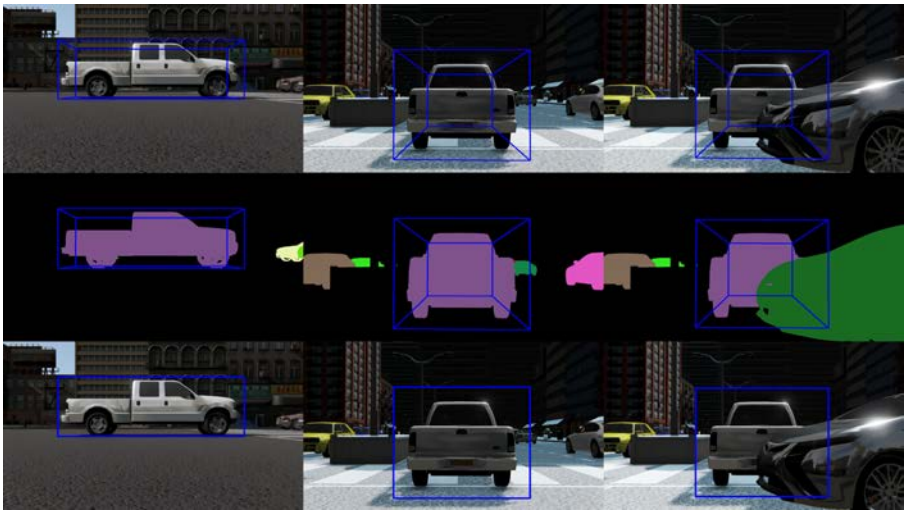

Figure A.5 – From 3D to 2D BBs.

method has the drawback of miss-annotating occluded objects; thus, potentially misleading the training of the object detector. The reason is that when a 2D BB is provided, it is supposed to contain a full object, not just a part of it. On the other hand, we can just project the vertices of each 3D BB to the image plane, and then find the 2D BB of the projected points, also with cropping at image borders. In this case, it is convenient to have a check for the degree of occlusion. This can be easily done by combining the pixel-wise instance ID information with the 2D BBs. When the ratio between the area of an instance ID and the area of the 2D BB is too low (the former must be contained in the latter), we can assume a heavy occlusion and discard the 2D BB. See Fig. A.5 for a visual example.

B Appendix

B.1 Scientific Articles

B.1.1 Journals

Gabriel Villalonga, Antonio M. López, *Co-Training for on-board deep object detection*, IEEE Access, 2020.

Gabriel Villalonga, Joost van de Weijer, Antonio M. López, *Recognizing new classes with synthetic data in the loop: application to traffic sign recognition*, Sensors, Special issue on *Advance in Sensors and Sensing Systems for Driving and Transportation*, 2020.

Antonio M. López, **Gabriel Villalonga**, Laura Sellart, Germán Ros, David Vázquez, Jiaolong Xu, Javier Marín, Azadeh Mozafari, *Training my car to see using virtual worlds*, Image and Vision Computing, Special issue on *Automotive Vision: Challenges, Trends, Technologies and Systems for Vision-Based Intelligent Vehicles*, 2017.

B.1.2 Internacional Conferences

Alejandro González, **Gabriel Villalonga**, Jiaolong Xu, David Vázquez, Jaume Amores, Antonio M López, *Multiview random forest of local experts combining RGB and LiDAR data for pedestrian detection*, IEEE Intelligent Vehicles Symposium (IV), 2015

Alejandro González, **Gabriel Villalonga**, German Ros, David Vázquez, Antonio M López, *3D-guided multiscale sliding window for pedestrian detection*, Iberian Conference on Pattern Recognition and Image Analysis, 2015.

Javad Zolfaghari Bengar, Abel Gonzalez-Garcia, **Gabriel Villalonga**, Bogdan Raducanu, Hamed Habibi Aghdam, Mikhail Mozerov, Antonio M. López, Joost van de Weijer, *Temporal coherence for active learning in videos*, International

Conference on Computer Vision, Workshop on *Computer Vision for Road Scene Understanding and Autonomous Driving*, 2019.

B.1.3 Book chapters

German Ros, Laura Sellart, **Gabriel Villalonga**, Elias Maidanik, Francisco Molero, Marc Garcia, Adriana Cedeño, Francisco Perez, Didier Ramirez, Eduardo Escobar, Jose Luis Gomez, David Vázquez, Antonio M. López, *Semantic segmentation of urban scenes via domain adaptation of SYNTHIA*, Chapter contribution to the book *Domain Adaptation in Computer Vision Applications*, Springer Series on *Advances in Computer Vision and Pattern Recognition*, 2017.

Bibliography

[1] Y. Abramson and Y. Freund. SEmi-automatic VIsuaL LEarning (SEVILLE): a tutorial on active learning for visual object recognition. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[2] Panos Achlioptas, O. Diamanti, Ioannis Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. In *International Conference on Machine Learning (ICML)*, 2018.

[3] Hamed H. Aghdam, Abel Gonzalez-Garcia, and Antonio M López Joost van de Weijer. Active learning for deep detection neural networks. In *International Conference on Computer Vision (ICCV)*, 2019.

[4] H.A. Alhaija, S. Karthik Mustikovela, L. Mescheder, A. Geiger, and C. Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer VisionSpecial issue on Synthetic Visual Data*, 126(9):961–972, 2018.

[5] R. Aljundi, C. Rahaf, and T. Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] R. Ambrus, V. Guizilini, J. Li, S. Pillai, and A. Gaidon. Two stream networks for self-supervised ego-motion estimation. In *Conference on Robot Learning (CoRL)*, 2019.

[7] Eduardo Arnold, Omar Y. Al-Jarrah, Mehrdad Dianati, Saber Fallah, David Oxtoby, and Alex Mouzakitis. A survey on 3D object detection methods for autonomous driving applications. *IEEE Trans. on Intelligent Transportation Systems*, January 2019.

[8] D. Arpit, S. Jastrzebsk, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien. A closer look at memorization in deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

[9] Alireza Asvadi, Luis Garrote, Cristiano Premebida, Paulo Peixoto, and Urbano J. Nunes. Multimodal vehicle detection: fusing 3D-LIDAR and color camera data. *Pattern Recognition Letters*, 115:20–29, November 2018.

[10] A. Awasthi and S. Sarawagi. Continual learning with neural networks: A review. In *ACM India Joint International Conference on Data Science and Management of Data*, 2019.

[11] C.H. Bahnsen, D. Vázquez, A.M. López, and T.B. Moeslund. Learning to remove rain in traffic surveillance by using synthetic data. In *International Conference on Computer Vision Theory and Applications (VISIGRAPP)*, 2019.

[12] M. Bai and R. Urtasun. Deep watershed transform for instance segmentation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] Maria-Florina Balcan, Avrim Blum, and Ke Yang. Co-training and expansion: Towards bridging theory and practice. In *Neural Information Processing Systems (NeurIPS)*, 2004.

[14] I. Barros, M. Cristani, B. Caputo, A. Rognhaugen, and T. Theoharis. Looking beyond appearances: Synthetic training data for deep cnns in re-identification. *Computer Vision and Image Understanding*, 167:50–62, February 2018.

[15] M. Bauman. Why waiting for perfect autonomous vehicles may cost lives. Rand Corporation, 2017.

[16] S. Beery, Y. Liu, D. Morris, J. Piavis, A. Kapoor, M. Meister, N. Joshi, and P. Perona. Synthetic examples improve generalization for rare classes. arXiv:1904.05916, 2019.

[17] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Conference on Computational Learning Theory (COLT)*, 1998.

[18] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[19] Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J. Weston. Generative and discriminative voxel modeling with convolutional neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2016.

[20] D.J. Butler, J. Wulff, G.B. Stanley, and M.J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision (ECCV)*, 2012.

[21] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[22] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2D pose estimation using part affinity fields. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[23] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teuliere, and T. Chateau. Deep MANTA: A coarse-to-fine many-task network for joint 2D and 3D vehicle analysis from monocular image. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[24] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: a survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[25] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-supervised Learning*. The MIT Press, 2006.

[26] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[27] C. Chen, A. Seff, A.L. Kornhauser, and J. Xiao. DeepDriving: Learning affordance for direct perception in autonomous driving. In *International Conference on Computer Vision (ICCV)*, 2015.

[28] Minmin Chen, Kilian Q. Weinberger, and John C. Blitzer. Co-training for domain adaptation. In *Neural Information Processing Systems (NeurIPS)*, 2011.

[29] Minmin Chen, K.Q. Weinberger, and Yixin Chen. Automatic feature decomposition for single view co-training. In *International Conference on Machine Learning (ICML)*, 2011.

[30] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[31] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D object detection network for autonomous driving. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[32] Y. Chen, W. Li, C. Sakaridis, D. Dai, and L. van Gool. Domain adaptive Faster R-CNN for object detection in the wild. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[33] Z. Chen and B. Liu. *Lifelong Machine Learning*. Morgan & Claypool, 2017.

[34] Chris Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision (ECCV)*, 2016.

[35] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D spatio-temporal convnets: Minkowski convolutional neural networks. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[36] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[37] Gabriela Csurka. *A Comprehensive Survey on Domain Adaptation for Visual Applications*, chapter 1. Advances in Computer Vision and Pattern Recognition. Springer, 2017.

[38] A. Dosovitskiy, G. Ros, F. Codevilla, A.M. López, and V. Koltun. CARLA: An open urban driving simulator. In *Conference on Robot Learning (CoRL)*, 2017.

[39] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[40] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/-index.html.

[41] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaeser, Fabian Timm, Werner Wiesbeck, and Klaus Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Trans. on Intelligent Transportation Systems*, February 2020.

[42] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao. Deep ordinal regression network for monocular depth estimation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[43] Matheus Gadelha, R. Wang, and Subhransu Maji. Multiresolution tree networks for 3d point cloud processing. In *European Conference on Computer Vision (ECCV)*, 2018.

[44] A. Gaidon, Q. Wang, Y. Cabon, and R. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[45] Yukang Gan, Xiangyu Xu, Wenxiu Sun, and Liang Lin. Monocular depth estimation with affinity, vertical pooling, and label enhancement. In *European Conference on Computer Vision (ECCV)*, 2018.

[46] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[47] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representation (ICLR)*, 2018.

[48] R.B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[49] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[50] C. Godard, O.M. Aodha, and G.J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[51] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth prediction. In *International Conference on Computer Vision (ICCV)*, 2019.

[52] A. González, G. Villalonga, J. Xu, D. Vázquez, J. Amores, and A. M. López. Multiview random forest of local experts combining rgb and lidar data for pedestrian detection. In *Intelligent Vehicles Symposium (IV)*, 2015.

[53] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Neural Information Processing Systems (NeurIPS)*, 2014.

[54] S: Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[55] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. arXiv:1802.05384, 2018.

[56] V. Guizilini, J. Li, R. Ambrus, S. Pillai, and A. Gaidon. Robust semi-supervised monocular depth estimation with reprojected distances. In *Conference on Robot Learning (CoRL)*, 2019.

[57] A. Gurram, O. Urfalioglu, I. Halfaoui, F. Bouzaraa, and Antonio M. Lopez. Monocular depth estimation by learning from heterogeneous datasets. In *Intelligent Vehicles Symposium (IV)*, 2018.

[58] U. Guz, D. Hakkani-Tür, S. Cuendet, and G. Tur. Co-training using prosodic and lexical information for sentence segmentation. In *Conference of the International Speech Communication Association (INTERSPEECH)*, 2007.

[59] V. Haltakov, C. Unger, and S. Ilic. Framework for generation of synthetic ground truth data for driver assistance applications. In *German Conference on Pattern Recognition (GCPR)*, 2013.

[60] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Neural Information Processing Systems (NeurIPS)*, 2018.

[61] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla. Understanding real world indoor scenes with synthetic data. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[62] H. Hattori, N. Lee, V.N. Boddeti, F. Beainy, K.M. Kitani, and T. Kanade. Synthesizing a scene-specific pedestrian detector and pose estimator for static video surveillance. *International Journal of Computer VisionSpecial issue on Synthetic Visual Data*, 126(9):1027–1044, 2018.

[63] D. Hernandez, L. Schneider, A. Espinosa, D. Vázquez, A.M. López, U. Franke, M. Pollefeys, and J.C. Moure. Slanted stixels: Representing san francisco's steepest streets. In *British Machine Vision Conference (BMVC)*, 2017.

[64] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M.M.A. Patwary, Y. Yang, and Y. Zhou. Deep learning scaling is predictable, empirically. arXiv:1712.00409, 2017.

[65] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell. CyCADA: Cycle-consistent adversarial domain adaptation. In *Machine Learning Research*, 2018.

[66] J. Hoffman, D. Wang, F. Yu, and T. Darrell. FCNs in the wild: Pixel-level adversarial and constraint-based adaptation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[67] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset. In *Conference on Robot Learning (CoRL)*, 2020.

[68] Nikita Jaipuria, Xianling Zhang, Rohan Bhasin, and Mayar Arafa. Deflating dataset bias using synthetic data augmentation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020.

[69] J. Janai, F. Guney, A. Behl, and A. Geiger. *Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art*. Now Publishers Inc, 2020.

[70] Jisoo Jeong, Seungeui Lee, , Jeesoo Kim, and Nojun Kwak. Consistency-based semi-supervised learning for object detection. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[71] C. Jiang, S. Qi, Y. Zhu, S. Huang, J. Lin, L.-F. Yu, D. Terzopoulos, and S.-C. Zhu. Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars. *International Journal of Computer VisionSpecial issue on Synthetic Visual Data*, 126(9):920–941, 2018.

[72] M. Johnson-Roberson, C. Barto, R. Mehta, S. Nittur S., K. Rosaen, and . Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? In *International Conference on Robotics and Automation (ICRA)*, 2017.

[73] E. Kalogerakis, Melinos Averkiou, Subhransu Maji, and S. Chaudhuri. 3d shape segmentation with projective convolutional networks. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[74] Dahun Kim, Donghyeon Cho, Donggeun Yoo, and In So Kweon. Learning image representations by completing damaged jigsaw puzzles. In *Winter conf. on Applications of Computer Vision (WACV)*, 2018.

[75] Seunghyeon Kim, Jaehoon Choi, Taekyung Kim, and Changick Kim. Self-training and adversarial background regularization for unsupervised domain adaptive one-stage object detection. In *International Conference on Computer Vision (ICCV)*, 2019.

[76] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A.A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[77] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[78] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2012.

[79] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. Joint 3D proposal generation and object detection from view aggregation. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[80] Sampo Kuutti, Saber Fallah, Konstantinos Katsaros, Mehrdad Dianati, Francis Mccullough, and Alexandros Mouzakitis. A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications. *IEEE Internet of Things Journal*, 5(2):829–846, 2019.

[81] J. Lahoud and B. Ghanem. 2d-driven 3d object detection in rgb-d images. In *International Conference on Computer Vision (ICCV)*, 2017.

[82] Alex Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[83] Kuan-Hui Lee, G. Ros, J. Li, and Adrien Gaidon. Spigan: Privileged adversarial learning from simulation. In *International Conference on Learning Representation (ICLR)*, 2019.

[84] Anat Levin, Paul Viola, and Yoav Freund. Unsupervised improvement of visual detectors using co-training. In *International Conference on Computer Vision (ICCV)*, 2003.

[85] J. Li, C. Wang, H. Zhu, Y. Mao, H.-S. Fang, and C. Lu. CrowdPose: Efficient crowded scenes pose estimation and a new benchmark. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[86] Z. Li and D. Hoiem. Learning without forgetting. In *European Conference on Computer Vision (ECCV)*, 2016.

[87] S. Liang, Y. Li, , and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representation (ICLR)*, 2018.

[88] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128:261–318, 2020.

[89] M. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. In *Neural Information Processing Systems (NeurIPS)*, 2017.

[90] S. Liu, J. Jia, S. Fidle, and R. Urtasun. SGN: sequential grouping networks for instance segmentation. In *International Conference on Computer Vision (ICCV)*, 2017.

[91] S. Liu, L. Liu, J. Tang, S. Wu, and J.-L. Gaudiot. *Creating Autonomous Vehicle Systems.* Morgan & Claypool, 2018.

[92] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. Berg. SSD: single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, 2016.

[93] X. Liu, M. Masana, L. Herranz, J. van de Weijer, A.M. López, and A.D. Bagdanov. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *International Conference on Pattern Recognition (ICPR)*, 2018.

[94] Vishnu Suresh Lokhande, Songwong Tasneeyapant, Abhay Venkatesh, Sathya N. Ravi, and Vikas Singh. Generating accurate pseudo-labels in semi-supervised learning and avoiding overconfident predictions via hermite polynomial activations. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[95] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[96] Antonio M. López, Gabriel Villalonga, Laura Sellart, Germán Ros, David Vázquez, Jiaolong Xu, Javier Marin, and Azadeh Mozafari. Training my car to see using virtual worlds. *Image and Vision Computing*, 68:102–118, 2017.

[97] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world. arXiv:2006.09348, 2020.

[98] J. Marin, D. Vázquez, D. Gerónimo, and A.M. López. Learning appearance in virtual scenarios for pedestrian detection. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[99] M. Masana, I. Ruiz, J. Serrat, J. van de Weijer, and A.M. López. Metric learning for novelty and anomaly detection. In *British Machine Vision Conference (BMVC)*, 2018.

[100] N. Mayer, E. Ilg, P. Fischer, C. Hazirbas, D. Cremers, A. Dosovitskiy, and T. Brox. What makes good synthetic training data for learning disparity and optical flow estimation? *International Journal of Computer VisionSpecial issue on Synthetic Visual Data*, 126(9):942–960, 2018.

[101] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[102] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka. 3D bounding box estimation using deep learning and geometry. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[103] M. Müller, N. Smith, and B. Ghanem. A benchmark and simulator for uav tracking. In *European Conference on Computer Vision (ECCV)*, 2016.

[104] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of cotraining. In *Int. Conference on Information and Knowledge Management (CIKM)*, 2000.

[105] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *International Conference on Computer Vision (ICCV)*, 2015.

[106] Avital Oliver, Augustus Odena, Colin Raffel, Ekin D. Cubuk, and Ian J. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In *Neural Information Processing Systems (NeurIPS)*, 2018.

[107] T. Zhou P. Isola, J. Zhu and A. Efros. Image-to-image translation with conditional adversarial nets. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[108] S.J. Pan and Q. Yang. A survey on transfer learning. *IEEE Trans. on Knowledge and Data Engineering*, 22(10):1345–1359, 2009.

[109] G.I. Parisi, R. Kemker J.L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks. *Neural Networks*, 113:54–71, May 2019.

[110] X. Peng, B. Sun, K. Ali, and K. Saenko. Learning deep object detectors from 3D models. In *International Conference on Computer Vision (ICCV)*, 2015.

[111] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3D geometry to deformable part models. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[112] Andreas Pfeuffer and Klaus Dietmayer. Optimal sensor data fusion architecture for object detection in adverse weather conditions. In *International Conference on Information Fusion (FUSION)*, 2018.

[113] S. Pillai, R. Ambrus, and A. Gaidon. SuperDepth: Self-supervised, super-resolved monocular depth estimation. In *International Conference on Robotics and Automation (ICRA)*, 2018.

[114] M.A.F. Pimentel, D.A. Clifton, L.A. Clifton, and L. Tarassenko. Review of novelty detection. *Signal Processing*, 99:215–249, June 2014.

[115] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3D object detection from RGB-D data. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[116] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Neural Information Processing Systems (NeurIPS)*, 2017.

[117] Siyuan Qiao, Wei Shen, Zhishuai Zhang, Bo Wang, and Alan Yuille. Deep co-training for semi-supervised image recognition. In *European Conference on Computer Vision (ECCV)*, 2018.

[118] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[119] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NeurIPS)*, 2015.

[120] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real time object detection with region proposal networks. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[121] S.R. Richter, Z. Hayder, and V. Koltun. Playing for benchmarks. In *International Conference on Computer Vision (ICCV)*, 2017.

[122] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[123] G. Ros, L. Sellart, J. Materzyska, D. Vázquez, and A.M. López. The SYNTHIA dataset: a large collection of synthetic images for semantic segmentation of urban scenes. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[124] C. Rosenberg, M. Hebert, and H. Schneiderman. Semi-supervised self-training of object detection models. In *Workshop on Applications of Computer Vision (WACV)*, 2005.

[125] S. Roy, A. Unmesh, and V.P. Namboodiri. Deep active learning for object detection. In *British Machine Vision Conference (BMVC)*, 2018.

[126] Ch. Sakaridis, D. Dai, and L. van Gool. Semantic foggy scene understanding with synthetic data. *International Journal of Computer VisionSpecial issue on Synthetic Visual Data*, 126(9):973–992, 2018.

[127] M. Savva, A.X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. arXiv:1712.03931, 2017.

[128] Burr Settles. *Active learning*, volume 6 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*, pages 1–114. Morgan & Claypool, 2012.

[129] S. Shah, D. Dey, C. Lovett, and A. Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics (FSR)*, 2017.

[130] N. Sharma, V. Jain, and A. Mishra. An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, 132:377–384, 2018.

[131] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[132] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipmanand, and A. Blake. Real-time human pose recognition in parts from a single depth image. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[133] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[134] C.M. Silva, B.M. Masini, Gianluigi Ferrari, and Ilaria Thibault. A survey on infrastructure-based vehicular networks. *Hindawi Mobile Information System*, 2017.

[135] J. Skinner, S. Garg, N. Sünderhauf, P. Corke, B. Upcroft, and M.l Milford. High-fidelity simulation for evaluating robotic vision performance. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[136] A. A. Soltani, H. Huang, J. Wu, T. D. Kulkarni, and J. B. Tenenbaum. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[137] C.R. de Souza, A. Gaidon, Y. Cabon, and A.M. López. Procedural generation of videos to train deep action recognition networks. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[138] C.R. de Souza, A. Gaidon, Y. Cabon, Naila Murray, and A.M. López. Generating human action videos by coupling 3d game engines and probabilistic graphical models. *International Journal of Computer Vision*, 128:1505–1536, 2019.

[139] H. Su, C.R. Qi, Y. Li, and L. Guibas. Render for CNN: viewpoint estimation in images using CNNs trained with rendered 3D model views. In *International Conference on Computer Vision (ICCV)*, 2015.

[140] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *International Conference on Computer Vision (ICCV)*, 2017.

[141] K. Sun, B. Xiao, D. Liu, and J. Wang. Deep high-resolution representation learning for human pose estimation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[142] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[143] Yongbin Sun, Yue Wang, Z. Liu, Joshua E. Siegel, and S. Sarma. Pointgrow: Autoregressively learned point cloud generation with self-attention. In *Winter conf. on Applications of Computer Vision (WACV)*, 2020.

[144] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. In *International Conference on Learning Representation (ICLR)*, 2017.

[145] G.R. Taylor, A.J. Chosak, and P.C. Brewer. OVVV: Using virtual worlds to design and evaluate surveillance systems. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[146] H. Thomas, C. R. Qi, J. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. KPConv: Flexible and deformable convolution for point clouds. In *International Conference on Computer Vision (ICCV)*, 2019.

[147] Y. Tian, X. Li, K. Wang, and F.-Y. Wang. Training and testing object detectors with virtual images. *IEEE/CAA Journal of Automatica Sinica*, 5(2):539–546, 2018.

[148] T.Kim, M. Cha, H. Kim, J.K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. In *Machine Learning Research*, 2017.

[149] I. Triguero, S. García, and F. Herrera. Self-labeled techniques for semi-supervised learning: Taxonomy, software and empirical study. *Signal Processing*, 42(2):245–284, 2015.

[150] Gokhan Tur. Co-adaptation: Adaptive co-training for semi-supervised learning. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2009.

[151] J. Uhrig, M. Cordts, U. Franke, and T. Brox. Pixel-level encoding and depth layering for instance-level semantic labelling. In *German Conference on Pattern Recognition (GCPR)*, 2016.

[152] J. Uhrig, E. Rehder, B. Fröhlich, U. Franke, and T. Brox. Box2Pix: Single-shot instance segmentation by assigning pixels to object boxes. In *Intelligent Vehicles Symposium (IV)*, 2018.

[153] Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109:373–440, 2020.

[154] G. Varol, J. Romero, X. Martin, N. Mahmood, M.J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[155] D. Vázquez, A.M. López, J. Marín, D. Ponsa, and D. Gerónimo. Virtual and real world adaptation for pedestrian detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 36:797–809, 2014.

[156] D. Vázquez, A.M. López, D. Ponsa, and J. Marin. Cool world: domain adaptation of virtual and real worlds for human detection using active learning. In *Neural Information Processing Systems (NeurIPS) Workshops*, 2011.

[157] Dominic Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems (RSS)*, 2015.

[158] Keze Wang, Xiaopeng Yan, Dongyu Zhang, Lei Zhang, and Liang Lin. Towards human-machine cooperation: Self-supervised sample mining for object detection. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[159] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, October 2018.

[160] Peng Wang, Xinyu Huang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2019.

[161] T. Wang, M. Liu, J. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[162] Wei Wang and Zhi-Hua Zhou. Analyzing co-training style algorithms. In *European Conference on Machine Learning (ECML)*, 2007.

[163] Wei Wang and Zhi-Hua Zhou. A new analysis of co-training. In *International Conference on Machine Learning (ICML)*, 2010.

[164] Karl Weiss, T.M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(9), 2016.

[165] WHO. Global status report on road safety. https://apps.who.int/iris/bitstream/handle/10665/277370/WHO-NMH-NVI-18.20-eng.pdf?ua=1, 2018.

[166] Garrett Wilson and Diane J. Cook. A survey of unsupervised deep domain adaptation. *ACM Transactions on Intelligent Systems and Technology*, 11(5), 2020.

[167] L. Woensel and G. Archer. Ten technologies which could change our lives: potential impacts and policy implication. European Parliamentary Research Service, 2015.

[168] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[169] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[170] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Data-driven 3d voxel patterns for object category recognition. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[171] J. Xu, S. Ramos, D. Vázquez, and A.M. López. Domain adaptation of deformable part-based models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 36(12):2367–2380, 2014.

[172] J. Xu, D. Vázquez, A.M. López, J. Marín, and D. Ponsa. Learning a part-based pedestrian detector in a virtual world. *IEEE Trans. on Intelligent Transportation Systems*, 15:2121 – 2131, 2014.

[173] J. Xu, D. Vázquez, K. Mikolajczyk, and A.M. López. Hierarchical online domain adaptation of deformable part-based models. In *International Conference on Robotics and Automation (ICRA)*, 2016.

[174] J. Xu, L. Xiao, and A.M. López. Self-supervised domain adaptation for computer vision tasks. *IEEE Access*, 7:156694–156706, November 2019.

[175] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18:3337, 10 2018.

[176] B. Yang, M. Liang, and R. Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In *Conference on Robot Learning (CoRL)*, 2018.

[177] B. Yang, W. Luo, and R. Urtasun. Pixor: Real-time 3d object detection from point clouds. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[178] G. Yang, X. Huang, Z. Hao, M. Liu, S. Belongie, and B. Hariharan. Pointflow: 3D point cloud generation with continuous normalizing flows. In *International Conference on Computer Vision (ICCV)*, 2019.

[179] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 1995.

[180] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representation (ICLR)*, 2016.

[181] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor W. Tsang, and Masashi Sugiyama. How does disagreement help generalization against label corruption? In *International Conference on Machine Learning (ICML)*, 2019.

[182] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.

[183] Y. Zhang, P. David, and B. Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *International Conference on Computer Vision (ICCV)*, 2017.

[184] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[185] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knowledge and Information Systems*, 24(3):415–439, 2010.

[186] J. Zhu, T. Park, P. Isola, and A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *International Conference on Computer Vision (ICCV)*, 2017.

[187] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu. Traffic-sign detection and classification in the wild. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[188] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. arXiv:1911.02685, 2020.

[189] Javad Zolfaghari, Abel Gonzalez, Gabriel Villalonga, Bogdan Raducanu, Hamed Aghdam, Mikhail Mozerov, A.M. López, and Joost van de Weijer. Temporal coherence for active learning in videos. In *International Conference on Computer Vision (ICCV) Workshops*, 2019.

[190] Y. Zou, Z. Yu, B.V. Kumar, and J. Wang. Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In *European Conference on Computer Vision (ECCV)*, 2018.

[191] Yang Zou, Zhiding Yu, Xiaofeng Liu, B.V.K. Vijaya Kumar, and Jinsong Wang. Confidence regularized self-training. In *International Conference on Computer Vision (ICCV)*, 2019.