LiU-ITN-TEK-A--21/043-SE

Synthetic Data for Training and Evaluation of Critical Traffic Scenarios

Sofie Collin

2021-06-17

Department of Science and Technology
Linköping University
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings universitet
601 74 Norrköping

LiU-ITN-TEK-A--21/043-SE

Synthetic Data for Training and Evaluation of Critical Traffic Scenarios

The thesis work carried out in Elektroteknik
at Tekniska högskolan at
Linköpings universitet

Sofie Collin

Norrköping 2021-06-17

Department of Science and Technology
Linköping University
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings universitet
601 74 Norrköping

Abstract

Modern camera-based vehicle safety systems heavily rely on machine learning and consequently require large amounts of training data to perform reliably. However, collecting and annotating the needed data is an extremely expensive and time-consuming process. In addition, it is exceptionally difficult to collect data that covers critical scenarios. This thesis investigates to what extent synthetic data can replace real-world data for these scenarios. Since only a limited amount of data consisting of such real-world scenarios is available, this thesis instead makes use of proxy scenarios, e.g. situations when pedestrians are located closely in front of the vehicle (for example at a crosswalk). The presented approach involves training a detector on real-world data where all samples of these proxy scenarios have been removed and compare it to other detectors trained on data where the removed samples have been replaced with various degrees of synthetic data.

A method for generating and automatically and accurately annotating synthetic data, using features in the CARLA simulator, is presented. Also, the domain gap between the synthetic and real-world data is analyzed and methods in domain adaptation and data augmentation are reviewed.

The presented experiments show that aligning statistical properties between the synthetic and real-world datasets distinctly mitigates the domain gap. There are also clear indications that synthetic data can help detect pedestrians in critical traffic situations.

Acknowledgments

First and foremost, I would like to express my gratitude to Veoneer for the opportunity to carry out this thesis. Especially, I would like to thank my supervisors Per Cronvall and Malcolm Vigren for their excellent knowledge and support. Their deep interest, strong commitment and continuous encouragement have been invaluable throughout this thesis. Additionally, I would like to thank Stefan Lilliehjort for welcoming me on my first day and providing me with the needed tools.

Furthermore, I would like to thank my supervisor Gabriel Eilertsen and my examiner Jonas Unger at Linköping University, for their guidance and insightful suggestions throughout this thesis.

Linköping, June 2021
Sofie Collin

Contents

List of Figures

List of Tables

1 Introduction

In this introductory chapter, the motivation and aim of the thesis are presented along with specific research questions and delimitations.

The study is carried out at Veoneer, one of the leading companies in automotive technology. They develop state-of-the-art solutions for advanced driving assistance systems (ADAS) and autonomous driving (AD).

1.1 Motivation

Each year, approximately 1.35 million people die in traffic [20]. Even though some accidents are caused by factors such as unsafe road infrastructure, most of the traffic accidents occur due to mistakes made by the driver. These mistakes are often triggered by speeding, distracted driving or driving under the influence of alcohol or drugs. The victims of road accidents are most often among the vulnerable road users, such as pedestrians, cyclists and motorcyclists.

In order to increase safety on the roads, ADAS systems are of great importance. These systems are designed to assist the driver in complex situations and by extension manage human errors and reduce road fatalities.

Modern camera-based ADAS features heavily depend on machine learning and consequently require large amounts of training data to perform reliably. One significant problem, in terms of safety, is the lack of training data for critical scenarios. Data that covers potential accidents is the most difficult to collect and at the same time the most important in order to ensure safe driving. One way to approach this problem could be to use synthetic data for these scenarios.

1.2 Aim

The purpose of this thesis is to generate synthetic data and investigate to what extent this data can replace real-world data for critical traffic scenarios involving pedestrians.

1.3 Research questions

The questions that this thesis aims to investigate and answer are the following:

1. What can be said about the domain gap between the synthetic and real-world dataset, in terms of differences in statistical properties and diversity, and how does it affect the detector's ability to detect pedestrians?
2. Is it possible to smooth the gap between the virtual and real domain by aligning statistical properties?
3. Can specific data augmentation techniques boost the detector's performance by providing more variation to the synthetic dataset?

1.4 Delimitations

The synthetic data will be generated using CARLA (0.9.11), an open-source autonomous driving simulator. No other platform will be investigated or used in this project.

The study will only focus on pedestrian detection. The critical scenarios are therefore limited to potential accidents involving pedestrians, namely situations when they are located close in front of the vehicle.

2 Background

In this chapter, the basic concepts of neural networks and image generation are presented. Also, the thesis is put in a context by providing relevant information as well as research in the fields of object detection, autonomous driving and synthetic data. Finally, an overview of CARLA is presented.

2.1 Neural networks

A neural network (NN) is a computational learning system that is designed to recognize patterns in input data and translate them into the desired output. Inspired by the human brain, artificial neural networks consist of a collection of connected neurons, or nodes, that transmit signals between each other. Figure 2.1 shows an illustration of a simple feedforward neural network. As can be seen in the figure, the nodes are organized into layers. The first layer consists of all input variables while the last layer delivers the output. The layers in between are referred to as hidden layers. If a network has more than one hidden layer, it is called a deep neural network (DNN) [9]. The size of the network and the number of nodes per layer vary greatly depending on the task.

Each node in the network combines its inputs with a set of coefficients, or weights. These weights help determine the significance of each input with respect to the task at hand. This, by amplifying or dampening their impact. These weighted inputs, along with a bias term, are summed together and passed through an activation function, see Figure 2.1. The bias term is added to allow the activation function to be shifted. The activation function decides if, or to what extent, data should be passed to the next layer. Some examples of popular activation functions are sigmoid, hyperbolic tangent (tanh) and Rectified Linear Unit (ReLU) [9].

During training, the weights are updated with the aim of improving the accuracy of the result. At the final layer, the predicted output is evaluated against the expected output using a loss function. Backpropagation is then used to minimize the loss by propagating back through the network and adjusting the weights. The level of adjustment is decided by the gradient of the loss function with respect to each of the weights. The update is typically done using gradient decent (or a related version such as stochastic gradient descent)

$$w_i^{t+1} = w_i^t - \eta \frac{\partial E^t}{\partial w_i^t}, \tag{2.1}$$

where η is the learning rate and E the loss function [11]. The index t represents the iteration steps and i the weights.

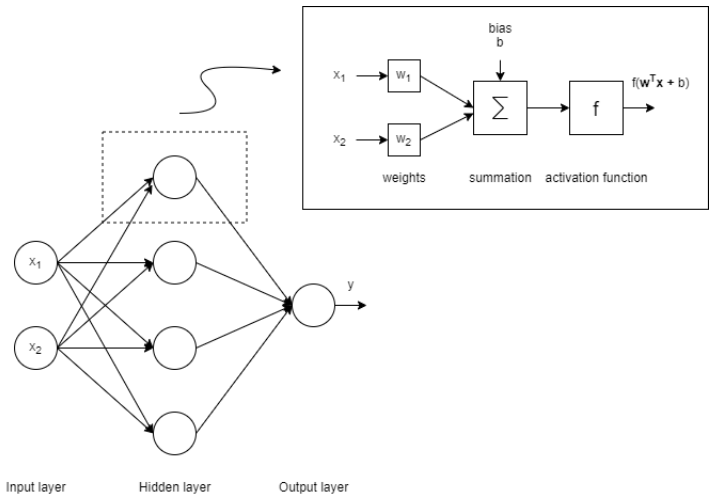


Figure 2.1: Example of a neural network with two input nodes, one output node and one hidden layer with four hidden nodes.

Convolutional neural networks

A convolutional neural network (CNN) is a type of neural network that uses convolution in at least one of its layers. These networks are well-suited for processing data that is organized in a grid-like manner, such as images [10].

A typical CNN architecture consists of three different types of layers stacked together; convolutional layers, pooling layers and fully connected layers [19]. In the first type of layer, convolution is performed between the input data and a filter. This, to extract information, or features, from the data. The so-called feature map is then fed to other layers in order to learn more features. This layer typically includes a non-linear activation function that controls the output. In the pooling layer, the feature map is downsampled. This reduces the computational cost and makes the representation invariant to small translations. The fully connected layer is typically placed at the end of the network. It uses the features provided by previous layers in order to classify the data.

2.2 Image generation

In the context of computer graphics, image generation can be described as the process of artificially generating images containing some specified content. The image generation pipeline can roughly be divided into two parts; modeling and rendering [32].

Modeling is the process of building a geometric representation of the virtual environment. This includes configuration of 3D models, textures and light sources. The extent of the virtual environment can range from a single scene to entire worlds where a simulated sensor captures images as it moves around.

Given a description of the scene, rendering is the process of actually producing and visually displaying the image. This includes simulating the interaction between light and surfaces given a certain camera configuration. This can be done using techniques such as rasterization or ray tracing [3]. In rasterization, the 3D objects in the scene are created from a mesh of polygons. These polygons are converted into pixels on the 2D screen and assigned a color. Complex lighting effects can be achieved by using the information about positions, textures,

colors and normals stored in each polygon (and its vertices). Rasterization is commonly used in game engines since it is a relatively fast technique. However, it does not take the full light transport into account. Ray tracing, on the other hand, tracks the light as it travels from its source and interacts with all objects in the scene. This can create a more realistic result but to a higher computational cost.

2.3 Object detection for autonomous driving

Object detection, one of the most challenging problems in computer vision, focuses on solving two tasks; to locate and to classify objects in images. This is a fundamental part of autonomous driving systems since they heavily depend on an accurate perception of the environment in order to perform reliably. To be able to make the right decisions, it is crucial that the autonomous vehicle can, in real-time, locate and classify nearby objects such as other vehicles, road signs and pedestrians.

The following sections describe common evaluation methods and popular net architectures.

Performance metrics

When evaluating the performance of a detector, there are some important concepts to consider [21]:

- True Positive (TP): A correct detection of an object.
- False Positive (FP): An incorrect detection of a nonexistent object.
- False Negative (FN): An undetected object.

The above-stated definitions require an establishment of what is considered correct and incorrect. This is typically done by setting a threshold for the intersection over union (IOU) between the predicted bounding box and the ground truth. The IOU is calculated by dividing the overlapping area by the united area of the two boxes B_{pred} and B_{truth}, according to

$$\text{IOU}_{truth}^{pred} = \frac{area(B_{pred} \cap B_{truth})}{area(B_{pred} \cup B_{truth})}. \tag{2.2}$$

If the IOU exceeds the given threshold, the detection is considered correct otherwise incorrect.

Most metrics used for object detection build upon these concepts. Two commonly used metrics are precision and recall, defined as

$$\begin{aligned} \text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}. \end{aligned} \tag{2.3}$$

Precision measures a model's ability to *only* detect relevant objects while recall describes the ability to find *all* relevant objects. Ideally, a detector should find all ground truth objects (FN=0, giving high recall) while only identifying relevant objects (FP=0, giving high precision). Depending on application, the cost of false negatives and false positives vary. For a pedestrian detector in autonomous driving, the cost of FN is very high since failing to detect a pedestrian could lead to fatal accidents. Therefore, recall is an important measure. On the other hand, too many FPs could cause confusion and disturbances in traffic.

Another way to keep track of the FP performance is to calculate the average number of FPs per image, according to

$$\text{FPPI} = \frac{\text{FP}}{N_{frames}}, \tag{2.4}$$

where N_{frames} is the total number of frames (images) in the test set.

Furthermore, it is common to combine precision and recall and create a metric called F1. F1 is the harmonic mean of a model's precision and recall, defined as

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}. \tag{2.5}$$

In this thesis the metrics recall, FPPI and F1 are chosen with an applied IOU threshold of 35%. From here on, they are referred to as TP-rate, FP/frame and F1-score.

R-CNN and YOLO

Major improvements have been made in the field of object detection, starting 2013 when Girshick et al. introduced Region Based Convolutional Neural Networks (R-CNN) [8]. In a first step, R-CNN proposes a bunch of candidate bounding boxes using selective search. Each of these proposed regions is then passed through a CNN to extract features. Finally, a Support Vector Machine (SVM) and a linear regression model are used to classify the region and tighten the bounding box. Although performing well, the R-CNN is quite slow. One reason for this is that the feature extraction is done independently for each of the proposed regions. Another reason is that it trains three different models separately; the feature extractor, the classifier and the regressor. In 2015, Girshick addressed these issues as he proposed Fast R-CNN [7]. One year later, Faster R-CNN was presented [26]. Despite further improvements, Faster R-CNN does not quite reach real-time speed.

You Only Look Once (YOLO)[23] is another approach to object detection, targeting real-time applications. In YOLO, the region proposal and the classification stages are merged together, creating a single-stage pipeline that is able to operate in real-time. This architecture has since been updated to a second [24] and third version [25]. Though, being superior in speed YOLO lacks a bit in accuracy compared to R-CNN.

SqueezeDet

SqueezeDet, the network used in this thesis, specializes in real-time object detection for autonomous driving [36]. The model claims high accuracy and real-time inference speed as well as small model size and energy efficiency. In autonomous driving systems, all these properties are important in order to ensure safety and enable embedded system deployment.

SqueezeDet has adapted YOLO's approach of using a single-stage detection pipeline in order to provide a fast inference speed. The full detection pipeline is shown in Figure 2.2. As can be seen in the figure, the input image is first put through a CNN which extracts a feature map. This feature map is then fed to what Wu et al. [36] refers to as the *ConvDet* layer. This layer efficiently generates thousands of region proposals, centered around uniformly distributed grids. Each bounding box is associated with a confidence score

$$\text{CS} = P(\text{Object}) \cdot \text{IOU}_{truth}^{pred}, \tag{2.6}$$

where $P(\text{Object})$ is the probability that the predicted box contains an object of interest and $\text{IOU}_{truth}^{pred}$ is the intersection over union between the predicted box and the ground truth. The

bounding box is assigned the class, c, that corresponds to the highest conditional probability, $P(\text{c}|\text{Object})$, and the final confidence of the bounding box prediction is expressed as

$$\max_c P(\text{c}|\text{Object}) \cdot \text{CS}. \tag{2.7}$$

The N bounding boxes with the highest confidence are kept. Finally, non-max suppression is used to filter out redundant bounding boxes.

As the backbone, SqueezeDet uses SqueezeNet [13], which achieves high accuracy with relatively few parameters.

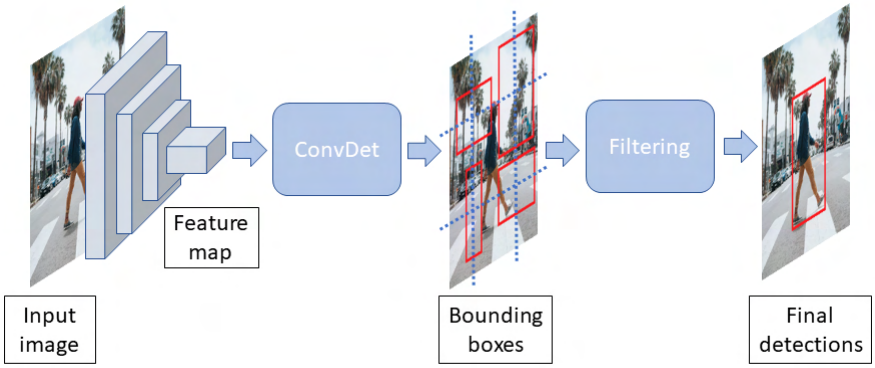


Figure 2.2: SqueezeDet detection pipeline.

2.4 Synthetic data in machine learning

Over the past decade, machine learning has made great progress and powers many of today's most innovative technologies. However, machine learning heavily depends on massive amounts of diverse and well-annotated data to perform reliably. The process of collecting and annotating this data is extremely expensive and time-consuming. Sometimes, as in the case of critical traffic scenarios, it is also exceptionally difficult to collect the needed data. In order to deal with these problems, synthetic data can be of great use. A synthetic dataset contains data that has been generated artificially rather than captured from the real world. Using a simulator to generate data provides the possibility to collect data of rare scenarios and also to efficiently annotate it. However, there are two major challenges that need to be addressed when generating and using synthetic data as a substitute for real-world data, namely achieving sufficient domain realism and feature variation [32]. This is a difficult task due to the obvious domain shift and the limited size and variation of virtual worlds.

The domain shift

Generalization, a model's ability to adapt and perform well on unseen data, is a crucial part of machine learning. Many machine learning techniques make the assumption that the training and test data are drawn from the same distribution. When this constraint is violated the model will not generalize well [15]. Even the slightest change, maybe not even observable by the human eye, may cause a model to perform poorly [17]. This problem is especially noticeable in deep learning algorithms, where complex models tend to pick up on and adapt to very subtle details in the training data.

This change of distribution between the training and test data is referred to as a *domain shift*. A domain shift can, for example, be caused by differences in image characteristics (color, contrast, brightness, noise, etc.) or sampling bias (when certain outcomes are more or less likely to occur) [16].

How to best address a domain shift is of great interest to the machine learning community and has been the topic of many research studies. There are several related fields that deal with domain shifts, two of the big ones are domain adaptation and domain generalization [35].

Domain adaptation is a special case of transfer learning [22] that aims to handle problems that arise when a model trained on data from a source distribution is meant to be tested on data from a different, but related, target distribution. It is assumed that the two domains share the same classes or objectives, i.e. that the same task is to be solved in both domains. There are multiple approaches to domain adaptation, both supervised and unsupervised. Both categories require target data for the training procedure, though the unsupervised techniques do not need it to be annotated [37]. The complexity of the studied methods varies from matching simple image statistics to finding more complex patterns using deep learning. Abramov et al. [1] present a simple yet effective method that involves alignment of color histogram, mean and covariance. More complex examples of discrepancy-based methods, that aim to learn deep neural transformations, are presented by Long et al. [18] and Tzeng et al. [34]. There are also examples of adversarial discriminative models, that pit a generator against a discriminator [33][6][29][12].

Unlike domain adaptation, domain generalization does not use any target data, not even unlabeled, in the training procedure. Thus, it deals with the challenge of generalizing to a completely unseen domain by learning from one or several source domains. According to Wang et al. [35], domain generalization techniques can be divided into three categories; data manipulation, representation learning and learning strategy. The methods in the data manipulation category focus on making modifications to the training data with the aim of learning general structures. This category includes, but is not limited to, methods in data augmentation. Augmentation methods can consist of typical image operations such as rotation, translation, scaling, flipping, cropping, adding noise, manipulation of contrast or color. These operations have been shown to have a remarkable effect on reducing overfitting and hence enhancing the generalization performance of deep learning algorithms [28]. Another family of techniques that fall under the topic of augmentation is domain randomization. Domain randomization focuses on generating new, diverse samples, that will contribute to a more complex environment and force the model to learn the essential features of specific objects. This includes, for example, randomizing positions, textures, shapes and quantities of objects. These techniques have been proven to be very effective in bridging the gap between the synthetic and the real world and have been used for several different tasks, including object localization [30] and object detection [31].

The second category, representation learning, can according to Wang et al. [35] be divided into two sub-categories. The first focuses on reducing specific feature discrepancies between several source domains to make these features domain-invariant and generalizable to unseen domains. The other sub-category focuses on feature disentanglement, i.e. to break down each feature into variables and turning these into separate dimensions. This, to identify which variables are domain-specific and which are domain-shared.

The third category, learning strategy, aims to boost the generalization performance by focusing on the learning strategy itself. For a deeper description of this category, readers are referred to the survey by Wang et al. [35].

The limited variation of virtual worlds

To create synthetic datasets, simulators built upon game engines are often used. Some examples of these are SYNTHIA [27] (created within the Unity framework) and GTA SIM10K [14] (generated from the video game GTA5, which uses the RAGE engine). In addition to lacking in realism, these datasets are often limited when it comes to feature variation and coverage. Virtual worlds are not as complex as the real world and struggle with providing unique enough combinations of features to match the real world. This, because virtual worlds are finite and models, such as cars and pedestrians, are often taken from a pre-defined library.

To deal with datasets with a lack of diversity, previously mentioned techniques in data augmentation are commonly used [28]. This, to cover a wider range of conditions that might be included in the target data.

2.5 CARLA– Possibilities and limitations

CARLA (Car Learning to Act) is a simulator for autonomous driving, licensed under the MIT license [5]. As can be seen in Figure 2.3, CARLA builds upon Unreal Engine 4 (UE4) and consists of a server-client architecture. The server-side of the architecture is responsible for the simulation itself, including sensor rendering, updating the current state and simulating physics. By leveraging scrips on the client-side, the user can control the content of the simulation.

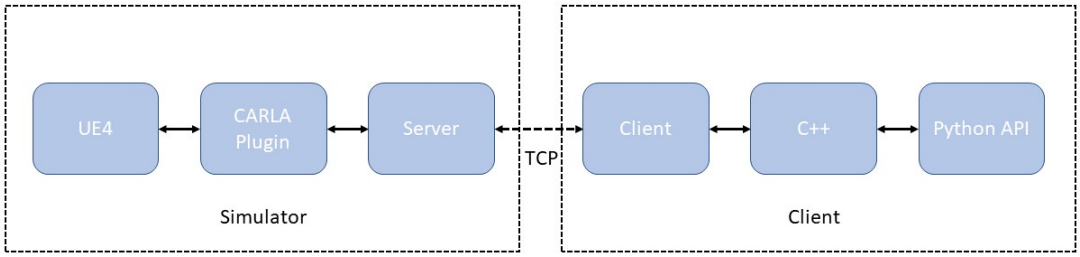


Figure 2.3: CARLA architecture.

The Python API offers numerous features useful for this thesis. For starters, it provides access to a number of predefined maps, all of which include a 3D model of a city and its road definition. The weather and lighting conditions of the world can either be manually tweaked or chosen from a set of predefined settings. The appearance of a vehicle or a pedestrian can be changed by choosing from the available blueprint library. All these functionalities are useful in order to generate data with as much variation as possible.

Another neat feature is the autopilot mode. It simulates traffic in the world and makes the vehicles roam the city streets. Also, pedestrians can be equipped with an AI which makes them move automatically and walk to random locations. All this makes the simulation look and behave like a real urban environment. By attaching a camera to a vehicle it is possible to continuously collect data as the vehicle drives around.

In order to collect data of critical scenarios, there is another interesting function, namely `set_pedestrians_cross_factor()`. By setting a percentage, the user can control how many of the pedestrians are allowed to randomly cross over roads. It is also possible to set the pedestrians' speed between walking and running.

As mentioned, the existing Python API provides a lot of useful features. However, the program is in constant development and lacks, at the time of writing, some functionalities needed for this thesis. The most crucial one is the possibility to extract 2D bounding boxes for pedestrians from a sensor's point of view. What is available, is the world coordinates of the 3D bounding boxes. These could be converted to 2D coordinates by using the camera matrix to project the boxes onto the image plane. What is left is then to decide which pedestrians are visible to the camera. This could, for example, be done by filtering pedestrians with respect to distance and angle. Potential occlusion could then be handled using a depth camera or a semantic LIDAR, both available in CARLA. When trying an open-source solution for this [2] (and slightly modifying it to work for pedestrians instead of vehicles), it was made clear that the pedestrian 3D bounding boxes provided by CARLA were not always that precise, resulting in imprecise 2D projections. An example of this is shown in Figure 2.4. Since they deviate a lot from the way Veoneer defines their bounding boxes, another method needed to be implemented. How this was done is described in Section 3.1.

It should be stated that Figure 2.4 shows some particularly poor results which are not representative for all cases. It should also be said that if this is caused by a bug in CARLA, it might be resolved in upcoming releases.

Figure 2.4: Example of imprecise bounding boxes.

3 Method

This chapter describes the used method and implementation approach. In Figure 3.1 an overview of the implementation is illustrated using a flowchart.

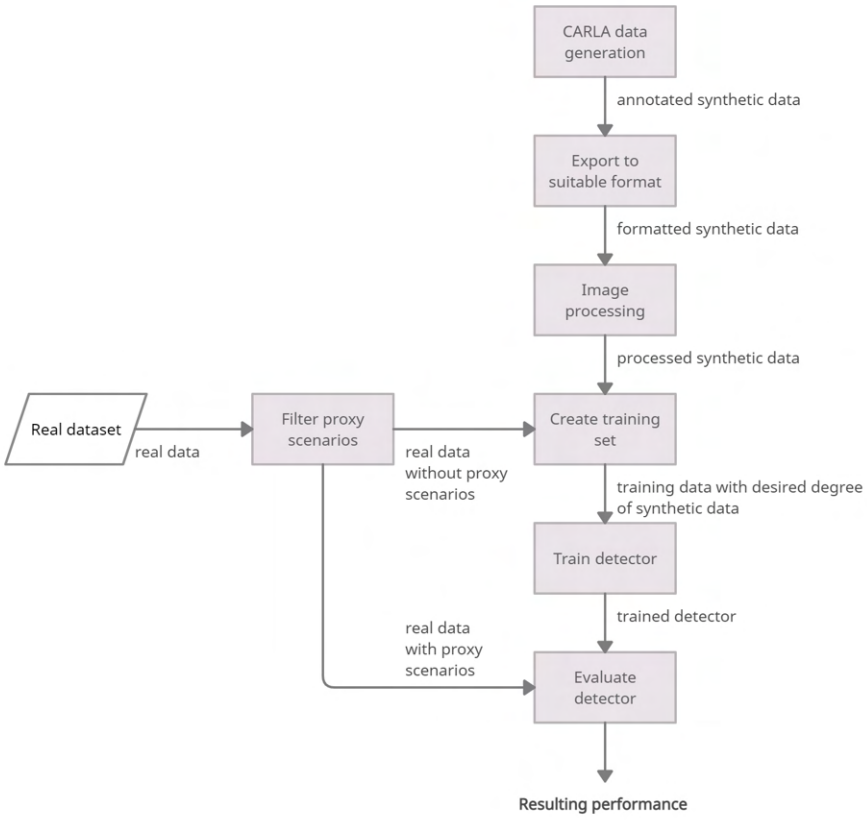


Figure 3.1: Flowchart describing the overall implementation.

As can be seen in the flowchart, the implementation approach starts with generating synthetic data using CARLA. This includes annotating each frame in a manner that is compatible with the ground truth of the real-world dataset, i.e. providing coordinates for the 2D bounding boxes of each pedestrian and an associated occlusion rate. In the next step, the domain gap is investigated and suitable image processing is applied. Finally, detectors are trained and their performances evaluated. The flowchart covers the main training and evaluation procedure. As will be described later in this chapter, other types of trainings were also performed in order to set a baseline.

More details about each step in the implementation chain are described in the following sections.

3.1 Generation of synthetic data using CARLA

The goal of the synthetic data generation was to produce datasets that include critical scenarios, i.e. potential vehicle-pedestrian collisions, which are hard to collect in the real world. For the data to be useful, it also needed to be carefully annotated and compatible with the, by Veoneer, provided real-world dataset and the implemented training procedure.

To generate the synthetic data, the autonomous driving simulator CARLA was used. As mentioned in Section 2.5, the existing API does not provide the feature of extracting 2D bounding boxes for visible pedestrians in an image. Therefore, a new procedure was implemented to serve this purpose. This procedure involves re-rendering each frame for every potentially visible pedestrian using a semantic segmentation camera (available in CARLA) and some masking techniques. The semantic segmentation camera makes it possible to identify each pixel classified as *pedestrian*. However, it does not differentiate between multiple instances of *pedestrian*. Thus, re-rendering was performed in order to handle situations when pedestrians occlude each other.

An overview of the data generation is shown in Figure 3.2 and the different parts are described in more detail below.

The different modules

In order to make the data generation as smooth as possible, the system was divided into three different modules, each with different responsibilities and purposes. Two of the modules, *WorldHandler* and *FrameHandler* were implemented from scratch as Python classes. The last module, *CarlaSyncMode*, was borrowed from CARLA Github [4] and slightly modified. The most important processes managed by the different modules are illustrated in Figure 3.2.

During a simulation, *CarlaSyncMode* serves as a context manager which synchronizes the outputs from different sensors. It is responsible for setting a fixed time-step, retrieving sensor data and ticking the world.

The *WorldHandler* class is responsible for managing the simulated world and all its actors. This includes loading the world map, setting the weather, spawning actors, enabling and disabling the environment and destroying the world at the end of the simulation. During a simulation, this class stores all useful information about the world and makes it easy to access whenever needed.

The *FrameHandler* class is responsible for handling the frames retrieved at each tick. This includes storing sensor data, processing the data, creating ground truth and saving everything to disk.

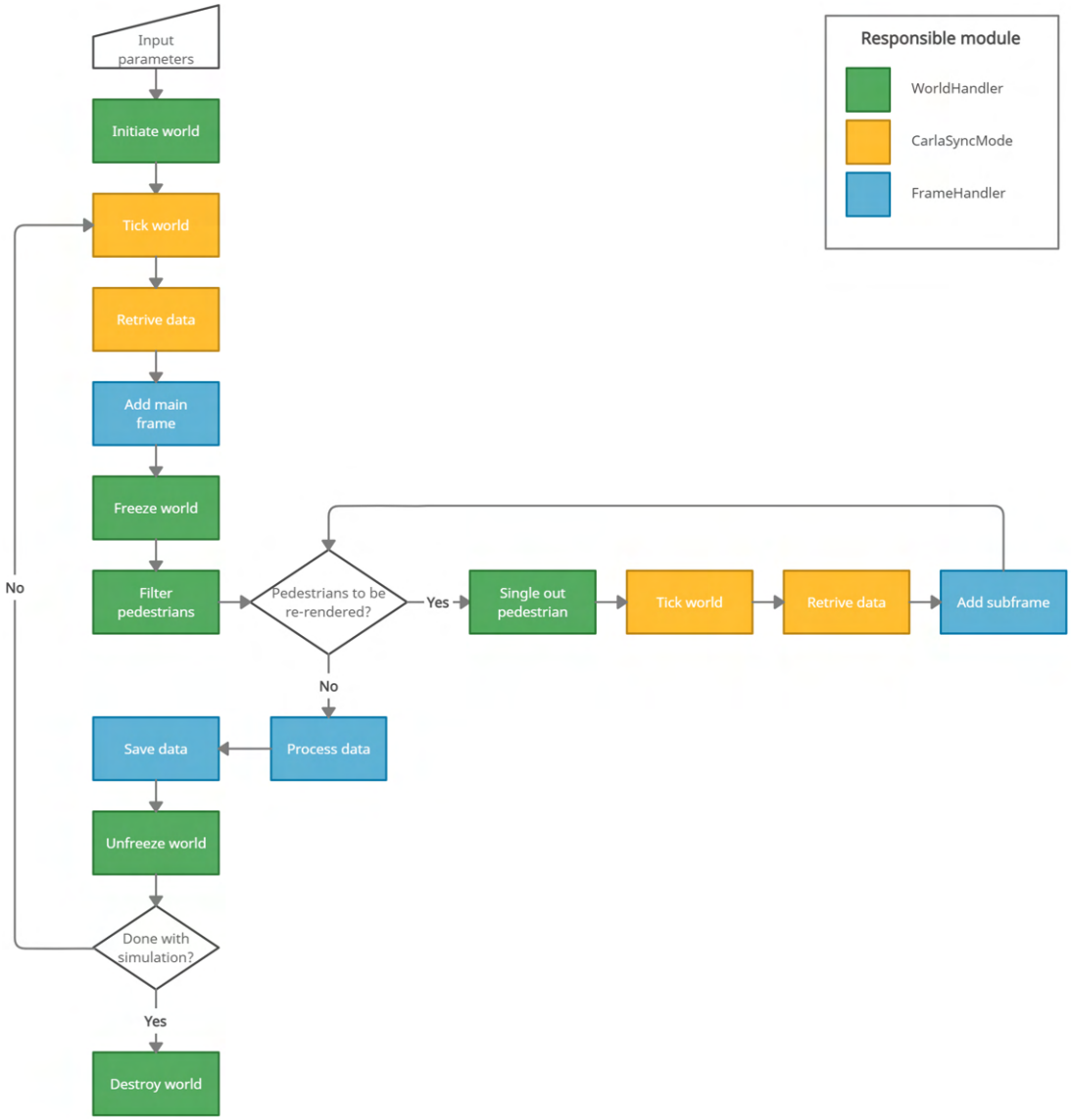


Figure 3.2: Flowchart describing the overall synthetic data generation. The processes are colored according to the responsible module.

Input parameters

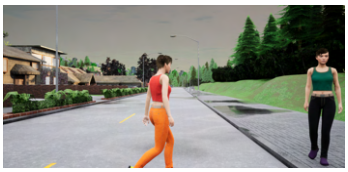
The data generation process was parameterized in order to provide flexibility and variation. The parameters that can be set and modified are shown in Table 3.1. They can roughly be divided into three categories; *World Environment*, *Camera Calibration* and *Pedestrian Visibility Settings*.

The parameters in the *World Environment* category have an impact on the appearance and properties of the world. There are also parameters that control the behavior of the spawned pedestrians. During the project, these were changed a lot in order to generate varying datasets. Some examples collected from different worlds with different weather conditions are shown in Figure 3.3.

It is also possible to set parameters that control the camera calibration. The extrinsic parameters, *cam_location* and *cam_rotation*, are defined relative to the ego-vehicle, i.e. the vehicle

to which the sensors are attached. During the project, both the extrinsic and intrinsic parameters were set to match the parameters of the camera used to collect the real-world data.

The parameters in the *Pedestrian Visibility Settings* category set the rules for when a pedestrian is considered visible.



(a) Town01, WetCloudySunset

(b) Town02, CloudyNoon

(c) Town10, ClearSunset

Figure 3.3: Examples from different worlds with different weather conditions.

Table 3.1: All input parameters

Parameter	Description
World Environment	
world	An available CARLA map.
weather	CARLA weather conditions.
number_of_vehicles	Maximum number of vehicles in the world.
number_of_walkers	Maximum number of pedestrians in the world.
walkers_running	Proportion of pedestrians running. Between 0.0 and 1.0.
walkers_crossing	Proportion of pedestrians randomly crossing roads. Between 0.0 and 1.0.
delta_sec	Fixed time-step of the simulation. Equal to the inverted frame rate.
Camera Calibration	
cam_location	Camera location relative to ego-vehicle.
cam_rotation	Camera rotation relative to ego-vehicle.
fov	Camera horizontal field of view in degrees.
Pedestrian Visibility Settings	
max_dist	Maximum distance in meters from the camera.
min_height	Minimum height in pixels.
occlusion_bounds	Upper bounds for the occlusion categories *none*, *light* and *heavy*.

Initiate and destroy the world

When the input parameters have been set and the script has been started, the first thing that happens is an initiation of the world. The specified world map is loaded and the weather is set, the desired number of pedestrians and vehicles are spawned and an RGB camera and a semantic segmentation camera are attached to the ego-vehicle. When all this is done, the data generation is ready to begin. When it is time to end the simulation, all the alive actors are destroyed. The script finishes and a new simulation may be started.

Tick the world and retrieve data

When the simulation should proceed forward in time, the *CarlaSyncMode* tick-function is called. At each tick, a new frame is produced and each sensor captures one image. This is done in a synchronous manner to make sure that each sensor delivers an image of the same scene.

Freeze and unfreeze the world

Since one sensor only produces one image per tick and the procedure for extracting 2D bounding boxes for visible pedestrians in an image involves re-rendering of each frame, a functionality to freeze the world needed to be implemented. This, to be able to make multiple ticks without changing the scene.

This was done by heavily decreasing the *delta_sec* making the world move extremely slowly (stopping it completely did not seem to be supported by CARLA) and freezing all movements of the pedestrians and vehicles. In this state, the world can tick and the sensors produce multiple images without changing the appearance of the scene. This provides the possibility to both capture an image of the whole scene and also single out and capture an image of each potentially visible pedestrian in the scene.

When the re-rendering is done the world is unfreezed, the *delta_sec* is restored to its original value and the simulation can proceed forward in time.

Filter and single out pedestrians

For the re-rendering part of the procedure, the pedestrians are filtered based on the horizontal field of view of the camera and the desired maximum distance from the camera. This is illustrated in Figure 3.4. The filtering step is done in order to rule out pedestrians that are clearly not visible to the camera and hence avoid unnecessary re-rendering.

The distance filtering is based on the Euclidean distance between the position of the pedestrian and the position of the camera. If this distance is smaller than the specified *max_dist* then the pedestrian is positioned within the distance of interest.

For the angle filtering, the specified *fov* is used. Firstly, the position of the pedestrian is transformed to the coordinate system of the camera. Secondly, the angle of the pedestrian is calculated relative to the camera. Finally, the angle of the pedestrian is compared to the specified *fov*. If the calculated angle is smaller than *fov*, it can be concluded that the pedestrian is within the angle of interest.

The required calculations for the distance and angle filtering are described by Equations 3.1 and 3.2, where x_p, y_p, z_p represent the position of the pedestrian while x_c, y_c, z_c represent the position of the camera.

$$\begin{aligned} &pedestrian\_dist = \sqrt{(x_p - x_c)^2 + (y_p - y_c)^2 + (z_p - z_c)^2} \\ &pedestrian\_dist < max\_dist \end{aligned} \tag{3.1}$$

$$\begin{aligned} &pedestrian\_angle = \arctan\left(\frac{y_p}{x_p}\right)\frac{180}{\pi} \\ &|pedestrian\_angle| < \frac{fov}{2} \end{aligned} \tag{3.2}$$

If a pedestrian fulfill both of the above stated conditions, it is marked as potentially visible and added to the queue for re-rendering. Before the re-rendering starts, everything except the current pedestrian should be removed from the scene. All environment objects in

the scene, such as terrain, buildings, roads and fences can be disabled by passing *False* to the CARLA function `enable_environment_objects()`. This does not disable the alive actors, so these are instead moved out of sight. An example of the re-rendering is displayed in Figure 3.5. When all of the filtered pedestrians have been re-rendered, the actors are moved back and the environment objects enabled.

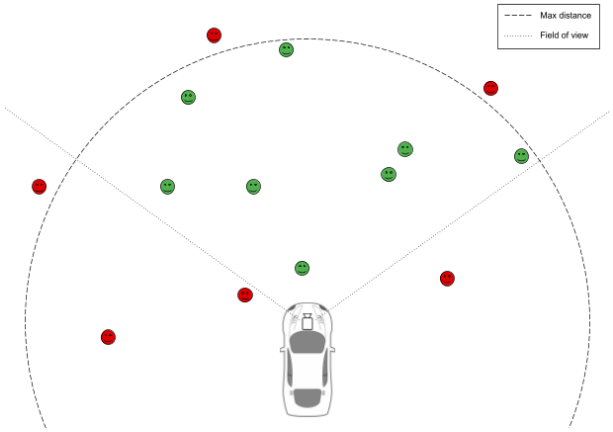


Figure 3.4: Filtering of pedestrians based on maximum distance and field of view. The green and red markings symbolize pedestrians inside and outside the area of interest.

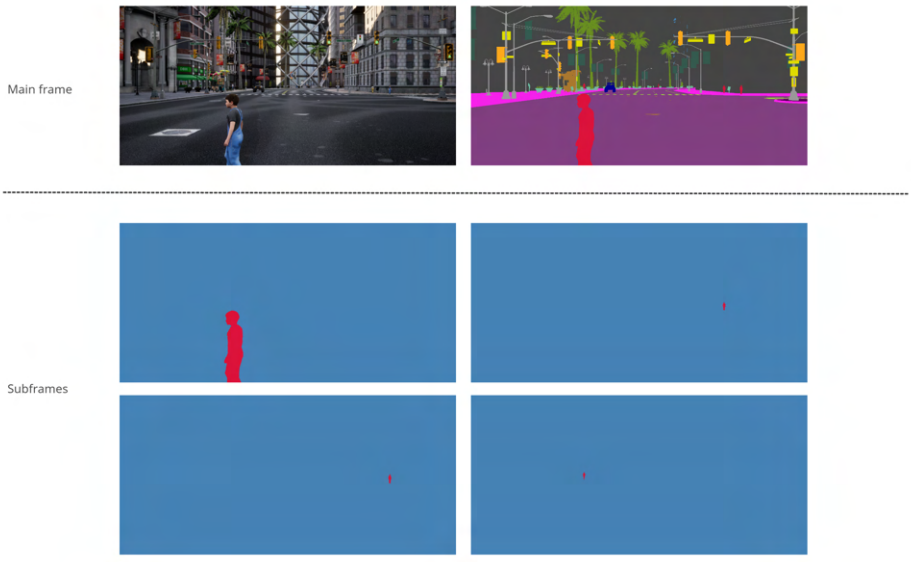


Figure 3.5: Re-rendering of all filtered pedestrians.

Process data

At the end of each iteration, when the main frame and all re-rendered subframes have been captured, it is time to process the data and extract the ground truth. This involves calculating the occlusion rate and bounding box for each pedestrian.

The method for calculating the occlusion rate is shown in Figure 3.6. The scheme follows the pedestrian named A. In the first step, the subframes retrieved from the re-rendering are used to create logical masks for pedestrian A and for the pedestrians located closer to the camera, in this case pedestrian B and C. Pedestrians further away from the camera are not

considered since they have no chance of occluding pedestrian A. The semantic segmentation image of the main frame is also used to get a logical mask of the environment. In the second step, a new mask that represents the occlusion caused by pedestrians closer to the camera is created. This is done by comparing the Boolean pixel values of mask A with mask B followed by mask C. If the value of a pixel matches, it is either an occluded pixel (if both are *True*) or a pixel belonging to the background (if both are *False*). This pixel is therefore set to *False*. If they do not match, the pixel is set to the value of mask A. In the third step, another mask is created. This mask builds upon the previous mask and takes the occlusion caused by the environment into account. This is done by simply using the logical AND operator on the previous mask and the environment mask. In the final step, the occlusion rate of pedestrian A is calculated by comparing the number of *True* pixels in the original versus final mask.

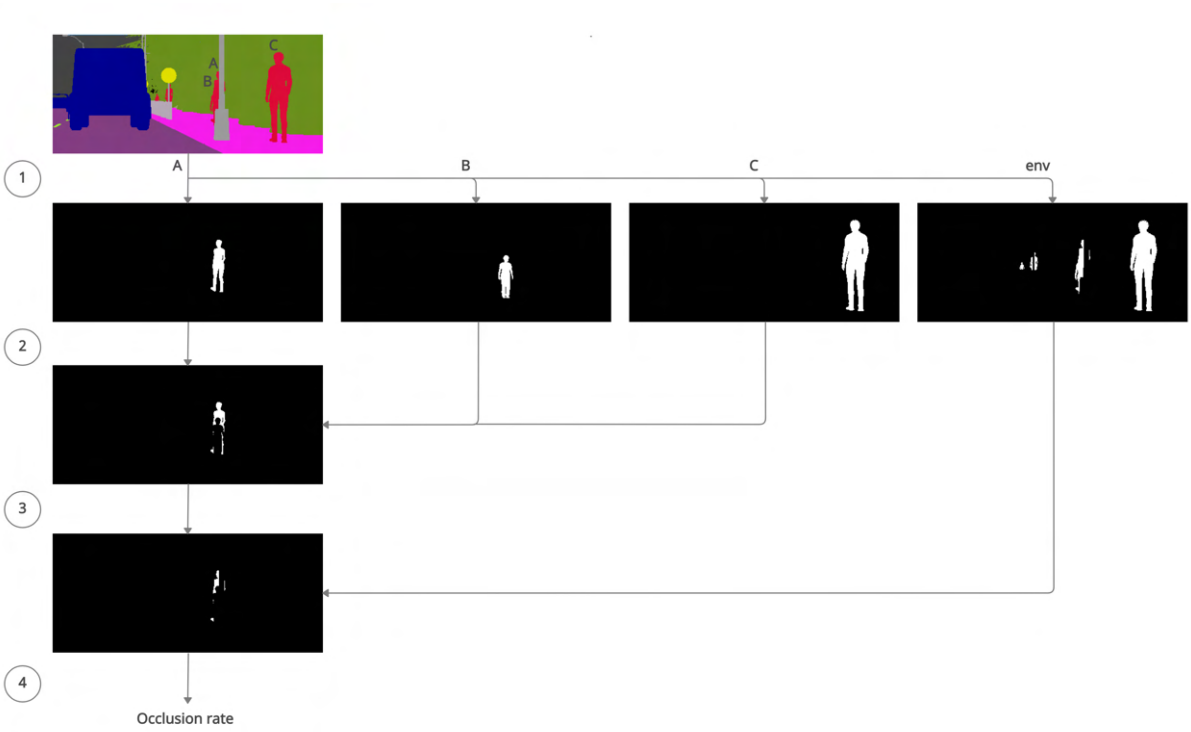


Figure 3.6: Illustration of how the occlusion rate is calculated for pedestrian A. 1, Creating logical masks for pedestrian A, pedestrians closer to the camera (i.e. B and C) and the environment (env). 2, Creating a mask for occlusion caused by pedestrians closer to the camera. 3, Creating a mask for occlusion caused by the environment. 4, Calculating occlusion rate by comparing the final mask with the original mask.

To calculate the bounding box of a pedestrian, the previously mentioned logical mask is used. The centroid of the shape is calculated along with the most upper and lower point. To match the ground truth of the real-world dataset, the bounding box is centered around the centroid while the ratio between the height and width is fixed, as shown in Figure 3.7.

(a)

(b)

Figure 3.7: Example showing how the bounding boxes (green) are centered around the centroid (orange), with a fixed ratio between width and height.

Save data

When the data is processed, the frame is saved to disk along with its annotations. The ground truth file contains information about all pedestrians in the scene. The information consists of the following:

- Center coordinates of the bounding box.
- Bounding box dimensions, i.e. width and height.
- Occlusion category, i.e. *none*, *light* or *heavy*.
- Pedestrian status, i.e. *visible* or *nondescript*.

The pedestrian status is decided by the occlusion category and the specified *min_height*. If a pedestrian belongs to one of the occlusion categories *none* or *light* and the height of its bounding box is bigger than *min_height* it is considered *visible*, otherwise *nondescript*. If a *nondescript* is detected during training, it will have no adverse impact on the result. A visualization of the ground truth is shown in Figure 3.8.

Figure 3.8: Visualization of the ground truth. A green box symbolizes no occlusion while a yellow box symbolizes light occlusion. A red box indicates that the pedestrian is classified as *nondescript*.

3.2 Domain adaptation

It is known that state-of-the-art pedestrian detectors perform well when both training and test data are drawn from the same distribution. However, in this study, the training set includes synthetic images which belong to a very different domain compared to the real-world images. There are considerable differences in lighting, image quality, object appearances etc. which are likely to cause a significant performance drop. In order to bridge this domain gap, some different methods were tested throughout the project.

Match color mean and standard deviation

By visually studying images from the synthetic dataset compared to images from the real-world dataset, it is clear that there are significant differences when it comes to color distribution. An intuitive first step towards bridging the gap was therefore to transform the synthetic images (source) to match the color mean and standard deviation of the real-world images (target) while retaining their individual content.

It is possible to change the mean and standard deviation of a source image by first standardize it to mean 0 and standard deviation 1 and then transform it to obtain the desired mean and standard deviation, in this case the mean and standard deviation of the target image. Given an image from the source domain $\mathbf{x}_s \in \mathbb{N}^{h_s \times w_s \times c}$ and an image from the target domain $\mathbf{x}_t \in \mathbb{N}^{h_t \times w_t \times c}$ with height h, width w and c channels (here $c = 3$ for an RGB space), this can be done separately for each color channel γ ($\gamma \in \{R, G, B\}$) according to

$$\mathbf{x}_{s \to t}^{\gamma} = \frac{(\mathbf{x}_s^{\gamma} - \overline{x}_s^{\gamma})}{\sigma_{x_s}^{\gamma}} \sigma_{x_t}^{\gamma} + \overline{x}_t^{\gamma}, \tag{3.3}$$

where $(\overline{x}_s^{\gamma}, \sigma_{x_s}^{\gamma})$ and $(\overline{x}_t^{\gamma}, \sigma_{x_t}^{\gamma})$ are the mean and standard deviation (per color channel) of the source and target image, respectively.

This method was used as a first step in the domain adaptation. But instead of matching the mean and standard deviation between single images, the average mean and standard deviation over the N samples in the synthetic training set and the M samples in the real-world training set were used. This was done according to

$$\mathbf{x}_{s \to t}^{\gamma} = \frac{(\mathbf{x}_s^{\gamma} - \overline{X}_s^{\gamma})}{\Sigma_{x_s}^{\gamma}} \Sigma_{x_t}^{\gamma} + \overline{X}_t^{\gamma}, \tag{3.4}$$

where $\overline{X}_s^{\gamma} = \frac{1}{N} \sum_N \overline{x}_s^{\gamma}$, $\Sigma_{x_s}^{\gamma} = \frac{1}{N} \sum_N \sigma_{x_s}^{\gamma}$, $\overline{X}_t^{\gamma} = \frac{1}{M} \sum_M \overline{x}_t^{\gamma}$ and $\Sigma_{x_t}^{\gamma} = \frac{1}{M} \sum_M \sigma_{x_t}^{\gamma}$.

Match color mean and covariance

As another experiment in domain adaptation, a method for matching the color mean and covariance between the two domains was tested. The procedure closely followed the one presented by Abramov, Bayer et al. [1], with the modification that instead of using the mean and covariance of single images, their respective average over both training sets were calculated.

The first step was to resize the images into what Abramov, Bayer et al. refers to as a feature matrix

$$\begin{aligned} \mathbf{x}_s \in \mathbb{N}^{h_s \times w_s \times c} &\to \mathbf{F}_s \in \mathbb{R}^{D_s \times c} \\ \mathbf{x}_t \in \mathbb{N}^{h_t \times w_t \times c} &\to \mathbf{F}_t \in \mathbb{R}^{D_t \times c}, \end{aligned} \tag{3.5}$$

where each row $\mathbf{f}^d \in \mathbf{F}$ represents one pixel and holds the corresponding color values. The second step was to calculate the average covariance and mean for the two dataset

$$\begin{aligned} \mathbf{\Sigma}_s &= \frac{1}{N} \sum_N \text{cov}(\mathbf{F}_s) \in \mathbb{R}^{c \times c} \\ \mathbf{\Sigma}_t &= \frac{1}{M} \sum_M \text{cov}(\mathbf{F}_t) \in \mathbb{R}^{c \times c} \\ \overline{\mathbf{F}}_s &= \frac{1}{N} \sum_N \mathbf{1}_{D_s}^T \overline{\mathbf{f}}_s \in \mathbb{R}^{D_s \times c} \\ \overline{\mathbf{F}}_t &= \frac{1}{M} \sum_M \mathbf{1}_{D_t}^T \overline{\mathbf{f}}_t \in \mathbb{R}^{D_t \times c}, \end{aligned} \tag{3.6}$$

where $\bar{\mathbf{f}} = \frac{1}{D}\sum_D \mathbf{f}^d \in \mathbb{R}^{1\times c}$. Note that the mean was reshaped from shape $1 \times c$ into shape $D \times c$ by duplicating rows. This was done in order to express the element-wise subtraction in the next step

$$\begin{aligned} \mathbf{F}_s^0 &= \mathbf{F}_s - \overline{\mathbf{F}}_s \\ \mathbf{F}_t^0 &= \mathbf{F}_t - \overline{\mathbf{F}}_t. \end{aligned} \tag{3.7}$$

Next, singular value decomposition (SVD) was performed on the covariance matrix of the source data in order to apply PCA-Whitening. Here, matrices $\mathbf{U}$ and $\mathbf{S}$ are used to rotate and scale the points

$$\begin{aligned} \mathbf{U}_s\mathbf{S}_s\mathbf{V}_s^* &= \mathrm{svd}(\mathbf{\Sigma}_s) \\ \hat{\mathbf{F}}_s^0 &= \mathbf{F}_s^0\mathbf{U}_s\mathbf{S}_s^{-\frac{1}{2}}. \end{aligned} \tag{3.8}$$

Then, the process was reversed by using the resulting matrices from the SVD on the covariance matrix of the target data

$$\begin{aligned} \mathbf{U}_t\mathbf{S}_t\mathbf{V}_t^* &= \mathrm{svd}(\mathbf{\Sigma}_t) \\ \mathbf{F}_{s\to t}^0 &= \hat{\mathbf{F}}_s^0\mathbf{S}_t^{\frac{1}{2}}(\mathbf{U}_t)^T. \end{aligned} \tag{3.9}$$

Finally, the resulting feature matrix was transformed to obtain the mean of the target data and reshaped back to the original image format

$$\begin{aligned} \mathbf{F}_{s\to t} &= \mathbf{F}_{s\to t}^0 + \overline{\mathbf{F}}_t \\ \mathbf{F}_{s\to t} &\to \mathbf{x}_{s\to t} \in \mathbb{N}^{h_s\times w_s\times c}. \end{aligned} \tag{3.10}$$

3.3 Data augmentation

Since the synthetic data generation builds upon finite worlds and a limited amount of features, it lacks variation compared to the real-world dataset. At this time, there are, for example, only 26 pedestrian blueprints and 8 predefined cities available in CARLA. Apart from being very few, many of the pedestrians are also quite similar when it comes to body features such as shape, height and haircut. When examining the two datasets it was also clear that the synthetic data contains less variation in camera positioning due to a limited amount of hills, lane changes etc. With these differences in mind, augmentation techniques in translation and stretch felt like interesting candidates to investigate.

To apply the augmentation techniques, the parameters t_x, t_y, s_x and s_y in the following transformation matrices were randomly drawn from predefined intervals.

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{S} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.11}$$

The two parameters in the translation matrix $\mathbf{T}$ control how much each pixel should move along the x- and y-axis. While the parameters in $\mathbf{S}$, control the stretching in each direction. Since both these augmentation techniques interact with the placement of pixels, the pedestrian bounding boxes needed to be transformed in the same manner. When an image is, for example, shifted upwards, it leaves an empty space at the bottom. Such gaps were filled by applying edge replication. Some examples when applying translation and stretch are displayed in Figure 3.9.

(a) No augmentation. (b) Vertical translation. (c) Horizontal translation. (d) Stretch.

Figure 3.9: Examples of translation and stretch-based augmentation.

With the aim of trying to find the best tuning, the translation and stretch-based methods were investigated sequentially and with gradually increasing parameter intervals. The considered intervals were the following

$$\begin{aligned} t_x &\in [0,\ 0.05iw], \qquad i = 1, 2, 4, 8, 10, 16 \\ t_y &\in [0,\ 0.05jh], \qquad j = 1, 2, 4, 8, 10, 16 \\ s_x,\ s_y &\in [\frac{1}{k},\ k], \qquad k = 1.1, 1.2, 1.45, 1.95 \end{aligned} \tag{3.12}$$

where w is the image width and h the image height. These intervals were chosen in order to cover modest augmentation as well as more extreme cases. The tests were performed by training networks on synthetic data with various degrees of augmentation and evaluate their performance when tested on real-world data. The performances were ranked by comparing achieved FP/frame for a fixed target TP-rate.

3.4 Training and evaluation of detectors

As stated in the introductory chapter, the main goal of the thesis is to investigate to what extent synthetic data can replace real-world data for the specified critical scenarios. Since only a limited amount of these scenarios are available in the real-world data, the approach was instead to study proxy scenarios, e.g. situations when pedestrians are located closely in front of the vehicle (for example at a crosswalk). The idea was to train a detector on real-world data where all samples of these proxy scenarios have been removed and compare it to other detectors trained on data where the removed samples have been replaced with various degrees of synthetic data. All detectors were then to be evaluated on the previously removed proxy scenarios.

Definition and filtering of proxy scenarios

A proxy scenario should as closely as possible resemble a critical scenario. Since a critical scenario, in this study, is defined as a potential vehicle-pedestrian accident, a proxy scenario should include a least one pedestrian situated closely in front of the vehicle. It is therefore reasonable that the definition of a proxy scenario is based on the distance and angle of the pedestrians relative to the vehicle. Since this information is not directly available in the real-world dataset, the proxy-filtering was instead based on the height and image centering of the pedestrians.

The pixel height of an average adult at the desired distance from the camera was estimated. This pixel height was then used, together with desired centering limits, to divide the real-world dataset into proxy and no-proxy scenarios. Some different values for height and centering limits were tested with the aim of finding a reasonable definition. Reasonable in this case means that the definition is strict enough to only cover what can be considered critical scenarios but loose enough that there is a clear performance drop when a detector trained

on real-world data without proxy scenarios is tested on proxy scenarios. This, to make it easier to register potential improvements when synthetic data is added.

First, a height corresponding to approximately 30 meters in distance and centering limits that excluded one-sixth on both sides of the image was tested. This resulted in a very small performance drop. This could possibly be explained by the fact that some pedestrians at a critical distance still were present in the training data. So, even though all centered pedestrians were removed, the presence of pedestrians close to the edges seemed enough for the detector to perform well. The next step was therefore to expand the centering limits to cover the whole width of the image and instead focus only on the distance. This caused a clear performance drop but resulted in a bit of an unbalanced ratio between the amount of proxy and no-proxy scenarios. The amount of non-critical pedestrians should, intuitively, outweigh the amount of critical ones. This was achieved by decreasing the critical distance to 20 meters, which also became the final definition used for all upcoming trainings.

The training and evaluation procedure

In order to get an understanding of how well a detector performs on critical (proxy) scenarios when trained on data where these scenarios have been replaced with different degrees of synthetic data, a baseline needed to be set. For this purpose, 5 different datasets were created. They are all listed in Table 3.2.

Table 3.2: An overview of the different datasets

Dataset	Frames	Description
Real	407000	Only real-world data
RealNoProxy	316300	Real-world data without proxy scenarios
RealProxy	90700	Real-world data with only proxy scenarios
Synthetic	134000	Only Synthetic data
Combined	450300	RealNoProxy + Synthetic

The real-world data was provided by Veoneer and covers a wide range of traffic scenarios. The samples are collected from different parts of the world and cover different weather conditions and traffic environments, such as highways, country roads and city streets. All photos were taken with the same camera during daytime.

The idea was that for every experiment (with and without domain adaptation and data augmentation), trainings should be performed on *Real*, *RealNoProxy*, *Synthetic* and *Combined*. Each of these networks should then be evaluated on *RealNoProxy*, *RealProxy* and *Synthetic*. The *Real* and *Synthetic* trainings were meant to provide an understanding of how well the detector performs in its original state, as a regular model for pedestrian detection in the different domains. They were also used to get an understanding of the domain gap by testing on data from the unseen domain. The *RealNoProxy* training was mainly responsible for setting a baseline for the *Combined* training. By comparing their performances on *RealProxy*, conclusions could be drawn about whether or not the synthetic data assists in detecting critical pedestrians.

During each training, 80% of the available data is used. The remaining 20% is used for evaluation. To be able to compare the performance of networks from different trainings, each detector is tuned to achieve the same TP-rate on the common dataset, i.e. *RealNoProxy*. A TP-rate of 80% was chosen to obtain a good balance between TP and FP performance. For the training with purely synthetic data, it is not possible to reach a TP-rate of 80% on *RealNoProxy* for all types. Instead, this detector is tuned to achieve a TP-rate of 95% on *Synthetic*. Since the synthetic data is less complex than the real-world data the *Synthetic* training is expected to perform very well when tested on data from its own distribution, hence the higher TP-tuning.

The performance measures presented in Chapter 4, show the performance after non-maximum suppression (with an IOU-threshold of 0.3), while the tuning is done directly on the output of the network. This will result in a TP-rate on the *RealNoProxy* dataset slightly lower than the 80% tuning.

4 Results

This chapter presents all results obtained during the project. This includes the raw results (the results obtained when training on unmodified data) and the results from using the different domain adaptation and data augmentation techniques.

4.1 Raw results

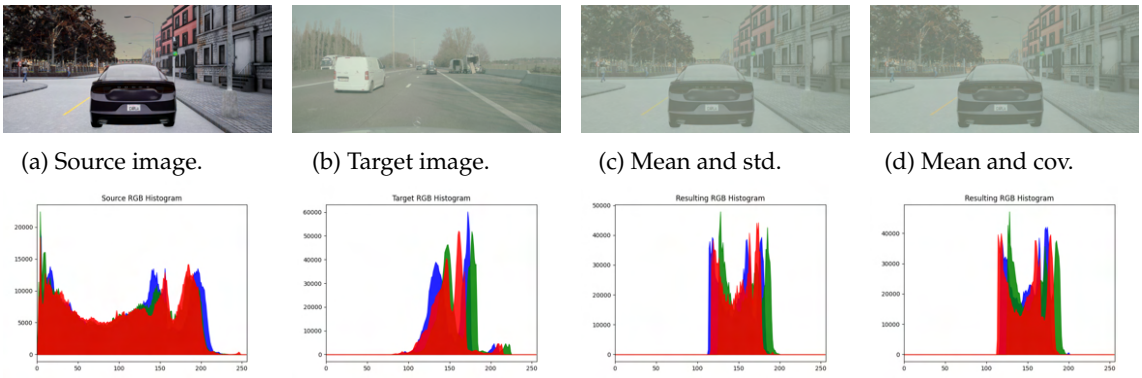
The results of the first set of trainings, which were done on unmodified data, are displayed in Table 4.1.

Table 4.1: Raw results. Each row represents a training with a specific dataset. Each column shows the performance when evaluating on the different datasets. Each cell contains the resulting mean (top) and the standard deviation (bottom, inside parenthesis). These values are based on the results of three separate runs.

	RealNoProxy			**RealProxy**			**Synthetic**		
	TP-rate (%)	FP/frame	F1-score	TP-rate (%)	FP/frame	F1-score	TP-rate (%)	FP/frame	F1-score
Real	78.6 (0.123)	0.0785 (0.00647)	0.844 (0.00325)	85.9 (1.06)	0.103 (0.00936)	0.897 (0.00755)	8.20 (2.60)	0.0355 (0.00418)	0.149 (0.0436)
RealNoProxy	78.5 (0.0170)	0.0943 (0.00481)	0.836 (0.00214)	47.4 (1.80)	0.162 (0.0211)	0.607 (0.0200)	5.88 (3.27)	0.0677 (0.0503)	0.106 (0.0566)
Synthetic	0.0380 (0.0378)	2.47e-4 (2.04e-4)	7.60e-4 (7.56e-4)	0.00214 (0.00302)	5.36e-4 (5.91e-4)	4.27e-5 (6.04e-5)	94.3 (0)	0.0348 (0.00404)	0.964 (8.30e-4)
Combined	78.8 (0.00943)	0.0644 (0.00821)	0.851 (0.00368)	49.3 (0.849)	0.120 (0.00960)	0.633 (0.00706)	90.4 (0.423)	0.0470 (0.00345)	0.940 (0.00174)

4.2 Domain adaptation results

The images displayed in Figure 4.1 are produced by following the procedure described in Section 3.2. Here, a source image is transformed to obtain the color mean and standard deviation (4.1c) followed by the color mean and covariance (4.1d) of a single target image. The figure also shows how the RGB histogram of the source image changes when applying the two different domain adaptation techniques.



(a) Source image. (b) Target image. (c) Mean and std. (d) Mean and cov.

Figure 4.1: The first row displays an example of how a synthetic image looks like before (a) and after matching color mean and standard deviation (c) and after matching color mean and covariance (d) of a real-world image (b). The second row displays the associated RGB histograms.

Furthermore, the average mean, standard deviation and covariance over both the real-world and the synthetic training set were calculated. The resulting statistics are displayed in Table 4.2 and Equation 4.1. Some extreme samples, based on these values, are shown in Figure 4.2.

Table 4.2: Mean and standard deviation of the real-world and synthetic dataset with pixel values between 0 and 255

	mean				**std**			
Dataset	total avg	max	min	rgb avg	total avg	max	min	rgb avg
real-world	134.7	213.8	26.43	129.6, 140.3, 128.9	29.89	56.92	5.940	28.72, 29.27, 29.90
synthetic	123.0	175.8	46.91	127.8, 122.8, 118.6	66.65	98.74	27.49	67.82, 65.99, 66.14

$$\boldsymbol{\Sigma}_t = \begin{pmatrix} 865.6 & 870.6 & 879.1 \\ 870.6 & 895.5 & 908.5 \\ 879.1 & 908.5 & 934.9 \end{pmatrix} \qquad \boldsymbol{\Sigma}_s = \begin{pmatrix} 4626.1 & 4387.9 & 4218.1 \\ 4387.9 & 4411.4 & 4280.8 \\ 4218.1 & 4280.8 & 4393.7 \end{pmatrix} \tag{4.1}$$

Figure 4.2: Extreme samples from the datasets. The first row contains extreme samples from the real-world dataset (from left: max mean, min mean, max std, min std). The second row contains the same type of samples from the synthetic dataset.

Using the calculated statistics and following the procedure described by Equations 3.4 and 3.5-3.10, resulted in the images displayed in Figure 4.3. The resulting performance using the domain adaptation method based on color mean and standard deviation is presented in Table 4.3. The results when applying matching of color mean and covariance are displayed in Table 4.4. Furthermore, some examples of predictions before and after domain adaptation are visualized in Figures 4.4 and 4.5.

(a) Source image. (b) Match total image mean and std. (c) Match per color mean and std. (d) Match per color mean and cov.

Figure 4.3: An example of color mean and standard deviation respective covariance matching using the average over the whole synthetic and real-world dataset.

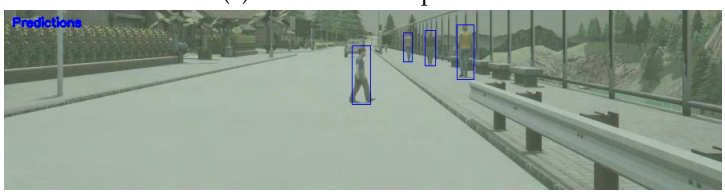
Table 4.3: Results for domain adaptation based on color mean and standard deviation. Each row represents a training with a specific dataset. Each column shows the performance when evaluating on the different datasets. Each cell contains the resulting mean (top) and the standard deviation (bottom, inside parenthesis). These values are based on the results of three separate runs. The cells marked in *cursive* are not affected by the domain adaptation since they do not include any synthetic data.

	RealNoProxy			**RealProxy**			**Synthetic**		
	TP-rate (%)	FP/frame	F1-score	TP-rate (%)	FP/frame	F1-score	TP-rate (%)	FP/frame	F1-score
Real	*78.6* *(0.123)*	*0.0785* *(0.00647)*	*0.844* *(0.00325)*	*85.9* *(1.06)*	*0.103* *(0.00936)*	*0.897* *(0.00755)*	57.8 (1.55)	0.0549 (0.0134)	0.722 (0.00990)
RealNoProxy	*78.5* *(0.0170)*	*0.0943* *(0.00481)*	*0.836* *(0.00214)*	*47.4* *(1.80)*	*0.162* *(0.0211)*	*0.607* *(0.0200)*	53.6 (2.06)	0.0820 (0.0156)	0.683 (0.0163)
Synthetic	25.1 (0.716)	0.115 (0.01940)	0.368 (0.00904)	22.4 (1.27)	0.0760 (0.0174)	0.354 (0.0139)	94.3 (0.0602)	0.0450 (0.00859)	0.961 (0.00202)
Combined	78.8 (0.0653)	0.0627 (0.00687)	0.852 (0.00352)	55.5 (2.87)	0.0638 (0.0123)	0.698 (0.0262)	89.3 (0.681)	0.0549 (0.0108)	0.932 (0.00547)

Table 4.4: Results for domain adaptation based on color mean and covariance.

	RealNoProxy			**RealProxy**			**Synthetic**		
	TP-rate (%)	FP/frame	F1-score	TP-rate (%)	FP/frame	F1-score	TP-rate (%)	FP/frame	F1-score
Real	*78.6* *(0.123)*	*0.0785* *(0.00647)*	*0.844* *(0.00325)*	*85.9* *(1.06)*	*0.103* *(0.00936)*	*0.897* *(0.00755)*	63.1 (1.57)	0.0478 (0.0108)	0.764 (0.00955)
RealNoProxy	*78.5* *(0.0170)*	*0.0943* *(0.00481)*	*0.836* *(0.00214)*	*47.4* *(1.80)*	*0.162* *(0.0211)*	*0.607* *(0.0200)*	59.6 (1.70)	0.0701 (0.00550)	0.733 (0.0134)
Synthetic	33.2 (3.25)	0.154 (0.0186)	0.447 (0.0303)	29.3 (0.784)	0.0855 (0.00471)	0.438 (0.00878)	94.3 (0.0497)	0.0513 (0.0203)	0.960 (0.00441)
Combined	78.8 (0.103)	0.0621 (0.00377)	0.852 (0.00194)	56.8 (1.95)	0.0769 (0.0339)	0.706 (0.0205)	89.7 (0.642)	0.0528 (0.0117)	0.935 (0.00236)

(a) No domain adaptation.



(b) Domain adaptation using color mean and covariance.

Figure 4.4: Examples of predictions for a detector trained on real-world data and tested on synthetic data, with and without domain adaptation.

(a) No domain adaptation.

(b) Domain adaptation using color mean and covariance.

Figure 4.5: Examples of predictions for a detector trained on synthetic data and tested on real-world data, with and without domain adaptation.

4.3 Data augmentation results

As explained in Section 3.3, a parameter search was performed with the aim of finding a suitable tuning for the translation and stretch parameters. During the search, the domain adaptation technique using color mean and covariance was applied. Section 3.3 also mentions that the trainings were done on purely synthetic data. The reason for not training directly on *Combined*, was mainly to save time. *Combined* contains much more samples and therefore requires a longer training time.

Table 4.5 displays the resulting FP/frame for a common TP-rate of 60% on *RealProxy*. For a network trained on *Synthetic* and tested on *RealProxy*, a TP-rate of 60% is quite challenging, hence the generally high FP/frame.

Table 4.5: Parameter search for data augmentation based on translation. The rows and columns represent different values of j and i in Equation 3.12. The cell in the upper left corner equals no augmentation and the vertical translation interval is gradually expanded moving downwards, while the horizontal translation interval is expanded moving right.

	0	1	2	4	8	10	16
0	16.6	7.38	15.6	5.22	4.14		
1	**1.04**	3.61					
2	2.26		3.97				
4	4.71			4.1			
8	2.14			3.28	1.99		
10	1.86			4.88		3.12	
16	2.72				**1.14**		2.36

As will be discussed in the next chapter, the results are very hard to make sense of. Yet, for the sake of testing, the two best versions were chosen for the next step; adding stretch. The results for type A (j=1, i=0) and type B (j=16, i=8) are shown in Table 4.6. Finally, the best tuning was used in a training on *Combined*, shown in Table 4.7.

Table 4.6: Extended parameter search based on stretch. The columns represent different values for k in Equation 3.12.

	1.1	1.2	1.45	1.9
A	1.72	3.93	8.29	9.32
B	1.19	1.12	1.82	**0.729**

Table 4.7: Results obtained by training on *Combined* and testing on *RealProxy* using the final augmentation tuning

	RealProxy		
	TP-rate (%)	FP/frame	F1-score
Combined	47.6 (3.72)	0.0761 (0.00304)	0.627 (0.0330)

5 Discussion

This chapter discusses the obtained results along with the advantages and disadvantages of the chosen method and implementation approach. Finally, some thoughts and suggestions on potential future work are presented.

5.1 Results

This section analyzes the obtained results and highlights the differences in performance when moving from raw data to domain adapted and augmented data.

Raw results

When studying the results from the first trainings, on unmodified data, one thing that can be noted is the clear performance drop between the trainings *Real* and *RealNoProxy* when tested on *RealProxy*. *RealNoProxy* has both lower TP-rate and higher FP/frame. It also has a lower F1-score. This is expected since a detector can not perform well on unseen types of scenarios. So, this is an indication that the proxy scenario definition is reasonable. Both networks are, on the other hand, expected to yield similar results when tested on *RealNoProxy*, since these scenarios are included in both trainings. This is confirmed by studying the table. The *Real* training has a bit lower FP/frame which probably can be explained by the fact that this network has been trained on more data. What can also be noted when studying the results from the trainings on real-world data, is the poor performances on synthetic data due to the domain shift.

When studying the results from the *Synthetic* training, the domain shift is also very distinct. The detector can not seem to find any pedestrians from the real-world data but performs very well on synthetic data. Since the domain gap is so big, adding synthetic data to the *RealNoProxy* dataset is not having any significant impact when tested on *RealProxy*.

Domain adaptation results

When investigating the statistical properties of the synthetic and the real-world dataset, it is clear that there are distinct differences. When comparing one synthetic image to one randomly picked real-world image, as in Figure 4.1, it is suggested that the two datasets are very

different when it comes to color distribution. The synthetic image covers a wider range of color intensities, while the real-world image has less color variation and dominance of green. This is also confirmed by Table 4.2 and Equation 4.1, which contain the average statistics over both datasets. The camera used for the real-world data has a high dynamic range to be able to cover a wide range of light conditions without saturating the image. This typically results in only a small part of the image range being used for an individual image.

Figure 4.1 displays how the color distribution of the synthetic image changes when applying matching of color mean and standard deviation along with color mean and covariance. The resulting images and the corresponding histograms show that the synthetic image becomes visually and statistically closer to the real-world image, without losing its individual content. This still holds when using the average statistics over the whole datasets, as shown in Figure 4.3.

All this seemed promising, in terms of trying to bridge the domain gap. Based on the resulting performances presented in Tables 4.3 and 4.4 and the visualized predictions in Figures 4.4 and 4.5, these simple techniques are proven to have a distinct impact on the training results. Detectors trained on real-world data are now much better at detecting synthetic data. Even more importantly, for this thesis, the same effect is obtained when detectors trained on synthetic data are tested on real-world data. For example, the achieved F1-score on *RealProxy* went from approximately 0 to 0.354 (using domain adaptation based on color mean and standard deviation) and 0.438 (using domain adaptation based on color mean and covariance). When studying the, for this thesis, crucial results, i.e. those obtained by training on *Combined* and testing on *RealProxy*, it is clear that improvements have been made. By using the domain adaptation technique based on color mean and standard deviation, the TP-rate increased from 49.3% to 55.5% and FP/frame decreased from 0.120 to 0.0638. Also, the F1-score increased from 0.633 to 0.698. The technique based on color mean and covariance yielded similar results with a TP-rate of 56.8%, FP/frame of 0.0769 and F1-score of 0.706. Although the results are not nearly good enough for a detector to be considered well equipped for dealing with critical scenarios, it is still clear that the synthetic data, in this domain adapted state, has a positive impact.

Data augmentation results

The results from the experiment with translation and stretch-based augmentation were quite hard to interpret. The result of the translation parameter search, displayed in Table 4.5, shows that no translation leads to the highest FP/frame. Since adding translation lowers the FP/frame for all cases, conclusions could be drawn that this type of augmentation contributes to some kind of improvement. Also, it could possibly be argued that horizontal translation has a greater effect than vertical. However, it is not clear which combination of parameter values is preferred. There is no clear pattern and the variations seem random. For example, the two best results are obtained by essentially different degrees of augmentation.

Although the results from the search for suitable translation parameters seemed unreliable, it still felt interesting to continue with the experiment to see where it would lead. By adding stretch to the two best translation candidates, it resulted in the FP performances listed in Table 4.6. Again, the values are quite confusing and contradictory. The lowest FP/frame was obtained by adding the most extreme case of stretch to the already extreme translation. However, adding much stretch to modest translation seems to worsen the result. It could possibly be concluded that a high degree of stretch needs to be combined with a high degree of translation in order to have a positive impact.

To finalize the augmentation experiment, the best tuning was applied to the synthetic images in the *Combined* dataset. Table 4.7 shows the performance when the trained network is evaluated on *RealProxy*. When comparing this result to the corresponding result in Table 4.4, it is clear that this augmentation tuning worsens rather than improves the performance. In fact, the performance gain achieved by the domain adaptation is now completely reduced.

Even if this augmentation experiment did not result in any improvements, it does not necessarily mean that translation and/or stretch-based augmentation is useless for the intended purpose. Rather, it questions the method used to find suitable parameters.

5.2 Method

When reviewing the advantages and disadvantages of the method and implementation approach there are definitely a few things worth discussing.

The generation of synthetic data

Using the CARLA simulator for generating the synthetic data might not have been the most appropriate choice for this thesis. Despite having many useful functionalities, CARLA does not seem to be fully adapted for pedestrian detection. As mentioned in Chapter 2, the existing Python API does not provide the functionality to directly extract pedestrian 2D bounding boxes from the simulation. This was a bit unexpected and caused some delays in the time plan since a new method needed to be implemented from scratch. The lost time could otherwise have been used for more extensive and sophisticated experiments in domain adaptation and data augmentation. An alternative approach could have been to go beyond the provided API and instead make changes in the underlying source code. In retrospect, this might have been more time-efficient.

Another aspect of CARLA worth discussing is the lack of realism and the limited variation of features. When looking at the produced images it is very clear that they are far from photo-realism and there are, for example, only 26 pedestrian blueprints and 8 relatively small worlds to choose from. The realism and feature variation are limiting factors in all simulators but it could have been worth investigating if some other simulator provides a more realistic rendering and/or more features to choose from. This, to reduce the initial domain gap.

The domain adaptation

There are both advantages and disadvantages with the used domain adaptation techniques. What speaks in their favor, is their simple and relatable nature. It is interesting to see that these simple techniques can have a clear impact on the results. It is also convenient to be able to pinpoint exactly which changes in image characteristics result in the increased performance. When learning more complex patterns using, for example deep learning, it is not as clear what these patterns actually are. However, when studying the results it is clear that the detector does not reach an acceptable performance level, especially if it is to be trusted in dangerous traffic scenarios.

It is also worth commenting on the choice of adapting images based on statistics calculated over the entire datasets. Since all real-world samples are collected with the same camera during daytime, this seemed like a reasonable thing to do. However, if the dataset was to be expanded to also include, for example, night sequences and images taken with other cameras, it could be wise to make distinctions. Such images would likely have different statistical properties and therefore benefit from being handled separately.

The data augmentation

The method for choosing the augmentation parameters could definitely be questioned. In retrospect, basing the parameter search on trainings with purely synthetic data does not seem like a suitable approach. It may be that a lot of augmentation benefits the training with purely synthetic data because it is very limited, but when adding real-world data it rather becomes a disturbance. Also, the choice to perform the parameter search sequentially rather than in

parallel could be debated. It saves time but it is not obvious which parameter to start with and it does not give a clear picture of their individual impact on performance.

5.3 Future work

Since this thesis only touches upon two augmentation techniques, translation and stretch, it could definitely be interesting to further explore this field. There might be improvements to be made by, for example, adding noise and manipulating brightness and contrast.

In the spirit of boosting variation, it could also be interesting to experiment with combining synthetic pedestrians and real-world backgrounds. By taking advantage of the implemented re-rendering process in the data generation pipeline, synthetic pedestrians could be masked out and then pasted into real-world images. This could potentially address the problem with limited content in the synthetic environment.

6 Conclusion

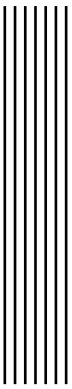
The aim of this thesis was to generate synthetic data and investigate to what extent this data can replace real-world data for critical traffic scenarios involving pedestrians. During the course of the project, some research questions were investigated. The first question was about analyzing the domain gap between the synthetic and real-world dataset and how it affects the detector's ability to detect pedestrians. The second question was about investigating the possibility to smooth the domain gap by aligning statistical properties. Finally, the last question was about investigating if the detector's performance could be improved by applying specific augmentation techniques.

What can be concluded is that the initial domain gap between the real-world dataset and the generated synthetic dataset is vast. There are considerable differences in image characteristics, such as color mean, standard deviation and covariance. The synthetic data also lacks feature coverage in relation to the real-world dataset. This substantial domain shift causes a detector to perform poorly when tested on data from the unseen domain. Especially, a detector trained on purely synthetic data does not seem to detect any pedestrians from the real-world dataset. Consequentially, it can be concluded that unmodified synthetic data can not replace real-world data for critical scenarios.

However, when applying simple domain adaptation techniques involving alignment of image statistics, clear improvements were made. A detector trained on the domain-adapted synthetic data is able to find pedestrians from the real-world domain. By removing the critical (proxy) scenarios from the real-world dataset and studying the performance on these scenarios before and after the adding synthetic data, it is clear that the synthetic data aids in detecting critical pedestrians. Adding synthetic data increases the TP-rate as well as decreases the FP/frame. It also increases the F1-score. Although this gain does not reach a performance level that matches real-world data, it is definitely a step in the right direction.

Since the synthetic data lacks diversity, tests were made to see if specific augmentation techniques could further boost the performance. Based on clear differences in, for example, camera positioning and body shapes of the pedestrians, an augmentation experiment with translation and stretch was performed. Unfortunately, this experiment did not result in any further improvements. Also, the experiment itself could probably have been carried out in a more suitable manner. So, even if this particular experiment failed to boost the performance, there may still be improvements to be made with a different parameter tuning or, for that matter, other augmentation techniques.

To sum up, this thesis indicates that synthetic data can help detect pedestrians in critical traffic situations. But to reach its full potential, more diverse and realistic synthetic data and/or more sophisticated methods for handling the domain shift are probably needed.



Bibliography

[1] Alexey Abramov, Christopher Bayer, and Claudio Heller. "Keep it Simple: Image Statistics Matching for Domain Adaptation". In: *CoRR* abs/2005.12551 (2020). arXiv: 2005.12551. URL: https://arxiv.org/abs/2005.12551.

[2] Mukhlas Adib. *CARLA Vehicle 2D Bounding Box Annotation Module*. GitHub, 2020. URL: https://mukhlasadib.github.io/CARLA-2DBBox/.

[3] Yangdong Deng, Yufei Ni, Zonghui Li, Shuai Mu, and Wenjun Zhang. "Toward Real-Time Ray Tracing: A Survey on Hardware Acceleration and Microarchitecture Techniques". In: *ACM Computing Surveys* 50 (2017). DOI: 10.1145/3104067.

[4] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. *CARLA Simulator*. Version 0.9.11. GitHub, 2020. URL: https://github.com/carla-simulator/carla.

[5] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[6] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. *Domain-Adversarial Training of Neural Networks*. 2016. arXiv: 1505.07818 [stat.ML].

[7] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 6.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 9.

[11] Frauke Günther and Stefan Fritsch. "neuralnet: Training of Neural Networks". In: *R Journal* 2 (June 2010). DOI: 10.32614/RJ-2010-006.

[12] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A. Efros, and Trevor Darrell. *CyCADA: Cycle-Consistent Adversarial Domain Adaptation*. 2017. arXiv: 1711.03213 [cs.CV].

[13] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*. 2016. arXiv: 1602.07360 [cs.CV].

[14] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. *Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?* 2017. arXiv: 1610.01983 [cs.CV].

[15] Wouter M. Kouw. "An introduction to domain adaptation and transfer learning". In: *CoRR* abs/1812.11806 (2018). arXiv: 1812.11806. URL: http://arxiv.org/abs/1812.11806.

[16] Wouter M. Kouw and Marco Loog. "A review of single-source unsupervised domain adaptation". In: *CoRR* abs/1901.05335 (2019). arXiv: 1901.05335. URL: http://arxiv.org/abs/1901.05335.

[17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *CoRR* abs/1607.02533 (2016). arXiv: 1607.02533. URL: http://arxiv.org/abs/1607.02533.

[18] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. *Learning Transferable Features with Deep Adaptation Networks*. 2015. arXiv: 1502.02791 [cs.LG].

[19] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[20] World Heath Organization. *Road traffic injuries*. 2020. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries (visited on 01/24/2021).

[21] Rafael Padilla, Sergio Netto, and Eduardo da Silva. *A Survey on Performance Metrics for Object-Detection Algorithms*. July 2020. DOI: 10.1109/IWSSIP48289.2020.

[22] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[23] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[24] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[25] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV].

[26] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[27] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. DOI: 10.1109/CVPR.2016.352.

[28] Connor Shorten and Taghi Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6 (July 2019). DOI: 10.1186/s40537-019-0197-0.

[29] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. *Learning from Simulated and Unsupervised Images through Adversarial Training*. 2017. arXiv: 1612.07828 [cs.CV].

[30] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. 2017. arXiv: 1703.06907 [cs.RO].

[31] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. *Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization*. 2018. arXiv: 1804.06516 [cs.CV].

[32] Apostolia Tsirikoglou, Gabriel Eilertsen, and Jonas Unger. "A Survey of Image Synthesis Methods for Visual Machine Learning". In: *Computer Graphics Forum* 39 (Sept. 2020). DOI: 10.1111/cgf.14047.

[33] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. *Adversarial Discriminative Domain Adaptation*. 2017. arXiv: 1702.05464 [cs.CV].

[34] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. *Deep Domain Confusion: Maximizing for Domain Invariance*. 2014. arXiv: 1412.3474 [cs.CV].

[35] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Wenjun Zeng, and Tao Qin. *Generalizing to Unseen Domains: A Survey on Domain Generalization*. 2021. arXiv: 2103.03097 [cs.LG].

[36] Bichen Wu, Alvin Wan, Forrest Iandola, Peter H. Jin, and Kurt Keutzer. *SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving*. 2019. arXiv: 1612.01051 [cs.CV].

[37] Sicheng Zhao, Xiangyu Yue, Shanghang Zhang, Bo Li, Han Zhao, Bichen Wu, Ravi Krishna, Joseph E. Gonzalez, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, and Kurt Keutzer. *A Review of Single-Source Deep Unsupervised Visual Domain Adaptation*. 2020. arXiv: 2009.00155 [cs.CV].