MSc thesis in Geomatics

Automated Semantic Segmentation of Aerial Imagery using Synthetic Data

Camilo Cáceres

2022

MSc thesis in Geomatics

Automated Semantic Segmentation of Aerial Imagery using Synthetic Data

Camilo Cáceres

June 2022

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

The work in this thesis was carried out in the:

3D geoinformation group
Department of Urbanism
Faculty of the Built Environment & Architecture
Delft University of Technology

READAR
BOX A2927
Potsbus 9607
1006 GC, Amsterdam
https://readar.com/

Supervisors:	Shenglan Du
	Prof. dr. Jantien Stoter
	Sven Briels
Co-reader:	Gao Weixiao

Abstract

Semantic segmentation of aerial images is the ability to assign labels to all pixels of an image. It proves to be essential for various applications such as urban planning, agriculture and real-estate analysis. Deep Learning techniques have shown satisfactory results in performing semantic segmentation tasks. Training a deep learning model is an expensive operation, while most of the time manually labelled images are required. Additionally, a bottleneck in semantic segmentation projects concerns the annotation of images. Consequently, synthetic data, which consists of images from a virtual world that simulates the real world, can be used as training data for segmentation tasks to improve the classification results. Therefore, this thesis aims to create a pipeline that generates synthetic images with semantic segmentation labels to be used in an existing deep learning model and discuss how the generated synthetic data improves the semantic segmentation of aerial images. In this research work, an existing model (FuseNet), which in previous works achieved satisfactory results, is trained with solely synthetic data and a mix of real data in different training and testing scenarios to classify true ortho imagery from Haaksbergen, Netherlands and Potsdam, Germany. In addition, a benchmark of domain adaptation techniques is performed to close the domain gap between the synthetic and real imagery. The semantic maps include *building*, *road* and *other* classes. Experiments are performed to test the performance of the synthetic data using 1) Different 3D models of the virtual world, 2) Different quantities of synthetic and real training data, 3) Different cross-geographical scenarios, and 4) Different domain adaptation techniques. The assessment is based on the (mean) intersection over union (IoU), F1 score, precision and recall and an extensive visual assessment. The virtual world is created through a pipeline in CityEngine using procedural modelling techniques and then rendered in Blender to create the training dataset. The results show that the synthetic data has a mIoU of 0.48, which is lower compared to cases when solely real data (0.75) are used, when the segmentation is performed in the same training and testing area. In addition, the 3D models partly affect the segmentation results. When using a mix of real and synthetic data, the results are maintained to a mIoU of 0.75. On the contrary, when training and testing in different areas, the use of synthetic data seems to improve the results on average by 21.5, 12.5, 1.5 and 2 percentage points on the mIoU, IoU for classes *building*, *road* and *other* respectively. Additionally, domain adaptation techniques such as Cycle GAN and Cycada improve the performance of synthetic datasets by 4 percentage points. Overall, this thesis shows that when the domain difference between the training and testing datasets is big, the addition of the synthetic data helps to improve the performance of the semantic segmentation of aerial images. Synthetic datasets improve the segmentation results by using a mix of existing labelled imagery from different geographical regions when a project lacks labelled imagery. In contrast, when labelled imagery is present in the same testing area, the real training data obtains robust results, thus the addition of synthetic data does not improve the segmentation results.

Acknowledgements

Firstly, I would like to thank my TU Delft supervisors Shenglan Du and Jantien Stoter, for the valuable guidance and feedback provided throughout the thesis. In addition, I would like to thank my co-reader, Geo Weixiao, for his useful suggestions. Moreover, I would like to express my special thanks of gratitude to Sven Briels for having me in READAR and for his supervision, guidance and support in completing the research work. Finally, I would like to thank Amber Mulder, also from READAR, for her technical support and knowledge on deep learning.

Finally, there is no words to describe my gratitude to my parents, whose invaluable support and love allowed me to finish this journey.

”No sólo no hubieramos sido nada sin ustedes, sino con toda la gente que estuvo a nuestro alrededor desde el comienzo, algunos siguen hasta hoy. Gracias Totales ”- Gustavo Cerati

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

1. Introduction

Image semantic segmentation is one of the fundamental tasks in remote sensing. Semantic segmentation is the ability to assign labels to all pixels of an image [Garcia-Garcia et al., 2017]. Semantic segmentation proves to be an essential prerequisite for various applications such as urban planning, agriculture, and real-estate [Kampffmeyer et al., 2016; Zhu et al., 2015]. For instance, classification between roads and buildings from aerial images is important for change detection and for updating cartography in the built environment [Saito et al., 2016]. Traditionally, unsupervised and supervised methods are used to tackle semantic segmentation problems with the use of statistical properties on the feature space of the image [Rosenberger et al., 2006]. Recently, automated segmentation of aerial imagery has been addressed with deep learning techniques that present satisfactory results for targeted classification tasks [Yuan et al., 2021]

Most deep learning models consist of four phases. The first phase is to obtain a representative input data for the problem to solve. Second, the input data is annotated according to their semantics to acquire the desired output dataset. Third, the model is trained to learn to infer the desired output from the input dataset. Finally, the trained model is applied to a real-image target in the inference phase [Nikolenko, 2021; Liu et al., 2017]. During the process of developing a deep learning model, with insufficient open datasets, around 80% of the project time can be spend on the annotation phase, which is done manually or is manually checked after an automatic process [Nikolenko, 2021]. In addition, the acquisition of labelled data is expensive for vast geographic regions [Kong et al., 2019]. Several approaches have been made to tackle this problem. For instance, Maggiori et al. [2017] created an extensive dataset with labelled data for five cities across the world, which helped scientists to lower the time of the annotation phase. Nevertheless, despite being a large dataset, this dataset lacks geographical variability in real-world scenarios [Kong et al., 2019].

Another way to tackle the problem of annotation is to create synthetic data for training deep learning models. Synthetic data refers to imagery from a virtual world that simulates the real-world [Nikolenko, 2021; Kong et al., 2019; Ros et al., 2016]. Compared to real imagery data, synthetic images have several significant advantages, such as simulating different conditions (e.g., lighting, camera positions), lowering production costs and producing unlimited possibilities of images with pixel-wise annotations [Nikolenko, 2021]. Nevertheless, the use of synthetic data has led to new challenges, such as the difference of domains between the real and virtual world [Nikolenko, 2021; Kong et al., 2019]. For this problem, various domain adaptation techniques have been developed to adapt the domain of synthetic imagery to the real imagery domain.

Synthetic training data allows for improving models, as well as lower the production time and costs [Nikolenko, 2021; Kong et al., 2019]. Consequently, the current thesis aims to create a benchmark to produce and evaluate synthetic data for automated aerial image segmentation.

For this research, ESRI CityEngine is employed to create and use virtual cities with procedural architecture techniques that enable design parameters such as type of roofs and road

designs. The next step is the renderization of depth and RGB images from a simulated camera. In this process, the pixel-perfect annotations are made automatically. Afterwards, an existing deep learning model is trained to evaluate the performance in real-world imagery. Finally, a domain adaptation technique exploration is performed on the synthetic images to further improve the results and minimize the domain differences between the real and the virtual world. The images in which the model will be evaluated are True Orthophoto (TO) and a Digital Surface Model (DSM) provided by READAR [1] for the case of the Netherlands. Additionally, the model is tested on a public dataset of Potsdam, Germany ISPRS [2020].

1.1. Objectives & research questions

The main objective of this thesis is to show a proof of concept to use synthetic imagery for semantic segmentation on aerial images. For this concept, we built an automated pipeline that generates a virtual city to render synthetic imagery. The output of the virtual city consists of three imagery types; a labelled image with the pixel-perfect annotation, a DSM and a RGB image. Furthermore, this thesis focuses on evaluating the performance of the synthetic images as training data for real imagery segmentation models.

The second objective is to evaluate different design parameters of the virtual city. We do this to investigate how the design of the 3D models affects the quality of the synthetic imagery. As the virtual world enables different design characteristics, it is possible to assess the inference of training data characteristics in the segmentation model's performance.

Finally, the last objective of this thesis is to evaluate, according to the prediction performance, different algorithms of domain adaptation capable of reducing the gap between the synthetic and the real-world domain.

To achieve the aforementioned objectives, this thesis is answering the following main research question:

To what extent can synthetic data improve the current Deep Learning-based models for automated semantic segmentation of aerial images?

The following sub-questions are listed to solve the previous question:

- How to build an automatic virtual city for creating synthetic imagery used as training data?
- How do the entities of 3D models (e.g. buildings, roads or trees) of a virtual city affect the results of semantic segmentation of aerial images?
- What is the most suitable quantity ratio between real and virtual training data for semantic segmentation of aerial images?
- Does synthetic data improve automated semantic segmentation of aerial images in cross-geographical scenarios? Cross-geographical is using real data from a particular area and testing data of a completely different area.
- Which domain adaptation technique is more effective for adapting synthetic imagery to real imagery domain?

[1] readar.com

1.2. Scope of research

This research aims to create a pipeline that generates synthetic images with semantic segmentation labels to be used in existing deep learning models. This thesis will focus on creating synthetic training data and how it improves semantic segmentation model which is used on real-world imagery. Moreover, this study will focus only on aerial images, specifically true orthophotos plus a DSM. This thesis will only focus on two geographical domains in the Netherlands, the cities of Haaksbergen for training and testing data and Haarlem, for testing data. In addition, Potsdam in Germany, for training and testing. Nevertheless, the use of synthetic data imagery could potentially be used in other locations if the true orthophotos and the DSM are available. Furthermore, this thesis will not evaluate the performance of different deep learning architectures. Instead, it will use a current, well-known model to assess the quality of the synthetic imagery.

1.3. Thesis outline

The research is organized as follows:

- Chapter 2 is an overview of the theoretical background and related work going from semantic segmentation with deep learning, synthetic data, and procedural modelling to domain adaptation models.
- Chapter 3 is the methodology of the pipeline to create synthetic imagery, including the design of the 3D models from procedural modelling, the experiment design, the renderization process and the evaluation steps.
- Chapter 4 describes the implementation details, the different testing scenarios and the performance measurements to analyze each synthetic dataset.
- Chapter 5 presents the results of the experiments with the analysis and the relevant discussion about synthetic data in automated semantic segmentation of aerial images.
- Chapter 6 gives the conclusions of this study and presents the main limitations alongside the recommendations for future work.

2. Theoretical Background and Related Work

2.1. Deep Learning for Automated Image Semantic Segmentation

Semantic segmentation is one of the main tasks for remote sensing, giving each pixel a meaningful class in an image according to human perception [Zhu et al., 2015]. Semantic segmentation is a supervised model trained by labelled data. Next, a model learns to predict the conditional probability of a pixel to be labelled as a fixed and unique class [Marmanis et al., 2016].

In the case of remote sensing data, semantic segmentation is essential to give meaning to the image, recognize it as an urban or rural environment or segment the image into meaningful classes (e.g. buildings, roads or vegetation). Another capability of semantic segmentation is to divide the building roofs into their semantics (e.g.dormers, chimneys or solar panels). Due to this variability, complexity, and heterogeneity of the remote sensing data, it is a complex problem to do semantic segmentation on these images [Kampffmeyer et al., 2016]. Nevertheless, compared with traditional methods, deep learning models have shown outstanding performance for semantic segmentation [Yuan et al., 2021], making an impact in remote sensing.

2.1.1. Deep Learning for images

Deep Learning is a machine learning technique that consists of methods that work by learning complex representations from raw data input [Goodfellow et al., 2016]. A deep learning model includes several sets of layers, called neural networks, which are non-linear functions that compute meaningful mappings between the input and the output layer [LeCun et al., 2015]. A neural network is composed of an input layer that contains the observable data, one or multiple hidden layers that extract features from the input data, and an output layer that includes the requested information of the input data [Goodfellow et al., 2016].

A Deep Learning method that learns features of images using filter layers through backpropagation is called Convolutional Neural Networks CNN. These networks are designed to deal with data in the form of arrays using convolutional filters to depict features of these images. A CNN is a neural network that uses convolutional operations. Traditional CNNs consist of four main types of layers: Convolutional layers, non-linear function layers, spatial pooling layers and either fully connected layers in traditional CNN, or transposed convolutional layers in semantic segmentation tasks [Liu et al., 2017].

The main characteristic of CNNs are the convolutional layers (Figure 2.2a), which are filters with learnable parameters. These filters help depict features of the image, from basic types

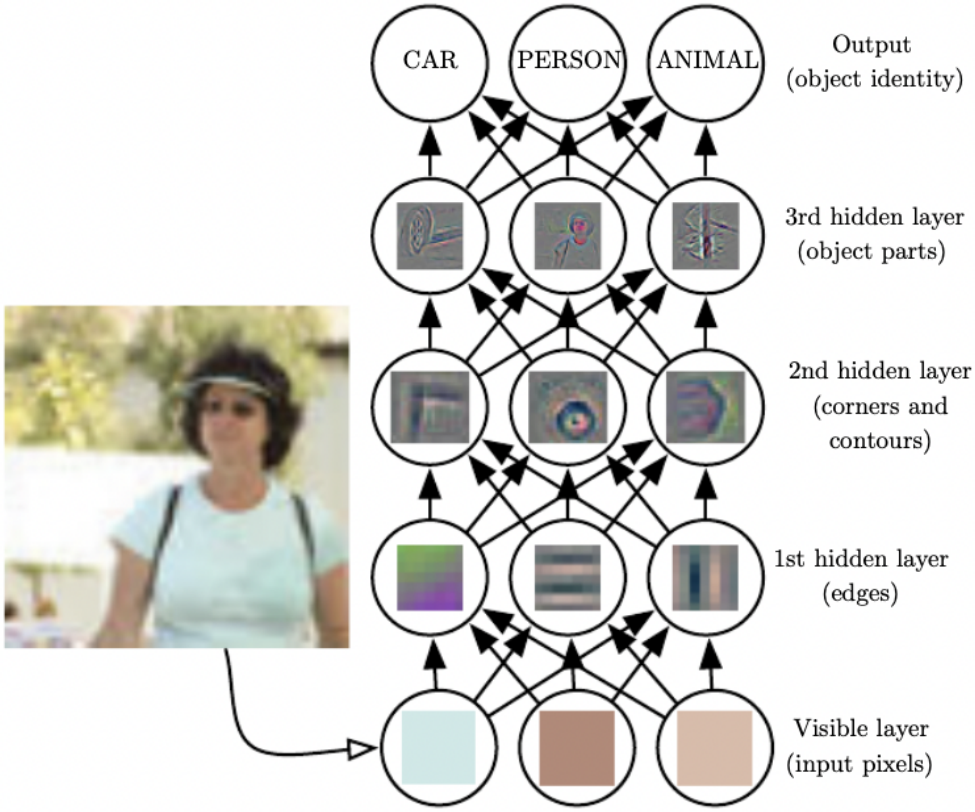
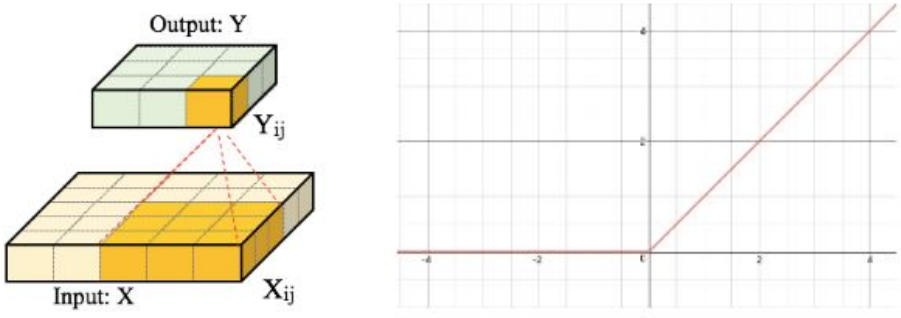


Figure 2.1.: Example of a Deep Learning Architecture with the neurons as the kernels inside the circles. The hidden layers extracts different features of the image to the final layer that identifies the object in the input layer. Image taken from: [Goodfellow et al., 2016]

such as edges or curvatures to more complex representations such as bicycles or a person. The operation includes an input image X with a size of $W \times H$ and a filter operation of size $w \times h$ called kernel as initial data. The filter operation performs the dot product between the kernel's and the corresponding part of the image, resulting in an output of size $W_2 \times H_2$ that is the convolution of the input. The resulting number of pixels depends on the stride S, which is the step that will be used in the next slide of the input and padding, which can be considered as an additional pixel in the borders of the input matrix, to be sure that features in the borders are into consideration [Liu et al., 2017]. Thus, the dimensions of the output are:

$$\mathcal{W}_2 = (W_1 - w + 2P)/S + 1, H_2 = (H_1 - h + 2P)/S + 1 \tag{2.1}$$

After performing a convolutional layer a non-linear function layer is used to introduce non-linearity to the network and be able to learn complex features. The most common function used in deep learning is the rectified linear unit ReLu function, *f(x) = max(0,x)* (Figure 2.2b) [Liu et al., 2017]. The spatial pooling layers are for filtering the input to reduce its size and



(a) (b)

Figure 2.2.: (a) Example of a Convolutional layer: the input size is $W_1 = H_1 = 5$; the convolution is performed with stride $S = 1$ and no padding ($P = 0$). The output Y is of size $W_2 = H_2 = 3$. Image taken from: [Liu et al., 2017]. (b) The Rectified Linear Unit Rectified Linear Unit (ReLu) activation function produces as an output when $x < 0$, and then produces a linear with slope of 1 when $x > 0$. Image taken from: [Agarap, 2018].

the number of parameters. The most common pooling functions are max, mean, and sum functions [Liu et al., 2017].

Last are the fully connected layers that create the class scores from the activation by reducing the dimension of the 2D input to 1D for the classification. These layers are mainly used when the classification task is to detect one class in an image. For semantic segmentation with more than one class, the transposed convolutional layer is used for upsampling operations of the input to be able to be the same size as the input image. It is the inverse operation of the convolution with the same sampled factors of stride and padding. Filter parameters can be either learned or manually implemented [Liu et al., 2017].

These upsampling operations output a vector score for each class in which then the highest score is taken [LeCun et al., 2015]. Afterwards, a loss function is used in the mask layer to calculate the error between the output scores and the ground truth. An optimizer is applied to minimize the cost function to reduce the loss. Next, the loss is multiplied by a learning rate to adapt the weights in the direction opposite to the gradient vector [LeCun et al., 2015].

2.1.2. CNNs Architectures for image semantic segmentation

Different architectures have been made to improve CNNs for semantic segmentation in remote sensing. For instance, Kampffmeyer et al. [2016] and Maggiori et al. [2017] used a Fully Convolutional Network (FCN) for semantic image segmentation. This study obtained 87% accuracy in both ISPRS datasets, taken from the cities of Potsdam and Vaihingen [ISPRS, 2020]. FCNs are composed of three phases: multi-layer convolution, deconvolution and fusion. Specifically, FCNs use convolutional layers to get a score for each class. As pooling is used in the convolutional processes, the output size is smaller than the original. Thus, the deconvolution step returns the size but loses spatial detail in the class score. An unsampled deep layer is extracted to get the spatial details back and then fused with a shallow layer by an additional element-sum operation. The deconvolution step enables the model to perform

semantic segmentation as it returns the score to the original size of the image. [Yuan et al., 2021; Long et al., 2014] (see Figure 2.3).

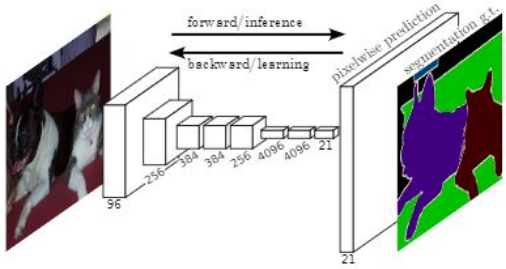


Figure 2.3.: Visual description on how FCNs can perform semantic segmentation by using a deconvolutional layers to get the scores back to the size of the original output. Image taken from: [Long et al., 2014]

Another architecture is U-Net, which consists of convolutional and deconvolutional layers and aims to use little training data. The U-Net is made of a contracting path to get the context of an image and a symmetric expanding path to get precise localization of features [Ronneberger et al., 2015]. This architecture plus fusion layer with a DSM is used by Xu et al. [2018] using very high-resolution aerial images. For the Vaihingen dataset, the U-Net has an accuracy of 96% and, for Potsdam dataset, 98%.

The SegNet architecture [Badrinarayanan et al., 2017] consists of two sub-networks. An encoder and a decoder. The encoder is a structure of convolutional and pooling layers to extract features. However, spatial information detail is lost. In the following network, the decoder is used to recuperate the lost spatial information using an upsampling process [Yuan et al., 2021]. Audebert et al. [2017] uses this architecture for the Vaihingen dataset, resulting in an accuracy of 89%. From SegNet, a new architecture was created called FuseNet, which uses three sub-networks, two encoders, the RGB values and depth values and the decoder with a fusion layer that processes the RGB-D values [Hazirbas et al., 2015]. Audebert et al. [2017] uses FuseNet with a DSM and very high-resolution images as input data, getting an accuracy for both Potsdam and Vaihingen datasets of 90%. On the other hand, Mulder [2020], in an implementation of a FuseNet model with 4-band images (RGB-Z), obtained mean Intersection over Union (mIoU) of 0.87 on experiments in the city of Haarlem in the Netherlands. These results were better than using SegNet architecture using the same training data and evaluation method.

FuseNet Architecture

FuseNet architecture is the one chosen to use in the current study because the fusion approach, in which is used an extra encoder rather than stacking an extra band (U-Net, SegNet, FCN), performs better for semantic segmentation of aerial images [Mulder, 2020] (see Chapter 3). This architecture is explained in further detail. FuseNet is based on two parts. First, the encoder part, which extracts the features of the image and second, the decoder part, which upsamples the feature maps back to the original input resolution [Hazirbas et al., 2015]. Another encoder is added to the architecture to consider the depth band. The name of fusion is because the feature maps from the depth data are constantly combined into the RGB feature maps. The encoder parts follows the 16-layer CNN based on [Simonyan and Zisserman, 2014]

which use a very deep CNNs that is suited to perform in large-scale image classification. In the encoder part, networks are in the following order: Convolutional layers, batch normalization of the feature maps and ReLu function to reduce the covariance shift [Hazirbas et al., 2015]. On the other hand, the decoder part uses unpooling layers to upsample the feature maps and uses the same structure as the encoder part. Furthermore, to combine the depth and RGB encoders, a fusion block is used. This layer is an element-wise summation, which allows for enhancing the mapping features of the RGB with the discontinuity of depth images extracted in the encoder networks. In Figure 2.4 shows a graphic summary of the FuseNet architecture.

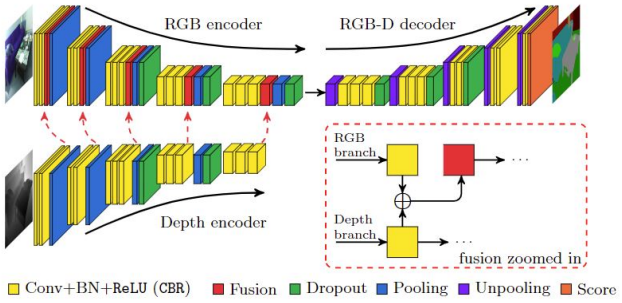


Figure 2.4.: FuseNet architecture, consisting in two encoder and one decoder. In the encoder part of RGB, fusions layer is added to constantly taking into account the depth feature maps. Image taken from: [Hazirbas et al., 2015]

2.2. Benchmarking of Synthetic Imagery to train Deep Learning Models

Synthetic imagery is defined as "imagery that has been captured from a simulated camera operating over a virtual world" [Kong et al., 2019]. Instead of training deep learning models with costly aerial or satellite images, a synthetic imagery approach can have several benefits, such as free labelled data as classes are defined by design, simulation of different seasons or adjustable lighting conditions. In addition, the variability of the images can be set in the design process of the virtual world. [Nikolenko, 2021; Kong et al., 2019].

2.2.1. Synthetic Imagery to train semantic segmentation networks

One of the first approaches of synthetic imagery for training deep learning models was the Synthetic collection of Imagery and Annotations of urban scenarios (SYNTHIA) dataset [Ros et al., 2016]. This dataset was created to have pixel-perfect semantic segmentation for automated car navigation. SYNTHIA images are created automatically in a virtual environment using Unity, and it uses a virtual array of cameras throughout the city that simulates a real car. The research provides two sets, one for training data for deep learning and another to analyze spatio-temporal constraints of objects [Ros et al., 2016]. The first set consists of 13400 images trained in a FCN model and evaluated in Cambridge-driving Labeled Video Database (CamVid). The results showed an accuracy of 78% only with CamVid dataset and,

with the addition of the SYNTHIA dataset, the accuracy increased to 84% [Ros et al., 2016]. Nevertheless, for the Street Scenes Challenge Framework (CBCL) dataset from Chicago, USA, an initial 79% of accuracy was observed using only their dataset, and 75% with SYNTHIA. They believe the decrement is due to the combination of early and late layers during upsampling, but no further evaluation was made [Ros et al., 2016].

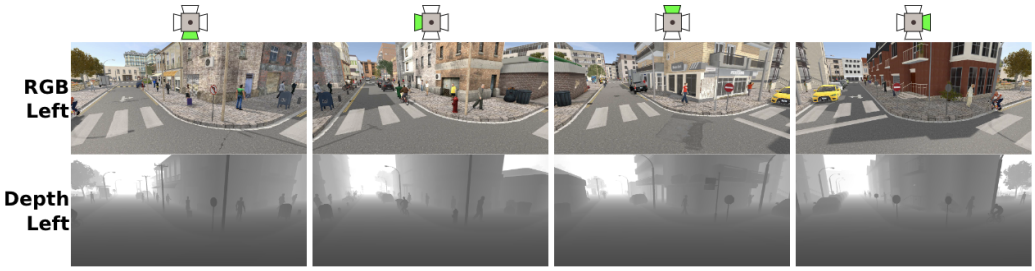


Figure 2.5.: Examples of SYNTHIA images plus the depth image. SYNTHIA dataset is for unmanned vehicles purposes. Image taken from: Ros et al. [2016]

Another example of a synthetic dataset is Procedural Synthetic Dataset (ProcSy) [Khan et al., 2019], which is a synthetic dataset from ESRI CityEngine; it is used for automatic driving semantic segmentation. They applied different weather conditions like clouds and rain and created 8000 images for experiments. The training data is input to a DeepLab v3+ model into the real-world dataset called CityScapes, obtaining favourable results with mIoU between 70 and 75% [Khan et al., 2019]. The virtual city is based on a real city in Canada, and they used the dataset to study the effect of different conditions in the current deep learning algorithms. This study did not show how the dataset will perform in another location and how it adapts from the virtual to the real domain.

In the case of synthetic data for aerial or satellite images, Kong et al. [2019] created a dataset called Synthinel. Using ESRI CityEngine, a virtual city was created from the program's default settings. They used 1640 images to train a DeepLab v3+ and a U-net model for building classification. The models were performed with real and synthetic training data to evaluate it on the Aerial Image Labeling Dataset (INRIA) [Maggiori et al., 2017] and ISPRS [ISPRS, 2020] datasets. For U-net, using INRIA dataset, the improvement of mIoU was 0.3%, from 69.0 to 69.3%. For DeepLab v3+ the improvement was greater (1.1%) from 72.2 to 73.3%. Nevertheless, the research also performed blind segmentation, which evaluates the ISPRS dataset with training data from both the Synthinel and INRIA datasets. As the domain changed, the results decreased, but the impact of the synthetic imagery increased. Using a U-net without the Synthinel, the mIoU was 45.0%, and with the synhtetic imagery, was 47.7%. On the other hand, for DeepLab v3+ the increment was greater, going from 58.1% to 63.5% [Kong et al., 2019].

2.2.2. Design of Synthetic Imagery

There are two approaches in the design of synthetic imagery for training Deep Learning networks. The first and most commonly followed method is to produce synthetic imagery with realistic features. As the domain difference is smaller with realistic features, the model learns easier, the context of the real world in a synthetic scene [Nikolenko, 2021]. Nevertheless, problems arise in designing a realistic synthetic world. First, the design process is complex as the variability of real environments is high (e.g.amount of different layouts, textures and

Figure 2.6.: Synthetic city from ESRI City Engine used in [Kong et al., 2019]. Default settings from City Engine creates this city. Image taken from: Kong et al. [2019]

shapes). In addition, hardware requirements for detailed virtual worlds are higher and costly. Moreover, making a real-world environment involves focusing on a specific area. Thus, cross-domain applications will require the creation of another virtual world [Kong et al., 2019].

The second approach is domain randomization. It aims to make the domain of the synthetic data distribution wider and with high variability to reach a robust training that takes into account the real domain [Nikolenko, 2021]. The design phase can achieve the variety needed by creating more objects and setting different textures and distractors that can work as negative training. Similarly, different lighting conditions, camera parameters, and render quality can obtain enough randomization [Nikolenko, 2021].

In contrast to indoor or car navigation semantic segmentation tasks, where the area of the virtual worlds is smaller and with more focus on small objects, in remote sensing tasks, the virtual worlds have to be bigger to achieve the real imagery resolution [Kong et al., 2019].

2.3. Procedural Modelling for creating a synthetic world

The creation of a synthetic dataset is not trivial. It has to be realistic and random to have generalization. The abstraction of the real world in computer features involves the creation of 3D model representations usually involving geometric, topological and semantic representations [Ohori et al., 2020]. In practice, according to the topic, balance decisions between these representations are made in order to be flexible and structured [Ohori et al., 2020]. In the case of the built environment, buildings can be modelled in high detail by hand with both indoor and outdoor specifications. Nevertheless, memory and time consumption can be important variables to consider. Procedural modelling comes at hand in creating a synthetic world which images can be used for training deep learning networks. Procedural modelling comes from Lindenmayer Systems (L-systems) where there is an ”initial axiom string from which the pattern propagates, and there exist rules to translate the string into generated structures after every iteration” [Khan et al., 2019; Prusinkiewicz et al., 2013; Lindenmayer, 1968]. Müller et al. [2006] adapts the L-systems to the modelling of 3D representations of the real world with a set of rules [Ebert et al., 2002]. Procedural modelling features an abstraction of the scene.

Rather than pointing out every detail, the scene is generated by an algorithm or a function. Moreover, procedural modelling gives parametric control to the scene (e.g. angle of roofs) [Ebert et al., 2002]. With procedural techniques, one can transform a simple square shape into a highly detailed building, as seen in Figure 2.7.

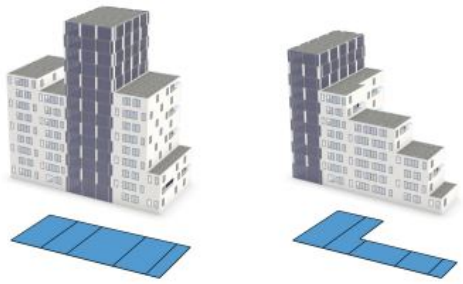


Figure 2.7.: Example of how procedural modeling can transform from a simple 2D shape to a 3D high detailed building. Image taken from [Schwarz and Müller, 2015]

2.3.1. Computed Generated Architecture Shaped Grammar

From the idea of the L-systems, a similar approach of procedural modelling was made by [Müller et al., 2006] in creating CGA grammar language. This grammar language was bought by ESRI and used in CityEngine. The primary purpose of this program is to rapidly create large urban environments that can be used as analytical information on different subjects such as urban planning, video games, and 3D cadasters, among others [ESRI, 2022].

Computed Generated Architecture CGA is a grammar-based programming language which sets a configuration of shapes, consisting of three orthogonal vectors to represent coordinates and an additional vector to represent the size. Each shape can be terminal or non-terminal, depending on the shape's capability to transform into a different one. CGA language consists of a series of "production processes" in which a rule X is performed into a predecessor shape A to create a successor shape B. The notation of these rules is the following:

$$id : A : cond \rightarrow B : prob \tag{2.2}$$

where *id* is the unique identifier, the *cond* is a logical expression to perform the rule, and *prob* is the probability of executing the rule. This procedure enables the creation of a hierarchical shape structure for the storage of the geometry, semantics and topology of all shapes [Müller et al., 2006].

CGA consists of two types of basic rules named *scope* and *split* rules. The *scope* rules modify the shape with basic geometric and semantic operations such as translation, rotation or scale, or by adding an *id* to the shape, and the *split* rules perform a division of a shape into two or more sub-shapes [Müller et al., 2006]. As CGA focuses on the built environment, it has simple building structures with common roof shapes integrated. In Figure 2.8 a basic grammar example is explained with the final output and the hierarchical tree that is inside the program.

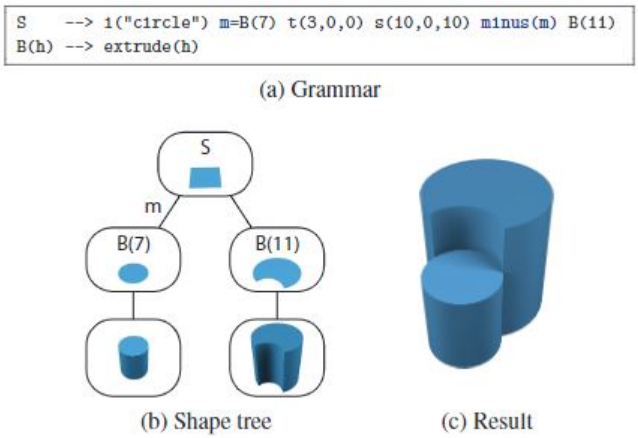


Figure 2.8.: Example of a grammar (a), showcasing labels and operations using other shapes. The according derivation process defines a shape tree hierarchy (b), whose leaf nodes determine the final result (c) [Schwarz and Müller, 2015]. Image taken from [Schwarz and Müller, 2015]

2.3.2. International CGA rule

Knowing the basic structure of how the CGA rules work. ESRI RD Center Zurich developed a CGA [Müller et al., 2006; Schwarz and Müller, 2015] rule that creates a random city for City Wizard's [1] tool in CityEngine. The input of this rule are 2D planes called *lots* (see Chapter 3), which are formed by the street network (see Figure 2.9). The international rule works with the following types of lots; open space, residential, apartment block, office block and high-rise block. One of these types is selected depending on the distance between the origin of the whole scene and the lots. The first main rule of the script is the lot preparation in which the area is divided into different footprints for the building and the ground area. Then splitting and scope rules are performed to transform the 2D footprint into a 3D model. At the same time, the texturing from the facades and the roof is performed. The shape of the building is created with a stochastic mass model that combines different basic shapes (see Figure 2.8). The creation of the whole city, including models of trees, roads, buildings and some other 3D objects, takes minutes for a large area (see Figure 2.10). These default models will be used to set a starting point for this research.

2.4. Domain Adaptation

Synthetic data is different from the real data as the source; randomness and environment conditions are different. The main drawback of synthetic data to be used to train segmentation networks is the domain difference between the synthetic training data and the real testing data [Nikolenko, 2021]. Domain Adaptation is a set of techniques to close the domain difference between synthetic and real data [Nikolenko, 2021]. In the case of the current work, some Domain Adaptation techniques will be performed to close the domain difference between

[1] https://doc.arcgis.com/en/cityengine/2019.0/get-started/get-started-workflows.htm

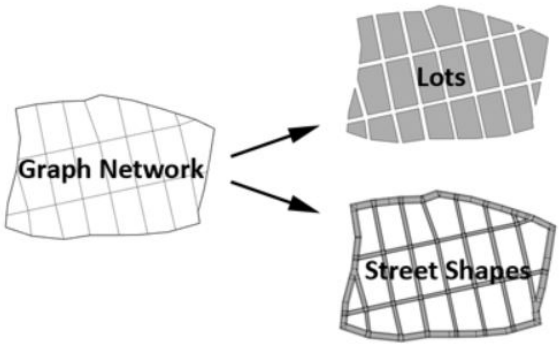


Figure 2.9.: Creation of lots and street shapes from a graph network. Image taken from [ESRI, 2022]

(a) (b)

Figure 2.10.: (a) View of the *international city* model using CGA rules (b) Overhead view of the same city of (a). Image taken from CityEngine.

the imagery taken from the virtual world and the real imagery to which the model will be applied.

2.4.1. Domain Adaptation to close domain difference between synthetic and real-world domain

Real and synthetic imagery have different distributions, leading to a shift in their domains. This shift decreases the performance of the models. DA techniques come handily [Wang and Deng, 2018; Sankaranarayanan et al., 2017] to reduce domain differences. One DA method is transfer learning. The main objective of these methods is to reduce the shift difference between source and target domains with statistical approaches. On the other hand, Generative Adversarial Networks Generative Adversarial Networks (GAN) are methods of DA that consist of a generative model and an adversarial model to transfer the domain from the target to the source dataset. The generative model depicts the domain distribution from both datasets, and the adversarial model creates a binary label to distinguish between the training and the real images [Wang and Deng, 2018; Liu and Tuzel, 2016].

2.4.2. Transfer Learning

The primary purpose of transfer learning is to reduce the domain difference between the source and the target domain. Sun et al. [2016] brings a "frustratingly easy" implementation called Correlation Alignment (CORAL) which uses the correlation, a second-order statistic, to align the domain between the source and the target set. For this alignment, the objective is to apply a A transformation to the covariance of the source image C_s that the difference between C_s and the target covariance C_t is 0.

$$\min_A \|A^T C_s A - C_t\|_F^2 \tag{2.3}$$

In visual terms, CORAL aligns the domain distribution by re-colouring whitened source features with the covariance of the target distribution [Sun et al., 2016]. This method consists of two steps. The first one is to get the covariance for both source and target domains and then apply the whitening and re-colouring transformations to the source image.

2.4.3. GANs

Generative Adversarial Networks GAN are a class of neural networks that combine a generative model with a discriminative model. Generative models create new instances from the source data, and adversarial models discriminate between the source and the target data. Formally, generative models extract the joint probability of both datasets $P(S, T)$ being S the source images and T the target images. In contrast, discriminative models depict the conditional probability of $P(T|S)$ [Goodfellow et al., 2014; Shor, 2022].

GANs have a discriminator that learns to distinguish real from fake instances. It learns through backpropagation by updating the weights from a discriminator loss function that penalizes the discriminator for misclassifying fake as real data. The input for the discriminator is the instance created by the generator. The generator consists of a random input that goes to a generator that forms the first instance to be classified in the discriminator network. Then, a generator loss function penalizes the generative network for failing to mislead the discriminator. In Figure 2.11, it is observed that the generative model includes the discriminator to produce the generative loss function as it needs the classifier [Goodfellow et al., 2014; Shor, 2022]. Training two models at the same time brings difficulties to convergence as the generative model becomes better for creating fake data that looks real, and the discriminative model fails to classify the generated instance [Shor, 2022]

The use of GANs for domain adaptation between synthetic and real data is extensive and includes different approaches. For example, Cycle GAN learns to adapt from unpaired images. In the standard version of GAN, there is always an input X that is manually mapped with the distribution Y to be able to transform its domain. With unpaired images, X does not have any meaningful map with Y. Thus, this mapping is created by the Cycle GAN [Zhu et al., 2017] which adds another generator that not only maps X with Y but also maps Y with X. Formally, an image X is a generator's input to create a new image Z in the Y domain. Then this Z image is an input of another generator that covers it back to the X domain, creating a cycle. Now, the image X can be mapped with the distribution of Z using cycled loss functions between these two sets.

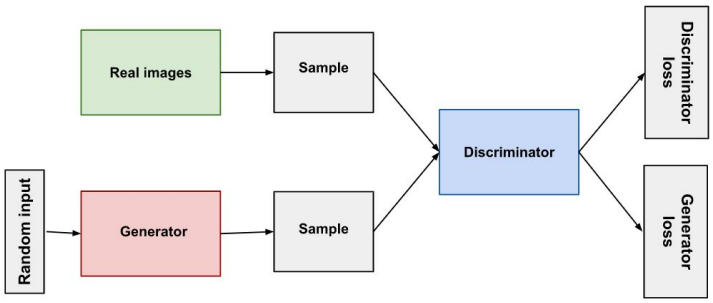


Figure 2.11.: Visual representation of a typical GAN architecture. Image adapted from [Shor, 2022]

Furthermore, Cycle Consistent GAN (Cycada) is developed with a CNN along with a Cycle consistency GAN [Zhu et al., 2017]. The GAN is to transfer feature maps between domains and CNN is to enforce consistency for the relevant semantics (i.e. segmentation labels). In more detail, a pixel-level adaptation is obtained with the training of the generator, from the source to the target domain, that tries to fool the adversarial discriminator that attempts to classify between source and target images. A cycle-consistency method is introduced (i.e.Cycle GAN by [Zhu et al., 2017]) to preserve the source content. With the use of the semantic labels, a pretrained classification model is incorporated to encourage the generated image to be classified in the same way as the source image after the style transferring [Hoffman et al., 2018]. Cycada is evaluated to adapt from GTA5 [Richter et al., 2016] to CityScapes with a mIoU of 34.8 using a FCN to perform semantic segmentation.

Other approaches, such as [Tsai et al., 2018] use the source domain label data to be the input of a generator that creates labels of the target domain. Next, GAN with the weight knowledge of the target label data is performed using the source domain image. This approach generates a mIoU of 35.0, adapting from GTA5 to CityScapes. Moreover, having extra info like DSM could be beneficial to the GAN model as it constrains the learning for semantic segmentation. This architecture has an mIoU of 36.8 in adapting from SYNTHIA to CityScapes.

3. Methodology

This chapter explains the methodology to perform an automated semantic segmentation of aerial images from synthetic imagery in the following main phases. (1) Synthetic city creation, (2) Experiment design to evaluate synthetic city, (3) Rendering training data from synthetic city, (4) Training CNN model for semantic segmentation and (5) Model assessment. In Figure 3.2 the main steps of the methodology are described.

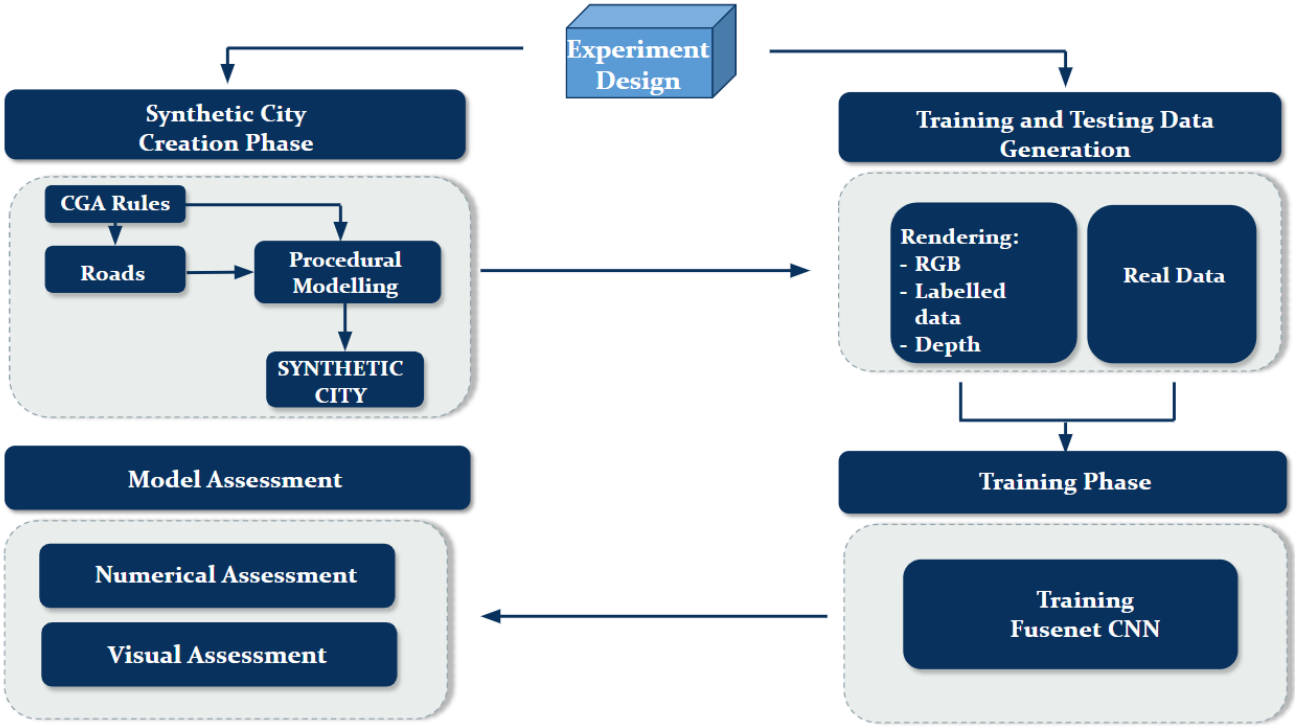


Figure 3.1.: General Methodology of the current thesis

3.1. Synthetic City Creation

The first objective of this thesis is to develop an automatic pipeline to create a synthetic city for the semantic segmentation of aerial images. This city should be generated in a random way so that it can be applied for testing different geographical environments [Kong et al., 2019]. For this reason, producing a digital twin of the testing city could result in satisfactory results. However, their quality might decrease considerably due to the variability of shapes and textures when different areas are tested [Nikolenko, 2021; Kong et al., 2019]. In addition,

in the virtual city, one should be able to build relatively easy, large urban areas with realistic visuals such as the colours, textures, roads and building layouts. Thus, realistic settings should be set, such as the length of roads, height of buildings and trees, and roof angle. These parameters can be retrieved from the real world. ESRI CityEngine [1] is used in this research because it uses simple rules to create random large-scaled virtual worlds which can be easily customised, either manually or automatically through a python Application Programming Interface (API). In addition, the software focuses mainly on urban planning and urban modelling, which makes it suitable for creating a virtual city with different building models.

Figure 3.2 shows the pipeline of CityEngine to create the virtual city. This thesis adjusts the pipeline by implementing changes to customise the city's design to answer the research questions.

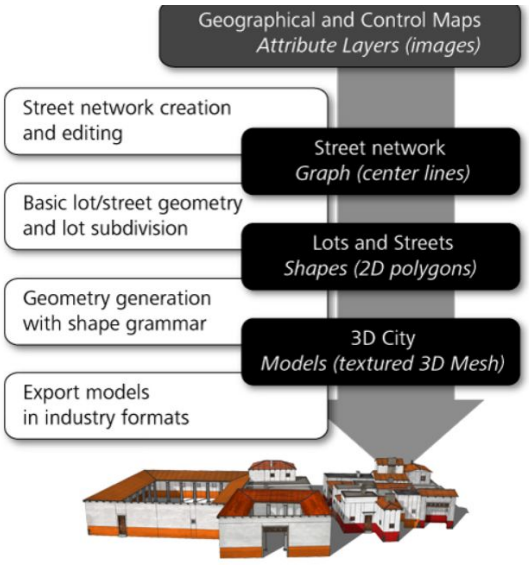


Figure 3.2.: Overview of the CityEngine modelling pipeline. Black boxes illustrate data types (layers), and white boxes the operations to create them. In the first step, the street network is created. Afterwards, the resulting blocks are subdivided into lots. Finally, the 3D models of the buildings are generated using the CGA rules. The output of CityEngine is polygonal building models. Image and reference taken from: [ESRI, 2022]

3.1.1. Geographical and Control Maps

Since we want a random city that can simulate realistically and randomly any city, geographical maps are not needed in this research. A terrain model is generally used for the control maps to adapt the 3D models to the terrain with different heights. For this thesis, a height approach is used in the real training and testing data (more details in Section 4.3.2) where mostly all the ground/terrain parts are normalised to an elevation of $0m$. Hence, a terrain control map is not needed. Furthermore, in the street network (Section 3.1.2), the default approach creates empty spaces in the surrounding areas of the virtual city, which does not help to capture images in these areas (i.e.there is no empty space in real world). For this problem,

[1] https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview

a control map, that consist on a *boolean* raster, is created to extend the city's area by a square of 1.5 by 1.5*km* (see Figure 4.1a).

3.1.2. Street Network

The next step of the methodology is to create the street network. A street network is composed of a graph network and blocks. For the street network, some parameters such as length and width of the roads are set (see Chapter 4 for more details). Then, the road growing pattern is performed. In CityEngine there are three types of patterns for roads. Namely, *organic, raster* and *radial*. These patterns can be applied both on main and secondary roads. The *organic* pattern consists of a road network without a geometrical pattern. *Raster* pattern is a grid-based network with mostly 90 degrees intersection. Last, *radial* pattern consist of curvilinear network. All patterns can be seen in Figure A.1. Moreover, the second component of the street network is the blocks which are the areas formed between the roads. These areas are subdivided into lots with different algorithms. City engine offers three types of subdivisions. During this research, the *offset* subdivision is applied as it creates lots only within a given distance from the street edges of the block. For further details go to Chapter 4.

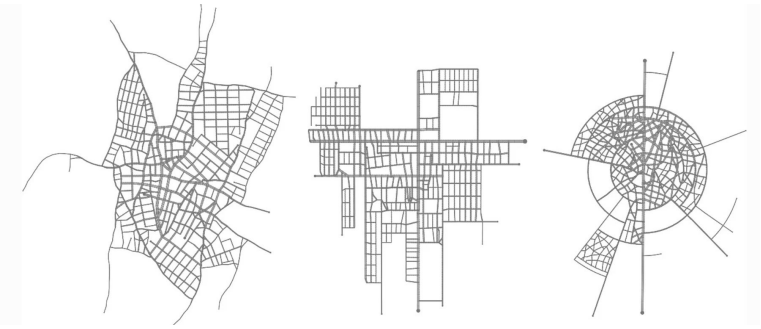


Figure 3.3.: Different street patterns for the creation of roads in CityEngine. Left is *Organic*; Center is *Raster*; Right is *Radial*. Image taken from: [Kelly, 2021]

3.1.3. River Creation

Water bodies are an essential part of the built environment. Unfortunately, there is no built-in approach or rule to make rivers or canals in CityEngine. For this purpose, a technique to create a river without affecting the composition of the virtual city is developed. For further details see Chapter 4.

3.1.4. Procedural Urban Models

After the creation of the road graph, the next step of the methodology is to apply textures to the roads. In this case the CGA built-in rule of *AdvancedStreet.cga* is used. It consists of rules to form the main road, sidewalks, intersections and junctions, trees along sidewalks, bike lanes, cars and population.

Furthermore, the 3D modelling of trees is based on a public library made by ESRI [ESRI, 2022]. It consists of 80 different models of trees, plants, flowers, and shrubs. These vegetation models are divided into three types of models according to their level of detail. The basic model for trees is the *fan* model, which is two intersecting images from a front view of the tree. The second type of tree, with more level of detail, is the *schematic* tree. This model is a generalised canopy of the real tree. The third model is the *realistic* that is a highly detailed, realistic tree (see Figure 3.4). According to Ortega-Córdova [2018], and ESRI [2022], these three models can be used according to the extent of the virtual world. For instance, for a large to a medium extent, the *fan* model is preferably used as it only has 12 points. In addition to the *fan* model, they mentioned that low *polycount* models are also an alternative to these large extents. Hence, two *poly tree* tree models are added to this study.

First, the low *polytree*, with a low number of points and no texture and the low *dead polytree* which is a model of a tree without leaves to simulate different times of the year. The main drawback of these types of trees is the non-realistic visuals. Nevertheless, the rendering performance in CityEngine is not affected by these types of tree models. More details about the file size and performance of tree models can be found in Chapter 4. Moreover, for medium to small extents of virtual worlds, the *schematic* models are used. Finally, for small and very detailed scenes, the *realistic* models are recommended. As we produce large worlds to simulate aerial images, it is not possible to render the highly detailed trees in CityEngine with the current hardware. Nevertheless, it was created medium-sized worlds with only trees and roads that could be used in addition to the normal worlds. Further details in Section 3.2.

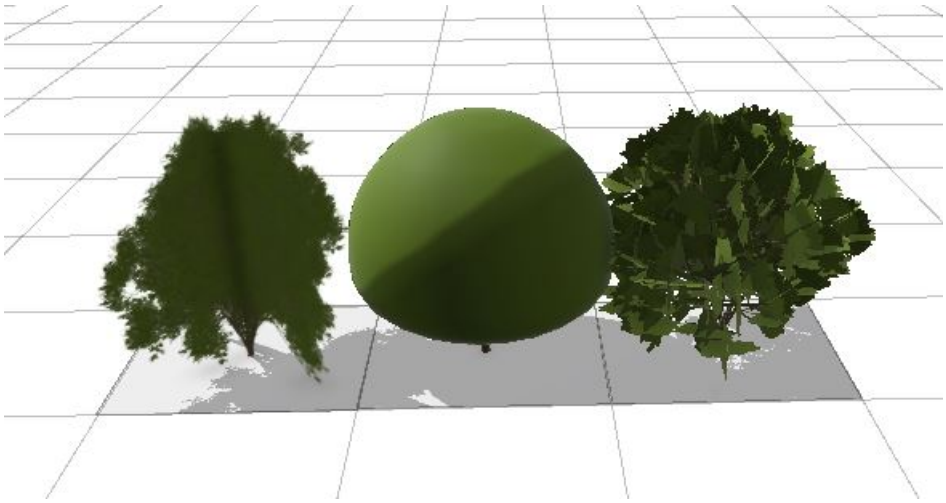


Figure 3.4.: Trees from ESRI [2022] public library. The different trees models are based on the same type of tree. Left: *fan*; Center: *schematic*; Right: *realistic*

The final step of the procedural models is the creation of 3D models of buildings. With the lots created in the street network, a CGA rule is applied to the lot depending on the distance from the centre of the virtual city. In the current thesis, two different rules are applied. The first rule is *international city* rule explained in Chapter 2. For the second rule, a new script was created so that the created environment resembles a Dutch urban area as much as possible. Buildings are visualised realistically and randomly to apply this technique in any Dutch city. The new CGA rule is called *Procedural Dutch City*. It is mainly based on the *international city* rule but adapted to the Dutch characteristics.

Variables	Source	Explanation
Type of building	Own	To build a procedural model and have a variety of buildings, we decided to divide the buildings into three groups. Office buildings, single buildings, and grouped buildings
Height of Building	3D BAG	The height of the buildings. It is the height of the wall surface
Type of roof	3D BAG	In 3D BAG, there are three types of roofs. Slanted, multiple horizontal, and single horizontal.
Roof Angle	3D BAG	The roof angle can be set in CityEngine
Garage	Own	We made a garage in the house which is a cube overlapping the main building
Dormers	Own	Dormers are structures above the roofs, giving more detail to plane roofs. Dormers can contain either flat or slanted roofs.
Solar Panel	Own	Solar panels give more realism to the roofs
Storage Room	Own	Most Dutch houses have in the backside of the little building rooms that should be detected in the semantic segmentation

Table 3.1.: Details concerning the datasets used for the experiments.

Now the question is **what is a typical Dutch building environment?**

To answer this question, the Register of Buildings and Addresses (BAG) [Kadaster, 2022] and 3D Register of Buildings and Addresses (3D BAG) [Peters et al., 2022] are used, where information about the registration and geometrical features of all buildings in the Netherlands are stored, respectively. This information is depicted from a municipality of the country (i.e.Haaksbergen) to perform some statistics and make the stochastic procedural modelling. Some design decisions are considered to enhance the learning of the deep learning model to detect the buildings. Details of how the variables were computed are in Chapter 4. In addition, Table 3.1 describes the variables used to make the stochastic model for the CGA rule.

3.2. Experimental Design Overview

Creating training data is advantageous in assessing the performance of semantic segmentation in deep learning models. Naturally, it is always better to train the model with data in the same domain. In the case of aerial images, the same domain is the same geographical area. In the case of the Netherlands, a cleaned rasterised version of BGT [PDOK, 2021] gives relatable ground truth data to achieve good results in the semantic segmentation, despite having numerous artifacts [Mulder, 2020]. As a consequence, the use of manual labelling is still important. In real applications, manual labelling is constantly required. The quality of the model depends on the annotation quality, the number of training data and the difference of domains between the training and testing data [Castillo-Navarro et al., 2019]. The current work's annotation quality is pixel-perfect as the computer makes it. The problem with synthetic data is the difference between real and virtual domains. To test these differences,

one can assess the number of images, the composition of the virtual worlds and the lighting parameters. This chapter explains the design steps to assess the quality of synthetic data as training data for automated semantic segmentation of aerial images in the Netherlands and Germany. The design of the experiments aims to answer the research questions mentioned in Chapter 1.

- The first experiment includes the quality of the building's 3D models in the training data. For this purpose, two CGA rules are used, being the *international city* which is the pre-built rule in CityEngine and the *dutch city* made for this thesis. The objective of this test is to see the ability of the model to detect buildings or, on the contrary, help to detect objects that are not buildings. In this experiment, the main difference between the two rules is their shape features. The textures are the same.
- Test different configurations of roads. As explained in Section 3.1.2 there are three different types of roads. In this test, the different patterns plus a combination of *organic and raster* patterns are used. This test aims to observe the ability to detect roads with different spatial patterns.
- The model has to detect vegetation as *other* correctly, regardless that the model is not set to classify vegetation. In addition, because the DSM is used as data fusion and the main objects that have height values in the real environment are buildings and vegetation (not counting bridges, cars, persons), the model should learn to classify *trees* as *other* and not as *roads* or *buildings*. For this purpose, different models of trees are tested. As mentioned in Section 3.1.4, there are different types of tree models. First are the low *poly models* which can be rendered in CityEngine. Then, the *fan* models and finally, testing as an augmented data, a city only with *realistic* trees and roads with a small extent of 100 by 100 meters.
- The quantity of training data has been a key variable in deep learning projects [Castillo-Navarro et al., 2019]. For this purpose, different quantities of synthetic data will be trained to compute the learning curve of the model. The first test will be with 378 images; the second test is with 756, the third test with 1512 and the last test with 3024 images. It is important to mention that the components of all these datasets are the same.
- Another approach towards reducing the domain difference between the real and synthetic imagery is to mix it with real training data when it is available [Kong et al., 2019]. In the current thesis, we will use real labelled training data taken from the BGT in the Netherlands and mix it with the synthetic imagery to find the best ratio between real and synthetic data for automated semantic segmentation. The tests include different combinations of real and synthetic data. These are, 100% real data; 80% real - 20% synthetic; 50% real - 50% synthetic, 20% real - 80% synthetic and 100% synthetic.
- In real-life applications, there is a lack of labelled data [Kong et al., 2019]. As a result, labelled data from different areas is frequently used. The use of these images creates a domain difference. It is believed that synthetic data can help close this difference with domain randomisation as the model can learn a broader domain taking the testing domain into account. This test is equal to the previous testing approach with different real and synthetic data ratios. However, the training data will now be in the Netherlands, and the testing data will be in Germany and vice versa.
- The final tests will be a benchmark within different DA techniques using as an input the synthetic data. The first test consists in performing the CORAL technique. The second

test is the same CORAL but by classes. The third test is Cycle GAN to transform the synthetic image to the style of the real one. Finally, is *Cycada* which performs the style transfer with semantic consistency.

In addition to the mentioned tests, different lighting conditions and different weights between classes are tested. Since aerial images have as metadata the date in which the picture was taken, this information can be set when rendering the synthetic world. Nevertheless, different lighting parameters are tested to see if the real lighting conditions are necessary when the model is in the learning phase. On the other hand, a weighted loss function is used for the class imbalance for both training and test data which reduces the domain difference. In addition, the weighted composition of the synthetic data can be set to mimic the real composition. Different datasets with different class-weight are rendered to see if the class imbalance affects the results. Results can be seen in Table A.5.

3.3. Rendering

After setting the synthetic city, the model is exported as a whole mesh to Blender [2] to be rendered. Three types of images have to be rendered to produce synthetic imagery; RGB, depth image and labelled image. In *blender* the mesh is divided into different parts according to the texture, and then the class is assigned as an attribute. Moreover, the locations for every image are computed following a real path of an aeroplane. For each location, a built-in camera in *Blender* is added, and the camera parameters are set according to the image input size of the semantic segmentation model (512,512). The resolution is the same as the real imagery (i.e.10 cm.).

3.4. Training CNN using Synthetic Images for Automated Semantic Segmentation

The first step towards performing semantic segmentation is training the CNN. The input imagery of the model is one RGB image, one 1-band image with height information (depth image or DSM), and one labelled image, also with 1-band with the ground truth of the segmentation. In the current thesis, the semantic segmentation is based in detecting three classes being, (1) *buildings*, (2) *roads*, (3) *other*. The main focus is to detect buildings as they are the instances in which more urban applications are based, such as [Kong et al., 2019; Maggiori et al., 2017] for updating cadastres or solar potential, respectively. Then, another important instance in the built environment are roads [Yucong Lin and Saripalli, 2012; Saito and Aoki, 2015] for emergency routing or navigation. Moreover, instances that are also detected in different semantic segmentation studies, such as cars [Benjdira et al., 2019; Yang et al., 2018] or vegetation [Kattenborn et al., 2021; Ichim and Popescu, 2020] are not taken into consideration in this study as more classes are included in the model, the analysis of the training data becomes nonviable as it will bring more depended variables. In addition, training more classes needs more training data [Castillo-Navarro et al., 2019]. Thus, rendering and training the model will take more time. Furthermore, the labelled data for vegetation in the real training

[2] https://www.blender.org/

set presents some inconsistencies, which make the segmentation of this class nonviable [Mulder, 2020]. As a consequence, in the current study. all the other objects present in the imagery, such as cars, vegetation, people or street utilities, are considered as *other*.

In this thesis, two types of training data are used, synthetic and real data. The synthetic data is taken from CityEngine world and rendered in Blender, and the real training data consists of a TO, a DSM, and the labelled images from the city of Haaksbergen. On the other hand, to perform the experiments for cross-geographical domains, a public dataset of ISPRS from the city of Potsdam is used. This dataset also consists of TO, a normalised DSM and a ground truth with different classes that were changed to be aligned with the classes used in this thesis. It is important to mention that the quality of the segmentation from the BGT could be improved as it misses some small buildings, shows some deviations with the TO image, and presents some overlapping objects [Mulder, 2020]. However, performing and assessing the deep learning model is sufficiently correct.

To train the model, 80% of the data goes to training the model, and the other 20% goes to validation. The training data is to enable the CNN to learn its features and set the model parameters, the validation data for assessment of model performance during the training and prevents overfitting of the model on the training data [Mulder, 2020].

The CNN architecture is FuseNet which is publicly available but adapted to the study context by READAR. To assess the training data is used the same hyper-parameters in all experiments. Details of the hyper-parameters can be seen in Table 4.5.

3.5. Domain Adaptation

An important section of the study is to apply different DA techniques to translate the synthetic images to the real domain. CORAL closes the synthetic domain covariance to the real domain by a covariance adaptation (see Figure 3.5 and Algorithm 3.1).

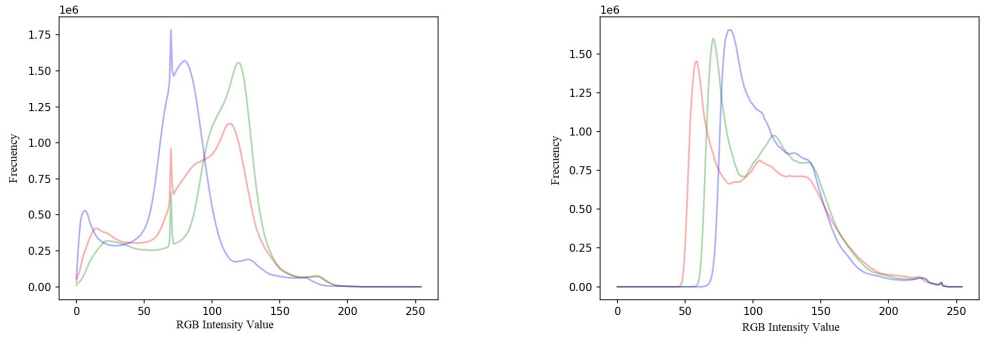
Algorithm 3.1: Coral Algorithm. Adapted from [Sun et al., 2016]

Input: Synthetic Data D_S, Real Data D_T
Output: Transferred Synthetic Data $D_S{}^*$

1 $C_S = \text{cov}(D_S) + \lambda * I$
2 $C_T = \text{cov}(D_T) + \lambda * I$
3 $D_S = D_S * {C_S}^{-1/2}$
4 $D_S{}^* = \text{D}_S * {C_T}^{1/2}$

Since the problem each class of the image has different distribution and in most cases of semantic segmentation problems with synthetic imagery, real training data is commonly used. The CORAL is adapted for each class by mapping it with the existing labelled data. These approaches were chosen because they are not difficult to implement and the results achieved by Sun et al. [2016] are significant.

On the other hand, a Cycle GAN is performed with the synthetic and real data to transfer the real style to the synthetic imagery. Cycle GAN is chosen because it is widely used for image map transferring and especially in style transfer from synthetic and real domains. In addition, this architecture is available as an open-source with different courses, papers and



(a) (b)

Figure 3.5.: (a) Distribution by band for the synthetic imagery (b) distribution by band for the real imagery. Red = Red band, Blue = Blue band and Green = Green band.

documentation. Similar as CORAL, Cycle GAN does not transfer the domain by class as, for example, some features and colours of the *building* class are wrongly transferred to the class *roads*. To correct this problem, the Cycada model uses the labelled data of the synthetic imagery to transfer the real domain while having consistency in every class.

3.6. Model Assessment

3.6.1. Overlap Processing in the Inference step

To assess the performance of the deep learning model, the inference has to be performed in a different area from the training images. The approach of inference will be the overlapping methodology which consists in cutting the whole imagery in overlapping tiles in order to get rid of inconsistencies in the border pixels of the tile [Mulder, 2020; Audebert et al., 2017]. This means that the area is cut into overlapping tiles of 512 by 512 pixels with overlapping of 256 pixels. Then, the outer pixels of each image are removed to eliminate the overlapping. Finally, the images are merged to have a single image with the original size. The network's output is one 2D array per class with a score of how likely the pixels belong to that class. As a consequence, this output is changed into one 2D array with a label of the highest score between classes for each pixel [Mulder, 2020].

3.6.2. Performance Measures

The main idea of this project is to evaluate all different training datasets with the same model parameters to study the importance of the design of the synthetic city depending on the class. Thus, the models will be assessed globally and by class. Performance measures per class are precision, recall, [Tempfli et al., 2009] and Intersection over Union (IoU) [Garcia-Garcia et al., 2017] and global measure are mIoU and F1 statistic.

For assessing each class separately, the following measures are explained:

- **Precision**: It is the ability of the classifier to mislabel as positive a sample that is negative Buitinck et al. [2013]. For example, in this context is the ability of the model to not classify a pixel that is *other* as *building*. This measure is important as the model tends to classify as *buildings*, trees that are part of the class *other* because of the use of the height value. Precision is the ratio between true positives and false positives FP plus true positives TP:

$$Precision = TP/(TP + FP) \tag{3.1}$$

- **Recall**: It reflects how the model identifies positive samples. In the context of the current thesis is important to reflect, for instance, the ability of the model to classify all the building pixels as *building*. The model can learn to classify buildings perfectly with a particular layout or size, but the recall gives a score on how it is classifying all types of buildings. The recall is the ratio between true positives TP to true positives TP plus false negatives FN.

$$Recall = TP/(TP + FN) \tag{3.2}$$

- **Intersection over Union** IoU: It is the ratio between the intersection and the union of the predicted samples and the ground truth for each class. It can also be the mean ratio between true positives TP and the sum of true positives TP , false negatives FN and false positives FP. [Garcia-Garcia et al., 2017]. This measure is the standard for semantic segmentation and on which most of the related work is based.

$$IoU = TP/(TP + FN + FP) \tag{3.3}$$

Moreover, evaluating the model is not as straightforward as evaluating each class due to the problem of class imbalance [Kazakeviciute-Januskeviciene et al., 2020]. To avoid the imabalance problem, mean intersection over union mIoU and the average F1 statistic are used. These two measures are used to compare them with the related work.

- **mean Intersection over Union**: It is the average of the IoU. Often is named as Jaccard index.

$$mIoU = \frac{1}{Classes} \sum (TP/(TP + FN + FP)) \tag{3.4}$$

- **average F1 score**: It is often interpreted as "the harmonic mean of the precision and recall" [Buitinck et al., 2013]. The relative contribution of recall and Precision are equal. Several studies for semantic segmentation present this score.

$$F_1 = 2 * (precision \times recall/precision + recall) \tag{3.5}$$

4. Implementation Details

This chapter is an overview of the implementation of every step mentioned in Chapter 3 with details about the datasets and tools used.

4.1. Synthetic City in City Engine

4.1.1. Street Network

The synthetic city is implemented in ESRI CityEngine program following the proposed pipeline (see Figure 3.2). The first step is to make the control map to restrict the extent of the virtual city. A control map consists of a *boolean* raster where 1 is the area to build, and 0 is the restricted area. Because the extent of the city is set of 1.5 by 1.5*km.* the control map has to be bigger in order to set the restricted area. Thus, the extent of the control area is 2 by 2*km.*, with 0.25*km.* of restricted length in all sides (see Figure 4.1a). The resolution of this control map is set to 5 meters to be easy to handle. The control map is made in Python with an output file as ESRI ASCII[1].

The next step is to create a street network. The patterns are set through a Python file executed in CityEngine environment. In this case, all patterns are tested, *radial*, *restar*, and *organic*. Moreover, CityEngine divides the roads into two groups, primary and secondary streets, and sets the length of a new street according to a Gaussian distribution given by the mean and standard deviation. In order to have a realistic length, we used the Transport Networks (RWS) which is the dataset of streets in the Netherlands and took the city of Haaksbergen to compute the statistics for the street network in this municipality. Some cleaning was made because the RWS can take as a street feature, multiple segments and for CityEngine a street is a simple segment composed of two points. The process is implemented in QGIS 3.14 [2] with *exploding* tool, to get single part from multipart polygons, and then the length is computed to get the statistics in a table. Because the number of short segments is much higher than the longer ones, we name as long streets the ones that are higher than the third quartile and short those lower. Then, the average and the standard deviation are computed. Results are shown in Table 4.1. On the other hand, the street width and the sidewalk width can only be set with one value for each primary and secondary street. Thus, the default values are used. It is important to note that these variables can be set in the process if a new city is being built by taking the values of a different city.

The final step of the street creation is to assign the CGA rule, *AdvancedStreet* for the street 3D model and textures. The final result can be seen in Figure 4.1b

Once the street graph is created, the blocks between the streets are subdivided according to different algorithms in CityEngine. There are three types of subdivisions; *recursive*, *offset*,

[1] https://desktop.arcgis.com/es/arcmap/10.3/manage-data/raster-and-images/esri-ascii-raster-format.htm
[2] https://qgis.org/es/site/index.html

Variables	Source	Value
Primary Street Length Mean	RWS	136 m.
Primary Street Length Standard Deviation	RWS	30 m.
Secondary Street Length Mean	RWS	70 m.
Secondary Street Length Standard Deviation	RWS	19 m.
Primary Street Width	CityEngine	12 m.
Secondary Street Width	CityEngine	8 m.
Sidewalk Width	CityEngine	1.5 m.

Table 4.1.: Details concerning the datasets used for the experiments.

(a) (b)

Figure 4.1.: (a) *boolean* control map (b) Street network with *organic* pattern for primary streets and *raster* for secondary streets

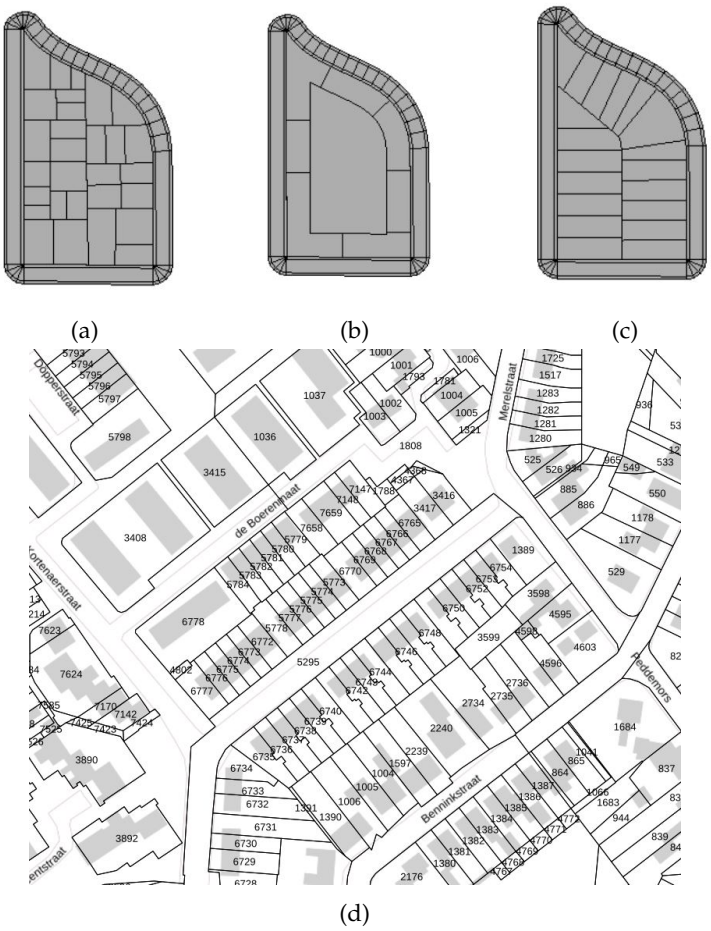


Figure 4.2.: (a) recursive subdivision creates rectangular lots by repeatedly splitting the block (b) offset subdivision creates lots only within a given distance from the street edges of the block. (c) skeleton creates street-aligned lots that always have access (d) example of parcel subdivision in real life (Haaksbergen). Images taken from [ESRI, 2022] and [PDOK, 2021].

skeleton (Figure 4.2). The best option for the parcel subdivision is the skeleton (Figure 4.2c). Nevertheless, as each parcel is modelled randomly, it is not possible to have parcels that share the same building (grey shapes in Figure 4.2). As a consequence, it is decided to work with the *offset* approach with a distance of 20 meters from the street, which is enough to have a storage or a garden at the back of the parcels.

4.1.2. Procedural Models

For the *international city* rule, the type of trees is set before generating the models. As discussed in the Chapter 3, for the customised size of 1,5 by 1,5 kilometres of the synthetic city, models such as the *realistic* and *schematic* take much space, considering that there should be around 2000 trees in the city's extent. Table 4.2 shows the size and number of points of every tree model in a GBL format.

Type	size	points
Fan	565 KB	12
Realistic	3916 KB	10332
Schematic	107 KB	3396
Poly Tree	50 KB	932
Poly Tree no leaves	60 KB	1738

Table 4.2.: Types of trees and size

For the implementation of the *Dutch City* rule, a stochastic model with similar values and shapes to the real world is created. For this purpose, the 3D BAG from the municipality of Haaksbergen is taken as CityJson [3] format.

To create the CGA rule, the buildings are divided into three groups:

- Office Building: This type of building occupies most of the area of the lot. Their roof is composed of different flat surfaces, and parking lots exist around them. This group was created in order to handle big buildings present in the built environment, such as schools, commercial venues and office buildings. This building's type is essential to learn by the model as sometimes they take the whole input training image.
- Single Buildings: The single buildings are built to imitate different types of houses without intersecting other buildings. They are mostly centred on their lot. Important features are modelled on the roof, such as dormers, solar panels and chimneys. The roof can have different shapes, and sometimes they contain a garage and a storage room at the back of the main building.
- Grouped Buildings: In a significant part of the Netherlands, one building is divided into different apartments (see Figure 4.2d). They can contain different types of roofs and structures above them, and the area of these buildings is bigger than the single ones. These types of buildings are also referred as terraced houses.

To divide these buildings, some assumptions are made. First, because the office buildings are mostly flat and have a big area, those buildings in which their area is higher than the third quartile and present a flat roof, are assigned as *office* buildings. Then, for the grouped buildings, those higher than the median and have slanted roofs and flats between the mean and the third quarter are taken. Flat and slanted roofs below the median are considered grouped buildings.

In Python, the height information is retrieved from the difference of *h_dak_50p*, which is the median elevation above sea level from all roof parts, and the *h_Smaaiveld* that is the elevation above sea level at the ground level of the building. Then, the roof angle is depicted by computing the normal of each roof plane. Depending on the angle, the buildings are divided into two types, flat roof if the angle is below 10° and slanted roof if the building has a roof angle above 10°. For all the variables, the mean and the standard deviation are the values used to make the stochastic rule model. Table 4.3 shows the three types of buildings. The percentage of appearance of each type of building is according to a weighted ratio between the roof area and the number of buildings.

[3] https://www.cityjson.org/

(a) (b)

Figure 4.3.: (a) aerial view of the three types of buildings On the left is the office building; center is single buildings; right grouped building (b) aerial image of the three types of buildings.

Type of Buildings	Type of Roof	Angle (°)	Height (m)	Weighted of total (%)
Office	Flat	0	5.8 $\pm$2.0	38
Single	Flat	0	2.6 $\pm$3.8	7
Single	Slanted	32.5 $\pm$9.3	5.4 $\pm$1.8	14
Grouped	Flat	0	6.0 $\pm$1.2	7
Grouped	Slanted	36.9 $\pm$11.0	5.7 $\pm$1.4	48

Table 4.3.: Stochastic values for *Dutch City.cga*

4.2. Rendering Synthetic Training Dataset

When the model in CityEngine is exported, it is then opened in Blender [4] which is an open-source 3D suite with a Python pipeline for automation. Having the synthetic city in Blender, the first step is to label each part after the category they belong to. As the whole city is a unique mesh, it has to be divided into different parts. Blender has two ways to implement this; division by material and parts. The division by material works for the goal of assigning a class to each part because all the textures in the design of the synthetic city are stored by category in CityEngine. After each part is divided, the class is assigned as an attribute. The assignment of the class consists of first creating a *database* to store the name and class of every texture. In this step, Blender reads from the *database* and, depending on the texture, the class is assigned. The class depends on the folder where it is stored when using CityEngine. If the user wants to add a new texture, it should be stored accordingly with the class.

Since each part of the model is assigned to a category, the next step is to render all three types of images. The first phase is to set the camera positions in the virtual world. For this, we simulate a real trajectory of an aeroplane. Figure 4.4 shows the planning for the cameras in the synthetic city. The camera's locations are set not to have empty spaces on the city's borders. In addition, there is an overlap of 30% for every image as the actual flights are doing the same to recreate pose estimations with photogrammetry techniques. The camera resolution is 0.1, which is taken from the real world imagery used for training and testing.

Figure 4.4.: Camera location planning. Every orange dot is a location of one camera in the extent of the synthetic city.

In the rendering process, the lighting conditions are set according to the real scenario. Parameters such as the sun intensity and the sun angle can also be set. Furthermore, when the

[4] https://www.blender.org/

Segmentation Class	BGT class
Building	Gebouw installatie
	Overig bouwwerk
	Pand
Road	Overbruggingsdeel
	Wegdeel
Other	Waterdeel
	Kunstwerkdeel
	Onbegroeid terreindeel
	Ondersteunend waterdeel
	Ondersteunend wegdeel
	Openbare ruimte

Table 4.4.: BGT class layers to make the ground truth. Adapted from [Mulder, 2020]

camera locations and parameters are set, the rendering starts using BlenderProc [5] procedural pipeline, which generates real looking images for training CNNs from a scene in Blender using different approaches such as labelling, depth, normal and pose estimation. First, the RGB images are rendered with a ray-tracing production-based engine called *cycles*, used widely in Blender. Secondly is the depth rendering, where the DSM of the synthetic world is rendered. The third step is the labelled image renderization, where each class is rendered with one colour.

4.3. Real Datasets

4.3.1. Basisregistratie Grootschalige Topografie BGT

The BGT is used as the ground truth data for the real training data and the inference imagery. The dataset is a base map of the Netherlands with the location of physical objects such as buildings, roads, water, railway lines and agricultural areas [PDOK, 2021]. Following the steps of Mulder [2020], the layers used for making the ground truth are in Table 4.4.

The dataset presents some errors on the boundaries of the classes, especially in buildings; also, some layers are wrongly classified [Mulder, 2020]. Nevertheless, for the area of Haaksbergen and Haarlem, the quality is enough to perform segmentation with *buildings*, *roads* and *other* classes. The methodology of [Mulder, 2020] was used to create the mask layer. First, the layers are merged in QGIS 3.14, assigning the name of the initial layer as an attribute. Then, with a field calculator, the layers were re-classified to the ones used for classification. Herewith, rasterisation is performed with a resolution of 8cm. Then, mask layer was clipped to the wanted extents [Mulder, 2020].

[5] https://github.com/DLR-RM/BlenderProc

4.3.2. True Orthophotos and Digital Surface Model from Netherlands

In the current work, TOs from the cities of Haaksbergen and Haarlem are used. The images are provided by READAR and have a pixel size of 8 and 10 cm. The 8 cm resolution images are then scaled to 10 cm for consistency. An area of 1km square is used for training, and another area of the same size, for testing. For the case of Haarlem, these images are taken from an area of 2 km. the same area used by [Mulder, 2020]. The TO is provided by RADAR, and is created from stereo imagery using a deep learning model for dense matching. For colouring the parts that are occluded in the stereo imagery, an interpolation is applied. The TO is an RGB image with the addition of an extra boolean band indicating if the pixel was interpolated.

Furthermore, for the DSM, as a result of the stereo matching algorithm, a point cloud is obtained. These points represent the height of the pixel, which is later converted into a DSM. To be able to use the model in cross-domain scenarios, the DSM is normalised by taking the mean of the height values of the class *road* using the ground truth created with the BGT.

4.3.3. ISPRS Potsdam dataset

The 2D semantic labelling challenge by ISPRS [2020] contains a mosaic of TO, the ground truth and DSM from the city of Potsdam, Germany. The resolution of the TO is 5cm, which is later resized to 10cm using GDAL [6] which is a translator library for raster and vector data. The DSM is generated by the Trimble INPHO 5.6 software and Trimble INPHO OrthoVista [7] to correct the relief displacement of the orthophoto. The occlusion parts from the relief displacement are also interpolated. Similarly to the DSM from the Netherlands, Potsdam's DSM is normalised by filtering the ground height for each pixel.

Moreover, the ground truth consists of six categories; impervious surfaces, buildings, low vegetation, tree, car and background. Because this study uses only three classes, all the classes are re-classified to *building*, *road* and *other*.

4.4. FuseNet Implementation

As the scope of this research work is to test and analyse the quality of training data in segmentation models, the only model used in this thesis is FuseNet with a fusion of height values, as it was the best performing model in [Mulder, 2020]. The model was given by READAR and uses the open source machine learning framework PyTorch [8]. To encode the inputs and outputs of the model, PyTorch uses Tensors, a data structure similar to an array structure in Numpy library but with the ability to run on GPU's. The model is adapted to work with spatial data to extract the exact location of the input data [Mulder, 2020]. With the use of PyTorch, sufficient computational power is needed. The models are trained using a single GPU (rtx2080ti 12 GB). The model is ran in a Docker container [9]. A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.

[6] https://gdal.org/
[7] https://geospatial.trimble.com/products-and-solutions/trimble-inpho
[8] https://pytorch.org/
[9] https://www.docker.com/

Parameters	Value
Loss Function	Weighted Cross Entropy
Optimizer	Adam
Optimizer Learning Rate	0.0001
Batch Size	4
Number of epochs of no improvement	20

Table 4.5.: Parameters of FuseNet model

Training and inference of the model work in three steps. The first step is the preparation of the training data. The training data is split into training and validation data in this step. Validation data is used to tune the hyperparameters and to assess the model during training to prevent overfitting of the model [Mulder, 2020]. Then, the inverse weights are calculated and incorporated into the weighted cross-entropy loss function. Furthermore, the training step begins by selecting the parameters in Table 4.5. It is important to mention that these parameters are selected according to Mulder [2020] as they bring the best performing results for the study case. These parameters are not changed in the tests of this work to be able to analyse only the influence of the training data.

Once the training step has 20 consecutive epochs without improving the mIoU, the five best performing models are saved for later use in the inference step. This step is to process the model in unseen imagery (i.e.testing imagery). The inference also is based according to Mulder [2020] overlap inference strategy. This strategy is based on cutting the testing image into overlapping tiles. Next, make the inference for each tile and remove the outer overlapping part to eliminate border inconsistencies and finally merge all the cutting tiles. Is important to mention that the inference is an average of the five best performance models during the training process for robustness purposes.

4.5. Domain Adaptation

The first approach of domain adaptation is CORAL. It is implemented in Python using Numpy arrays. First, the real and synthetic images are imported as an array with a size of 512 by 512 pixels for each image. That means for a dataset of 756 images, 198 million pixels are used to compute the covariance. The computer crashes due to the amount of data processing when the covariance is calculated with Numpy *cov*. Schubert [2018] compares different methods on stable computational algorithms for the covariance. One of them is the *two-pass* algorithm that consists of computing first the two sample means and then the covariance.

CORAL algorithm (Algorithm 3.1) is performed after the calculation of the covariance of the real dataset.

The second approach is CORAL by classes. The implementation of this method was made in Python using the ground truth and the real image dataset. First, we take the location of one class in the ground truth and extract the positions of this class. Then, the RGB values are depicted according to the location of the ground truth. When this step is made with all the classes, the covariance is calculated for each class in both real and synthetic images. Next, CORAL is performed for each class and finally every transformed class is located back to the original image positions according to the ground truth.

CycleGan implementation is downloaded from Github [10] as a PyTorch implementation that enables to work with different types of images. Then, the real and synthetic datasets are used as input for the model. The model has its own Docker file which enables the download of all the requirements in a controlled environment.

Finally, Cycada implementation was downloaded from JoliBrain Github [11]. JoliBrain is a group of veterans of software development with open-source implementations for different models of Deep Learning. For running the Cycada model, the input is the ground truth and RGB images from both real and synthetic datasets. The parameters are tweaked, especially for the *lambda* parameter, which is the tradeoff between the reconstruction loss and the visual style loss [Hoffman et al., 2018]. The final *lambda* is set to 5, which gives the best reconstruction.

4.6. Evaluation

The evaluation process starts with the inferred image and the ground truth. The performance measures are computed in Python with the library scikit-learn [12] by computing the confusion matrix. On the other hand, the visual assessment is done with the help of QGIS.

[10] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix
[11] https://github.com/jolibrain/joliGAN
[12] https://scikit-learn.org

5. Results and Analysis

In this chapter, the results of the experiments are presented. First, the results between the two different CGA rules for buildings are presented. The second is a comparison between different road patterns, followed by a comparison of different types of tree models. Then, the tests between different quantities of images and the mix with real training data are presented. Next, a comparison with related work is presented, and finally, the results of DA techniques are shown.

5.1. Building Models Experiment

The first dataset tested contains the default settings of CityEngine with the rule *International Cities* and with the default settings of height values (Figure 5.1). The test is performed in the city of Haaksbergen. The mIoU and average F1 scores are presented in Table 5.1, and the results by class are presented in Table 5.2. In addition, a visual assessment of the results is important to get the first ideas of how the synthetic data works for semantic segmentation of aerial images (Figure 5.2c).

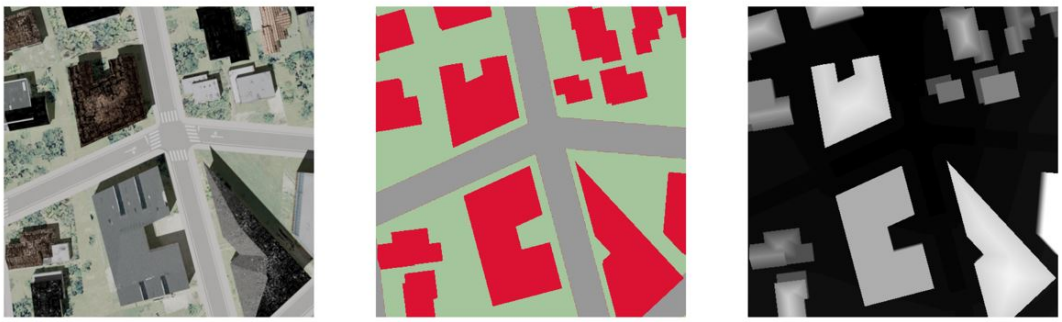


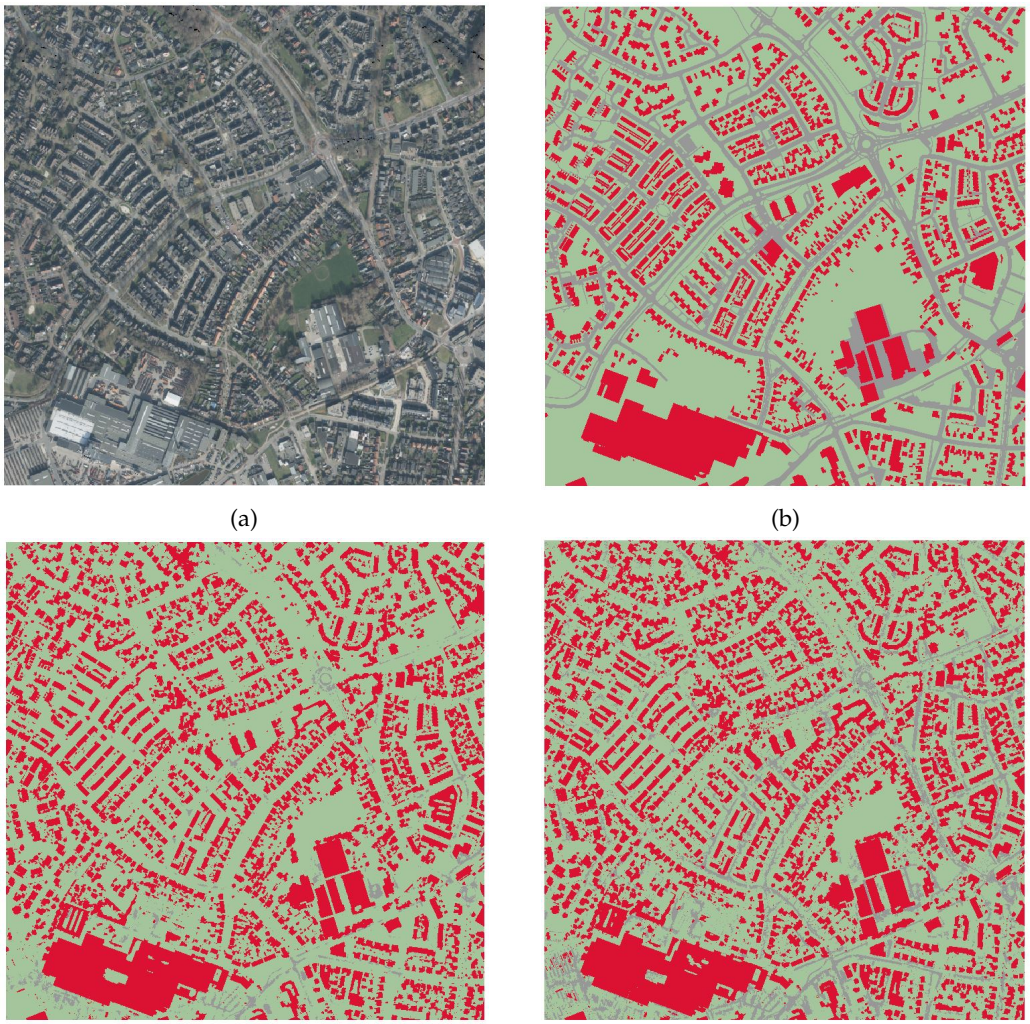
Figure 5.1.: Sample Image from the default settings of CityEngine. Left: RGB image; center: segmented image; Right depth image. Red = Building, Gray = Road and Green = Other.

Building Rule	mIoU	F1
***International City* (Default City Engine)**	0.40	**0.52**
***Dutch City* (Own)**	**0.46**	**0.52**

Table 5.1.: Global results for *International City* and *Dutch City* predicting in the city of Haaksbergen

	building			road			other		
	IoU	prec.	recall	IoU	prec.	recall	IoU	prec.	recall
International City	0.57	**0.62**	0.88	0.04	0.55	0.08	0.59	0.69	**0.80**
Dutch City	**0.58**	**0.62**	**0.90**	**0.20**	**0.60**	**0.23**	0.59	**0.73**	0.76

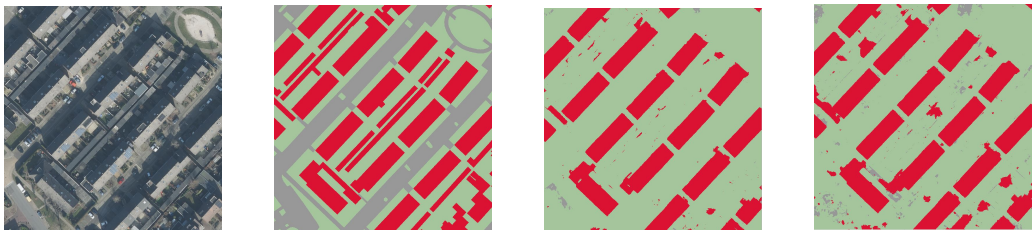
Table 5.2.: Results for *Dutch City* predicting in the city of Haaksbergen



(a) (b)

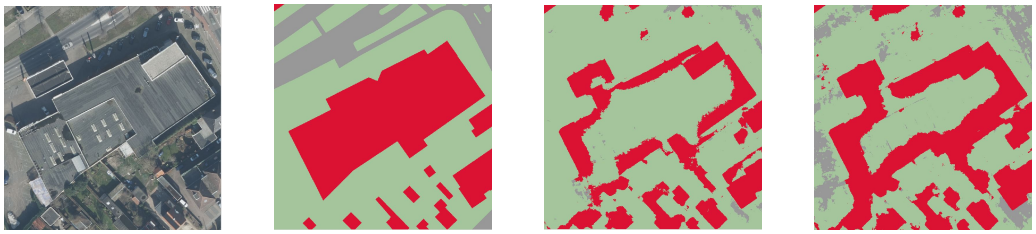
(c) (d)

Figure 5.2.: (a) TO test image of Haaksbergen (b) Ground Truth from BGT (c) *International city* prediction (d) *Dutch City* prediction. Red = Building, Gray = Road and Green = Other.



(a) (b) (c) (d)

Figure 5.3.: (a) TO small building example (b) Ground Truth (c) *International city* prediction (d) *Dutch City* prediction. It is seen that both approaches miss the small buildings in the backyards of the houses. Red = Building, Gray = Road and Green = Other.



(a) (b) (c) (d)

Figure 5.4.: (a) TO big building example (b) Ground Truth (c) *International city* prediction (d) *Dutch City* prediction. It is seen that *Dutch city* predict more pixels in big buildings than *international city* . Red = Building, Gray = Road and Green = Other.

First, the model detects buildings better than roads due to the height difference and the well-defined edges. Different objects on the roads, such as trees, cars and shades, make road detection more difficult. The model can detect 88% of the *building* pixels. Nevertheless, it is not detecting small buildings and some big buildings. Small buildings with grey-ish textures are not detected as it confuses *building* pixels as *road* or *other* because the lower difference of height and the similarity of colour (Figure 5.3). On the other hand, big buildings are not detected because they have multiple levels with different heights and textures (Figure 5.4), which confuses the model. Missing these types of buildings affect the intersection between the prediction and the ground truth, reducing the IoU of *building* class. Moreover, the IoU and the precision are considerably low as the model detects the trees as buildings because they have height values greater than the ground level (Figure 5.5).

Furthermore, the model confuses *road* with *other*. This problem could be because the ground and streets have the same height values. As a consequence, the values of IoU for *other* are low as the union increases with detecting *road* pixels as *other*.

In order to detect smaller buildings and more prominent buildings with multiple floors, the model is trained with the *Dutch city* CGA rule with the same road structure as *international city* rule but with different building model specifications.

The results of *Dutch city* are also presented in Table 5.1 and Table 5.2. Despite having more detailed buildings and specifications of the real environment, the IoU of *building* increases only 1 Percentage Points (p.p.). This increment is because it detects more pixels in the big buildings, which are not detected in *international city* (Figure 5.4). Is believed that that is caused because

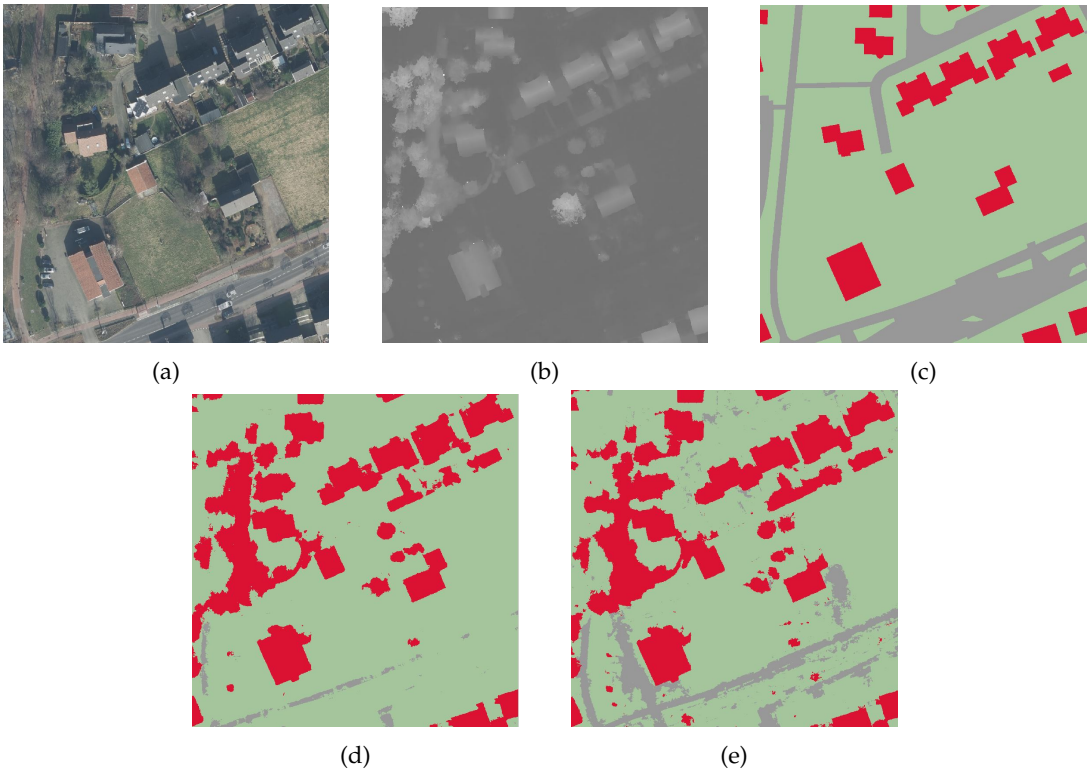


Figure 5.5.: (a) TO with trees present (b) DSM (c) Ground Truth (d) *International city* prediction (e) *Dutch City* prediction. It is seen that both *Dutch city international city* are predicting trees as buildings due the height presence is trees. Red = Building, Gray = Road and Green = Other.

	mIoU	F1
Organic	0.42	0.59
Radial	0.35	0.61
Raster	**0.44**	**0.63**
Organic Raster	0.39	0.57

Table 5.3.: Global Results for different road patterns

	building			road			other		
	IoU	prec.	recall	IoU	prec.	recall	IoU	prec.	recall
Organic	0.45	**0.55**	0.71	0.28	0.44	0.43	**0.54**	0.74	**0.66**
Radial	0.46	0.47	**0.99**	**0.36**	0.41	**0.74**	0.22	**0.82**	0.24
Raster	**0.49**	0.54	0.86	**0.36**	**0.49**	0.58	0.48	0.78	0.55
Organic Raster	0.43	0.52	0.72	0.30	0.40	0.56	0.43	0.55	0.60

Table 5.4.: Results by class for different road patterns

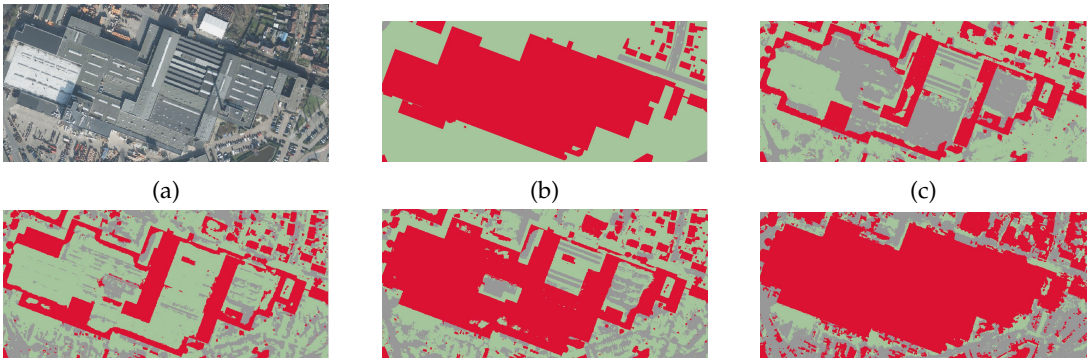
dutch city follows to height distribution of the real context which enables the model to detect more *building* pixels. In contrast, the model also fails to detect small buildings. The precision is still the same as the *Dutch city* is also incorrectly detecting trees as buildings.

For the mIoU and F1 score, the differences are higher as the *Dutch city* learns better to classify roads. It is believed that this is since the *Dutch city* has more roof details such as the dormers, chimneys and solar panels that could make the model more sensitive to changes in height values as the difference of height between the roads and the ground truth are minimal (0.2 m. in both real and synthetic domain). On the other hand, despite that models being trained three times to consider the model's randomness, the results persisted. Finally, for the mIoU and F1 score is clear the ability of the synthetic data to classify buildings as it is presenting a recall of 0.9. More details about the synthetic road behaviour are presented in the following section.

5.2. Road Experiments

In this section, different types of road patterns are tested to see the influence of road shapes on the model. The textures of the roads and the building modelling rule are the same in all experiments. In addition, tree models are not present. In this experiment we use four types of *road* patterns explained in Chapter 4. Table 5.3 presents the global results, and in Table 5.4, the results by class. Figure A.3 shows the visual results.

For the global results, it is believed that none of the road patterns has sufficient ability to classify properly road shapes in the real domain. Values of IoU are below 0.36 for all cases. However, there are some differences between the results of each experiment. Most notably, the *raster* pattern presents the best results. The main difference between this pattern and the others is the number of intersections and that most of these intersections have an angle of 90 degrees. Since the pattern is the only variable that is different in these experiments, it is believed that the model could focus more on detecting defined shapes of roads than the other



(a) (b) (c)

(d) (e) (f)

Figure 5.6.: The different patterns detect differently a big building with multiple levels (a) TO (b) Ground Truth (c) *Organic* pattern (d) *Organic Raster* pattern (e) *Raster* pattern (f) *Radial* pattern. Red = Building, Gray = Road and Green = Other.

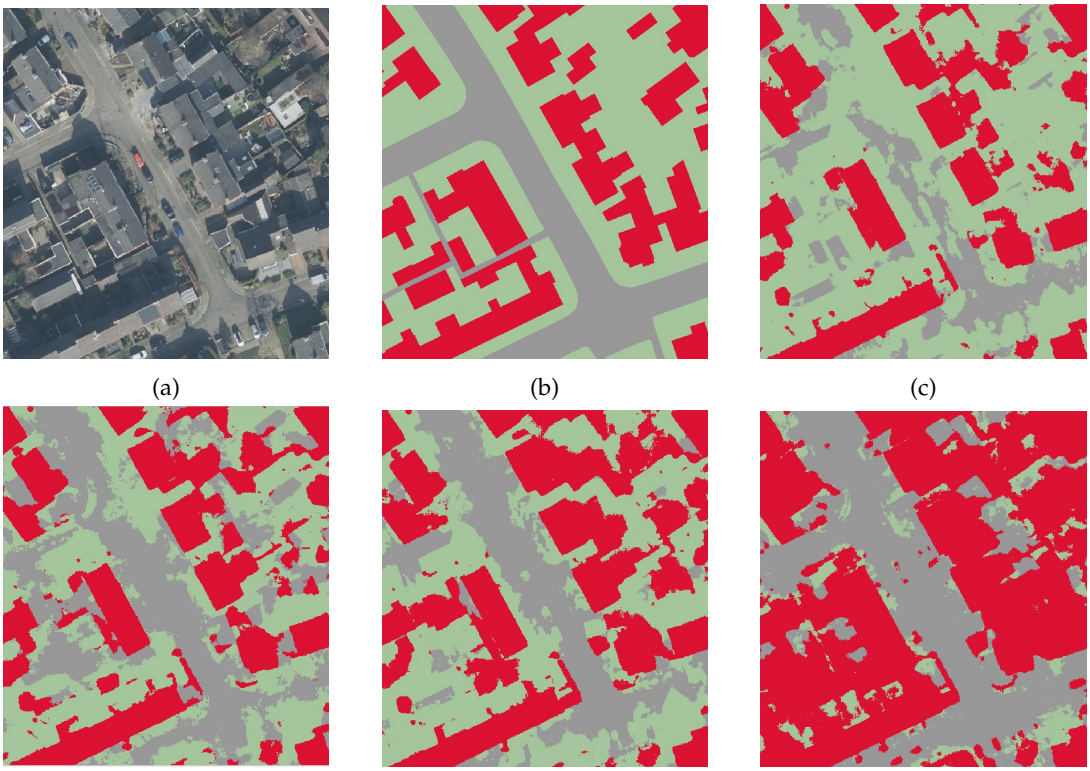
patterns. As a single model is used of all experiments, the ability to detect the roads affects both the *buildings* and *other* classes.

In more detail, the best pattern to detect the class *road* is the *raster* with an IoU of 0.36. The *radial* approach has the same value, but it has more incorrect predictions of road class, lowering the precision. In the case of the *radial* pattern, the model is believed to be very sensitive to heights because every object with a height greater than the ground height is classified as *building* and the rest as *road* with some clear green textures that are classified as *other*. As a result, the recall of *radial* pattern for *building* is approximately 1.0 and for *road* is higher than the other patterns. *Organic* pattern seems to be very influenced by the texture of roads, labelling more *building* pixels as *road* or *other* as it is seen in most of big buildings (Figure 5.6c). The *organic and raster* pattern detects more of these pixels, but the *raster* pattern seems to be more sensitive to height than the other two, detecting almost all of the area of the big buildings. A model that is sensitive to heights will detect more trees as buildings.

Moreover, road shapes in the ground truth have a well-defined shape of a segment. Nevertheless, in the TO, they have cars, building or tree shades that make changes in the shape and height of the roads. When training with synthetic cities, only shades of buildings are present. As a consequence, the model occludes cars and objects present on the roads, labelling it as *building* or *other* (Figure 5.7). The ability of the synthetic model to learn different textures of roads is limited since it uses some textures simulating the grey pavement. However, the *raster* pattern can learn to detect red pavement and other colours, mainly because the model focus more on learning shapes and edges that defines a road (Figure 5.8).

5.3. Tree Experiments

Although the trees do not belong to any of the classes for this research, they are essential for detecting other classes. For instance, as seen in previous experiments, models which are sensitive to height values may confuse trees with buildings. Modelling trees may help the model to detect irregular shapes with height values as *other* instead of *building*. As a result, the precision should increase, and the union between prediction and ground truth should



(a) (b) (c)

(d) (e) (f)

Figure 5.7.: Road prediction when cars are present on the roads (a) TO (b) Ground Truth (c) *Organic* pattern (d) *Organic Raster* pattern (e) *Raster* pattern (f) *Radial* pattern. Red = Building, Gray = Road and Green = Other.

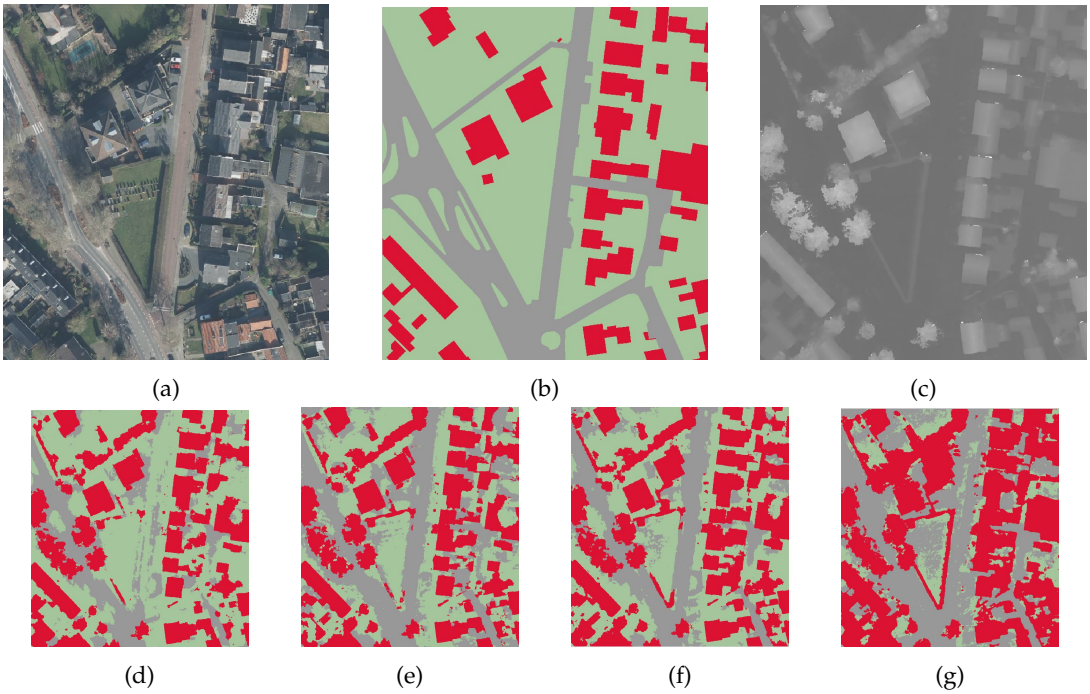


Figure 5.8.: Road prediction when coloured road textures (a) TO (b) Ground Truth (c) DSM (d) *Organic* pattern (e) *Organic Raster* pattern (f) *Raster* pattern (g) *Radial* pattern. Red = Building, Gray = Road and Green = Other.

	mIoU	F1
No Trees	**0.39**	0.57
Poly Tree	**0.39**	0.57
Fan Tree	0.30	0.53
Realistic Tree	0.38	**0.61**

Table 5.5.: Results for different types of trees

	building			road			other		
	IoU	prec.	recall	IoU	prec.	recall	IoU	prec.	recall
No Trees	0.43	0.52	0.72	**0.30**	**0.40**	0.56	**0.43**	0.55	**0.60**
Poly Tree	0.43	0.54	0.69	**0.30**	0.38	0.59	0.42	0.72	0.51
Fan Tree	0.40	**0.67**	0.50	0.22	0.24	0.71	0.29	0.75	0.33
Realistic Tree	**0.57**	0.66	**0.81**	0.29	0.31	**0.79**	0.29	**0.80**	0.31

Table 5.6.: Results by class for different types of trees

decrease to improve the IoU for *building* class. Three types of trees are tested, plus a synthetic dataset without any trees for comparison. The visual results can be found in Figure A.4.

For the global results (Table 5.5), the mIoU is equal for *no trees*, *poly trees* and *realistic trees*, and similar F1 scores with *realistic* approach being the best result as the precision and recall for each class is higher. Mainly, because the recall of the class *building* is significantly higher (0.81) than the other approaches. Whereas *no trees* and *poly tree* are classifying more precisely the class *road*, the *fan trees* and *realistic trees* models have a higher precision in the class *other*.

In more detail, the effect of the trees on the model results is significant. First, the model without trees appears to be sensitive to height values, labelling tree pixels and cars as *building* (Figure 5.9d). Second, not only having irregular shapes with height values works for detecting trees as *poly tree* approach cannot help to classify trees correctly. Also, because of the time when the TOs were taken, the trees are not green (Figure 5.9e). In other words, the texture of trees is as important as the height values. Third, the *fan* approach, which is two intersecting planes, helps to detect green fences with regular shapes or canopies with continuous foliage of trees. However, the sensitivity of height is decreased, affecting the detection of buildings (Figure 5.9f). Finally, the best approach is with the *realistic* trees as it helps the model to detect the trees as *other* with an IoU of 0.57. Nevertheless, some drawbacks appear with the addition of trees, as the model starts only detecting high vegetation as *other* and lower vegetation as *roads*, decreasing the IoU of this class and affecting the mIoU taking it to similar levels as the other approaches (Figure 5.9g).

5.4. Quantity of Images

To study the ability of the CNN to perform semantic segmentation of aerial images with synthetic images as training data, the learning curve of the model is investigated (Figure A.9). For this, different quantities of training data are tested with the same design settings of the virtual world. Beforehand, it is known that because the maximum variability of scenes that

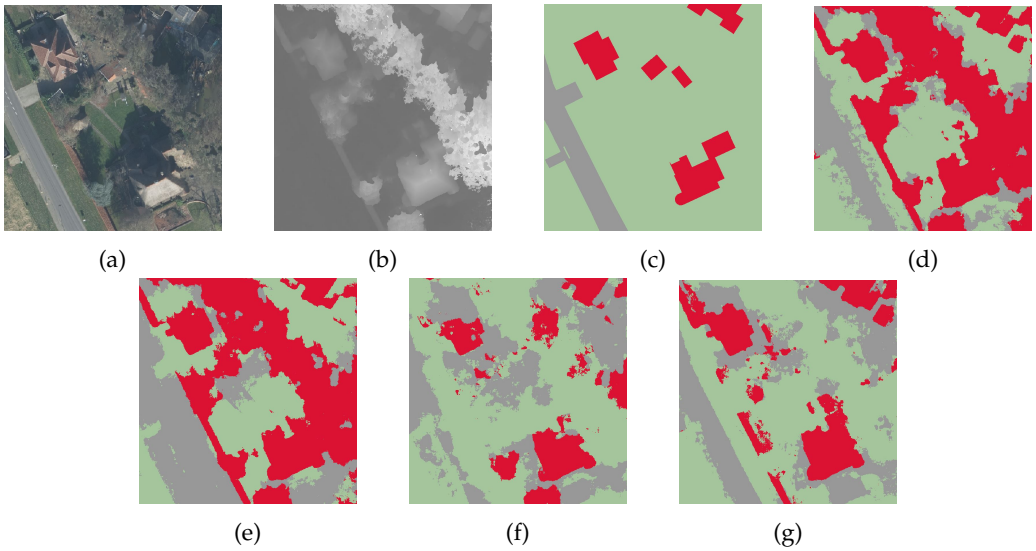


Figure 5.9.: (a) TO (b) DSM (c) Ground Truth (d) *no trees* (e) *poly trees* (f) *fan trees* (g) *realistic* augmentation. Tree detection when a canopy of trees is present. Also a green fence is present. Red = Building, Gray = Road and Green = Other.

# Images	mIoU	F1
188	0.31	0.46
376	0.44	0.62
756	0.45	0.61
1512	**0.48**	**0.65**
3024	0.46	0.64

Table 5.7.: Results for different quantities of synthetic images

are generated as synthetic images is limited, the model will arrive at a certain limit that the evaluation measures will be constant as it has learnt the whole domain of the synthetic images. In Table 5.7 the global results for every dataset are presented. In Table 5.8 the values per class are presented and in Figure A.5 the visual prediction results are presented.

The results suggest that the synthetic images converge between 756 and 1512 images with approximate maximum mIoU of 0.50. The model cannot learn more visual cues to perform semantic segmentation at this quantity of images. The reason could be the limited samples of textures and shapes generated in the synthetic city. In the IoU for *building* class, with the addition of more images, more pixels are classified as *building*, increasing the recall. However, on the contrary, the precision decreases. This phenomena suggests that the addition of more images, enables the model correctly classify *building* but also classify *other* and *road* pixels as *building* such as trees, ultimately keeping the IoU constant.

	building			road			other		
# Image	IoU	prec.	recall	IoU	prec.	recall	IoU	prec.	recall
188	0.47	0.61	0.67	0.24	0.26	**0.76**	0.21	0.71	0.22
378	0.52	**0.63**	0.76	0.31	0.40	0.56	0.51	**0.76**	0.60
756	**0.56**	0.62	**0.84**	0.26	0.41	0.42	0.53	0.75	0.65
1512	0.52	0.60	0.81	**0.33**	0.58	0.43	**0.57**	0.75	**0.71**
3024	0.50	0.56	**0.84**	0.32	**0.60**	0.40	0.56	0.75	0.69

Table 5.8.: Results by class for different quantities of synthetic images

Ratio	# Images	mIoU	F1
100% Real	1444	0.75	**0.86**
100% Real	722	0.75	**0.86**
100% Real	288	0.72	0.84
100% Real	144	0.70	0.82
20% Synthetic - 80% Real	1444	**0.76**	**0.86**
50% Synthetic - 50% Real	1444	0.75	**0.86**
80% Synthetic - 20% Real	1444	0.71	0.83
90% Synthetic - 10% Real	1444	0.70	0.81
100% Synthetic	1444	0.35	0.51

Table 5.9.: Results for different ratios between Synthetic and Real training data

5.5. Training with a mix of Real and Synthetic Training data

Additional real data can be used as a booster, primarily when using synthetic data to close the domain gap. This section presents the results when training with a mix of real and synthetic data with different ratios (Table 5.9 and Figure A.6). In addition, some analysis of the quantity of real data is presented.

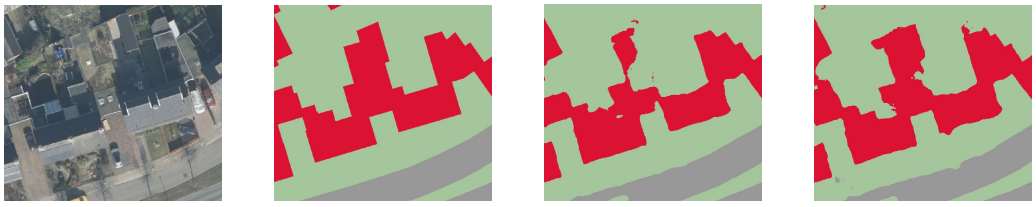
The results show that for a fixed number of real images, adding synthetic data does not either improve or decrease the mIoU and F1 average score. When taking a model trained on synthetic data as basis, adding real images shows a sharp increase in performance. However, training with the same amount of real images only gives similar results. Moreover, with 20% of synthetic data, the mIoU improves by 1%, mainly because the recall for the class *other* is a little better in areas of pavement textures that are not labelled as *road*.

The real data seems to converge at approximately 700 images, which could be why the synthetic data is not helping to improve the model. Despite the real training data coming from an uncleaned version of the BGT, the model can detect new buildings that are not included. Moreover, real training data fails to detect buildings with green roofs because the original roof texture is lost (Figure 5.10). On the other hand, small buildings, mainly located at the back of the properties, are also not detected by the model. In these cases, the visual results suggest that synthetic data helps detect green roofs as *buildings*. For the second case, the synthetic data does not seem to help.

When taking as a reference training with 144 real images, the improvement of adding more images is 5 p.p. when training with 1444. This indicates that the model is able to learn with a

	building			road			other		
# Image	IoU	prec.	recall	IoU	prec.	recall	IoU	prec.	recall
100% Real 1444 Im.	**0.82**	**0.87**	0.93	0.66	0.73	**0.87**	**0.79**	**0.93**	0.84
100% Real 722 Im.	0.81	0.86	0.93	0.65	**0.75**	0.83	**0.79**	0.91	**0.85**
100% Real 288 Im.	0.79	0.85	0.93	0.61	0.74	0.77	0.77	0.89	**0.85**
100% Real 144 Im.	0.79	0.84	0.93	0.58	0.74	0.73	0.76	0.88	**0.85**
20% Synthetic - 80% Real 1444 Im.	0.81	**0.87**	0.93	**0.67**	**0.75**	0.86	**0.79**	0.92	**0.85**
50% Synthetic - 50% Real 1444 Im.	0.81	0.87	0.93	0.65	0.74	0.84	**0.79**	0.92	**0.85**
80% Synthetic - 20% Real 1444 Im.	0.79	0.85	0.91	0.60	0.73	0.77	0.76	0.88	**0.85**
90% Synthetic - 10% Real 1444 Im.	0.75	0.84	0.84	0.58	0.69	0.74	0.75	0.87	**0.85**
100% Synthetic 1444 Im.	0.49	0.50	**0.99**	0.39	0.44	0.75	0.31	0.85	0.33

Table 5.10.: Results by class for different ratios of synthetic and real images



(a) (b) (c) (d)

Figure 5.10.: (a) TO green roof (b) Ground Truth (c) Prediction with 100% real training data (d) Prediction with 50% real training data and 50% synthetic data. Synthetic data seems to help real data to detect green roofs. Red = Building, Gray = Road and Green = Other.

small number of images the main characteristics of the data to perform semantic segmentation. In more detail, the addition of more real data only significantly improves the *road*, which could be due to the variability of shapes that this class has.

Furthermore, in the case of roads, both the real and the synthetic training datasets are sensitive to pavement textures. As a consequence, pavement inside properties is classified as *road*, but in BGT the pavement inside the properties is classify as *other* decreasing the IoU in *roads*. In practice, these areas should be classified as *roads*. Hence, the IoU should be higher.

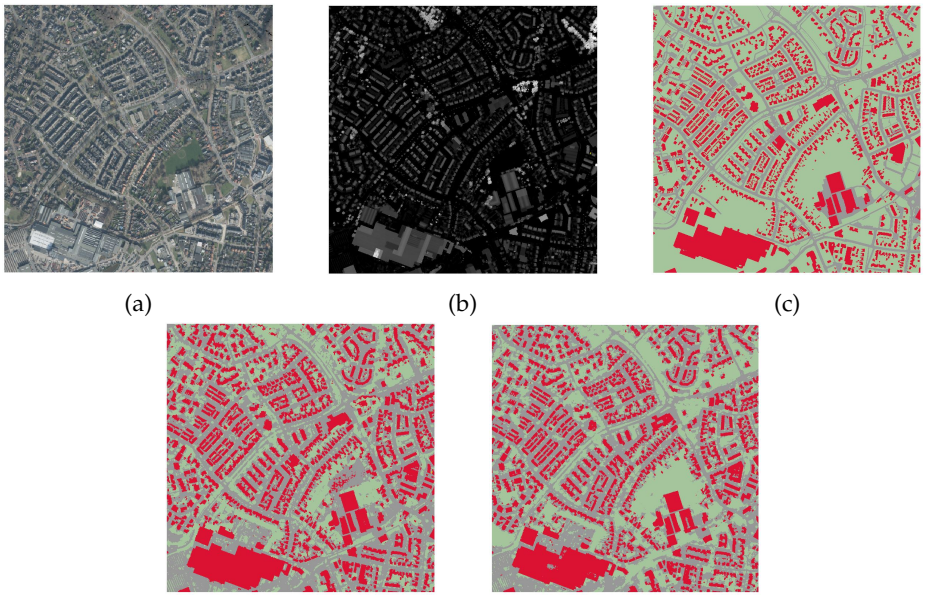
5.6. Cross-Domain Inference

This section uses the results of inference with cross-domain scenarios where different geographic areas are taken in the training and testing data. In Figure 5.11, Figure 5.12 and Table 5.11 the results for different scenarios are presented.

Synthetic data improves semantic segmentation when the training and the testing data are from different areas. In both cases, Potsdam - Haaksbergen and Haaksbergen - Potsdam, the results of mIoU improve 4 p.p. and 8 p.p., respectively. In the case of training in Potsdam and testing in Haaksbergen, the model confuses some trees with buildings because of the height values (Figure 5.13). On the other hand, for the case of Haaksbergen and Potsdam for training and testing, respectively, the Synthetic data helps the model detect more structures like dormers and other different textures not present in the Haaksbergen dataset. Visually, the texture variety of Potsdam seems more exhaustive than the one in Haaksbergen. Hence, the results are better when trained on Potsdam and test in Haaksbergen than vice versa. In addition, the synthetic city helps to add more variety of textures which the model can learn, helping in the generalisation of it (Figure 5.14). The domain generalisation could potentially work for roads as the amount of textures in Haaksbergen are less varied than in Potsdam, which makes the real training data from Haaksbergen fail in detecting roads.

Synthetic data improves the results in cross-domain scenarios, and the improvement depends on how big is the domain difference. In scenarios in which the domain difference is big (i.e. Haaksbergen to Potsdam), the improvement is double that in the scenario where the difference is smaller. In addition, the average F1 score also improves considerably, 0.7 p.p. and 0.5 p.p. for Potsdam to Haaksbergen and for Haaksbergen to Potsdam, respectively.

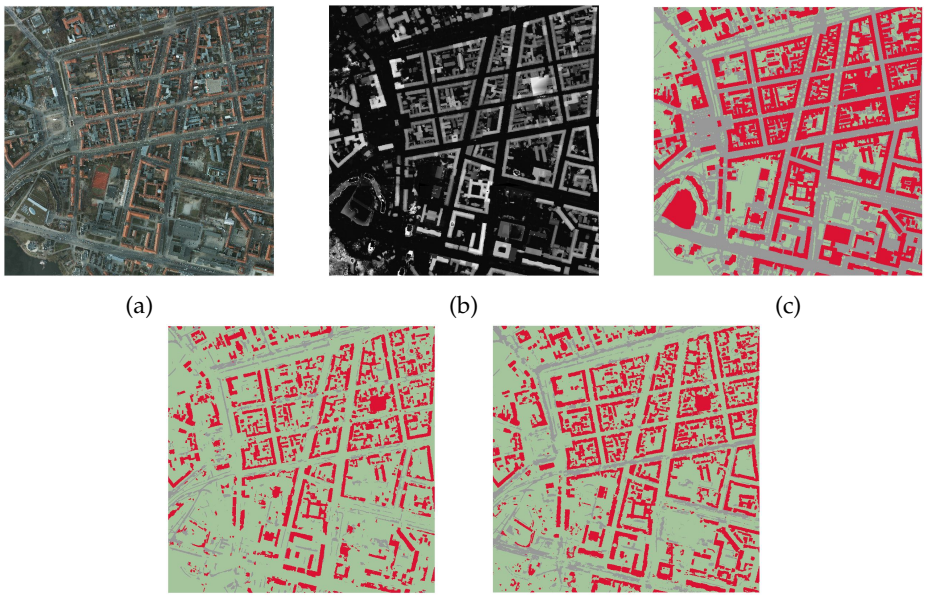
In more detail, for classes *road* and *other* , the reduced performance is primarily due to the definitions of what is a road and what is other. For instance, a road in Potsdam can be a



(a) (b) (c)

(d) (e)

Figure 5.11.: (a) Haaksbergen TO (b) Haaksbergen DSM (c) Haaksbergen Ground Truth (d) Prediction with Potsdam training data (e) Prediction with training 50% Potsdam and 50% synthetic. It is seen that the addition of synthetic data improves the model results.



(a) (b) (c)

(d) (e)

Figure 5.12.: (a) Potsdam TO (b) Potsdam DSM (c) Potsdam Ground Truth (d) Prediction with Haaksbergen training data (e) Prediction with training 50% Haaksbergen and 50% Synthetic data. It is seen that the addition of synthetic data improves the model results. Red = Building, Gray = Road and Green = Other.

Training	Images	Test	mIoU	F1
Potsdam	1156	**Haaksbergen**	0.45	0.61
50% Potsdam - 50% Synthetic	1156	**Haaksbergen**	**0.49**	**0.68**
Haaksbergen	1444	**Potsdam**	0.34	0.63
50% Haaksbergen - 50% Synthetic	1444	**Potsdam**	**0.42**	**0.68**

Table 5.11.: Results for cross-domain scenarios

	building			road			other		
Scenario	IoU	prec.	recall	IoU	prec.	recall	IoU	prec.	recall
Potsdam → Haaksbergen	0.62	0.64	**0.97**	**0.35**	**0.39**	**0.77**	0.39	**0.89**	0.41
50% Potsdam - 50% Synthetic → Haaksbergen	**0.75**	**0.79**	0.94	0.31	0.34	0.75	**0.40**	0.83	**0.43**
Haaksbergen → Potsdam	0.54	**0.96**	0.56	0.13	0.89	0.13	0.35	0.35	**0.98**
50% Haaksbergen - 50% Synthetic → Potsdam	**0.66**	0.95	**0.69**	**0.20**	**0.93**	**0.20**	**0.38**	**0.39**	0.96

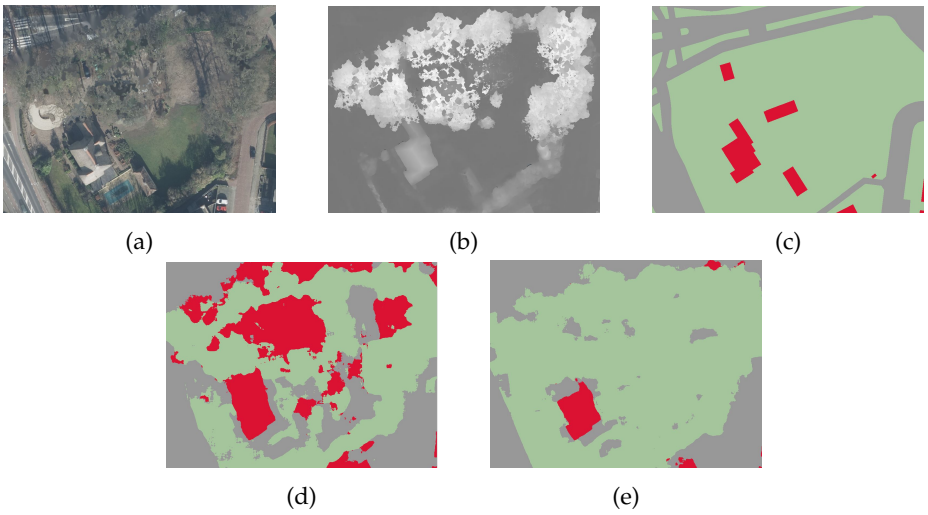
Table 5.12.: Results by class for different quantities of synthetic images

backyard of a house or a parking spot. Whilst in Haaksbergen, these cases are considered as *other*. Another difference is that the annotation for Potsdam is specially made for that specific image. Thus, the current shape of trees are considered as *other* and in Haaksbergen is only the base of the tree that is consider *other*. For *building* class, it is clear that it is easier to adapt from Potsdam to train in Haaksbergen than to train the other way around. Furthermore, for the case of training in Potsdam and testing in Haaksbergen, the precision increases when adding synthetic data. This is mainly due to the increased precision of the *building* class.

5.7. Comparison with other studies

First, a comparison with [Mulder, 2020] research in the city of Haarlem is presented, followed by a comparison with other synthetic datasets created by [Kong et al., 2019].

The study of Mulder [2020] consists of the semantic segmentation with the addition of one more class (*water*) in comparison to this research. However, the definitions of the classes *building* and *road* are the same. The inference of Haarlem was made by training the deep learning model with a mix of real datasets located in Haaksbergen and with the synthetic dataset developed in this research. In Table 5.15 and Figure A.7 the results are presented.



(a) (b) (c)

(d) (e)

Figure 5.13.: (a) TO Haaksbergen (b) DSM Haaksbergen (c) Ground Truth (d) Prediction with Potsdam training data (e) Prediction with training 50% Potsdam and 50% synthetic. It is seen that the addition of synthetic data improves the model results as it helps not to detect trees as buildings. Red = Building, Gray = Road and Green = Other.

(a) (b) (c) (d) (e)

Figure 5.14.: Potsdam TO (b) Potsdam DSM (c) Potsdam Ground Truth (d) Prediction with Haaksbergen training data (e) Prediction with training 50% Haaksbergen and 50% Synthetic data. It is seen that the addition of synthetic data improves the model results by detecting a wider variety of textures. Red = Building, Gray = Road and Green = Other.

Training	Images	Test	mIoU	F1
Haarlem [Mulder, 2020]	1444	**Haarlem**	**0.82**	**0.90**
Haaksbergen	1444	**Haarlem**	0.68	0.81
Haaksbergen	722	**Haarlem**	0.62	0.76
50% Haaksbergen - 50% Synthetic	1444	**Haarlem**	0.63	0.77
Synthetic	1444	**Haarlem**	0.50	0.70

Table 5.13.: Results in Haarlem with a comparison with Mulder [2020] research

Scenario	IoU Building	IoU Road	IoU Other
Haarlem [Mulder, 2020]	**0.88**	**0.79**	**0.81**
Haaksbergen (1444 im.)	0.78	0.57	0.70
Haaksbergen (722 im.)	0.76	0.44	0.65
50% Haaksbergen - 50% Synthetic (1444 im.)	0.75	0.49	0.66
Synthetic (1444 im.)	0.57	0.50	0.44

Table 5.14.: Results by class in Haarlem with a comparison with Mulder [2020] research

The cross-domain training data shows a decrease in performance with respect to the original study. When the mix is half real and half synthetic, the model under-performed in big buildings with flat roofs. On the other hand, the synthetic data helps the model to detect buildings with different height levels. Nevertheless, for continuous blocks of buildings, the model presents some errors, and it is believed that this is because of the lack of this type of buildings in the training data.

The quantity of training data, in this case, seems important as the mIoU decreases six p.p.. from training with 1444 to 722 images. Consequently, compared with inferring in Haaksbergen, where the saturation point is with fewer images (approximately 700), the saturation point is higher when performing cross-domain inference, indicating that the saturation point also depends on the testing data. This could be due to the fact that domain matching, when training and testing in the same area, is better than in a cross-domain scenario. Moreover, the addition of synthetic images in cross-domain inference helps the generalisation of the model to encompass the domain of the testing images.

As mentioned in the mix analysis before, adding synthetic data improves the model when the number of real images is less than the saturation point. Likewise, in this scenario, when adding synthetic images to the 722 images dataset, the model improves compared with only 722 real images. However, the improvements are more significant when adding real data than synthetic data.

The results of cross-domain and synthetic imagery are far from the ones in Mulder [2020]. The difference in training in the same domain with sufficient domain matching still proves better results than generalising training data to work in different domains.

Furthermore, Kong et al. [2019] uses synthetic data in combination with real data to detect buildings. Similarly to the current research, they test the data in ISPRS [2020] dataset but the train with a different dataset. They obtain an IoU of 0.67 with the addition of synthetic data and an improvement of 1 p.p. with respect to training with only real data. This research work accomplishes similar results with an IoU of 0.66 and an improvement of 12 p.p. with the addition of synthetic data. The training dataset in Kong et al. [2019] consists of several

Training	Test	IoU Building
4 Different cities [Kong et al., 2019]	**Potsdam + Vaihingen [ISPRS, 2020]**	0.67
4 Different cities [Kong et al., 2019] + Synthetic Data [Kong et al., 2019]	**Potsdam + Vaihingen [ISPRS, 2020]**	**0.68**
Haaksbergen	**Potsdam [ISPRS, 2020]**	0.54
50% Haaksbergen - 50% Synthetic	**Potsdam [ISPRS, 2020]**	**0.66**

Table 5.15.: Results in Potsdam with a comparison with Kong et al. [2019] research

cities across the world, which makes the generalisation higher and makes the addition of synthetic data less significant. Due to this research only used real training data from one city, the improvement with the addition of synthetic imagery is greater.

5.8. Domain Adaptation

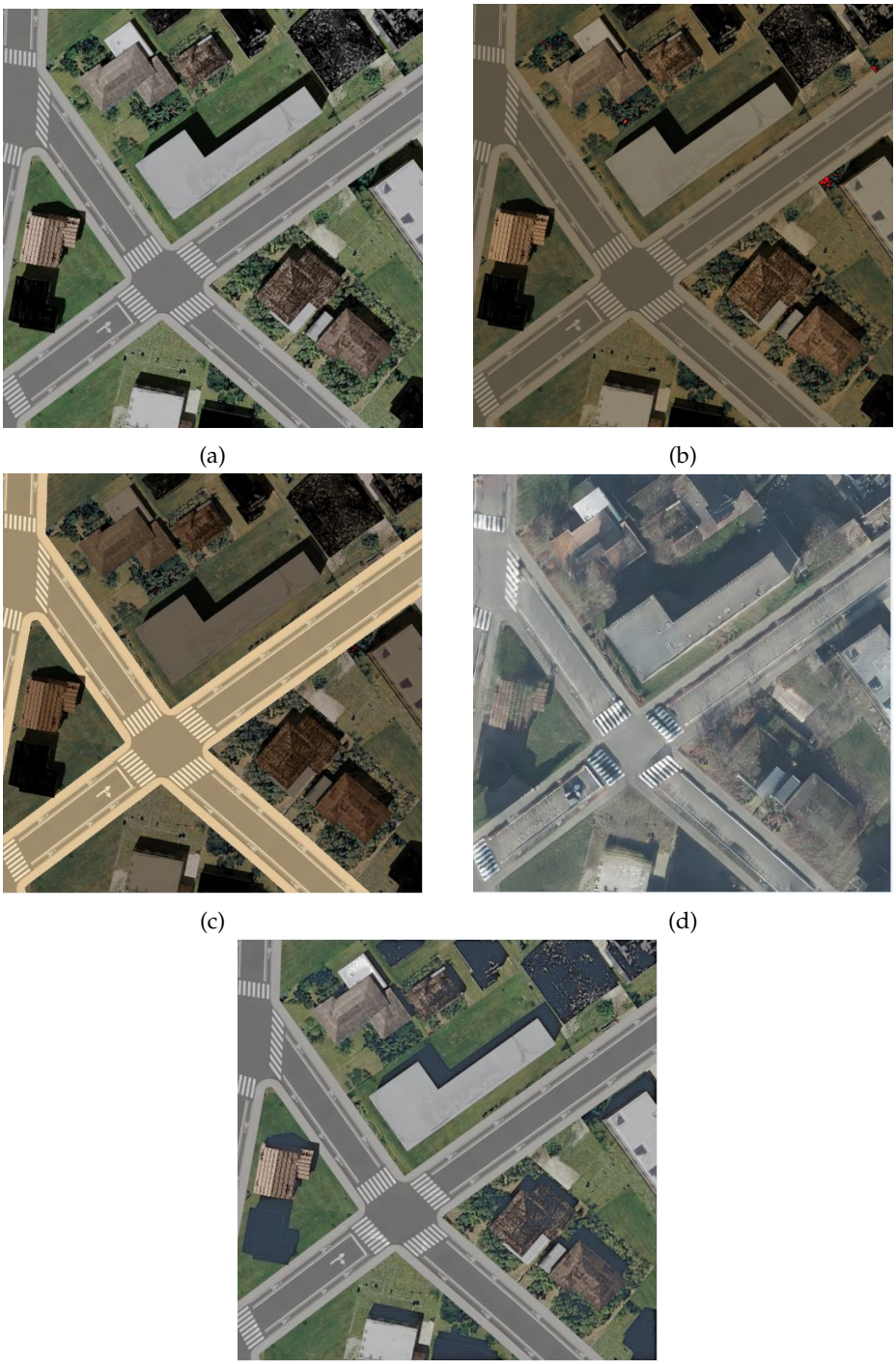
In the current section, the results of different DA techniques are presented. The domain adaptation was performed on the same synthetic dataset for these experiments. Figure 5.15 shows an example of the final transformation for each technique and Figure A.8 shows the prediction results. In addition, the results for semantic segmentation are presented in Table 5.16.

Despite that CORAL shortens the difference of the domain covariance between the synthetic and real dataset, the results decrease considerably. Mainly because with CORAL the model stops detecting roads and labels most of these pixels as *other*. Similarly, relatively small and big buildings are not detected. Despite that the general covariance is closer to the real data, *road* and *building* covariance are still different, which causes more confusion for the model. In the visual assessment, the *building* class detects more defined edges than the original synthetic dataset.

When performing the CORAL alignment by each class, the mIoU decreases more than the general CORAL. Interestingly, these experiments show that even though the domain covariance is closer to the real covariance for every class, the models fail to perform in real distributions.

On the other hand, the mIoU increases from 0.45, with the original synthetic dataset to 0.49 with Cycle GAN. However, for the class *building* it decreases from 0.56 to 0.50. Mainly because with Cycle GAN, the model does not predict well in big buildings. Visually, with the style transfer, building's edges are lost even though the distribution domain is similar. The style transfer seems to work for the class roads as they have a more defined shape. Nevertheless, some flat roofs with grey textures are labelled as *road*, reducing the precision for this class and the recall for *building*. The style transfer for trees seems not to be working as it detects most of the trees as *building*.

Finally, the results of *Cycada* where the style transfer is consistent by class, has an improvement of 0.4 p.p. in the mIoU in comparison to the original synthetic dataset. Similarly to Cycle GAN, the big buildings with flat surfaces are not predicted correctly, and the *roads* improve the overall IoU. Despite the results being similar to Cycle GAN, the visual assessment shows that *Cycada* detects *road* with more defined edges and the difference between the class *other* and



(a) (b) (c) (d) (e)

Figure 5.15.: (a) Original Synthetic city image (b) Synthetic image with CORAL alignment (c) Synthetic image with CORAL by class alignment (d) Synthetic image with Cycle GAN transfer (e) Synthetic image with *Cycada* transfer. It is seen that CORAL approaches seems to be synthetic. Whilst, Cycle GAN appears more realistic but with more noise. Finally *Cycada* seems more realistic than the synthetic but not as realistic as Cycle GAN

Domain Adaptation Technique	mIoU	F1
Synthetic City	0.45	0.61
CORAL	0.36	0.48
CORAL by classes	0.36	0.47
Cycle GAN	**0.49**	0.65
Cycada	**0.49**	**0.66**

Table 5.16.: Results for different types of DA techniques

road is clearly visible. Thus, the style transfer by class seems to work to differentiate between these two classes. On the other hand, for both *Cycada* and Cycle GAN, the models seem to become more sensitive to textures than to height. Big buildings with grey roofs are not correctly detected, and these types of buildings should be detectable with the height values.

	building			road			other		
Domain Adaptation Technique	IoU	prec.	recall	IoU	prec.	recall	IoU	prec.	recall
Synthetic City	0.56	0.62	**0.84**	0.26	0.41	0.42	0.53	0.75	0.65
CORAL	0.46	0.62	0.64	0.05	0.33	0.05	0.58	0.65	0.83
CORAL by classes	0.44	0.62	0.61	0.04	0.44	0.04	**0.59**	0.65	**0.86**
Cycle GAN	0.51	0.61	0.76	**0.40**	**0.49**	**0.69**	0.55	**0.82**	0.63
Cycada	**0.52**	**0.63**	0.76	0.39	**0.49**	**0.69**	0.56	0.81	0.65

Table 5.17.: Results by class for different types of DA techniques

6. Conclusions and Future Work

This chapter presents the conclusions of the study. The research questions are answered. Next, this study's main contributions and limitations are provided, and finally, some recommendations for future work are discussed.

6.1. Conclusion

The objective of this research was to create a pipeline that generates synthetic images with semantic segmentation labels that can be used in existing deep learning models. A synthetic city was generated to produce images for training a semantic segmentation algorithm, achieving a significant improvement in scenarios in which the domain gap between the training and testing datasets is big. In real applications, labelling real data with a minimal gap is costly. For these applications, synthetic data improves the segmentation results with a combination of existing sets of public images. On the other hand, the use of synthetic data does not improve the results on inference with small domain gap between the training and testing datasets.

6.2. Research Overview

This study addressed the main question with multiple sub-questions on this topic are reviewed below.

Main research question: *To what extent can synthetic data improve the current Deep Learning-based models for automated semantic segmentation for aerial images?*

The results and analysis have shown that synthetic imagery can potentially improve segmentation problems of aerial images in contexts where a lack of labelled data is present. Since this labelling process is extensive and expensive, synthetic data can help perform such segmentation problems. With the addition of already generated real training data, synthetic data can help the model in generalization and, thus, perform better in unseen scenarios. Nevertheless, the domain and visual mismatch between real and synthetic data can be significant as the randomization of scenes and the texture library are limited. This can make it difficult for a model trained on synthetic data to detect objects in real imagery. On the other hand, designing synthetic data with procedural methods helps create different virtual scenarios to understand further how deep learning algorithms learn different features present in the training sets.

a: How to build an automatic virtual city to create synthetic imagery which can be used as training data?

Building a virtual city is not trivial as it needs to be sufficiently random to imitate any urban environment. Also, it should be realistic as possible to make the learning process of a deep learning model. We used ESRI CityEngine to create random cities with procedural modelling

in this research. This program brings several advantages to freely design 3D models in urban environments. Using random parameters to create buildings brings more randomization to the image. In addition, its pipeline allows to generate realistic urban networks in which parcels, open space and roads are distributed accordingly. To increase realism, the procedural rules were changed to adapt better to a real environment.

CityEngine capabilities such as different road patterns, 3D modelling rules, and parcel distributions are adapted to the pipeline to create different virtual cities. In this thesis, approximately 22.000 synthetic areas were created, and distributed in different datasets to assess their use in semantic segmentation problems.

b: How do the specifications of 3D models (trees, buildings or roads) of a virtual city affect the results of semantic segmentation of aerial images?

Deep learning models not only learn from texture but also from spatial features present in the image. Consequently, 3D models have an important role in building synthetic imagery for semantic segmentation. For the case of buildings, a default virtual city from CityEngine was tested alongside a built procedural city with stochastic specifications of real buildings in the testing imagery. This experiment aimed to see if having similar features to the real environment improves the model while maintaining the same texture library. Results showed a minimum difference between the two models, but the level of detail in the models can help the model learn smaller features.

Furthermore, for the case of roads, different patterns showed different results. Patterns with regular distribution such as *raster* (Figure 5.6a), performs slightly better to detect roads. The road patterns seem to partly define the model's sensitivity to heights, shape or texture values. If the shape of roads have different intersection angles or more curvilinear edges, the model tends to focus more on height values. In contrast, in the other case, the model focuses more on the shape, even learning to detect textures not present in the synthetic dataset.

Finally, the presence of trees in the virtual city, even though it is not defined as a class to detect, is important as the model detects trees as buildings. Having more realistic trees helps the model to detect them as *other*.

c: What is the most suitable quantity ratio between real and virtual training data for semantic segmentation of aerial images?

The most suitable radio depends highly on the ability of the real dataset for generalization. In other words, the synthetic dataset improves the model performance only if the amount of the real dataset is less than the saturation point. This saturation point depends on the randomization of the real world and the variety of shapes and textures. In general, the model will perform better with more real training data, but with a ratio of 50% for each real and synthetic dataset, the results maintained their quality.

d: Does synthetic data improve automated semantic segmentation of aerial images in cross-geographical scenarios? Cross-geographical is using real data from a particular area and testing data of a completely different area.

In real-world applications, it is a common issue not to have labelled data in the area of inference. Training with real data from different areas is an option for these scenarios. Adding synthetic data improves the results as more generalization is added to the training process. Nevertheless, the results depend highly on the real data used and its ability to generalize in other domains.

e: Which domain adaptation technique is more effective for adapting from synthetic imagery to real imagery domain.

Different domain adaptation techniques were implemented in the current study. The correlation alignment CORAL performed in the whole image, and the CORAL by class do not improve the results for these synthetic datasets. On the other hand, style transfer approaches such as Cycle GAN and *Cycada* improve the overall results, especially in *road* class which is the one with a lesser variety of textures. The main drawback of these models is that the shapes of buildings become less defined during the style transfer.

6.3. Discussion

6.3.1. Contributions

This thesis is built upon previous studies where the segmentation performance is already high. Nevertheless, from the development of more robust software to create synthetic cities, such as game engines, this research gives a initial point to comprehend synthetic data better. Overall, there are three main areas where these results are helpful for semantic segmentation of aerial images:

- **Cross-Domain Projects**: Synthetic data can play an essential role in cross-domain applications where labelling data is not available. Since the only cost for a synthetic dataset is a licence for CityEngine and some needed hardware memory capabilities, it is cheaper than manual labelling. In addition, if some variables in the real environment are known, different configurations can be set to enhance the contribution of the synthetic data. Synthetic datasets can be constantly improve with creating new datasets with different characteristics.
- **Understanding the learning of Deep Learning models**: Since the synthetic imagery can be fully designed, testing with different settings of 3D models, quantity and textures is helpful to a better understanding of artificial learning. This study shows how the models become more sensitive to height or texture values or even defined or random shapes with certain road parameters or style transferring. This understanding, along with the ability to choose parameters in the making of the virtual city, gives the option to improve the design of the training data. For instance, the height distribution of the buildings, the pattern of the street network, or the type of trees can be set according to the target domain to be able to improve the current segmentation.
- **Starting point to produce synthetic aerial imagery**: Despite the results not being as good as using real training data, the potential of synthetic data is significant. A model trained solely on synthetic data and applied on synthetic imagery can detect almost 95% of the building pixels. This means that with more realistic trees and city utilities and more variety of textures, the precision could improve to the point that it is better than the real training data. Throughout the different experiments performed in the study, it is clear that the domain gap is smaller with more realistic synthetic cities, and the results will be better.

6.3.2. Limitations

The most important limitations of this study are:

- **Graphic Computation for the creation of virtual world**: Due the high quantity of trees in a real environment, it is important to model them in the virtual world. However, with the rendering times with the current approach of rendering in *Blender* it was not possible to have realistic trees in the synthetic datasets. As mentioned in the results, trees are an essential aspect for detecting *buildings*. Thus, more graphic capabilities would enable the rendering of more realistic worlds.
- **Realism for Synthetic City**: The built synthetic city could have lacked realism, which is essential to close the domain gap between the real and synthetic environments. However, designing a realistic city requires a lot of manual work and graphic memory, primarily because of the bigger scale of synthetic data. Complete textured models are available on the internet and are costly. In addition, another challenge is to label these models according to the classification task.
- **Texture Library**: Even thought that more textures were added to the standard CityEngine's library, the library was still limited. For more generalization of the training data, it is important to have a broader texture library.
- **Training and testing data quality** : Results could be lower than real results due to the limitations of the BGT layer. A manual check on the test area of Haaksbergen showed that the ground truth of BGT has an mIoU of about 0.95. Mostly, new buildings and small storage buildings are not present in the BGT. Furthermore, in both Potsdam and Haaksbergen datasets, some errors in the DSM are present. Some areas are occluded because of their point cloud origin, and another seems to have a temporal misalignment with the image and the ground truth, which confuses the model.

6.3.3. Recommendations and Future Work

In the different chapters of this study, new ideas surged which are recommended for future work. Firstly, gaming worlds seem to be a key player in creating synthetic cities since one of the main objectives in the gaming industry is the realism of the gaming world. Gaming engines such as Unreal Engine [1] can achieve the desired realism for synthetic imagery with bigger texture libraries to replace CityEngine and with a broader camera and rendering possibilities to replace blender. Secondly, improvements in the virtual world storage model. Even thought this step seems trivial, several 3D models come in the form of a unique mesh. Approaches such as CityGML [2] or CityJSON [3] clearly define every object present in the virtual city, making the labelling process more straightforward. Thirdly, a review of different deep learning architectures using synthetic data is important as this type of data is becoming more common. In addition, rather than focusing on roads or buildings, the *other* class, which is composed of tall and low vegetation, water bodies, and car, among others, are important for the creation of a synthetic city. Moreover, the ground textures represent a challenge in the design of synthetic data as the real environment presents a wide variety of textures. Finally,

[1] https://www.unrealengine.com/
[2] https://www.ogc.org/standards/citygml
[3] https://www.cityjson.org/about/

adding extra classes such as cars, vegetation or water bodies could improve the classification if they are modelled in the synthetic world.

A. Appendices

A.1. River Creation

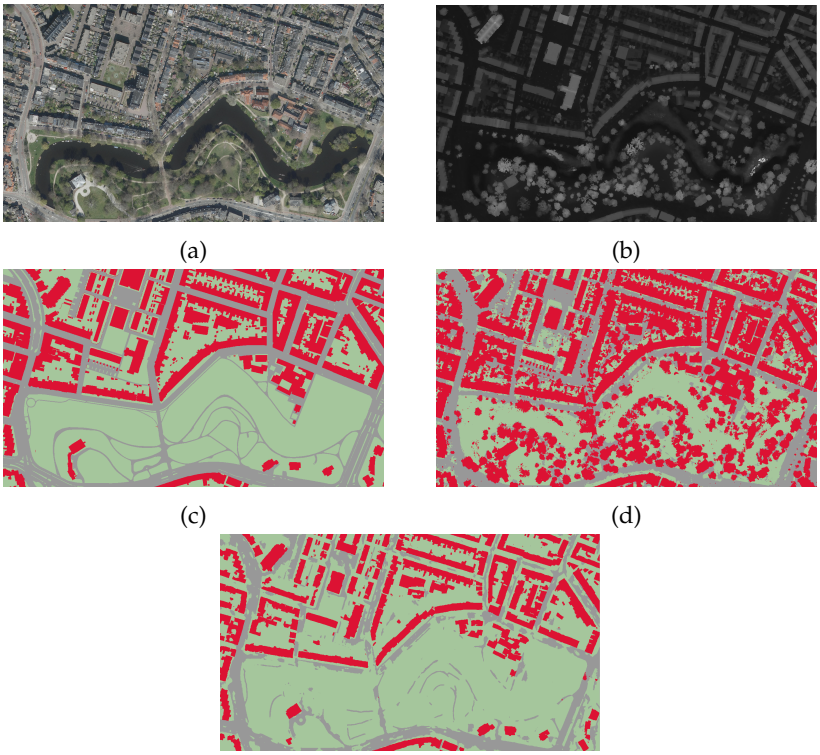
A.1.1. River in Synthetic City

Water bodies are an essential part of the built environment. Unfortunately, there is no built-in approach or rule to make rivers or canals in CityEngine. For this purpose, a technique to create a river without affecting the composition of the virtual city is developed.

A Python script in CityEngine's environment is made to create the river. First, a random segment of a street is taken. With the topological information of this segment, one can gather the following segment until the next segment exceeds a threshold angle of 40 degrees. Then, we set the width of the river or canal according to a random value between 8 and 40 meters. Finally, a realistic texture is applied to this set of segments through a simple CGA rule.

For testing the use of the river in the performance of the semantic segmentation, the model was tested in Haarlem. Figure A.2 shows the visual results for the addition of the river and is observed that the dark texture of the river helps the model to detect the river. Nevertheless, as the DSM is used, due to the occlusion of water, the height values are not precise which affects the segmentation in detecting water bodies as *buildings*. Using real data does not seem to affect as both DSM between the training and testing data have these height errors.

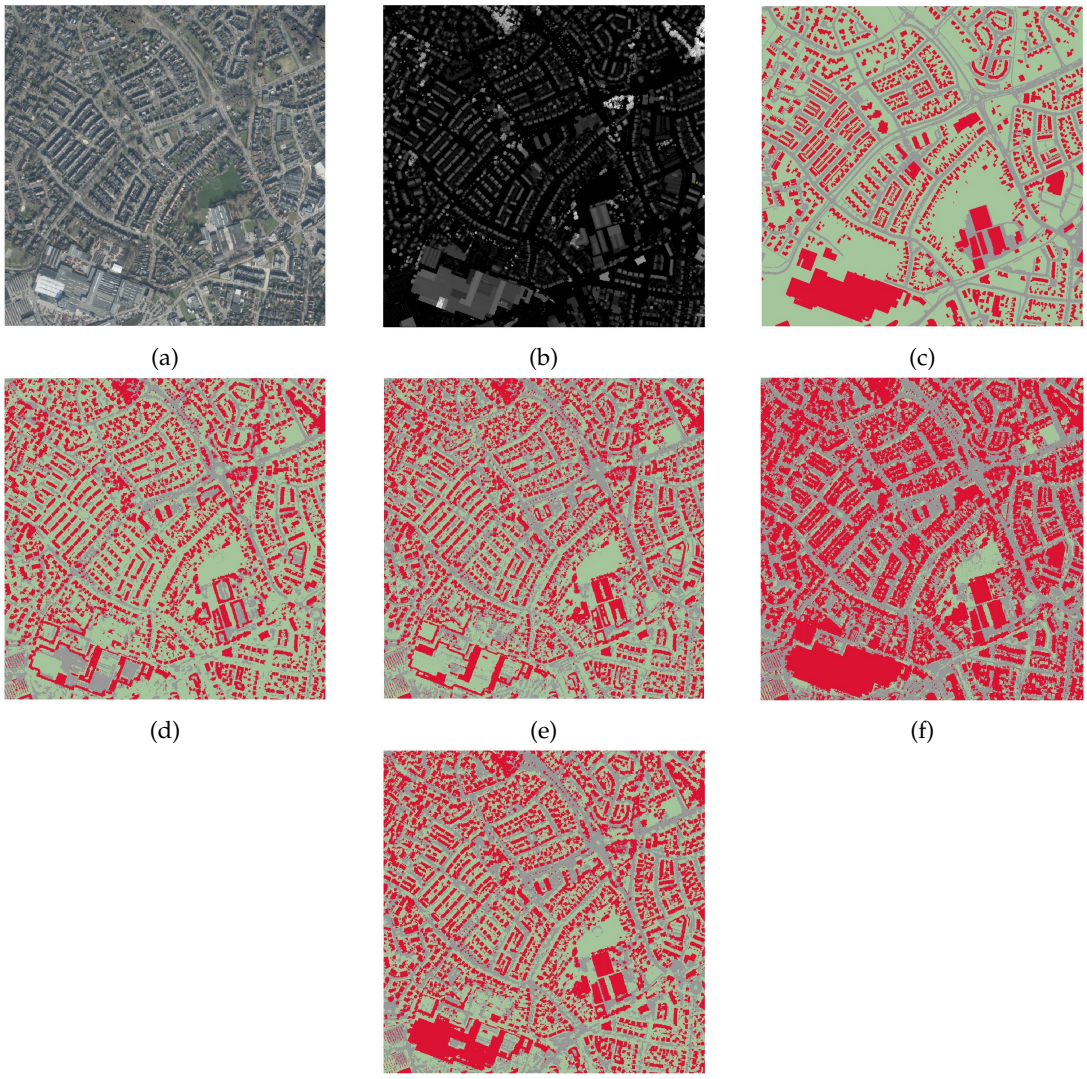
Figure A.1.: Aerial imagery of the river in CityEngine



(a) (b) (c) (d) (e)

Figure A.2.: Prediction with different patterns of roads (a) TO (b) DSM (c) Ground Truth (d) Synthetic dataset as training data (e) Real dataset as training data. Red = Building, Gray = Road and Green = Other.

A.2. Model Results for the different experiments



(a) (b) (c)

(d) (e) (f)

(g)

Figure A.3.: Prediction with different patterns of roads (a) TO (b) DSM (c) Ground Truth (d) Organic Road (e) Organic-Raster Road (f) Radial Road (g) Raster Road. Red = Building, Gray = Road and Green = Other.

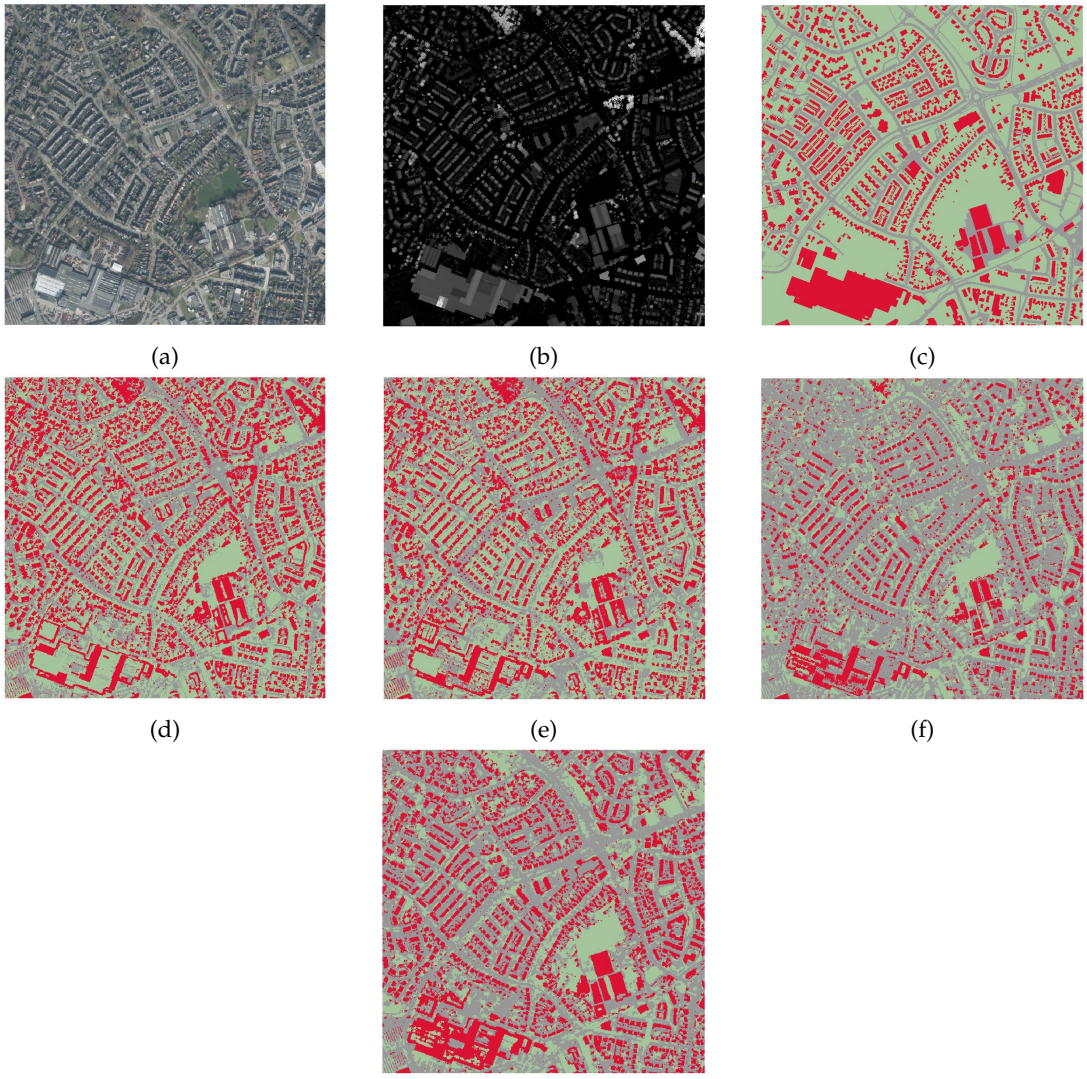
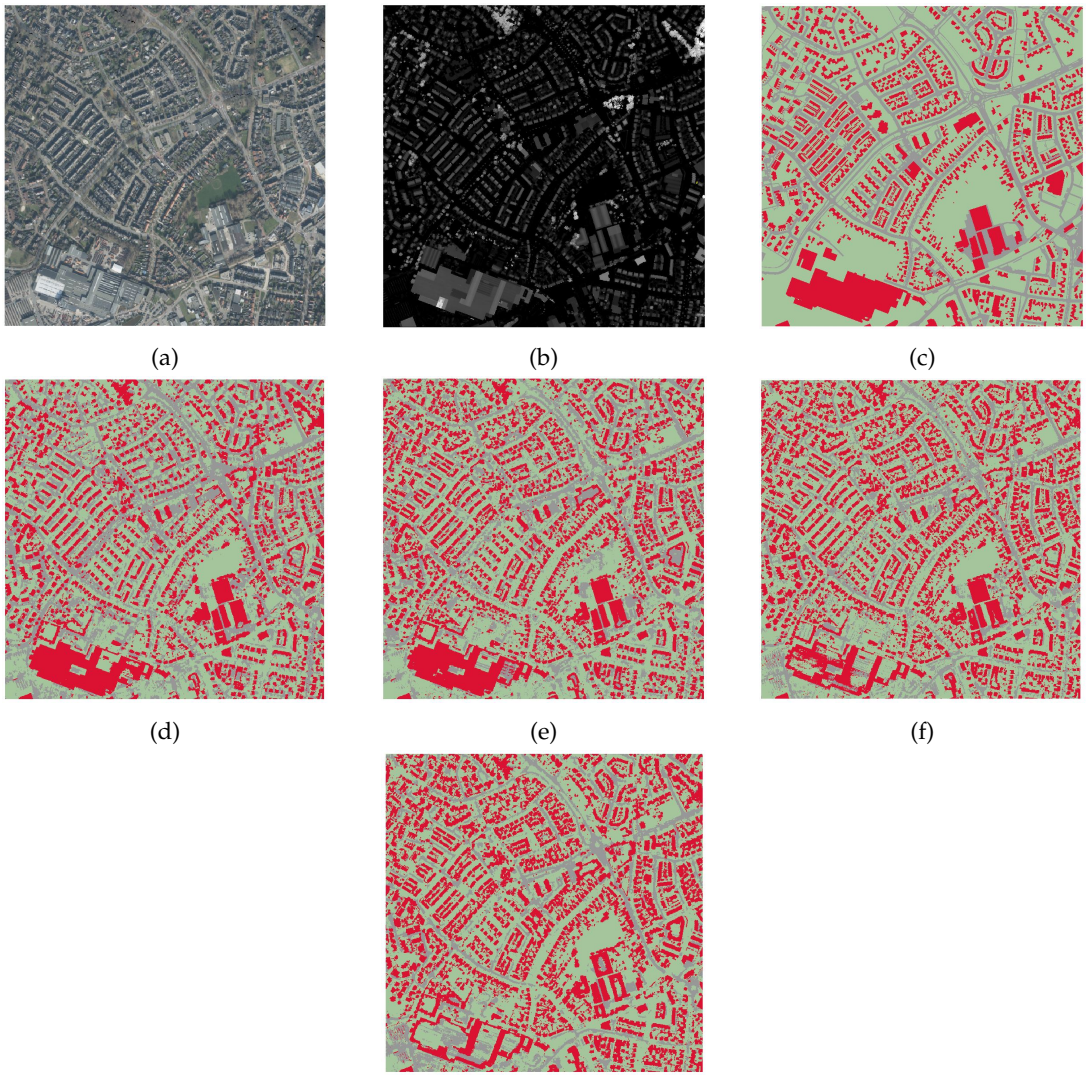


Figure A.4.: Prediction results for different types of tree models (a) TO (b) DSM (c) Ground Truth (d) *no trees* (e) *poly trees* (f) *fan trees* (g) *realistic* augmentation. Red = Building, Gray = Road and Green = Other.



(a) (b) (c)

(d) (e) (f)

(g)

Figure A.5.: Prediction with different amount of synthetic data (a) TO (b) DSM (c) Ground Truth (d) 378 images (e) 756 images (f) 1512 images (g) 3024 images. Red = Building, Gray = Road and Green = Other.

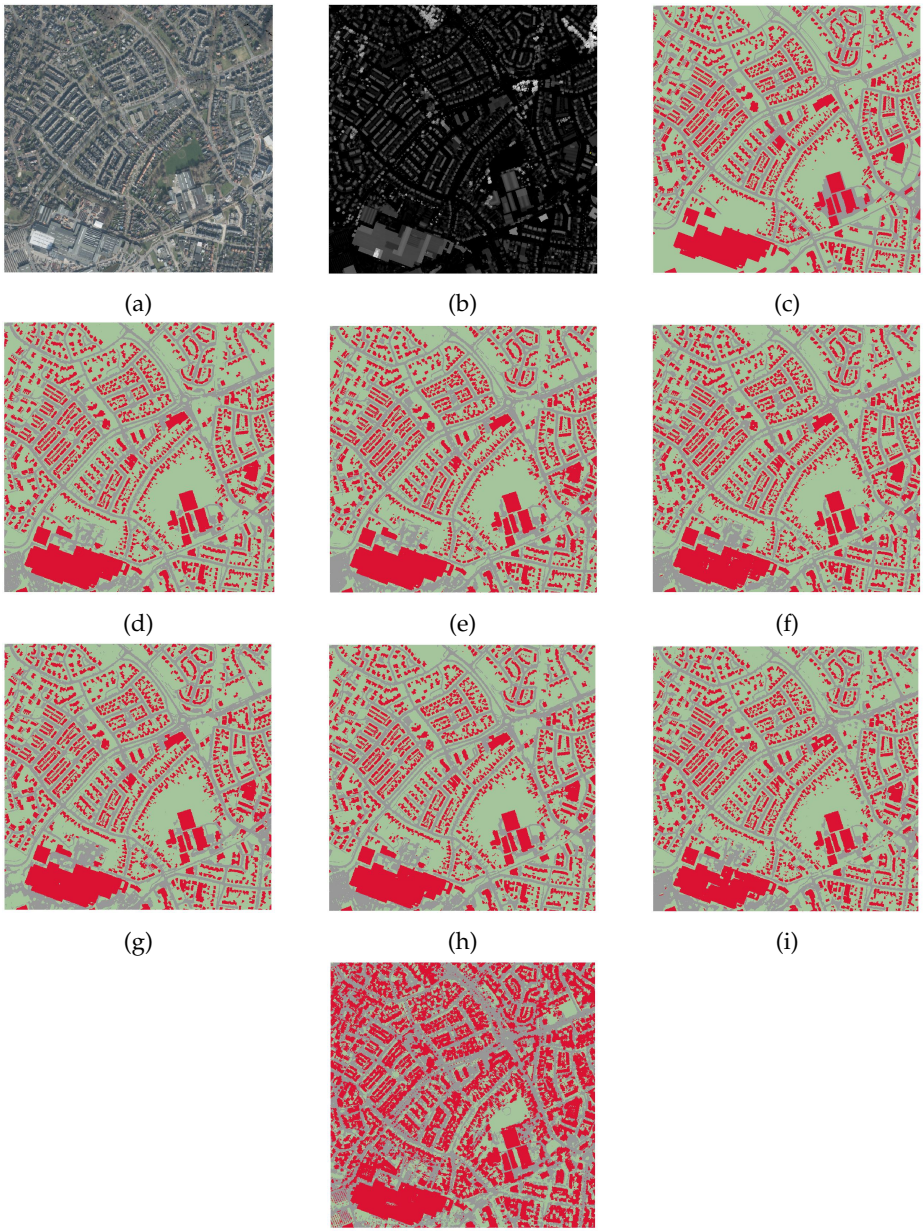


Figure A.6.: Prediction with different combinations of synthetic and real data (a) TO (b) DSM (c) Ground Truth (d) 100% real and 1444 images (e) 100% real and 722 images (f) 100% real and 288 images (g) 20& Synthetic - 80% real and 1444 images (h) 50% Synthetic - 50% real and 1444 images (i) 80% Synthetic - 20% real and 1444 images (j) 100% Synthetic and 1444 images. Red = Building, Gray = Road and Green = Other.

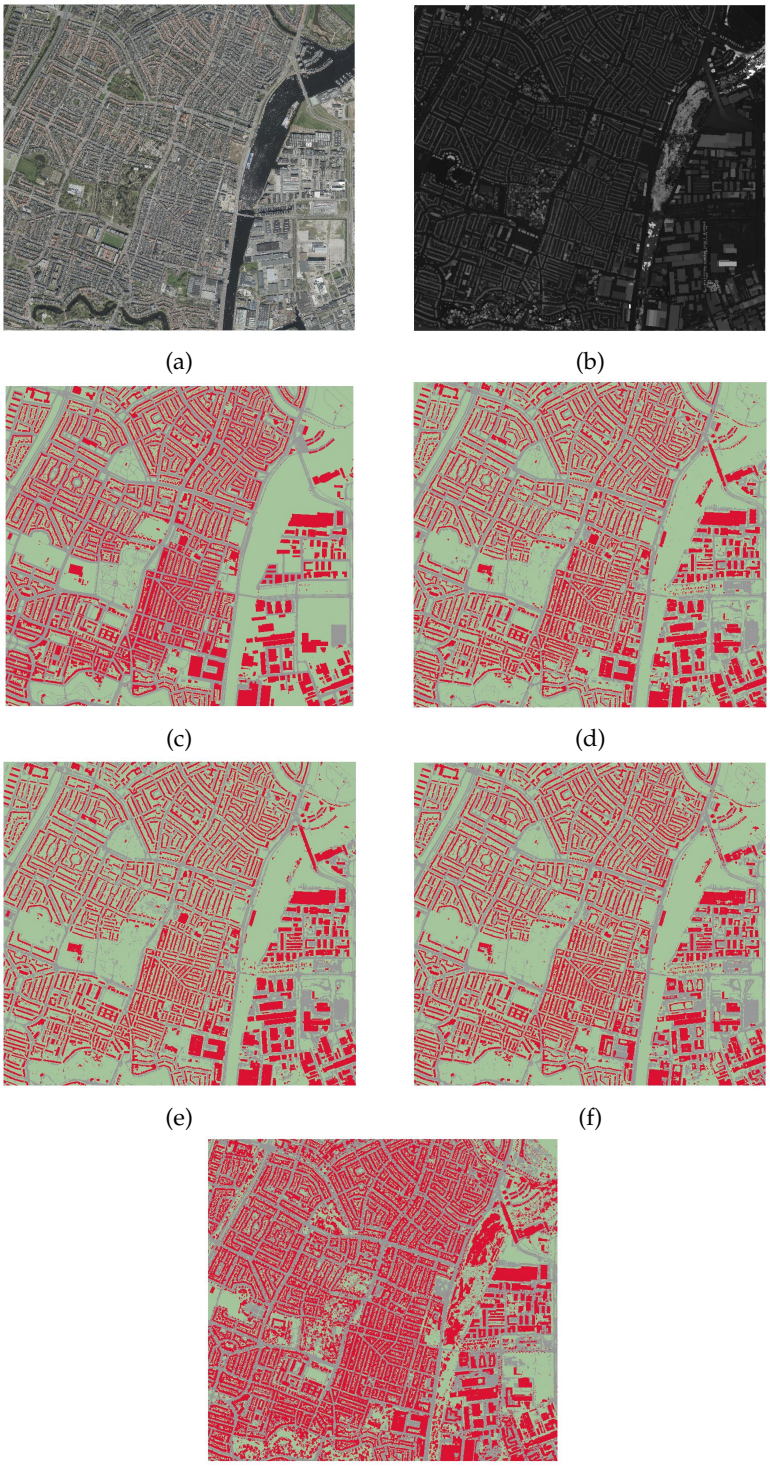


Figure A.7.: Prediction results in the city of Haarlem (a) TO (b) DSM (c) Ground Truth (d) 100% Training real data from Haaksbergen. 1444 images (e) 100% Training real data from Haaksbergen 722 images (f) 50% Training real data from Haaksbergen - 50% Synthetic data. 1444 images (g) 100% synthetic images. 1444 images. Red = Building, Gray = Road and Green = Other.

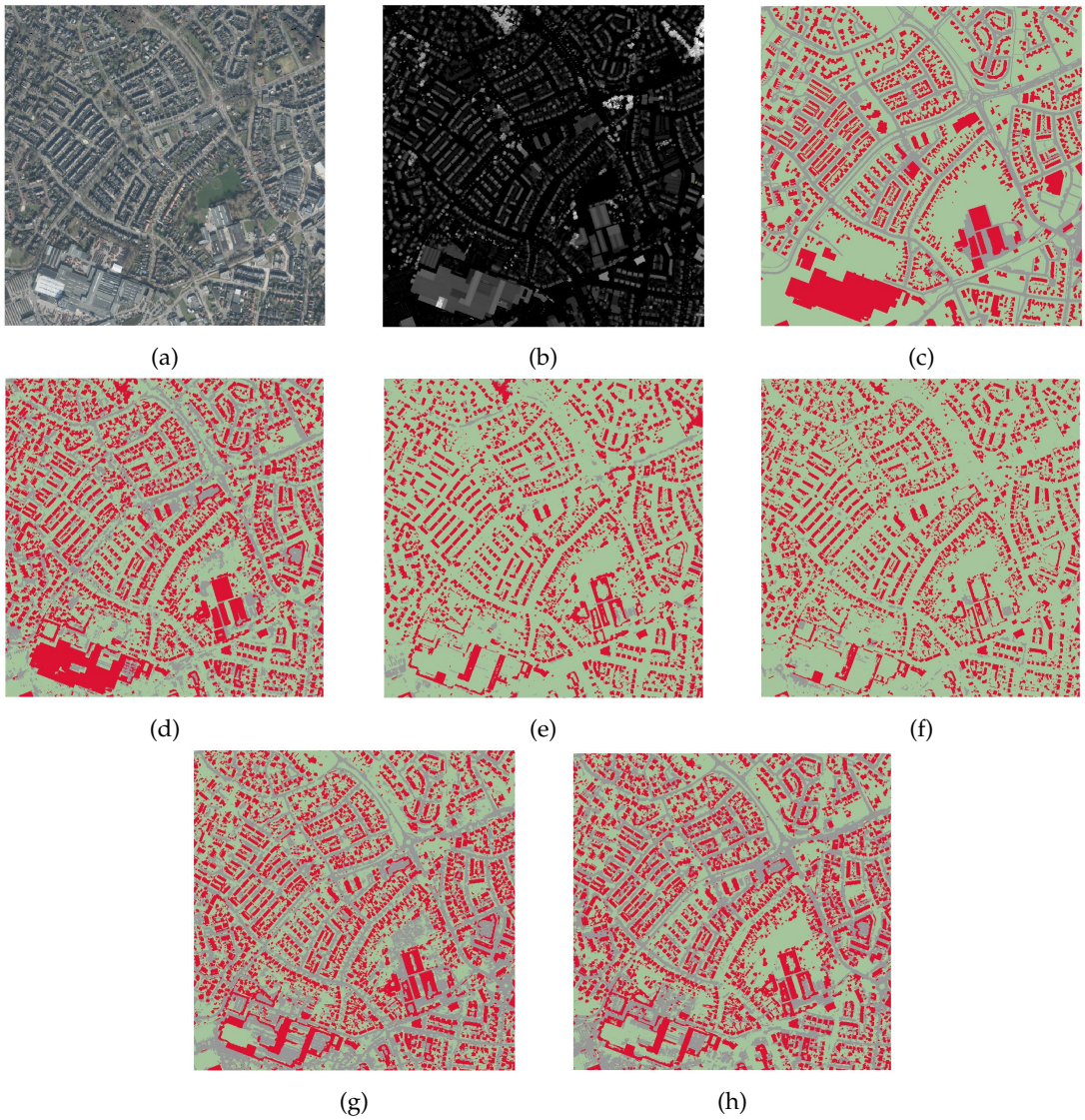


Figure A.8.: Prediction results for different techniques of domain adaptation (a) TO (b) DSM (c) Synthetic City (d) CORAL (e) CORAL by class (f) Cycle GAN (g) Cycada. Red = Building, Gray = Road and Green = Other.

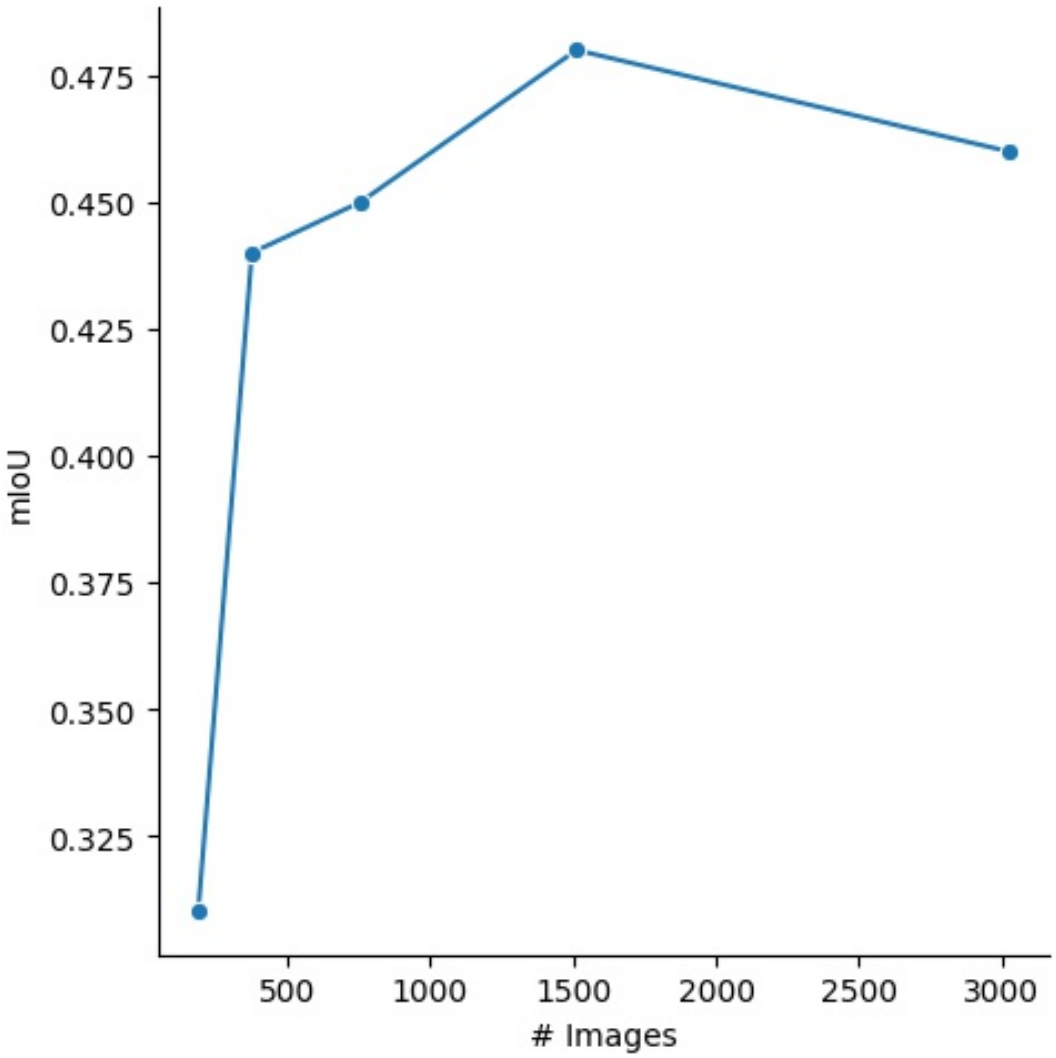
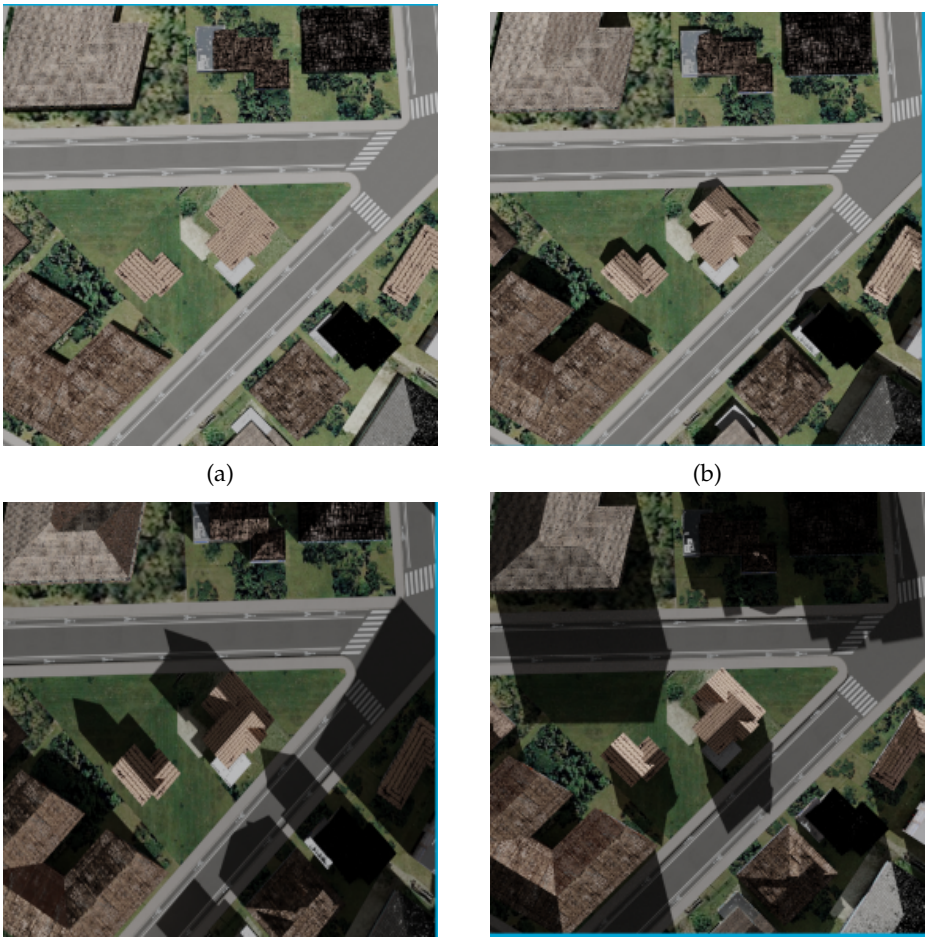


Figure A.9.: Synthetic imagery learning curve



(a) (b)

(c) (d)

Figure A.10.: Prediction results for different angles of sun position (a) No shading (b) angle at 13-14 hours (c) horizon angle 18-19 hours (d) fake angle.

A.3. Testing Different Lighting Parameters

Different lighting parameters are tested as follows: a synthetic dataset without shading, an angle between 13-14 hours, a horizon angle between 18-19 hours and a fake angle for the northern hemisphere pointing from north to south, see Figure A.10. In addition, the results are shown in Table A.1 and Table A.2. The results show that using a sun angle that simulates the real image's sun angle gives the best results than not setting any angle or setting different angle values. The ability of setting different lighting parameters is beneficial to the synthetic datasets in order to close the domain between real and synthetic data and also helps to generalize the synthetic data that contains different real domains.

Training	mIoU	F1
No shading	0.42	0.60
Angle 13-14 hours	0.45	0.59
Angle Horizon 18-19 hours	0.31	0.53
Fake angle, north to south	0.41	0.46

Table A.1.: Results with different lighting parameters

	building			road			other		
Scenario	IoU	prec.	recall	IoU	prec.	recall	IoU	prec.	recall
No shading	0.50	0.60	0.76	0.34	0.39	0.75	0.42	0.82	0.47
Angle 13-14 hours	0.56	0.62	0.84	0.26	0.41	0.42	0.53	0.75	0.65
Angle Horizon 18-19 hours	0.48	0.58	0.73	0.26	0.28	0.79	0.19	0.73	0.21
Fake angle, north to south	0.51	0.60	0.76	0.34	0.39	0.75	0.42	0.60	0.66

Table A.2.: Results by class for different quantities of synthetic images

A.4. Testing different weighting compositions between classes

Different parameters can be set when creating the synthetic datasets, such as defining the compositions of the virtual city. For instance, more roads can be added to have more presence on the ground truth map. In addition, the density of the buildings can also be set to have more *building* or *other* presence. In Table A.3 shows the different weight distributions of the synthetic data, and in Table A.4 shows the results for the different weight distributions. The results show that the best weighting distribution is when the same distribution is taken as the real testing data. We suggest implementing these values in the synthetic data when these values are known. Nevertheless, when this information is not known, using the default settings in the creating of the synthetic data results in similar values.

Training Model	Building	Road	Other
A Default CityEgnine City	26%	33%	40%
B with parking lots in inner parcels	33%	43%	23%
C with less roads	28%	24%	48%
D Real imagery configuration	18%	25%	57%

Table A.3.: Weighting distribution tests

Training Model	IoU Building	IoU Road	IoU Other	mIoU
A Default CityEgnine City	0.55	0.20	0.52	0.42
B with parking lots in inner parcels	0.45	0.44	0.25	0.38
C with less roads	0.38	0.46	0.30	0.37
D Real imagery configuration	0.56	0.26	0.53	0.45

Table A.4.: Results of different weighting distribution tests

Performance Measure	Building	Road	Other
IoU	1.00	0.98	0.97
Precision	1.00	0.99	0.99
Recall	1.00	0.99	0.99

Table A.5.: Results of training and testing in Synthetic Data

A.5. Training and Testing with synthetic data

To see if the synthetic data is able to perform semantic segmentation, we built two different virtual cities with the same parameters. The first city is used to train the model and the second to perform the inference. The results are 0.98 of mIoU and 0.99 for the F1 score In Table A.5 shows the results of the semantic segmentation. The results shows that the performance are almost to 1 in all classes. The results are outstanding as the parameters are the same and the variability of textures and shapes of the synthetic data is lesser than the real world.

B. Reproducibility self-assessment

B.1. Marks for each of the criteria

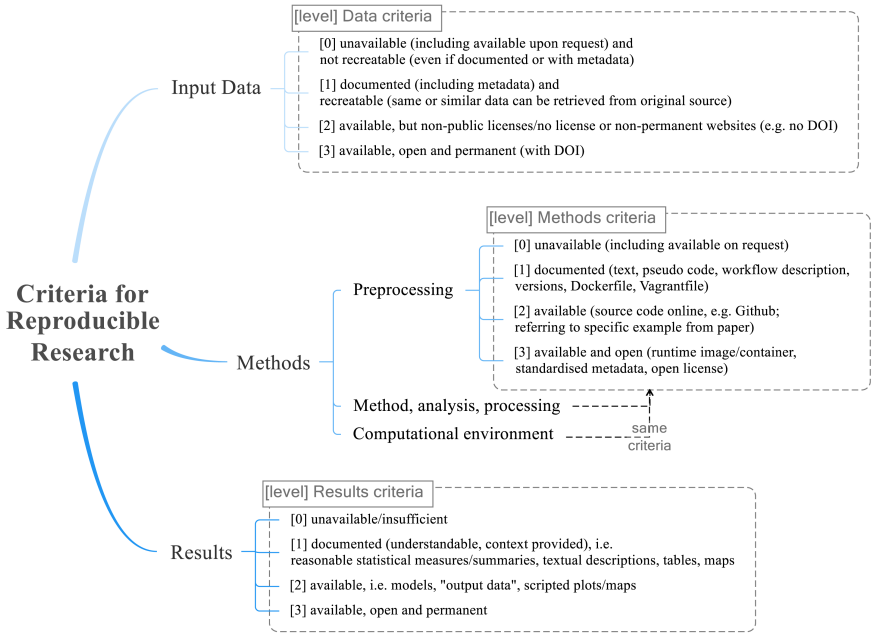


Figure B.1.: Reproducibility criteria to be assessed.

Grade/evaluate yourself for the 5 criteria (giving 0/1/2/3 for each):

1. input data: 1
2. preprocessing: 1
3. methods: 1
4. computational environment: 1
5. results: 1

B.2. Self-reflection

This work was made with the collaboration of READAR, which provided high quality and cleaned data and external software. For the input data, the imagery and the DSM maps are

developed by READAR. Thus, it cannot be made openly available. On the other hand, the use of ISPRS [2020] imagery of Potsdam is openly available and documented. The preprocessing of the data is explained in the document to be able to reproduce. For the methods and analysis, the creation of the synthetic city is described in the report, and the code is openly available with the restriction of having a licence for CityEngine software. In addition, for the rendering process, an open source pipeline of Blender Proc and open rendering software of Blender is used and described in the report. On the other hand, despite being described in the report, the deep learning model cannot be openly available as this belongs to READAR.

Finally, the classification results are obtained with the use of the deep learning model so reproducibilityis not possible. Nevertheless, the steps are explained in Chapter 3. The images were created and manually adjusted in QGIS, which makes them not easy to recreate. The images are openly available for the synthetic dataset and can be used for future works.

Bibliography

Agarap, A. F. (2018). Deep Learning using Rectified Linear Units (ReLU).

Audebert, N., Saux, B. L., and Lefèvre, S. (2017). Beyond RGB: Very High Resolution Urban Remote Sensing With Multimodal Deep Networks. *ISPRS Journal of Photogrammetry and Remote Sensing*, 140:20–32.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495.

Benjdira, B., Khursheed, T., Koubaa, A., Ammar, A., and Ouni, K. (2019). Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3. *2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)*, 8(5):1–6.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Grobler, A. G., Jaques, Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn
}project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

Castillo-Navarro, J., Audebert, N., Boulch, A., Le Saux, B., and Lefevre, S. (2019). What Data are needed for Semantic Segmentation in Earth Observation? In *2019 Joint Urban Remote Sensing Event (JURSE)*, pages 1–4. IEEE.

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., and Worley, S. (2002). *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition.

ESRI (2022). ArcGis CityEngine.

Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., and Garcia-Rodriguez, J. (2017). A Review on Deep Learning Techniques Applied to Semantic Segmentation.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks.

Hazirbas, C., Ma, L., Domokos, C., and Cremers, D. (2015). FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture. Technical report.

Hoffman, J., Tzeng, E., Park, T., Zhu, J.-Y., Isola, P., Saenko, K., Efros, A., and Darrell, T. (2018). CyCADA: Cycle-Consistent Adversarial Domain Adaptation. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1989–1998. PMLR.

Ichim, L. and Popescu, D. (2020). Segmentation of Vegetation and Flood from Aerial Images Based on Decision Fusion of Neural Networks. *Remote Sensing*, 12(15):2490.

ISPRS (2020). ISPRS Semantic Labeling.

Kadaster (2022). Basisregistratie Adressen en Gebouwen.

Kampffmeyer, M., Salberg, A.-B., and Jenssen, R. (2016). Semantic Segmentation of Small Objects and Modeling of Uncertainty in Urban Remote Sensing Images Using Deep Convolutional Neural Networks. Technical report.

Kattenborn, T., Leitloff, J., Schiefer, F., and Hinz, S. (2021). Review on Convolutional Neural Networks (CNN) in vegetation remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, 173:24–49.

Kazakeviciute-Januskeviciene, G., Janusonis, E., Bausys, R., Limba, T., and Kiskis, M. (2020). Assessment of the Segmentation of RGB Remote Sensing Images: A Subjective Approach. *Remote Sensing*, 12(24):4152.

Kelly, T. (2021). CityEngine: An Introduction to Rule-Based Modeling. In Shi Wenzhong }and Goodchild, M. F., Michael, B., Mei-Po, K., and Anshu, Z., editors, *Urban Informatics*, pages 637–662. Springer Singapore, Singapore.

Khan, S., Phan, B., Salay, R., and Czarnecki, K. (2019). ProcSy: Procedural Synthetic Dataset Generation Towards Influence Factor Studies Of Semantic Segmentation Networks. Technical report.

Kong, F., Huang, B., Bradbury, K., and Malof, J. (2019). *Computer Analysis of Images and Patterns*, volume 1089 of *Communications in Computer and Information Science*. Springer International Publishing, Cham.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lindenmayer, A. (1968). Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299.

Liu, M.-Y. and Tuzel, O. (2016). Coupled Generative Adversarial Networks.

Liu, Y., Minh Nguyen, D., Deligiannis, N., Ding, W., and Munteanu, A. (2017). Hourglass-ShapeNetwork Based Semantic Segmentation for High Resolution Aerial Imagery. *Remote Sensing*, 9(6):522.

Long, J., Shelhamer, E., and Darrell, T. (2014). Fully Convolutional Networks for Semantic Segmentation.

Maggiori, E., Tarabalka, Y., Charpiat, G., and Alliez, P. (2017). Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Image Labeling Benchmark. Technical report.

Marmanis, D., Wegner, J. D., Galliani, S., Schindler, K., Datcu, M., and Stilla, U. (2016). SEMANTIC SEGMENTATION OF AERIAL IMAGES WITH AN ENSEMBLE OF CNNS. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-3:473–480.

Mulder, A. E. (2020). MSc thesis in Geomatics for the Built Environment Semantic Segmentation of RGB-Z Aerial Imagery Using Convolutional Neural Networks. Technical report, TU Delft, Delft.

Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. (2006). Procedural modeling of buildings. *ACM Transactions on Graphics*, 25(3):614–623.

Nikolenko, S. I. (2021). Synthetic Data for Deep Learning. Technical report.

Ohori, K. A., Ledoux, H., and Peters, R. (2020). 3D modelling of the built environment. Technical report, Tu Delft, Delft.

Ortega-Córdova, L. M. (2018). MSc thesis in Geomatics for the Built Environment Urban Vegetation Modeling 3D Levels of Detail. Technical report.

PDOK (2021). Basisregistratie Grootschalige Topografie (BGT).

Peters, R., Dukai, B., Vitalis, S., van Liempt, J., and Stoter, J. (2022). Automated 3D Reconstruction of LoD2 and LoD1 Models for All 10 Million Buildings of the Netherlands. *Photogrammetric Engineering & Remote Sensing*, 88(3):165–170.

Prusinkiewicz, P., Hanan, J., Hammel, M., and Mech, R. (2013). L-systems: from the Theory to Visual Models of Plants. Technical report, Collingwood.

Richter, S. R., Vineet, V., Roth, S., and Koltun, V. (2016). Playing for Data: Ground Truth from Computer Games.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In Navab Nassirand, H., Joachimand Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, pages 234–241, Cham. Springer International Publishing.

Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. M. (2016). The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. Technical report.

Rosenberger, C., Chabrier, S., Laurent, H., and Emile, B. (2006). Unsupervised and Supervised Image Segmentation Evaluation. In *Advances in Image and Video Segmentation*, pages 365–393. IGI Global.

Saito, S. and Aoki, Y. (2015). Building and road detection from large aerial imagery. In Lam, E. Y. and Niel, K. S., editors, *Image Processing: Machine Vision Applications VIII*, volume 9405, page 94050K. SPIE.

Saito, S., Yamashita, T., and Aoki, Y. (2016). Multiple Object Extraction from Aerial Imagery with Convolutional Neural Networks. *Journal of Imaging Science and Technology*, 60(1):10402–1.

Sankaranarayanan, S., Balaji, Y., Jain, A., Lim, S. N., and Chellappa, R. (2017). Learning from Synthetic Data: Addressing Domain Shift for Semantic Segmentation.

Schwarz, M. and Müller, P. (2015). Advanced procedural modeling of architecture. *ACM Transactions on Graphics*, 34(4):1–12.

Shor, J. (2022). GANs.

Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

Sun, B., Feng, J., and Saenko, K. (2016). Return of Frustratingly Easy Domain Adaptation. Technical report.

Tempfli, K., Huurneman, G. C., Bakker, W. H., Janssen, L. L. F., Feringa, W. F., Gieske, A. S. M., Grabmaier, K. A., Hecker, C. A., Horn, J. A., Kerle, N., van der Meer, F. D., Parodi, G. N., Pohl, C., Reeves, C. V., van Ruitenbeek, F. J. A., Schetselaar, E. M., Weir, M. J. C., Westinga, E., and Woldai, T. (2009). *Principles of remote sensing : an introductory textbook*. ITC Educational Textbook Series. International Institute for Geo-Information Science and Earth Observation, Netherlands.

Tsai, Y.-H., Hung, W.-C., Schulter, S., Sohn, K., Yang, M.-H., and Chandraker, M. (2018). Learning to Adapt Structured Output Space for Semantic Segmentation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7472–7481. IEEE.

Wang, M. and Deng, W. (2018). Deep Visual Domain Adaptation: A Survey.

Xu, Y., Wu, L., Xie, Z., and Chen, Z. (2018). Building Extraction in Very High Resolution Remote Sensing Imagery Using Deep Learning and Guided Filters. *Remote Sensing*, 10(1):144.

Yang, M. Y., Liao, W., Li, X., and Rosenhahn, B. (2018). Deep Learning for Vehicle Detection in Aerial Images. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 3079–3083. IEEE.

Yuan, X., Shi, J., and Gu, L. (2021). A review of deep learning methods for semantic segmentation of remote sensing imagery. *Expert Systems with Applications*, 169:114417.

Yucong Lin and Saripalli, S. (2012). Road detection from aerial imagery. In *2012 IEEE International Conference on Robotics and Automation*, pages 3588–3593. IEEE.

Zhu, H., Meng, F., Cai, J., and Lu, S. (2015). Beyond Pixels: A Comprehensive Survey from Bottom-up to Semantic Image Segmentation and Cosegmentation. *Journal of Visual Communication and Image Representation*, 34:12–27.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.