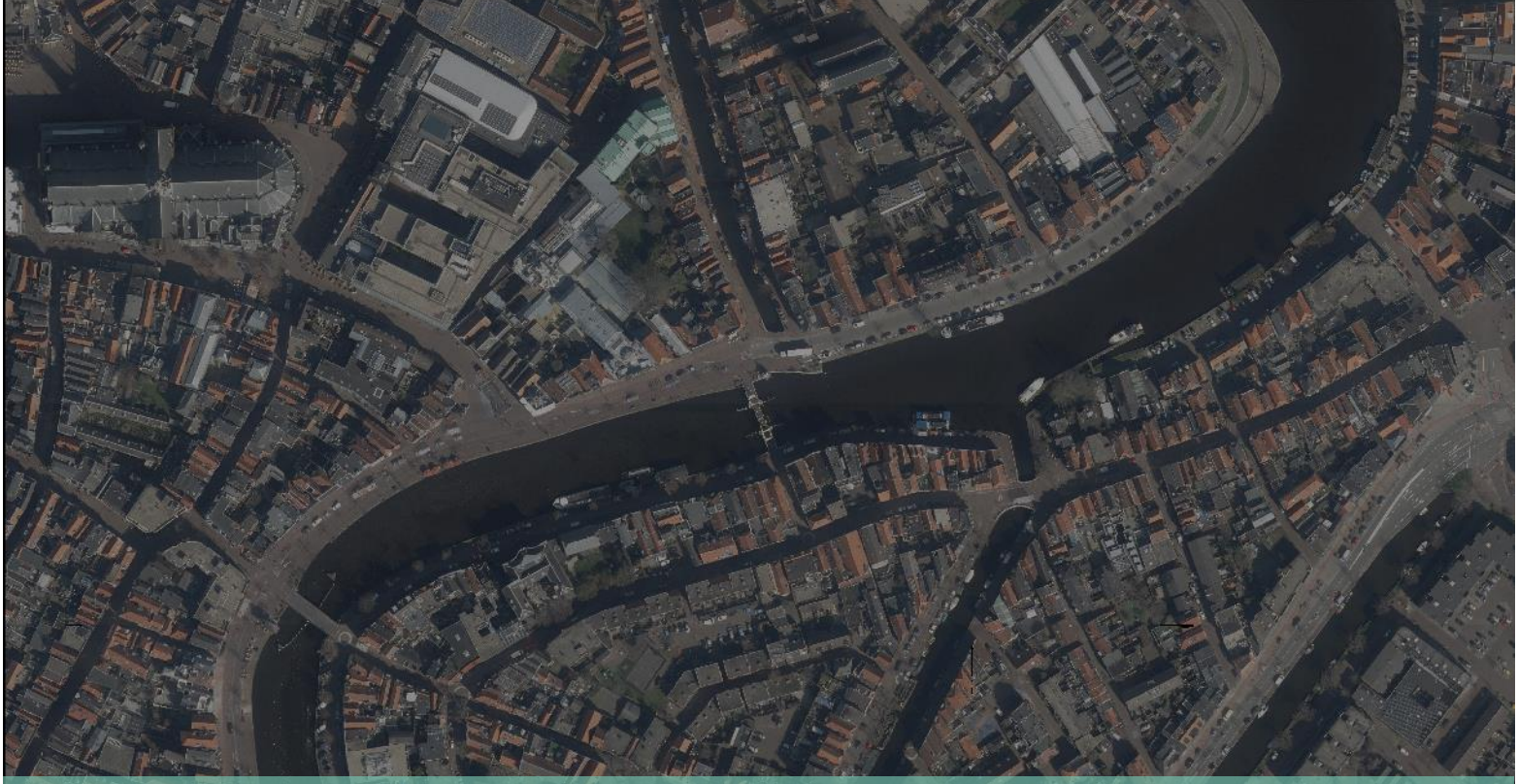


Semantic Segmentation of RGB-Z Aerial Imagery Using Convolutional Neural Networks

Amber E. Mulder 2020

MSc thesis in Geomatics for the Built Environment

TU Delft

READAR
real estate radar

Cover illustrations:
Top: True ortho imagery of Haarlem, generated by READAR.
Bottom: Reclassified version of the Basisregistratie Grootschalige Topografie. Retrievable from https://www.pdok.nl/.

MSc thesis in Geomatics

Semantic Segmentation of RGB-Z Aerial Imagery Using Convolutional Neural Networks

Amber E. Mulder

June 2020

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 with:

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

READAR
Vondellaan 16
3521GD Utrecht
https://readar.com/

Supervisors:	Balázs Dukai
	Ravi Peters
Co-reader:	Jantien Stoter
Company supervisors:	Sven Briels
	Jean-Michel Renders

Abstract

Semantic segmentation (or pixel-level classification) of remotely sensed imagery has shown to be useful for applications in fields as mapping of land cover, object detection, change detection and land-use analysis. Deep learning algorithms called convolutional neural networks (CNNs) have shown to outperform traditional computer vision and machine learning approaches in tackling semantic segmentation tasks. Furthermore, addition of height information (Z) to aerial imagery (RGB) is believed to improve segmentation results. However, discussion remains on the following: to what extent height information adds value; the best way to combine RGB information with height information; and what type of height information can best be used. This study aims to answer these questions. In this research work, the CNN architectures FCN-8s, SegNet, U-Net and FuseNet-SF5 are trained to semantically segment 10 cm resolution true ortho imagery of Haarlem, potentially augmented with height information. The outputted topographic maps contain the classes `building`, `road`, `water` and `other`. Experiments are conducted that allow for the comparison of 1) models trained on RGB and on RGB-Z, 2) models combining RGB and height information through data fusion and through data stacking, and 3) models trained using different types of absolute and relative height approaches. Performances are compared based on scores on the performance measure (mean) intersection over union (IoU) and through visual assessment of outputted prediction maps. The results indicated that on average segmentation performance improves by approximately 1 percent when absolute height information is added. The class `building` showed to benefit the most from the addition of height information. Furthermore, extracting features from height information in a separate encoder and fusing these into RGB feature maps, led to a higher overall segmentation quality than when height information is provided as a stacked extra band and processed in the same encoder as the RGB information. Finally, models using relative height delivered a higher quality segmentation than when absolute height approaches were used, especially for large objects. The best performing model; FuseNet-SF5 trained on RGB imagery and pixel-level, relative height, retrieved a mean IoU of 0.8427 and IoUs of 0.8744, 0.7865, 0.9131 and 0.7966 for the classes `building`, `road`, `water` and `other` respectively. This model was able to correctly classify over 90% of the pixels of 67% of all the objects present in the ground truth. Overall, this study showed that, when considering semantic segmentation of aerial RGB imagery, 1) height information can improve segmentation results, 2) adding height information through data fusion can result in a higher segmentation quality than when data stacking is used, and 3) providing relative height to a network, rather than absolute height, can improve semantic segmentation quality.

Acknowledgements

I would like to express my gratitude to my supervisors at TU Delft, Balázs Dukai and Ravi Peters, for their suggestions, advice and guidance throughout the execution of my research work. In addition, I would like to thank my co-reader, Jantien Stoter, for her constructive feedback and useful questions. Furthermore, I would like to thank Sven Briels from READAR for his technical supervision and helpful ideas. Finally, I would like to express my gratitude to Jean-Michel Renders, also from READAR, who gave me invaluable help by sharing his knowledge trough answering all my question on deep learning.

Contents

List of Figures

List of Tables

Acronyms

AHN Actueel Hoogte Bestand . . . 22
BGT Basisregistratie Grootschalige Topografie . . . 1
BN Batch Normalization . . . 15
CNN Convolutional Neural Network . . . 1
CRS Coordinate Reference System . . . 32
DEM Digital Elevation Model . . . 1
DSM Digital Surface Model . . . 1
DTM Digital Terrain Model . . . 22
FCN Fully Convolutional Network . . . 8
IDW Inverse Distance Weighting . . . 31
IoU Intersection over Union . . . 24
IRRG Infrared Red Green . . . 13
MFB Median Frequency Balancing . . . 11
mIoU mean Intersection over Union . . . 12
NAP Normaal Amsterdams Peil . . . 20
nDSM Normalized Digital Surface Model . . . 13
NDVI Normalized Difference Vegetation Index . . . 13
ReLU Rectified Linear Unit . . . 9
SGD Stochastic Gradient Descent . . . 22
SPP Spatial Pyramid Pooling . . . 31
TIF Tagged Image File . . . 25

1 Introduction

Semantic segmentation can be described as the process in which every pixel of an image is associated with a class label. This process allows for the division of an image into meaningful, non-overlapping parts [Zhu et al., 2016]. Automatic semantic segmentation of aerial imagery can be useful for many different types of applications that currently require extensive manual work by experts, which is time-consuming and costly. Applications are present in fields as mapping of land cover, object detection, land-use analysis and change detection [Saito et al., 2016; Kampffmeyer et al., 2016].

For example, ensuring high quality for topographic maps, such as the large scale dataset Basisregistratie Grootschalige Topografie (BGT) of the Netherlands [Kadaster, 2019], requires regular updating of mutations. Current change detection techniques are often based on visual comparison of aerial images by experts. Automatic semantic segmentation can automate part of this process by semantically segmenting images of the same location of two different years. The segmentation results can be subject to simple overlay operations in order to detect mutations.

Even though semantic segmentation is researched by many, the topic remains challenging. The constantly increasing spatial and spectral resolution of remotely sensed imagery can be considered as one of the main difficulties. This high resolution has the benefit of being able to capture small details such as small objects. However, it also complicates the semantic segmentation process by introducing higher imbalances in class-distributions, large variance between classes and small differences within each class [Wang et al., 2016; Yuan et al., 2016].

In the last couple of years, the deep learning revolution has stimulated the use of deep architectures, usually a Convolutional Neural Network (CNN), to successfully tackle general semantic segmentation problems (i.e. Long et al. [2015]; Hariharan et al. [2014]; Feng Ning et al. [2005]), including remote sensing related ones (i.e. Kampffmeyer et al. [2016]; Paisitkriangkrai et al. [2015]; Saito et al. [2016]). When well trained, these algorithms act as non-linear functions that have the ability to take an image as an input, and provide a segmented version of this image as an output. CNNs have shown to outperform traditional computer vision and machine learning approaches in terms of accuracy and in some cases efficiency [Garcia-Garcia et al., 2017].

The original focus in semantic segmentation has been on two-dimensional imagery. However, the fast growing technological development in acquiring and analyzing 2.5D or 3D data, allows for the introduction of a new dimension next to RGB information [Garcia-Garcia et al., 2017; Qin et al., 2016]. This information is often available in the form of a depth map, Digital Elevation Model (DEM) or point cloud. It is believed that the added 2.5D or 3D information, has the ability to improve semantic segmentation results [Qi et al., 2017; Qin et al., 2016]. Even though inclusion of three dimensional information in semantic segmentation problems has been researched for regular imagery (i.e. Qi et al. [2017]; Gupta et al. [2014]; Couprie et al. [2013]), the inclusion of pixel-level height information to improve the quality of the semantic segmentation is less represented in the current literature. Especially the extent to which height information contributes to segmentation quality currently remains less researched. Therefore, this study aims to examine the added value of height information, Z, to semantic segmentation of aerial imagery. It is strived to explore the capability of CNNs to automatically, semantically segment RGB-Z aerial imagery.

For this study 10 cm resolution, true ortho imagery of the city of Haarlem in the Netherlands is used. This imagery is developed by the company READAR[1]. True ortho imagery is aerial imagery that is corrected for relief displacement [Sheng et al., 2003]. Furthermore, height information provided to the networks will be derived from a Digital Surface Model (DSM) generated from the same imagery, also by READAR.

[1] https://readar.com/

1.1 Objectives & research questions

The main objective of this study is to generate a CNN model that performs automatic, pixel-level segmentation of remotely sensed imagery, potentially complemented with height information. Every pixel of an aerial image is provided with a class label. The model should provide high resolution topographic maps as output. Garcia-Garcia et al. [2017] showed that this problem can be formulated as the following; a procedure is sought to assign a state given in the predefined label space $L = \{l_1, l_2, ..., l_k\}$, to each element, present in a set of random variables $X = \{x_1, x_2, ..., x_N\}$. Every label l represents a distinct object or class, such as building, road, water etc. k represent the amount of possible labels and generally X represents a two dimensional image containing W x H = N pixels (x), each consisting of three bands (i.e. RGB). In this study, due to the addition of height information, three dimensional imagery is used. This added information can be considered as an extra, fourth band.

The second objective of this study comprises the analysis of the added value of the included height information. Studies have concluded that inclusion of 3D or 2.5D data can improve segmentation quality of natural imagery (i.e. Couprie et al. [2013]). However, this conclusion can not be adopted for the auxiliary value of height information to segmentation of remotely sensed data, without any further consideration.

Finally, if height information shows to be a useful addition for semantic segmentation, the last objective of this work is to explore in what way the height information can best be presented to the algorithms. This includes both the approach used to combine height information with RGB information, as the form in which the height information is delivered (i.e. absolute height or relative height).

In order to reach the described objectives, this study strives at answering the following question;

To what extent can convolutional neural networks be used for automatic semantic segmentation of RGB-Z aerial imagery?

In order to answer this question, the following sub-questions are specified;

- Which neural network architectures are a suitable starting point for semantic segmentation of aerial RGB-Z imagery?
- To what extent does the addition of height information improve semantic segmentation results?
- For which classes is the segmentation most successful; for `building`, `road`, `water` or `other`?
- How does the performance compare of different approaches on combining height information with RGB information (*stacking* and *fusion*) in a network?
- What type of height information provided to a network leads to the most accurate results?

1.2 Scope

Considering the scope of this study, the following should be noted;

- For this research work it is aimed to exploit existing algorithms and to make adjustments to let the existing implementations fit to the problem of this study. The goal is *not* to develop a new CNN architecture to perform the semantic segmentation. Therefore, combining several architectures is also out of scope of this study.
- This study aims to explore the potential of RGB-Z imagery. Experiments including imagery of other parts of the spectrum, i.e. infrared or near-infrared, are beyond the scope of this work.
- Imagery other than aerial imagery, such as street view imagery, is out of scope of this study.

- The focus of this study is on the generation of a model that performs well on the city of Haarlem. Therefore, imagery belonging to geographic locations other than Haarlem are not taken into consideration. However, it is strived to generate a model that is easily adapted or fine-tuned to other locations.

1.3 Report structure

The remainder of the report is structured as follows;

Chapter 2 gives an overview of the theoretical background required for a complete understanding of the rest of this research work. In addition a review is provided on related studies. This includes an introduction to deep learning and CNNs, as well as researched approaches on semantic segmentation of aerial RGB-(Z) imagery.

Chapter 3 presents the developed methodology together with argumentation on the design choices of this methodology.

Chapter 4 provides information on the used datasets, describes implementation details of the methodology and presents the experimental setup of this study.

Chapter 5 presents the results of the experiments, alongside with an in-depth analysis of these outcomes.

Chapter 6 discusses the main limitations of the methodology of this study and provides answers to the research questions. In addition, recommendations are given for future work.

2 State of the art

In this chapter an overview is provided on the theoretical background related to this thesis, alongside with a review on related studies. Firstly, relevant concepts on deep learning will be explained (Section 2.1). Subsequently, CNNs and their role in semantic segmentation are discussed (Section 2.2), followed by an overview of the addition of 2.5D and 3D information to the input of CNN algorithms (Section 2.3). Hereafter, studies on semantic segmentation of remotely sensed data are discussed (Section 2.4). Lastly, four different neural network architectures, which will be used in this research work, are described in more detail (Section 2.5).

2.1 Deep learning

Deep learning comprises a class of techniques in the field of machine learning in which computational models consisting of multiple processing layers, called neural networks, learn to represent data with different levels of abstraction [LeCun et al., 2015]. A neural network consists of an input layer, an output layer and one to many hidden layers. Each layer of a neural network consists of multiple neurons (Figure 2.1 and Figure 2.2). Each neuron can be seen as a feature, and is a mathematical operation which has its own learnable weights and biases. Input is provided to each neuron, which is multiplied by its corresponding weight and then summed. Hereafter, the function corresponding to the neuron, called the activation function, is applied to the result. The bias is an extra parameter that allows to shift an activation function to the right or left, through the addition of a constant to the input. The output of this function is passed on to neurons in consecutive layers [Hush and Horne, 1993]. Key to deep learning is that these layers of features used in the network are not provided, but are decided on by the network itself, using general-purpose learning [LeCun et al., 2015].

At each layer in the network, the input data is transformed to a higher level of abstraction. Very complex functions can be learned when enough of these transformations are executed. In the case of classification, representations of higher levels of abstraction allow for the elimination of unimportant variations and amplification of parts of the input that are important for distinction [LeCun et al., 2015].

2.1.1 Training a network

The parameters (weights and biases) in a neural network are learned through a training process. Neural networks are most often trained through supervised learning. For example, a goal could be to build a model that performs classification of images which contain a car, a face, a dog or a tree. The first step comprises the generation of a dataset containing images of these four classes and labeling every image with the corresponding class. The labels of the images are the ground truth. When RGB imagery is used, the input data for the neural network is provided as three 2D arrays (tensors), giving pixel intensities of the three channels. Throughout the training procedure, an image is presented to the network. Then, the network calculates an output, which is provided as a vector of scores; one score for each class. It is aimed to get the highest score for the class that corresponds to the ground truth label [LeCun et al., 2015].

A loss function is used to calculate the error between the output scores of the network and the desired output scores yielded from the ground truth. Training examples are usually presented to a network in batches, for which the loss is averaged over the batch. As an attempt to reduce the calculated loss, the weights and biases are updated with the help of an optimizer. The role of the optimizer is minimizing

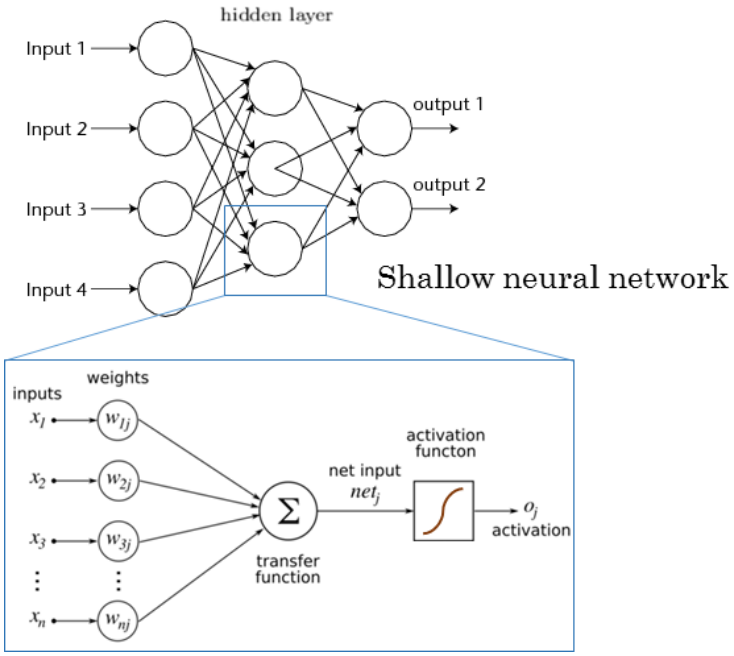


Figure 2.1: A diagram of a simple (shallow) neural network, containing only one hidden layer. Figure from RSIP Vision [2018].

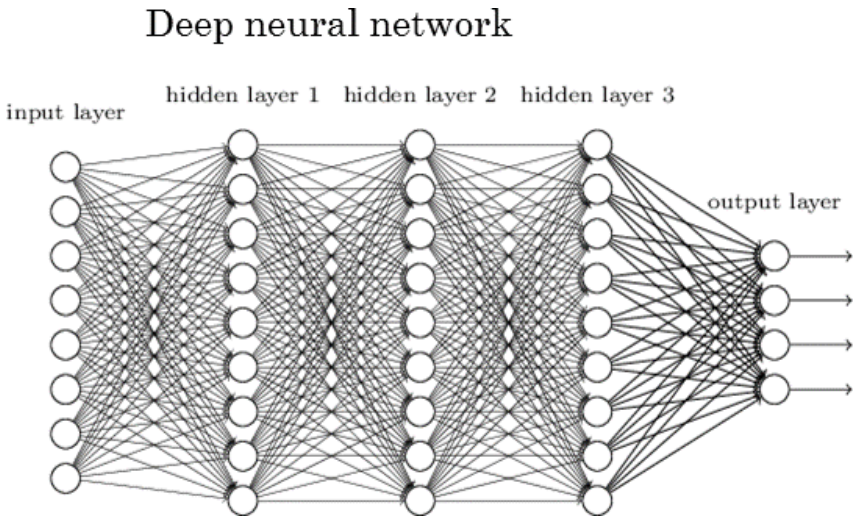


Figure 2.2: A diagram of a deep neural network, containing several hidden layers. Figure from NYU [2019].

the cost function. A gradient vector is generated that indicates for each weight by what value the calculated loss would decrease or increase if the weight is increased by a small amount. The gradient vector is multiplied by the learning rate α and the weights are then adapted in the direction which is opposite to the gradient vector [LeCun et al., 2015]. Different optimizers take different approaches in using the learning rate α, which is a hyperparameter that is specified in advance. However, the general remark can be made that a too small learning rate can make training slow, whilst a too large learning rate may lead to failure of convergence, or even divergence, due to the overshooting of minima [Zeiler, 2012]. The general formula for the updating of a weight W is provided in Equation 2.1.

$$W' = W - \alpha * gradient \tag{2.1}$$

During training this process is repeated; 1) the network is fed with the vectors corresponding to a batch of examples, 2) these vectors are fed forward through the network using the current weights and biases, 3) the loss function is used to calculate the error of the output, 4) these errors are propagated backwards through the network, 5) errors are used to modify the weights and bias in order to minimize the loss in the future, 6) the way the modifications are executed is described by the optimizer, 7) the process continues until the average loss stops decreasing. One iteration comprises one forward plus one backward pass of one batch. One complete forward and backward pass of all the training examples is called an epoch [LeCun et al., 2015]. Which loss function, optimizer, learning rate and what batch size is used, is specified in advance.

After termination of the training, the performance of the model is assessed on a different set of examples, which it has not seen before. This allows to examine the generalization capability of the model.

2.2 CNNs for semantic segmentation

2.2.1 Convolutional Neural Networks

When compared to ordinary neural networks, CNNs are a distinct type of networks often used for image analysis. These types of networks are specialized in detecting patterns and decipher them. Key to CNNs are the convolutional layers, which consist of filters that perform the pattern detection. For each convolutional layer, the amount of filters are specified. Each filter is specialized in detecting a specific pattern [LeCun et al., 2015]. Filters in early layers of a CNN detect simple geometric features, such as edges, corners, circles or squares. The deeper the layer of the network, the more sophisticated the filter. Instead of detecting simple features, in later stages more complex patterns can be detected, such as eyes or feathers. In even further stages, complete objects can be detected such as dogs and cars. A filter is a matrix with predefined dimensions. Before starting a training, the values of these matrices are initialized with random numbers. The filters 'convolve', or slide over, the complete input (image) and produce the dot product of the filter and the input pixels. How many pixels at a time the filter shifts, is given by a parameter definable per layer, called the stride. In order to also fully exploit the input information at border pixels, padding can be used. Padding is the addition of extra pixels, generally with the value 0, to the image before it is being processed by a filter. Consequently, the kernel has sufficient space to process the whole image, leading to preservation of border information. Figure 2.3 provides an example of the application of a 3x3 filter to an image. Equation 2.2 provides the calculation of the output dimensions of a convolutional operation. I represents the input dimensions (assumed to be a squared image: width = height), O represents the output dimensions, K represents the kernel size (i.e. 3 when kernel is 3x3) and $\lfloor \, .. \, \rfloor$ represent a floor division.

$$O = \lfloor \frac{I - K + 2 * padding}{stride} \rfloor + 1 \tag{2.2}$$

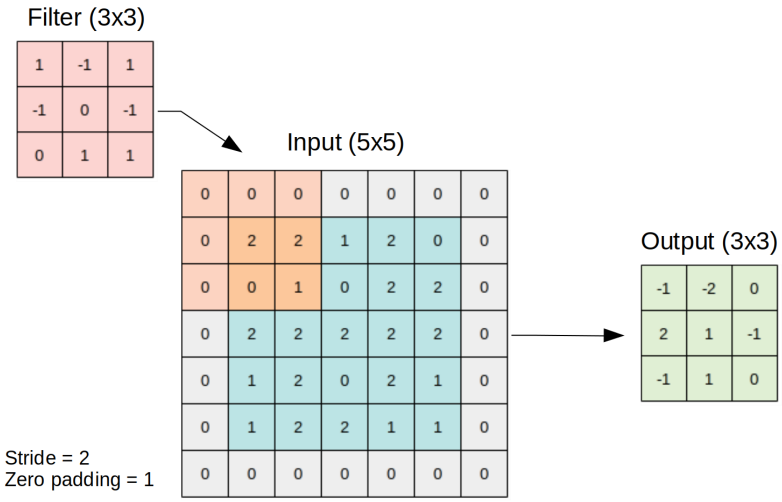


Figure 2.3: A filter with kernel size 3x3 and a stride of 2 is applied to a 5x5 image with a zero padding of 1.

After the filter has slided over the complete input, an output has been generated which is a matrix of the stored dot products. This output is a new representation of the input. A non-linearity equation is applied to this new representation, whereafter it is passed on to the next layer of the network [LeCun et al., 2015].

In traditional CNNs, fully connected layers are present after the convolutional and pooling layers (see Section 2.2.2). These layers have complete connections to all the activations in the previous layer. Their role is to convert the 2D output of all previous operations into a 1D vector from which classification can be derived [Voulodimos et al., 2018].

2.2.2 Allowing for semantic segmentation

CNNs are often used for semantic segmentation, especially since Long et al. [2015] have introduced Fully Convolutional Network (FCN)s [Audebert et al., 2018]. In the semantic segmentation task, both global and local information needs to be exploited. Global information tells 'what' is present, whilst local information is essential in solving 'where' the observed is present. The features in a network contain both the semantical and the location information in a hierarchy which can be thought of as a pyramid, starting from local (in first layers) to global (in deeper layers). The challenge is to find a way to extract or decode both these types of information from this 'feature pyramid'.

Classical classification networks (i.e. AlexNet and VGG net) solely have the goal to distinguish the presence of a class. In their work, Long et al. [2015] have converted these classical classification networks into FCNs. These networks allow for input imagery of arbitrary size and output correspondingly-sized, pixel-level, semantic segmentation. The FCN architecture replaces the fully connected layers of traditional CNNs by convolutional layers. Consequently, the classification networks can produce low resolution class presence heatmaps (Figure 2.4). These heatmaps are upsampled with the use of deconvolutions (reverse of convolution) that are initialized with billinear interpolation filters. In this study, experiments will be executed with one of the architectures proposed by Long et al. [2015]. Therefore, more details on their work is provided in Section 2.5.

As mentioned before, in semantic segmentation, one is not only interested in classification, but also in the projection of the classification onto pixel space. A general network architecture for semantic segmentation consists of an encoder network, connected to a decoder network. The encoder fulfills the role of classification, whilst the decoder ensures dense classification by projecting the classification onto

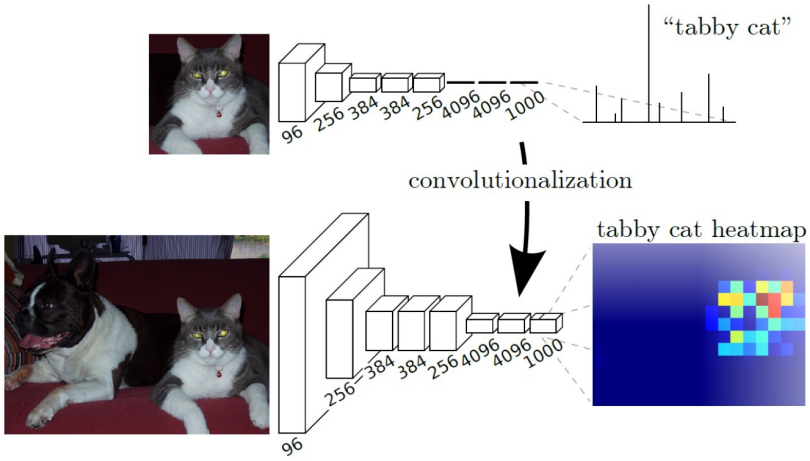


Figure 2.4: Using convolutional layers to replace fully connected layers allows classical classification networks to generate class heatmaps. Figure from Long et al. [2015].

pixel space. Through down-sampling, the model aims to understand what is present in the image. However, information on where it is present is then lost. The decoder aims at recovering this lost information. Currently, architectures mostly differ in their decoder mechanism [Shah, 2017].

In parallel to the developments of FCN-based architectures, architectures based on residual learning have also shown to be successful in semantic segmentation tasks [He et al., 2016]. These architectures are adjusted versions of ResNet classification networks, which allow for state of the art performing semantic segmentation (i.e. Zhao et al. [2017] and Pohlen et al. [2017]).

In general, four basic types of layers can be distinguished that are commonly used in CNNs for semantic segmentation, namely;

- **Convolutional layer**: consists of simple filters which contain learnable parameters. Each neuron in the layer searches for a specific pattern. As the aim is to search for the same patterns throughout the whole input, the learnable weights and biases of the neurons of the output are shared [Liu et al., 2017].
- **Transposed convolutional layer**: These layers allow for upsampling of the input; the dimensions of the input are increased. The parameters can be based on simple bilinear interpolation or they can be learned [Long et al., 2015].
- **Non-linear function layer**: often referred to as 'activation function' and usually present after a convolutional layer. This type of layer adds non-linearity to the network, by introducing for example the Sigmoid function or the Rectified Linear Unit (ReLU) function; $f(x) = max(0, x)$. ReLU is currently used most often in deep learning research [Glorot et al., 2011]. This introduction of non-linearity allows the network to represent a more complex function [Liu et al., 2017].
- **Spatial pooling layer**: uses a filter to reduces the size of the input. Functions commonly used are max, sum and mean [Saxe et al., 2011].

2.3 2.5D and 3D information

The rise of technological developments of sensors and algorithms for 3D (or 2.5D) data acquisition, generation and analysis, has led to a large increase in access to 2.5D and 3D information. This information is often available in the form of a depth map, DEM or (LiDAR based) pointcloud [Qin et al., 2016].

Even though the original focus of semantic segmentation in the literature has been on two-dimensional imagery, these technological developments allow for the inclusion of an extra dimension next to the regular photometric data [Garcia-Garcia et al., 2017]. It is believed that depth or height data contains complementary information to RGB channels and contains valuable information on structure of the scene [Hazirbas et al., 2017]. Even though generally class objects can be distinguished by their color and texture, which is encoded in RGB information, it is possible that objects belonging to different classes have comparable appearances. The addition of depth or height information potentially reduces the uncertainty of the semantic segmentation of these types of objects [Hazirbas et al., 2017]. Several studies using imagery plus depth information (RGB-D) for scene segmentation, have shown an increase in labeling precision when compared to solely using RGB data [Garcia-Garcia et al., 2017]. For example, Couprie et al. [2013] showed with their study on indoor scenes that semantic segmentation of classes having comparable appearance, depth and location, is improved when depth information is added. However, it was also concluded that for classes with a high variability of the depth values, using solely RGB leads to better results than when depth information is included. Consequently, the optimal way to incorporate depth (or height) information is still a point of discussion.

Different approaches have been proposed for incorporating 2.5D or 3D information. In these approaches, two main groups can be distinguished, namely *data stacking* and *data fusion*. The first group comprises the stacking of the extra 3D or 2.5D information to the RGB channels. If not yet supported, networks can be adjusted to allow for the input of (an) extra band(s). Hereafter, using a four (or more) channel input, networks can be trained on RGB-D (or -*Z*; height) data (see i.e. Liu et al. [2017]). An alternative approach comprises fusion of photometric and 2.5D or 3D data. In this case, feature extraction from the 3D or 2.5D data occurs separate from feature extraction from RGB data. The information is then combined in a later stage in the architecture (see i.e. Eitel et al. [2015]).

Furthermore, it should be noted that the added value of depth information on indoor scenery segmentation can not necessarily be considered as equal to the potential auxiliary value of height information to segmentation of remotely sensed imagery. Even though inclusion of 3D data to improve semantic segmentation results in natural imagery has been investigated, the added value of height information to semantic segmentation of aerial imagery is less represented in the current literature.

2.4 Semantic segmentation of remote sensing data

Since computer vision solutions have been significantly improved by the use of deep learning, these techniques are widely adopted to tackle remote sensing related problems [Audebert et al., 2018]. Two types of approaches are distinguishable in current studies on semantic segmentation of (aerial) imagery; patch-based methods and pixel-based methods. In the following paragraphs, both method types will be discussed alongside with different approaches on incorporating height information into the algorithms.

2.4.1 Patch-based methods

Patch-based methods either produce coarse maps, with one label per patch, or use a small window to construct a label for each pixel independently. Consequently, with the latter, labels assigned to each pixel are only based on its near surrounding pixels. In addition, the process is slow and computationally expensive [Sermanet et al., 2013; Audebert et al., 2018]. An example of a patch-based approach is the work of Saito et al. [2016], who used a five-layered CNN to automatically detect objects from aerial imagery. Their goal was to generate a multi-channel label output from the input image, with one channel per class. Even though promising results were obtained, only the classes `building` and `road` were included and no height information was incorporated. This study did show that predicting classes simultaneously can lead to a higher accuracy than when each class is predicted separately.

As an attempt to improve the expensive and slow patch-based labeling process for very high resolution dense semantic segmentation of urban scenes, Campos-Taberner et al. [2016] used a superpixel-based labeling approach. Superpixels are regions in an image that are perceptually uniform [Liu et al.,

2011]. In their work Campos-Taberner et al. [2016] used a combination of patch-based segmentation methods and an unsupervised pre-segmentation. His reseaerch work has led to several other initiatives using superpixels, such as the multi-scale approach of Lagrange et al. [2015].

2.4.2 Pixel-based methods

When compared to patch-based methods, pixel-based methods have a different approach by inferring the labels for all of the pixels at the same time. When semantically segmenting remote sensing imagery, this type of method has outperformed patch-based methods (see Kampffmeyer et al. [2016] and Volpi and Tuia [2016]). Therefore, more recently, semantic segmentation of aerial imagery has moved towards FCN models [Audebert et al., 2018; Maggiori et al., 2017; Volpi and Tuia, 2016]. By directly performing pixel-wise classification, FCNs have shown to be suitable for semantic segmentation of aerial imagery. FCNs have the ability to detect the spatial relationships between classes without pre-processing of the imagery (such as superpixel segmentation). In addition, they can output dense predictions of high resolution [Audebert et al., 2018]. Two studies that worked with FCN pixel-based approaches for the semantic segmentation of aerial imagery are Kampffmeyer et al. [2016] and Liu et al. [2017]. Both these studies use a data stacking approach to include height information in their algorithms. In their research work, Kampffmeyer et al. [2016] compared a patch-based approach with a pixel-based approach and used the the ISPRS Vaihingen 2D semantic labeling dataset [ISPRS, 2018a]. This dataset contains six classes, namely; `impervious surfaces`, `building`, `low vegetation`, `tree`, `car`, `clutter/ background`. Their results showed that the pixel-based (FCN) implementation (`building` F1 = 0.9581) generally outperformed the patch-based (PB) implementation (`building` F1 = 0.9501).

In addition, Kampffmeyer experimented with median frequency balancing incorporated into the cross-entropy loss function (referred to as FCN-Median Frequency Balancing (MFB)), to take class imbalance into consideration. This approach provided the best average F1 score results (average F1 = 0.9084, `building` F1 = 0.9530). Finally, it was concluded that combining all three models into one pipeline (PB + FCN + FCN-MFB) retrieved even better results than when compared to individual architectures. This conclusion was based on considering accuracy for small objects while still maintaining a high overall accuracy. Nevertheless, the average F1 score retrieved from this implementation (average F1 = 0.8879, `building` F1 = 0.9577) did not exceed the FCN-MFB implementation. Some important remarks need to be made when considering these results. Firstly, it should be noted that these resulting performance measures presented are calculated on the validation data and not on an independent test dataset. It is assumed that the validation data is therefore used to both tune the parameters and to provide final results. However, no elaboration on this matter is present in the paper. Furthermore, to lower the negative effect of class boundaries, ground truth data for which boundaries were eroded were used during performance measure computation. This erosion process comprises the elimination of boundary pixels, leading to the ignoring of boundaries when performance measures are calculated. Finally, the class `clutter/ background` was not included in the calculations. Logically, these methodological choices have resulted in relatively higher values for the F1 scores.

Liu et al. [2017] also designed their own architecture for semantic segmentation of remotely sensed data. Their implementation integrates an 'inception' and 'residual' module into the conventional encoder-decoder paradigm. The inception module comprises the collection of filters of different sizes into one layer, enabling the gathering of information from receptive areas of different scales. The residual module allows to directly feed forward information from the encoder to the decoder. Liu et al. [2017] used the same dataset and classes as the work of Kampffmeyer et al. [2016]. With their best performing architecture without post-processing, Liu and his colleagues retrieved an average F1 score of 0.8752 and an F1 of 0.9466 for the class `building`, on the boundary eroded ground truth data. Using the same architecture but then non-eroded ground truth data, an average F1 score of 0.8341 was achieved and an F1 of 0.9237 for the class `building`. The class `clutter/ background` was not included in the calculation of average F1. These result show the influence that using eroded ground truth data has on the performance measures; the F1 values increased with more than 4% for the average F1 and more than 2% for the F1 score of the class `building`. Furthermore, again, the paper does not mention the use of an independent test dataset for calculation of the performance measure. Therefore it is possible that both the parameter selection procedure as the final inference is done on the validation set.

Both the studies of Kampffmeyer et al. [2016] and Liu et al. [2017] did not examine the added value of the included height information. They solely focused on optimizing the semantic segmentation while using the extra (stacked) height band. This thesis will assess the added value of height information. In addition, both Kampffmeyer et al. [2016] and Liu et al. [2017] only added the height information by stacking it to the other bands, whilst this thesis also explores an alternative approach that involves fusion of height information.

Even though the objects of interests, or classes, in both the researches of Kampffmeyer et al. [2016] and Liu et al. [2017] did not overlap with the objects of interests in this proposed research work (apart from the class `building`), examination of their implementation, or even using it as a starting point would have been interesting. Unfortunately, the corresponding codes of the implementations could not be discovered. However, findings of all the mentioned papers will be taken into consideration while executing this study. For example, the conclusions from Kampffmeyer et al. [2016] that pixel-based approaches (i.e. FCN) outperformed patch-based approached for semantic segmentation of aerial imagery, will be taken into account when selecting architectures for this study. Furthermore, the findings of Saito et al. [2016] that simultaneous class prediction may lead to a higher accuracy than predicting each class separately, was an important reason for deciding on a simultaneous class prediction approach in this thesis.

2.4.3 Data fusion

The traditional way of providing extra information to a neural network, next to RGB, is through data stacking. This simply involves providing a four-band input to a network instead of a three-band input, whilst keeping the rest of the network's architecture the same. All the studies described above that included height information used this stacking approach. As discussed in Section 2.3, an alternative for incorporating height information is data fusion. An important research on this fusion approach is the work of Hazirbas et al. [2017], who developed FuseNet. The FuseNet architecture was originally developed for semantic segmentation of indoor scenes using RGB-D data. In his paper, Hazirbas argues that stacking the depth information to the RGB information as a fourth band and training a network accordingly, does not fully exploit the potential of the depth information. They suggest an alternative approach to extract more informative features on the structure of the scene, which are encoded in the depth data. The main idea of their CNN encoder-decoder type architecture is that the encoder part consists of two branches of networks. These two branches operate simultaneously, extracting features from RGB and depth images. At different depth levels of the network, retrieved depth features are fused into the feature maps of the RGB imagery. It is argued that feature maps retrieved by fusion of depth information into RGB information are more discriminant than when data stacking is applied. Low-level features, extracted in RGB and depth imagery, have the ability to complement each other. For example, an object lacking texture can be detected by its structure, whilst an object lacking structure can be distinguished through its color. When these low-level features of RGB and depth are combined, this also helps the network to detect high-level features, leading to a higher accuracy [Hazirbas et al., 2017].

In his study, Hazirbas experiments with fusion at different stages in his architecture. Using the SUN RGB-D dataset of indoor scenery [Song et al., 2015], it is concluded that FuseNet performs significantly better than the network trained with stacked RGB-D. The mean Intersection over Union (mIoU) (a widely used performance measure for semantic segmentation) of their best performing model FuseNet-SF5 is 0.3729. The model trained on stacked RGB-D achieved a mIoU of 0.3195. Consequently they state that when compared to stacking depth data to RGB, depth fusion is a better approach for retrieving informative features from depth information and combining them with color features. It should be noted that it is not specifically stated which RGB-D stacking architecture is used for this comparison.

In his work, Hazirbas successfully introduced depth fusion as an alternative to data stacking for indoor scenery imagery. However, it should be noted that this thesis works with height information rather than depth information, and remotely sensed imagery rather than natural indoor imagery. Therefore, findings for the added value of fusion of the extra 2.5D information rather than stacking, can not be adopted without further consideration.

Nevertheless, Audebert et al. [2018] did show that FuseNet can successfully be used to fuse LiDAR data with remotely sensed imagery. In their study, Audebert and his colleagues compared SegNet and ResNet implementations on either Infrared Red Green (IRRG) imagery or stacked DSM, Normalized Digital Surface Model (nDSM) (relative height) and Normalized Difference Vegetation Index (NDVI) with FuseNet based implementations. Early fusion (original FuseNet architecture) is compared to a late fusion approach, where prediction fusion through residual correction is executed rather than fusion of features. In addition, an adjusted version of FuseNet is proposed, referred to as V-FuseNet. In this architecture, instead of fusing the height information into the RGB encoder path, a third encoder path is introduced that does not treat the height information as auxiliary data. Rather than fusing the contributions by summing activations of the height encoder into the RGB branch, contributions are fused by using a convolutional block and a summation. In addition, Audebert and his colleagues have implemented the fusing properties of FuseNet into a ResNet-34 architecture, that was adapted to allow semantic segmentation (see He et al. [2016]). The results of their tests on the Vaihingen dataset[1] are presented in Table 2.1. The highest F1 score on the Vaihingen dataset for the class `building`, retrieved by their SegNet-RC approach, is 0.945.

Model	Input data	Average F1
SegNet	IRRG	0.893
SegNet	DSM/nDSM/NDVI	0.816
SegNet-RC	IRRG + DSM/nDSM/NDVI	0.892
FuseNet-SF5	IRRG + DSM/nDSM/NDVI	0.901
V-FuseNet	IRRG + DSM/nDSM/NDVI	0.903
ResNet-34	IRRG	0.891
ResNet-34	DSM/nDSM/NDVI	0.834
ResNet-34-RC	IRRG + DSM/nDSM/NDVI	0.891
FusResNet	IRRG + DSM/nDSM/NDVI	0.893

Table 2.1: Results of Audebert et al. [2018] on the validation data of the Vaihingen dataset. The addition of '-RC' denotes architectures with prediction fusion through residual correction.

Overall, SegNet based architectures performed slightly better than ResNet based architectures. In addition, ResNet required significantly more memory, especially when fusion is used. Furthermore, when considering the average F1 scores, early fusion showed a slightly better, but very similar performance when compared to late fusion (0.901 for FuseNet and 0.892 for SegNet-RC). Audebert and his colleagues showed that in comparison early fusion has the ability to learn stronger features, whilst late fusion can sometimes correct for errors on challenging pixels. Finally, the results show that V-FuseNet gives a similar performance when compared to FuseNet (average F1 scores of 0.903 and 0.901 respectively).

No implementation of V-FuseNet was publicly available and V-FuseNet did not show a remarkable improvement when compared to the original FuseNet of Hazirbas et al. [2017]. Considering the findings of the study of Audebert et al. [2018], it is decided to explore the original implementation of FuseNet for semantic segmentation of aerial imagery and height information in this thesis. More details on this selected architecture are provided in Section 2.5.4.

Even though in their study Audebert et al. [2018] did use FuseNet for semantic segmentation of remotely sensed data, no research was executed that allowed for comparison of a fusing approach with a stacked approach (i.e. IRRG-Z). Therefore, the added value of applying fusion to combine height features with color information, rather than using a different tactic, is not provided. This will be examined in this thesis. In addition, no experiments were executed with RGB imagery, solely with IRRG imagery.

[1] Same dataset as used by Kampffmeyer et al. [2016] and Liu et al. [2017], see ISPRS [2018a]

2.5 Architectures

In the following paragraphs, the four different architectures which are used in this thesis will be presented, namely FCN-8s, SegNet, U-Net and FuseNet-SF5.

2.5.1 FCN-8s

As discussed in Section 2.2.2, by replacing the traditional fully connected layers in CNNs with convolutional layers, Long et al. [2015] presented an approach to convert traditional classification networks into FCNs performing pixel-level semantic segmentation. The adjusted classification networks can now produce low resolution class presence heatmaps. These heatmaps are upsampled in the expanding path of the network with the use of deconvolutions that are initialized with billinear interpolation filters. To recover relevant segmentation information lost while the resolution is reduced in the encoder/contracting path, Long et al. [2015] have constructed an approach that has the ability to combine semantic information present in coarse, deep layers with information on appearance from fine, shallow layers. Their paper presents three self-developed FCN architectures, namely FCN-32s, FCN-16s and FCN-8s. These architectures are originally designed for object classification in natural imagery. These architectures differ in stride at the final prediction layer and the skip connections used to produce the output semantic segmentation. These skip connections allow for recovery of spatial information at the expanding path, through the merging of features that originate from different resolution levels in the contracting path [Drozdzal et al., 2016]. The three architectures support an input of arbitrary dimensions and perform pixel-level predictions. Using the PASCAL VOC2011 validation dataset [Everingham et al., 2010], FCN-8s showed the best performance and was able to capture the most detail. FCN-8s can be considered as a baseline architecture on which (improved) architectures have been built [Shah, 2017]. Nevertheless, FCN-8s is one of the selected architectures for this research work as the complete architecture, or parts of it, is used by several of the most successful participants in The ISPRS Semantic Labeling Challenge [ISPRS, 2018b; Duc, Minh and Viet, Sang, 2018; Duy, Tring, Van and Sang, Dihn, Viet, 2018] on the Vaihingen dataset [ISPRS, 2018a] mentioned before. In addition, the implementation of the FCN-8s architecture is relatively simple. This makes it a suitable starting point for understanding CNNs and the training process.

A schematic representation of the FCN-8s architecture is presented in Figure 2.5. The architecture consists of 5 max-pooling layers that perform downsampling and 15 convolutional layers. The output stride of the network is 8. Predictions from the pool 3 and pool 4 layer are forwarded and combined with (intermediate) output layers in order to restore details in the segmented output.

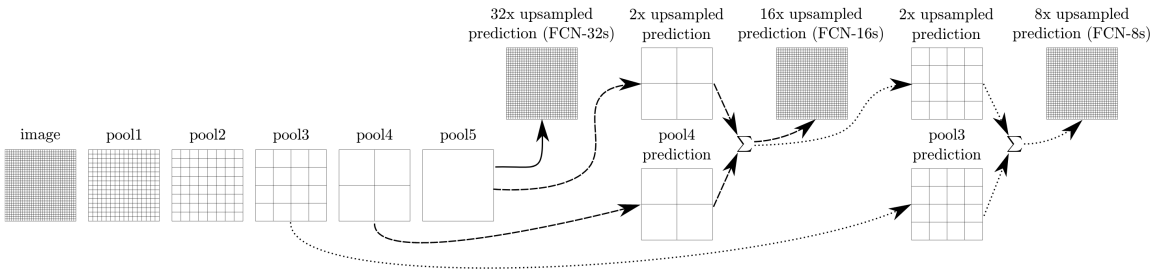


Figure 2.5: The pooling and prediction layers of the FCN-8s architecture, consisting of 4 max-pooling layers and 15 convolutional layers. FCN-16s and FCN-32s are also displayed. Figure from Long et al. [2015].

2.5.2 SegNet

The SegNet architecture, presented by Badrinarayanan et al. [2017], is a variation of the FCN architecture of Long et al. [2015]. Badrinarayanan and his colleagues argue that the FCN models can produce coarse

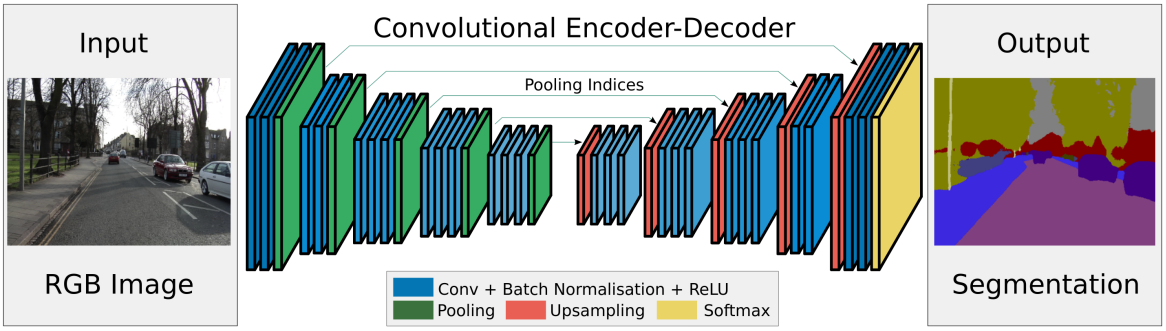


Figure 2.6: The SegNet architecture. The encoder consists of 13 convolution layers and 5 max-pooling layers. Using the transferred pooling indices, the decoder upsamples the encoder output and executes convolutions to densify the feature maps. Figure from Badrinarayanan et al. [2017].

outputs with fuzzy boundaries, mostly due to the reduction of feature map resolution during max-pooling and sub-sampling. They aim to tackle this issue and to find a way to map features of low resolution to input-resolution allowing for classification at pixel-level. In this process, features must be produced that are valuable for the localization of boundaries.

SegNet was primarily designed for applications focusing on the understanding of road scenery. For this task it is necessary to model appearance (i.e. building or road) and shape (i.e. pedestrians and cars). In addition, with road scenery it is important to understand the spatial-relationships that exist between different classes (i.e. sidewalks present next to roads). Furthermore, as most of the pixels in such scenery belong to large classes (i.e. `building` or `road`), smooth segmentation results are desired, whilst also allowing for detection of objects based on their shape, even when having a small size. In order to achieve this, good boundary representation is necessary [Badrinarayanan et al., 2017].

The SegNet architecture is selected for this thesis as even though the perspective of aerial imagery differs strongly from street-view imagery, it is believed that it can be considered as a similar semantic segmentation task. In both cases one is not interested in solely delineating one specific object out of the scene, but segmentation of objects of different shapes and sizes throughout the whole image is required.

SegNet consists of an encoder network that is connected to a decoder network and a final layer performing pixel classification. A schematic overview of the architecture is presented in Figure 2.6. The encoder network correspond to the first 13 convolutional layers of the object classification network VGG-16 [Simonyan and Zisserman, 2014]. It contains five convolution blocks, having two or three convolutional layers with kernel size 3x3 and a padding of one, connected to both a Batch Normalization (BN) layer and a ReLU layer. Every convolution block is connected to a 2x2 max-pooling layer. Consequently, the feature maps produced by the encoder have a resolution of W/32 x H/32, when the dimensions of the original image are WxH [Audebert et al., 2018].

For every encoder layer, there is a corresponding layer in the decoder network. Therefore, the decoder network also consists of 13 layers and has a symmetrical structure to the encoder. The decoder completes both the upsampling and the classification. The novelty of the SegNet architecture is the way in which the decoder performs upsampling to its lower resolution feature maps. The encoders pass on the indices computed in the max-pooling step to the corresponding decoders, which use this information to execute non-linear upsampling of their input feature maps. Consequently it is not necessary to learn to upsample. The resulting upsampled maps are sparse and are combined with the use of filters that can be trained and deliver dense feature maps. This process allows to preserve high-frequency information. This is especially useful for small objects that might otherwise be misclassified or misplaced [Badrinarayanan et al., 2017; Audebert et al., 2018]. In addition, the process of reusing max-pooling indices improves boundary localization, lowers the number of parameters and the process can be easily

implemented into any encoder-decoder architecture [Badrinarayanan et al., 2017]. However, the unpooling does result in loss of neighboring information. A schematic overview of this max-pooling and unpooling process is provided in Figure 2.7. The output of the final decoder is passed on to a multi-class softmax classifier that provides for every pixel the class probabilities. The softmax function is a simple function that takes a vector of K real numbers as input and normalizes these into a probability distribution of K probabilities. After applying the function, all elements will be in the range of [0-1] and the sum of the elements is 1.

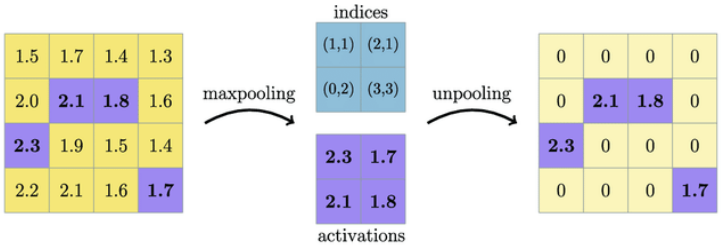


Figure 2.7: The impact of the max-pooling and unpooling operation. Figure from Audebert et al. [2018].

In comparison; the decoder of SegNet uses the max-pooling indices for upsampling the feature maps (without learning) and convolves the result with trainable decoder filters, whilst the FCN architecture of Long et al. [2015] performs upsampling by learning to deconvolve the input feature map of the decoder, and adds it to the encoder feature map (output of the max-pooling layer) that corresponds to it to deliver the output of the decoder (Figure 2.8).

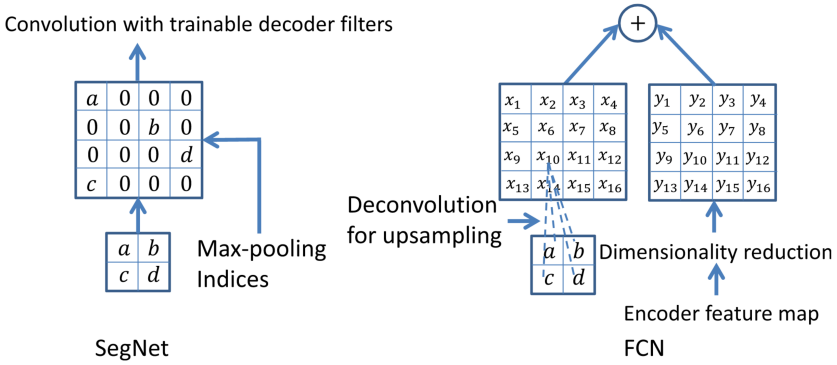


Figure 2.8: Comparison of SegNet and FCN Long et al. [2015] decoders. a, b, c and d represent values in a feature map. While the FCN architecture from Long et al. [2015] learns deconvolution filters to perform upsampling, SegNet uses max-pooling indices retrieved from corresponding encoder layers for upsampling. Figure from Badrinarayanan et al. [2017].

2.5.3 U-Net

The U-Net architecture of Ronneberger et al. [2015] was originally designed for image segmentation in the biomedical community. Its architecture is built upon the FCN architecture of Long et al. [2015]. Ronneberger and his colleagues have modified and extended the FCN architecture with the goal to allow it to work with very little training data and to increase segmentation performance. One of the main modifications is the addition of a large number of feature channels in the upsampling part of the network. This addition allows the network to pass on contextual information to layers with a higher resolution. No fully connected layers are present in the architecture, solely the valid part part of every convolution is used. Consequently, the output segmentation only consists of the pixels for which the complete context is available. Due to this approach, seamless segmentation of tiles is possible but it requires the input tiles to have overlap. Image border region pixel predictions are achieved by extrapolating the missing context by mirroring of the input image [Ronneberger et al., 2015].

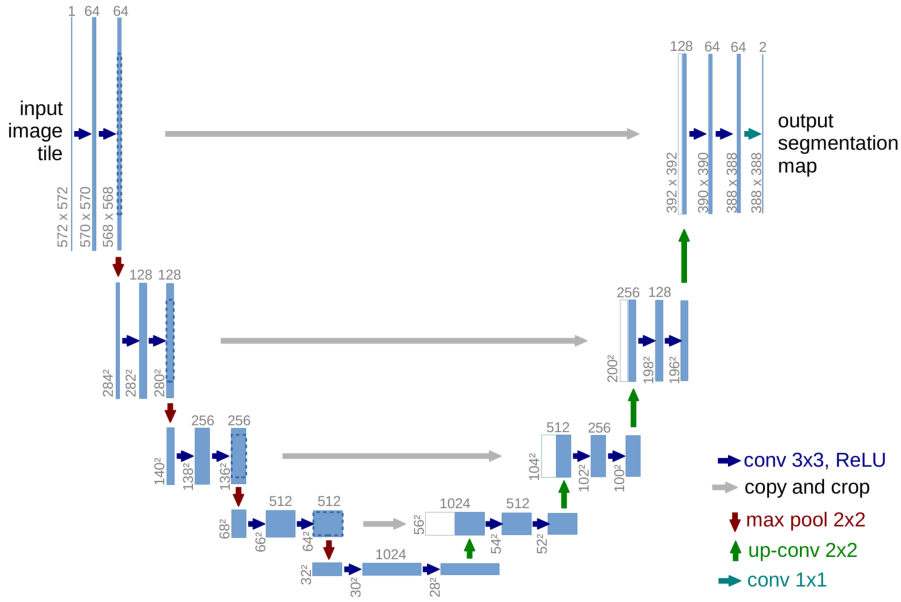


Figure 2.9: The U-Net architecture. Every blue box represents a multi-channel feature map. On top of every box the number of channels is given. The dimensions are given at the left lower edge of each box. A white box corresponds to copied feature maps. Figure from Ronneberger et al. [2015].

An overview of the U-Net architecture is presented in Figure 2.9. In total, 23 convolutional layers are present in the architecture. The encoder (left side of the U-shape, referred to as contracting path) consists of five blocks with two 3x3 unpadded convolutions, all followed by a ReLU. After each convolutional block, downsampling occurs through a max-pooling operation (2x2) with stride 2. The number of feature channels is doubled at each downsampling step. Every step in the decoder network (right side of U-shape, referred to as expansive path) entails an upsampling of the feature map, an up-convolution (2x2) halving the number of feature channels, a concatenation with the feature map from the encoder that is correspondingly cropped, and finally, two convolutions (3x3), both followed by a ReLU. Cropping is necessary as at every convolution border pixels are lost. Concatination of the feature maps from the encoder to the upsampled feature maps from the decoder, allows the decoder to retrieve important features that were lost during pooling operations in the encoder. Finally, a 1x1 convolution is used at the last layer of the decoder network for mapping the 64-elements feature vector to the correct number of classes [Ronneberger et al., 2015].

When compared to SegNet, instead of transferring and reusing pooling indices, U-Net transfers the entire feature map to the matching decoders and concatenates them to the by deconvolution upsampled feature maps of the decoder, so that the target details can be recovered [Badrinarayanan et al., 2017]. Transferring entire feature maps does cost more memory, but it preserves neighboring information that is lost when only passing on indices.

Even though a lack of training data is not an issue in this research work, and, as mentioned before, the U-Net architecture was originally designed for semantic segmentation tasks in the biomedical field, the architecture was selected for experimentation in this study as it has shown to be promising also in the field of remote sensing image segmentation. When considering the The Dstl Satellite Imagery Feature Detection competition of Kaggle [Kaggle, 2017], held in 2017, most of the participants, including the most successful ones, selected a U-Net (based) architecture. It should be noted that lack of training data was an issue in this competition, which might be a part of the explanation why this architecture was so often selected and successful. Even though some of the code of the winners of the competition was made available, the implementations are extremely case-specific and miss required information on what to adapt to apply the pipeline to a different problem or dataset. Code is not clean and it is explicitly mentioned that the scripts that are made available are not suitable for production environment. Therefore, it

is decided to use the baseline architecture for this study as a starting point for U-Net experimentation.

2.5.4 FuseNet-SF5

As discussed in Section 2.4.3, Hazirbas et al. [2017] developed the FuseNet-SF5 architecture for semantic segmentation of indoor scenes using RGB-D data. It was aimed to find an alternative for the data stacking approach through the introduction of data fusion. FuseNet-SF5 can be considered as a multi-modal architecture built on SegNet [Audebert et al., 2018]. Instead of one encoder branch, two encoder branches are present, which simultaneously extract features either from the RGB imagery or the depth images. The information is combined by fusing the feature maps retrieved in the depth branch into the feature maps corresponding to the RGB branch [Hazirbas et al., 2017]. A schematic representation of the FuseNet-SF5 architecture is given in Figure 2.10.

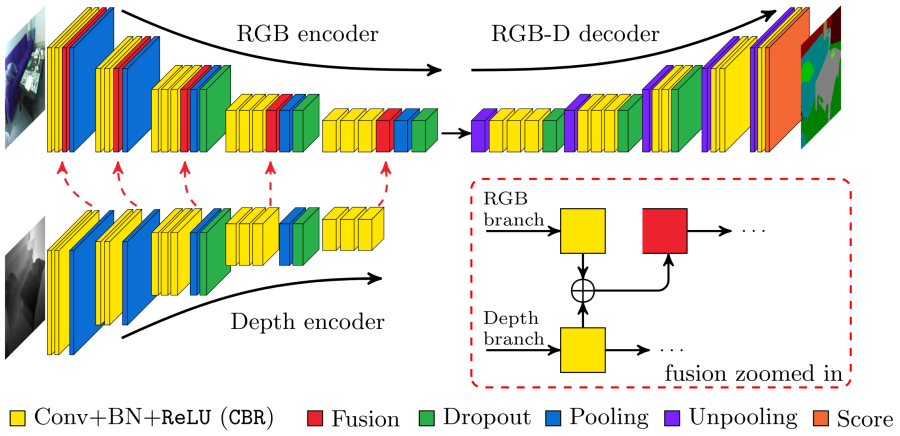


Figure 2.10: The FuseNet-SF5 architecture. Red arrows demonstrate the fusion of the depth feature maps into the RGB feature maps. Figure from Hazirbas et al. [2017].

The encoders of FuseNet-SF5 are similar to the 16-layer VGG classification network [Simonyan and Zisserman, 2014], without the fully connected layers fc6, fc7 and fc8, as these enhance the difficulty of upsampling. After each convolution layer in the network BN is applied for reduction of the internal covariate shift [Ioffe and Szegedy, 2015], followed by a ReLU. The BN layer ensures normalization of the feature maps with a zero mean and unit-variance, and hereafter applies scaling and shifting. During the training, the parameters connected to this scaling and shifting are learned. Consequently, no overwriting of the RGB features by the depth features occurs; the network learns how they can be optimally combined [Hazirbas et al., 2017]. The fusion layers in the FuseNet-SF5 network are present after a convolution-BN-ReLU block and before pooling layers. By element-wise summation, the discontinuities of the depth features maps are added to the RGB encoder branch, allowing for enhancement of the RGB feature maps. This fusion approach of FuseNet can also be implemented into other deep neural networks [Audebert et al., 2018].

A single decoder is used to upsample the output feature maps of the encoder part by applying memorized unpooling. Convolutional layers followed by BN and ReLU are also present in the decoder. During training, dropout of 0.5 is used in both the encoder and the decoder part of the network [Hazirbas et al., 2017]. Dropout is a technique to prevent overfitting of the model by randomly dropping units (neurons) from the neural network together with their connections during the training. This process prevents redundant co-adaptation of neurons [Srivastava et al., 2014]. Dropout is not used during inference.

3 Methodology

In this chapter the methodology of this study presented. A simplified illustration of the methodology is provided in Figure 3.1. A flowchart covering a more detailed version of the methodology can be found in Figure A.1. In this chapter, firstly, Section 3.1 describes the selection procedure of the used architectures. Hereafter, the steps on the generation of training and validation data are given in Section 3.2, followed by a description of the training process of the CNNs in Section 3.3. Next, the test data generation and the inference procedure is described in Section 3.4. Finally, Section 3.5 elaborates on the procedure of answering the research questions. Implementation details on the presented methodology are provided in Chapter 4.

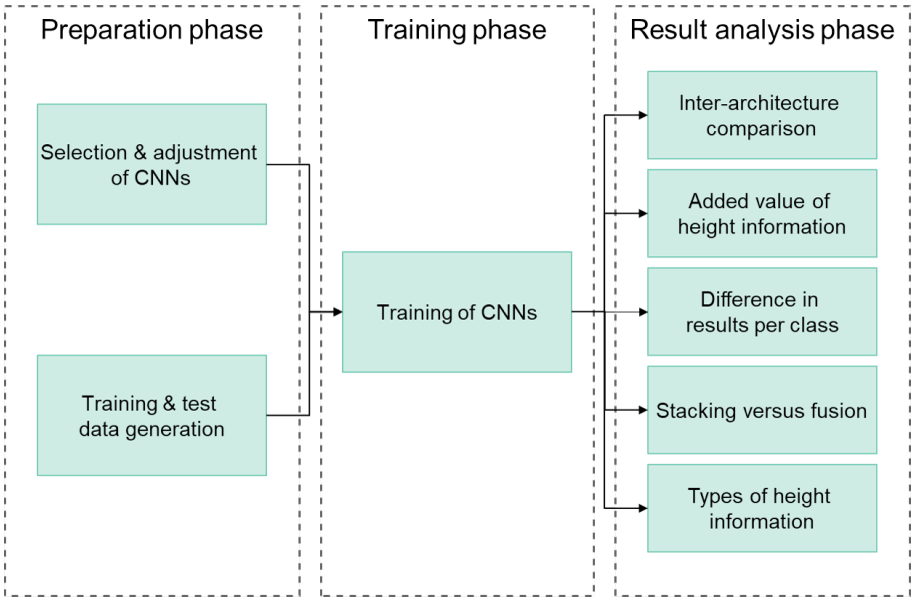


Figure 3.1: Overview of the methodology.

3.1 Selection of CNNs

Firstly, a selection is made of existing neural networks that are considered to be suitable for this study. This is done through a literature study and by examining code implementations corresponding to promising research papers. Encountered networks were considered to be suitable when adherent to the following criteria;

- Shown successful performance of semantic segmentation of any type of imagery. The performance is based on quality measures in research papers and performances shown at The Dstl Satellite Imagery Feature Detection competition of Kaggle [Kaggle, 2017] and The ISPRS Semantic Labeling Challenge dataset [ISPRS, 2018a].
- Source code is available online, without any license restrictions.
- Implementation is not too complex or too specific for one segmentation task and allows for use of own dataset.

- Implementation is done in Python.

As a high number of architectures exists, it was aimed to select the few most promising ones. This network selection procedure led to a state of the art literature review, presented in Chapter 2. The architectures used in this study are FCN-8s, SegNet, U-Net and FuseNet-SF5. If necessary, the used implementations were adjusted to allow for the support of the added height information. The rest of the network's architecture, such as the amount of layers and the number of nodes/neurons in each layer, was kept exactly the same (see Section 4.2.5). For FCN-8s, SegNet and U-Net a data stacking approach is used to accept the added height information, whilst FuseNet-SF5 works with a separate encoder.

3.2 Training and validation data

In order to train a network, example data is needed. For this study, data of the city of Haarlem in the Netherlands is used. An area of approximately 4km^2 (2.048x2.048 km) is selected, containing both urban and more rural areas (Figure 3.2). The training area is subdivided into smaller parts of 512x512 pixels, with each pixel being 10x10 cm, leading to 1600 tiles (40x40). Data corresponding to each tile is one training example. The dimensions of 512x512 is selected as 512 is a multiple factor of 2. Therefore, the images can be downsampled by the networks, for example through max-pooling, several times without the need of rounding off the dimensions to the nearest integer.

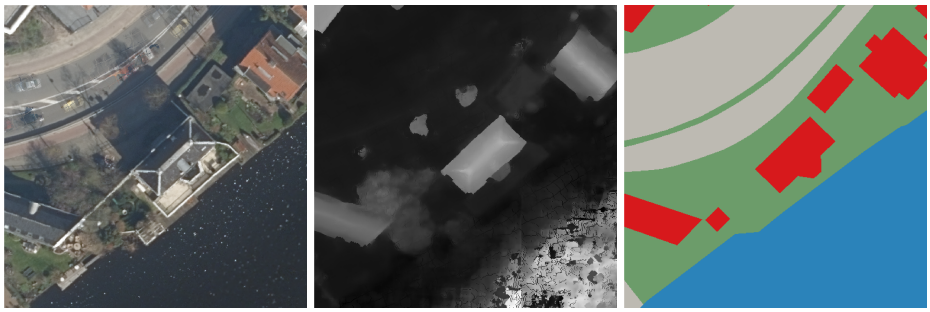
Each training example is a combination of an RGB true ortho image (see Section 4.1.2), plus pixel-level height information and a mask layer. Depending on the experiment performed, the height information is included or left out in the training of a network. The mask layer shows the ground truth; the correct semantic segmentation of the image. In order to retrieve this information the Dutch national topographic dataset BGT is used. Before using the dataset, an examination of the BGT was executed to asses its strengths and weaknesses to serve as a mask layer (see Section 4.1.1). In addition, this analysis allowed for the selection of the used classes in this study. A cleaned and rasterized version of this BGT is used. In this cleaned version, terminated objects are removed and no overlapping objects are present, by keeping only objects visible from the air. For example, at locations where a bridge is going over water, the bridge is kept and the water polygon is removed. This cleaned version of the BGT is provided by READAR. The dataset is reclassified into `building`, `road`, `water` and `other`. For more details on the BGT and the use of it in this study, see Section 4.1.1.

In this thesis, there is experimented with different types of height information input, namely absolute height relative to Normaal Amsterdams Peil (NAP), rescaled height ([0-1]) using Min-Max Feature Scaling and two approaches of relative height. All these types of height input are based on a DSM of READAR (see Section 4.1.2), which is matching to the true ortho imagery. The DSM has the exact same resolution and pixel locations as the true ortho imagery, and is cut in the exact same pieces as the RGB and mask data. Consequently, when the networks are trained with the additional height information, every pixel contains a separate value for the red band, the green band, the blue band and the height information. In addition, for every pixel one class label derived from the BGT is available (Figure 3.3). Further details on the generation of the different types of height information can be found in Section 3.2.1.

A random 80 percent of the examples are used for the actual training of the models, and the remaining 20 percent are used for validation. Whilst the actual training examples allow for the learning of model parameters (weights and biases), the validation data allows for assessment of model performance during the training and prevents overfitting of the model on the training data. In addition, the validation data is used for deciding on the tuning of the hyperparameters, which are the parameters set before the learning process starts. Model parameters, the ones that are learned, are never updated with the use of the validation data.

The U-Net implementation requires deviating input dimensions of 572x572. This dimensions are required as within the U-Net architecture unpadded convolutions are used together with cropping before max-pooling operations are applied. In order to get the same amount of training examples, the training area is made slightly larger (22,880x22,880 pixels of 10x10cm). This area is again subdivided into 1600

Figure 3.2: The training area (green) and test (red) area in Haarlem, the Netherlands.



(a) **(b)** **(c)**

Figure 3.3: A training example. (a) RGB true ortho (b) DSM (c) Ground truth. Red = building, gray = road, blue = water, green = other.

(40x40) non-overlapping tiles. As the output dimensions of U-Net differ from the input dimensions, the 1600 ground truth images are cropped so that the center 388x388 pixels are kept and match with the output prediction dimensions. The examples are again randomly subdivided in examples for training (80%) and examples for validation (20%).

3.2.1 Height approaches

As an attempt to exploit the height information provided by the DSM to its full potential, several approaches were examined (Figure A.2 and Figure A.3).

- **Absolute height**: As the height information provided in the DSM is a physical value provided in a physical unit (m relative to NAP), and therefore is in its purest form, it could be argued that altering the data could lead to a loss of valuable information. In addition, many convolutional operations are relative operations. Therefore, pre-processing of height might not be necessary. Consequently, the 'baseline' approach of using the height information is providing the pure values present in the DSM.
- **Rescaled height [0-1]**: Rescaling the height information from 0 to 1 ensures that the range of the distribution of the height values is equal to the ones of the normalized RGB input. This can be beneficial as during training the gradients are multiplied by the learning rate, before updating model parameters (weights). Consequently, deviating value ranges can lead to corrections of the weights that are proportionally different from each other, resulting in over- and under-compensation of errors (see Section 2.1.1). Nevertheless, when the Adam optimizer is used, individual learning rates per parameter are maintained, which potentially already solves (part of) the problem (see Section 4.2.7). However, as experiments are also executed with the Stochastic Gradient Descent (SGD) optimizer in this study, and the problem solving property of Adam for the deviating value distribution ranges is not certain, it is decided to use scaled height as one of the height approaches.

  In order to generate the rescaled DSM, the Min-Max Feature Scaling algorithm is used (Equation 3.1). This algorithm is applied to each tile/input image independently in the data loader, ensuring that all height values are rescaled to a value ranging from 0 to 1. In this algorithm X represents the original feature (height) value, $X\text{min}$ and $X\text{max}$ the minimum and maximum value of the feature (height) in the complete tile, and X' the new rescaled feature value. In a later stage of this study, experiments are also executed in which the Min-Max Feature scaling is based on the minimum and maximum value of the whole training or test area, instead of on each tile individually.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

- **Relative height (pixel-level)**: Rather than using absolute height, experiments are also performed with relative height. Ideally, relative height provides the actual height of the pixel object, as the height of the terrain is subtracted from it. Therefore, it could be argued that relative height is more informative than absolute height. On the other hand, as many convolutional operations are already relative operations, the use of relative height might be unnecessary or even detrimental.

  The pixel-level, relative height is generated by subtracting the Digital Terrain Model (DTM) of Actueel Hoogte Bestand (AHN)3 [Actueel Hoogtebestand Nederland, 2020] from the DSM. As the resolution of the DTM (0.5 m) differed from the resolution of the DSM (0.1 m), the DTM was first upsampled using QGIS 3.12.1 to match the cell size of the DSM. This was done by subdividing each pixel in the DTM into 25 (5x5) pixels with the same value. Even thought this approach does not alter the resolution of the DTM, it does allow for a pixel-level subtraction operation. In order to perform this subtraction, the GDAL raster calculator tool is used[1].

[1] https://gdal.org/programs/gdal_calc.html

- **Relative height (tile-level)**: As the resolution of the DTM and the DSM differ, pixel-level subtraction potentially leads to a more fuzzy representation of object boundaries in the height information. In addition, due to the flat nature of the area, terrain heights in Haarlem do not fluctuate strongly. Therefore, as alternative to the pixel-level subtraction approach previously mentioned, the median of the DTM is calculated at a tile-level, and then subtracted from every pixel in the DSM.

3.2.2 Data augmentation

After generation of the training data, data augmentation is applied. Data augmentation comprises different techniques that augment the size and the quality of the training data. This allows for the generation of deep learning models of a higher quality [Shorten and Khoshgoftaar, 2019]. Data augmentation is considered as an easy set of methods to reduce overfitting of a model on the training examples [Krizhevsky et al., 2017]. Examples of image augmentation techniques are geometric transformations such as rotation, zooming and flipping, but also adjustment of brightness and hue. Not every data augmentation technique is suitable for this study. For example, rotation is considered to be unsuitable as it may result in unrealistic shadows, which can provide a limitation to successful training of the network. In addition, as the scope of this study is Haarlem and the data that will be used as input for inference on trained models has similar spectral properties as to the training data, the focus of data augmentation in this thesis is on generating extra, realistic example data. Therefore, horizontal flipping is applied to the training examples (Figure 3.4). By mirroring the images across the horizontal axis, it is believed that the issue of unrealistic shadows is prevented. The horizontal flipping is only applied to the data that is used for model parameter learning and not to the validation data. Due to this data augmentation, the amount of training examples is doubled.

Figure 3.4: Horizontal flipping as data augmentation technique. The flipping is also applied to the height information and the mask layer.

3.3 Training of CNNs

Due to a limited availability of computational power and storage on an ordinary laptop, an external server is used for training and testing of the neural networks. The implementations of the selected architectures used in this study are equal to the architectures described in each of the corresponding research papers. All architectures are implemented using the open source machine learning framework PyTorch[2] (see Section 4.2).

The selected data stacking networks are first trained and tested using only the true ortho (RBG) imagery, without the height information. It is aimed to first achieve the highest possible performance of the networks on the aerial images, before including the height information. This procedure ensures a valid assessment of the added value of the height information to the segmentation results.

[2] https://pytorch.org

3.3.1 Tweaking of hyperparameters

An important part of the training process comprises the tweaking of the hyperparameters such as the initial learning-rate, batch size and number of epochs, and other design choices, such as the cost-function and the optimizer used. For example, models can be either pretrained on a different dataset, or not pretrained. Working with a pretrained model, also when trained on an unrelated dataset, could potentially save training time and lead to better results [Azizpour et al., 2015]. Details on the experimental setup of this thesis are provided in Section 4.3.

3.3.2 Performance measures

In order to assess the performance of a model, the output of a network needs to be converted to a final prediction: one class label per pixel. The output of a network is provided as four 2D arrays, one per class, that give for every pixel of the input image a score. The higher the score, the more likely that the pixel belongs to that class. Therefore, during prediction, every pixel receives the label that corresponds to the highest score for that pixel.

After this procedure, performance measures can be calculated. In this thesis, models will be assessed based on the (mean) Intersection over Union (IoU), also known as the Jaccard Index. IoU is a standard performance measure in segmentation. In contrast to conventional accuracy measures, this measure can overcome the problem of class imbalance. IoU represents the resemblance between the ground truth and the predicted segmentation. It calculates the ratio between the intersection and the union of the ground truth and the prediction. The intersection is the number of true positives and this is divided by the union; the sum of the true positives, the false negatives and the false positives [Garcia-Garcia et al., 2017]. IoU is computed for each class and then averaged, providing the mIoU. The equation for the calculation of the mIoU is provided in Equation 3.2. In this equation, k represents number of classes, i is the actual class of the pixel, j is the predicted class of the pixel, p_{ii} is the number of true positives, p_{ij} represents the number of false positives and p_{ji} is the number of false negatives.

$$mIoU = \frac{1}{k+1} \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - p_{ii}} \tag{3.2}$$

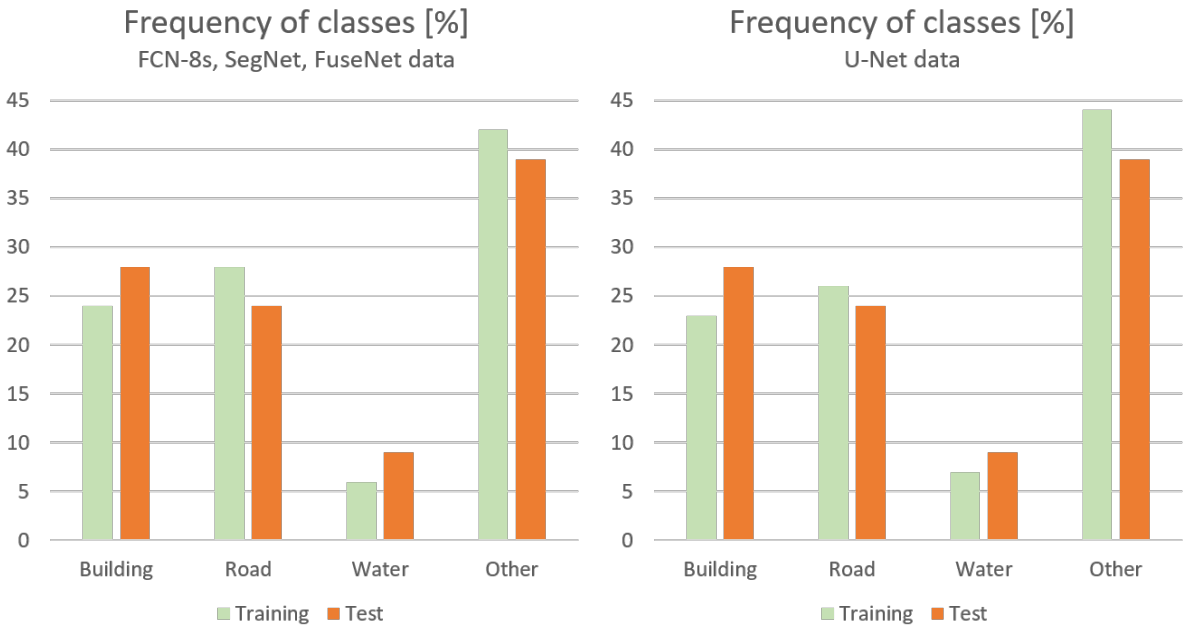
Next to the mIoU, the (average) F1 score is calculated to allow for comparison with researches that did not use the mIoU. The F1 score is first computed per class (Equation 3.3), and hereafter averaged. The F1 score uses precision and recall (Equation 3.4) to assess the accuracy. In these equations p_{ii} represents the number of true positives for the class i, P_i corresponds to the number of pixels assigned to class i by the prediction and C_i is the actual total number of pixels belonging to the class i.

$$F1_i = 2\frac{precision_i \times recall_i}{precision_i + recall_i} \tag{3.3}$$

$$precision = \frac{p_{ii}}{C_i}, recall = \frac{p_{ii}}{P_i} \tag{3.4}$$

3.3.3 Early stopping

In order to decide on when a training process should be terminated, early stopping is implemented. After every epoch, the model performance is evaluated using the validation data. Without updating the parameters, the validation data is fed to the network and performance measures are calculated for the total of the outputted predictions. If the calculated mIoU is higher than the highest mIoU calculated on



(a) (b)

Figure 3.5: The frequencies of the different classes occurring in the training area and in the test area. (a) For the FCN-8s, SegNet and FuseNet-SF5 training area. (b) For the U-Net training area.

the validation data so far, the current model with its learned parameters is stored. If no improvement of the mIoU is detected, the current parameters will not be stored and the training process continues. If no improvement is detected on the validation data for X epochs in a row, the training process terminates. Therefore, X is a hyperparameter that needs to be specified in advance. This approach ensures that when the training process has terminated, the parameters which resulted in the best performance on the validation data can be retrieved.

3.4 Test data and inference

In order to test the performance of the final, trained, models, independent data is needed. This is data that the algorithm has not seen before during training, and has not been used for hyperparameter tweaking. In order to generate this test data, another area in Haarlem is selected with the exact same dimensions and a similar frequency of the classes as the training area (Figure 3.5a). Due to memory limitations, it is not possible to perform inference on the whole training area at once. Therefore, the test area is subdivided into smaller parts. Cutting the test area into non-overlapping tiles and feeding these to a trained network can lead to inconsistencies in the segmentation at the border pixels of a tile. These inconsistencies negatively influences the accuracy of the prediction [Liu et al., 2017]. In order to solve this problem, it is decided to use overlap processing in the inference phase, which is inspired on the U-Net overlap-tile strategy of Ronneberger et al. [2015].

Firstly, the test area is cut into overlapping tiles of 512x512 pixels. The amount of overlap is, similar to Audebert et al. [2018], set to half the image (256 pixels), leading to 6,241 test images (79x79). Normal inference is executed on each of the constructed tiles and one predicted class label per pixel is stored. Hereafter, from each output prediction tile, the outer 128 pixels (256/2) are removed from each side. These cropped images are compared to the equally cut and cropped ground truth, to calculate the performance measures. Hereafter, the cropped prediction results are merged to form one Tagged Image File (TIF). In this way, it is aimed to eliminate the border issue.

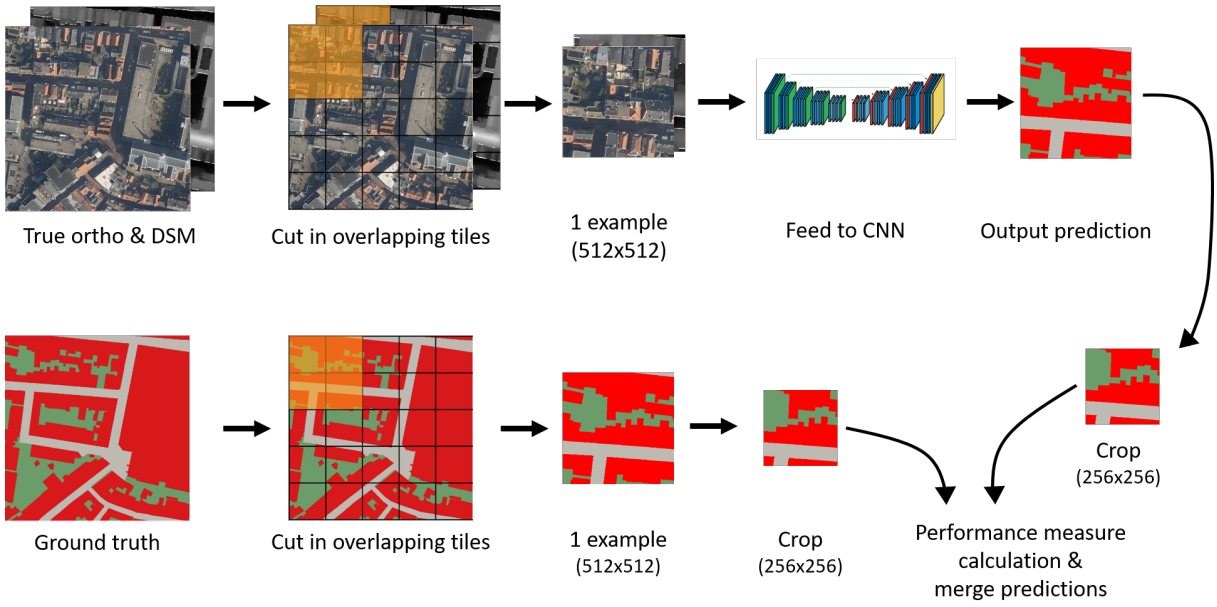


Figure 3.6: The overlap inference strategy.

The idea behind this approach is that for every pixel prediction present in the merged output, enough neighboring information was present to perform the prediction. In other words, it is assured that for all the predictions provided in the output, the complete context was available. An illustration of the overlap inference process is given in Figure 3.6. Comparing performance measures calculated on the test data to performance measures calculated on the validation data provides an indication on consistency of the quality of the model. Nevertheless, it should be noted that tile border pixels have not been removed when calculating performance measures on the validation data.

As mentioned before, the U-Net implementation used has specific requirements for the input data. Therefore, the dimensions of the test extent deviate slightly from the extent of the test data used for the other models. In order to still allow for comparison of all the models, it is aimed to keep the geographic location of the test data as similar as possible. The U-Net test extent is 22,688x22,688 pixels. Every image is 572x572 pixels and images have an overlap of 184 pixels, leading to 3,364 test images in total (58x58). The overlap is based on the amount of pixels that are cropped of the input dimensions by the algorithm during the inference. Consequently, merging the output predictions of U-Net will automatically lead to a seamless map.

3.5 Drawing conclusions

This section describes how conclusions are drawn after the models have been trained. To answer the research questions, both performance measure comparisons (Section 3.5.1) and visual assessments (Section 3.5.2) are required. In addition, the object-level performance (Section 3.5.3) will be assessed for the best performing model.

3.5.1 Performance measure comparison

Firstly, the general performances of the different architectures are compared. Hereafter, the segmentation results and the corresponding mIoU scores derived from models when height information is included, are compared to the results of the same architectures without the inclusion of height information. In addition, to further assess the value of the height information, the difference in segmentation results per class are examined and compared to the results without inclusion of height information.

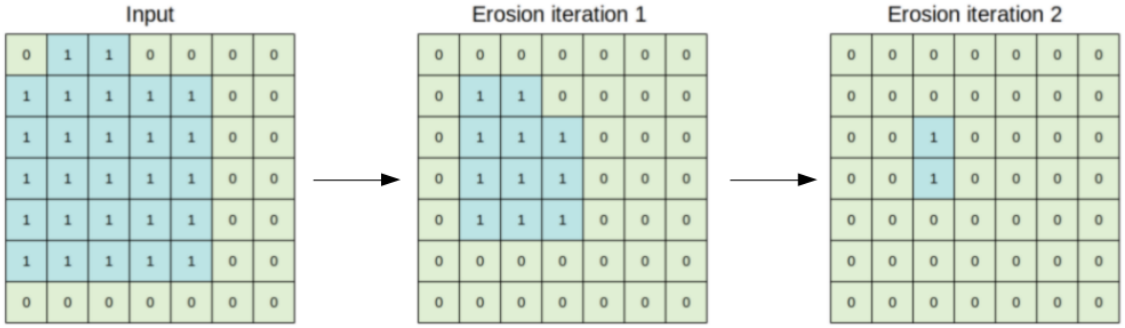


Figure 3.7: Two iterations of binary morphological erosion applied to a 7x7 input.

Hereafter, the best performing data stacking models (which included height information) are compared to FuseNet-SF5 results. Finally, the performances of the different height approaches are assessed.

3.5.2 Visual assessment of results

The mIoU and the IoU per class provide an important indication on the performance of a model and allow for easy comparison of performances of different models. Nevertheless, as the goal is to generate high quality maps, assessing visualizations of the output is essential. It requires visual assessment to detect what kind of errors are made and therefore what challenges are for a specific model. This can not be done through solely taking the mIoU into consideration. Therefore, complementary to merged prediction outputs, binary error maps are generated, indicating per pixel if the class label assigned to the pixel by the algorithm is correct (0) or incorrect (1) according to the ground truth. These maps allow for detection of clusters or areas in an image that are mis-classified. Error maps are also generated per class and erosion is applied to them using the SciPy Python library[3]. Erosion comprises the morphological operation of shrinking the clusters or shapes in an image, in this case in the error maps, by eliminating border pixels (Figure 3.7). The goal of this process is to emphasize large clusters corresponding to large errors and eliminate small errors corresponding to, for example, slightly deviating borders. Four iterations of erosion will be applied to erode the outer 40cm (4 pixels) of the error clusters. This value is selected, with a little slack, to compensate for the difference in resolution of the ground truth (20cm) and the aerial imagery (10cm).

3.5.3 Object-level performance

The IoU represents the performance of a class as a whole. However, it does not provide information on object-level performance. Originally, the ground truth represents complete objects, and not pixels. Consequently, it could be argued that, when assessing the performance of a model, it is interesting to see how well individual objects are detected. Therefore, for each object in the ground truth, the percentage of correctly classified pixels in the algorithm's prediction is calculated. Firstly, using the Zonal histogram tool in QGIS 3.12.1., counts are generated per ground truth object for each unique value present in the outputted prediction raster. Hereafter, the field calculator is used to divide the count of correctly classified pixels by the total number of pixels that covers the object's location. The calculated percentages are displayed in histogram plots per class. These histograms are generated using NumPy's Histogram function[4]. This method allows to receive an estimation on percentages of objects present in the ground truth that are detected by the algorithm and percentages of objects that the algorithm has missed. It should be noted that this method only allows for an estimation on the object-level true

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.binary_erosion.html
[4] https://numpy.org/doc/1.18/reference/generated/numpy.histogram.html

positives and false negatives. It does not include any information on false positives; the objects that the algorithm has detected but which are not present in the ground truth. It is believed that false positive information is required to assess the quality of a model as a whole. Therefore, these histograms are not taken into consideration for model comparison and selection, but are solely generated for the final, best performing model.

In addition, it is attempted to receive an indication on the amount of false positives on object-level. Using the raster calculator in QGIS 3.12.1, error maps are generated containing only pixel-level false positives. Hereafter, four iterations of erosion are applied to erode the outer 40cm (4 pixels) of the false positive clusters. The resulting raster map is vectorized using the Polygonize tool in QGIS 3.12.1. The generated polygons represent the false positives on an object-level.

4 Datasets, implementations and experiments

This chapter provides a description of the used datasets (Section 4.1) and an analysis of the suitability of the BGT to serve as ground truth. Hereafter, implementation details are provided on the methodology in Section 4.2, followed by the presentation of the experimental setup of this study in Section 4.3.

4.1 Datasets

4.1.1 Basisregistratie Grootschalige Topografie

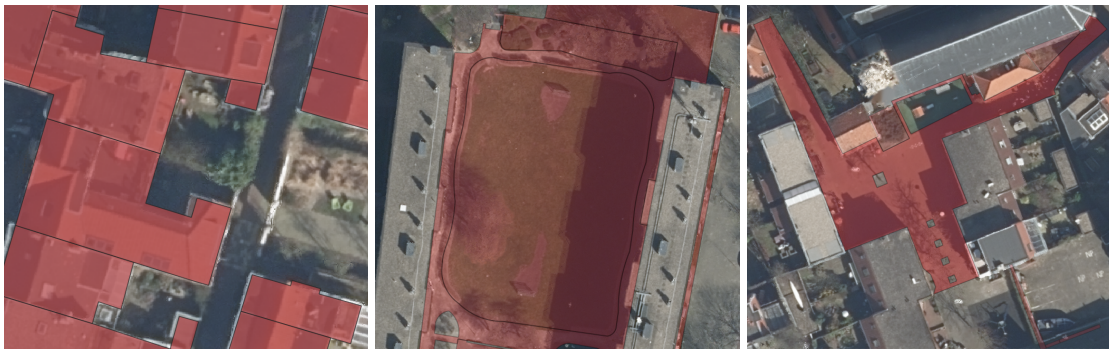
In this research work, the BGT dataset is used as a mask layer for the training examples. This dataset has an accuracy of 20 centimeters and provides detailed topographic information of the Netherlands. It can be accessed from the PDOK portal[1], which is the national open dataset portal for geo-information of the Dutch government. The BGT is regulated by law and freely available for any user. It is developed through a collaboration between municipalities, provinces, waterboards, the Ministry of Economic Affairs, the Ministry of Defence, Rijkswaterstaat and ProRail. Every so called 'bronhouder' is responsible for delivering a specific piece of the dataset. Rules are set on the minimal required quality that should be delivered, as an attempt to guarantee high quality topographic information [Kadaster, 2018].

To ensure that the quality of the dataset is sufficient for this study and to assess the suitability of the dataset for being a mask layer, the BGT was assessed manually before preparing the training data. The metadata of the BGT was examined and the data covering Haarlem was visually compared to corresponding aerial imagery. The following paragraphs summarize the most important findings.

The BGT covers many different classes, which suggests that this dataset has the potential to train a network to distinguish a broad amount of objects. In addition, the large size and extent of the dataset ensures a high number of instances per class, leading to a high amount of training data when combined with aerial imagery. Furthermore, the borders of the geometries of the objects in the BGT generally match with the borders of the objects visible in the aerial imagery. Nevertheless, in some rare occasions the boundaries of the polygons show some deviation from the true ortho imagery (Figure 4.1a). However, it is believed that this will not have a large impact on the quality of the results, as the amount of detected occurrences of this problem is small and the algorithms take surrounding pixels into consideration during the convolutional operations.

A limitation detected is the content of the classes `begroeid terreindeel` and `onbegroeid terreindeel` which indicate continuous vegetation and terrain without continuous vegetation respectively. It was discovered that in practice both these classes have polygons containing vegetation and no vegetation. In addition, the classification was incomplete and inconsistent. For example, patches containing grass and trees were classifies as `onbegroeid terreindeel` (Figure 4.1b) and patches completely covered by tarmac were classified as `begroeid terreindeel` (Figure 4.1c). It is believed that this inconsistency, which was also detected in class attributes defining the type of object, strongly limits the ability of networks to properly learn to distinguish vegetation from non-vegetation. Consequently, when no manual preselection through visual assessment is executed of the used training data, the BGT is considered as unsuitable for training a network to classify vegetation. Therefore, this class is not included in this study.

[1] `http://pdok.nl`



(a) (b) (c)

Figure 4.1: Errors are present in the BGT. (a) The boundaries of the BGT sometimes show a slight deviation from the boundaries visible in the true ortho imagery. (b) Field with grass and trees is wrongly classified in the BGT as `onbegroeid terreindeel` (terrain without continuous vegetation) with the attribute 'open verharding' (open pavement). (c) Patch with tarmac is wrongly classified in the BGT as `begroeid terreindeel` (continuous vegetation) with the attribute 'heesters' (shrubs).

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

Table 4.1: Mapping of BGT classes to the segmentation classes.

Overall it can be concluded that both the quantity and quality of the BGT are considered to be sufficient for the dataset to serve as a mask layer in this study, for the classes `building`, `road`, `water` and `other`.

As mentioned in Section 3.2, a cleaned version of the BGT of Haarlem was provided by READAR, in which only relevant classes are present, terminated objects were removed and no overlapping objects are present as only objects visible from above are preserved. In order to allow this cleaned BGT to serve as mask layer, some processing of this dataset was needed. First, using the field calculator in QGIS 3.12.1, the objects were reclassified into the classes `building`, `road`, `water` and `other` (Table 4.1). Hereafter, the shapefiles were merged to one layer. Next, the GDAL Rasterize tool was used to convert the vector data to a raster with pixels of 10x10 cm. This raster was then clipped to the training and test extents.

4.1.2 True ortho imagery and Digital Surface Model

The aerial imagery used as input for the neural networks is true ortho imagery generated by READAR. This is RGB aerial imagery with a 10 cm resolution and which is corrected for relief displacement. Relief displacement comprises the problem that due to deviating distances from the central perspective and vertical relief, too much information can be visible on one side of objects, while occlusions occur

on the other sides of objects in an image [Sheng et al., 2003; Lemmens, 2011]. The true ortho imagery is generated using READAR's dense matching software which is based on deep learning techniques [Zuurmond, Cor, 2018]. In order to create this imagery, the software uses the national high resolution stereo imagery captured during spring 2018, from Beeldmateriaal Nederland [Beeldmateriaal Nederland, 2020] and PSM-Net. PSM-Net is a pyramid stereo matching network proposed by Chang and Chen [2018] and consists of two modules; Spatial Pyramid Pooling (SPP) and 3D CNN. The SPP module is responsible for retrieving information from different granularities, whilst the 3D CNN learns to regularize the output of the SPP to generate disparity maps. These maps provide the horizontal shift in pixels for each pixel in the reference image.

In addition, interpolation techniques based on Inverse Distance Weighting (IDW) are used for estimating values where occlusion has occurred. Holes are interpolated using the lowest points in its surrounding. This methodology is used as the occlusion has occurred due to high objects located next to these holes. Therefore, pixels located in the holes are expected to be more similar to the surrounding low-lying pixels than to the surrounding pixels with high elevations. The resulting true ortho imagery contains an extra band providing information on whether or not the present pixel values were interpolated.

The point clouds obtained through the stereo matching represent the absolute height relative to NAP per pixel and are converted to a DSM. This DSM, fulfills the role of the fourth band, the Z-band, in the data stacking approaches in this study. In addition, this DSM is used in the generation of the alternative height approaches with which is experimented with FuseNet-SF5.

4.1.3 Digital Terrain Model

In order to generate the data corresponding to the relative height approaches described in Section 3.2.1, the height of the terrain is required. The AHN3 DTM provides this information. AHN is digital height data covering the Netherlands and is a collaboration between provinces, the central government and the water boards. In order to generate the DTM, a point cloud is generated using LiDAR, from which non-ground points are removed. The remaining points are resampled to generate a raster using Squared IDW. This data can be extracted from the PDOK portal[2]. For this study, the 0.5m resolution DTM is used. It should be noted that the data for the DTM is not acquired at the same date as the true ortho imagery. Nevertheless, it believed that this will not result in large errors due to the assumption that the terrain height will not alter significantly over a short amount of time.

Due to the removal of non-ground points, holes are present in the DTM. This is not a problem for the tile-level, relative height approach, where the median per tile is calculated. However, the pixel-level, relative height approach, which involves pixel level subtraction, does require terrain data for each pixel. Therefore, before using the DTM for the pixel-level subtraction approach, the holes are interpolated using IDW.

4.2 Implementations

The existing neural networks which will be tested are implemented in Python, using the open source machine learning framework PyTorch[3]. This package uses tensors which are similar to NumPy's ndarrays. These tensors can use the power of GPUs to allow for fast computations. The PyTorch-SemSeg repository on GitHub [Shah, 2017] aims at mirroring successful and popular semantic segmentation architectures in PyTorch. In this repository currently 11 different architectures are implemented, together with basic code for training the networks. Configuration files are used to specify hyperparameters and other settings in advance, before the execution of a training. This repository is used as a starting point for this study. It is selected as it is well documented, relatively easily modifiable to your own needs and models are direct implementations of the architectures described in the corresponding research papers.

[2] https://www.pdok.nl
[3] https://pytorch.org

The repository was set up in 2017, but it is still maintained and most of the model implementations are less than 1.5 years old. Even though the repository is considered to be a suitable starting point, adjustments had to be made and complementary code had to be developed to execute this study. The following paragraphs describe these adjustments, alongside with details on the implementations of the described methodology in Chapter 3.

4.2.1 Spatial data and performance measures

As the PyTorch-SemSeg repository has not been developed with spatial data in mind, support of a Coordinate Reference System (CRS) needed to be implemented in order to know the exact location of the segmented output. Using Rasterio[4], the CRS information of the input tile is copied to the output tile. Hereafter, if not yet present, the performance measures of interest, such as the F1 score, had to be implemented.

4.2.2 Data loader

A data loader needed to be generated to allow for the architectures to use this specific training data. In this data loader, input tiles are converted to numpy arrays and input imagery is normalized from [0-1], using the Min-Max Feature scaling algorithm (see Equation 3.1). The *min* was set to 0, and the *max* to 255. For the ground truth information, if a pixel contained no data, the pixel received the class label 0 (`other`). When height is included in the training, this information is added to the numpy array as an extra dimension. This allows all the training data to be stored in one numpy array instance per tile. Before being presented to the network, the numpy arrays are converted to multi-dimensional tensors. These tensors are the required input for neural networks programmed in PyTorch.

4.2.3 Support of FuseNet-SF5

As FuseNet-SF5 was not one of the 11 implemented architectures, the architecture needed to be incorporated in the used framework. In order to achieve this, a FuseNet implementation equal to the FuseNet-SF5 architecture of Hazirbas et al. [2017] is used, which was available on GitHub[5].

4.2.4 Early stopping

As mentioned in Section 3.3.3, early stopping is used during training. Early stopping is an alternative to fixing the number of iterations in advance. Beforehand, the user provides the number of consecutive epochs for which no improvement of the mIoU is detected, when compared to the best performing set of parameters so far. As the used framework did not support early stopping but used a pre-fixed number of iterations before termination, early stopping needed to be implemented. In addition, in order to improve the the training of the model parameters, after every epoch the training examples are reshuffled.

4.2.5 Extra band and pretrained weights

In order to ensure that the data stacking architectures (FCN-8s, SegNet and U-Net) allowed for support of an extra band next to RGB, the number of in-channels in the first convolutional layer needed to be changed from three to four. The rest of the architecture could be kept equal. As mentioned before, using pretrained models can save training time. However, a problem occurs when working with RGB-Z data, as these pretrained networks generally lack the support of an extra band next to RGB [Kampffmeyer et al., 2016]. As the architectures of FCN-8s and SegNet have encoders that are based on the VGG-16

[4] https://rasterio.readthedocs.io/en/latest/
[5] https://github.com/zanilzanzan/FuseNet_PyTorch/blob/master/models/fusenet_model.py

classification network [Simonyan and Zisserman, 2014], weights and biases of a trained VGG-16 model were used to initialize these three networks. These were available through the PyTorch Torchvision package[6]. With FCN-8s and SegNet, when trained with height information, the weights and biases corresponding to the fourth band were randomly initialized through the default procedure of PyTorch. In this procedure, the initialization function depends on the type of layer[7]. Experiments were executed which compared model performances when pretrained and when not pretrained. Due to a different encoder structure, the VGG-16 weights could not be used for U-Net. Therefore U-Net is only trained with randomly initialized weights.

As the encoders of FuseNet-SF5 are also based on the VGG-16 architecture, pretrained weights can be used for this architecture. Since FuseNet-SF5 has a complete encoder dedicated to the height information, rather than one extra band, it requires a slightly different weight initialization approach. It is decided to also copy the complete pretrained weights of VGG-16 to the height encoder and to take the average of the first layer of weights over the channel dimension as the height encoder requires a one-dimensional input (only height) rather than three (RGB).

4.2.6 Loss function

In this study, there is experimented with two different loss functions. The standard cross-entropy loss function (Equation 4.1), and the weighted cross-entropy loss function (Equation 4.2). These equations describe the calculation of the loss for an individual pixel. $x[class]$ represents the target value for that pixel (based on the correct class in the ground truth) and $x[j]$ indicates the prediction of that pixel for class j. During training, the loss function is used to calculate the error between the prediction of the network and the ground truth (see Section 2.1.1).

$$loss(x, class) = -\log\left(\frac{\exp(x\,[class])}{\sum_j \exp(x\,[j])}\right) = -x\,[class] + \log\left(\sum_j \exp(x\,[j])\right) \tag{4.1}$$

$$loss(x, class) = weight\,[class]\left(-x\,[class] + \log\left(\sum_j \exp(x\,[j])\right)\right) \tag{4.2}$$

In order to calculate the loss for a batch, the output of the neural network is provided as input to the loss calculations. The process works as follows; for each pixel four values are provided, one for each class. The target class of each pixel, which is the label for that pixel in the ground truth, is known. The softmax of the input is calculated, resulting in a probability distribution which sums up to one. These probabilities indicate the chance of that pixel corresponding to a particular class. Hereafter, the loss is calculated by taking the $-log$ of the probability corresponding to the class present in the ground truth. For each batch, the average loss is calculated. This loss is then back propagated and weights are updated with the help of the optimizer, with the goal to reduce the error.

As can be seen in Figure 3.5, class imbalance is present in both the training as the test area. The classes do not cover the same amount of pixels. Consequently, the frequencies of classes can be strongly inhomogeneous in the segmented output. This can potentially affect the performance of the cross-entropy loss function [Liu et al., 2017]. As an attempt to reduce this (potential) negative influence and to improve the mIoU of a model, experiments are executed in which class weights are added, leading to the weighted cross-entropy loss function [Audebert et al., 2018]. These weights are calculated using the inverse class frequencies (1/frequency of class in the whole area). For example, if 20 percent of the pixels in the training area have the label `building` in the ground truth, than the class weight of building is 5 (1/0.2). Consequently, classes with a low frequency receive a higher weight resulting in an increased influence in the calculation of the loss. The calculated weights used are provided in Table 4.2. As the training area of U-Net slightly differs from the other models, the weights are calculated separately.

[6] https://pytorch.org/docs/stable/torchvision/models.html
[7] For more information see: https://pytorch.org/docs/master/nn.init.html#nn-init-doc

	Building	Road	Water	Other
FCN-8s, SegNet, FuseNet-SF5	4.221	3.624	15.412	2.368
U-Net	4.321	3.766	14.825	2.296

Table 4.2: The class weights used for the weighted cross-entropy loss function.

4.2.7 Optimizer

In this study, experiments are executed with two different optimizers, namely SGD and Adam. SGD is a traditional optimizer in which the weights are adjusted based on an estimation of the actual gradient of the loss; the average gradient of training examples [LeCun et al., 2015]. Adam is considered as an improvement of the SGD optimizer. Whilst SGD is widely used in different types of machine learning applications, the Adam optimizer has been specifically designed for deep neural networks [Kingma and Ba, 2014]. The biggest difference between the optimizers is the approach towards the learning rate; the proportion that the parameters (weights) are updated. Traditional SGD maintains one learning rate for all the updates of the weights in a network, and this rate does not change over time. On the contrary, Adam maintains individual learning rates per parameter in a network. These are separately adapted throughout the learning procedure. For both optimizers the implementations available in PyTorch were used[8]. There is experimented with different (initial) learning rates, namely 1e-3, 1e-4 and 1e-5.

4.2.8 Server and Docker

As mentioned before, the computational power of an external server is used in this study. Models are trained using a single GPU (rtx2080ti 12 GB). In order to allow for training and testing of the networks on this server, Docker[9] containers are used. A Docker container is a software unit in which all the code and dependencies of an application can be stored. Consequently, the application can be easily used in different computing environments.

4.2.9 TensorBoard

In order to follow the progress of a training and to detect configuration errors while the training is being executed, support of TensorBoard[10] is implemented. TensorBoard is a visualization toolkit developed for machine learning experimentation. It is used for the generation of real-time plots of performance measures, such as the mIoU and the per class IoU (Figure 4.2). In addition, after every epoch the predicted semantic segmentation of one validation tile is provided to visually keep track of the performance of the model over time. In order to see the visualizations generated through TensorBoard, the local host of the server was connected to the local host of the used laptop.

4.3 Experiments

In order to allow for a valid comparison between different architectures, RGB and RGB-Z input, stacking and fusion approaches and use of different height types, several experiments needed to be executed. A schematic overview of the performed tests done to detect the most promising hyperparameters, are presented in Table 4.3. In order to select best performing hyperparameters per model, relative performance (performance of one setting or value of the hyperparameter compared to performance using a different setting or value) is more important than absolute performance. In other words, for being able to distinguish the best setting of a hyperparameter, it is initially only relevant to assess the difference

[8] For more information, see: https://pytorch.org/docs/stable/optim.html
[9] https://www.docker.com/
[10] https://www.tensorflow.org/tensorboard

Figure 4.2: A screenshot of the TensorBoard interface. This toolkit allows for keeping track of the training process by plotting performance measures.

		RGB			*RGB-Z (data stacking)*			*RGB-Z (data fusion)*
Hyperparameter	**Options**	**FCN-8s**	**SegNet**	**U-Net**	**FCN-8s**	**SegNet**	**U-Net**	**FuseNet-SF5**
Weight initialization	Pretrained / random	x	x		x	x		x
(Initial) learning rate	1e-3 / 1e-4 / 1e-5	x	x	x				x
Optimizer	SGD / Adam	x	x	x				x
Loss function	CP / WCP	x	x	x				x
# epochs no improvement	10 / 20 / 50	x	x	x				x
Horizontal flipping	Yes/no	x	x	x				x
Height type	AH / SHT / SHW / RHP / RHT				AH & SHT	AH & SHT	AH & SHT	x

Table 4.3: Schematic overview of the experiments conducted for hyperparameter selection. After these experiments, models are trained in which the best performing settings are combined. An x indicates that experiments are executed with all the options of the hyperparameter. CP = Cross-entropy, WCP = Weighted cross-entropy, AH = Absolute height, SHT = Rescaled height [0-1] (tile-level), SHW = Rescaled height [0-1] (whole area), RHP = Relative height (pixel-level), RHT = Relative height (tile-level)

in results when that parameter is altered. After deciding on which hyperparameter settings are most promising, experiments are executed in which these settings are combined. Consequently, some hyperparameter settings needed to be selected as 'baseline settings'. These settings include; (initial) learning rate of 1e-4, epochs of no improvement of 20, the Adam optimizer, the cross-entropy loss function, no data augmentation and when possible, initialization with pretrained weights. When height information was included, absolute height was used as default. The selection of these settings is based on a trade off between the impact on computation time and the expected influence of that parameter on absolute performance. In addition, the batch size is set to 5 for all FCN-8s, SegNet and U-Net experiments, and 4 for FuseNet-SF5 experiments. This batch size is selected to fit the memory of the used server.

With the RGB-Z data stacking architectures there is not experimented with all the hyperparameters, as most of the promising hyperparameter settings will be discovered during the RGB experiments. The RGB-Z data stacking architectures are trained with absolute height and scaled absolute height (tile-level). In addition, experiments are performed with pretrained weights (if available) and randomly initialized weights. Randomly initialized weights are also tested with the RGB-Z experiments to allow for comparison of randomly initialized weights with the performance of the developed pretrained weights approach for the height band, as described in Section 4.2.5).

With the FuseNet-SF5 architecture experiments are performed with all the hyperparameter settings and all the different height approaches. The decision to not test all the height approaches for the other three architectures is based on the fact that FuseNet-SF5 was introduced to this study at a later phase (see Figure A.1). The use of FuseNet-SF5 was a response to the demand for an alternative way to use

the height information, which became clear through training results of the first three architectures (see Chapter 5).

5 Results and analysis

In this chapter the results of the performed experiments are presented and analyzed. Firstly, Section 5.1 provides the hyperparameter settings that correspond to the best performing models per architecture. Hereafter, in Section 5.2, the results of the models trained on only RGB imagery, without height information, are discussed. Next, in Section 5.3, the added value of height information to segmentation quality is examined for the overall performance of models (Section 5.3.1), and for the performance of the individual classes (Section 5.3.2). Next, in Section 5.4, the performance of the data stacking approach is compared to the data fusion approach, followed by an analysis of the results of the experiments with different height approaches in Section 5.5. Hereafter, in Section 5.6 the object-level performance of the most successful model will be assessed. Next, in Section 5.7 it is demonstrated that 'errors' in the outputted predictions are not always actual errors, but can be the result of missing objects in the BGT. Finally, Section 5.8 attempts to compare the best performing models of this study to results gained in related work.

An overview of all the calculated performance measures, on both the validation and the test data, of all the executed experiments, is given in Appendix B.2.

5.1 Hyperparameters

The hyperparameter settings corresponding to the best performing models per architecture are provided in Table 5.1. These combinations of parameters have resulted in the highest mIoU scores on the *validation* data. In this section short remarks will be made on the selected parameters and their performance on the validation data. The results on the *test* data of the models corresponding to these settings will be used for analysis in the remainder of this study, except when stated otherwise.

For all the architectures, when available, pretrained weights lead to a higher performance than randomly initialized weights (i.e. +5% for FCN-s (RGB), +6% for SegNet (RGB) and +2% for FuseNet-SF5). Furthermore, inclusion of horizontal flipping resulted in a slight increase in performance for each of the architectures (on average +1%). In addition, with all the models the Adam optimizer with a initial learning rate of 1e-4 resulted in higher mIoU than when compared to SGD and other initial learning rates. It should be noted that a successful training was only achieved with SGD for FCN-8s, as other models retrieved extremely low mIoU scores. Nevertheless, the training with SGD and FCN-8s took almost 43

Hyperparameter	**FCN-8s**	**SegNet**	**U-Net**	**FuseNet-SF5**
Weight initialization	Pretrained	Pretrained	Random	Pretrained
(Initial) learning rate	1e-4	1e-4	1e-4	1e-4
Optimizer	Adam	Adam	Adam	Adam
Loss function	CP	CP	CP	CP
# epochs no improvement	50	50	50	50
Horizontal flipping	Yes	Yes	Yes	Yes
Height type (only with RGB-Z)	SHT	SHT	SHT	RHP

Table 5.1: Hyperparameter settings corresponding to the best performing models per architecture. The 'height type' is only relevant for the experiments in which height was included. For an overview of the different hyperparameter options with which is experimented, see Table 4.3. CP = Cross-entropy, AH = Absolute height, SHT = Rescaled height [0-1] (tile level), RHP = Relative height (pixel-level).

Model	mIoU	Average F1
FCN-8s	0.8121	0.8958
SegNet	**0.8219**	**0.9015**
U-Net	0.7637	0.8647

Table 5.2: Performance measures of the models on the test data when trained on RGB only.

hours, which was approximately 10 times longer than the same experiment with Adam. In addition, the mIoU was 7 percent lower for this SGD experiment, when compared to the same experiment with the Adam optimizer. Therefore, it was decided to select the Adam optimizer and to not consider SGD in the remainder of the experiments.

Moreover, the use of the weighted cross-entropy loss function, did not lead to an increase in model performance. For all four architectures, the normal cross-entropy loss function outperformed, or equally performed when compared to the weighted loss function. U-Net even showed a decrease in mIoU on the validation data of more than 2 percent when the weighted loss function was used.

Furthermore, when considering the termination of the training procedures, the number of epochs of no improvement corresponding to the best performing models was 50. Training with this setting did take approximately 2 to 3.5 times longer for these models than when 20 epochs of no improvement was used.

Finally, for the data stacking models, when height was included, scaled absolute height (tile-level) showed a slightly better mIoU performance than normal absolute height (on average almost +1%). For the data fusion approach, the absolute height slightly outperformed scaled height (∼+0.5%). In addition, when considering the height experiments executed with FuseNet-SF5, pixel-level, relative height showed to outperform the other height approaches. A more extensive analysis of the performance of the different height approaches is presented in Section 5.5.

5.2 RGB baseline comparison

In order to allow for an assessment of the added value of height information, baseline models were trained in which solely RGB information was used. The mIoU and average F1 scores of the best performing models on the *test* data are presented in Table 5.2. In addition, the mIoUs together with the outputted segmentations allowed for a comparison of the suitability of the different architectures for semantic segmentation of the true ortho imagery.

The mIoUs indicate that SegNet provides the best segmentation performance. However, the achieved performance measures are very similar to FCN-8s. U-Net scores approximately 6 percent lower on mIoU, when compared to SegNet. U-Net did require only three hours to finish the training procedure, whilst FCN-8s and SegNet took approximately 4 times longer.

Visual assessment of the segmented output led to similar findings. On first sight, the segmented output of FCN-8s and SegNet appears of similar quality, whilst U-Net generally delivered more messy and less homogeneous predictions (Figure 5.1). In areas with straight streets, straight house blocks and little vegetation, all three models perform well (Figure 5.2). Especially FCN-8s and SegNet show homogeneously segmented patches with little to no distortion in residential areas. In general, for all three models, the biggest challenge for the segmentation quality seems to be imposed by shade. Shady areas belonging to the classes `road` or `other` are often misclassified as `water`. A possible explanation could be that the dark color of shady areas confuses the algorithms, resulting in a misinterpretation for `water`. Generally, SegNet outperforms the other models at shady areas and especially U-Net shows to have difficulty with this issue (Figure 5.3).

As mentioned in Section 2.5.2, one of the goals of the development of the SegNet architecture was the improvement of boundary predictions, when compared to FCN performance. Even though SegNet

Figure 5.1: The test area with the predictions of models on RGB imagery. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) Ground truth (c) Prediction FCN-8s (d) Prediction SegNet (e) Prediction U-Net.

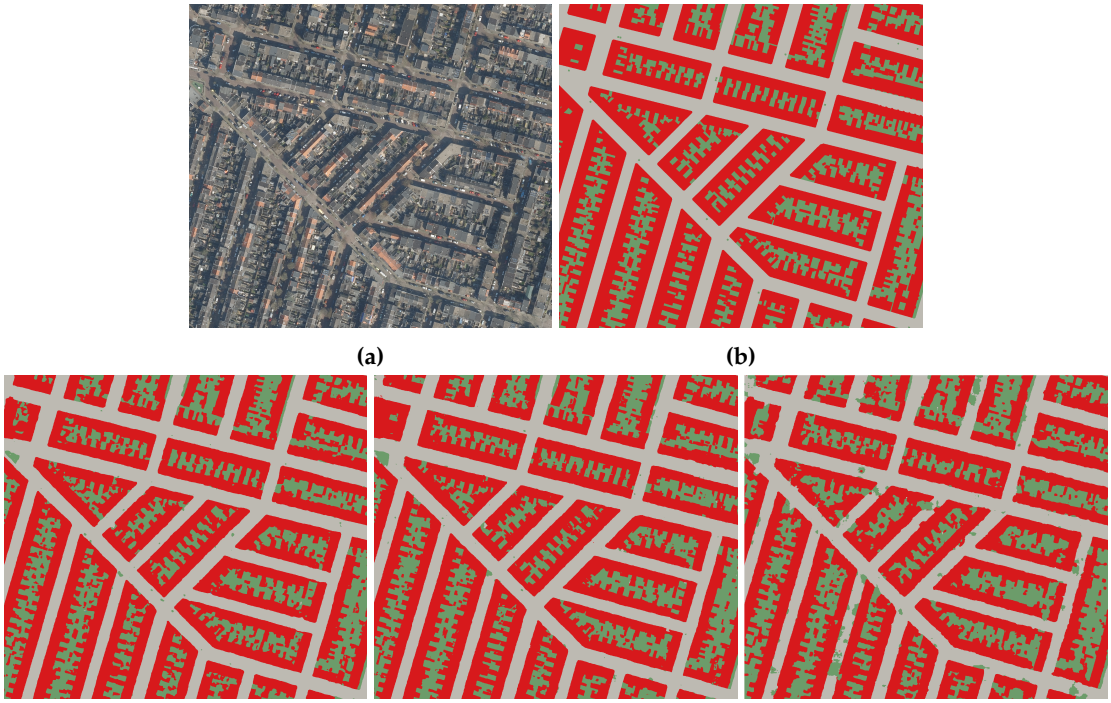
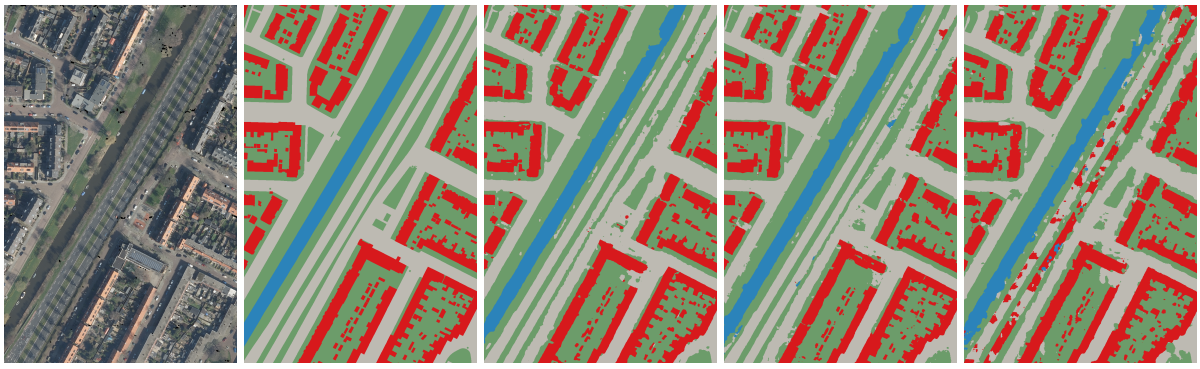


Figure 5.2: All models show to be successful in segmenting areas with straight streets, straight house blocks and little vegetation. Red = building, gray = road, green = other. (a) RGB true ortho (b) Ground truth (c) Prediction FCN-8s (d) Prediction SegNet (e) Prediction U-Net.

Figure 5.3: Shade performs a challenge on the segmentation quality of the algorithms using RGB data. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) Ground truth (c) Prediction FCN-8s (d) Prediction SegNet (e) Prediction U-Net.



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

Figure 5.4: FCN-8s and SegNet show a similar quality of boundary representation on the RGB data, whilst U-Net's boundaries are of a poorer quality. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) Ground truth (c) Prediction FCN-8s (d) Prediction SegNet (e) Prediction U-Net.

does provide a slightly higher mIoU, visual assessment of the output predictions did not indicate a better boundary representation (Figure 5.4).

Due to the self-learning nature and the tens of millions of trainable parameters present in the individual architectures, directly pointing out why an individual architecture outperforms another is challenging. The results on the RGB data indicate that especially the strategy of SegNet leads to a high performance. Therefore, it could be argued that the transferring of max-pooling indices from the encoder to the decoder for upsampling and using trainable decoder filters, is a successful approach for semantically segmenting aerial imagery. However, it should be noted that using pretrained weights instead of randomly initialized weights resulted in a increase in mIoU on the validation data of approximately 5 and 6 percent for FCN-8s and SegNet respectively. Therefore, a large part of the lower performance of U-Net can presumably be attributed to the absence of pretrained weights. Nevertheless, a comparison between FCN-8s and SegNet with randomly initialized weights to U-Net also indicated a slightly lower performance of U-Net, namely; approximately 1 and 3 percent lower when compared to FCN-8s and SegNet respectively. These outputs suggest that even though U-Net performance is lower than FCN-8s and SegNet, the difference in performance is presumably smaller than the current mIoU suggests.

5.3 Data stacking: RGB versus RGB-Z

In this section, the results of the RGB-Z stacking experiments, in which height is included, are presented and compared to the results gained with the models discussed in the previous section where only RGB information was used. First, the overall performance of the different models will be compared in Section 5.3.1. Hereafter, in Section 5.3.2 the performances per class will be assessed.

5.3.1 Overall performance

In Table 5.3 the mIoU and average F1 scores show that for every architecture the addition of height information led to a (slight) improvement. However, as the added value to the mIoU for both FCN-8s and SegNet is solely approximately half a percent, this overall improvementis considered as small. U-Net does show an increase in the mIoU of just over two percent when height information is included, indicating that U-Net benefits more from the added information than the other architectures. Nevertheless, even with this increase, U-Net still shows a 4 percent lower mIoU performance when compared to SegNet. Therefore, the inclusion of height information did not alter the hierarchy of performance of the architectures.

Model	Input	mIoU	Average F1
FCN-8s	RGB	0.8121	0.8958
FCN-8s	RGB-Z	0.8177	0.8990
SegNet	RGB	0.8219	0.9015
SegNet	RGB-Z	**0.8257**	**0.9039**
U-Net	RGB	0.7637	0.8647
U-Net	RGB-Z	0.7851	0.8786

Table 5.3: Data stacking: performance measures on the test data of models trained on RGB and RGB-Z.

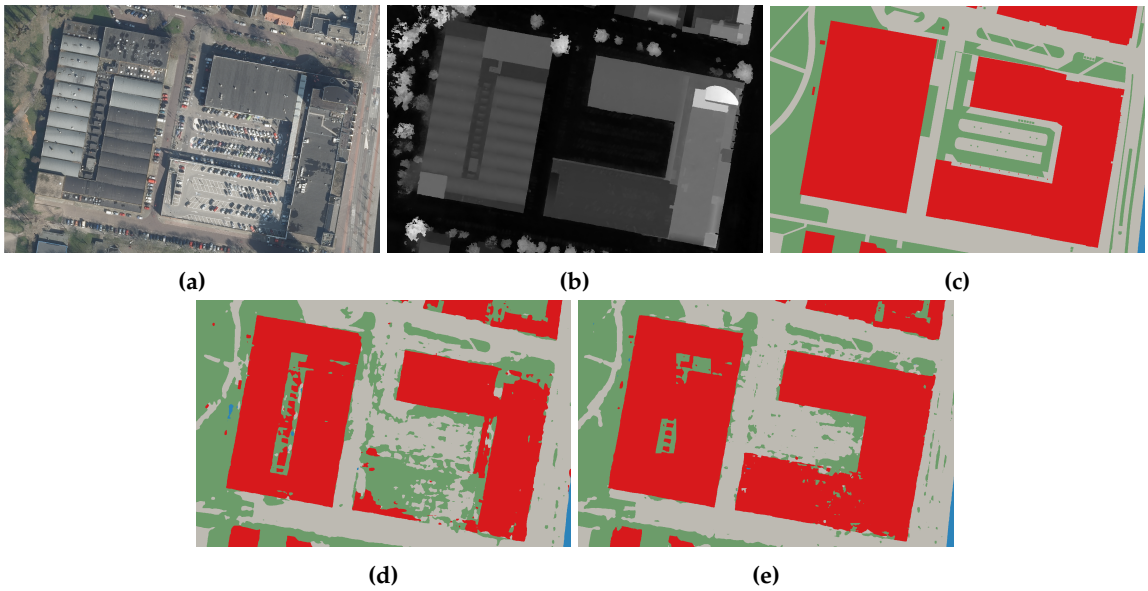
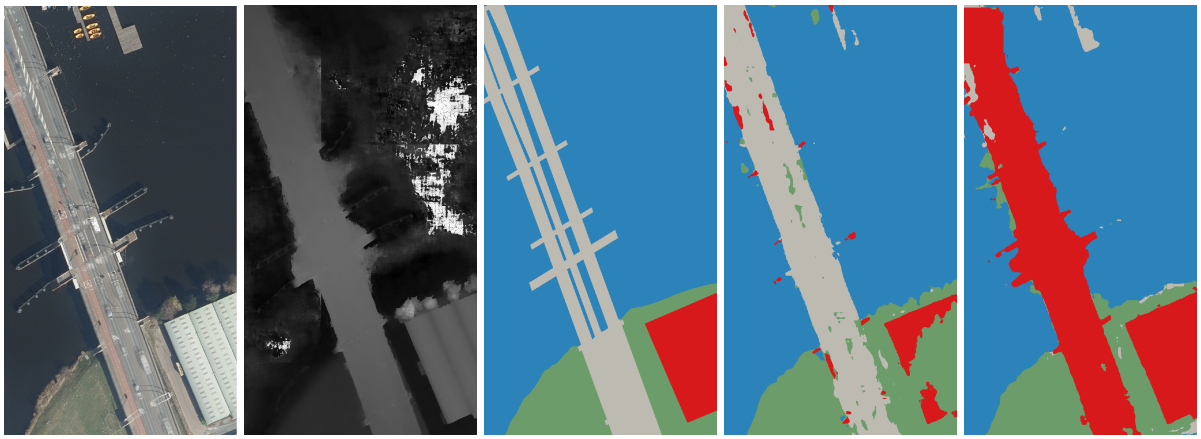


Figure 5.5: Height information shows to have the potential to aid the prediction in challenging cases. A parking lot is present on top of a building. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) DSM (c) Ground truth (d) Prediction FCN-8s (RGB) (e) Prediction FCN-8s (RGB-Z)

Visual assessment of the outputted maps indicates that the prediction quality of U-Net indeed improves when height information is added. Predictions are more smooth and homogeneous and less messy. In addition, boundary representation is less fuzzy. On first sight, the quality of the segmented outputs of FCN-8s and SegNet do not show any improvement when height information is added. However, interesting differences can be observed when zooming in. For example, in some challenging environments the addition of height information does indeed show to aid the prediction. A clear example is demonstrated in Figure 5.5, in which a parking lot in present on top op a building. When no height information is included, the pixels corresponding to this parking lot on top of the building are misclassified as `road` and `other`. With height information included, most of the pixels receive the correct label of `building`.

The result presented in Section 5.2 showed that the presence of shade negatively influences the performance of the algorithms. As height information could show for shady areas that they actually do not differ in structure from areas next to it where no shade is present, it was hoped that the addition of the DSM would improve segmentation results in shady areas. However, for none of the architecture this seemed to be the case (see Figure A.4). A possible explanation could be that the stacking approach does not allow the algorithms to learn a different type of features from the height information, other than the features that are also relevant for the RGB data. This could be a problem as the RGB data encodes information on color and texture, rather than on structure.



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

Figure 5.6: Addition of height information leads to misclassification of `road` pixels corresponding to a large bridge. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) DSM (c) Ground truth (d) Prediction FCN-8s (RGB) (e) Prediction FCN-8s (RGB-Z)

Even though throughout most of the test area added height information seems to have either no influence or a positive influence, one occasion is detected in which the added height information seemed to confuse the algorithms. This occurs at a large bridge that passes over water (Figure 5.6). Before addition of the height information, most of the pixels are correctly classified as `road`. However, when height information is added, the algorithms misinterprets the bridge for a building, especially FCN-8s has this issue. A possible explanation could be that the algorithms have learned that one of the properties of `road` pixels is that they belong to the lowest pixels of the surrounding, as roads usually have the same elevation as the terrain. When a large bridge crosses over water, the road is suddenly elevated when related to its surrounding, which might cause confusion. Luckily, the classification of other, smaller and lower bridges going over water in the test area did not show the same issue.

5.3.2 Class performances

In order to assess the performance per class, the IoUs corresponding to each class are given in Table 5.4. From these results in can be noted that in general, with both RGB and RGB-Z input, the algorithms predictions for the classes `building` and `water` are of a higher quality than for the classes `road` and `other`. Furthermore, the results indicate that for FCN-8s and SegNet the addition of height information positively influences the class `building` with approximately 2.5 percent and 1 percent respectively. The other classes do not seem to be influenced by the height information, with the exception of `road`. When height information is used, the segmentation quality of `road` for FCN-8s is lowered by 1 percent. U-Net clearly benefits the most from the addition of height information. An increase in performance is detected of almost 6 percent for `building`, 1.6 percent for `road` and almost 3 percent for `other`. However, the class `water` does not benefit from the height information.

Even though the IoUs provide a good first indication, it should be noted that the IoUs presented correspond to the models that showed the best overall performance; the best mIoU. Consequently, if for example one set of parameters leads to a lower than average performance on the class `water`, but shows to be extremely successful in segmenting the class `building`, it is still possible that this model is selected as final model, as the mIoU outperforms others. However, this model then does not reflect the potential performance on the class `water`. Therefore, it is interesting to assess the IoUs achieved on the validation data throughout the training process (Figure 5.7). From this information it can be detected if, on average, height information has an influence on the segmentation performance of a class.

The plots validate that for FCN-8s, SegNet and U-Net the performance of the class `building` improves when height information is added (Figure 5.7 a, b & c). For U-Net, the same can be concluded for the

Model	Input	Building	Road	Water	Other
FCN-8s	RGB	0.8305	0.7822	0.8661	0.7698
FCN-8s	RGB-Z	**0.8567**	0.7714	0.8700	0.7725
		+0.0262	-0.0108	+0.0039	+0.0027
SegNet	RGB	0.8426	0.7810	**0.8907**	0.7735
SegNet	RGB-Z	0.8538	**0.7827**	0.8841	**0.7822**
		+0.0112	+0.0017	-0.0066	+0.0087
U-Net	RGB	0.7814	0.6974	0.8535	0.7225
U-Net	RGB-Z	0.8384	0.7134	0.8365	0.7521
		+0.0570	+0.0160	-0.0170	+0.0296

Table 5.4: Class IoU performance of RGB and RGB-Z data stacking approaches on the test data. The added value of height information is presented in green (positive influence) and red (negative influence).

classes `road` and `other` (Figure 5.7 f & i). For FCN-8s and for SegNet, the class `other` generally shows an increase in performance as the RGB-Z line (red) is mostly located above the RGB line (blue) (Figure 5.7 j & k). For FCN-8s and SegNet `road` does not show any influence of height information (Figure 5.7 d & e). Slightly contrary to the IoU, `water` for FCN-8s seems to be lightly negatively influenced when height information is added whilst for the class `water` for SegNet no impact seems to be present (Figure 5.7 g & h).

In Table 5.5 and Table 5.6 the confusion matrices of SegNet trained on RGB and SegNet trained on RGB-Z are given. In these tables the values are provided in percentages of the total pixels belonging to that class in the ground truth. The original confusion matrices with the absolute values, are provided in Table B.1 and Table B.2. From the matrices representing the absolute values, it can be noticed that for both models the biggest sources of errors are buildings predicted as `other`, roads predicted as `other` and pixels which actually belong to `other` predicted as `road`. The addition of height information allowed the algorithm to correctly classify over one percent more `building` pixels, mostly by reducing the amount of actual `building` pixels being wrongly predicted as `other`. However, the height information did not provide a solution for `road` pixels being classified as `other` and `other` pixels being classified as `road`. In addition, the added height resulted in over 0.7 percent less `water` pixels being correctly classified, and over 1 percent more `water` pixels now being misclassified for `building`.

		Prediction			
		Building	**Road**	**Water**	**Other**
Actual	**Building**	**90.55**	1.04	0.09	8.32
	Road	1.19	**89.49**	0.14	9.19
	Water	1.98	0.70	**92.31**	5.01
	Other	4.15	8.01	0.67	**87.16**

Table 5.5: Confusion matrix of SegNet (RGB) on the test data. The values are given in percentages.

		Prediction			
		Building	**Road**	**Water**	**Other**
Actual	**Building**	**91.77**	0.84	0.15	7.24
	Road	1.19	**89.51**	0.18	9.11
	Water	3.13	0.57	**91.58**	4.71
	Other	3.91	8.06	0.59	**87.44**

Table 5.6: Confusion matrix of SegNet (RGB-Z) on the test data. The values are given in percentages.

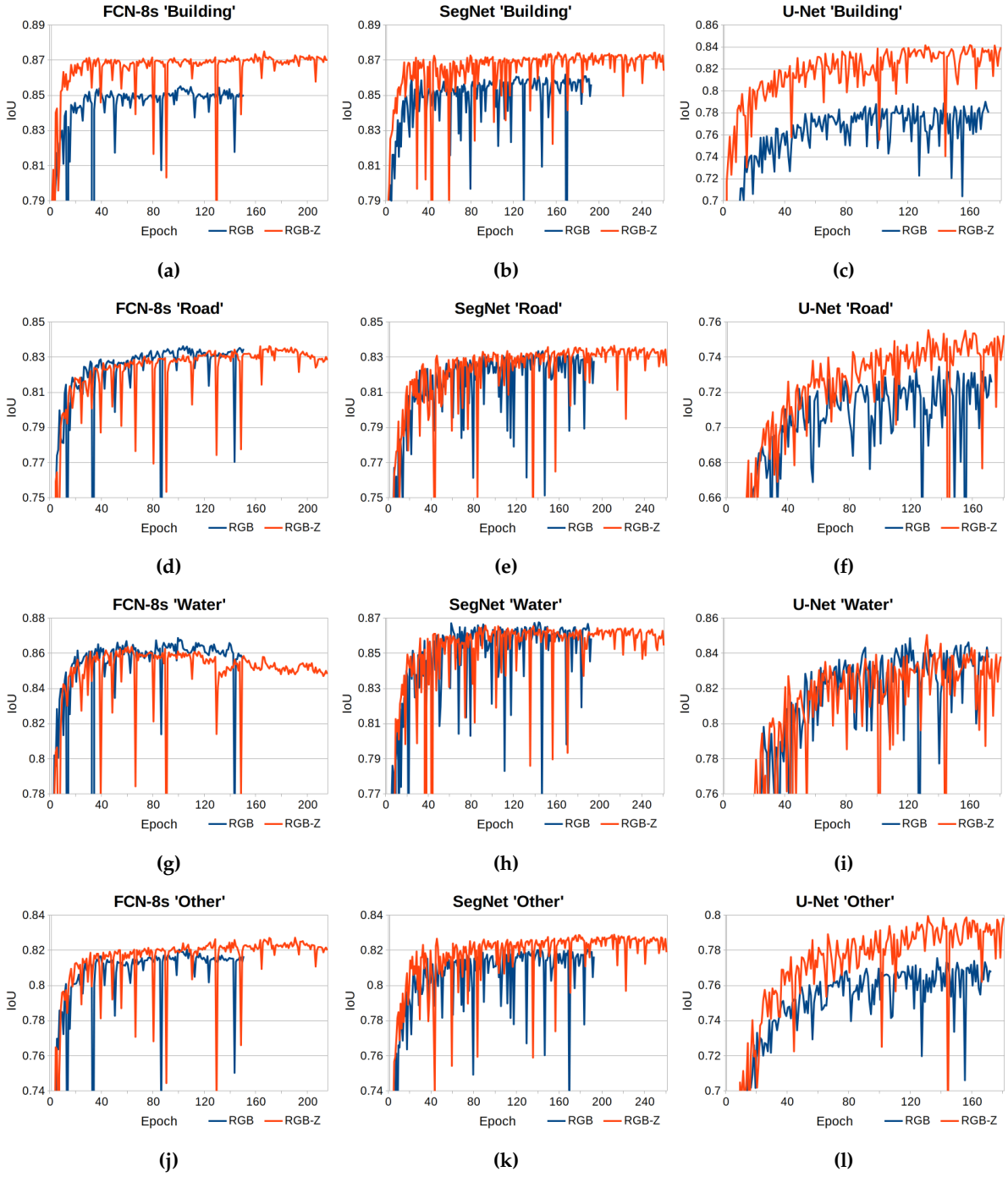


Figure 5.7: The performance on the validation data achieved per class during training, for; FCN-8s (a, d, g, j), SegNet (b, e, h, k) and U-Net (c, f, i, l).

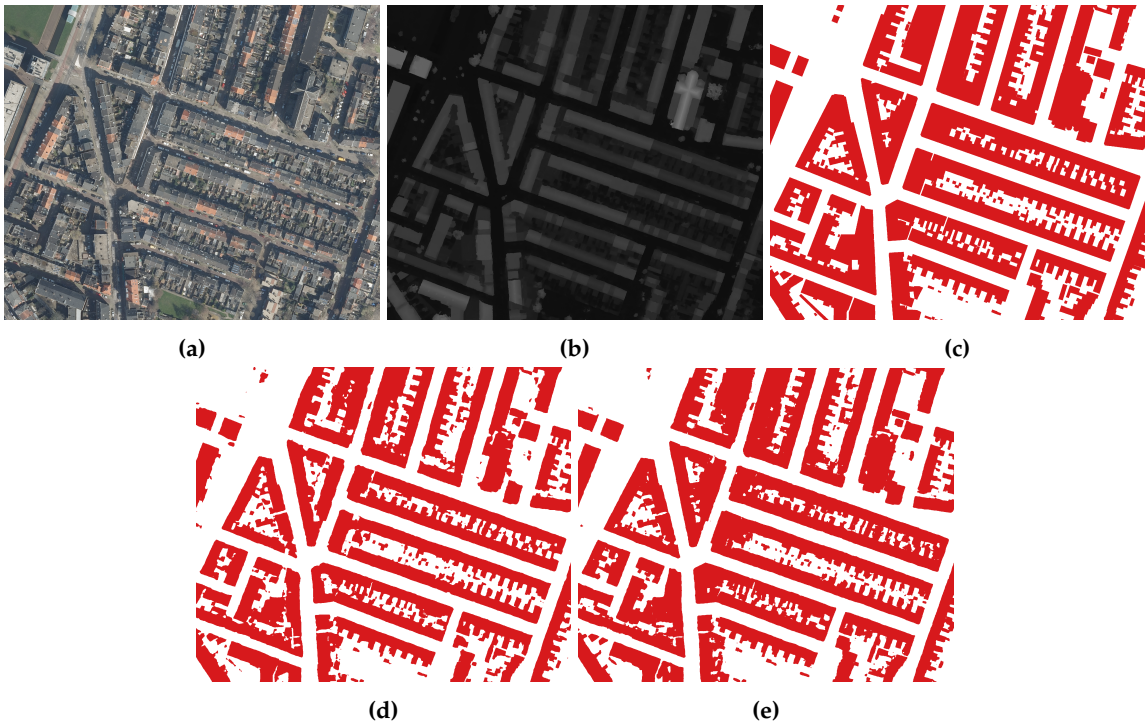


Figure 5.8: Addition of height information increases the details of building predictions. (a) RGB true ortho (b) DSM (c) Ground truth (only buildings) (d) Prediction FCN-8s (RGB) (e) Prediction FCN-8s (RGB-Z)

Visual assessment of the segmented outputs of the different models indicated that the increase in performance for the class `building` is mostly visible in the form of the filling up of incorrect holes and the improvement of details at parts of buildings that are not directly attached to the road (usually backside of houses, house extensions and sheds) (Figure 5.8). This finding is in line with the information retrieved from the confusion matrix that when height information is added less `building` pixels are misclassified as `other`.

It is expected that in general the class `building` performs better than the classes `road` and `other` as buildings usually have crisp, straight boundaries that are clearly visible from the air. These boundaries could potentially easily be detected by edge detection filters. The improvement of segmentation quality of the class `building` when height information is included, could be explained by the fact that buildings are always elevated from the ground. This elevation in combination with usually clearly visible crisp boundaries is believed to boost the segmentation performance.

On the other hand, roads in aerial imagery contain a large amount of noise. Cars, trees, shade from buildings and other obstacles present next to, or on top of, roads, limit the visibility of the roads and their boundaries from the air. In addition, the ground truth information of the class `road` appears to be of a relatively lower quality than the other classes. Especially small roads are not always consistently mapped in the BGT (see also Section 5.7 & Figure 5.20). This has a negative influence on the training procedure and results in an erroneous reduction of the calculated performance measure. The noise of the objects present on top of, and next to, roads are also present in the height information. This, in combination with the fact that roads often follow the height of the terrain, is believed to make height information less distinctive for `road` than for `building`. This could be a possible explanation for why the addition of height information is currently not contributory for the segmentation of the class `road`. However, roads have the property that they are usually flat and of equal height (and width) in the cross sectional view. Therefore, it is believed that when obstacles are less present, height information in general still contains the potential to improve the segmentation of this class.

Generally, segmentation of `water` is believed to be of a relatively high quality as the consistent dark

color of water is discriminant in the RGB imagery and, comparable to `building`, water usually has crisp borders. At large water bodies, the DSM is of low quality due to the challenge that the highly resembling water pixels impose on the dense matching technique used to generate the data. As a result of the flat nature of water, it is believed that at water bodies where the DSM does provide correct information, height information can be valuable for the segmentation. However, at areas with water where the DSM is of low quality, it is believed that it will not aid the prediction, and might even confuse the algorithm leading to a lower performance.

The class `other` is a challenging class in general as it contains everything that does not belong to one of the other three classes. This includes for example high vegetation but also bare ground, shrubs and construction sites. As the spectral properties differ strongly of these different types of areas, it is assumed that this explains the relatively lower performance of the class `other`. Figure 5.7 j, k and l, showed that addition of height information did provide some added value to the segmentation of `other`. This is somewhat contradicting to findings of Couprie et al. [2013], discussed in Section 2.3, that for classes with a high variability of the depth values, using solely RGB leads to better results than when depth information is included.

5.3.3 Inclusion of height information trough data stacking

The comparison of models trained on RGB with models using a RGB-Z stacking approach led to a number of findings. The addition of height information through data stacking resulted in;

- an increase in mIoU of over 2 percent for U-Net but only a marginal improvement for FCN-8s and SegNet;
- an improvement of the segmentation quality of `building` for all the models and a potentially slight improvement of the class `other`;
- addition of valuable information at complex segmentation task (i.e. parking lot on top of building);
- no improvement or even introduction of confusion for the class `water` and no real influence on the class `road` (with exception of the positive influence on U-Net).
- no improvement of segmentation of areas containing shade

Overall, these findings indicate that height information does have the potential to add value to semantic segmentation of aerial imagery. However, this added value currently differs per class. Solely providing height information for specific classes goes against the self-learning nature of CNN algorithms and is therefore not considered as an option. In addition, it is argued that the stacking approach does not allow to decode relevant information from the height data related to structure, which could be useful in segmenting areas containing shade. The question therefore arises if height information can be exploited to its full potential with height stacking approaches. This question resulted in the search for an alternative approach, which led to the experimentation with data fusion.

5.4 Stacking versus fusion

In this section a comparison will be made with the best performing data stacking model using RGB-Z; SegNet, and FuseNet-SF5 which includes height information through data fusion. For this comparison, only the experiments of FuseNet-SF5 with absolute and rescaled height (tile-level) were considered, as these were the only height approaches with which experiments were executed with the stacking models. From the results presented in Table 5.7 it can be noted that FuseNet-SF5 outperforms SegNet by slightly over 1 percent on mIoU. FuseNet-SF5 retrieved a slightly better performance using absolute height (mIoU: 0.8381) when compared to rescaled height (mIoU: 0.8326). Therefore, the model trained on absolute height is used for the comparison with SegNet. Nevertheless, the rescaled height approach also outperformed the stacking models.

A clear increase in performance is present for the classes `building` and `water` and a marginal increase is present for the class `other`. It should be noted that SegNet trained on RGB-Z did not provide the highest IoU for the classes `building` and `water`. Nevertheless, FuseNet-SF5 also outperformed the highest performances of all the stacking models on these classes. However, FuseNet-SF5 did not outperform the data stacking models for the class `road`. For this class FuseNet-SF5 scored approximately half a percent lower.

Model	Building	Road	Water	Other	mIoU
SegNet (RGB-Z)	0.8538	**0.7827**	0.8841	0.7822	0.8257
FuseNet-SF5	**0.8723**	0.7767	**0.9143**	**0.7890**	**0.8381**
	+0.0185	-0.0060	+0.0302	+0.0068	+0.0124

Table 5.7: IoU performance of the SegNet RGB-Z data stacking approach versus the FuseNet-SF5 RGB-Z data fusion approach on the test data. The added value of the fusion approach when related to the stacking approach is presented in green (positive) and red (negative).

		Prediction			
		Building	**Road**	**Water**	**Other**
Actual	**Building**	**93.10**	0.92	0.04	5.94
	Road	1.18	**88.94**	0.07	9.81
	Water	1.34	0.82	**93.61**	4.23
	Other	3.78	8.02	0.47	**87.73**

Table 5.8: Confusion matrix of FuseNet-SF5 on the test data, using absolute height. Values are given in percentages.

The confusion matrix with percentages corresponding to the prediction of FuseNet-SF5 with absolute height on the test area is presented in Table 5.8. In Table B.3 the confusion matrix with absolute values is provided. By comparing these matrices to the confusion matrices of SegNet (RGB) (Table 5.5) and SegNet (RGB-Z) (Table 5.6), it can be noticed that FuseNet-SF5 is able to extend the benefit that SegNet (RGB-Z) showed to have over SegNet (RGB). FuseNet-SF5 was able to reduce the misclassification of `building` pixels for `other` and to increase the amount of correctly classified `building` pixels by both approximately 2.5 percent, when compared to SegNet (RGB). Furthermore, in Section 5.3.2 it was mentioned that the addition of height to SegNet resulted in a reduction of correctly classified `water` pixels, and introduced extra errors in which `water` pixels were wrongly predicted as `building`. FuseNet-SF5 shows to not have this problem and is able to classify approximately 1.5 percent more `water` pixels correctly when compared to SegNet (RGB). This is mostly achieved by reducing the misclassification of `water` pixels for `building` and `other`. However, FuseNet-SF5 was not able to reduce the problem of `road` and `water` pixels being misclassified for each other and actually ensured that even more `road` pixels are misclassified for `other`.

A disadvantage of FuseNet-SF5 when compared to the other models is the training time. When 50 epochs of no improvement on the validation data was used, it took approximately 28 hours to train the model, which is 10 hours longer than SegNet. However, a model only needs to learn its parameters once, whereafter it can be used without limits.

Visual assessment of the output predictions did not show a remarkable difference between the segmentation quality of SegNet (RGB-Z) and FuseNet-SF5. However, when zooming in, it could be noted that FuseNet-SF5 provides a higher segmentation quality in shady areas when compared to SegNet (Figure 5.9). However, shady areas sometimes still showed to be a challenge for FuseNet-SF5.

Overall, the results of the comparison of FuseNet-SF5 to SegNet indicate that an increase in overall semantic segmentation performance on RGB-Z aerial imagery can be achieved by, instead of providing the height information as an extra band (data stacking), using a separate encoder for the height data and by fusing the retrieved features into the feature maps of the RGB imagery. In addition, even though shade is still a challenge for FuseNet-SF5, the fusion approach showed to outperform the other models

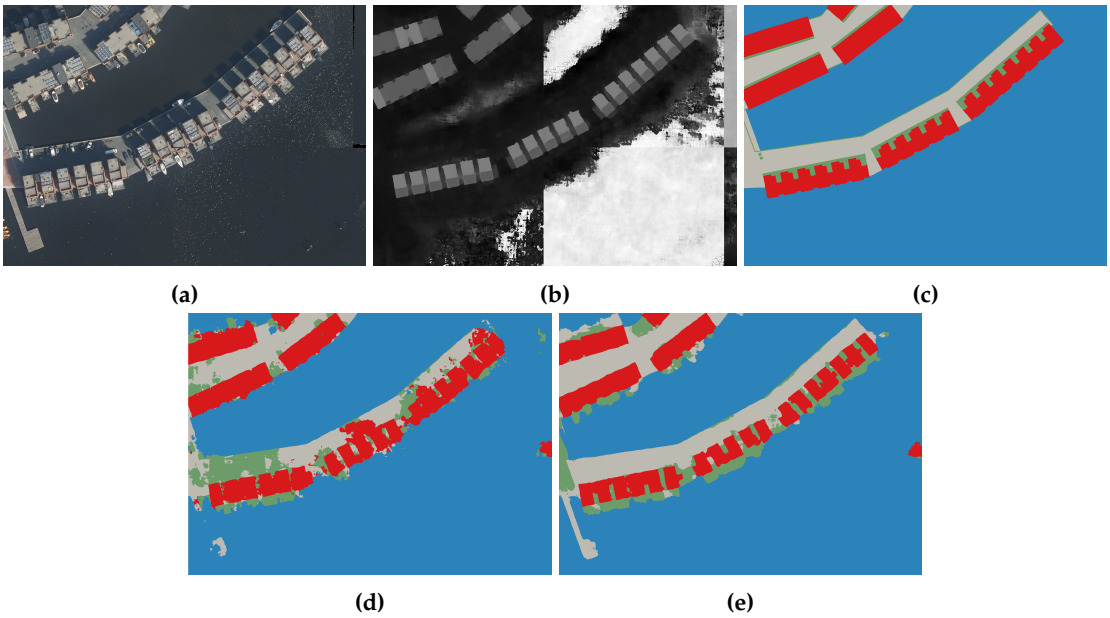


Figure 5.9: FuseNet-SF5 shows to outperform SegNet at areas containing shade. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) DSM (c) Ground truth (d) Prediction SegNet (RGB-Z) (e) Prediciton FuseNet-SF5

in areas were shade is present. This suggests that different types of features are learned from the height information when two separate encoders are used. Therefore, the potential of height information is believed to be exploited to a higher extent than when data stacking is used. FuseNet-SF5 improved the segmentation for `building` and `water`, but did not solve the problem present throughout all models that `road` pixels are misinterpreted for `other`, and the other way around.

5.5 Height approaches

In this study experiments were executed with several different approaches on representing the height information. Experimentation with all these approaches was done with FuseNet-SF5, as this architecture showed to outperform the other architectures in the previous experiments. For each of these experiments an initial learning rate of 1.0e-4, the Adam optimizer, 50 epochs of no improvement on the validation data, horizontal flipping and the cross-entropy loss function were used. These settings were selected as they showed the best performance for FuseNet-SF5 in the previous experiments. The results are presented in Table 5.9. It can be noted that overall performances are very similar. When considering the mIoU, it can be concluded that using pixel-level, relative height led to the highest semantic segmentation performance of 0.8427. This approach scored 1 percent higher than the least well performing approach; using height rescaled per tile. When assessing the performance of the algorithms on the validation data, the outperforming of the pixel-level, relative height approach was more outstanding than when considering the performances on the test data (Table 5.10).

The absolute and the rescaled height approaches contain the same information, but are presented to the algorithm in a different way. During visual assessment of the outputted prediction maps it was noticed that height rescaled at a tile-level resulted in some squared 'holes' at the center of buildings. A possible explanation for this phenomenon is that when the input tile is completely covered by a building and rescaling is applied at tile-level, no ground pixels are present in the tile that indicate that these building pixels are elevated from its surrounding. Therefore, the rescaling results in a loss of information and small differences due to roof structures are now wrongly emphasized during relative operations performed by convolutional filters. This potentially leads to misclassification of the `building` pixels and noisy predictions. Based on these findings, it was decided to also train a model using height

Height type	Building	Road	Water	Other	mIoU
Absolute	0.8723	0.7767	0.9143	0.7890	0.8381
Rescaled [0-1] (tile-level)	0.8671	0.7750	0.9023	0.7860	0.8326
Rescaled [0-1] (whole area)	0.8708	0.7846	**0.9152**	0.7897	0.8401
Relative (pixel-level)	0.8744	**0.7865**	0.9131	**0.7966**	**0.8427**
Relative (tile-level)	**0.8792**	0.7785	0.9070	0.7891	0.8384

Table 5.9: IoU performance on the *test* data of FuseNet-SF5 using different height inputs.

Height type	mIoU
Absolute	0.8573
Rescaled [0-1] (tile-level)	0.8543
Rescaled [0-1] (whole area)	0.8599
Relative (pixel-level)	**0.8729**
Relative (tile-level)	0.8574

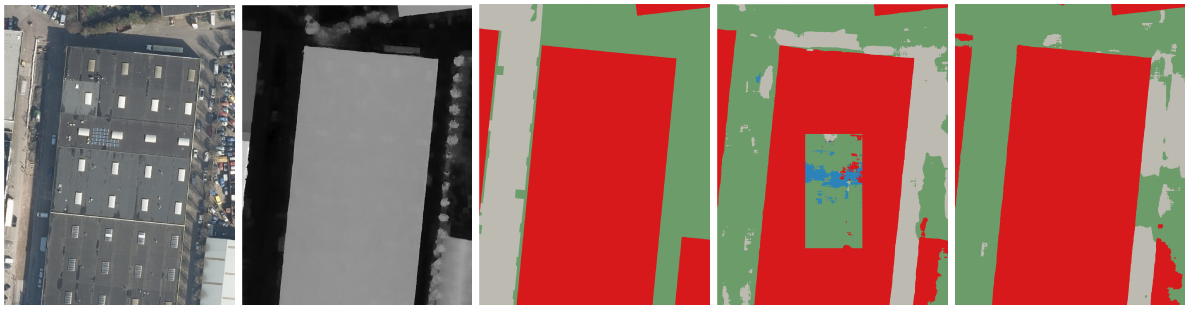
Table 5.10: mIoU performance on the *validation* data of FuseNet-SF5 using different height inputs.

rescaled by using the minimum and maximum value of the whole training area. This approach showed to strongly reduce this issue (Figure 5.10). It lead to a slight increase in IoU for all the classes and an increase of 0.75 percent for the mIoU.

When comparing the rescaled height (whole area) approach with absolute height, no noteworthy difference in either mIoU or map quality was detected. A possible explanation is sought in the property of FuseNet-SF5 that it uses a separate encoder for learning from the height information. Consequently, the importance of the values of RGB and height information being in the same distribution range diminishes.

Through assessment of the outputted predictions, it was observed that in general the relative height approaches provided higher quality segmentation maps than the approaches using absolute or rescaled absolute height. Large buildings at rural and industrial areas, which often contained holes or messy predictions with absolute height, showed to be more smooth and homogeneously segmented when relative height was included (Figure 5.11). The same phenomenon was observed for long and thick road segments. Predictions of both approaches on residential areas showed to be of similar (high) quality.

Visual comparison of the performance of pixel-level, relative height and tile-level, relative height indicated that even though the predictions clearly differed from each other, no conclusions could be



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

Figure 5.10: Rescaling the height information per tile results in errors at the center of large buildings. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) DSM (c) Ground truth (d) Prediction FuseNet-SF5 with rescaled height (tile-level) (e) Prediction FuseNet-SF5 with rescaled height (whole area)

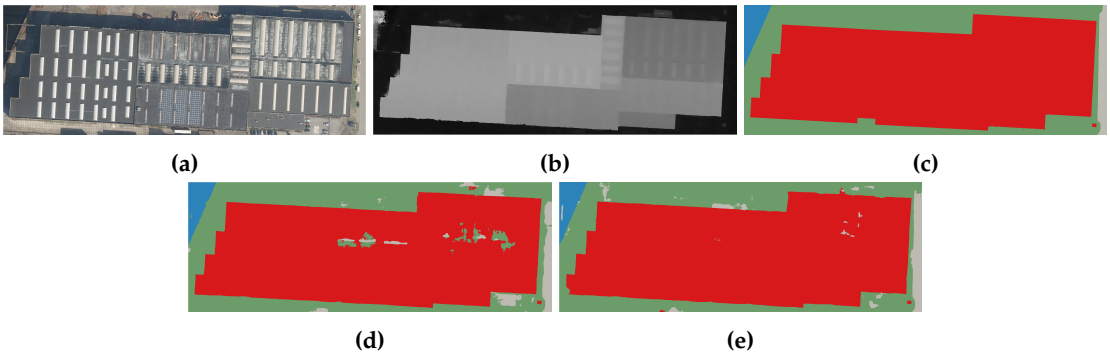


Figure 5.11: Using relative height information allows for more homogeneous and filled up prediction of large buildings. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) DSM (c) Ground truth (d) Prediction FuseNet-SF5 with rescaled height (whole area) (e) Prediction FuseNet-SF5 with relative height (tile-level)

drawn on which approach performed better. Per case it differed which model was the most successful. However, the mIoU did indicate an almost half a percent better performance of the pixel-level, relative height approach on the test data when compared to the tile-level, relative height approach.

Nevertheless, is should be noted that pixel-level, relative height information comes with a risk. For example, imagine a house being located on the side of a hill. When the terrain height is subtracted per pixel from the absolute height to generate the relative height, the resulting height values will represent the roof of the building in a strangely skewed way. The same problem occurs with a road passing through a hilly or mountainous landscape. Consequently, distinctive properties of specific objects, such as the non-skewness of roads in the cross-sectional view, will be lost. As a result, the ability of the algorithm to distinguish specific objects can be reduced. It is believed that due to the flat nature of Haarlem this potential disadvantage of the pixel-level, relative height approach is not reflected in the results. Therefore, for this study, FuseNet-SF5 trained with pixel-level, relative height is selected as the most suitable model for semantic segmentation of aerial RGB-Z imagery of the city of Haarlem. However, when applied to data corresponding to a different geographical location, with similar tile dimensions, but with more variation in terrain height, it is advised to focus on the tile-level relative height approach.

The semantic segmentation of the test area of the most successful model in this study; FuseNet-SF5 trained with pixel-level, relative height information is provided in Figure A.5.

5.6 Object-level detection

As mIoU and IoU does not provide information on the object-level performance of the model, this section presents the capability of FusNet-SF5 using pixel-level, relative height to detect individual objects in the test area. In Section 5.6.1 the ability of the model to detect the objects in the ground truth is assessed per class. Section 5.6.2 describes the results of the attempt to capture the object-level false positives of the model's prediction.

5.6.1 Ground truth detection

In Figure 5.12 histograms are provided representing object-level performance of FuseNet-SF5 using pixel-level, relative height on the test data. Additional figures that aid in understanding the intuition of the method are presented in Appendix A.5. In total, the ground truth contains 19,796 `building` objects, 6,176 `road` objects, 75 `water` objects and 6,769 `other` objects. For each class it can be noted that the last bin is the largest. This bin indicates the percentage of objects for which 90 to 100 percent of the pixels

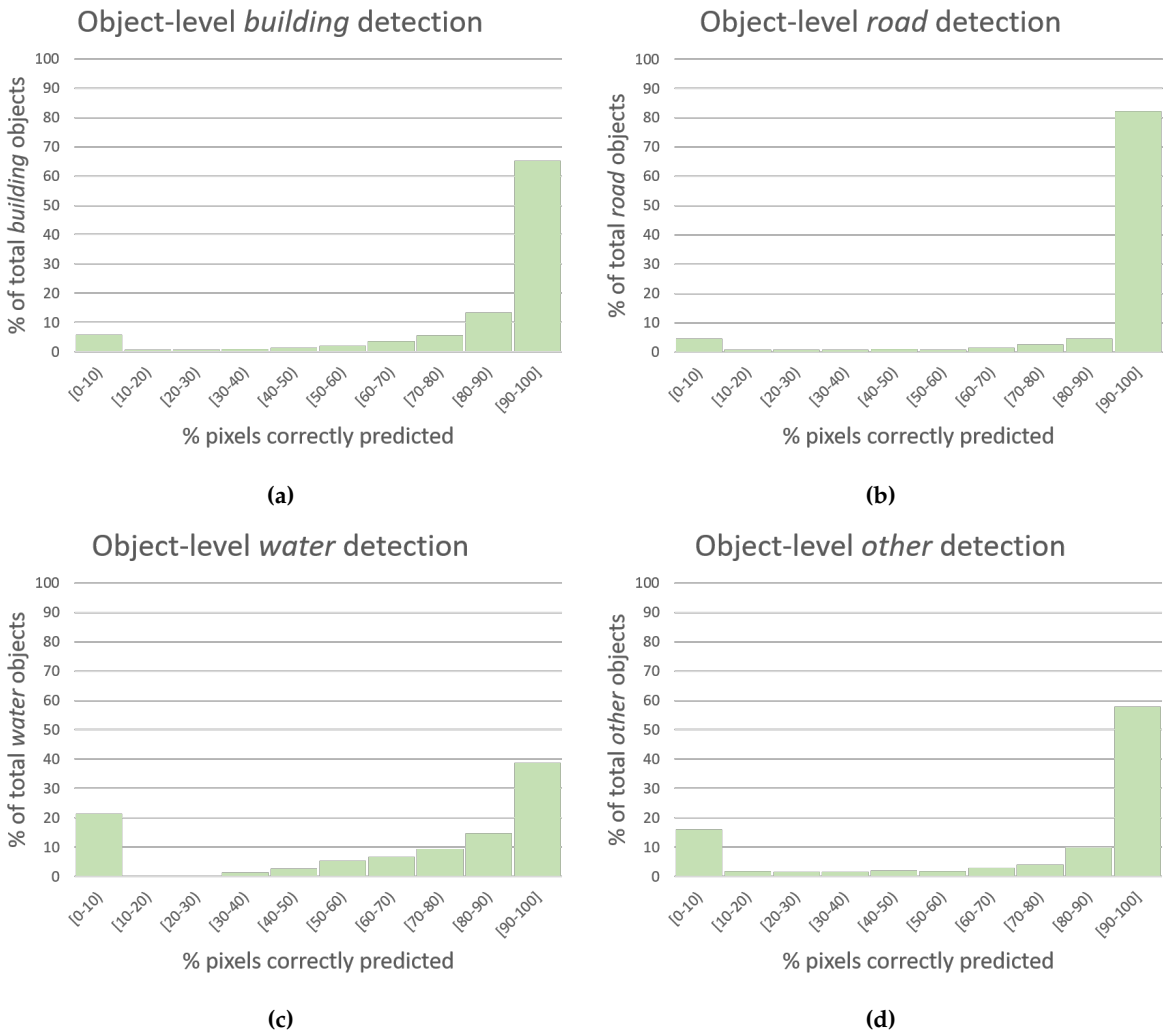


Figure 5.12: Histograms for the object-level performance of FuseNet-SF5 using pixel-level, relative height information. The y-axes represent the percentage of total objects in the ground truth of that class in the test area. The x-axes show the percentage of pixels corresponding to the ground truth object in the prediction that are correctly classified by the algorithm. a) Class `building` b) Class `road` c) Class `water` d) Class `other`

corresponding to the object are correctly classified by the algorithm. For the classes `building`, `road` and `other`, the majority of ground truth objects belong to this bin (65%, 82% and 58% respectively). For the class `water` only 39% of the objects in the ground truth are completely (90+%) detected by the algorithm. Overall, the algorithm was able to correctly classify over 90% of the pixels of 67% of all the objects present in the ground truth.

Furthermore, for each class a local peak is present at the first bin. This bin shows the object-level false negatives; objects present in the ground truth that are missed by the algorithm. When visually assessing these false negatives for the class `building`, two types of cases could be distinguished. An estimated half of these cases occur as a consequence of the limited visibility of the object from the air, due to the presence of trees (Figure 5.13a). The largest part of the other half of the missed objects are not observed in the aerial imagery or DSM, and are therefore considered as errors in the ground truth (Figure 5.13b). Finally, a small part of these missed objects are considered to be actual errors of the algorithm as a `building` object is visible in the imagery, but not detected. Similar discoveries were made for the missed objects of the class `road`. In addition, as noted before, shade present on top of roads resulted in the lack of detection of (parts of) the objects (Figure 5.14). The false negatives of the class

(a) **(b)**

Figure 5.13: Typical examples of `building` objects in the ground truth for which 0 pixels have been 'correctly' predicted by the algorithm. Red indicates the prediction of `building` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `building` objects in the ground truth. The numbers indicate the percentage of correctly classified pixels by the algorithm for that object. a) The presence of trees blocks the visibility of the object. b) The object is an error in the ground truth; it is not present in reality.

(a) **(b)**

Figure 5.14: Typical examples of `road` objects in the ground truth that are missed by the algorithm. Purple indicates the prediction of `road` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `road` objects in the ground truth. The numbers indicate the percentage of correctly classified pixels by the algorithm for that object. a) The presence of trees blocks the visibility of the objects. b) The objects are errors in the ground truth; it is not present in reality.

`water` are mostly observed at thin water bodies; usually ditches. In addition, the visibility of ditches from the air is frequently blocked due to the presence of trees. Consequently, only parts of the object are detected at these cases (Figure 5.15). Unfortunately, this is a limitation of using remotely sensed imagery and is therefore an issue that can not be solved completely. For the class `other`, objects missed by the algorithm are often; 1) small objects that are not (clearly) distinctive in the aerial imagery, 2) thin segments misinterpreted for `road` and 3) objects that are errors in the ground truth (Figure 5.16).

Interestingly, these histograms suggest that the class `road` is the most successful and the class `water` the least. This is contradicting to the calculated IoU scores for this model that indicated the exact opposite. This phenomenon can be explained by the fact that this method, in contrast to the IoU, does not provide any information on false positives. These are objects that are detected by the algorithm but which are not present in the ground truth. When examining the confusion matrix corresponding to FuseNet-SF5 using pixel-level, relative height (Table B.4), it can be noted that at pixel-level, the class `road` contains over 8 percent more false positives than when compared to the class `water`. Therefore, even though the highest percentage of objects have been detected for the class `road`, a large amount of pixels have received the label `road` whilst a different label was present in the ground truth. In addition, this relatively low performance of `water` for object-level detection, when compared to its IoU score, can also be explained by the fact that most of the `water` objects missed by the algorithm correspond to small and thin water bodies. These objects solely consist of a small amount of pixels in the aerial imagery,

Figure 5.15: Small and thin ditches are sometimes (partly) missed by the algorithm. In addition, trees next to ditches often limit the visibility of the water bodies from the air. Blue indicates the prediction of `water` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `water` objects in the ground truth. The numbers indicate the percentage of correctly classified pixels by the algorithm for that object.

(a) **(b)**

Figure 5.16: Typical examples of `other` objects in the ground truth that are missed by the algorithm. Green indicates the prediction of `other` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `other` objects in the ground truth. The numbers indicate the percentage of correctly classified pixels by the algorithm for that object. a) Small objects which are not clearly distinguishable are often missed by algorithm. b) Thin or small segments on top of roads are often misinterpreted for `road`. In addition, at the bottom left, a building is mis-labeled for `other` in the ground truth and leads to a wrong false negative.

Figure 5.17: Polygonalization of the pixel-level false positives resulted in the count of 11 separate false positives in this example, where it could be argued that only 4 are present. Red indicates the polygonized false positive prediction of `building` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `building` objects in the ground truth.

resulting in a small impact on the overall IoU score. Contradictory, the size of the object is not of any value for object-level performance assessment, as every individual object is counted as one.

5.6.2 Object-level false positives

Generation of object-level false positives through polygonizing the eroded, false positive pixel predictions, showed to be an unsuccessful approach. Visual assessment indicated that the generated polygons did not completely represent the desired object-level false positives. Counts of generated polygons (i.e. 11,955 for `building`) were too high due to the introduction of noise. This is a result of the multitude of small clusters of pixels and individual pixels that were converted to individual polygons. Each cluster is counted as an individual false positive, whilst visual assessment indicates that a combination of clusters should often be counted as one. An example that demonstrate this issue is provided in Figure 5.17. These clusters are a consequence of the property of the algorithm that it performs pixel-level predictions, rather than instance-level predictions. These predictions are often not completely homogeneous and smooth. Excluding polygons that are smaller than a predefined threshold is not believed to solve the issue. Many small polygons also exist individually at separate locations that should be counted as separate false positive. Therefore, it is believed that the generation and quantification of meaningful, object-level false positives is currently not possible and can only be done if the algorithm produces completely smooth and homogeneous predictions. It is believed that false positives information is required to assess the quality of a model as a whole. Therefore, these findings support the methodological decision for this research work to use the mIoU and IoU for model performance comparison, rather than object-level performance.

5.7 Disputable inconsistencies

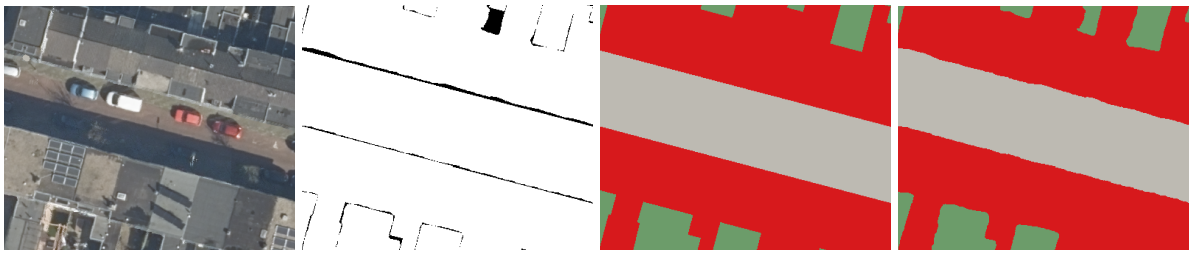
Through analysis of pixel-level error maps, hotspots of errors can be detected. In the original error map of FuseNet-SF5 using pixel-level, relative height, complete outlines of objects are visible (Figure 5.18a).

This indicates that either the prediction of borders by the algorithm is not accurate, or the ground truth of borders is not accurate. It should be noted that the 20 centimeter accuracy of the BGT is of a lower quality than the 10 centimeter resolution of the aerial imagery and the algorithm predictions. In addition, borders in the BGT are often represented as sharp transitions which do not always truly reflect the borders present in the imagery. Therefore, border pixels are often wrongly considered as mislabeling errors (Figure 5.19). Unfortunately, with the current ground truth data this problem can not be resolved.

(a) (b)

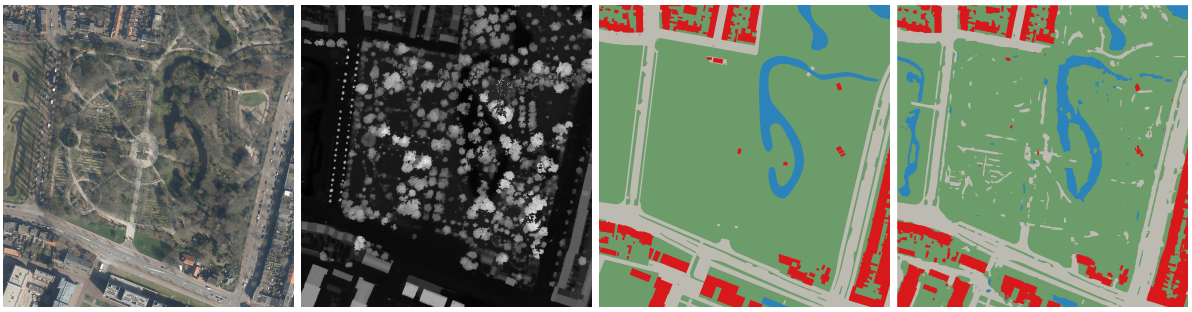
Figure 5.18: Error maps of the prediction of FuseNet-SF5 using pixel-level, relative height, on the test area. Per pixel it is indicated if the prediction corresponds to the ground truth (white), or if it deviates (black). a) Original error map. b) Eroded error map (40cm).

Morphological erosion eliminates the boundary errors from the error maps, allowing for a more clear overview of errors with a different cause (Figure 5.18b). The eroded error maps per class are provided in Figure A.10. Assessment of these maps showed that an extensive amount of these groups of misclassified pixel were not a mistake of the algorithm, but the result of errors in the BGT. In Section 5.6.1, analysis of the object-level false negatives already showed that many of the 'missed objects' are actually BGT errors. In addition, in Section 5.3.2 it was briefly mentioned that small roads are not always consistently mapped in the BGT. However, by analyzing the error hotspots, it became clear that more objects were missing or misplaced in the BGT. In Figure 5.20, Figure 5.21 and Figure 5.22 examples are presented where FuseNet-SF5 detects roads, water bodies and buildings that are missing in the ground



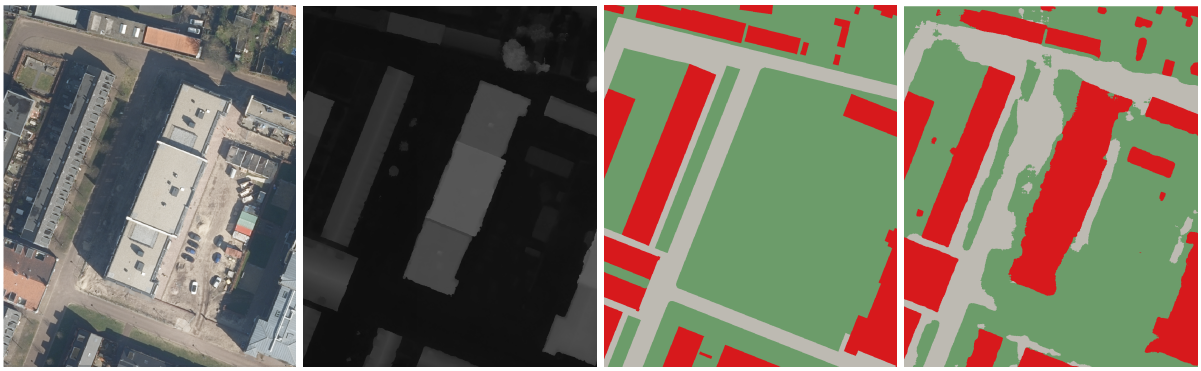
(a) (b) (c) (d)

Figure 5.19: Predicted border pixels are wrongly denoted as an error. Red = building, gray = road, green = other. (a) RGB true ortho (b) Error map (black is error) (c) Ground truth (d) Prediction FuseNet-SF5 using pixel-level, relative height information.



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

Figure 5.20: Small roads are not consistently mapped in the BGT whilst the algorithm does detect its presence. In addition, the BGT misses the ditch at the left of the image, which is detected by the algorithms. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) DSM (c) Ground truth (d) Prediction FuseNet-SF5 using pixel-level, relative height information.



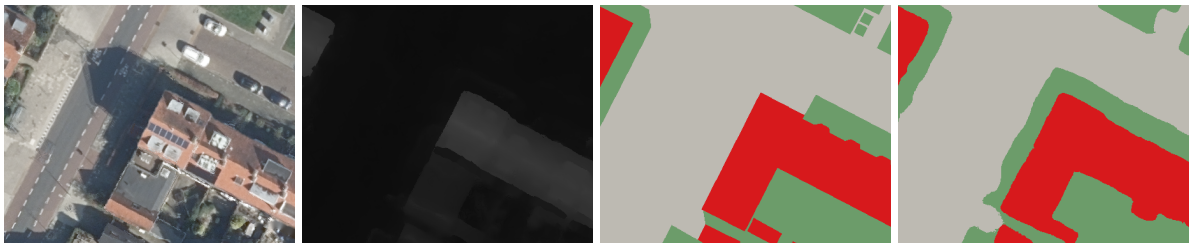
(a) (b) (c) (d)

Figure 5.21: A building is missing in the BGT while FuseNet-SF5 detects it. Red = building, gray = road, green = other. (a) RGB true ortho (b) DSM (c) Ground truth (d) Prediction FuseNet-SF5 using pixel-level, relative height information.

truth information. The problem of misplacement in the BGT is especially present for the class `building`, at the back side of houses (Figure 5.23). The combination of these missing and misplaced objects in the BGT and the previously described border issue, results in an unfair reduction of the calculated mIoU and IoU scores.

5.8 Comparison performance to related work

Due to the use of different datasets and different classes, comparison of the gained results to other studies is a challenging task. Nevertheless, the class `building` is also present in the ISPRS Vahingen 2D semantic labeling dataset, which is used by the researches of Kampffmeyer et al. [2016], Liu et al. [2017] and Audebert et al. [2018], discussed in Section 2.4.2 and Section 2.4.3. This dataset has a resolution of 9x9 cm per pixel, which is similar to the 10x10 cm used in this research work. Instead of RGB, the imagery in this dataset is IRRG. For height information, the researches use both an DSM and relative height information. In Table 5.11 an overview is provided of the F1 scores of the best performing models of these three studies on the class `building`. On first sight, it seems that the gained results in this study on the test data for the class `building` are slightly lower than the best performing approaches of related studies. However, some important remarks need to be made on this matter. Firstly, as discussed before, the studies have provided their results on the validation data instead of on separate test data. Furthermore, for some of these results eroded ground truth information is used, which will lead to an



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

Figure 5.22: `other` object around the buildings is not consistently mapped in the ground truth whilst the algorithm does detect its presence. Red = building, gray = road, green = other. (a) RGB true ortho (b) DSM (c) Ground truth (d) Prediction FuseNet-SF5 using pixel-level, relative height information.

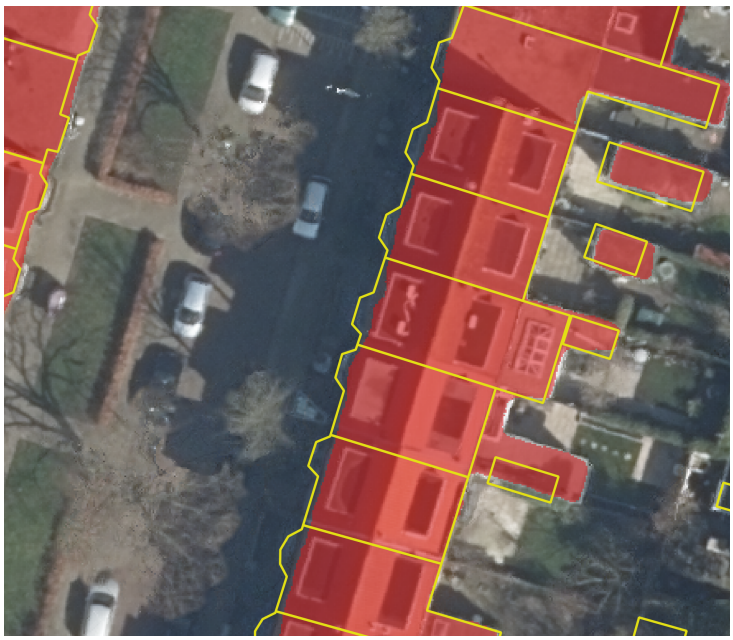


Figure 5.23: Misplacement of `building` objects in the BGT results in wrongly detected mis-labeling errors. Red indicates the prediction of `building` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `building` objects in the ground truth.

Method	F1 Building	Note
PB + FCN [Kampffmeyer et al., 2016]	**0.9586**	On validation data, with eroded ground truth boundaries.
HSN + OI erGT [Liu et al., 2017]	0.9466	On validation data, with eroded ground truth boundaries.
HSN + OI GT [Liu et al., 2017]	0.9237	On validation data, no eroded ground truth boundaries.
SegNet-RC [Audebert et al., 2018]	0.9450	On validation data, unclear if boundaries are eroded.
This study		
FuseNet-SF5-RHT (validation)	0.9436	On validation data, no eroded ground truth boundaries.
FuseNet-SF5-RHP (validation)	0.9429	On validation data, no eroded ground truth boundaries.
FuseNet-SF5-RHT (test)	0.9330	On test data, no eroded ground truth boundaries.
FuseNet-SF5-RHP (test)	0.9288	On test data, no eroded ground truth boundaries.

Table 5.11: Results gained by related studies and this study for the class `building`. PB = Patch based, HSN = Houreglass-shaped network, OI = Overlap inference, GT = Ground truth, erGT = Eroded ground truth, RC = Residual correction, RHP = Relative height (pixel-level), RHT = Relative height (tile-level).

increase in F1 performance (i.e. more than +2% for the work of Liu et al. [2017]). Therefore, it could be argued that at least 2 percent of the F1 score achieved by Kampffmeyer et al. [2016] needs to be subtracted, to allow for a more valid comparison to results of this thesis. Consequently, while still emphasizing the limitations of this comparison, it can be argued that the quality of the class `building` for the best performing models of this study, are comparable to the state of the art.

6 Conclusions & future work

6.1 Conclusion

The conclusion of this study consist of three parts. First, limitations of the used methodology are discussed (Section 6.1.1). Next, answers are provided to the research questions (Section 6.1.2). Finally, the contributions of this study are provided (Section 6.1.3).

6.1.1 Discussion

Even though the methodology used is considered to be of sufficient quality for the objectives of this study, some remarks need to be made. In the following paragraphs the most important limitations will be discussed.

Firstly, as for U-Net not the exact same training data extent is used to learn the model parameters. Therefore, it could be argued that a direct comparison to the other architectures is not completely valid. A better approach would have been to select a training extent that is divisible in tiles with the dimensions required by the used implementation of U-Net (572x572) *and* by the used tile dimensions for the other models (512x512). Nevertheless, a solution than still needs to be sought for the problem that due to the deviating dimensions it is not possible to use the exact same training data set and validation data set.

Next, the hyperparameter selection procedure used in this study could be improved. Currently, hyperparameter selection is based on a comparison in performance of an experiment executed with one specific setting (i.e.initial learning rate of 1.0e-3), to an equal experiment with another setting (i.e. initial learning rate of 1.0e-4). If the initialized weights are equal for both experiments, and no other form of randomness is introduced throughout the training, two experiments with the exact same settings will provide the exact same results. However, with the pretrained weights RGB-Z stacking experiments, the Z-band is initialized with random weights. For FuseNet-SF5 the exact same pretrained weights are used for each experiment. However, for all the experiments, after every epoch, the training data is shuffled. Consequently, all the experiments include a source of randomness. Therefore, the current experimental setup does not allow for conclusions on if the used hyperparameter settings significantly outperform alternative settings. This can especially be a problem when the difference in performance is very small. A way to test for significance is to run the experiment X times with random initialization, where X is i.e. 10 or 100. Hereafter, using mIoU as metric, an unpaired t-test can be executed to establish a significant difference. However, due to the limit of computational power available, the procedure is considered as out of scope for this study. Even though the objective of this study to automatically generate high quality topographic maps was still achieved, inclusion of the t-tests is believed to improve the validity of the drawn conclusions.

Furthermore, more attention could have been drawn to the training procedure executed to generate the pretrained weights that are used in this study. Currently, it is not taken into consideration in what way the training examples were provided to the algorithm used to generate the pretrained weights. By normalizing the training examples in this study using mean and standard deviation values predefined by VGG-16, it is possible that results of a higher quality could have been achieved.

Lastly, several remarks needs to be made on the used height approaches. The pixel-level, relative height approach, in which the DTM (0.5m resolution) is subtracted from the DSM (0.1m resolution), has shown to outperform the other approaches. Even though pixel-level subtraction is performed, the deviating resolutions ensure that the subtraction is as detailed as the lowest resolution (0.5m). Therefore,

the pixel-level subtraction approach could be considered as a 0.5m level approach; one DTM value is subtracted of a patch of 5x5 pixels in the DSM.

Moreover, IDW interpolation was used to fill the holes in the DTM, before it was upsampled and subtracted from the DSM. The reasoning behind the interpolation was that the presence of holes could be considered as a pre-filtering of building objects. The output of the building and water detection technique of AHN would then be part of the input of the neural network and could influence the performance. This is an undesired consequence as this pre-filtering would guide the algorithm, whilst the goal is to let the algorithms learn for themselves. However, when holes are large (i.e. due to a large building), the contours of the holes were still visible in the interpolated output (Figure A.11). Relatively sharp, unrealistic transitions in terrain height are present at locations with large building blocks, grand buildings and large water bodies. In addition, the terrain height transition inside these interpolated holes is relatively smooth (Figure A.12). This potentially influences the pixel-level, relative height generation. At these locations, large blocks with sharp edges but smooth height transitions on the inside are subtracted from the DSM, rather than more realistic terrain height values that are comparable to the non-interpolated areas. Theoretically, this can make objects easier to distinguish by the algorithm for the wrong reasons. Whilst the relative height for areas that were not interpolated in the DTM will have incorporated the more rough and realistic terrain transitions, only smooth terrain heights were subtracted from areas corresponding to the in the DTM interpolated regions. Consequently, it is possible that the success of the pixel-level, relative height approach can not only be explained by the fact that it provides detailed information on the actual height of objects and by the flat nature of Haarlem. It is possible that part of the success originates from that the interpolated holes in the DTM ensure that `building` and `water` objects together become more distinctive from the `road` and `other` objects, and the other way around.

In order to examine if these interpolated holes influence the algorithms predictions, the 3D BAG of the TU Delft 3D geoinformation group [Dukai, 2018] was used. This dataset helped to identify buildings that are missing in the DTM of AHN, but that are present in the DSM of READAR. By examining the algorithms predictions at these locations, it could be noted that the algorithm was still able to detect the buildings (Figure A.13). This finding shows that *if* the algorithm has learned features on the typical appearance of areas that were interpolated in the DTM, it is *not* completely dependent on these features for detection of `building` objects. It would be even more valuable to examine the prediction of an area where a building is present in the DTM, but absent in the DSM. Unfortunately, no such building could be identified.

6.1.2 Research questions

This study addressed the question; *to what extent can convolutional neural networks be used for automatic semantic segmentation of RGB-Z aerial imagery?* In order to answer this question, five sub-questions were specified. The following paragraphs will answer each of these questions based on the results of this study.

- **Q: Which neural network architectures are a suitable starting point for semantic segmentation of aerial RGB-Z imagery?**

  A: The architectures FCN-8s, SegNet, U-Net and FuseNet-SF5 are considered to be a suitable starting point for the semantic segmentation of aerial, RGB-Z imagery.

  These architectures were selected through a literature research. Each of the architectures showed a successful performance of semantic segmentation on aerial imagery, or a similar segmentation task. The source code was openly available online in the form of a non-complex implementation in Python, using PyTorch. The implementations were not explicitly designed for one dataset and therefore allowed for the use of self-introduced input data. For FCN-8s, SegNet and U-Net height information can be added in the form of an extra band stacked on top of RGB. FuseNet-SF5 uses a separate encoder for the height information and fuses the extracted features into the feature maps corresponding to the RGB branch.

- **Q: To what extent does the addition of height information improve semantic segmentation results?**

  A: The addition of height information has improved the overall semantic segmentation quality (mIoU) of the data stacking approaches on average by 1 percent. The added value of the height information deviated per class. Visual analysis of the outputted predictions indicated that height data can provide valuable information that is essential for successful segmentation of complex environments.

  For all the data stacking architectures, the addition of height information resulted in an increase in overall segmentation quality. U-Net showed to benefit the most from the addition of height information (+2.14% on mIoU), the increase was marginal for FCN-8s (+0.56% on mIoU) and SegNet (+0.38% on mIoU). The additional value of height information to segmentation results strongly differed per class. Analysis of outputted prediction maps showed that height information can be crucial in complex semantic segmentation tasks, in which height is the only property that allows for distinction between classes (i.e. parking lots on top of building or shady areas). Therefore, it can be concluded that height data contains valuable information for the semantic segmentation of aerial imagery and when exploited in a suitable manner, has the ability to improve semantic segmentation quality.

- **Q: For which classes is the segmentation most successful; for `building`, `road`, `water` or `other`?**

  A: The segmentation was most successful for the classes `water` and `building`. The class `building` showed to benefit the most form the addition of height information.

  It can be concluded that both the RGB based models and the RGB-Z based models, provided the highest segmentation quality for `water` and `building`. The best performing model; FuseNet-SF5 using pixel-level, relative height information, scored IoUs of 0.8744 for `building`, 0.7865 for `road`, 0.9131 for `water` and 0.7966 for `other`. It is argued that the segmentation quality of `water` outperforms the other classes as the dark color, in combination with crisp borders, is believed to make the class clearly distinctive in RGB imagery. In addition, `building` is believed to outperform `road` and `other` as buildings usually have straight, crisp boundaries that are clearly visible from the air. On the other hand, many objects are present on top of, or directly next to roads, making them less visible in the imagery. Furthermore, the diversity in objects and terrain types that belong to the class `other` is believed to complicate the semantic segmentation of this class. When comparing data stacking models trained on RGB with models trained on RGB-Z, it could be concluded that the class `building` benefits the most from the addition of height information (+3% on average). The other classes were not affected to the same extent by height information. A possible explanation for the success for `building` is the property of buildings that they are always elevated from their surrounding, with a usually clearly visible, sharp transition. It is believed that height information is less distinctive for the other classes.

  The most successful model was able to correctly classify over 90% of the pixels of 65% of the `building` objects, 82% of the `road` objects, 58% of the `other` objects and 39% of the `water` objects, present in the ground truth. It should be noted that even though these numbers provide an indication on the model's capability to detect the ground truth objects, they do not provide any information on the false positives of the algorithm's prediction.

- **Q: How does the performance compare of different approaches on combining height information with RGB information (*stacking* and *fusion*) in a network?**

  A: Fusing height features, retrieved in a separate encoder, into RGB feature maps led to a higher overall semantic segmentation quality (+1.24% on mIoU) than when height information was provided as a stacked extra band on top of RGB information.

  The data fusion approach of FuseNet-SF5 outperforms the most successful data stacking approach (SegNet RGB-Z) by over 1 percent on mIoU. When compared to data stacking, data fusion showed an increase in performance of the classes `building` (+1.85%) and `water` (+3.02%), a marginal increase in performance of the class `other` (+0.68%), but a slight decrease in the segmentation quality of the class `road` (-0.60%). In addition, even though shade remained to impose a challenge, visual assessment showed that the fusion approach is able to achieve better segmentation results

in shady areas. This finding indicates that the use of two separate encoders, one for RGB and one for height data, allows for the extraction of a different type of features from the height information than when all data is stacked and fed to one encoder. Therefore, it is believed that with data fusion the potential of height information for semantic segmentation of aerial imagery is exploited to a higher degree than when data stacking is used.

- **Q: What type of height information provided to a network leads to the most accurate results?**

  A: Relative height, generated through pixel-level subtraction of the DTM from the DSM, provided the most accurate semantic segmentation results.

  In this study, experiments were executed with;

  - Absolute height (DSM)
  - Rescaled absolute height [0-1] (per tile)
  - Rescaled absolute height [0-1] (based on whole training/test area)
  - Relative height (pixel-level subtraction)
  - Relative height (tile-level subtraction of the median)

  Relative height generated through the pixel-level subtraction of the DTM from the DSM resulted in the highest mIoU of 0.8427 on the test data. Even though the achieved mIoUs indicated similar performances, visual comparison of the prediction maps showed that the relative height approaches produced higher quality maps when compared to the absolute height approaches. The most remarkable improvement is that relative height approaches produce more homogeneous and less messy predictions for large objects. Especially the model that used tile-level, rescaled height contained large holes and noise in predictions on large objects. No remarkable difference in performance was detected when visually comparing the pixel-level, relative height approach with the tile-level relative height approach. However, as the performance measure indicated a better performance of the pixel-level approach, this approach is selected as the height information leading to the most accurate results. The remark is made that it is believed that the flat nature of the terrain of Haarlem plays an important role in the success of the pixel-level, relative height approach. Therefore, it is stressed that it is not guaranteed that this approach also leads to the best performance at other geographical locations with more variation in terrain height.

6.1.3 Contributions

The main contributions of this research comprise:

- It is established that height information can add value to semantic segmentation of aerial RGB imagery.
- It is shown that adding height information through data fusion can result in a higher segmentation quality of aerial imagery than when data stacking approaches are used.
- It is demonstrated that presenting relative height, rather than absolute height, to a network can improve semantic segmentation quality of aerial imagery, especially for large objects.

6.2 Future work

Throughout this study, several new ideas arose that are recommended for future work. This section describes these recommendations. Firstly, Section 6.2.1 elaborates on the removal of wrong mislabeling errors from performance measure computation. Next, Section 6.2.2 suggests extended data augmentation. Hereafter, Section 6.2.3 recommends the subdivision of segmentation classes. Next, Section 6.2.4 proposes an alternative procedure for the generation of the relative height information. Finally, Section 6.2.5 provides a suggestion on stacking of different height types before data fusion.

6.2.1 BGT error removal

In Section 5.7 it was noted that some objects are misplaced or missing in the BGT and imprecise border representations are present as a result of deviating resolutions. Consequently, algorithm predictions at these locations are often wrongly considered as mislabeling errors. This leads to lower scores on performance measures. By distinguishing what part of the error is due to the algorithms and what part is due to the BGT, a more realistic estimate can be made on the quality of the produced models. Therefore, it is recommended to eliminate the BGT errors and boundary errors before calculating the performance measures. By applying morphological erosion to the error maps, clusters of errors became clearly visible (Figure 5.18b). One can manually assess these clusters and if indeed an error in the BGT is present, eliminate the corresponding pixels from the performance measure calculation procedure. In addition, similar to the ISPRS Vaihingen 2D semantic labeling dataset used by related studies (see Section 2.4.2), eroded ground truth data could be used during performance measure computation. This will eliminate the negative effect of class boundaries on performance scores.

6.2.2 Extended data augmentation

In this study, horizontal flipping is applied to reduce overfitting of the models to the training examples and therefore to generate higher quality models. For future work, it is recommended to explore additional forms of data augmentation techniques. Currently the assumption is made that vertical flipping can lead to unrealistic shadows. This possibly has a negative effect on the training procedure. However, this theory has not yet been researched in this thesis. In addition, the added value of zooming, adjustment of brightness and hue and a larger amount of overlap between training examples could be explored.

6.2.3 Subdivision of classes

The confusion matrices of the models on the test area have shown that a large part of the errors are pixels corresponding to the class `road` that are misclassified for `other` and pixels of the class `other` that are misclassified for `road`. A suggestion on a method to reduce these kind of errors comprises the introduction of new segmentation classes. This can be achieved through the subdivision of the existing ones. For this subdivision, object attributes present in the BGT, such as 'FysiekVoorkomen' (physical appearance), could be used. For example, it could be explored if dividing the class `road` into `paved road` and `dirt road` will lead to a reduction of misclassification errors and therefore a higher mIoU value.

6.2.4 Relative height without the DTM of AHN3

In this study, in order to generate the relative height approaches, the DTM of AHN3 is subtracted from the DSM of READAR. Alternatively, it is suggested to generate a DTM directly from the used DSM and subtract it from the DSM. This can be done using software such as TerraScan[1] or the PDAL library[2]. Even though these implementations are originally developed for LiDAR pointclouds which contain 'last return' information, they can also be used for pointclouds generated through stereo matching. The advantage of this technique is that both models then have the same resolution, allowing for actual pixel-level subtraction. In addition, data present in the models then originates from the same date. These properties could lead to a more accurate relative height representation. However, it should be noted that the DTM of AHN3 is manually checked for errors. Ideally, the proposed alternative method for generation of the DTM would also undergo manual assessment, to ensure that no new errors are introduced at locations were the model is not generated correctly. However, this would require extensive manual work. Furthermore, the generated interpolated DTM will still contain outlines for `building` and `water`

[1] http://www.terrasolid.com/products/terrascanpage.php
[2] https://pdal.io/tutorial/ground-filters.html

objects. Therefore, the issue is not solved that pre-filtering information is passed on to the algorithms that potentially influences the detection of these types of objects.

6.2.5 Fusing stacked height information

In this study, only one type of height information is used per experiment. Resulting prediction maps indicated that no height approach clearly outperformed all the other approaches in every segmentation case. Therefore, it is possible that different height approaches have the ability to augment each other. As an attempt to further exploit the potential added value that height information showed to have, it is suggested to combine different height types. Whilst still using separate encoders for RGB information and height data, several height approaches could be stacked before fed to its predestined encoder.

A Additional figures

A.1 Methodology flowchart

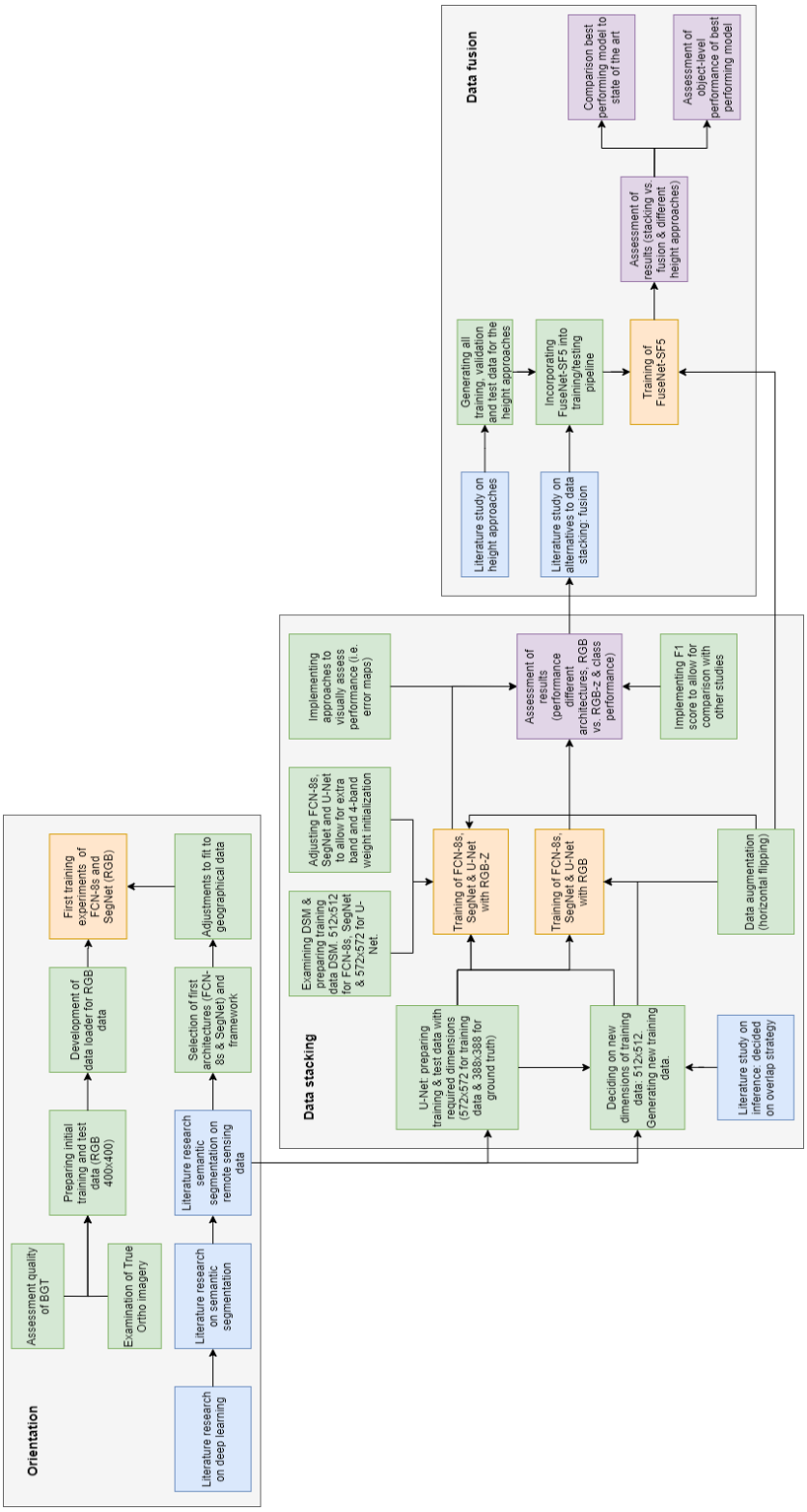


Figure A.1: Flowchart of the used methodology. Blue = literature study, green = preparation/implementation, orange = algorithm training, purple = result analysis.

A.2 Height approaches

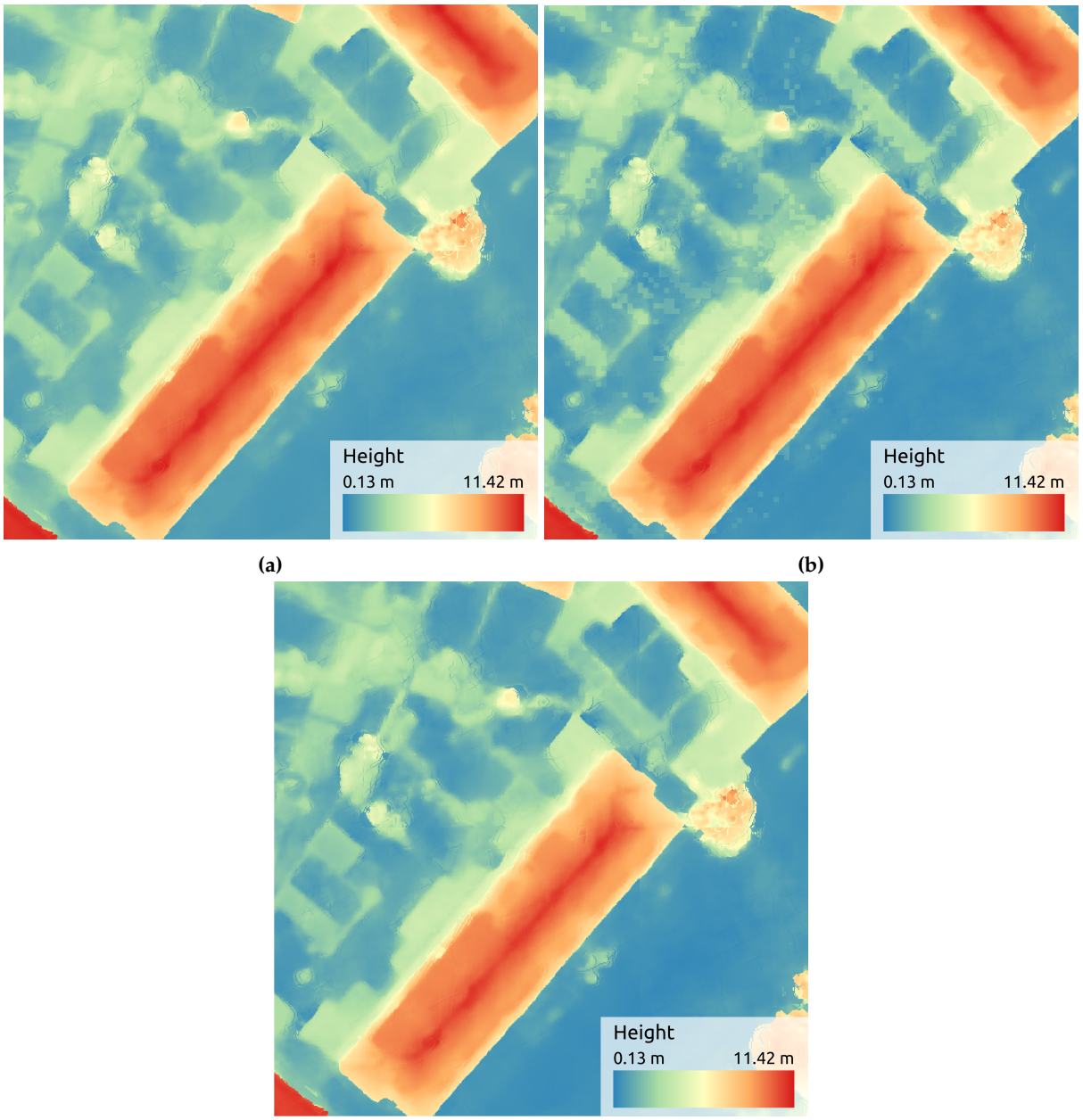
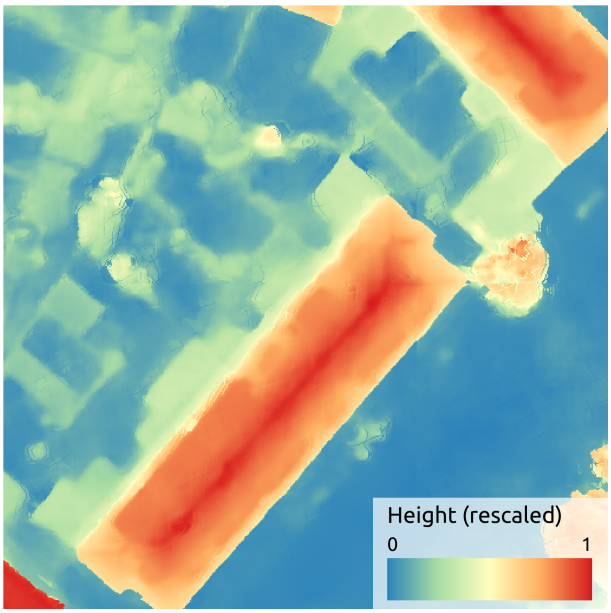
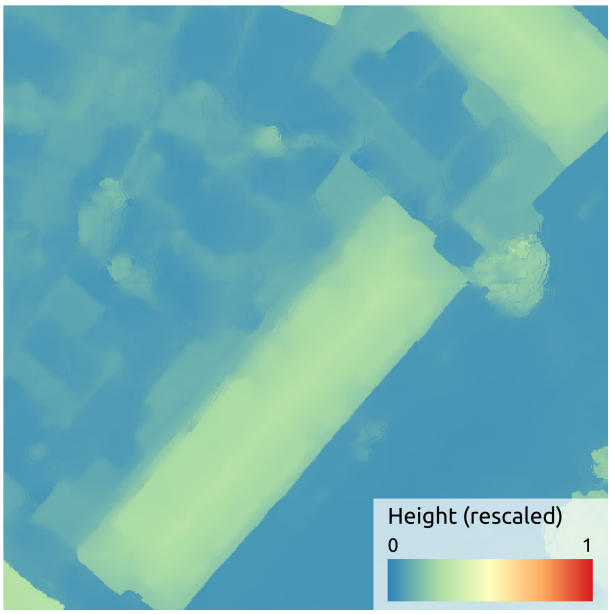


Figure A.2: Height information corresponding to the absolute and relative approaches of one training example. The color ramps are intentionally set to be equal, allowing for visual assessment of the differences. (a) Absolute height (DSM) (b) Relative height (pixel-level) (c) Relative height (tile-level)



(a)



(b)

Figure A.3: Height information corresponding to the rescaling approaches of one training example. The color ramps are intentionally set to be equal, allowing for visual assessment of the differences. (a) Rescaled height [0-1] (tile-level) (b) Rescaled height [0-1] (whole area)

A.3 Shade

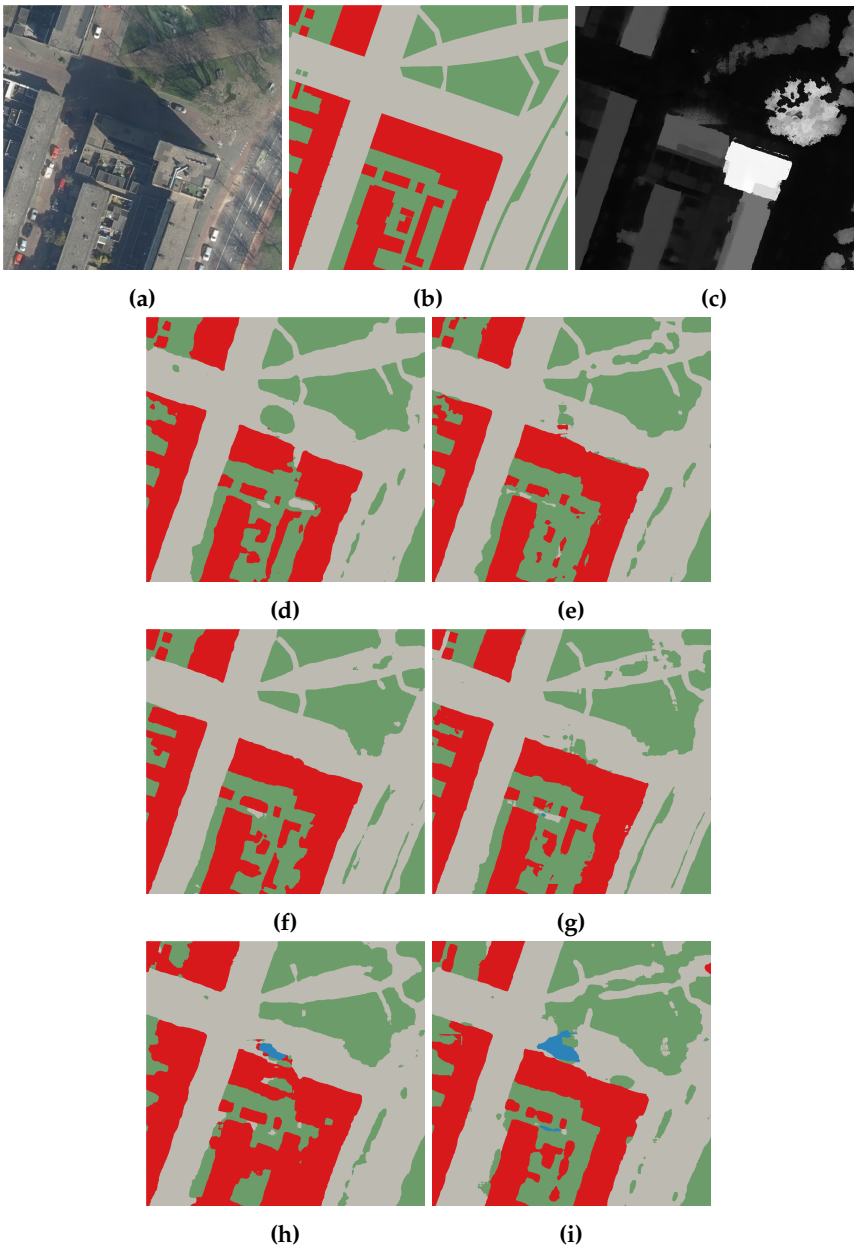


Figure A.4: For all data stacking models, the addition of height information does not improve segmentation quality at shaded areas. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) Ground truth (c) DSM (d) Prediction FCN-8s (RGB) (e) Prediction FCN-8s (RGB-Z) (f) Prediction SegNet (RGB) (g) Prediction SegNet (RGB-Z) (h) Prediction U-Net (RGB) (i) Prediction U-Net (RGB-Z)

A.4 Final prediction of FuseNet-SF5

(a) (b)

(c) (d)

Figure A.5: The prediction on the test area corresponding to the most successful model in this study; FuseNet-SF5 trained using pixel level, relative height. Red = building, gray = road, blue = water, green = other. (a) RGB true ortho (b) Relative height (DSM - DTM) (c) Ground truth (d) Prediction FuseNet-SF5

A.5 Object-level detection

Figure A.6: Object-detection for the class `building`. Red indicates the prediction of `building` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `building` objects in the ground truth. The numbers indicate the percentage of correctly classified pixels by the algorithm for that object.

Figure A.7: Object-detection for the class `road`. Purple indicates the prediction of `road` of FuseNet-SF5 using pixel-level, relative height. Purple indicates the outlines of the `road` objects in the ground truth. The numbers indicate the percentage of correctly classified pixels by the algorithm for that object.

Figure A.8: Object-detection for the class `water`. Blue indicates the prediction of `water` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `water` objects in the ground truth. The numbers indicate the percentage of correctly classified pixels by the algorithm for that object.

Figure A.9: Object-detection for the class `other`. Green indicates the prediction of `other` of FuseNet-SF5 using pixel-level, relative height. Yellow indicates the outlines of the `other` objects in the ground truth. The numbers indicate the percentage of correctly classified pixels by the algorithm for that object.

A.6 Eroded error maps

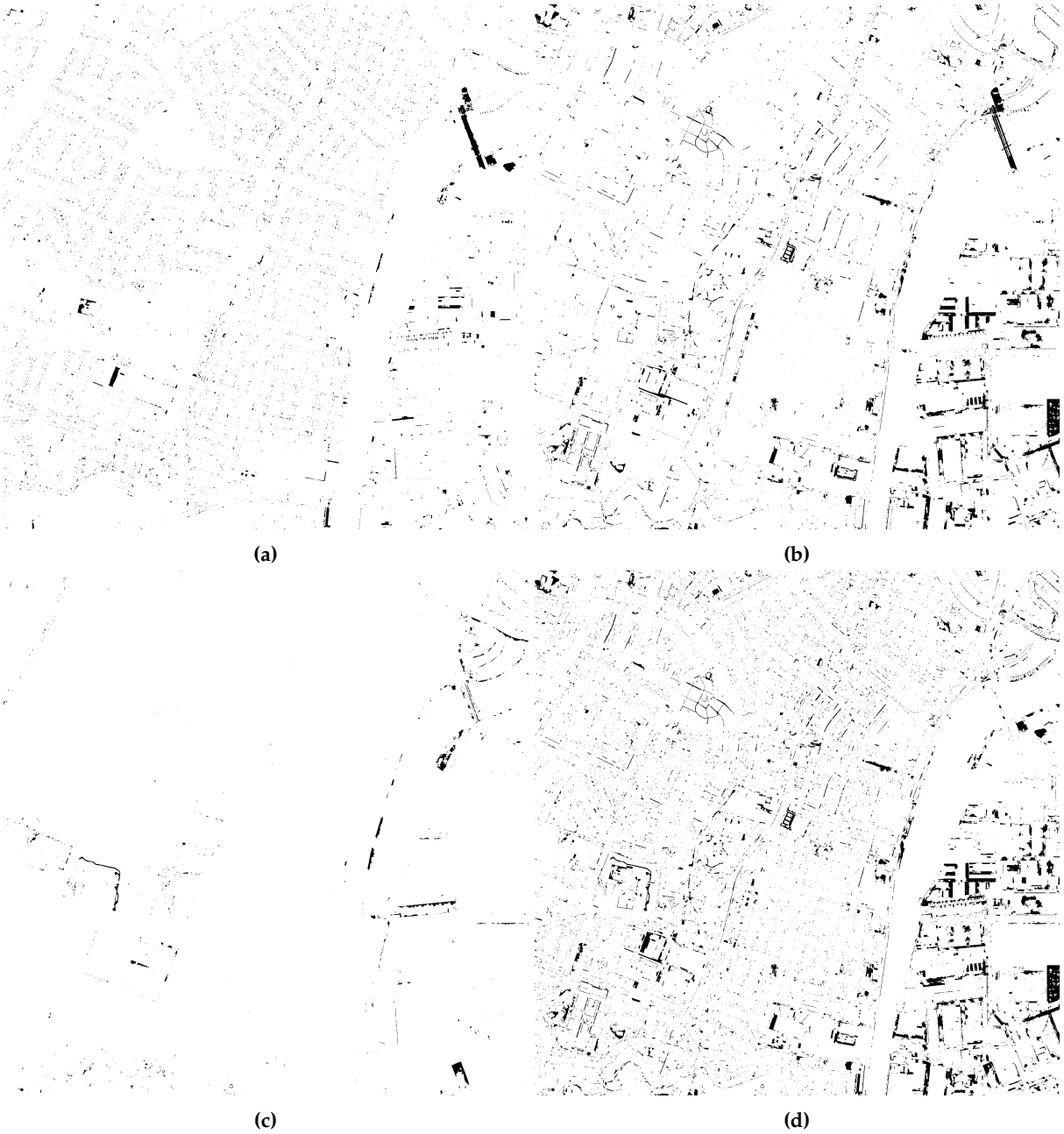
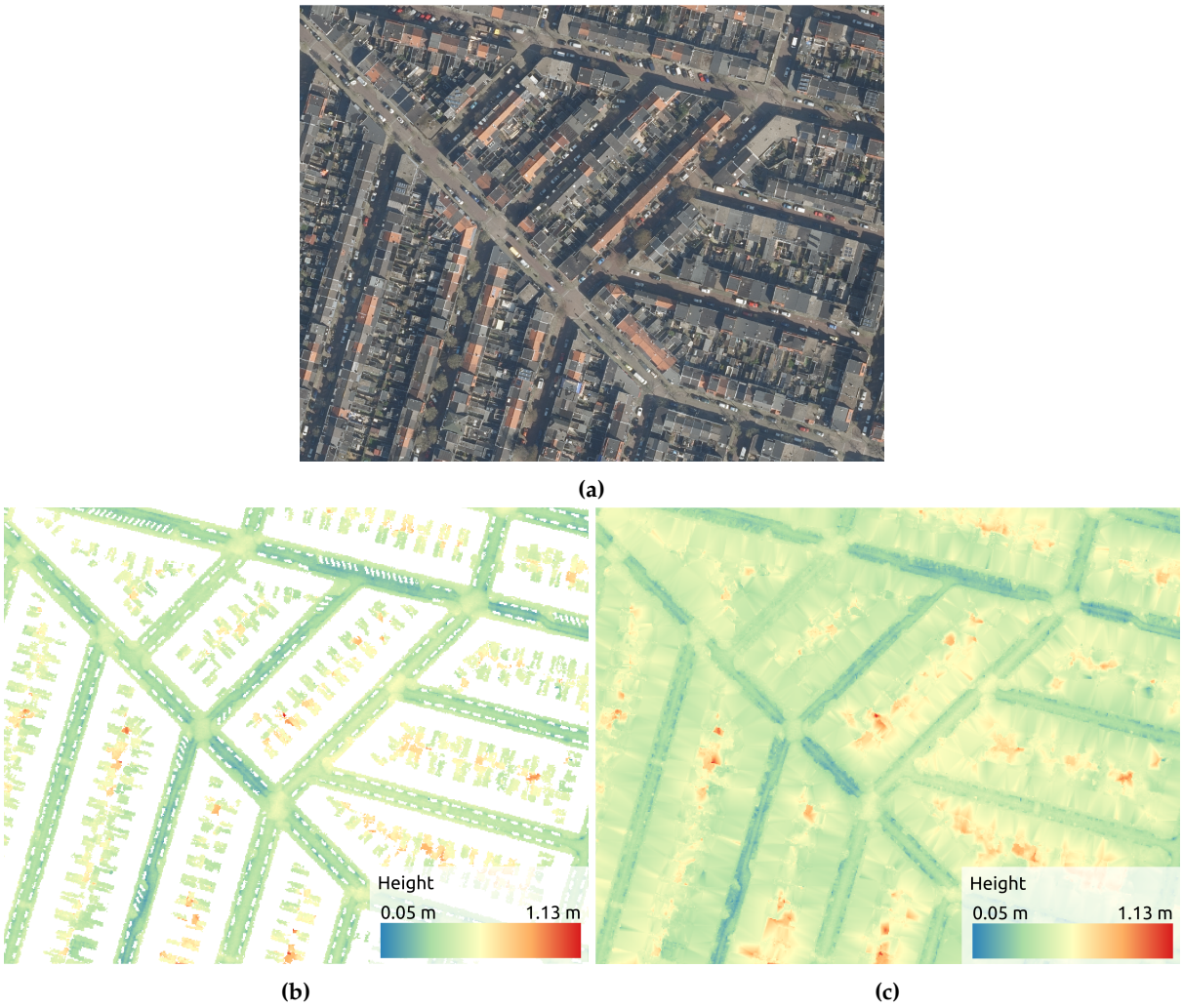


Figure A.10: The eroded error maps per class for the prediction of FuseNet-SF5 using pixel-level, relative height, on the test area. Per pixel it is indicated if the prediction corresponds to the ground truth (white), or if it deviates (black). a) Class `building` b) Class `road` c) Class `water` d) Class `other`

A.7 Interpolation of the DTM

Figure A.11: The interpolated DTM still contains some contours of buildings.



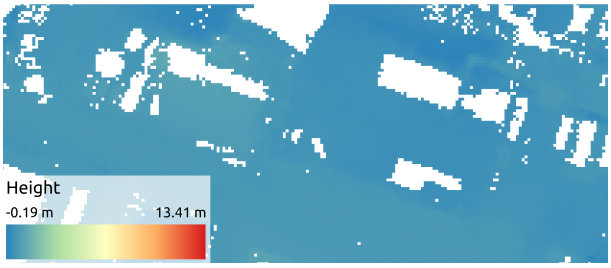
(a)

(b)

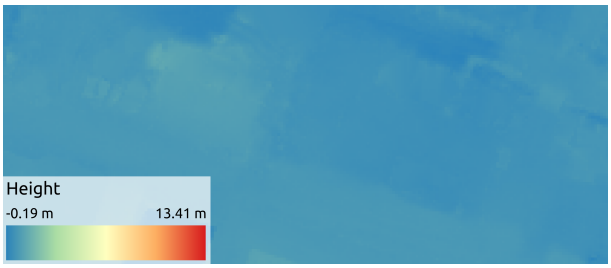
(c)

Figure A.12: Interpolating holes of the DTM often results in sharp height transitions at the borders and relatively smooth height transitions within the holes. a) RGB true ortho b) DTM c) Interpolated DTM

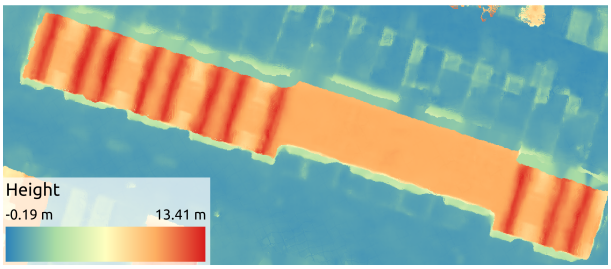
(a)



(b)



(c)



(d)

(e)

Figure A.13: Example of when no building object is present in the DTM but there is a building in the DSM. The algorithm is still able to detect the building object. a) RGB true ortho b) DTM c) Interpolated DTM d) DSM e) Prediction of FuseNet-SF5 using pixel-level, relative height. Red = building, gray = road, blue = water and green = other.

B Additional tables

B.1 Confusion matrices

		Prediction				
		Building	**Road**	**Water**	**Other**	**Sum**
Actual	**Building**	**103,645,897**	1,190,460	103,950	9,526,305	114,466,612
	Road	1,162,577	**87,718,204**	136,307	9,003,977	98,021,065
	Water	715,728	254,878	**33,385,942**	1,811,799	36,168,347
	Other	6,660,397	12,851,906	1,076,145	**139,765,704**	160,354,152
	Sum	112,184,599	102,015,448	34,702,344	160,107,785	409,010,176

Table B.1: Confusion matrix of SegNet (RGB) on the test data.

		Prediction				
		Building	**Road**	**Water**	**Other**	**Sum**
Actual	**Building**	**105,046,640**	961,453	174,759	8,283,760	114,466,612
	Road	1,165,707	**87,743,326**	178,998	8,933,034	98,021,065
	Water	1,133,757	206,011	**33,124,569**	1,704,010	36,168,347
	Other	6,274,098	12,917,119	942,875	**140,220,060**	160,354,152
	Sum	113,620,202	101,827,909	34,421,201	159,140,864	409,010,176

Table B.2: Confusion matrix of SegNet (RGB-Z) on the test data.

		Prediction				
		Building	**Road**	**Water**	**Other**	**Sum**
Actual	**Building**	**103,645,897**	1190,460	103,950	9,526,305	114,466,612
	Road	1,162,577	**87,718,204**	136,307	9,003,977	98,021,065
	Water	715,728	254,878	**33,385,942**	1,811,799	36,168,347
	Other	6,660,397	12,851,906	1,076,145	**139,765,704**	160,354,152
	Sum	112,184,599	10,2015,448	34,702,344	160,107,785	409,010,176

Table B.3: Confusion matrix of FuseNet-SF5 on the test data, using absolute height.

		Prediction				
		Building	**Road**	**Water**	**Other**	**Sum**
Actual	**Building**	**106,806,528**	850,443	56,360	6,753,281	114,466,612
	Road	1,441,692	**87,912,694**	265,227	8,401,452	98,021,065
	Water	474,888	160,855	**34,197,218**	1,335,386	36,168,347
	Other	5,762,744	12,747,545	961,288	**140,882,575**	160,354,152
	Sum	114,485,852	101,671,537	35,480,093	157,372,694	409,010,176

Table B.4: Confusion matrix of FuseNet-SF5 on the test data, using pixel-level, relative height.

		Prediction			
		Building	**Road**	**Water**	**Other**
Actual	**Building**	**93.31**	0.74	0.05	5.90
	Road	1.47	**89.69**	0.27	8.57
	Water	1.31	0.44	**94.55**	3.69
	Other	3.59	7.95	0.60	**87.86**

Table B.5: Confusion matrix of FuseNet-SF5 using pixel-level, relative height, on the test data. The values are given in percentages.

B.2 Results of experiments

				Hyperparameters						*Result on validation data*										*Results on test data*									
	Weights	**Height info**	**Hor. flip.**	**Loss function**	**Optimizer**	**Init. learning rate**	**Batch size**	**#ENI**	**Training time**	**Mean IoU**	**Average F1**	**IoU Other**	**IoU Building**	**IoU Road**	**IoU Water**	**F1 Other**	**F1 Building**	**F1 Road**	**F1 Water**	**Mean IoU**	**Average F1**	**IoU Other**	**IoU Building**	**IoU Road**	**IoU Water**	**F1 Other**	**F1 Building**	**F1 Road**	**F1 Water**
FCN-8s (RGB)																													
Weight init.																													
	Random	-	No	CP	Adam	1.0 e-4	5	20	6h 6m	0.7754	0.8733	0.7528	0.7834	0.7559	0.8093	0.8590	0.8786	0.8610	0.8946	0.7512	0.8567	0.6953	0.7728	0.7019	0.8348	0.8203	0.8718	0.8248	0.9100
	Pretrained	-	No	CP	Adam	1.0 e-4	5	20	4h 18m	0.8292	0.9065	0.8028	0.8462	0.8133	0.8544	0.8906	0.9167	0.8971	0.9215	0.8005	0.8887	0.7594	0.8238	0.7673	0.8515	0.8633	0.9034	0.8683	0.9198
Learning rate																													
	Pretrained	-	No	CP	Adam	1.0 e-3	5	20	3h 9m	0.7809	0.8768	0.7586	0.8053	0.7580	0.8016	0.8628	0.8922	0.8623	0.8899	0.7626	0.8638	0.7079	0.7892	0.6988	0.8544	0.8289	0.8822	0.8227	0.9215
	Pretrained	-	No	CP	Adam	1.0 e-4	5	20	4h 18m	0.8292	0.9065	0.8028	0.8462	0.8133	0.8544	0.8906	0.9167	0.8971	0.9215	0.8005	0.8887	0.7594	0.8238	0.7673	0.8515	0.8633	0.9034	0.8683	0.9198
	Pretrained	-	No	CP	Adam	1.0 e-5	5	20	8h 1m	0.8060	0.8924	0.7792	0.8223	0.7873	0.8351	0.8759	0.9025	0.8810	0.9101	0.7807	0.8760	0.7251	0.8055	0.7440	0.8484	0.8406	0.8923	0.8532	0.9180
Horizontal flip.																													
	Pretrained	-	Yes	CP	Adam	1.0 e-4	5	20	7h 30m	0.8426	0.9144	0.8175	0.8536	0.8318	0.8674	0.8996	0.9210	0.9082	0.9290	0.8147	0.8973	0.7713	0.8311	0.7821	0.8742	0.8709	0.9078	0.8777	0.9329
	Pretrained	-	No	CP	Adam	1.0 e-4	5	20	4h 18m	0.8292	0.9065	0.8028	0.8462	0.8133	0.8544	0.8906	0.9167	0.8971	0.9215	0.8005	0.8887	0.7594	0.8238	0.7673	0.8515	0.8633	0.9034	0.8683	0.9198
Optimizer																													
	Pretrained	-	No	CP	SGD	1.0 e-4	5	20	42h 55m	0.7600	0.8634	0.7357	0.7766	0.7371	0.7905	0.8478	0.8743	0.8487	0.8830	0.7580	0.8610	0.7010	0.7713	0.7099	0.8496	0.8243	0.8709	0.8303	0.9187
	Pretrained	-	No	CP	Adam	1.0 e-4	5	20	4h 18m	0.8292	0.9065	0.8028	0.8462	0.8133	0.8544	0.8906	0.9167	0.8971	0.9215	0.8005	0.8887	0.7594	0.8238	0.7673	0.8515	0.8633	0.9034	0.8683	0.9198
# Epochs no impr.																													
	Pretrained	-	No	CP	Adam	1.0 e-4	5	10	2h 0m	0.8282	0.9059	0.7983	0.8445	0.8124	0.8577	0.8879	0.9157	0.8965	0.9234	0.7996	0.8880	0.7514	0.8216	0.7639	0.8596	0.8580	0.9021	0.8675	0.9245
	Pretrained	-	No	CP	Adam	1.0 e-4	5	20	4h 18m	0.8292	0.9065	0.8028	0.8462	0.8133	0.8544	0.8906	0.9167	0.8971	0.9215	0.8005	0.8887	0.7594	0.8238	0.7673	0.8515	0.8633	0.9034	0.8683	0.9198
	Pretrained	-	No	CP	Adam	1.0 e-4	5	50	7h 51m	0.8343	0.9095	0.8072	0.8463	0.8240	0.8597	0.8933	0.9168	0.9035	0.9246	0.7998	0.8882	0.7549	0.8211	0.7635	0.8578	0.8603	0.9017	0.8672	0.9234
Loss function																													
	Pretrained	-	No	WCP	Adam	1.0 e-4	5	20	1h 53m	0.8211	0.9015	0.7889	0.8388	0.8035	0.8530	0.8820	0.9124	0.8910	0.9207	0.7972	0.8864	0.7464	0.8196	0.7586	0.8643	0.8548	0.9009	0.8627	0.9272
	Pretrained	-	No	CP	Adam	1.0 e-4	5	20	4h 18m	0.8292	0.9065	0.8028	0.8462	0.8133	0.8544	0.8906	0.9167	0.8971	0.9215	0.8005	0.8887	0.7594	0.8238	0.7673	0.8515	0.8633	0.9034	0.8683	0.9198
Best combination																													
	Pretrained	-	Yes	CP	Adam	1.0 e-4	5	50	11h 49m	0.8443	0.9155	0.8199	0.8552	0.8343	0.8678	0.9010	0.9219	0.9097	0.9292	0.8121	0.8958	0.7698	0.8305	0.7822	0.8661	0.8699	0.9074	0.8778	0.9282
FCN-8s (RGB-Z)																													
Weight init.																													
	Random	AH	No	CP	Adam	1.0 e-4	5	20	4h 1m	0.7748	0.8724	0.7437	0.8506	0.7331	0.7718	0.8530	0.9193	0.8460	0.8712	0.7566	0.8590	0.6837	0.8385	0.6666	0.8375	0.8121	0.9122	0.7999	0.9116
	Pretrained	AH	No	CP	Adam	1.0 e-4	5	20	1h 41m	0.8335	0.9088	0.8029	0.8756	0.8036	0.8517	0.8907	0.9337	0.8911	0.9199	0.8133	0.8958	0.7587	0.8613	0.7485	0.8849	0.8628	0.9255	0.8561	0.9389
Height type																													
	Pretrained	AH	No	CP	Adam	1.0 e-4	5	20	1h 41m	0.8335	0.9088	0.8029	0.8756	0.8036	0.8517	0.8907	0.9337	0.8911	0.9199	0.8133	0.8958	0.7587	0.8613	0.7485	0.8849	0.8628	0.9255	0.8561	0.9389
	Pretrained	SHT	No	CP	Adam	1.0 e-4	5	20	3h 17m	0.8364	0.9107	0.8105	0.8627	0.8143	0.8580	0.8954	0.9263	0.8977	0.9236	0.8137	0.8967	0.7726	0.8523	0.7722	0.8577	0.8717	0.9203	0.8714	0.9234
Best combination																													
	Pretrained	AH	Yes	CP	Adam	1.0 e-4	5	20	5h 34m	0.8472	0.9171	0.8230	0.8801	0.8261	0.8598	0.9029	0.9362	0.9048	0.9246	0.8265	0.9040	0.7784	0.8685	0.7660	0.8931	0.8754	0.9296	0.8675	0.9435
	Pretrained	SHT	Yes	CP	Adam	1.0 e-4	5	50	16h 54m	0.8483	0.9178	0.8261	0.8750	0.8342	0.8578	0.9048	0.9333	0.9096	0.9235	0.8177	0.8990	0.7725	0.8567	0.7714	0.8700	0.8717	0.9228	0.8710	0.9305

Table B.6: Results of all **FCN-8s** experiments performed in this study. ENI = Epochs of no improvement (on validation data), CP = Cross-entropy, WCP = Weighted cross-entropy, AH = Absolute height, SHT = Rescaled height (tile-level), SHW = Rescaled height (whole area), RHP = Relative height (pixel-level), RHT = Rescaled height (tile-level).

				Hyperparameters				*Result on validation data*									*Result on test data*						
	Weights	Height info	Hor. flip	Loss function	Optimizer	Init. learning rate	Batch size	#ENI	Training time	Mean IoU	Average F1	IoU Other	IoU Building	IoU Road	IoU Water	F1 Other	F1 Building	F1 Road	F1 Water	Mean IoU	Average F1	IoU Other	IoU Building	IoU Road	IoU Water	F1 Other	F1 Building	F1 Road	F1 Water
SegNet (RGB)																													
Weight init.	Random	No	CP	CP	Adam	1.0 e-4	5	20	3h 10m	0.7961	0.8664	0.7737	0.8176	0.7788	0.8145	0.8724	0.8976	0.8756	0.8977	0.7811	0.8762	0.7330	0.7992	0.7351	0.8570	0.8459	0.8984	0.8473	0.9230
	Pretrained	No	CP	CP	Adam	1.0 e-4	5	20	2h 36m	0.8337	0.9092	0.8096	0.8531	0.8179	0.8544	0.9048	0.9207	0.8998	0.9215	0.8116	0.8952	0.7622	0.8339	0.7702	0.8801	0.8651	0.9094	0.8702	0.9362
Learning rate	Pretrained	No	CP	CP	Adam	1.0 e-3	5	20	4h 43m	0.7961	0.8664	0.7701	0.8159	0.7856	0.8129	0.8701	0.8896	0.8799	0.8968	0.7741	0.8717	0.7203	0.7976	0.7307	0.8479	0.8374	0.8874	0.8444	0.9177
	Pretrained	No	CP	CP	Adam	1.0 e-4	5	20	2h 36m	0.8337	0.9092	0.8096	0.8531	0.8179	0.8544	0.9048	0.9207	0.8998	0.9215	0.8116	0.8952	0.7622	0.8339	0.7702	0.8801	0.8651	0.9094	0.8702	0.9362
	Pretrained	No	CP	CP	Adam	1.0 e-5	5	20	4h 59m	0.8091	0.8943	0.7860	0.8300	0.7895	0.8310	0.8802	0.9071	0.8824	0.9077	0.7955	0.8850	0.7400	0.8176	0.7470	0.8773	0.8586	0.8997	0.8552	0.9346
Horizontal flip	Pretrained	Yes	CP	CP	Adam	1.0 e-4	5	20	4h 34 m	0.8394	0.9126	0.8152	0.8573	0.8265	0.8586	0.8982	0.9232	0.9050	0.9238	0.8226	0.9020	0.7762	0.8420	0.7836	0.8880	0.8740	0.9142	0.8796	0.9411
	Pretrained	No	CP	CP	Adam	1.0 e-4	5	20	2h 36m	0.8337	0.9092	0.8096	0.8531	0.8179	0.8544	0.9048	0.9207	0.8998	0.9215	0.8116	0.8952	0.7622	0.8339	0.7702	0.8801	0.8651	0.9094	0.8702	0.9362
Optimizer	Pretrained	No	CP	SGD	Adam	1.0 e-4	5	20	4h 17m	0.2679	0.3826	0.5832	0.1837	0.2850	0.1216	0.7467	0.3103	0.4412	0.4424	0.2563	0.3738	0.5452	0.2004	0.2534	0.0216	0.7057	0.3339	0.4044	0.0812
	Pretrained	No	CP	CP	Adam	1.0 e-4	5	20	2h 36m	0.8337	0.9092	0.8096	0.8531	0.8179	0.8544	0.9048	0.9207	0.8998	0.9215	0.8116	0.8952	0.7622	0.8339	0.7702	0.8801	0.8651	0.9094	0.8702	0.9362
# Epochs no impr.	Pretrained	No	CP	CP	Adam	1.0 e-4	5	10	2h 36m	0.8371	0.9112	0.8145	0.8561	0.8207	0.8572	0.9075	0.9015	0.9231	0.8127	0.8952	0.7624	0.8346	0.7686	0.8811	0.8652	0.9098	0.8691	0.9368
	Pretrained	No	CP	CP	Adam	1.0 e-4	5	20	2h 36m	0.8337	0.9092	0.8096	0.8531	0.8179	0.8544	0.9048	0.9207	0.8998	0.9215	0.8116	0.8952	0.7622	0.8339	0.7702	0.8801	0.8651	0.9094	0.8702	0.9362
	Pretrained	No	CP	CP	Adam	1.0 e-4	5	30	8h 40m	0.8412	0.9136	0.8149	0.8567	0.8256	0.8677	0.8980	0.9228	0.9045	0.9292	0.8143	0.8969	0.7676	0.8326	0.7761	0.8808	0.8685	0.9086	0.8759	0.9366
Loss function	Pretrained	No	WCP	CP	Adam	1.0 e-4	5	20	3h 45m	0.8362	0.9106	0.8084	0.8519	0.8189	0.8655	0.8941	0.9200	0.9034	0.9279	0.8115	0.8949	0.7561	0.8327	0.7664	0.8898	0.8611	0.9087	0.8677	0.9422
	Pretrained	No	CP	CP	Adam	1.0 e-4	5	20	2h 36m	0.8337	0.9092	0.8096	0.8531	0.8179	0.8544	0.9048	0.9207	0.8998	0.9215	0.8116	0.8952	0.7622	0.8339	0.7702	0.8801	0.8651	0.9094	0.8702	0.9362
Best combination	Pretrained	Yes	WCP	CP	Adam	1.0 e-4	5	30	9h 48m	0.8426	0.9144	0.8163	0.8592	0.8269	0.8660	0.8989	0.9242	0.9064	0.9282	0.8178	0.8988	0.7636	0.8351	0.7748	0.8976	0.8660	0.9101	0.8701	0.9460
	Pretrained	Yes	CP	CP	Adam	1.0 e-4	5	30	12h 27m	0.8452	0.9160	0.8199	0.8605	0.8326	0.8677	0.9010	0.9250	0.9087	0.9292	0.8219	0.9015	0.7735	0.8426	0.7810	0.8987	0.8723	0.9146	0.8770	0.9422
SegNet (RG-FZ)																													
Weight init.																													
	Random	No	CP	CP	Adam	1.0 e-4	5	20	4h 27m	0.8036	0.8888	0.7709	0.8549	0.7633	0.8134	0.8706	0.9218	0.8658	0.8971	0.7840	0.8773	0.7215	0.8370	0.7134	0.8644	0.8382	0.9113	0.8327	0.9271
	Pretrained	No	CP	CP	Adam	1.0 e-4	5	20	2h 35m	0.8337	0.9103	0.8094	0.8737	0.8079	0.8519	0.8947	0.9236	0.8937	0.9200	0.8140	0.8966	0.7636	0.8497	0.7627	0.8801	0.8659	0.9187	0.8654	0.9362
Height type	Pretrained	AH	No	CP	Adam	1.0 e-4	5	20	2h 35m	0.8337	0.9103	0.8094	0.8737	0.8079	0.8519	0.8947	0.9236	0.8937	0.9200	0.8140	0.8966	0.7636	0.8497	0.7627	0.8801	0.8659	0.9187	0.8654	0.9362
	Pretrained	SHT	No	CP	Adam	1.0 e-4	5	20	2h 57m	0.8417	0.9139	0.8185	0.8655	0.8216	0.8611	0.9002	0.9279	0.9021	0.9253	0.8221	0.9017	0.7767	0.8476	0.7783	0.8858	0.8743	0.9175	0.8753	0.9395
Best combination																													
	Pretrained	SHT	Yes	CP	Adam	1.0 e-4	5	30	16h 54m	0.8501	0.9189	0.8290	0.8700	0.8366	0.8619	0.9065	0.9322	0.9110	0.9258	0.8357	0.9039	0.7822	0.8538	0.7827	0.8841	0.8778	0.9211	0.8761	0.9385

Table B.7: Results of all **SegNet** experiments performed in this study. ENI = Epochs of no improvement (on validation data), CP = Cross-entropy, WCP = Weighted cross-entropy, AH = Absolute height, SHT = Rescaled height (tile-level), SHW = Rescaled height (whole area), RHP = Relative height (pixel-level), RHT = Rescaled height (tile-level).

				Hyperparameters						*Result on validation data*										*Results on test data*									
	Weights	**Height info**	**Hor. flip.**	**Loss function**	**Optimizer**	**Init. learning rate**	**Batch size**	**#ENI**	**Training time**	**Mean IoU**	**Average F1**	**IoU Other**	**IoU Building**	**IoU Road**	**IoU Water**	**F1 Other**	**F1 Building**	**F1 Road**	**F1 Water**	**Mean IoU**	**Average F1**	**IoU Other**	**IoU Building**	**IoU Road**	**IoU Water**	**F1 Other**	**F1 Building**	**F1 Road**	**F1 Water**
U-Net (RGB)																													
Learning rate																													
	Random	-	No	CP	Adam	1.0 e-3	5	20	1h 0m	0.7611	0.8641	0.7572	0.7762	0.7212	0.7899	0.8618	0.8740	0.8380	0.8826	0.7469	0.8539	0.7007	0.7715	0.6853	0.8302	0.8240	0.8710	0.8133	0.9072
	Random	-	No	CP	Adam	1.0 e-4	5	20	0h 54m	0.7679	0.8683	0.7650	0.7809	0.7175	0.8080	0.8669	0.8770	0.8355	0.8938	0.7580	0.8612	0.7233	0.7776	0.6927	0.8382	0.8394	0.8749	0.8185	0.9120
	Random	-	No	CP	Adam	1.0 e-5	5	20	1h 36m	0.7268	0.8415	0.7332	0.7356	0.6826	0.7558	0.8461	0.8476	0.8114	0.8609	0.7220	0.8366	0.6803	0.7351	0.6431	0.8295	0.8097	0.8473	0.7828	0.9068
Horizontal flip.																													
	Random	-	Yes	CP	Adam	1.0 e-4	5	20	2h 21m	0.7833	0.8779	0.7725	0.7868	0.7314	0.8424	0.8716	0.8807	0.8449	0.9144	0.7640	0.8646	0.7213	0.7831	0.6874	0.8643	0.8381	0.8784	0.8147	0.9272
	Random	-	No	CP	Adam	1.0 e-4	5	20	0h 54m	0.7679	0.8683	0.7650	0.7809	0.7175	0.8080	0.8669	0.8770	0.8355	0.8938	0.7580	0.8612	0.7233	0.7776	0.6927	0.8382	0.8394	0.8749	0.8185	0.9120
Optimizer																													
	Random	-	No	CP	SGD	1.0 e-4	5	20	0h 14m	0.1195	0.1636	0.4706	0.0000	0.0015	0.0058	0.6400	0.0001	0.0030	0.0115	0.1064	0.1512	0.4176	0.0000	0.0014	0.0064	0.5892	0.0001	0.0029	0.0128
	Random	-	No	CP	Adam	1.0 e-4	5	20	0h 54m	0.7679	0.8683	0.7650	0.7809	0.7175	0.8080	0.8669	0.8770	0.8355	0.8938	0.7580	0.8612	0.7233	0.7776	0.6927	0.8382	0.8394	0.8749	0.8185	0.9120
# Epochs no impr.																													
	Random	-	No	CP	Adam	1.0 e-4	5	10	0h 47m	0.7639	0.8656	0.7550	0.7778	0.7074	0.8133	0.8604	0.8750	0.8286	0.8983	0.7278	0.8418	0.7097	0.7554	0.6730	0.7729	0.8302	0.8607	0.8046	0.8719
	Random	-	No	CP	Adam	1.0 e-4	5	20	0h 54m	0.7679	0.8683	0.7650	0.7809	0.7175	0.8080	0.8669	0.8770	0.8355	0.8938	0.7580	0.8612	0.7233	0.7776	0.6927	0.8382	0.8394	0.8749	0.8185	0.9120
	Random	-	No	CP	Adam	1.0 e-4	5	50	2h 16m	0.7809	0.8765	0.7714	0.7872	0.7334	0.8316	0.8709	0.8810	0.8462	0.9081	0.7583	0.8612	0.7188	0.7788	0.6900	0.8454	0.8364	0.8757	0.8166	0.9162
Loss function																													
	Random	-	No	WCP	Adam	1.0 e-4	5	20	0h 48m	0.7444	0.8530	0.7376	0.7658	0.6888	0.7857	0.8490	0.8673	0.8157	0.8800	0.7242	0.8391	0.6900	0.7565	0.6654	0.7850	0.8166	0.8614	0.7991	0.8795
	Random	-	No	CP	Adam	1.0 e-4	5	20	0h 54m	0.7679	0.8683	0.7650	0.7809	0.7175	0.8080	0.8669	0.8770	0.8355	0.8938	0.7580	0.8612	0.7233	0.7776	0.6927	0.8382	0.8394	0.8749	0.8185	0.9120
Best combinations																													
	Random	-	Yes	CP	Adam	1.0 e-4	5	50	2h 51m	0.7837	0.8782	0.7715	0.7882	0.7334	0.8417	0.8710	0.8815	0.8462	0.9140	0.7637	0.8647	0.7225	0.7814	0.6974	0.8535	0.8389	0.8773	0.8217	0.9210
U-Net (RGB-Z)																													
Height type																													
	Random	AH	No	CP	Adam	1.0 e-4	5	20	0h 45 m	0.7739	0.8716	0.7614	0.8376	0.7013	0.7954	0.8645	0.9116	0.8244	0.8860	0.7655	0.8648	0.7172	0.8492	0.6600	0.8355	0.8353	0.9184	0.7952	0.9104
	Random	SHT	No	CP	Adam	1.0 e-4	5	20	1h 7m	0.7909	0.8827	0.7843	0.8252	0.7300	0.8241	0.8791	0.9042	0.8440	0.9036	0.7814	0.8761	0.7485	0.8323	0.7037	0.8410	0.8561	0.9085	0.8261	0.9137
Best combination																													
	Random	SHT	No	CP	Adam	1.0 e-4	5	50	3h 14m	0.8100	0.8946	0.7995	0.8416	0.7554	0.8434	0.8886	0.9140	0.8606	0.9151	0.7851	0.8786	0.7521	0.8384	0.7134	0.8365	0.8585	0.9121	0.8328	0.9110

Table B.8: Results of all **U-Net** experiments performed in this study. ENI = Epochs of no improvement (on validation data), CP = Cross-entropy, WCP = Weighted cross-entropy, AH = Absolute height, SHT = Rescaled height (tile-level), SHW = Rescaled height (whole area), RHP = Relative height (pixel-level), RHT = Rescaled height (tile-level).

				Hyperparameters						*Result on validation data*										*Results on test data*									
	Weights	**Height info**	**Hor. flip.**	**Loss function**	**Optimizer**	**Init. learning rate**	**Batch size**	**#ENI**	**Training time**	**Mean IoU**	**Average F1**	**IoU Other**	**IoU Building**	**IoU Road**	**IoU Water**	**F1 Other**	**F1 Building**	**F1 Road**	**F1 Water**	**Mean IoU**	**Average F1**	**IoU Other**	**IoU Building**	**IoU Road**	**IoU Water**	**F1 Other**	**F1 Building**	**F1 Road**	**F1 Water**
Weight init.																													
	Random	AH	No	CP	Adam	1.0 e-4	5	20	7h 45m	0.8290	0.9062	0.8004	0.8720	0.7972	0.8466	0.8891	0.9316	0.8871	0.9169	0.8202	0.9001	0.7672	0.8668	0.7570	0.8900	0.8683	0.9286	0.8617	0.9418
	Pretrained	AH	No	CP	Adam	1.0 e-4	4	20	7h 27m	0.8502	0.9189	0.8261	0.8837	0.8252	0.8660	0.9048	0.9382	0.9042	0.9282	0.8366	0.9099	0.7842	0.8747	0.7715	0.9161	0.8791	0.9332	0.8710	0.9562
Learning rate																													
	Pretrained	AH	No	CP	Adam	1.0 e-3	4	20	5h 58m	0.7955	0.8835	0.7667	0.8644	0.7551	0.7956	0.8680	0.9273	0.8605	0.8862	0.7978	0.8859	0.7410	0.8610	0.7187	0.8705	0.8513	0.9253	0.8363	0.9308
	Pretrained	AH	No	CP	Adam	1.0 e-4	4	20	7h 27m	0.8502	0.9189	0.8261	0.8837	0.8252	0.8660	0.9048	0.9382	0.9042	0.9282	0.8366	0.9099	0.7842	0.8747	0.7715	0.9161	0.8791	0.9332	0.8710	0.9562
	Pretrained	AH	No	CP	Adam	1.0 e-5	4	20	3h 18m	0.6022	0.7398	0.6568	0.7543	0.3548	0.6430	0.7929	0.8599	0.5238	0.7827	0.5981	0.7215	0.6332	0.7782	0.2354	0.7457	0.7754	0.8753	0.3811	0.8543
Horizontal flip.																													
	Pretrained	AH	Yes	CP	Adam	1.0 e-4	4	20	16h 23m	0.8572	0.9229	0.8344	0.8931	0.8315	0.8695	0.9097	0.9436	0.9080	0.9302	0.8392	0.9116	0.7915	0.8752	0.7809	0.9094	0.8836	0.9334	0.8770	0.9525
	Pretrained	AH	No	CP	Adam	1.0 e-4	4	20	7h 27m	0.8502	0.9189	0.8261	0.8837	0.8252	0.8660	0.9048	0.9382	0.9042	0.9282	0.8366	0.9099	0.7842	0.8747	0.7715	0.9161	0.8791	0.9332	0.8710	0.9562
Optimizer																													
	Pretrained	AH	No	CP	SGD	1.0 e-4	4	20	1h 2m	0.1378	0.2178	0.3729	0.0554	0.1110	0.0118	0.5432	0.1049	0.1999	0.0233	0.1313	0.2106	0.3506	0.0579	0.1038	0.0131	0.5192	0.1095	0.1881	0.0258
	Pretrained	AH	No	CP	Adam	1.0 e-4	4	20	7h 27m	0.8502	0.9189	0.8261	0.8837	0.8252	0.8660	0.9048	0.9382	0.9042	0.9282	0.8366	0.9099	0.7842	0.8747	0.7715	0.9161	0.8791	0.9332	0.8710	0.9562
# Epochs no impr.																													
	Pretrained	AH	No	CP	Adam	1.0 e-4	4	10	3h 27m	0.8333	0.9088	0.8096	0.8803	0.8055	0.8376	0.8948	0.9363	0.8923	0.9116	0.8244	0.9024	0.7700	0.8652	0.7567	0.9057	0.8700	0.9277	0.8615	0.9505
	Pretrained	AH	No	CP	Adam	1.0 e-4	4	20	7h 27m	0.8502	0.9189	0.8261	0.8837	0.8252	0.8660	0.9048	0.9382	0.9042	0.9282	0.8366	0.9099	0.7842	0.8747	0.7715	0.9161	0.8791	0.9332	0.8710	0.9562
	Pretrained	AH	No	CP	Adam	1.0 e-4	4	50	20h 39m	0.8564	0.9225	0.8336	0.8892	0.8370	0.8658	0.9093	0.9414	0.9113	0.9281	0.8362	0.9097	0.7851	0.8729	0.7761	0.9106	0.8796	0.9322	0.8739	0.9532
Loss function																													
	Pretrained	AH	No	WCP	Adam	1.0 e-4	4	20	7h 26m	0.8403	0.9129	0.8103	0.8805	0.8113	0.8591	0.8952	0.9365	0.8958	0.9242	0.8258	0.9033	0.7691	0.8706	0.7601	0.9034	0.8695	0.9308	0.8637	0.9492
	Pretrained	AH	No	CP	Adam	1.0 e-4	4	20	7h 27m	0.8502	0.9189	0.8261	0.8837	0.8252	0.8660	0.9048	0.9382	0.9042	0.9282	0.8366	0.9099	0.7842	0.8747	0.7715	0.9161	0.8791	0.9332	0.8710	0.9562
Height type																													
	Pretrained	AH	Yes	CP	Adam	1.0 e-4	4	50	27h 40m	0.8573	0.9230	0.8361	0.8913	0.8355	0.8663	0.9107	0.9425	0.9104	0.9284	0.8381	0.9109	0.7890	0.8723	0.7767	0.9143	0.8821	0.9318	0.8743	0.9552
	Pretrained	SHT	Yes	CP	Adam	1.0 e-4	4	50	32h 59m	0.8543	0.9212	0.8327	0.8869	0.8305	0.8671	0.9087	0.9400	0.9074	0.9288	0.8326	0.9077	0.7860	0.8671	0.7770	0.9023	0.8802	0.9288	0.8733	0.9487
	Pretrained	SHW	Yes	CP	Adam	1.0 e-4	4	50	28h 7m	0.8599	0.9245	0.8408	0.8921	0.8438	0.8628	0.9135	0.9430	0.9153	0.9264	0.8401	0.9121	0.7897	0.8708	0.7846	0.9152	0.8825	0.9309	0.8793	0.9557
	Pretrained	RHP	Yes	CP	Adam	1.0 e-4	4	50	16h 47m	0.8729	0.9319	0.8456	0.8920	0.8428	0.9114	0.9163	0.9429	0.9147	0.9536	0.8427	0.9137	0.7966	0.8744	0.7865	0.9131	0.8868	0.9330	0.8805	0.9546
	Pretrained	RHT	Yes	CP	Adam	1.0 e-4	4	50	24h 53m	0.8574	0.9231	0.8353	0.8932	0.8322	0.8690	0.9103	0.9436	0.9084	0.9299	0.8384	0.9111	0.7891	0.8792	0.7785	0.9070	0.8821	0.9357	0.8754	0.9512

Table B.9: Results of all **FuseNet-SF5** experiments performed in this study. ENI = Epochs of no improvement (on validation data), CP = Cross-entropy, WCP = Weighted cross-entropy, AH = Absolute height, SHT = Rescaled height (tile-level), SHW = Rescaled height (whole area), RHP = Relative height (pixel-level), RHT = Rescaled height (tile-level).

C Reproducibility self-assessment

C.1 Reproducibility criteria

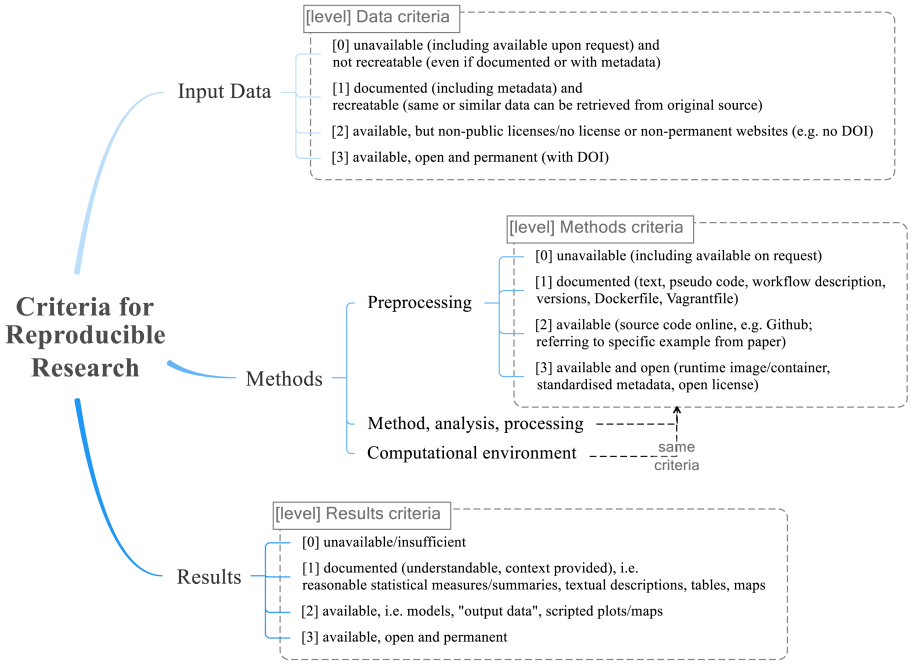


Figure C.1: Reproducibility criteria. Figure from Nüst et al. [2018].

C.2 Reproducibility scores

Criteria	Score
Input data	0
Preprocessing	1
Methods, analysis, processing	1
Computational environment	1
Results	1

Table C.1: The self-assigned scores for this study on the criteria for reproducibility, presented in Figure C.1.

C.3 Self-reflection

This research work was executed for the company READAR. This collaboration enabled the use of high quality input data and an external server that strongly enhanced the available computational power and memory. Nevertheless, the cooperation also came with restrictions leading to limitations in the reproducibility of this study. As the used true ortho imagery and matching DSM is developed by READAR, input data used in this work can not be made openly available, nor exactly recreated. The same restrictions are present for the source code corresponding to the preprocessing steps and methodology implementations and the Docker containers used. Nevertheless, details on the used methodology are described in the report in such a way that they could be implemented by the readers themselves. When considering the results of this study, all the calculated performance measures for all the experiments are provided. For the best performing models, textual descriptions and maps are given. This information allows for verification of the drawn conclusions. However, the trained models are not made openly available as these belong to READAR.

Bibliography

Actueel Hoogtebestand Nederland (2020). Data. https://www.ahn.nl/data. Accessed 13 May. 2020.

Audebert, N., Le Saux, B., and Lefèvre, S. (2018). Beyond RGB: Very high resolution urban remote sensing with multimodal deep networks. *ISPRS Journal of Photogrammetry and Remote Sensing*, 140:20–32.

Azizpour, H., Razavian, A. S., Sullivan, J., Maki, A., and Carlsson, S. (2015). Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1790–1802.

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.

Beeldmateriaal Nederland (2020). Voorjaarsvlucht. https://www.beeldmateriaal.nl/voorjaarsvlucht. Accessed 16 Jun. 2020.

Campos-Taberner, M., Romero-Soriano, A., Gatta, C., Camps-Valls, G., Lagrange, A., Saux, B. L., Beaupère, A., Boulch, A., Chan-Hon-Tong, A., Herbin, S., Randrianarivo, H., Ferecatu, M., Shimoni, M., Moser, G., and Tuia, D. (2016). Processing of Extremely High-Resolution LiDAR and RGB Data: Outcome of the 2015 IEEE GRSS Data Fusion Contest–Part A: 2-D Contest. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(12):5547–5559.

Chang, J.-R. and Chen, Y.-S. (2018). Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418.

Couprie, C., Farabet, C., Najman, L., and LeCun, Y. (2013). Indoor Semantic Segmentation using depth information.

Drozdzal, M., Vorontsov, E., Chartrand, G., Kadoury, S., and Pal, C. (2016). The importance of skip connections in biomedical image segmentation. In *Deep Learning and Data Labeling for Medical Applications*, pages 179–187. Springer.

Duc, Minh and Viet, Sang (2018). Method Description for Vaihingen:2D Labelling Challenge. http://ftp.ipi.uni-hannover.de/ISPRS_WGIII_website/ISPRSIII_4_Test_results/papers/BKHN10_method.pdf. Accessed 16 Jun. 2020.

Dukai, B. (2018). 3D Registration of Buildings and Addresses (BAG). http://3dbag.bk.tudelft.nl/. Accessed 10 Jun. 2020.

Duy, Tring, Van and Sang, Dihn, Viet (2018). Method Description for Vaihingen:2D Labelling Challenge. http://ftp.ipi.uni-hannover.de/ISPRS_WGIII_website/ISPRSIII_4_Test_results/papers/report_BKHN_4.pdf. Accessed 16 Jun. 2020.

Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M., and Burgard, W. (2015). Multimodal deep learning for robust RGB-D object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 681–687.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338.

Feng Ning, Delhomme, D., LeCun, Y., Piano, F., Bottou, L., and Barbano, P. (2005). Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 14(9):1360–1371.

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.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

Gupta, S., Girshick, R., Arbeláez, P., and Malik, J. (2014). Learning Rich Features from RGB-D Images for Object Detection and Segmentation.

Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. (2014). Simultaneous Detection and Segmentation.

Hazirbas, C., Ma, L., Domokos, C., and Cremers, D. (2017). FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-Based CNN Architecture. In Lai, S.-H., Lepetit, V., Nishino, K., and Sato, Y., editors, *Computer Vision – ACCV 2016*, volume 10111, pages 213–228. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hush, D. R. and Horne, B. G. (1993). Progress in supervised neural networks. *IEEE signal processing magazine*, 10(1):8–39.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

ISPRS (2018a). 2D Semantic Labeling - Vaihingen data. http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-vaihingen.html. Accessed 16 Jun. 2020.

ISPRS (2018b). 2D Semantic Labeling Contest. http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html. Accessed 16 Jun. 2020.

Kadaster (2018). *BGT*. Published: hhttps://zakelijk.kadaster.nl/bgt.

Kadaster (2019). BGT. https://zakelijk.kadaster.nl/bgt. Accessed 10 Feb. 2020.

Kaggle (2017). Dstl Satellite Imagery Feature Detection. https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection/overview. Accessed 16 Jun. 2020.

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. pages 1–9.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM*, 60(6):84–90. Place: New York, NY, USA Publisher: Association for Computing Machinery.

Lagrange, A., Saux, B. L., Beaupère, A., Boulch, A., Chan-Hon-Tong, A., Herbin, S., Randrianarivo, H., and Ferecatu, M. (2015). Benchmarking classification of earth-observation data: From learning explicit features to convolutional networks. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 4173–4176.

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

Lemmens, M. (2011). *Geo-information: technologies, applications and the environment*, volume 5. Springer Science & Business Media.

Liu, M., Tuzel, O., Ramalingam, S., and Chellappa, R. (2011). Entropy rate superpixel segmentation. In *CVPR 2011*, pages 2097–2104.

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. (2015). Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440.

Maggiori, E., Tarabalka, Y., Charpiat, G., and Alliez, P. (2017). Convolutional Neural Networks for Large-Scale Remote-Sensing Image Classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):645–657.

Nüst, D., Granell, C., Hofer, B., Konkol, M., Ostermann, F. O., Sileryte, R., and Cerutti, V. (2018). Reproducible research and giscience: an evaluation using agile conference papers. *PeerJ*, 6:e5072.

NYU (2019). Deep-Neural-Network. https://itp.nyu.edu/classes/roy20/illusion/deep-neural-network-1/. Accessed 18 Mar. 2019.

Paisitkriangkrai, S., Sherrah, J., Janney, P., and Van-Den Hengel, A. (2015). Effective Semantic Pixel Labelling With Convolutional Networks and Conditional Random Fields. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

Pohlen, T., Hermans, A., Mathias, M., and Leibe, B. (2017). Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Qi, X., Liao, R., Jia, J., Fidler, S., and Urtasun, R. (2017). 3D Graph Neural Networks for RGBD Semantic Segmentation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5209–5218, Venice. IEEE.

Qin, R., Tian, J., and Reinartz, P. (2016). 3D change detection – Approaches and applications. *ISPRS Journal of Photogrammetry and Remote Sensing*, 122:41–56.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation.

RSIP Vision (2018). Deep Learning and Convolutional Neural Networks: RSIP Vision Blogs. https://www.rsipvision.com/exploring-deep-learning/. Accessed 18 Mar. 2020.

Saito, S., Yamashita, T., and Aoki, Y. (2016). Multiple Object Extraction from Aerial Imagery with Convolutional Neural Networks. *Electronic Imaging*, 2016(10):1–9.

Saxe, A. M., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. (2011). On random weights and unsupervised feature learning. In *ICML*, volume 2, page 6.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

Shah, M. P. (2017). Semantic Segmentation Architectures Implemented in PyTorch. *https://github.com/meetshah1995/pytorch-semseg*.

Sheng, Y., Gong, P., and Biging, G. S. (2003). True Orthoimage Production for Forested Areas from Large-Scale Aerial Photographs. *Photogrammetric Engineering & Remote Sensing*, 69(3):259–266.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Song, S., Lichtenberg, S. P., and Xiao, J. (2015). SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Volpi, M. and Tuia, D. (2016). Dense semantic labeling of subdecimeter resolution images with convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):881–893.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018. Publisher: Hindawi.

Wang, Q., Lin, J., and Yuan, Y. (2016). Salient Band Selection for Hyperspectral Image Classification via Manifold Ranking. *IEEE Transactions on Neural Networks and Learning Systems*, 27(6):1279–1289.

Yuan, Y., Ma, D., and Wang, Q. (2016). Hyperspectral Anomaly Detection by Graph Pixel Selection. *IEEE Transactions on Cybernetics*, 46(12):3123–3134.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid Scene Parsing Network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhu, H., Meng, F., Cai, J., and Lu, S. (2016). Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation. *Journal of Visual Communication and Image Representation*, 34:12–27.

Zuurmond, Cor (2018). Height Estimation from Aerial Imagery with Stereo Matching Networks. https://scripties.uba.uva.nl/scriptie/662652. Accessed 12 Apr. 2020.

Colophon

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