Deep Learning for Pixelwise Classification of Hyperspectral Images

A generalizing model for a fixed scene subject to temporally changing weather, lighting and seasonal conditions

I.A.F. Snuverink

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Deep Learning for Pixelwise Classification of Hyperspectral Images

A generalizing model for a fixed scene subject to temporally changing weather, lighting and seasonal conditions

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

I.A.F. Snuverink

November 23, 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by TNO. Their cooperation is hereby gratefully acknowledged

Copyright © Delft Center for Systems and Control (DCSC)
All rights reserved.

Abstract

In hyperspectral (HS) imaging, for every pixel a spectrum of wavelengths is captured. These spectra represent material properties, i.e. the spectral signatures. So, classification of HS imagery is based on material properties. This thesis describes a framework to perform pixel-wise classification of HS images of a fixed scene subject to varying ambient conditions. TNO has recorded HS images over the course of one year, every hour between sunrise and sunset. Therefore, this data set is subject to a range of lighting, weather and seasonal conditions, degrading the recorded data. Traditionally, atmospheric models are used to correct for these effects, recovering spectral information. In this work an Fully Convolutional Network (FCN) is trained to perform the segmentation task which also learns to correct for ambient conditions, eliminating the need of implementing an atmospheric model or applying image normalization.

A single FCN (U-Net) is implemented to solve a five-class segmentation problem, distinguishing broad leaf trees, grass, sand, asphalt and artificial grass. To start training a neural network, the training data set requires a corresponding ground truth. A sparsely annotated mask is designed which fits images covering the entire recording period. In single-scene segmentation, annotating using a sparse mask is a quick method which allows for moving object borders. Furthermore, it avoids inclusion of mixed pixels. In order to reduce computational load, the training data set is formed by many small patches taken from the original HS images. Consequently, the network is trained with local spectral-spatial information. Training the standard U-Net proves to be limited to training data sets under relatively constant ambient conditions. In order to further enhance generalization over seasons, the network weights are rearranged so that a similar number of weights is maintained. This network is essential for training a complex training data set, as it is able to extract more informative features.

The standard U-Net trained with a simple training data set (i.e. relatively constant ambient conditions) achieves an accuracy $A > 94\%$ for both sunny and rainy test images irrespective of time of the day. However, this model is valid for a limited period of time. The customized U-Net trained with a complex training data set (i.e. highly varying ambient conditions) yields segmentations with an accuracy ranging between 86 and 93%. This model is valid for a longer period of time, covering multiple seasons. So the experiments show that there is a trade-off between segmentation accuracy and duration of model validity, which is controlled by network weight arrangement.

Table of Contents

List of Figures

List of Tables

Preface

My graduation process started approximately a year ago when I applied for a project at the Intelligent Imaging department of TNO. I felt intrigued by the possibilities of artificial neural networks, therefore I chose a project in which I could develop some experience and knowledge about the applications of such networks. A whole new world of opened up to me. I would like to thank everyone from the Intelligent Imaging department for giving me the possibility to expand my knowledge on convolutional neural networks and image processing. I have felt welcomed by the team, it was an enriching year for me.

I would like to thank Dr. Klamer Schutte for supervising and supporting my graduation process. He provided me with many opportunities to learn and ask critical questions, which have contributed to a successful conclusion of the project. Furthermore, I would like to thank Laurens Bliek for his help as a daily supervisor from the DCSC team. I appreciate his participation in our fortnightly discussions and his useful feedback on my work. Finally, I would like to thank Prof. Dr. Ir. Michel Verhaegen, Dr. Maarten Kruithof and Dr. Nanda van der Stap for the assistance during the process.

For remarks or questions concerning the research, please contact me via the following e-mail address: `irissnuverink@gmail.com`.

Delft, University of Technology — I.A.F. Snuverink
November 23, 2017

“Kop d’r veur!”

— *Marlies Snuverink*

Chapter 1

Introduction

The Extended Image Sensing Technologies (EXIST) program aims to research technologies for newly developed Complementary Metal Oxide Semiconductor (CMOS) image sensors. Those technologies are needed in the next generation of applications in digital lifestyle, food sorting, security and medical health care. The program is organized and funded by the European Commission [11]. Pixels in the newly developed sensor will have increased functionality, as number of pixels per chip keeps increasing and sensitivity is extended to infra-red. TNO focuses at developing technologies in high-end security and medical applications using hyperspectral (HS) images. This thesis is part of research on security applications.

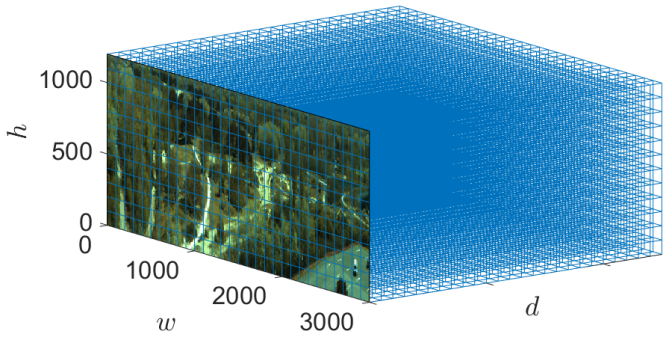

Figure 1-1: In HS imaging, the $(h \times w \times d)$ data cube contains information regarding d spectral bands for each pixel. The data set available for research contains images of size $(1280 \times 3033 \times 25)$.

In HS imaging, a spectrum of wavelengths is captured for every pixel. Regular Red-Green-Blue (RGB) cameras capture information in three bands in the visual wavelengths only, whereas HS cameras capture information in a larger number of smaller bands. An HS image can been considered as a $(h \times w \times d)$ data cube, in which d is the number of spectral bands (figure 1-1). The EXIST sensor is still under development. The sensor will capture HS information in 25 spectral bands, including the infra-red spectrum. TNO has recorded a data set containing HS images which are converted to simulate the EXIST sensor. Every image has size $(1200 \times 3033 \times 25)$. The use of HS data enables distinguishing objects using

reflectance information, as the data contains reflectance spectra for every pixel (figure 1-2). Each object has its characteristic spectrum, which is material dependent. This material dependent spectrum is called the spectral signature. The spectral signatures are used to classify the pixel content.

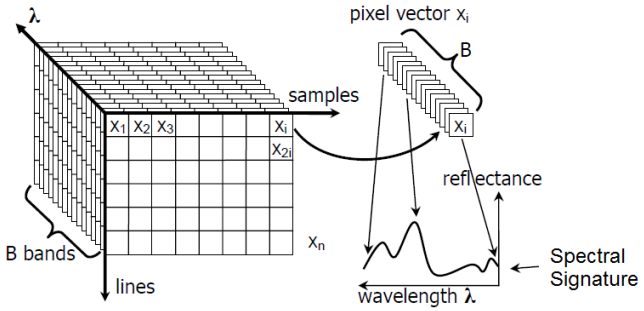

Figure 1-2: In the HS data cube, for every pixel a spectral signature is available. [1]

Pixelwise classification of HS images is based on information which is not limited to visual differentiation, as objects of similar color but different materials can be recognized. For security applications, it is essential to identify and localize objects within an image in order to recognize possible threats. Especially, distinguishing between man-made and natural objects is of great importance. For a computer to automatically recognize and identify content of all pixels in HS images, an algorithm to perform pixelwise classification is required. Pixelwise classification is also known as image segmentation or semantic segmentation.

The HS data set was intermittently recorded over the course of a year. The images within the data set display a fixed scene, showing vegetation, sand, asphalt and artificial grass. For at least one week in every season, every hour between sunrise and sunset an HS image was taken. Not only the range of lighting conditions cause segmentation to be challenging task, as the images are also subject to seasonal change and a range of weather conditions (figure 1-3). Consequently, the spectra of recorded HS data set are affected by atmospheric effects and illumination conditions. The atmospheric effects are caused by absorption of gases and water and scattering of molecules [12]. The spectral signatures are degraded by ambient conditions. HS data analysis traditionally aims at compensating for atmospheric effects and solar illumination by modeling atmospheric characteristics. Additional to the large internal variability of the data set, the dimensionality of the HS data is a challenge for segmentation.

1-1 Problem Statement

The goal of this research is to build a framework to perform pixelwise classification on HS images from the TNO data set, irrespective of recording date. This data set contains images subject to a wide range of ambient conditions:

1. **Illumination** Illumination conditions in this scene will vary over a day and throughout the entire year, affecting the reflectance spectra of all pixels. The goal is to achieve robustness to illumination variance.

2. **Weather** Atmospheric effects such as rain, sun, clouds and fog highly affect the reflectance spectra, complicating accurate classification.
3. **Seasonal appearance** Physical characteristics of the objects within the scene change over the seasons, such as color of grass and size of trees. Seasonal change affects material properties of vegetation, hence influences the spectral signature.

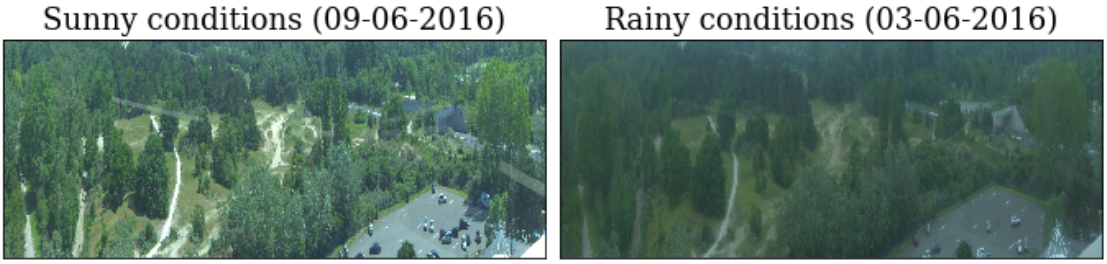

Figure 1-3: The scene displayed in the TNO data set under sunny and rainy weather conditions. These are RGB images used for visualization only.

In order to perform segmentation, a ground truth of the HS images is required. Making an adequate ground truth will be a preparatory, but essential step in HS image segmentation.

1-2 Research Approach

Large differences between the images from the data set are evident. Initial tests using traditional machine learning mechanisms, such as Support Vector Machines show that they are unable to cope with such a variability within the data set. Deep learning techniques are proposed to perform the segmentation task, as neural networks have proven to be an effective mechanism in complex image classification and segmentation problems [13] [6]. Neural network are known for its generalizing abilities. From training a network using pairs of unlabeled and labeled data, the network will eventually learn how to predict outcomes for new, unlabeled data.

Segmentation networks learn characteristic features which describe the classes present in the data, those features are abstract and not necessarily interpretable for human beings. The features will be based on both spectral and spatial data. A model should map spectral information of each spatial pixel to a single class. This mapping is not based on the spectral information only, it takes into account neighboring pixels as well. Deep learning techniques, such as Convolutional Neural Networks, are a promising method to perform segmentation on a HS data set exposed to varying ambient conditions. Classification is not based on spectral signatures only, spatial information about neighboring pixels is also incorporated to enhance performance.

This work will describe the implementation of a Fully Convolutional Neural network to perform pixelwise classification of HS images. Furthermore, a framework to form the required training data set using many small patches taken from the original images is proposed. This framework will also include a method to design a sparsely annotated ground truth for this data set. The neural network will predict pixel labels accurately, irrespective of the illumination and weather conditions of the image to be segmented. Furthermore the network should

be able to cope with changes in seasonal appearance of the scene. The goal of the thesis is expressed in two research questions and additional sub questions:

1. What is the best method for ground truth design and sample (pixel) selection to create a training data set which is representative of the HS images and yields high segmentation accuracy?

1. Is it possible to segment five classes (broad leaf tree, grass, sand, asphalt and artificial grass) using a single network?
2. Does a sparse mask as ground truth for multiple days yield accurate segmentation results?
3. In what way does the ground truth (e.g. level of detail) affect segmentation results?
4. Can patch-wise training be used in order to deal with the large HS image size?
5. Does balancing of classes in the training data set influence segmentation accuracy?

2. Can one use a convolutional neural network for HS image segmentation for a fixed scene with varying lighting, weather and seasonal conditions without correcting atmospheric effects in the recorded HS data?

1. What accuracy is achieved in training a network with simple training data set; data recorded over the course of one week under similar weather conditions. What are the generalizing properties of such a network, is it representative for data recorded beyond the training data set or under different weather conditions?
2. Does training a network with a more complex training data set (data recorded over the course of a couple of weeks under multiple weather conditions) increase generalizing properties regarding weather conditions?
3. Does training a network with a more complex training data set (data recorded over the course of a couple of weeks under similar weather conditions) increase generalizing properties regarding seasonal change?

1-3 Outline

Chapter 2 will focus on the working principle of neural networks and will discuss neural networks for image processing. Chapter 3 will explain the neural network setup used in the experiments associated with this thesis. Furthermore 4 will discuss what experiments were carried out in order to answer the research questions. In chapter 5 the results of the experiments belonging to the first research question are presented. It is aimed at giving general guidelines for composing a training data set. Chapter 6 shows the results concerning the second research question, it discusses image segmentation under varying ambient conditions. Consequently, the research questions are answered in chapter 7.

For readability, please read this thesis in a color version.

Chapter 2

Background and Related Work

This chapter is aimed at introducing the topic of fully convolutional neural networks, networks devoted to the pixelwise classification of images. Firstly, a brief introduction to artificial neural networks is given in section 2-1. Subsequently, the Convolutional Neural Network (CNN) will be discussed more thoroughly in section 2-2. This type of networks are designed for image processing specifically. Finally, the Fully Convolutional Network (FCN) is introduced in section 2-3.

2-1 Introduction to Artificial Neural Networks

Artificial neural networks *(or neural networks)* are inspired by the biological neural system. Neural networks consist of an input layer, hidden layers and an output layer. Cybenko [14] proved that in a neural network a single hidden layer containing a finite number of neurons is capable of approximating any continuous function to any desired precision.

Cybenkos Theorem Let σ be any continuous discriminatory function (e.g. sigmoidal). Then finite sums of the form:

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j) \tag{2-1}$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum $G(x)$, of the above form, for which:

$$|G(x) - f(x)| < \epsilon \qquad \forall x \in I_n \tag{2-2}$$

Therefore given a desired function $f(x)$, which is to be computed with certain accuracy $\epsilon > 0$, Cybenkos theorem states that when using enough hidden neurons, there is always a network with output $G(x)$ which satisfies $|G(x) - f(x)| < \epsilon$.

2-1-1 Neuron Model

In order to model a neural network, a mathematical neuron model is required. Originally the development of neural networks have been based on works that try to model the biological neural system. Neurons are the basic computational units in the brain. A neuron input is received from their dendrites, the produced output is sent from their axons (figure 2-1). A synapse is the transition between the axon of one neuron and dendrite of another.

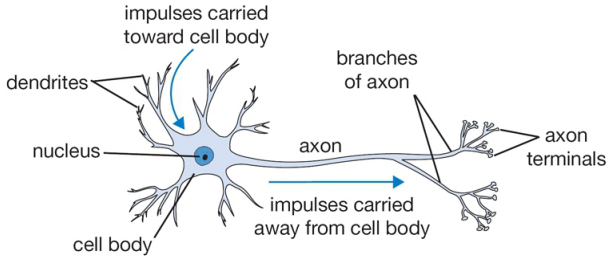

Figure 2-1: An example of a biological neuron. [2].

The synaptic strength determines the level of interaction between them. In the mathematical model these synaptic strengths are learnable, modeled with the so-called weights w_i. Dendrites transport the input signals x_i to the cell body, here the inputs are summed. When the total sum reaches a certain threshold, the neuron is able to produce an output using its axons. The output production rate of a neuron is modeled by an non-linear activation function $f(x)$. The activation of a neuron is modeled as shown in equation 2-3. This equation models decision-making, by varying the weights and activation function the output is affected.

$$a_i = f(\Sigma_i(w_i x_i) + b_i) \tag{2-3}$$

Here a_i represents activation of a neuron, $f(x)$ is the non-linear activation function, w_i is the synaptic strength weight, x_i is the neuron input and b is the neuron bias. Neural networks were originally modeled using sigmoidal and hyperbolic tangent functions, but Rectified Linear Unit (ReLU) functions have proven to be computationally more efficient.

2-1-2 From Neurons to Neural Networks

Neural networks are modeled as a structure of neurons connected in layers. The outputs of neurons of a certain layer become input of neurons of the next layer, these networks are called feed-forward neural networks. The input is always passed forward, there are no loops present. Therefore, neurons within a layer cannot be connected. Layers between the input and output layer are called hidden layers (figure 2-2). In the first layer simple decisions are made based on the input. However, already in the second layer complex decision are made based on decisions made in the first layer. Going deeper into the network more complex and abstract decisions appear. The word *deep* in the term *deep neural networks* refers to a network having multiple hidden layers.

In order to train the network to learn a general rule from the given examples (or training data), the weights w_i from equation 2-3 need to be adjusted. This process of adjusting and updating the weights for training data is called backpropagation. In section 2-2-4 the working principle of this process in explained.

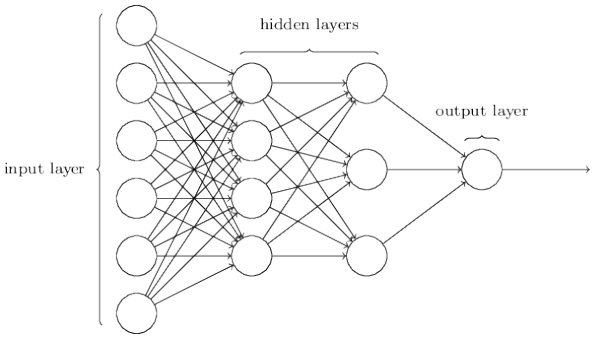

Figure 2-2: A network with two hidden layers. [3].

2-1-3 Introducing Convolutional Neural Networks

A CNN is a type of neural network which is able of extracting information out of images; data containing spatial information. This thesis will focus on the use of CNNs to solve the image segmentation problem. The image size of the recorded hyperspectral (HS) data is $1280 \times 3033 \times 25$. When using a neural network, a single fully connected neuron in the first hidden layer would have to optimize $1280 \cdot 3033 \cdot 25 = 9.7 \cdot 10^7$ weights. The full connectivity of each neuron results in a vast amount of weights to be optimized and will increase the chance of overfitting the training data.

A CNN constrains the image input by its architecture, as will be explained in section 2-2. The neurons in CNNs are arranged in three dimensions; width w, height h and depth d. Depth usually refers to the depth of the input; in case of the first network layer for HS images it is the number of spectral bands. Deeper into the network it refers to the number of feature maps of the convolutional layer input. Neurons in a layer will only be connected to a small region of the neurons in the layer before, instead of a fully connectivity to all neurons. This property decreases the number of weights compared to a regular neural network.

2-1-4 Image Classification; A Brief Timeline

The goal of this thesis is to identify predefined classes in every pixel of a spectral image of a fixed scene. For segmentation, an image is fed into a neural network after which the network outputs a segmented map consisting of coherent class regions. First *Image classification* is explored, which is a more mature field of study. Classification aims labeling entire images, rather than labeling pixels within an image.

LeCun et al. successfully applied CNNs in 1998 [15], it was the first step towards developing CNNs for complex tasks. Krizhevsky et al. improved the CNN architecture in 2012, it caused CNNs to regain attention. Consequently, deep learning for computer vision tasks has gained interest and popularity.

LeNet The first successful neural network that was build is LeNet-5 [15]. This network was capable of reading handwritten zip codes and digits.

AlexNet The architecture of AlexNet [13] is relatively simple. It is very similar to LeNet, but deeper and bigger. The network was trained on data from the ImageNet database in order to

participate in the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), which they won by a large margin. The non-linear activation functions $f(x)$ that were used are ReLU instead of the hyperbolic tangent or sigmoid function (equations 2-4 and 2-5 respectively).

$$f(x) = \tanh(x) \tag{2-4}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2-5}$$

Using ReLU significantly speeds up the convergence of the stochastic gradient descent, as it does not require computation of for example exponentials. The ReLU nonlinear activation function is given in equation 2-6 and displayed in figure 2-5.

$$f(x) = \max(0, x) \tag{2-6}$$

Furthermore data augmentation techniques were used and drop-out layers were introduced to reduce overfitting of the data.

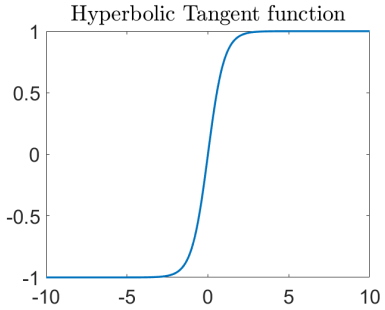

Figure 2-3: Hyperbolic tangent, range [-1, 1]

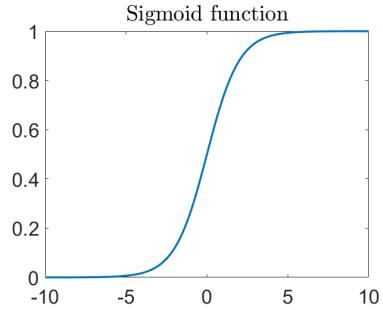

Figure 2-4: The sigmoid function, range [0,1].

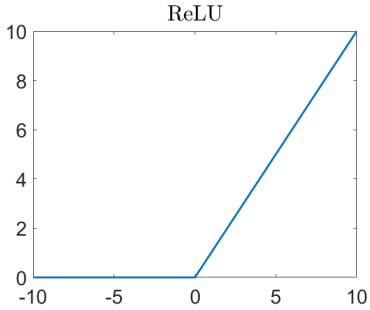

Figure 2-5: The activation threshold at zero.

GoogLeNet Google presented the inception module in 2015, introducing a complex network called GoogLeNet [16]. This network won the ILSVRC in 2014 and used a new approach to designing CNN architectures.

VGGNet This network is build by the Visual Geometry Group of the University of Oxford [17]. The network achieved the second place in the ILSVRC 2014 challenge. Simonyan and Zisserman showed the importance of choosing a suitable depth of a network.

Microsoft ResNet The ILSVRC 2015 was won by Microsoft Research Asia, participating with their ResNet architecture [18]. They achieved an error rate of 3.6%, performing better than humans, as human performance have an error rate around $5-10\%$. The network contains 152 layers.

Image classification was the first step toward more complex tasks, such as image segmentation and instance-aware image segmentation. Section 2-3 describes *Fully Convolutional Networks* which are used to perform image segmentation.

2-2 Convolutional Neural Networks

A CNN is a type of deep neural network. As already mentioned in section 2-1-2, neural networks consist of many artificial neurons, which are the core of the algorithm intelligence.

The development of CNNs has grown rapidly over the past five years. Companies such as Facebook use them for photo tagging algorithms and Google applies it in image search engines.

2-2-1 Convolutional Layer

Images are represented by matrices containing color information in the form of Red-Green-Blue (RGB) color codes. An image therefore has size $h \times w \times d$, where color channel depth $d = 3$. Convolutional layers are essential layers in CNNs, producing feature maps from input images or lower level feature maps.

Convolutional layers includes a kernel (or filter). Let K be a kernel with x rows, y columns and depth d. Then the kernel with size $(K_x \times K_y \times d)$ works on a receptive field $(K_x \times K_y)$ on the image. The kernel height and width are smaller than the input image height and width. The kernel slides over (convolves with) the image, producing an feature map (figure 2-6). Convolution is the sum of the element-wise multiplication of the kernel and the original image. Note that the depth d of the kernel is equal to the depth of its input. Therefore, it varies within the network. Usually the depth of an image is the number of color channels, the three RGB channels. However, in HS images the depth is the number of spectral bands.

The kernel stride is a free parameter in convolutional layers which has to be defined before training. The stride is the number of pixels by which the kernel shifts at a time. A drawback of using convolutional layers is that it decreases the output map size. A larger stride will result in a smaller sized output. Equations 2-7 show the relationship between output size O and input size of an image I after convolution with stride s and kernel K. Furthermore, the feature map size decreases as the number of convolutional layers increases. Row output size O_x and column output size O_y of convolutional layers are determined as follows:

$$\begin{aligned} O_x &= \frac{I_x - K_x}{s} + 1, \\ O_y &= \frac{I_y - K_y}{s} + 1 \end{aligned} \tag{2-7}$$

As an example, an image of size $(32 \times 32 \times 3)$, a kernel of size $(3 \times 3 \times 3)$ and a stride $s = 1$ result in an activation map of size $(30 \times 30 \times 1)$. Using additional n kernels, the activation map becomes $(30 \times 30 \times n)$. So, additional kernels will increase the depth of the convolutional layer output.

After each convolutional layer, the ReLU non-linearity is introduced. This step is an elementwise operation, so every pixel from the activation map will be exposed to this nonlinear operation. Every negative value will be replaced with zero (figure 2-5). After this nonlinear layer, the network proceeds with another layer, for example a new convolutional layer or a pooling layer.

2-2-2 Pooling Layer

Pooling layers are also known as downsampling layers. A commonly used pooling method is maxpooling (figure 2-7). The downsampled output is produced by taking the maximum input value within the kernel, resulting in an output of decreased size. There are several

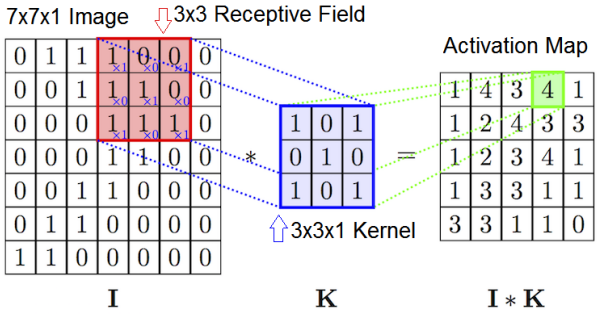

Figure 2-6: Convolution of image I with kernel K and stride 1, the weights in the kernel are the parameters to be trained. This is an explanatory example with one color channel. [4]

other methods which are commonly used in neural networks, such as average pooling and L2-norm pooling. A pooling layer has a kernel and a stride of similar length. Two important arguments of implementing pooling layers are decreasing the number of weights and decreasing the chance of overfitting the training data.

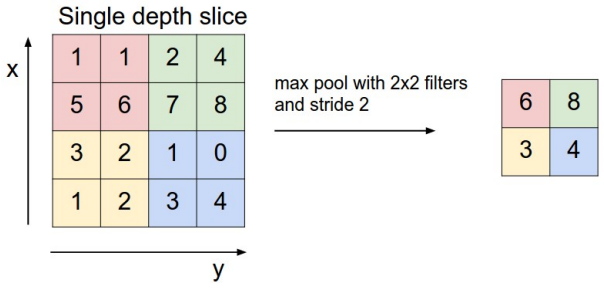

Figure 2-7: Maxpooling with a (2×2) kernel and stride $s = 2$. Maxpooling layers reduce spatial dimension the input [2].

2-2-3 Overfitting

Overfitting is a problem that arises in neural network training. When a model is overfitted to the training data, it loses its capability of generalization. The model has learned the training data, including noise, in such a great extend that it has failed to capture underlying general information. CNNs have a large number of weights to be trained, therefore overfitting can occur due to training too few training examples. Dropout layers are a tool to prevent overfitting (figure 2-8). In dropout, nodes and its connections are randomly dropped from the network. Dropout constrains the network adaptation to the training set, consequently it prevents that the weights are not too much fitted this data. The difference in performance between training data and validation data will decrease. Dropout layers are used during training only, not during validation or testing.

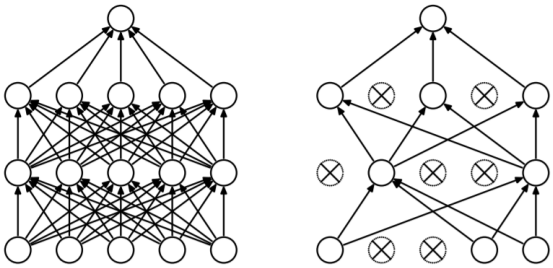

Figure 2-8: A neural network structure before and after applying dropout. [5]

2-2-4 Backpropagation

The CNN requires to adjust and update its kernel parameters, or weights, for the given training data. Backpropagation is an efficient method for computing gradients required to perform gradient-based optimization of the weights in neural networks [19]. The specific combination of weights which minimize the loss function (or error function) is the solution of the optimization problem. The method requires the computation of the gradient of the error function at each iteration, therefore the loss function should be both continue and differentiable at all iteration steps.

The initial weights of an untrained CNN are randomly chosen. Consequently before training, the neural network cannot make meaningful predictions for network input, as there is no relation between an image and the its labeled output yet. By exposing the network to a training data set, comprising images and their labeled outputs with correct classes, the weights are adjusted. Training is the adaptation of the weights in such way that the difference between desired output and network output is minimized, which means that the network is trained to find the right features required for classification. There are two computational phases in a neural network, the forward pass and the backward pass in which the weights are adapted.

Forward pass An image is fed into a network. The first network layer outputs an activation map. Then, this activation map is the input to the first hidden layer, which computes another activation map. Using the values of this activation map as inputs to the second hidden layer, again another activation map is computed. Carrying out this process for every layer will eventually yield the network output.

Backward pass In this phase the weights are updated by backpropagation. One epoch of backpropagation consists of multiple parts, usually multiple epochs are carried out for a training image:
1. Loss function In forward pass, the inputs and desired outputs are presented. A predefined loss function L is used to minimize the difference between the input and desired output. The goal is to adjust the weights so that the loss function value decreases, this is achieved by calculating the derivative with respect to the weights of the loss function.
2. Backward pass During the backward pass, the weights that have contributed the most to the loss are determined in order to adjust them so that the total loss decreases.
3. Weight update In the final part all weights are updated in the negative direction of the loss function gradient.

Therefore the core of the backpropagation problem is to compute the gradient of the loss function with respect to the network weights. Computing the partial derivative $\frac{\partial L}{\partial w}$ is essential

(carried out in the backward pass) to minimize the loss function value. Stochastic Gradient Descent (SGD) is the most common way to optimize neural networks.

Backpropagation Example for a Multi-Layer Network

A simple example of computation of weight updates in backpropagation will be given using the network shown in figure 2-9. The cost function L is given below, e_l is the error between

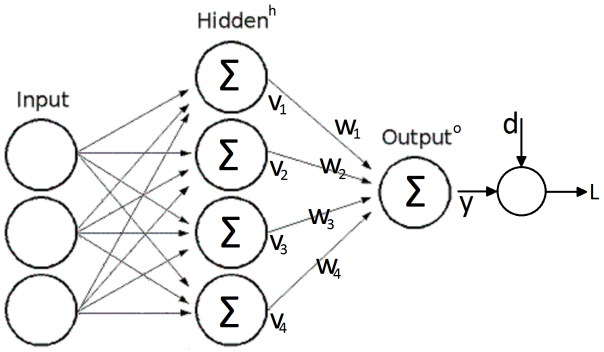

Figure 2-9: A simple multilayer network.

the desired output d_l and network output y_l. The network output y_l is computed in the forward pass and depends on outputs of the previous layer v_j and the output layer weights w_j^o.

$$\begin{aligned} L &= \frac{1}{2}\sum_l (e_l)^2 \\ e_l &= d_l - y_l \\ y_l &= \sum_j w_j^o v_j \end{aligned} \tag{2-8}$$

The Jacobian is given by:

$$\frac{\partial L}{\partial w_{jl}^o} = \frac{\partial L}{\partial e_l}\frac{\partial e_l}{\partial y_l}\frac{\partial y_l}{\partial w_{jl}^o} \tag{2-9}$$

Calculating the partial derivatives yields the following Jacobian for the output layer:

$$\frac{\partial L}{\partial w_{jl}^o} = -v_j e_l \tag{2-10}$$

Using the SGD update rule, which will be explained in section 2-2-6, the output weights are updated using:

$$w_{jl}^o(n+1) = w_{jl}^o(n) + \alpha(n) v_j e_l \tag{2-11}$$

After having updated the output weights, the weights in the hidden layers can be updated. As it is a backward pass, first gradients of the output layers are computed, then the gradients of the hidden layers. The Jacobian is given by:

$$\frac{\partial L}{\partial w_{ij}^h} = \frac{\partial L}{\partial v_j}\frac{\partial v_j}{\partial y_l}\frac{\partial z_j}{\partial w_{ij}^h} \tag{2-12}$$

Calculating the partial derivatives yields:

$$\frac{\partial L}{\partial w_{ij}^h} = -x_i \sigma_j'(z_j) \sum_l e_l w_{jl}^o \tag{2-13}$$

Which yields the update rule for the hidden layers:

$$w_{ij}^h(n+1) = w_{ij}^h(n) + \alpha(n) x_i \sigma_j'(z_j) \sum_l e_l w_{jl}^o \tag{2-14}$$

Finally the network is tested using a test dataset, this dataset contains images that differ from the ones in the training dataset. By increasing the amount of training data, the more training iterations are carried out, the better the weights are tuned.

2-2-5 Loss Function

The value of the loss function L represents the difference between the training image after it has propagated through the network and desired annotated output image.

Two assumptions are made about this loss function. First it should be able to define the loss function as the average over the loss functions for individual training images, as the training often is carried out in batches. The loss function is evaluated and average at the end of each batch, then the weights are updated. Secondly, the loss function should be able to be defined as a function of the network outputs. Below a brief overview is given of some widely used loss functions, where x_i are the neuron outputs and $\hat{x}_i$ are the desired outputs.

Quadratic Cost Function The Mean Squared Error (MSE) cost function is one of the simplest cost functions.

$$L = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 \tag{2-15}$$

Cross Entropy Cost Function The cross entropy cost function is commonly used in convolutional network applications.

$$L = \frac{1}{N} \sum_{i=1}^{N} (\hat{x}_i \ln(x_i) + (1 - \hat{x}_i) \ln(1 - x_i)) \tag{2-16}$$

Exponential Cost Function The exponential cost function requires an additional parameter τ.

$$L = \frac{1}{N} \tau \exp \frac{1}{\tau} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2. \tag{2-17}$$

2-2-6 Stochastic Gradient Descent Variants

In both Gradient Descent (GD) and Stochastic Gradient Descent (SGD) parameters are updated according to an update rule to minimize a loss function in an iterative manner. Computing the exact gradient using GD in large datasets is expensive (GD is deterministic),

as this method runs through all training samples to perform a single update for one iteration step. In Stochastic Gradient Descent (or *on-line* Gradient Descent) an approximation of the true gradient is computed. This is done by using only one or a subset of training samples for a parameter update. When using a subset of training samples, this method is called mini-batch SGD.

SGD is a method to minimize the loss function $L(\theta)$ parametrized by θ. This is achieved by updating the parameters θ in the negative gradient direction of the loss function $\nabla_\theta L(\theta)$ with respect to the parameters, in order to decrease the loss function value. The learning rate η determines the step size to get to the local or global minimum. The update rule is given in equation 2-18.

$$\theta = \theta - \eta \nabla_\theta L(\theta) \tag{2-18}$$

Mini-batch Stochastic Gradient Descent This method performs an update for every mini-batch of n training samples. Mini-batch SGD reduces the variance of the parameter updates. Larger mini-batches reduce the variance of SGD updates by taking the average of the gradients in the mini batch. This allows taking bigger step sizes. In the limit, if each batch contains one training sample, it is the same as regular SGD.

2-2-7 Learning Rate Scheduling in Gradient Descent Optimization

There are several variants of SGD available. Determining the appropriate learning rate, or step size, often is a complex problem. Applying too high learning rates causes suboptimal performance, too low learning rates causes slow convergence. Learning rate scheduling is used as an extension of the SGD algorithm to improve performance. In learning rate scheduling, the learning rate is a decreasing function of the iteration number. Therefore, first iterations have larger learning rates and consequently cause bigger parameter changes. Later iterations have smaller learning rates, responsible for fine-tuning. Below an overview of some gradient descent optimization algorithms is given.

Momentum Momentum is a method to speed up the SGD is the relevant direction. A fraction γ of the previous update is added to the current update. The Momentum update rule is give in equation 2-19.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta L(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{2-19}$$

Nesterov Accelerated Gradient The Momentum method does not take into account direction it is going in, the Nesterov Accelerated Gradient method computes an approximation of the next position of the parameters. The update rule is given in 2-20.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta L(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{2-20}$$

Adagrad The Adagrad [20] method adapts the updates to the slope of the error function. The algorithm adapts the learning rate to the parameters, so that size of the updates for each parameter depends on its importance. The Adagrad algorithm gives larger updates for infrequent parameters and smaller updates for frequent parameters, the update rule in given in equation 2-21. Here G_t is a diagonal matrix containing the sum of squares of past gradients

with respect to θ. The main advantage of Adagrad is that one does not need to manually tune the learning rate. However Adagrad faces problems due to the accumulation of squared gradients in the denominator, since every additional term is positive, the accumulated sum keeps growing. Therefore the learning rate becomes smaller as training progresses.

$$\begin{aligned} g_{t,i} &= \nabla_\theta L(\theta_i) \\ \theta_{t+1,i} &= \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \end{aligned} \tag{2-21}$$

Adadelta Adadelta [21] is an extended version of Adagrad which reduces the problem of the decreasing learning rate. It restricts the range of accumulated squared gradients to a certain fixed size.

RMSprop RMSprop [22] is also an adaptive learning rate method that tackles the problem of the accumulation of squared gradients in Adagrad. RMSprop divides the learning rate by an exponentially decaying average of squared gradients. It is an unpublished algorithm by G. Hinton.

Adam The Adaptive Moment Estimation (Adam) optimizer [23] also determines an adaptive learning rate for each parameter. Adadelta and RMSprop store an exponentially decaying average of past squared gradients v_t, but Adam also keeps an exponentially decaying average of past gradients m_t. Vectors v_t and m_t are estimates of the mean and the uncentered variance of the gradients respectively which are biased towards zero. Bias-corrected estimates $\hat{v}_t$ and $\hat{m}_t$ are computed for the update rule (equation 2-22).

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{2-22}$$

2-2-8 Hyperparameter Tuning

Training a network is done by a process called backpropagation, as explained in section 2-2-4. Within the process of training, there are many choices regarding learning rate, optimizer, dropout and batch size. Batch size is number of training examples that is used in one epoch, the higher the batch size, the more memory space is required. These are examples of *hyperparameters*, network settings which have to be defined before training. Furthermore, there are hyperparameters regarding network architecture, such as convolutional kernel size, non-linear activation functions, loss function and number of network layers. Hyperparameter choice is highly dependent on the size and type of the training dataset. There are no strict rules on how to choose the parameters, only general guidelines.

In deep learning the loss function value is tracked per epoch. One epoch is one forward plus backward pass during training. By monitoring loss during training, one is able to observe if an appropriate learning rate is chosen. High learning rates will initially decrease the loss faster, but will eventually yield at a suboptimal place in the energy landscape. Furthermore the degree of overfitting can be seen from the difference in accuracy between training and validation.

Hyperparameter tuning is an important but complex part of neural network training. Choosing correct settings is essential in obtaining desired results, however hyperparameter tuning is

often based on experience rather than theoretical knowledge. Especially choosing such a large number of different settings which are mutually connected causes it to be a non-transparent process. Moreover, trade-offs are inherent in the parameter selection process, for instance computer memory is restricted.

2-2-9 Patch-wise Training

Image size for training is limited by computer memory and Graphics Processing Unit (GPU). The images in the TNO data set have extremely large dimensions, so that the full images cannot be used for training. Training a neural network in a patch-wise manner is a way to deal with the large amount of data, in this method many small patches taken from the original HS images form the training data set. After training the network, an original sized test image is fed into the network to yield a segmentation.

In remote sensing applications, aerial and satellite images have large dimensions, comparable to the image size from the TNO dataset. In case of segmentation using neural networks, training is also done using patches of the original image. Marmanis et al [24] have randomly sampled patches from a training subset for training the network weights. Maggiori et al [25] have also used an FCN network for segmentation of satellite images. Volpi and Tuia [26] have trained a CNN patch-wise, they selected patches large enough to include spatial context and small enough to meet computer memory requirements.

In patch-wise training the global scene context is not used. The scene composition remains the same over all images, for example the parking area, sandy tracks and tree locations will not move over time. As this information will not be presented during training, these global spatial features are not learned. Therefore, pixelwise predictions will only be based on features about spatial structure covering an area as large as the patches. Patch-wise trained networks have larger generalizing properties in the sense they can be applied on scenes with slightly different spatial structure.

2-3 Fully Convolutional Networks

Classification of an image is just the first step towards more complex tasks, such as the detection of objects within an image and assigning captions to images describing the scene. Image segmentation is the pixel-wise classification of an image, thus labeling every single pixel with a class. First, the FCN is introduced in section 2-3-1. Next, the deconvolutional layer is discussed in section 2-3-2. In section 2-3-3 several types of FCNs are explained. Finally, section 2-3-4 discusses the network architecture which is used throughout all experiments of this thesis. *Image segmentation is also referred to as semantic segmentation, pixelwise segmentation or pixelwise classification.*

2-3-1 Introducing Fully Convolutional Networks

Long et al [6] introduced the FCN for image segmentation in 2015. This network was designed and trained for pixel-wise segmentation. FCN are modified classification CNNs. In order to make a network suitable for pixel-wise segmentation, certain layers have been modified which

enable the generation of segmented output maps. The idea of using a modified CNN to use for pixel-wise segmentation is not new. Matan et al [27] modified the LeNet network for the purpose of recognizing strings of digits. More recent publications show results on segmentation using CNN for making dense predictions, however these methods are limited by e.g. post-processing.

In order to build FCNs, the fully connected layers in the CNNs are replaced by a convolutional layers. Therefore an FCN is a classification network without any fully connected layers. The goal of an FCN is to capture image context; what objects are located where. Architectures of neural networks for image segmentation inherently includes the trade-off on how to process both coarse, global information and detailed, local information.

Regular pre-trained classification networks actually do contain information about location of classes, however it is hidden by the fully connected final layer for the single classification output (figure 2-10). When a pre-trained classification network has been transformed into a fully convolutional network, it is able of predicting locations per class.

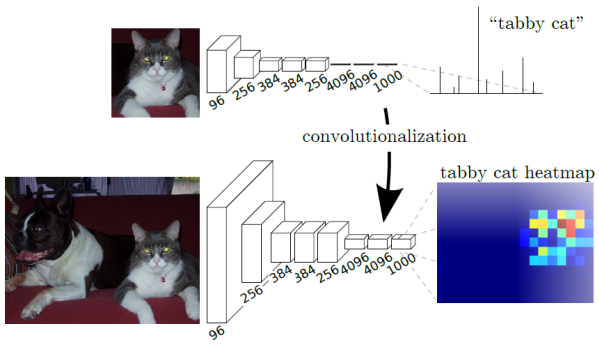

Figure 2-10: Transforming a classification network into a fully convolutional (segmentation) network shows that classification networks contain information about location. [6]

Zeiler and Fergus [7] propose a visualization method to give insight into the CNN performance. This visualization technique was introduced to show the contribution of each layer to the final classification of an image. Each layer provides a feature map that shows what the layer responds to. This work clearly shows the hierarchical nature of convolutional networks. The first layer responds to low level features, such as color, lines and edges, the intermediate levels react to more complex features, such as textures, and the final layers react to class specific information. The lower level layers converge relatively quickly compared to the final layers, just after a few epochs. The final layers require time to develop, which shows the importance of convergence of the model to obtain high performance.

Figure 2-11 and 2-12 show feature maps after being processed by the second and third network layer respectively. The figures show the top nine activations for a given feature map and corresponding image patches. This example shows the evolution of the features through the network, from general and simple to complex and class-specific features. Figure 2-11 clearly shows low level features, such as colors, lines and edges. Figure 2-12 displays more complex features such as texture and shape.

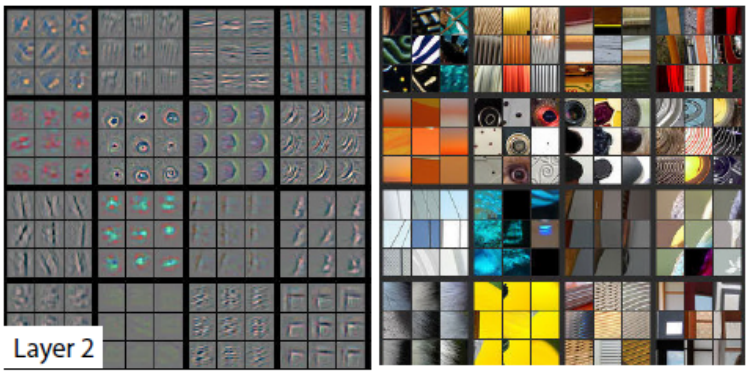

Figure 2-11: Feature map of the second layer, showing low level features [7]

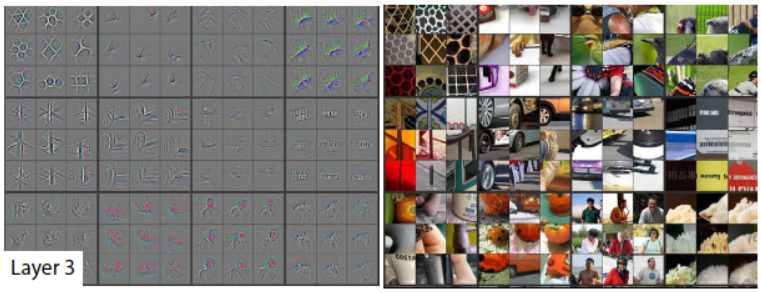

Figure 2-12: Feature map of the third layer, showing more complex features [7]

2-3-2 Deconvolutional Layer

A deconvolutional layer in a neural network is a layer which is able to obtain a dense map from downsampled and course input [6]. A more appropriate name is the *transposed convolutional layer*, as the term *deconvolution* may be misleading as deconvolutional layers also perform convolutions.

Pooling layers in convolutional networks are required in order to decrease the number of network parameters. Unpooling layers perform the reverse of pooling layers (figure 2-13). The location of the maximum activation in the pooling layer is recorded in switch variables [8], in the unpooling layer it is placed back. The output of such an unpooling layer is sparse, as it is an enlarged version of the input map. The deconvolution layer then produces a dense output map from the unpooling layer output. In summary, convolutional layers map multiple activations in a receptive field to a single activation, deconvolutional layers map one single activation to a field or window of multiple activations. Convolutional layers learn a filter which map a $K_x \times K_y$ receptive field to one value, consequently deconvolutional layers learn filters that perform the opposite.

2-3-3 Types of Fully Convolutional Networks

The FCN-type networks have dominated online image segmentation challenges since the release of the paper by Long et al. The original FCNs have a drawback of generating lower resolution output predictions, due to pooling layers and striding in convolutional layers. New

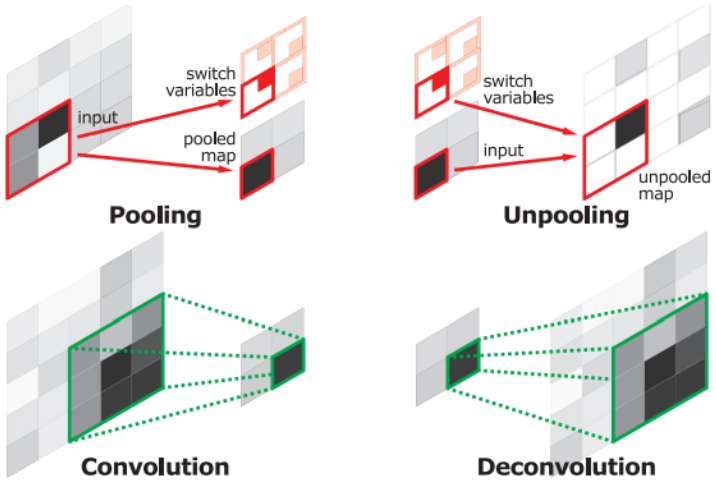

Figure 2-13: Schematic representation of pooling, unpooling, convolution and deconvolution [8].

network architectures have been developed that tackle the output resolution problem. Four types of FCN are discussed (figures 2-14 to 2-17), these types all have a different approach to collecting both global and contextual features.

- Image pyramid
- Encoder-decoder
- Spatial pyramid pooling
- Atrous convolutions

These network types are aimed at collecting global features or contextual information in order to increase segmentation performance. Furthermore, these methods try to recover objects at multiple scales in the final segmented output. Methods combining a neural network and Conditional Random Field (CRF) have not been taken into account.

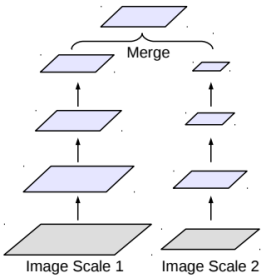

Figure 2-14: Image pyramid [9].

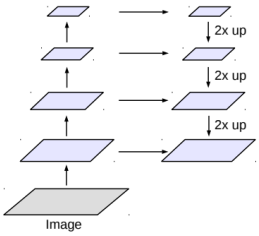

Figure 2-15: Encoder-decoder [9].

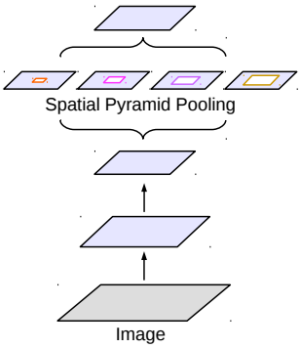

Figure 2-16: Spatial pyramid Pooling [9].

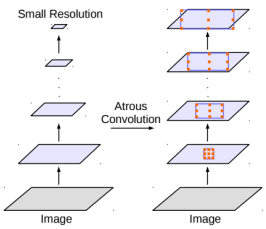

Figure 2-17: Atrous convolutions [9].

Image pyramid

This network combines multi-scale inputs (figure 2-14). Small scale inputs account for contextual information while large scale inputs provides details. The approach is as follows; an

image at multiple scales is fed trough a network in order to merge feature maps of all scales. A major drawback is that this network is not suitable for deeper and larger architectures, due to GPU memory.

Encoder-decoder
This type of network consists of two parts, the encoder and the decoder (figure 2-15). The encoder part reduces spatial dimensions of features maps. The decoder part recovers details and spatial dimensions. Examples of encoder-decoder networks are SegNet [28], U-Net [10] and RefineNet [29]. In order to increase resolution, SegNet uses pooling indices from the encoder layers to learn upsampling in the decoder layer. U-Net makes uses of skip connections between encoder and decoder feature maps to increase resolution.

Spatial pyramid pooling
This type of networks use spatial pyramid pooling to capture context at multiple levels of detail (figure 2-16). Examples of such networks are ParseNet [30] and PSPNet [31].

Atrous convolutions
This method a new approach to image segmentation. It contains no deconvolution layers, instead it uses atrous convolutions to recover spatial resolution [9].

2-3-4 Encoder-Decoder Architecture: U-Net

For biomedical image segmentation purposes, Ronneberger et al [10] have designed a network called U-Net. This network won the ISBI cell tracking challenge in 2015. The network does not include the usual stacking of layers. A different approach is taken, the network consists of an encoder path followed by a symmetric decoder path (figure 2-18). The name *U-net* refers to the U-shape in which the layers are sequenced. The idea behind this architecture is to combine lower and higher level feature maps through skip connections, which will improve localization of high resolution features. This network is build upon the FCN architecture proposed by

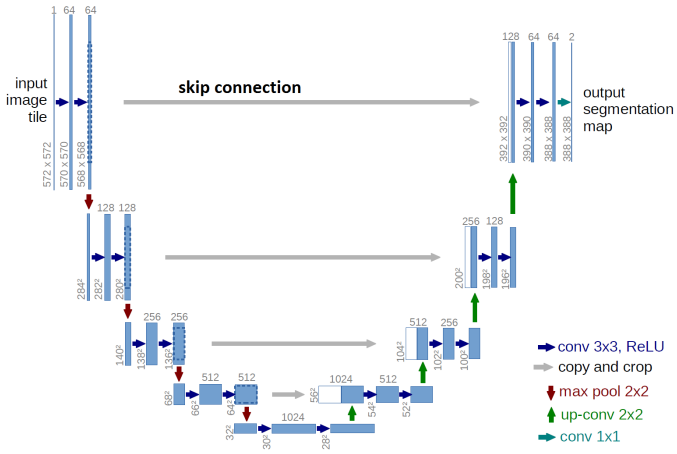

Figure 2-18: The U-Net architecture [10].

Long et al. The network is designed so that it is able to obtain accurate segmentation using only a few training images. Deconvolutional layers increase the final resolution of the output map. In the decoder path, a large number of feature maps per convolutional layer is applied. This will propagate contextual information to higher resolution layers.

In this thesis, all experiments are carried out with a network based on U-Net (section 3-3). The U-Net architecture is chosen to perform pixel-wise classification on the HS images, as it has proven to be an effective network for segmenting a single class in multi-spectral satellite data [32]. Furthermore, this network can be trained from scratch, as it does not rely on pre-trained networks. U-Net was trained used biomedical images of neuronal structures only. Other encoder-decoder architectures depend on the use of pre-trained networks, these are trained with training data sets comprising RGB images. SegNet uses a pre-trained VGG16-network [17] in the encoder part. RefineNet makes use of pre-trained ResNet [18]. Therefore, these networks have learned to extract features based on color information. For a network to effectively perform HS image segmentation, features should be able to extract HS information. Due to the clear network structure, it is possible to quickly alter existing layers and design custom-made layers to fit segmentation goals. For example batch normalization layers, convolutional layers or entirely new layers can be added. Also the non-linear activation function in the final layer can be adapted, as well as the network depth and width. Moreover, no fixed input size required for training this network.

Chapter 3

Neural Network Setup

Obtaining segmented images requires several steps regarding the training data set selection framework and network architecture design. This chapter starts with a detailed overview of design steps required to start training a neural network (section 3-1). Thereupon each step is discussed in greater detail in the subsequent sections. All choices regarding architecture and training data set parameters will be clarified, as well as details about recording the hyperspectral (HS) data set.

3-1 Overview of Design Steps

An overview of all design steps is displayed in figure 3-1. This overview shows that the process can be divided into four parts:

- Recording and preprocessing of the HS data set
- Setting up the network and the training, validation and test data set
- Network training and inferencing
- Converting network output to segmented images

This chapter will mainly focus on discussing the steps taken in setting up the network.

3-2 Data Set

The TNO dataset was recorded using a line scanning camera. Data recording took place over the course of several months in 2016. The images show a fixed scene, a dune area with natural (vegetation) and man-made objects particularly. The 2D scanning camera (consists of a light distributing prism-grating-prism and a grayscale CCD camera without NIR filter) rotates horizontally with fixed angle step. At each step a *line-image* was recorded, resulting in a field of view of 90 degrees. Every line-image contains HS information in wavelength ranging from 432 to 902nm. This spectrum is divided into 1024 spectral bins, yielding an image of

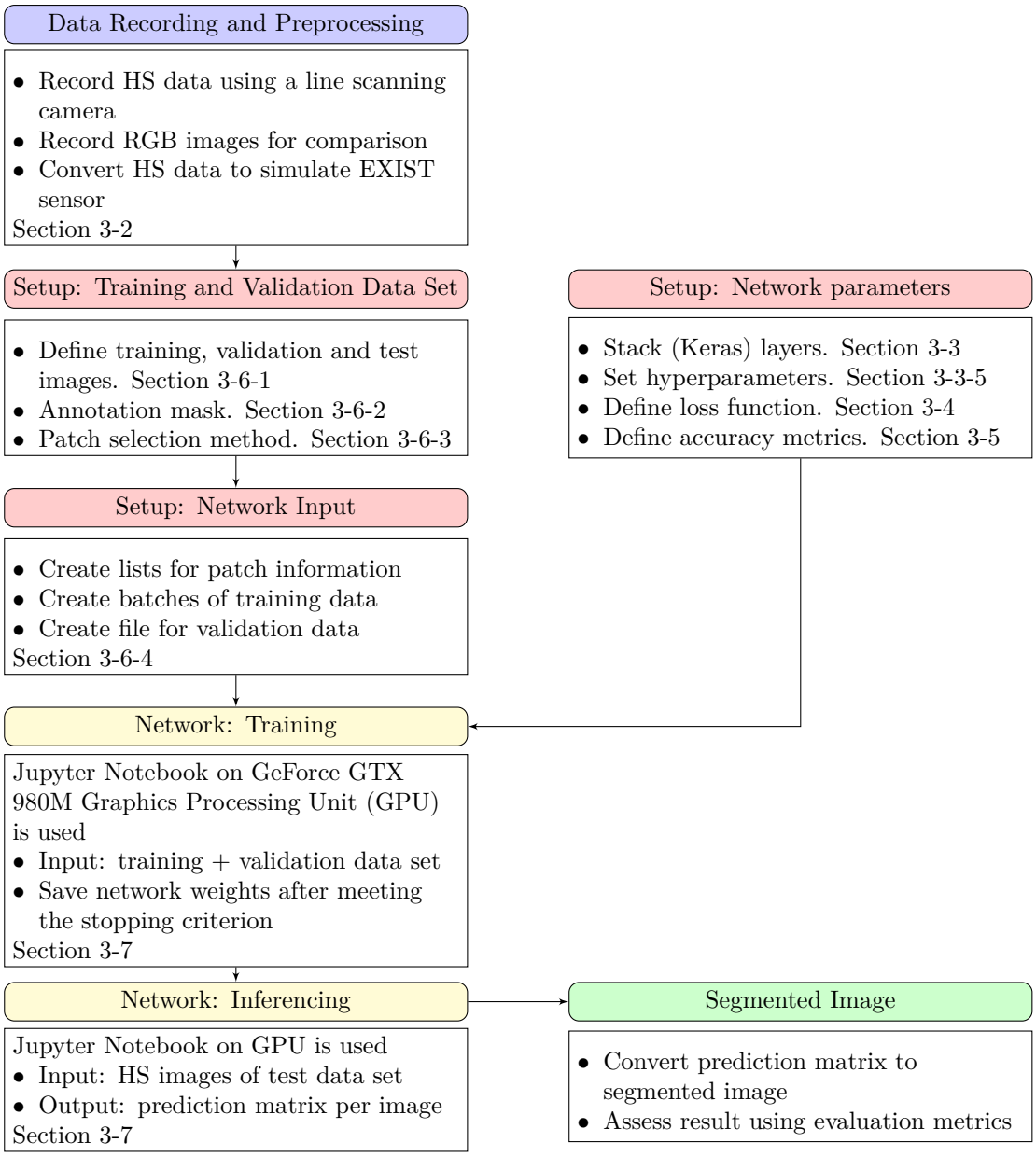

Figure 3-1: Overview of framework to perform pixelwise classification on HSI data.

size 1024×1280, in which 1280 represents a fixed height of the image. For every HS image 3033 line-images were recorded, covering the total field of view. This results in a HS cube of $3033 \times 1280 \times 1024$.

The EXIST sensor is still under development [33]. The EXIST sensor data output will differ from the sensor used for recording. Therefore the recorded data was converted to simulate the EXIST sensor. The EXIST sensor will only capture wavelengths in a range from 600 to 900nm, divided in 25 HS bands. Data outside this range was not used. The remaining data was re-binned into 25 bands, by taking the sum (due to summation of photons) of all bands from the original data within the new bin. Furthermore, the EXIST sensor will have

a different sensitivity for each HS band, therefore each band was multiplied with a sensor specific response curve delivered by the manufacturer. Finally, the resulting simulated data was used for training a neural network.

The data set was recorded during multiple recording weeks, covering all seasons. Scans were made in April, May, June, July, August, October and December 2016. The HS images were recorded at least during one week per month. Scans were made every hour between sunrise and sunset to capture all lighting conditions depending on the sun's position and weather conditions. Consequently the data set contains scans with a large range of weather types, such as sunny, rainy and foggy conditions.

3-3 Model Architecture

The neural network was designed using Keras [34]. Keras is a neural networks Application Programming Interface (API) written in Python, it runs on top of either TensorFlow, Theano or Microsoft Cognitive Toolkit (CNTK), which are software libraries for machine learning. The network was designed by stacking network layers on top of each other, as each layer type has its own function in Keras.

Figure 3-2 shows the U-Net architecture which was used for training (code in appendix B-1), this network is based on the U-Net by Ronneberger [10]. The network shows five contracting stacks of layers, the first block includes an input layer with a depth 25 channels, corresponding to 25 HS bands. This input layer is followed by a batch normalization layer, normalizing images within a batch, which is discussed in section 3-3-1. This architecture includes $7,853,142$ weights for an input size of $(25 \times 64 \times 80)$.

Five contracting stacks of layers are followed by stacks of expansive layers. These stacks start by merging an activation map from the contracting path with an activation map from the expansive path, by skip connections. Merging is followed by a dropout layer, two convolutional layers and an upsampling layer. The maxpooling and upsampling layers have (2×2) kernels, with a stride of one. All convolutional layers have kernels of size $(3 \times 3 \times d)$, except for the final convolutional layer, this kernel is (1×1). Furthermore are all convolutional layers followed by a ELU nonlinear activation function, except for the final convolutional layer. This layer is followed by a Softmax activation function, as discussed in section 3-3-3.

3-3-1 Batch Normalization Layer

Network convergence is improved by whitening the inputs, which means the input data is linearly transformed to have zero mean, unit variances and decorrelated properties [35]. Normalization by the batch normalization layer is carried out in the first layer of the network. This layer contains trainable weights, so that normalization becomes part of the model architecture [36]. Normalization is performed on each mini-batch during training. Per batch of input x, normalization is performed as shown in equation 3-1.

$$\hat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} \tag{3-1}$$

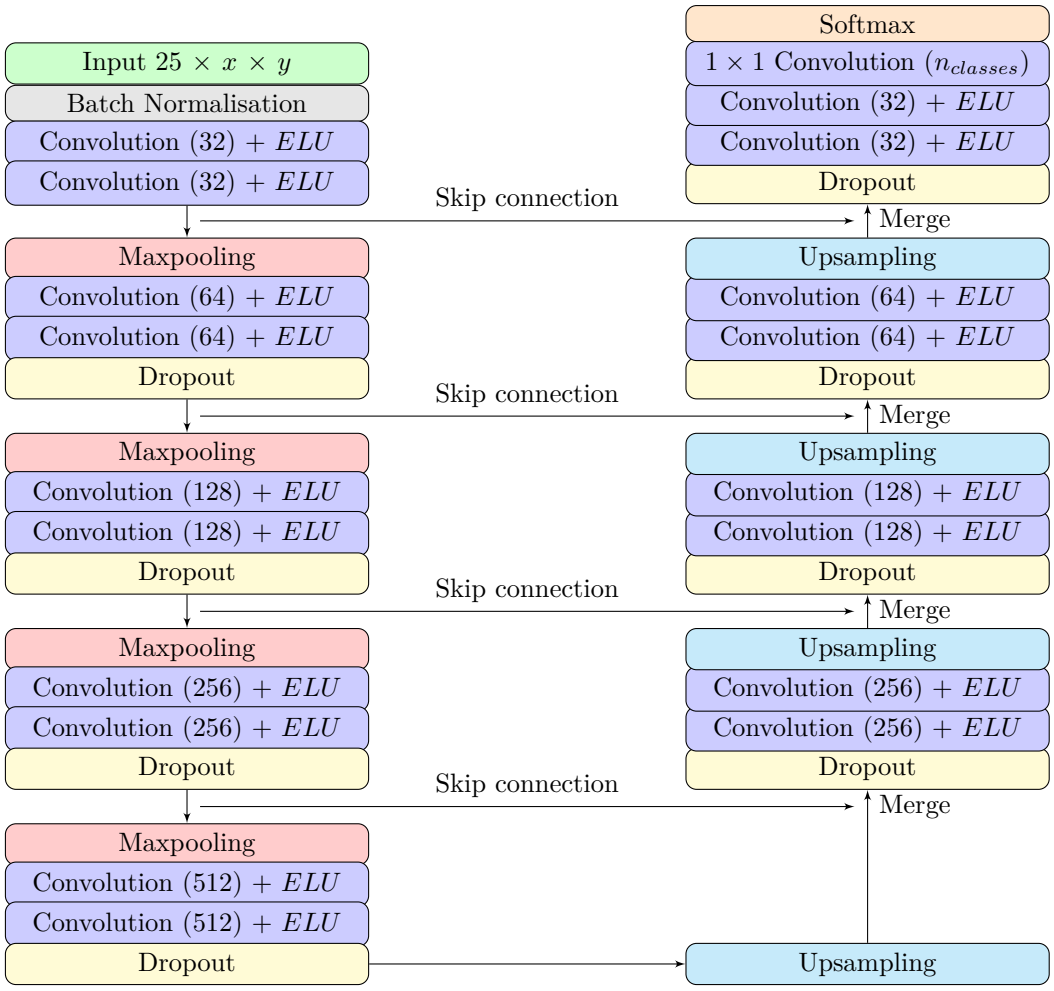

Figure 3-2: U-Net architecture; the skip connection merges two activation maps by an concatenating operation.

An additional small number ϵ is added to the variance in order to prevent division by zero. The normalized input $\hat{x}$ is then scaled by γ and shifted with β (equation 3-2). Therefore, per HS band trainable weights γ and β are learned.

$$\hat{x}_{BN} = \gamma \cdot \hat{x} + \beta \tag{3-2}$$

3-3-2 Convolutional Layer

Convolutional layers in the network (figure 3-2) have kernels of size $(3 \times 3 \times d)$, consequently the receptive field size is (3×3) and d is the input depth. The Keras convolutional layer function includes a *same border mode* parameter, which adds (zero) padding around the input to produce a same sized feature map as layer output.

A single kernel produces a two-dimensional feature map; the result of a $(25 \times 64 \times 80)$ patch convolved with (element-wise multiplication) a zero padded single $(3 \times 3 \times 25)$ kernel is a (64×80) feature map. Multiple kernels are used in a convolutional layer, producing multi-dimensional feature maps. The number of kernels per convolutional layer are displayed in

figure 3-2, the number between brackets in each convolutional layer. For every layer deeper into the network, a larger kernel is used. Deeper layers extract more complex and class specific information, which require a larger number of trainable parameters.

A kernel is a matrix of trainable weights, hence the total number of weights depends on its size. A single kernel of size $(3 \times 3 \times 25)$ in the first convolutional layer contains 225 weights. In this convolutional layer, 32 kernels therefore contain $225 \times 32 = 7200$ weights. Hence every kernel adds additional trainable weights to the neural network.

Non-linear Activation Function

All convolutional layers are followed by a nonlinear activation function ϕ. The nonlinear activation function used in the network is the Exponential Linear Unit (ELU) function (equation 3-3) [37]. This function has been chosen as it prevents the network performance degrading due to so-called *Dying Gradients*. A dead Rectified Linear Unit (ReLU) always outputs a zero value, for example caused by a large gradient update. For ReLUs this is an irreversible process. Recovering a dead weight is very unlikely, since weights are not updated for a zero gradient. As the ReLU gradient at zero is zero, weights remain unchanged. Dead weights do not contribute to the learning process [38].

Figures 3-3 and 3-4 display the ELU and ReLU activation function respectively. Using the ReLU function, negative input values will be set to zero, while for the ELU function, negative values will be set to a small number close to zero.

$$\phi_{ELU}(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{3-3}$$

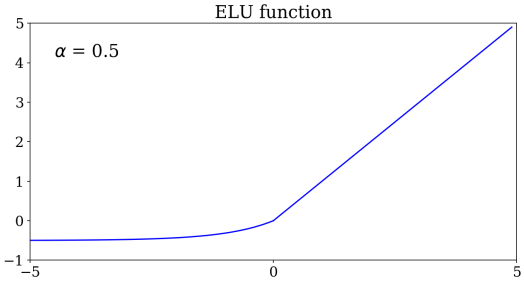

Figure 3-3: Exponential Linear Unit function, an illustrative example with $\alpha = 0.5$.

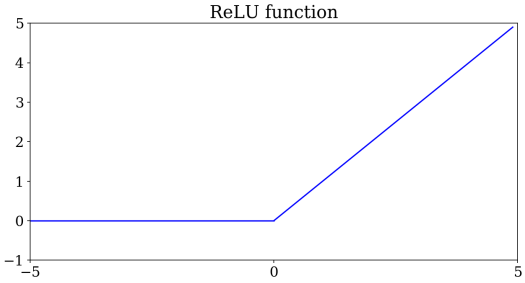

Figure 3-4: The rectified Linear Unit function is prone to dying gradients.

3-3-3 Softmax Final Activation Function

The final activation function differs from all other activation functions within the network. In the pixel-wise classification problem, classes are mutually exclusive. A softmax final activation function (equation 3-4) incorporates a mechanism in the neural network to benefit from this. The softmax activation function ensures a diffuse network output, so that the class with a high probability score is highlighted and classes with lower scores are suppressed. Furthermore, all outputs add up to one.

Equation 3-4 displays the softmax function and the function input z. In this equation the trainable weights are described by w.

$$\phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\Sigma_{j=0}^{k} e^{z_k^{(i)}}} \tag{3-4}$$
$$z = w_0 x_0 + w_1 x_1 + ... + w_m x_m$$

Let I be an image with x rows and y columns. The segmentation task targets distinguishing a certain amount of classes n_c. Therefore, a network input of $(25 \times I_x \times I_y)$ results in a network output of $(n_c \times I_x \times I_y)$. In the network output, for every pixel (I_x, I_y) a vector of probabilities is computed. This vector has size $(n_c \times 1)$. The class corresponding to the highest vector value, which can be interpreted as highest probability, is used to produce the final segmented image.

3-3-4 Feature Map Sizes

Every network layer has a corresponding feature maps size, starting with an input of $(n_b \times 25 \times 64 \times 80)$ (table 3-1). Batch size n_b is a hyperparameter which has to be set before training. Also patch size is a free parameter. Let P be a patch with x rows and y columns, then a patch containing 25 HS bands has size $(25 \times P_x \times P_y)$. In the example patch size is set to $(25 \times 64 \times 80)$. The table shows all transformation steps taken to go from the input to an output with correct size $(n_b \times n_c \times 64 \times 80)$, which is the patch size with a depth corresponding to the amount of classes.

From table 3-1 it becomes clear that the patch size becomes smaller going deeper into the network, reaching (4×5) pixels at the smallest point. Furthermore the amount of feature maps within the tensors grows with the depth of the network, with the maximum reached after merging two tensors at the 'deepest' point. Hence the input patch is transformed into stacks of feature maps, resulting in tensors with smaller size and larger depth.

3-3-5 Hyperparameters

There are a large number of hyperparameters which have to be defined before training a network. Apart from all parameters which circumscribe the training data set, there are certain parameters required for setting up a network and training it. Table 3-2 shows hyperparameter settings which have been used throughout all experiments.

The parameters have mainly been found by looking at typical segmentation networks, for example kernel size of (3×3) and stride of 1 are widely used in pixel-wise classification networks. Furthermore most parameters have been tuned by trial and error.

3-4 Loss Function

During training the goal is to adjust the neural network weights so that the predictions match the ground truth by adjusting the weights according to a decreasing loss value. The ground

Layer	Output Size	Layer	Output Size
Input	$(n_b \times 25 \times 64 \times 80)$	Upsampling	$(n_b \times 512 \times 8 \times 10)$
Batch Normalization	$(n_b \times 25 \times 64 \times 80)$	Merge	$(n_b \times 768 \times 8 \times 10)$
Convolution (32)	$(n_b \times 32 \times 64 \times 80)$	Dropout	$(n_b \times 768 \times 8 \times 10)$
Convolution (32)	$(n_b \times 32 \times 64 \times 80)$	Convolution (256)	$(n_b \times 256 \times 8 \times 10)$
Maxpooling	$(n_b \times 32 \times 32 \times 40)$	Convolution (256)	$(n_b \times 256 \times 8 \times 10)$
Convolution (64)	$(n_b \times 64 \times 32 \times 40)$	Upsampling	$(n_b \times 256 \times 16 \times 20)$
Convolution (64)	$(n_b \times 64 \times 32 \times 40)$	Merge	$(n_b \times 384 \times 16 \times 20)$
Dropout	$(n_b \times 64 \times 32 \times 40)$	Dropout	$(n_b \times 384 \times 16 \times 20)$
Maxpooling	$(n_b \times 64 \times 16 \times 20)$	Convolution (128)	$(n_b \times 128 \times 16 \times 20)$
Convolution (128)	$(n_b \times 128 \times 16 \times 20)$	Convolution (128)	$(n_b \times 128 \times 16 \times 20)$
Convolution (128)	$(n_b \times 128 \times 16 \times 20)$	Upsampling	$(n_b \times 128 \times 32 \times 40)$
Dropout	$(n_b \times 128 \times 16 \times 20)$	Merge	$(n_b \times 192 \times 32 \times 40)$
Maxpooling	$(n_b \times 128 \times 8 \times 10)$	Dropout	$(n_b \times 192 \times 32 \times 40)$
Convolution (256)	$(n_b \times 256 \times 8 \times 10)$	Convolution (64)	$(n_b \times 64 \times 32 \times 40)$
Convolution (256)	$(n_b \times 256 \times 8 \times 10)$	Convolution (64)	$(n_b \times 64 \times 32 \times 40)$
Dropout	$(n_b \times 256 \times 8 \times 10)$	Upsampling	$(n_b \times 64 \times 64 \times 80)$
Maxpooling	$(n_b \times 256 \times 4 \times 5)$	Merge	$(n_b \times 96 \times 64 \times 80)$
Convolution (512)	$(n_b \times 512 \times 4 \times 5)$	Dropout	$(n_b \times 96 \times 64 \times 80)$
Convolution (512)	$(n_b \times 512 \times 4 \times 5)$	Convolution (32)	$(n_b \times 32 \times 64 \times 80)$
Dropout	$(n_b \times 512 \times 4 \times 5)$	Convolution (32)	$(n_b \times 32 \times 64 \times 80)$
		Convolution (n_c)	$(n_b \times n_c \times 64 \times 80)$

Table 3-1: Feature map (tensor) sizes through the network, the input has size $(n_b \times 25 \times 64 \times 80)$, with batch size n_b and patches of size $(25 \times 64 \times 80)$.

Hyperparameter	Setting	Hyperparameter	Setting
Activation function	ELU	Dropout rate	10%
Weight initialization	Glorot normal [39]	Optimizer	RMSprop
		Initial learning rate	1e-5
Convolution border mode	Same	Batch size	100
Stride	1	Gradient clipnorm	1.0
Kernel size	(3×3)	Weight regularizer	None

Table 3-2: Hyperparameter settings

truth y_t and predictions patches y_p are matrices of the same size $(P_x \times P_y \times n_c)$. The ground truth patches have zeros on pixels without annotation, annotated pixels have ones on the specific class dimension in the matrix. The ground truth matrix (figure 3-5) shows three different types of pixels; not annotated (black), annotated in Class 1 (blue), annotated in Class 3 (red). After training a network, a prediction matrix is obtained (figure 3-6). Note that the sum of the predictions per pixel is one, due to the final softmax activation function. If the network is trained well, it should output high predictions at the annotated locations and low predictions at zero locations.

Loss value L (equation 3-6) is determined using the difference between the ground truth y_t and the masked predictions $y_{p,m}$ (figure 3-7). In the masked predictions $y_{p,m}$ not only pixels without corresponding annotation are masked, also entries which correspond to incorrect classes in annotated pixels are set to zero. Using the difference between ground truth y_t and (unmasked) prediction y_p in determining loss L impedes convergence of training. Updating weights is carried out in the direction of reducing loss L. As the final softmax function

outputs values for all classes in y_p, the difference $y_t - y_p$ will never reach zero and loss L will not converge. Thus, the difference $y_t - y_{p,m}$ will reach zero, hence training converges.

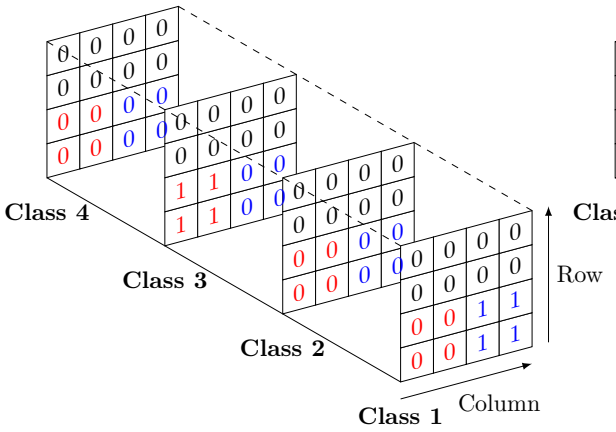

Figure 3-5: Example of a (4×4) Ground Truth patch y_t.

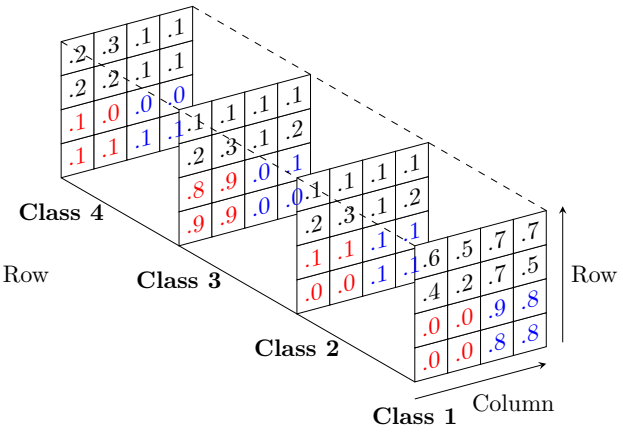

Figure 3-6: Example of a (4×4) prediction patch y_p.

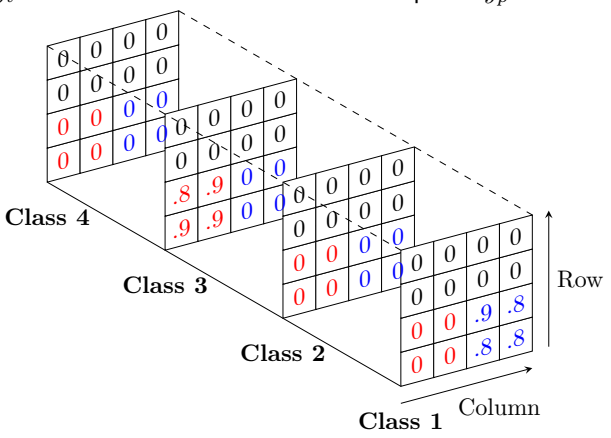

Figure 3-7: An example of a masked prediction matrix $y_{p,m}$.

So, the loss function uses the ground truth patches y_t and network prediction patches y_p to define a single loss value. The loss function is defined by the user. However, the best choice depends on the network architecture and type of data it is used for. The network weights are updated in the direction of decreasing loss function value. The loss value in the neural network is determined as follows:

1. Define loss function. For all experiments described here, a quadratic loss function has been used (equation 3-6).
2. Set predictions without complementary annotations to zero by masking them using the ground truth. This is done by taking the Hadamard product of y_t and y_p:

$$y_{p,m} = y_p \circ y_t \tag{3-5}$$

3. Determine the loss value L per patch; mean of the squared element-wise difference of y_t

and $y_{p,m}$. Here N is the number of pixels per patch:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_t - y_{p,m})^2 \tag{3-6}$$

4. Usually loss values are plotted per epoch. For each batch within an epoch, the average loss of that batch is computed. Then the average loss per epoch is calculated.

Note that all elements in ground truth patches y_t are either one or zero. The prediction matrix y_p is masked before calculating the loss, expressed by $y_{p,m}$. Thus pixels without annotations were omitted in determining the loss value. The code for the loss function used throughout this research is displayed in appendix B-2.

Furthermore y_t and y_p are used to determine overall accuracy and F-score during training. Sections 3-5-1 and 3-5-2 discuss calculation of these evaluation metrics (code in appendix B-3).

3-5 Evaluation Metrics

Evaluation metrics used to assess performance of the network are based on the confusion matrix, which displays the difference between predictions and ground truth per class. The confusion matrix has two dimensions, the predicted values y_p and the true values y_t. An example of a confusion matrix is given in table 3-3. In case that all pixels have been predicted correctly, numbers will only appear at the diagonal. Using the confusion matrix, the overall accuracy and F-scores per class are calculated as discussed in sections 3-5-1 and 3-5-2 respectively.

		Predicted					
		NA	Broad Leaf	Grass	Sand	Asphalt	Art. Grass
Actual	NA	2435189	0	0	0	0	0
	Broad Leaf	0	528442	641	0	0	0
	Grass	0	9943	112582	5243	202	66
	Sand	0	1	448	26285	1931	0
	Asphalt	0	376	393	31181	163265	933
	Art. Grass	0	3	0	0	0	636

Table 3-3: Example of a confusion matrix, NA refers to pixels without annotations.

3-5-1 Overall Accuracy

The overall accuracy of a prediction for an image segmentation is defined as the fraction of all correctly predicted pixels among the total amount of predicted pixels (equation 3-7). Note that pixels without annotations are not taken into account. Consequently these are all values from the diagonal of the confusion matrix, except for the first entry which is ignored:

$$\text{overall accuracy} = \frac{\text{correctly predicted pixels}}{\text{all predicted pixels}} \tag{3-7}$$

3-5-2 F-Score

The F-score is a metric to evaluate the accuracy of the predictions (equation 3-8). For every class, an F-score will be determined. The F-score combines precision (equation 3-9) and recall (equation 3-10) featuring a score of 1 as the perfect score:

$$\text{F-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3-8}$$

For every segmented image from the test data set, precision and recall of every class has been determined. Precision is the fraction of true positives (TP) among the true and false positives (TP + FP):

$$\text{precision} = \frac{TP}{TP + FP} \tag{3-9}$$

Recall is the fraction of true positives (TP) among the true positives and false negatives (TP + FN):

$$\text{recall} = \frac{TP}{TP + FN} \tag{3-10}$$

3-6 Data Set Partitioning

3-6-1 Training, Validation and Test Data Sets

The available HS images are divided into two sets. The first set contains the test images. The second set is used for patch selection, the patches are randomly allocated in either the training data set or the validation data set. The method used for generating the data sets is depicted in figure 3-8. It is essential to keep training and test data separate in predictive modeling, as it prevents too optimistic results or data leakage.

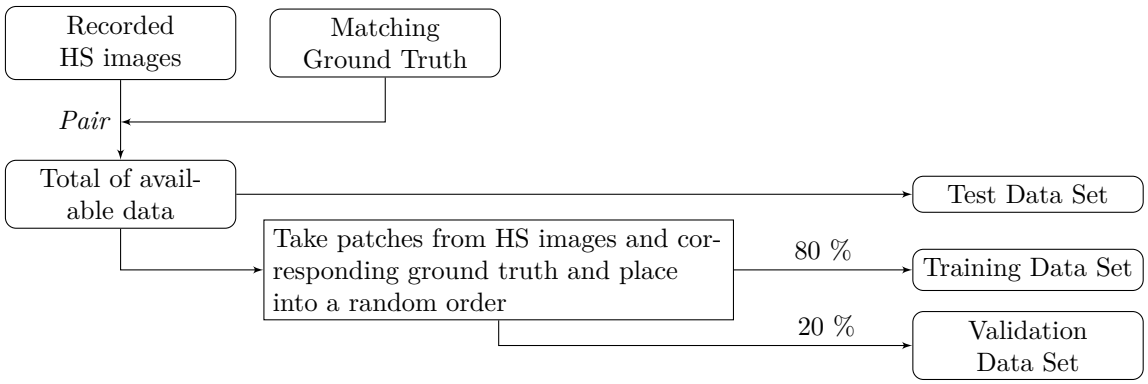

Figure 3-8: Method used for generating the test, train and validation set.

The ground truth is a mask valid for all HS images, which is discussed in detail in section 3-6-2. Furthermore a method of patch selection is required, as the full HS images are too large to be used for training the network. The method of patch selection is discussed in section 3-6-3.

3-6-2 Annotation Mask

The recorded HS images from the TNO dataset require a corresponding ground truth in order to train a neural network. A mask was sparsely annotated by assigning fixed pixels to a specific class. A single mask is used for multiple images, covering the whole recording period (appendix A). The annotation mask contains several classes which can be chosen freely (an example in figure 3-9). All pixels in white are not annotated. Arguments for using a sparse annotation mask are listed below:

- Quick method; annotating every single pixel is a time-consuming task
- Inclusion of mixed pixels in the training data set are avoided, i.e. pixels containing spectral information of multiple classes
- Not all object borders, or borders between classes, require dense annotations in order to yield accurate predictions
- Moving object borders due to the temporal characteristics of the data set do not require annotations
- The mask allows for fast alteration of the mask, e.g. addition of a class

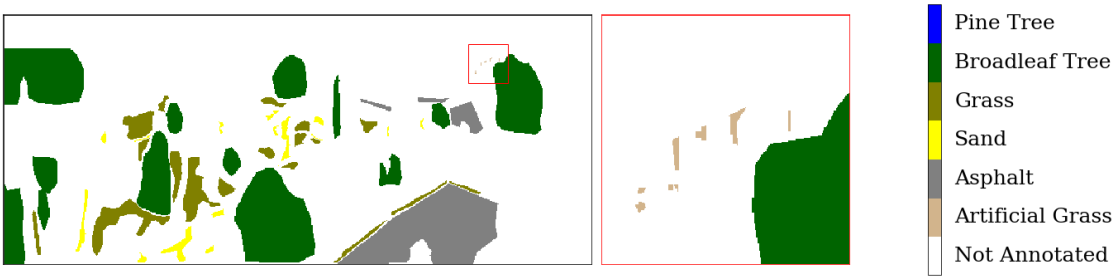

Figure 3-9: Visualization of the annotation mask; every color corresponds to another class. Zoomed in on the rare artificial grass class. White pixels do not correspond to a class.

Consequently the pixels which are not annotated cannot be regarded as a separate class for background as those pixels may contain any class. Instead, they are simply not considered during the training of the network by not having them contribute to loss function (equation 3-6). Furthermore the annotation mask can only be applied for images with similar physical appearance. Training is carried out under the assumption that the annotation mask is valid for all HS images within the training data set. However, object borders change due to vegetation growth (figure 3-10). This might cause pixels with incorrect annotations to be present in the training data set. Especially for the very small and oversampled artificial grass class, incorrectly annotated pixels have large effect on the final segmentations. Thus, in case the annotations start to diverge from the image content, another mask has to be used.

3-6-3 Patch Selection Method

The training and validation data set consist of pairs of patches from the original HS images and ground truth. So, a method of patch selection is defined. The patches are selected in a random fashion (figure 3-11). Every x- and y-coordinate of the top left corner of each patch is selected randomly. First patch size ($P_x \times P_y$) and amount of patches per image n_p have to be selected. After selecting patches from every HS image, all patches without annotated

Figure 3-10: RGB images of the artificial sports field from May to August.

pixels are removed. This results in a list of patches, which is then reordered randomly and divided into a train and validation data set.

As an example; 10 HS images with a setting of taking 1000 patches per image ($n_p = 1000$) results in a list of 10.000 patches. After removing the patches with no corresponding annotated pixels, this list reduces depending on P_x, P_y, n_p and annotation mask design.

3-6-4 Prepare Data for Network Input

After setting up the neural network, the data sets have to be prepared to meet the network requirements. For training, the network requires validation data and batches of training data as input. Firstly, information about image directory, patch coordinates within the images and patch size are saved in patch lists. The patch lists are then used to generate files comprising training and validation data in correct format in order to meet network requirements. Thus, files are created which contain:

- patch information for training `patch_list_train.p` and validation `patch_list_val.p` (using random patch selection, code in appendix B-4)
- information for test images `patch_list_test.p`
- for training: pairs of HS patches `X` and corresponding ground truth `Y` in batches `X_Y_train.h5`
- for validation: pairs of HS patches `X` and corresponding ground truth `Y` in `X_Y_val.h5`

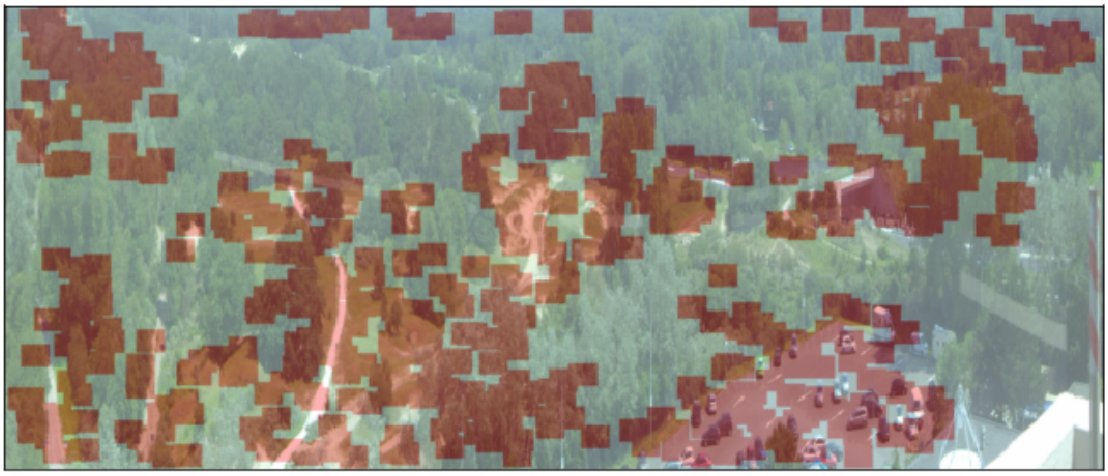

Figure 3-11: An example of an image with all patch locations in red (RGB image for visualization purposes only). Patch size is (64×80) and number of initial patches per image $n_p = 1500$.

- for testing: HS images in `X_test.npz`

The neural network is trained using train data in `X_Y_train.h5` and validation data in `X_Y_val.h5`. Inference is carried out using test data in `X_test.npz`, as explained in section 3-7.

3-7 Network Training and Inference

Batches of training data and validation data are the network input. Training is carried out as long as the stopping criterion has not been met. Examples of stopping criteria are: a set number of epochs, a fixed number for training loss and fixed number of epochs for which the loss may increase before stopping the process. After training, the network weights are saved. The combined weights and network architecture describe the model, which is then used to predict a class per pixel for the test dataset.

Inference is carried out in order to yield the pixelwise predictions for the test data. Inference is the process of using a trained neural network to simulate predictions for the test data set. Firstly, the weights determined in training are loaded into network. Secondly inference is applied. Finally the output prediction matrices are converted into images, which can be evaluated using the overall accuracy or F-score.

Chapter 4

Experimental Setup

This thesis is aimed at answering two separate research questions. The first question concerns selecting the optimal training data set so that the segmentations meet user requirements. The experiments corresponding to this question are discussed in section 4-1. The first part of this research is mainly aimed at preparing training data and supporting decisions in order to answer the second research question. Subsequently, the second question is aimed at evaluating neural network performance under temporally changing conditions. Section 4-2 discusses the experiments regarding the second research question.

4-1 Experiments for Optimal Training Data Set Selection

For this set of experiments all results are obtained with a test, training and validation data set composed of hyperspectral (HS) images as shown in table 4-1. In total, 28 images are used for training and validation. The selection of images are taken over a short period of time, hence vegetation shows very little seasonal change. Furthermore the weather conditions in this training data set are very similar. Two test HS images are used to assess segmentation results, the test images are subject to similar and different weather conditions compared to the training data set. The RGB visualization the scene of one of the test images is displayed in figure 4-1.

Date	**Hours**	**Data Set**	**Weather**
03-06-2016	13	Test	Foggy
04-06-2016	11-17	Training + Validation	Sunny
05-06-2016	11-17	Training + Validation	Sunny
06-06-2016	11-17	Training + Validation	Sunny
09-06-2016	13	Test	Sunny
10-06-2016	11-17	Training + Validation	Sunny

Table 4-1: HS images for test, training and validation data sets.

For all experiments applies that training is stopped at the epoch which has the lowest validation loss and highest overall accuracy. Therefore, not all experiments are subjected to the same amount of epochs for training.

Figure 4-1: Image from the test data set, recorded on 09-06-2016 at 13 p.m. Note that RGB images are not used for training, they are used for visualization purposes only.

The results of the experiments for optimal training data set selection are presented in chapter 5. Guidelines for annotation mask design are discussed in section 5-1, and guidelines for patch selection clarified in section 5-2.

4-1-1 Annotation Mask

Annotation mask design is subject to variation, as it is a manual task depending on the design goals. For example, differences appear in the level of detail of annotated pixel regions, number of annotated pixels per class and number of annotated classes. As the mask design has countless different options, some general guidelines help obtain insight on its effects on final segmentation results.

Questions arise when designing an annotation mask. Does the level of detail in the annotation mask influence segmentation output? Do small objects from the annotation mask appear in the segmentation? Is it required to annotate every example available within a class (e.g. distant and close trees)? Does addition of a class, which is very similar to an already existing class, influence segmentation quality? These questions are answered in several experiments, which focus on the following aspects of annotation mask design:

- Level of detail of small objects (results in section 5-1-1)
- Level of detail at class borders (results in section 5-1-2)
- Close and distant examples within a class (results in section 5-1-3)
- Introducing an additional class (results in section 5-1-4)

A benchmark annotation mask has been made to compare results with (figure 5-1). Four additional masks have been made by adding or removing pixels from the benchmark annotation mask.

The following parameters for train set selection have been used: $n_p = 1000$, $P_x = 64$, $P_y = 80$, $f_o = 70$ and $f_u = 0$ (the definition of factors f_o and f_u is explained in section 4-1-2). Note

that the annotation mask contains only few artificial grass pixels. For this class only a small number of pixels was available for annotation, due to its small relative size compared to the total image and due to trees partly covering the area.

4-1-2 Patch Selection

Not only the annotation mask influences the final segmentation results, also the method of selecting patches for train and validation sets is of importance. The initial number of patches taken per HS image n_p is a variable which largely influences the number of training samples (section 3-6-3). Furthermore, the annotation mask (as well as the original HS image) contains a lot of broad leaf tree pixels and only few artificial grass pixels, the training data set is subject to a large imbalance. By applying an oversampling method on the rare class and an undersampling method on the most common class, the training data set will become more balanced and the network will convergence to a point of lower loss value.

Subsequently, questions arise regarding the patch selection method. How many patches per image n_p are required to make sure the network converges? What is the optimal training data set composition regarding segmentation accuracy? Does segmentation accuracy improve by undersampling the most common class or oversampling the rarest class? Is it necessary to keep the relative size of pixels per class similar to the annotation mask? Is there a threshold value for the amount of annotated pixels in order to segment a class, or is the ratio of pixels per class more essential? In order to answer these questions, the experiments focus on the following aspects of patch selection:

- Initial amount of patches per image n_p (results in section 5-2-1)
- Factor f_o oversampling on rarest class (results in section 5-2-2)
- Factor f_u undersampling on most common class (idem)
- Combined over- and undersampling (idem)

All experiments are carried out using the same annotation mask (figure 5-1). The following parameters for training data set selection have been used $P_x = 64$ and $P_y = 80$.

Oversampling with Factor f_o

The experiments include oversampling the rarest class by factor f_o. Oversampling is carried out by including additional patches containing the rare class to the training and validation data set. These additional patches are selected close to the location of the initial patches, only a few pixels up, down, right and left from the original patch. The factor f_o stands for the factor by which the initial training and validation data set is increased. So, if the initial train set consist of 50 patches of a rare class, an oversampled training data set with factor $f_o = 10$ contains 500 patches of this rare class.

Undersampling with Factor f_u

The experiments include undersampling the most common class by factor f_u. Undersampling is carried out by randomly removing patches containing this common class. Only patches containing a single class are removed. So, if a train set consists of 1000 patches of a common

class, undersampling with factor $f_u = 0.7$ will result in a undersampled train set of 300 patches containing this class.

4-2 Experiments for Temporally Changing Conditions

For this set of experiments the extend to which the neural network can be used to compensate for temporally changing ambient conditions is evaluated. Networks are trained using data from several periods of time and multiple weather conditions. Questions arise: Are neural networks capable of compensating for differing lighting and weather conditions without an atmospheric model? Are neural networks capable of generalizing over seasons? Are results only valid at days close to the recording days of the training data? The results of the experiments are presented in chapter 6.

Neural networks are known for its generalizing properties. In case of a scene subject to temporally changing ambient conditions, there are two options for assessing its generalizing capabilities:

- Interpolating between training data set images
- Extrapolating beyond training data set images

For all experiments, testing neural network performance is based on interpolation. Using interpolation, the test images are selected in such way that they fall within the period of time the training data set was taken. In extrapolating, the test images go beyond this period of time. All experiments were carried out using the same annotation mask (figure 5-1) and the following training data set parameters $n_p = 1500$, $f_o = 70$ $P_x = 64$ and $P_y = 80$. Furthermore, the number of samples in the training data set is kept approximately the same for all experiments.

4-2-1 Interpolating Between Days in One Week

Experiments start by evaluating results from a relatively simple training data set formed using images taken over the course of a week under similar weather conditions. This is done for a week in May, June, July and August. Subsequently, similar experiments are carried out, but then replacing some of the images with an image for a different weather type to assess the generalizability of weather. The HS images for the training, validation and test data sets are listed in appendix C-2-1. Results are presented in section 6-1.

- Interpolating between days in one week (Four periods; May, June, July and August)
  - Similar weather conditions
  - Different weather conditions
- Extrapolating

The HS images recorded in the beginning of May are subject to large seasonal change, as growing season is at its highest point. Images from June, July and August are more similar. For all training data sets in this series of experiments, a total of 28 HS images taken from 4 separate days were used.

The HS images from April and May were recorded with different camera settings, subsequently no annotated artificial grass pixels are present in the images. Hence oversampling is not

possible, which results in a small training data set size. To avoid overfitting of this training data set, a larger number of patches per image were used ($n_p = 2000$).

4-2-2 Interpolating Between Days in Four Weeks

The period of time in which the training data set is selected, is expanded to a period of four weeks. The four-week experiment is carried out twice, as images taken in May and June are subject to larger vegetation changes than images from July and August due to growing season. The HS images for the training, validation and test data sets are listed in appendix C-2-2. Results are presented in section 6-2.

- Interpolating between days in four weeks (Two periods; May and June, July and August)
  - Similar weather conditions
  - Different weather conditions
- Extrapolating

For all training data sets in this series of experiments, a total of 32 HS images taken from 8 separate days were used.

4-2-3 Interpolating Over a Longer Period of Time

In these experiments the training data set comprises images from April to August, aiming at training a network which generalizes input data irrespective of time of the day, season and weather. As this training data set will cover a long period of time, the training data set is subject to a large range of variation. The HS images for the training, validation and test data sets are listed in appendix C-2-3. Results are presented in section 6-3.

- Interpolating between days in five months (April to August)
  - Similar weather conditions
  - Different weather conditions
- Extrapolating

For all training data sets in this series of experiments, a total of 32 HS images taken from 8 separate days were used.

Chapter 5

Training Data Set Optimization

The weights learned in the training process of neural networks largely depend on the content of the training data set. Successfully accomplishing the segmentation task depends on the information that the network is exposed to. Therefore, training data set selection is an essential step in the framework. This chapter discusses the effect of annotation mask design and patch selection on training results. First the mask design is discussed (section 5-1). Secondly, the number of patches required for training is discussed, as well as class balancing by over- and undersampling the training data set (section 5-2). Finally, typical development of segmented test images during training is evaluated (section 5-3)

5-1 Guidelines for Annotation Mask Design

The benchmark annotation mask (figure 5-1) has been used as a reference for all experiments in this section. First the influence of a changed level of detail in the annotation mask has been investigated. Then effects of the inclusion of distant and close examples of broad leaf trees are discussed. Finally, the introduction of an additional class has been examined.

Figure 5-1: Benchmark annotation mask.

5-1-1 Level of Detail of Small Objects

The benchmark annotation mask has been changed by removing the details demarcated with the red boxes (figure 5-2). Those details represent narrow sand paths. The removed details represent two individual regions of sand pixels, and a region of pixels containing both grass and sand. After training ($n_p = 1500$, $f_o = 70$) the results are evaluated.

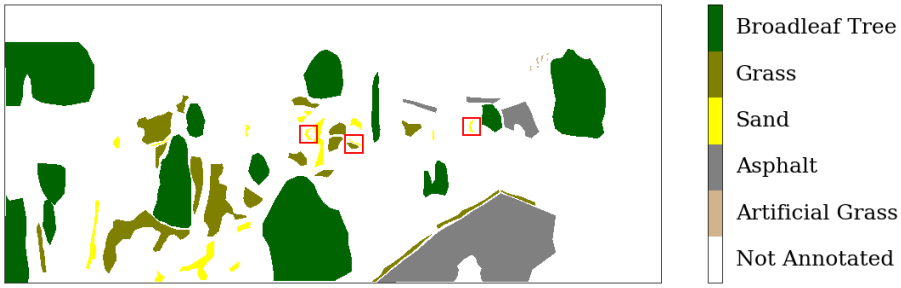

Figure 5-2: Annotation mask; the small details of the sand class within the red boxes are removed in order to assess its influence on the final segmentation.

From figure 5-3 it becomes clear that removing the detail containing sand pixels only (red box) does not significantly reduce the level of detail in the segmentation at that specific location. Due to the use of small patches and the sparsity of the annotation mask, the network is not able of learning the context, and therefore location of those details. Removing the detail containing both sand and grass pixels (blue box) actually does reduce the level of detail in the segmented map. The segmentation based on training a network with a detailed mask, shows a distinguished sand path (figure 5-3, top row). The segmented map obtained with a less detailed mask does not show this. U-Net learns relative class locations. Increased level of segmentation detail can be achieved by increasing the density of the annotations at class borders, i.e. training patches should contain multiple classes.

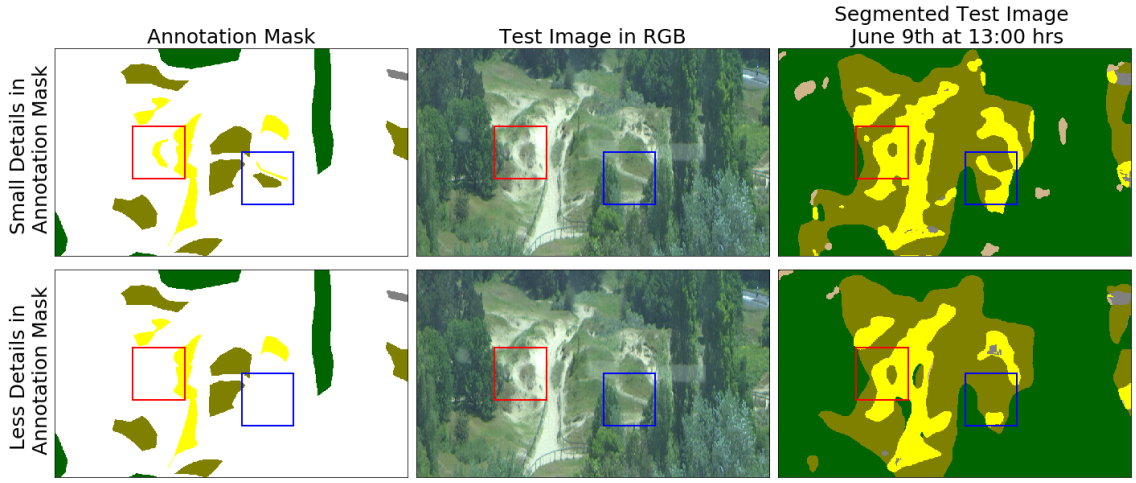

Figure 5-3: The red box contains detail of sand pixels only, the blue box contains a detail including grass and sand pixels.

5-1-2 Level of Detail at Class Borders

Section 5-1-1 discusses increasing level of detail in the final segmentation maps by introducing small details in the annotation mask. In this section increasing details in the final segmentation map by increasing density of annotation at class borders will be investigated. In order to examine the result, the benchmark mask has changed by removing the grass strip near the asphalt parking area (figure 5-4).

Figure 5-4: Annotation mask with a lower level of detail near the edge of asphalt in the lower right corner.

The network has been trained using an annotation mask with lower level of detail. The segmentation maps from both the detailed mask (benchmark) and less detailed mask have been compared (figure 5-5). All test images show that the class borders are less tight in the case of a less detailed annotation mask.

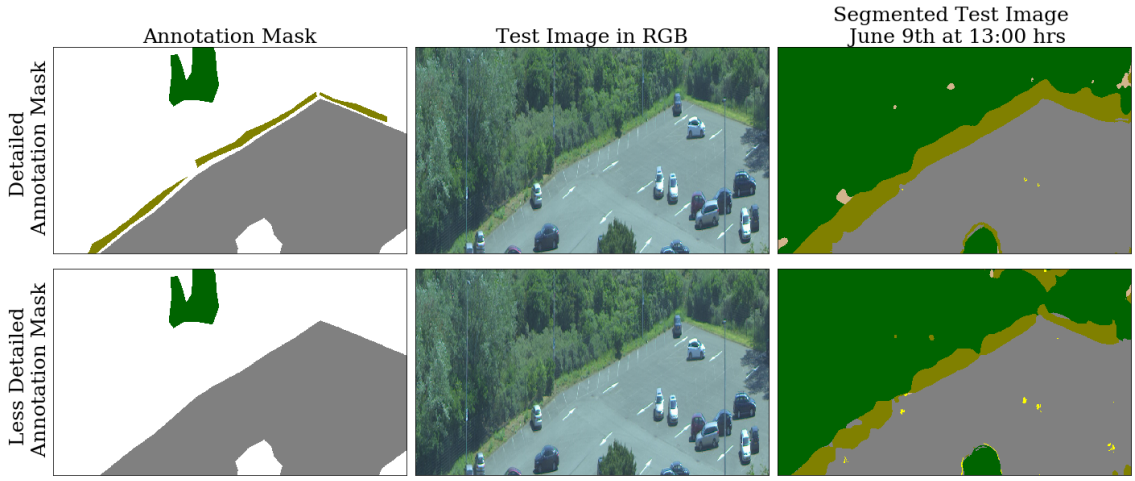

Figure 5-5: Differences in using a detailed or less detailed annotation mask for training.

5-1-3 Annotation of Distant and Close Examples

Generally, exposing a network to many different examples increases performance. Adding examples of both distant and close trees seems useful, as both examples appear dissimilar. The benchmark annotation mask has been altered by removing annotations for distant broad leaf trees (figure 5-6).

Figure 5-6: Annotation mask without annotations for distant broad leaf trees.

Visual observation shows that distant and close trees indeed have a different appearance (figure 5-7), possibly caused by the atmospheric conditions. Training with an annotation mask without distant trees results in a slightly different segmented image. For both masks, classification of the distant and close trees are correct. The difference between these images emerge in the other classes, especially in distinguishing artificial grass, grass and broad leaf trees. A trained network based on an annotation mask including distant trees tends to predict too many artificial grass pixels. However, for the annotation mask without distant trees, artificial grass is predicted as grass. Therefore, including distant trees in the mask does not necessarily improve predictions of trees in general. However, it does influence classification of other classes. Note that the annotation mask includes a smaller amount of broad leaf tree pixels, which influences the training data set balancing.

Figure 5-7: Segmented maps in case of distant trees in annotation mask and in case of no distant trees in annotation mask.

5-1-4 Additional Class

The number of classes defined in the training data set affect final segmentation results. It is beneficial to choose classes which are easily separable and which have a lot of data readily

available. The benchmark annotation mask has been changed by adding annotated pixels of pine trees (figure 5-8).

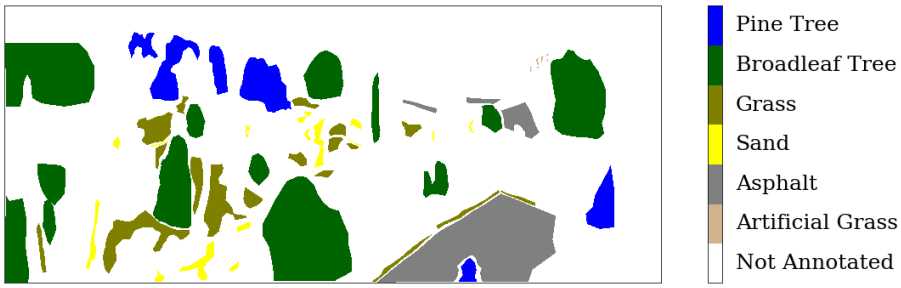

Figure 5-8: Annotation mask with an additional pine tree class.

Adding a class in training, degrades segmentation performance under similar training settings ($n_p = 1500$, $f_o = 70$ for artificial grass). Pine trees are distinguished, however the F-score of other classes drop (figure 5-9). Differentiating between six classes in a sparse annotation mask becomes very complex.

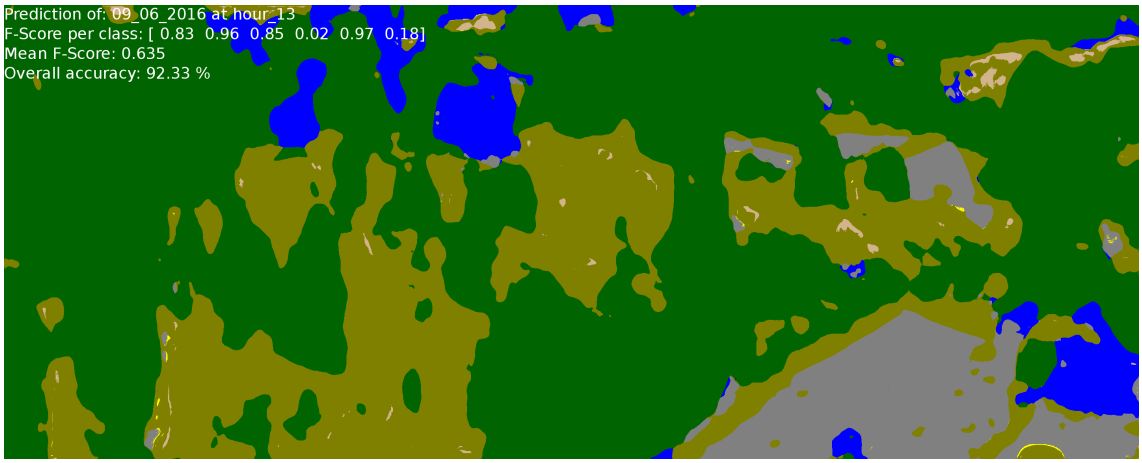

Figure 5-9: Segmented test image including the pine tree class (blue). F-scores represent pine trees, broad leaf trees, grass, sand, asphalt and artificial grass respectively.

Close observations of the segmented images and the original scene show that visually distinguishing pines from broad leaf trees is not straightforward (figure 5-10). The network might require more information about location, which is not given by the sparse annotation mask. Segmenting the additional class results in lower overall performance, especially performance of the sand class has dropped significantly to a value of 0.02.

In further research, pine trees are not used for segmentation as it decreases overall performance. Furthermore, segmenting this class has no added value for the application.

Figure 5-10: Distinguishing pines trees from the RGB image visually is not straightforward. However, the neural network is able to predict pine tree pixels with an F-score of 0.83.

5-2 Guidelines for Patch Selection

Using the patch selecting method described in section 3-6-3, the training data set does not only depend on the annotation mask, but also on the number of patches per image n_p (section 5-2-1). Initially selected training patches are not corrected for classes with higher chance to be present in the training data set due to the number of annotated pixels per class, size of annotated regions or the number of annotated regions. The training data set is balanced to increase performance of all classes (section 5-2-2).

5-2-1 Number of Training Patches

Increasing the training data set size improves neural network performance, however limitations are imposed by computer memory and computing time. Therefore choosing the correct number of patches per image is essential in obtaining the optimal result. The following results are obtained training two networks with different training data set sizes, using $n_p = 1000$ and $n_p = 1500$. Both sets are created using the same 28 hyperspectral (HS) images. The first data set contains 10805 patches, the second data set has 16189 patches, effectively using 385 and 578 patches per image respectively (details in section 3-6-3). Both training data sets have similar compositions regarding percentage of pixels per class (appendix C-1-1). No over- or undersampling is applied in this experiment, therefore $f_o = 0$ and $f_u = 0$.

Using more patches, training is more stable and convergence is faster (figures 5-11 and 5-12). Furthermore, when training a small training data set, the difference between training and validation accuracy grows per epoch, which is an indication for overfitting the training data. However, the downside of using more patches is slower training and larger memory usage. Furthermore, for both settings of patches per image n_p, the segmented images are not predicted perfectly (figures 5-13 and 5-14). Both segmentation maps show misclassification. Moreover, artificial grass has not been distinguished in both cases. However, training data set selection using $n_p = 1500$ significantly improves results, especially due to less sand-asphalt confusion.

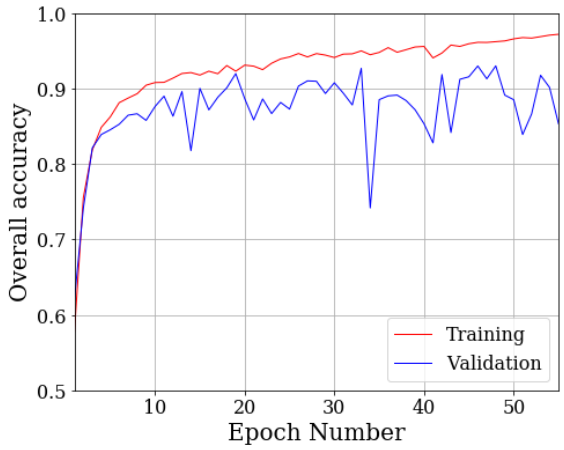

Figure 5-11: Training and validation accuracy per epoch for a train set containing 10805 patches.

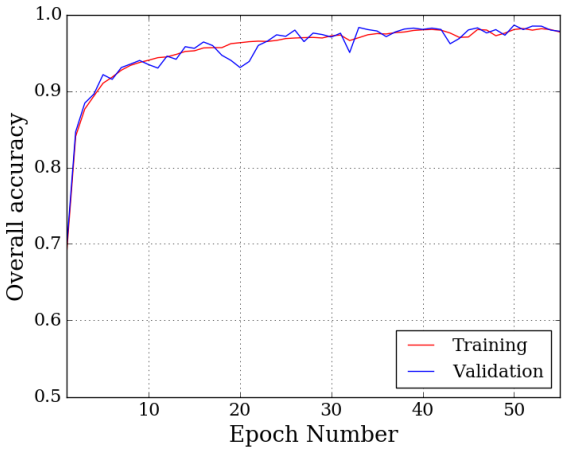

Figure 5-12: Training and validation accuracy per epoch for a train set containing 16189 patches.

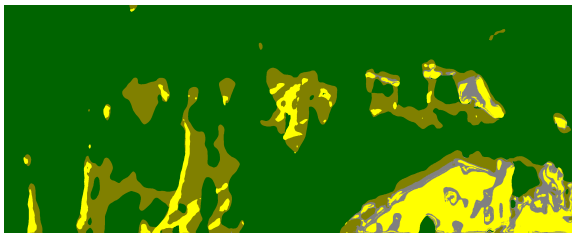

Figure 5-13: Segmentation of test image (June 9^{th} at 13 p.m.) obtained by a network trained with a set of 10805 patches.

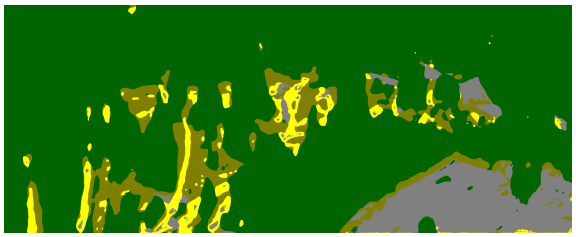

Figure 5-14: Segmentation of test image (June 9^{th} at 13 p.m.) obtained by a network trained with a set of 16189 patches.

5-2-2 Selection of Training Patches

In order to classify all classes perfectly, both training and validation loss is required to converge approaching zero. Furthermore the accuracy should converge to one. Balancing classes in the training data improves network performance. In experiments oversampling on rare classes is applied, as well as undersampling on frequent classes.

No Class Balance Applied

Figures 5-13 and 5-14 clearly show that not all classes are predicted correctly. In the experiments no oversampling or undersampling has been applied to the training data set. The training data set class composition for $n_p = 1500$ is displayed in table 5-1, the training data set for $n_p = 1000$ is very similar (appendix C-1-1). The network is trained to segment five different classes, however the segmentation maps show four classes only. One class has not been distinguished at all.

Figure 5-15 shows is an enlarged version of figure 5-14, but printed with corresponding evaluation metrics. It is a segmentation obtained from an unbalanced training data set using $n_p = 1500$. The F-score supports the qualitative findings done using the visual appearance of

Class	Pixels in training data set [%]
Broad leaf tree	57,85
Grass	15,31
Sand	3,46
Asphalt	23,30
Artificial grass	0,08

Table 5-1: Composition of training data set without balancing for $n_p = 1500$ ($f_o = 0$, $f_u = 0$)

the map only. The F-score for artificial grass is zero, which sorts with the absence of this class in the map. Artificial grass has mainly been classified as broad leaf tree (red box in figure 5-15). Furthermore, there is some confusion between asphalt and grass. As artificial grass accounts for a very small amount of all pixels in the test images, the improvements regarding the segmentation map come down to two main aspects:

- Increase overall accuracy
- Increase F-score for artificial grass

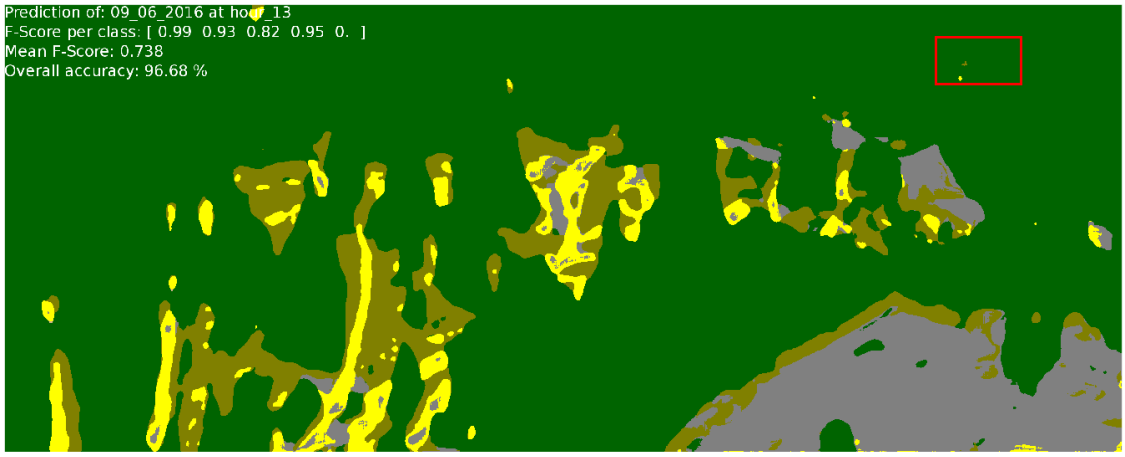

Figure 5-15: Segmented map with corresponding evaluation metrics, for a training data set without balancing and $n_p = 1500$. The red box indicates the correct location of artificial grass. F-scores represent pine trees, broad leaf trees, grass, sand, asphalt and artificial grass respectively. RGB image for comparison in figure 4-1.

The network requires to converge to a smaller loss value in order to increase segmentation accuracy. Furthermore, to increase the amount of examples of artificial grass, the relative amount of annotated pixels for artificial grass has to be increased. Both aspects are dealt with in balancing the training data set. Balancing is done by decreasing the dominant class, by enlarging the rare class or by a combination of these two methods.

Effect of Oversampling

The artificial grass class has been oversampled with a factor f_o, using the method explained in section 4-1-2. The number of patches has been increased by a factor $f_o = 10$ and a factor $f_o = 70$, the composition of both training data sets are shown in section C-1-2. For

oversampling the rare class with $f_o = 10$, this class remains the smallest in size. Oversampling with factor $f_o = 70$ causes the artificial grass class to approximately equal the sand class in size.

The networks are trained with several training data sets, varying values for n_p and f_o (results in appendix D-1). Figure 5-16 shows the segmentation maps for a test image under similar and different weather conditions compared to the training data set. The experiments show there is variation in the final result, both in overall accuracy and in F-score. Settings of n_p and f_o have a different effect on segmented test images in case weather conditions deviate from training data set. A training data set generated with a higher factor f_o is more robust for varying weather conditions. Concluding, oversampling on the rare class does increase the F-score for that class. All experiments show that artificial grass has been distinguished. Summarizing, oversampling on artificial grass has the following effects on the segmented test images:

- Oversampling increases F-score artificial grass
- Oversampling with $f_o = 70$ increases overall accuracy

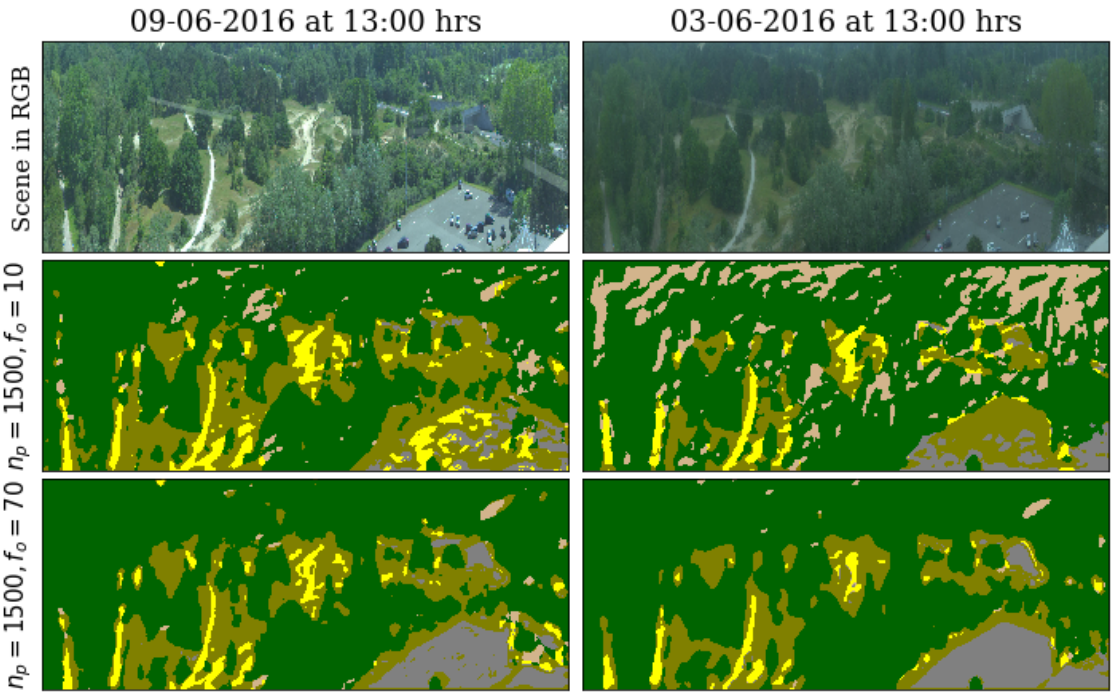

Figure 5-16: Networks trained with training data sets under sunny weather conditions, for $n_p = 1500$ and several values of f_o. Test images are subject to both sunny (09-06-2016) and foggy (03-06-2016) conditions.

It is important to note that the number of training patches is a lot higher in case of $f_o = 70$ compared to $f_o = 10$. For both data sets applies that $n_p = 1500$; the training data set oversampled with $f_o = 10$ contains 17328 patches, the training data set oversampled with $f_o = 70$ contains 24410 patches.

Effect of Undersampling

Approximately 60 per cent of all pixels in the training data set are annotated as broad leaf tree, all other classes are significantly smaller. By undersampling this dominating class, the amount of pixels per class is more balanced (training data set composition in appendix C-1-3). Undersampling is applied by removing a certain percentage of all patches containing the dominating class in a random fashion.

The broad leaf tree class remains the largest after undersampling with factor $f_u = 0.6$. In undersampling with factor $f_u = 0.8$ the broad leaf trees becomes the third largest class. These factors f_u have been chosen in order to find out if the ordering of class sizes in the training data set should be according to ordering of class sizes is the test images. Experiments have been carried out for different values of n_p and f_u (results in appendix D-2). Experimental results show that undersampling the broad leaf tree class has the following effects on the segmented test images:

- Undersampling broad leaf tree does not increase F-score artificial grass
- Undersampling broad leaf tree does not increase overall accuracy
- No indications that same ordering should be kept

Combined Oversampling and Undersampling

In order to increase the extend to which the training data set is balanced, both oversampling and undersampling is applied. The rare class has been oversampled by factor f_o. Furthermore the most frequent class has been undersampled by factor f_u. However, the network did not yield stable and robust results training such a training data set. In all further experiment training data sets will be generated with $n_p = 1500$, $f_o = 70$ and $f_u = 0$, as distinguishing artificial grass is a research objective.

5-3 Development of Segmentation Map during Training

In this experiment the training data set consists of 27600 patches ($n_p = 1500$, $f_o = 70$) with a batch size $n_b = 100$. Typical development of a segmented test image during training is evaluated after training several batches or epochs (figure 5-17). One epoch consists of $\frac{\text{training data set size}}{\text{batch size}} = \frac{27600}{100} = 276$ batches. Thus after 138 batches, the network has processed half of the first epoch. After each epoch the network used all patches in the training data set for updating the weights. Consequently, after the second epoch all patches have been used twice.

10 batches	Low level features appear, e.g. edges
Half epoch	Low level start to disappear, contours of objects start to develop
Epoch 1	Network distinguishes asphalt, grass and broad leaf tree
Epoch 5	Idem
Epoch 10	Network distinguishes asphalt, grass, broad leaf tree and sand
Epoch 15	Idem
Epoch 20	Idem
Epoch 30	All classes distinguished, higher level of detail in classification

First, weights in shallow layers are updated, as the test image mainly shows edges after training 10 batches. Next, weights in deeper layers are trained, enlarging the degree of complexity of decisions for classification. Classes which can be differentiated using less complex features are classified first. These classes are broad leaf tree, grass and asphalt. After training deeper layers, the final two classes are segmented. Evidently, sand and artificial grass require more complex decision boundaries. Possibly, training less classes in a single model significantly increases overall accuracy and F-score of the final result.

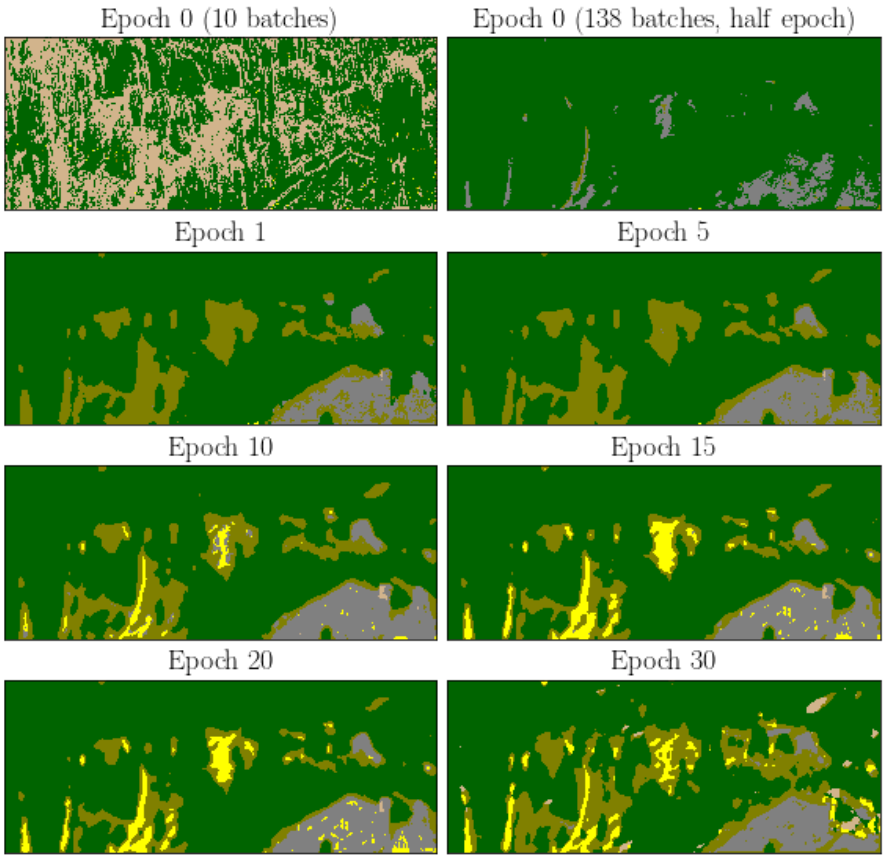

Figure 5-17: Typical segmentation development of an image from the test data set. Segmentations are generated at several moments during training the network.

Chapter 6

Experimental Results

The scene displayed in the hyperspectral (HS) images is subject to temporally changing ambient conditions. Throughout the year, illumination conditions as well as physical appearances change. Vegetation is subject to its growth cycle, subsequently scene appearance keeps changing over time. Furthermore weather conditions have impact on HS data. In order to answer the second research question, experiments on this temporally changing scene have been carried out. Test images will be segmented by a network trained with data taken over the course of a week (section 6-1), four weeks (section 6-2) and five months (section 6-3).

6-1 Interpolating Between Days in One Week

In this set of experiments training and testing have been carried out using data taken over a period of approximately one week. Training, validation and test data sets used in the following experiments are listed in appendix C-2-1. All results are shown in appendix E, the most important and revealing results are displayed below. Experiments are carried out with images from a single week in May, June, July and August.

6-1-1 A Single Week in June

The HS data for this experiment was taken between 02-06-2016 and 11-06-2016. This week falls right after growing season, therefore the images within the training data set are very similar. A network trained using data under a single weather condition yields results with an overall accuracy ranging between $95-97\%$ for a sunny test image. For rainy test images results range between $94-95\%$ (both in figure 6-2). So, two segmented test images generated by a model trained with data under a single -*sunny*- weather condition (figure 6-1) are assessed. Interestingly, this model is accurate in segmenting both sunny and cloudy test images.

As lighting conditions change over the day, generalization should work irrespective of time of the day. The training data set includes images recorded at all hours between 11 a.m. and 17

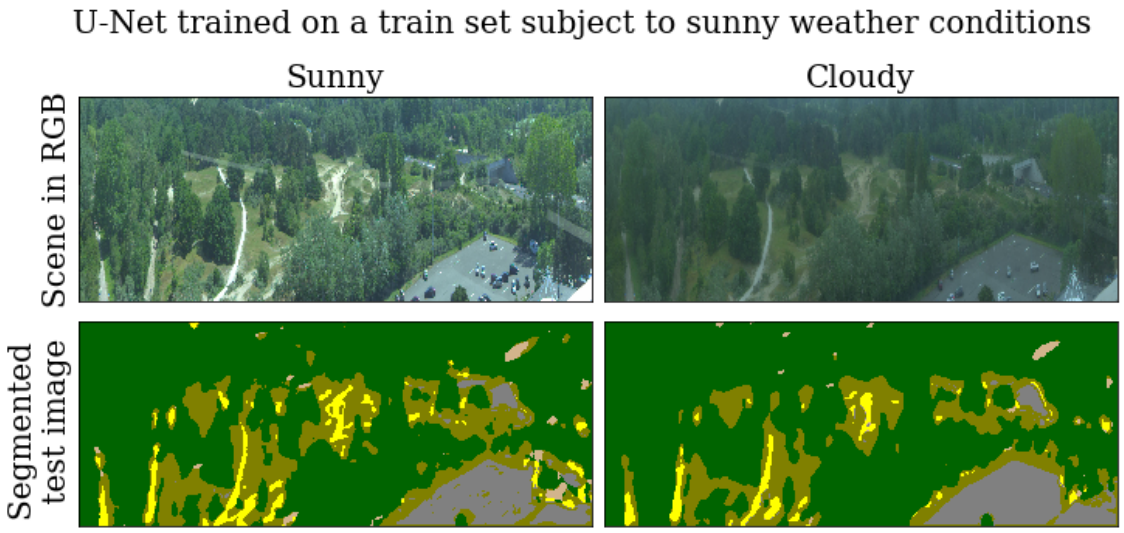

Figure 6-1: A single model yields $> 95\%$ accuracy on both sunny (97% accuracy, 0.96 mean F-score) and rainy (96% accuracy, 0.86 mean F-score) test images. Both test images are recorded at 13 p.m.

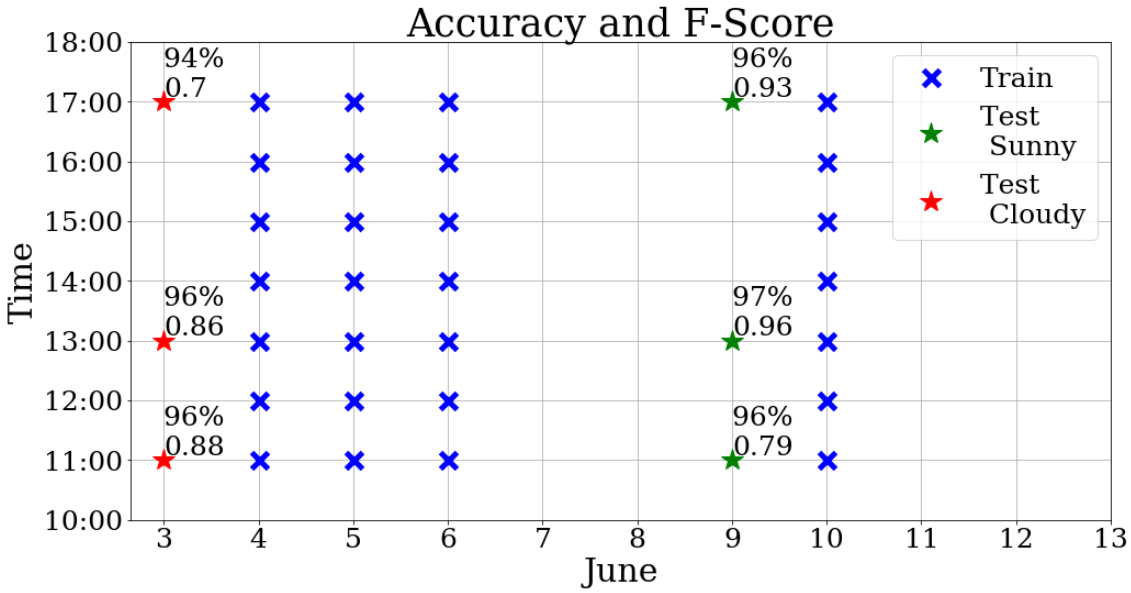

Figure 6-2: Accuracy and F-score of test images over time (model trained trained with a single -*sunny*- weather type in June).

p.m. Observation of the segmented test images show that the network yields an accuracy of $> 94\%$ (figure 6-2) for all times of the day.

Results for a network trained using data under multiple weather conditions are significantly worse (appendix E-2, figure E-2). Overall accuracy for a rainy test image ranges between $70 - 74\%$, for sunny test images results between $75 - 77\%$ are obtained. Both training and validation loss do not converge to a low value. Comparing the spectral signatures of the classes

at rainy and sunny days (figure 6-3) show large differences, as signatures from sunny days have a higher intensity. Therefore, training data sets including multiple weather conditions are more complex, as the range of spectral signature values per class is larger.

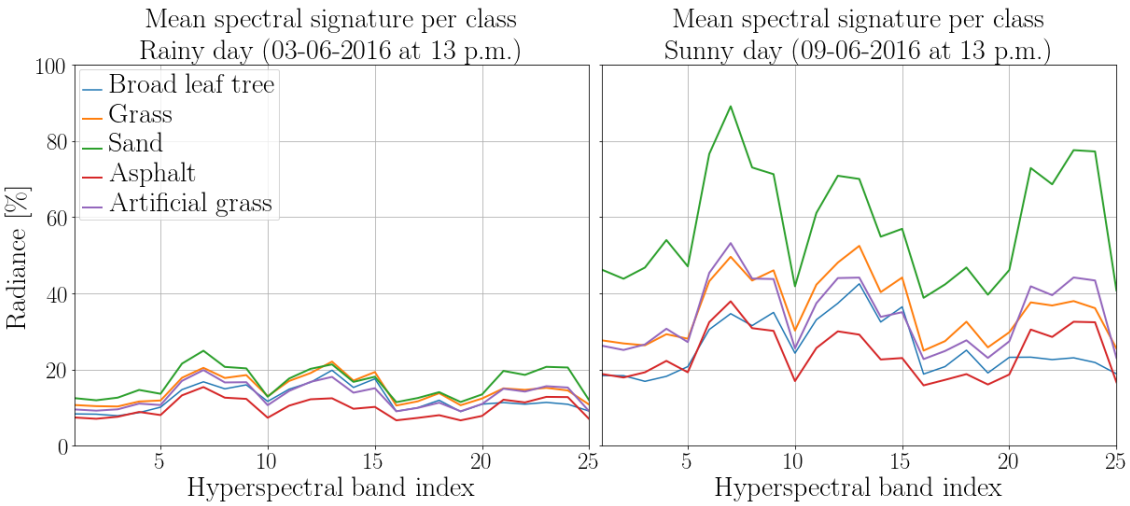

Figure 6-3: Typical shapes of spectral signatures, mean of annotated pixels per class.

When comparing training data sets under a single and multiple weather conditions, differences in mean μ and standard deviation σ between both sets become clear (table 6-1). Due to multiple weather conditions, the spectral signatures have a large range of variation, as indicated by σ. Furthermore, the mean value μ for the training data set comprising multiple weather types is lower (due to lower spectral intensities from cloudy days).

Training data set *Weather type*		**Class** *Broad leaf Tree*	*Grass*	*Sand*	*Asphalt*	*Artificial Grass*
Single *Sunny*	μ	1100	1419	2081	1002	1232
	σ	172	211	306	148	288
Multiple	μ	891	1136	1603	813	1002
	σ	404	526	808	371	497

Table 6-1: Mean μ and standard deviation σ for training data sets subject to a single and multiple weather conditions (experiments for a single week in June).

In order to gain insight about the generalizing abilities of a network trained on a single week in June (subject to a single weather condition), test images recorded in April, May, June, July and August are segmented (figure 6-4). Accuracy ranges between $36 - 97\%$ (appendix E-2, table E-1). It becomes clear that a network trained with data from June, also performs well in July and August. However, performance in May is low and degrades even more in April. Degradation of results in extrapolation can partly be explained by a lower level of chlorophyll in vegetation, which has large effect on the spectral signature shape. Vegetation (leafs) in June contain high levels of chlorophyll, which is what the network has been subject to during training. Observation of test images from April and May reveals that performance on non-vegetation classes (asphalt and sand) remains similar to a test image from June (present F-scores). Therefore, in order to increase the period in which a single model is valid, additional training data is required from this period of time.

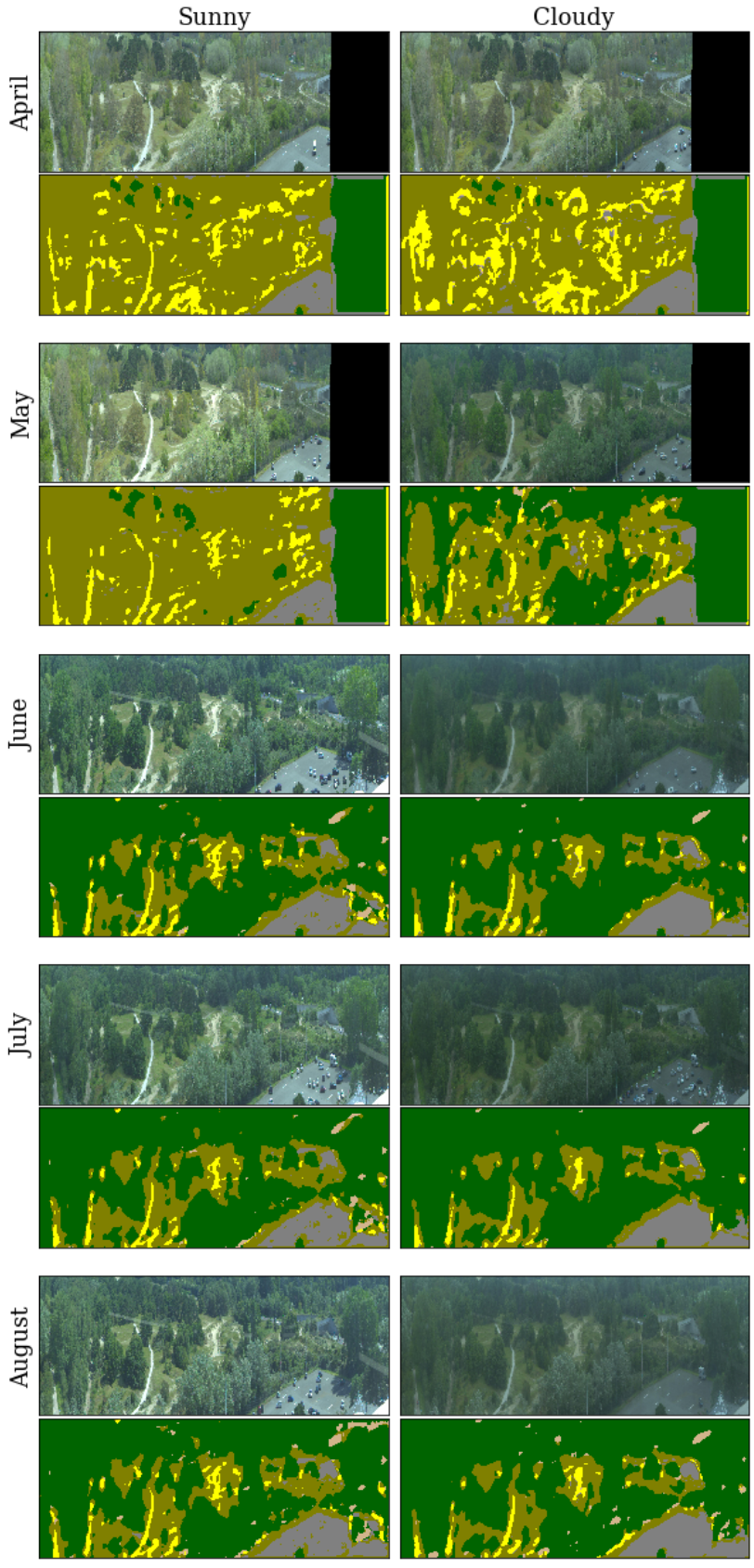

Figure 6-4: Extrapolation beyond the training data set; generated using U-Net trained on data from a single week in June (sunny weather conditions). Accuracy ranges between $36 - 97\%$. Mean F-scores range between $0.29 - 0.96$. All values for accuracy A, F-score per class and mean F-score $\bar{F}$ in appendix E-2, table E-1.

6-1-2 A Single Week in July

The HS data for this experiment was taken between 01-07-2016 and 07-07-2016. Furthermore the annotation mask which was also used in experiments for May and June data, was used in the July-experiments. However, it became clear that a new annotation mask (*July new*) is required for artificial grass, as broad leaf trees have grown to pixels which have been annotated as artificial grass. Subsequently, the training data set includes a large number of pixels of broad leaf tree which are annotated as artificial grass, which is clearly visible in segmented test images (appendix E-3, figure E-4). Test images show there is artificial grass - broad leaf tree confusion especially. Since artificial grass is a rare class and the training data set is strongly oversampled on this class, it has large effect on the segmentation results.

This effect is observed in the mean spectral signature of all pixels annotated as artificial grass (figure 6-5). Note that the broad leaf tree class is not affected by using the new mask, as it has been adapted for artificial grass pixels only. For artificial grass, not all HS bands are equally affected. For example band numbers 16, 19 and 25 appear not to be influenced by the new mask. However band numbers 7, 8, 22 and 23 are very sensitive to incorrect annotation. Mean μ and standard deviation σ of these bands of artificial grass indicate the new mask contains less incorrectly annotated pixels (table 6-2).

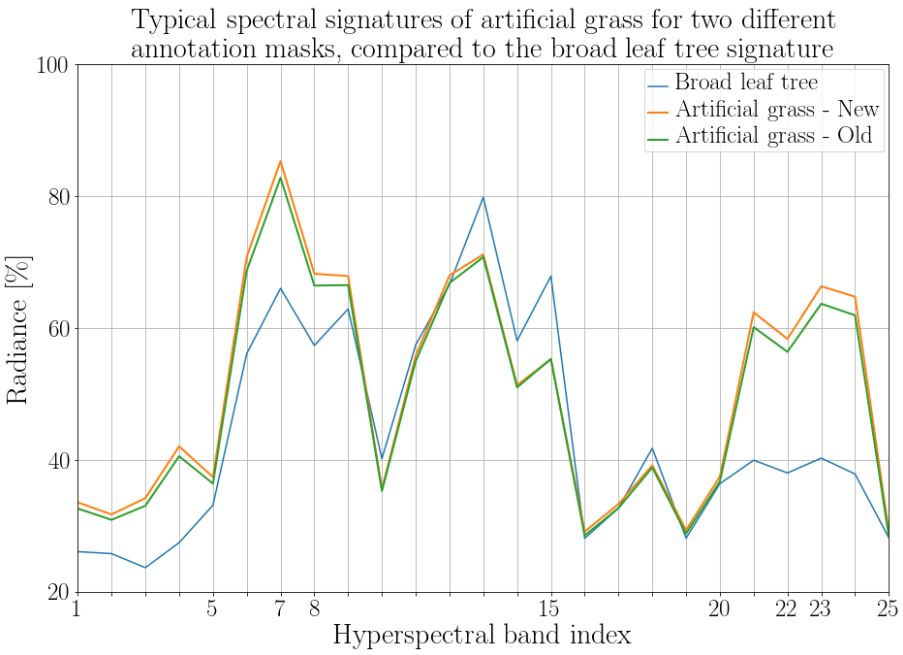

Figure 6-5: Mean spectral signatures of annotated pixels for two different masks (image from 07-07-2016 at 15 p.m.).

Applying the new mask, a network trained with data under similar weather conditions yields results with an overall accuracy ranging between $87-96\%$ for a sunny test image. For a rainy test images results range between $93-96\%$ (both in appendix E-3, figure E-5).

Furthermore, a network trained with data under multiple weather conditions did not yield accurate results (appendix E-3, figure E-3). Effects due to weather are similar to experiments

Mask		**μ and σ of selected band numbers of artificial grass spectral signature**				
		7	*8*	*22*	*23*	*25*
Initial	μ	1336	1113	971	1067	591
	σ	230	186	165	192	95
New July	μ	1395	1156	1018	1129	609
	σ	205	172	151	172	95

Table 6-2: Mean μ and standard deviation σ for several HS channels of pixels annotated as artificial grass (image 07-07-2016 at 15 p.m.) for two different annotation masks. Band 25 for comparison.

carried out using June data. Test results regarding extrapolating capabilities are very similar to experiments from June data as well (appendix E-3, figure E-6).

6-1-3 Remaining results

A Single Week in May

The HS data for this experiment was taken between 27-04-2016 and 04-05-2016. This week falls just before the start of growing season. Visually, there is a large variation in of colors (buds on the trees turn green at different times, grass color is pale) and structures (tree trunks, branches are visible etc.) compared to data recorded in summer months. Comparing the mean μ and standard deviation σ from training data set from May and June support this (table 6-3)

Training data set *Single weather type*		**Class** *Broad leaf Tree*	*Grass*	*Sand*	*Asphalt*	*Artificial Grass*
May	μ	723	1028	1404	643	NA
	σ	328	455	668	277	NA
June	μ	1100	1419	2081	1002	1232
	σ	172	211	306	148	288

Table 6-3: Mean μ and standard deviation σ for training data sets from a single week in May and June (single -*sunny*- weather type).

A network trained using data under a single weather condition yields results with an overall accuracy ranging between $75 - 78\%$ for a sunny test image. For a rainy test images results range between $66 - 79\%$ (both in appendix E-1, figure E-1). These are low accuracy scores, which translate to many misclassified pixels of certain classes. Especially sand - asphalt confusion results in performance degradation. Results for a network trained under multiple weather conditions are similar. Segmentation accuracy of test images obtained with this training data set are clearly lower than segmentations obtained with networks trained on data from summer months (section 6-1-1 to 6-1-3).

A Single Week in August

The HS data for this experiment was taken between 10-08-2016 and 16-08-2016. Results deviate from previous experiments concerning a week in June and a week in July, as there are

many pixels misclassified as artificial grass for distant objects. An adapted annotation mask (*August New*) was used to fit the training data, as broad leaf trees might have grown to pixels annotated as artificial grass. However, it did not fully solve the problem. Also mean μ and standard deviation σ of the HS bands of artificial grass in the training data set (table 6-4) do not show clear improvement, as standard deviation σ does not significantly decrease for every band. Possibly, the misclassification error has another cause, for example the spectral signatures might be affected by humidity due to high August temperatures (especially distant objects will be affected).

Mask		**μ and σ of selected band numbers of artificial grass spectral signature**				
		7	*8*	*22*	*23*	*25*
Initial	μ	1336	1113	971	1067	591
	σ	230	186	165	192	95
July New	μ	1395	1156	1018	1129	609
	σ	205	172	151	172	95
August New	μ	1309	1078	976	1091	560
	σ	193	155	150	173	81

Table 6-4: Mean μ and standard deviation σ of several HS bands of pixels annotated as artificial grass. Values generated from training data sets using the initial mask, a masked adapted to fit images from July and August respectively.

A network trained using data under similar weather conditions yields results with an overall accuracy ranging between $88 - 92\%$ for a sunny test image. For a rainy test images results range between $77 - 86\%$ (appendix E-4, figure E-7).

6-2 Interpolating Between Days in Four Weeks

In this set of experiments training and testing have been carried out from data taken over a period of approximately four weeks. Training, validation and test data sets used in the following experiments are listed in appendix C-2-2. Experiments are carried out for May to June and July to August, segmented test images are displayed in appendices E-5 and E-6 respectively.

6-2-1 May-June

The HS data for this experiment was taken between 03-05-2016 and 10-06-2016. This period falls right into growing season, hence there is large variation in appearance of the scene. The test set image with the highest accuracy is 87% with an F-score of 0.55, obtained with a training data set subject to sunny weather conditions (appendix E-5, figure E-9). Test images generated with a test set under multiple weather conditions achieve a lower maximum score, namely 72% accuracy with an F-score of 0.38 (appendix E-5, figure E-11).

6-2-2 July-August

The HS data for this experiment was taken between 01-07-2016 and 16-08-2016. This period in summer has no large changes regarding vegetation, hence images are similar. The maximum score for the training data set under sunny weather conditions is 86% with an F-score of 0.58 (appendix E-6, figure E-13). The maximum score for the training data set under multiple weather conditions is 81% with an F-score of 0.38 (appendix E-6, figure E-15).

Hence these experiments support the findings that the standard U-Net architecture is unable to generalize highly varying data, as the network does not converge for training data sets under multiple weather or seasonal conditions. However, in order to extend the period in which U-Net yields accurate results, the network should be able to process diverse training data.

6-3 Interpolation Over a Longer Period of Time

To increasing the period in which the trained network is valid, a training data set is taken from a longer period of time. However, training a network using data from a longer period of time is not straightforward (sections 6-2-1 and 6-2-2). Also, experiments using multiple weather types for training (section 6-1) show inaccurate results. Training data sets with a large variation of spectral signatures show low performance, great variation within a training data set contributes to the increasing complexity of the features to be learned. Therefore, features should have incorporated information regarding weather or season. Then the network will be able to generalize over a longer period of time. Increasing generalizability has been researched according to two different approaches; normalizing the raw input data (appendix F-1) and rearranging U-Net weights (appendix F-2).

Finally, rearranging the U-Net weights (appendix F-2, figure F-2) and altering its hyperparameters proved to be an effective way to increase generalizability. The customized U-Net has a different number of feature maps per convolutional layer, keeping the total number of weights roughly the same (appendix F-2, table F-1). The general U-Net architecture remains the same, only the number of weights per network layer has been adapted. The new weights are structured in such a way that it extracts more informative lower-level features from the HS data. Subsequently, these features are used as a basis for higher-level feature-maps, creating a bottom-up effect.

Using the customized U-Net architecture, experiments are done using training data including several types of weather (section 6-3-1). Moreover, the altered network is able to handle a training data set taken over a larger period of time including several seasons (section 6-3-2).

6-3-1 Multiple Weather Type Training

Experiments presented in section 6-1 show accurate results are achieved only when using a single weather type, i.e. sunny weather. Therefore, the goal is to adjust the architecture so that it is able to handle patches with larger variability. The customized U-net architecture is altered to extract more informative features from all HS bands in order to increase generalizability of classes under large variability caused by a range of ambient conditions. The

customized U-Net is capable of yielding accurate results for a training data set subject to multiple weather types (figure 6-6). The original U-Net achieves 70% accuracy on a rainy test image, whereas the customized U-Net yields 93%. The difference for sunny test images is slightly smaller.

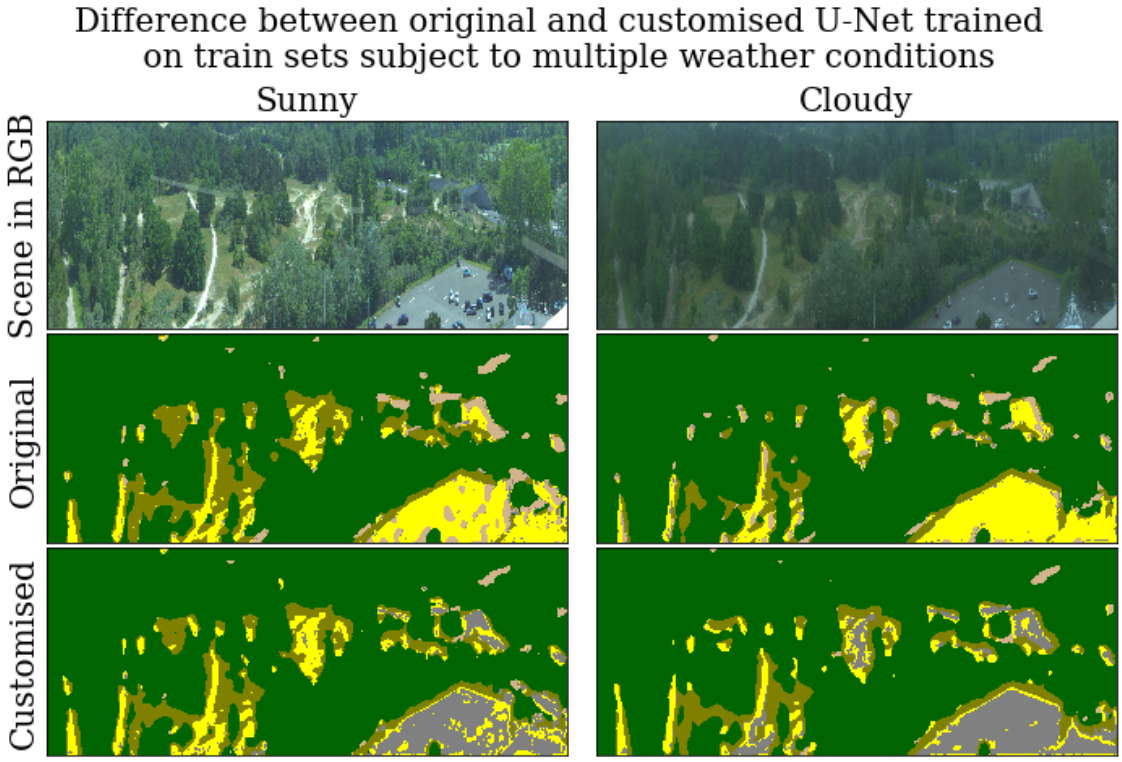

Figure 6-6: Networks trained on multiple weather conditions. The original U-Net yields $> 70\%$ accuracy on both sunny (77% accuracy, 0.45 mean F-score) and rainy (70% accuracy, 0.38 mean F-score) test images. The customized U-Net yields $> 93\%$ accuracy on both sunny (94% accuracy, 0.79 mean F-score) and rainy (93% accuracy, 0.75 mean F-score) test images.

Thus, the customized network performs better on a training data set under large variability (due to multiple weather conditions) than the original U-Net. However, the results show slightly lower performance than training the original U-Net with a training data set comprising sunny images only. Both accuracy and F-score are lower, as the pixelwise classified test images show some sand-asphalt confusion. This type of confusion specifically occurs in the parking area at locations with parked cars. These results obtained with this complex training data set gives rise to training a network with a training data set subject to multiple seasons.

6-3-2 Multiple Season Training

The following experiment is carried out under a single weather condition (sunny) using the customized network. The training and test data set comprise patches from days in April to August. Segmented test images show that the network is able to generalize over this period of time (figure 6-7), yielding a maximum accuracy of 93%. Irrespective of weather conditions

the network yields robust segmentations over time, as the accuracies range between 87 and 93% (appendix E-7, table E-2).

Segmented test images under cloudy or rainy conditions are less accurate, especially due to sand-asphalt and grass-asphalt confusion. This can be explained by the content of the training data set, as it does not include images subject to rainy conditions. None of the segmented images show classification of artificial grass, also indicated by a relatively low F-score.

In order to assess generalizing properties beyond the training data set, the customized U-Net is used to yield segmentations for test images taken in October and December (appendix E-7, figure E-16). Note that the scene is slightly different as the camera angle has changed and painted markings have been added to the parking area. Accuracy of both test images have dropped significantly to 65% in October and 54% in December. The training data set did not include patches subject to the specific atmospheric conditions of these months. In both images non-natural material (asphalt, sand and artificial grass) is classified as asphalt. Natural materials are classified as broad leaf tree.

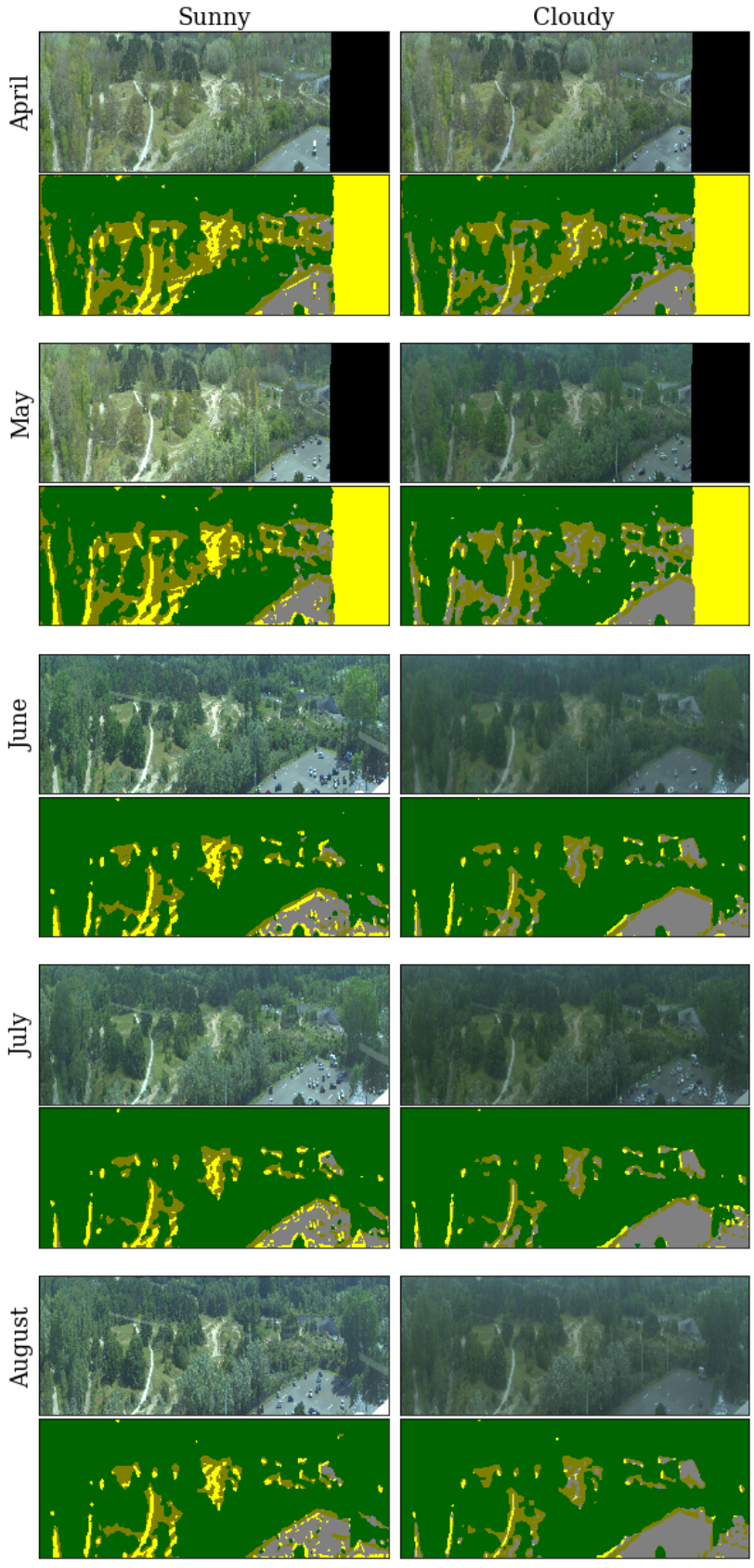

Figure 6-7: The customized U-Net is able to generalize over a longer period of time, i.e. April to August. The segmented test images (all recorded at 13 p.m.) are subject to multiple weather conditions. Accuracies range between 87 and 93%. Mean F-scores range between 0.55 and 0.69. All values for accuracy A, F-score per class and mean F-score $\bar{F}$ in appendix E-7, table E-2.

Chapter 7

Conclusions and Recommendations

The TNO hyperspectral (HS) data set comprises images of a fixed scene recorded over the course of a year. So, the images are subject to a range of lighting, weather and seasonal conditions. Atmospheric effects, such as illumination and humidity, disturb the spectral information. This thesis addresses accurate pixelwise segmentation of HS images irrespective of recording date, using a Fully Convolutional Network (FCN) (U-Net [10]). These networks are known for their generalizing properties regarding image segmentation problems. Limitations of training U-Net to generalize for the temporally changing ambient conditions are explored. The network has learned to define features based on both local spectral and spatial information to yield accurate classifications for every pixel. No restoring atmospheric model is defined explicitly, the model learns to correct for lighting, weather and seasonal conditions implicitly.

Conclusions

For the ground truth required for training a neural network, a sparse annotation mask has been developed. This mask fits images over the entire recording period. Using such a mask is time-saving, allows for changing object borders (due to growing season) and avoids inclusion of mixed pixels.

Besides the annotation mask, a patch-wise training method has been used. In this method, many small patches are taken from the original images to form the training data set, instead of taking the full ($1200 \times 3033 \times 25$) image dimensions. It reduces the computational load. Patch-wise training in combination with a sparse mask results in a network that is able to generalize beyond local information given by the training patches. Network performance is assessed with accuracy metrics based on the annotated pixels only. Unmasked pixels show correct classification, as the segmented test images suit with the Red-Green-Blue (RGB) images which were made for comparison.

The sparse annotation mask has been investigated qualitatively, as it influences the level of detail attained in the segmented test images. Increasing the level of detail is achieved by

annotating at least two neighboring classes densely, however this is at odds with avoiding inclusion of mixed pixels. Furthermore, the problem of classifying extremely rare classes has been solved by using an oversampling technique. The choice of the oversampling factor and the sensitivity of the annotation mask have been investigated, as well as the effects of undersampling the most common class.

The U-net architecture has been adapted in order to extend the period of time for which the network yields accurate results. This customized U-Net architecture contains slightly less network weights compared to the original architecture. The original U-Net can be trained on a simple training data set only, so that it yields accuracies of $> 95\%$ for a limited period of time. The customized U-Net is able to handle more complex training data sets, as it extracts more informative features from the HS training patches. This network is able to yield accurate segmentations over a period of April to August. However, accuracy is lower compared to the original U-Net, as the network yields accuracies of $> 87\%$ for all test images within this period of time.

Concluding, convolutional neural networks are capable of generalizing HS images under varying lighting, weather and seasonal conditions. Pre-processing techniques such as image normalization or atmospheric correction have not been used. Training a network with local spectral-spatial information only, yields a model which is able to segment images over a longer period of time irrespective of season or weather. In this application, neural network design is a trade-off between segmentation accuracy and duration of model validity, which is controlled by network weight arrangement. The design is dependent on the goal of the segmentation task.

Recommendations

In future work, a cross entropy loss function could be implemented instead of a quadratic loss function, minimizing the error between the ground truth y_t and masked predictions $y_{p,m}$. For this loss function, only the predictions y_p of unlabeled pixels should be masked. The cross entropy loss function fits the goal of estimating a single class per pixel better, as the final softmax activation function minimizes the cross-entropy between the estimated class probabilities and the *true* distribution.

Furthermore, network performance can be increased by adding information to training data set. This additional information makes uses of physical characteristics of objects. Specific spectral bands are combined to form a discriminative index. For example, the *normalized difference vegetation index* discriminates vegetation from non-vegetation.

Further research could also address the effects of increasing the number of training samples subject a wider range of weather conditions. This possibly enhances segmentation performance when training a network using a training data set subject to multiple weather conditions.

Appendix A

Annotation Mask

The annotation mask used for hyperspectral (HS) images over a period of a several months. Figures A-1 to A-4 show the mask covering RGB images of the scene in May, June, July and August. The same mask is used for all months, except for the artificial grass class in July and August. Annotated pixels for this class have been modified in order to compensate for growing vegetation. Note that parked cars are annotated as asphalt.

Figure A-1: RGB image taken in May (01-05) covered by the annotation mask.

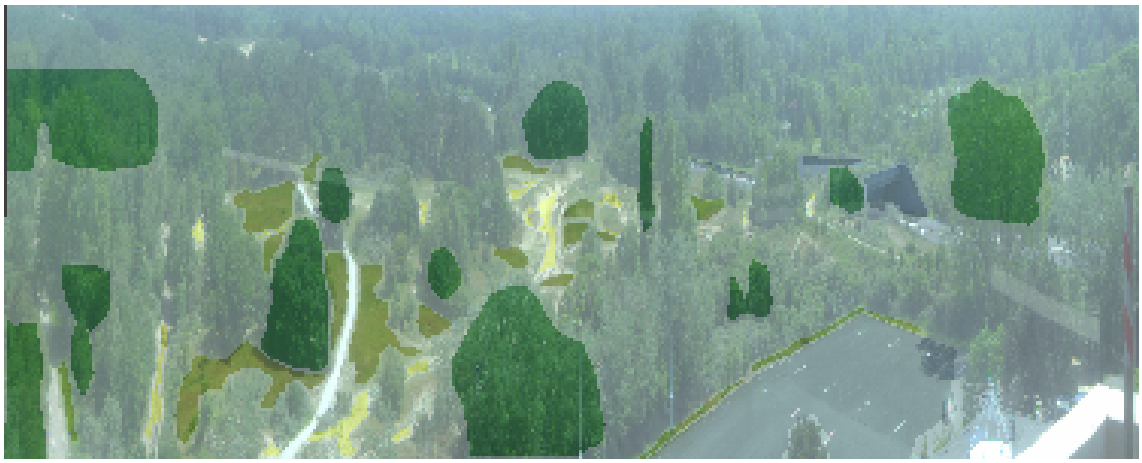

Figure A-2: RGB image taken in June (05-06) covered by the annotation mask.

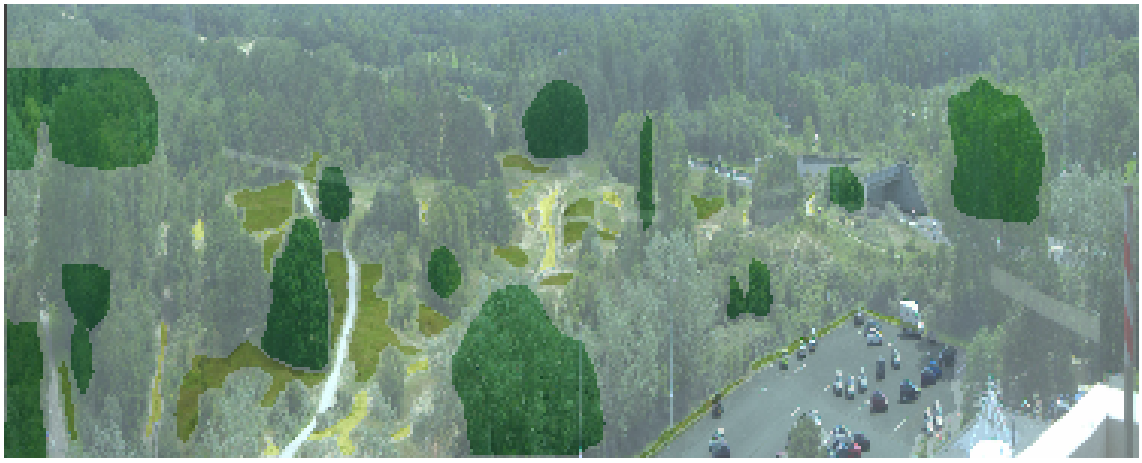

Figure A-3: RGB image taken in July (06-07) covered by the annotation mask.

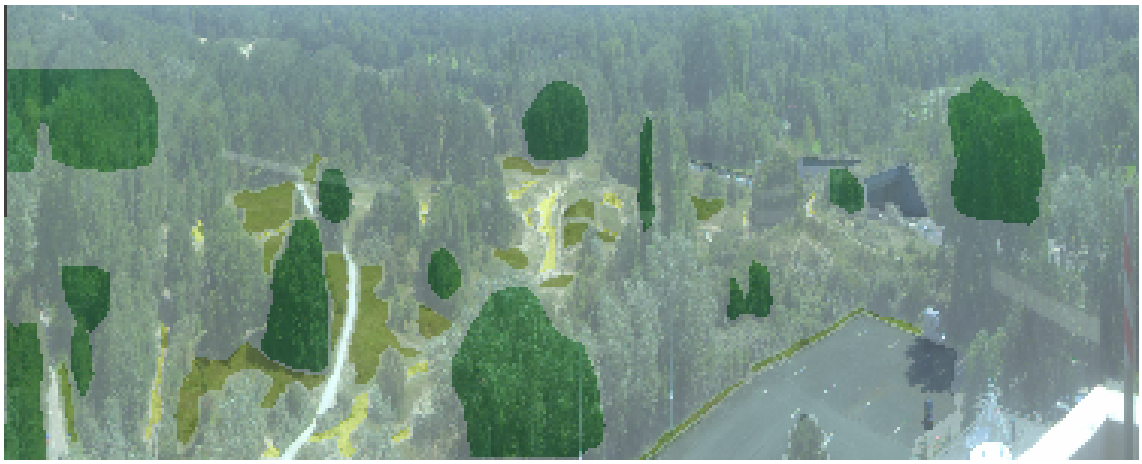

Figure A-4: RGB image taken in August (14-08) covered by the annotation mask.

Appendix B

Python Code

B-1 U-Net Architecture

```
1  def make_unet_model(optimizer, img_rows, img_cols, classes, dropoutrate,
       weights=None):
2  win = 'glorot_normal'
3  ac = 'elu'
4
5  inputs = Input((25, img_rows, img_cols))
6  conv1 = BatchNormalization(axis=1, epsilon=0.001)(inputs)
7  conv1 = Convolution2D(32, 3, 3, activation=ac, border_mode='same', init=
       win)(conv1)
8  conv1 = Convolution2D(32, 3, 3, activation=ac, border_mode='same', init=
       win)(conv1)
9  conv1 = Dropout(dropoutrate)(conv1)
10
11 conv2 = MaxPooling2D(pool_size=(2,2))(conv1)
12 conv2 = Convolution2D(64, 3, 3, activation=ac, border_mode='same', init=
       win)(conv2)
13 conv2 = Convolution2D(64, 3, 3, activation=ac, border_mode='same', init=
       win)(conv2)
14 conv2 = Dropout(dropoutrate)(conv2)
15
16 conv3 = MaxPooling2D(pool_size=(2,2))(conv2)
17 conv3 = Convolution2D(128, 3, 3, activation=ac, border_mode='same', init=
       win)(conv3)
18 conv3 = Convolution2D(128, 3, 3, activation=ac, border_mode='same', init=
       win)(conv3)
19 conv3 = Dropout(dropoutrate)(conv3)
20
21 conv4 = MaxPooling2D(pool_size=(2,2))(conv3)
22 conv4 = Convolution2D(256, 3, 3, activation=ac, border_mode='same', init=
       win)(conv4)
```

```
23 conv4 = Convolution2D(256, 3, 3, activation=ac, border_mode='same', init=
       win)(conv4)
24 conv4 = Dropout(dropoutrate)(conv4)
25
26 conv5 = MaxPooling2D(pool_size=(2,2))(conv4)
27 conv5 = Convolution2D(512, 3, 3, activation=ac, border_mode='same', init=
       win)(conv5)
28 conv5 = Convolution2D(512, 3, 3, activation=ac, border_mode='same', init=
       win)(conv5)
29 conv5 = Dropout(dropoutrate)(conv5)
30
31 up6 = UpSampling2D(size=(2,2))(conv5)
32 conv6 = merge([up6, conv4], mode='concat', concat_axis=1)
33 conv6 = Dropout(dropoutrate)(conv6)
34 conv6 = Convolution2D(256, 3, 3, activation=ac, border_mode='same', init=
       win)(conv6)
35 conv6 = Convolution2D(256, 3, 3, activation=ac, border_mode='same', init=
       win)(conv6)
36
37 up7 = UpSampling2D(size=(2,2))(conv6)
38 conv7 = merge([up7, conv3], mode='concat', concat_axis=1)
39 conv7 = Dropout(dropoutrate)(conv7)
40 conv7 = Convolution2D(128, 3, 3, activation=ac, border_mode='same', init=
       win)(conv7)
41 conv7 = Convolution2D(128, 3, 3, activation=ac, border_mode='same', init=
       win)(conv7)
42
43 up8 = UpSampling2D(size=(2,2))(conv7)
44 conv8 = merge([up8, conv2], mode='concat', concat_axis=1)
45 conv8 = Dropout(dropoutrate)(conv8)
46 conv8 = Convolution2D(64, 3, 3, activation=ac, border_mode='same', init=
       win)(conv8)
47 conv8 = Convolution2D(64, 3, 3, activation=ac, border_mode='same', init=
       win)(conv8)
48
49 up9 = UpSampling2D(size=(2,2))(conv8)
50 conv9 = merge([up9, conv1], mode='concat', concat_axis=1)
51 conv9 = Dropout(dropoutrate)(conv9)
52 conv9 = Convolution2D(32, 3, 3, activation=ac, border_mode='same', init=
       win)(conv9)
53 conv9 = Convolution2D(32, 3, 3, activation=ac, border_mode='same', init=
       win)(conv9)
54
55 conv11 = Convolution2D(len(classes), 1, 1, activation=ac, border_mode='
       same', init=win)(conv9)
56 conv11 = core.Reshape((len(classes),img_rows*img_cols))(conv11)
57 conv11 = core.Permute((2,1))(conv11)
58
59 conv12 = core.Activation('softmax')(conv11)
60
61 conv13 = core.Permute((2,1))(conv12)
62 conv13 = core.Reshape((len(classes),img_rows,img_cols))(conv13)
63
```

```
64 model = Model(input=inputs, output=conv13)
65
66 model.compile(optimizer=optimizer, loss=custom_loss(weights), metrics = [
       fscore, overall_acc])
67
68 return model
```

B-2 Loss Function

```
1  from keras import backend as K
2
3  def custom_loss(weights=None):
4  def custom_loss_value(y_true, y_pred):
5  if weights is not None:
6  weights_tensor = np.array(weights)[:, np.newaxis, np.newaxis]
7  w_tensor = weights_tensor * K.ones_like(y_true)
8  cover_pred = y_true * y_pred
9  minimize =  (y_true - cover_pred)*w_tensor
10 else:
11 cover_pred = y_true * y_pred
12 minimize = y_true - cover_pred
13
14 loss = K.mean(K.square(minimize))
15 return loss
16
17 return custom_loss_value
```

B-3 Accuracy Metrics

```
1  from keras import backend as K
2
3  def fscore(y_true, y_pred):
4  y_true_2D = K.max(y_true, axis=1, keepdims=False)
5  y_pred_2D = K.max(y_true*y_pred, axis=1, keepdims=False)
6
7  smooth = 1.
8  y_true_f = K.flatten(y_true_2D)
9  y_pred_f = K.flatten(y_pred_2D)
10 intersection = K.sum(y_true_f * y_pred_f)
11 f = (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) +
       smooth)
12
13 return f
```

```
from keras import backend as K

def overall_acc(y_true, y_pred):
y_true_2D = K.max(y_true, axis=1, keepdims=False)
y_pred_2D = K.max(y_true*y_pred, axis=1, keepdims=False)

y_true_f = K.sum(K.flatten(y_true_2D))
y_pred_f = K.sum(K.flatten(y_pred_2D))

acc = y_pred_f / (y_true_f + 0.0001)

return acc
```

B-4 Patch lists

```
def load_data_description(img_rows, img_cols,nr_of_patches, video_days,
    video_names, ann_orig, data_dir,classes, dir_exp, data_name,splitratio
    =0.2):

patch_list = []
camera = 'VIS'
for video_day in video_days:
for video_name in video_names:
video = 'hsi_rc_{}_{}.h5'.format(video_day, video_name) #Name video
img_path = os.path.join(data_dir, video) #Path to video
f = h5py.File(img_path)
dataset_name = [key for key in f.keys()][0]
[x_hsi, y_hsi, channels_hsi] = f[dataset_name].shape
f.close()

# random selection of coordinates top left point of patch
[row_max, col_max] = ann_orig.shape
row_max_sub = row_max - img_rows
col_max_sub = col_max - img_cols

for nr in range(0,nr_of_patches):
row_rand = random.randrange(0,row_max_sub,1)
col_rand = random.randrange(0,col_max_sub,1)
row = row_rand
col = col_rand
# only select patches if at least one class is present
if len(set(classes).intersection(set(np.unique(ann_orig[row:row +
    img_rows, col:col + img_cols])))) > 0:

classx = [int(x) for x in set(np.unique(ann_orig[row:row + img_rows, col:
    col + img_cols])) if x!=0]
patch_list += [{
  'col_left': col,
  'row_top': row,
```

```
31      'width': img_cols,
32      'height': img_rows,
33      'video_day': video_day,
34      'video_hour': video_name,
35      'class': classx
36  }]
37
38  # shuffle list and make list dataframe
39  random.seed(1000)
40  patch_list = random.sample(patch_list, len(patch_list))
41  patch_list_df = pd.DataFrame(patch_list)
42
43  # split list in train and validation set
44  split = int(splitratio*len(patch_list))
45  patch_list_val = patch_list_df[:split]
46  patch_list_train = patch_list_df[split:]
47
48  return patch_list_train, patch_list_val, patch_list_df
```

Appendix C

Experiment Details

C-1 Experiments Train Set Optimization

C-1-1 Number of training patches

Composition of the training data sets used:

Class	Pixels in training data set [%]
Broad leaf Tree	59,52
Grass	14,92
Sand	3,20
Asphalt	22,29
Artificial Grass	0,07

Table C-1: Number of training patches is 10805 ($n_p = 1000$, $f_o = 0$, $f_u = 0$)

Class	Pixels in training data set [%]
Broad leaf Tree	57,85
Grass	15,31
Sand	3,46
Asphalt	23,30
Artificial Grass	0,08

Table C-2: Number of training patches is 16189 ($n_p = 1500$, $f_o = 0$, $f_u = 0$)

C-1-2 Oversampling

Composition of the training data sets used, oversampled on artificial grass:

Class	Pixels in training data set [%]
Broad leaf tree	58,55
Grass	14,99
Sand	3,44
Asphalt	22,37
Artificial grass	0,66

Table C-3: Number of training patches is 11735 ($f_o = 10$ and $n_p = 1000$)

Class	Pixels in training data set [%]
Broad leaf tree	56,65
Grass	14,14
Sand	3,15
Asphalt	22,47
Artificial grass	3,59

Table C-4: Number of training patches is 16399 ($f_o = 70$ and $n_p = 1000$)

Class	Pixels in training data set [%]
Broad leaf tree	58,22
Grass	15,30
Sand	3,26
Asphalt	22,53
Artificial grass	0,69

Table C-5: Number of training patches is 17328 ($f_o = 10$ and $n_p = 1500$)

Class	Pixels in training data set [%]
Broad leaf tree	57,04
Grass	14,43
Sand	3,13
Asphalt	21,97
Artificial grass	3,43

Table C-6: Number of training patches is 24410 ($f_o = 70$ and $n_p = 1500$)

C-1-3 Undersampling

Composition of the training data sets used, undersampling applied on broad leaf tree:

Class	Pixels in training data set [%]
Broad leaf tree	40,53
Grass	20,79
Sand	4,95
Asphalt	33,62
Artificial grass	0,11

Table C-7: Number of training patches is 7815 ($f_u = 0.6$ and $n_p = 1000$)

Class	Pixels in training data set [%]
Broad leaf tree	27,34
Grass	27,13
Sand	5,57
Asphalt	39,81
Artificial grass	0,15

Table C-8: Number of training patches is 6880 ($f_u = 0.8$ and $n_p = 1000$)

Class	Pixels in training data set [%]
Broad leaf tree	38,85
Grass	22,32
Sand	4,77
Asphalt	33,94
Artificial grass	0,12

Table C-9: Number of training patches is 11840 ($f_u = 0.6$ and $n_p = 1500$)

Class	Pixels in training data set [%]
Broad leaf tree	27,23
Grass	26,45
Sand	5,51
Asphalt	40,68
Artificial grass	0,13

Table C-10: Number of training patches is 10388 ($f_u = 0.8$ and $n_p = 1500$)

C-1-4 Combined Over- and Undersampling

Composition of the training data sets used:

Class	Pixels in training data set [%]
Broad leaf tree	36,64
Grass	21,33
Sand	4,46
Asphalt	32,53
Artificial grass	5,04

Table C-11: Combined over- and under-sampling; $n_p = 1000$, $f_o = 70$ and $f_u = 0.7$.

Class	Pixels in training data set [%]
Broad leaf tree	38,48
Grass	22.29
Sand	4,70
Asphalt	33,66
Artificial grass	0,88

Table C-12: Combined over- and under-sampling; $n_p = 1000$, $f_o = 10$ and $f_u = 0.7$.

C-2 Experiments for temporally changing conditions

C-2-1 Interpolation over a week

Images in training (T), validation (V) and test data set end of April, beginning of May 2016 (tables C-13 and C-14).

Date	Hours	Data Set	Weather
28-04	11-17	T+V	Partly Cloudy
29-04	13	Test	Cloudy
01-05	13	Test	Sunny
02-05	11-17	T+V	Sunny
03-05	11-17	T+V	Sunny
04-05	11-17	T+V	Sunny

Table C-13: Interpolating one week in April and May, similar weather conditions.

Date	Hours	Data Set	Weather
27-04	11-17	T+V	Mostly Cloudy
28-04	11-17	T+V	Partly Cloudy
29-04	13	Test	Cloudy
01-05	13	Test	Sunny
02-05	11-17	T+V	Sunny
04-05	11-17	T+V	Sunny

Table C-14: Interpolating one week April and May, multiple weather conditions.

Images in training (T), validation (V) and test data set June 2016 (tables C-15 and C-16).

Date	Hours	Data Set	Weather
03-06	13	Test	Foggy
04-06	11-17	T+V	Sunny
05-06	11-17	T+V	Sunny
06-06	11-17	T+V	Sunny
09-06	13	Test	Sunny
10-06	11-17	T+V	Sunny

Table C-15: Interpolating one week June, similar weather conditions.

Date	Hours	Data Set	Weather
02-06	11-17	T+V	Rainy
03-06	13	Test	Foggy
04-06	11-17	T+V	Sunny
05-06	11-17	T+V	Sunny
09-06	13	Test	Sunny
10-06	11-17	T+V	Sunny

Table C-16: Interpolating one week June, multiple weather conditions.

Images in training (T), validation (V) and test data set July 2016 (tables C-17 and C-18).

Date	Hours	Data Set	Weather
02-07	11-17	T+V	Mostly sunny
03-07	13	Test	Mostly sunny
04-07	11-17	T+V	Mostly sunny
05-07	13	Test	Cloudy (dark)
06-07	11-17	T+V	Sunny
07-07	11-17	T+V	Sunny

Table C-17: Interpolating one week July, similar weather conditions.

Date	Hours	Data Set	Weather
01-07	11-17	T+V	Foggy (dark)
02-07	11-17	T+V	Mostly sunny
03-07	13	Test	Mostly sunny
04-07	11-17	T+V	Mostly sunny
05-07	13	Test	Cloudy (dark)
07-07	11-17	T+V	Sunny

Table C-18: Interpolating one week July, multiple weather conditions.

Images in training (T), validation (V) and test data set August 2016 (tables C-19 and C-20).

Date	Hours	Data Set	Weather
10-08	11-17	T+V	Mostly sunny
12-08	11-17	T+V	Cloudy
13-08	13	Test	Cloudy
14-08	11-17	T+V	Sunny
15-08	13	Test	Sunny
16-08	11-17	T+V	Sunny

Table C-19: Interpolating one week August, similar weather conditions.

Date	Hours	Data Set	Weather
10-08	11-17	T+V	Mostly sunny
11-08	11-17	T+V	Rainy
12-08	11-17	T+V	Cloudy
13-08	13	Test	Cloudy
15-08	13	Test	Sunny
16-08	11-17	T+V	Sunny

Table C-20: Interpolating one week August, multiple weather conditions.

C-2-2 Interpolation over four weeks

Images in training (T), validation (V) and test data set May-June 2016 (tables C-21 and C-22).

Date	Hours	Data Set	Weather
03-05	13	Test	Partly Cloudy
04-05	11,13,15,17	T+V	Sunny
05-05	11,13,15,17	T+V	Sunny
06-05	13	Test	Sunny
08-05	11,13,15,17	T+V	Sunny
09-05	11,13,15,17	T+V	Sunny
03-06	13	Test	Foggy
04-06	11,13,15,17	T+V	Sunny
05-06	11,13,15,17	T+V	Sunny
06-06	11,13,15,17	T+V	Sunny
09-06	13	Test	Sunny
10-06	11,13,15,17	T+V	Sunny

Table C-21: Interpolating May-June, similar weather conditions.

Date	Hours	Data Set	Weather
03-05	13	Test	Partly Cloudy
04-05	11,13,15,17	T+V	Sunny
05-05	11,13,15,17	T+V	Sunny
06-05	13	Test	Sunny
09-05	11,13,15,17	T+V	Sunny
10-05	11,13,15,17	T+V	Foggy (dark)
02-06	11,13,15,17	T+V	Rainy
03-06	13	Test	Foggy
04-06	11,13,15,17	T+V	Sunny
05-06	11,13,15,17	T+V	Sunny
09-06	13	Test	Sunny
10-06	11,13,15,17	T+V	Sunny

Table C-22: Interpolating May-June, multiple weather conditions.

Images in training (T), validation (V) and test data set July-August 2016 (tables C-23 and C-24).

Date	Hours	Data Set	Weather
02-07	11,13,15,17	T+V	Mostly sunny
03-07	13	Test	Mostly sunny
04-07	11,13,15,17	T+V	Mostly sunny
05-07	13	Test	Cloudy (dark)
06-07	11,13,15,17	T+V	Sunny
07-07	11,13,15,17	T+V	Sunny
10-08	11,13,15,17	T+V	Mostly sunny
12-08	11,13,15,17	T+V	Cloudy
13-08	13	Test	Cloudy
14-08	11,13,15,17	T+V	Sunny
15-08	13	Test	Sunny
16-08	11,13,15,17	T+V	Sunny

Table C-23: Interpolating July-August, similar weather conditions.

Date	Hours	Data Set	Weather
01-07	11,13,15,17	T+V	Foggy (dark)
02-07	11,13,15,17	T+V	Mostly sunny
03-07	13	Test	Mostly sunny
04-07	11,13,15,17	T+V	Mostly sunny
05-07	13	Test	Cloudy (dark)
07-07	11,13,15,17	T+V	Sunny
10-08	11,13,15,17	T+V	Mostly sunny
11-08	11,13,15,17	T+V	Rainy
12-08	11,13,15,17	T+V	Cloudy
13-08	13	Test	Cloudy
15-08	13	Test	Sunny
16-08	11,13,15,17	T+V	Sunny

Table C-24: Interpolating July-August, multiple weather conditions.

C-2-3 Interpolation over a longer period of time

Images in training (T), validation (V) and test data set April-August 2016 (tables C-25 and C-26).

Date	Hours	Data Set	Weather
29-04	13	Test	Cloudy
30-04	13	Test	Sunny
05-05	11,13,15,17	T+V	Sunny
04-05	13	Test	Sunny
09-05	11,13,15,17	T+V	Sunny
10-05	13	Test	Cloudy (dark)
03-06	13	Test	Foggy
04-06	11,13,15,17	T+V	Sunny
05-06	11,13,15,17	T+V	Sunny
06-06	11,13,15,17	T+V	Sunny
09-06	13	Test	Sunny
10-06	11,13,15,17	T+V	Sunny
02-07	11,13,15,17	T+V	Sunny
03-07	11,13,15,17	T+V	Sunny
05-07	13	Test	Cloudy (dark)
06-07	13	Test	Sunny
13-08	13	Test	Cloudy
16-08	13	Test	Sunny
29-10 *	13	Test	Cloudy
02-12 *	13	Test	Sunny

Table C-25: Interpolating April-August, similar weather conditions.

Date	Hours	Data Set	Weather
29-04	13	Test	Cloudy
30-04	13	Test	Sunny
04-05	13	Test	Sunny
05-05	11,13,15,17	T+V	Sunny
09-05	11,13,15,17	T+V	Sunny
10-05	13	Test	Cloudy (dark)
03-06	13	Test	Foggy
02-06	11,13,15,17	T+V	Rainy
04-06	11,13,15,17	T+V	Sunny
06-06	11,13,15,17	T+V	Sunny
09-06	13	Test	Sunny
10-06	11,13,15,17	T+V	Sunny
01-07	11,13,15,17	T+V	Foggy(dark)
02-07	11,13,15,17	T+V	Sunny
05-07	13	Test	Cloudy (dark)
06-07	13	Test	Sunny
13-08	13	Test	Cloudy
16-08	13	Test	Sunny
29-10 *	13	Test	Cloudy
02-12 *	13	Test	Sunny

Table C-26: Interpolating April-August, multiple weather conditions.

* *Extrapolating test data set images; the recorded scene is slightly different. The camera angle has changed and painted markings on the parking area have been added.*

Appendix D

Experimental Results Train Set Optimisation

Evaluating test images on number of patches per image n_p, oversampling factor f_o and undersampling factor f_u. Experiments are carried out on oversampling the artificial grass class (section D-1) and undersampling this class (section D-2).

D-1 Oversampling

Accuracy and F-score for test image recorded at 09-06-2016 , 13:00 hrs (table D-1). Networks trained with train sets for several patches per images n_p and oversampling factor f_o.

Experiment	**Test image** Similar to train set	**Accuracy**	**F-score** Broadleaf Tree	Grass	Sand	Asphalt	Artificial Grass
$n_p = 1000$ $f_o = 0$	09-06-2016 Hour 13	82.14%	0.98	0.91	0.27	0.47	0
$n_p = 1000$ $f_o = 10$	09-06-2016 Hour 13	88.21%	0.99	0.72	0.86	0.67	0.88
$n_p = 1000$ $f_o = 70$	09-06-2016 Hour 13	96.35%	0.99	0.88	0.94	0.94	0.86
$n_p = 1500$ $f_o = 0$	09-06-2016 Hour 13	96.7%	0.99	0.93	0.82	0.95	0
$n_p = 1500$ $f_o = 10$	09-06-2016 Hour 13	80.26%	0.98	0.68	0.54	0.35	0.16
$n_p = 1500$ $f_o = 70$	09-06-2016 Hour 13	97.25%	1.00	0.91	0.98	0.94	0.97

Table D-1: Accuracy and F-score per experiment for different values of n_p and f_o for a test image under similar weather conditions compared to the train set.

Segmented test images show that number of patches per image n_p and oversampling factor f_o affect the final segmented result (figure D-1).

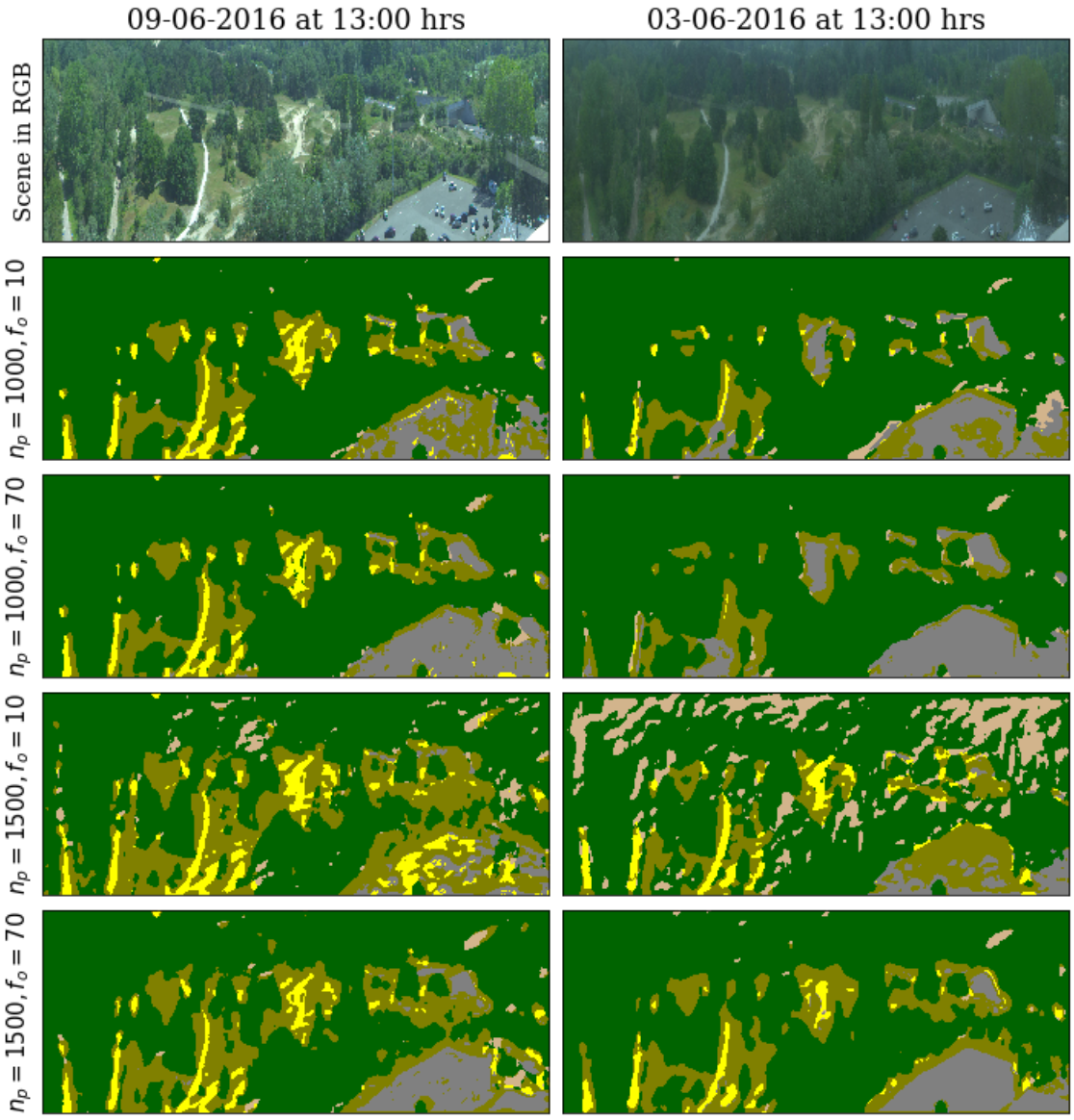

Figure D-1: Segmented test images; oversampled train sets with several values for n_p and f_o.

D-2 Undersampling

Accuracy and F-score for test image recorded at 09-06-2016, 13:00 hrs (table D-2). Networks trained with train sets for several patches per images n_p and undersampling factor f_u.

Experiment	**Test image**	**Accuracy**	**F-score**				
			Broadleaf Tree	Grass	Sand	Asphalt	Artificial Grass
$n_p = 1000$ $f_u = 0$	09-06-2016 Hour 13	82.14%	0.98	0.91	0.27	0.47	0
$n_p = 1000$ $f_u = 0.6$	09-06-2016 Hour 13	94.79%	0.98	0.89	0.73	0.93	0
$n_p = 1000$ $f_u = 0.8$	09-06-2016 Hour 13	78.05%	0.96	0.72	0.27	0.40	0
$n_p = 1500$ $f_u = 0$	09-06-2016 Hour 13	96.7%	0.99	0.93	0.82	0.95	0
$n_p = 1500$ $f_u = 0.6$	09-06-2016 Hour 13	89.55%	0.95	0.7	0.64	0.91	0.04
$n_p = 1500$ $f_u = 0.8$	09-06-2016 Hour 13	92.64%	0.95	0.70	0.95	0.97	0

Table D-2: Accuracy and F-score per experiment for different values of n_p and f_u.

Segmented test images show that number of patches per image n_p and undersampling factor f_u affect the final segmented result (figure D-2).

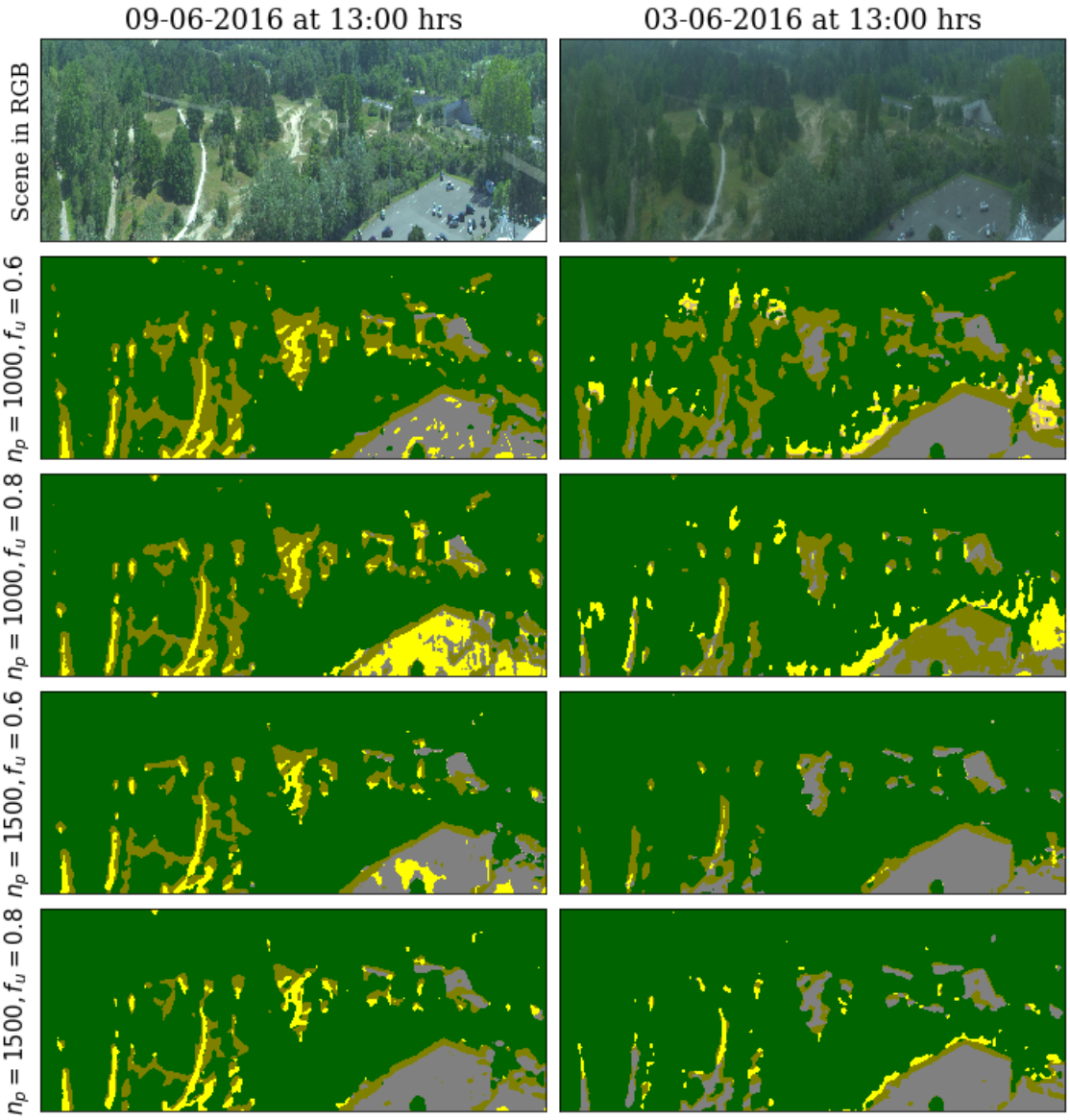

Figure D-2: Segmented test images; undersampled train sets with several values for n_p and f_u.

Appendix E

Experimental Results Temporally Changing Conditions

In order to research the network limitations regarding generalizability for weather and seasonal conditions, experiments have been carried out over the course of a week (section E-1 to E-4), four weeks (section E-5 to E-6) and five months (section E-7).

E-1 Interpolating between one week in April-May

Segmented test data set images, networks trained with data from a week in April-May (figure E-1).

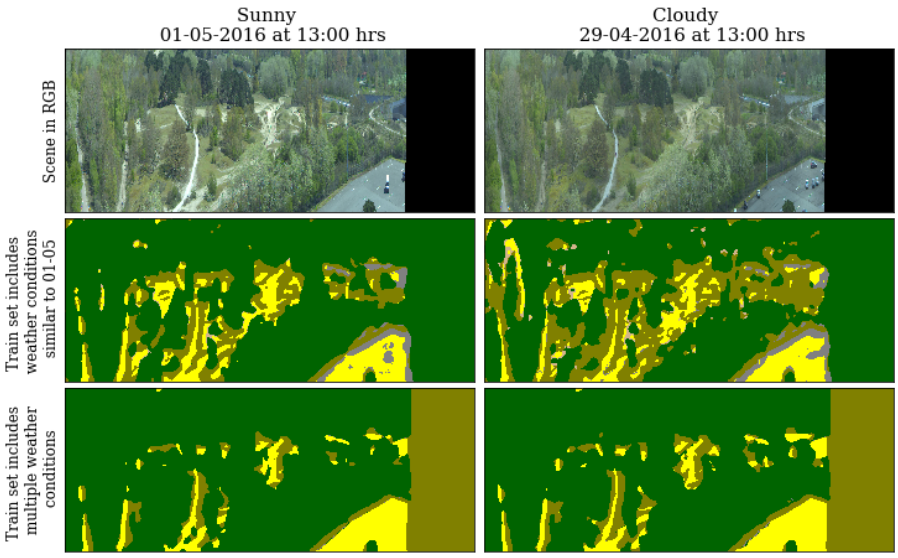

Figure E-1: Segmentations for a sunny and cloudy test image; obtained with (1) a training data set comprising sunny images only and (2) a training data set comprising multiple weather conditions.

E-2 Interpolating between one week in June

Segmented test data set images, networks trained with data from a week in June (figure E-2).

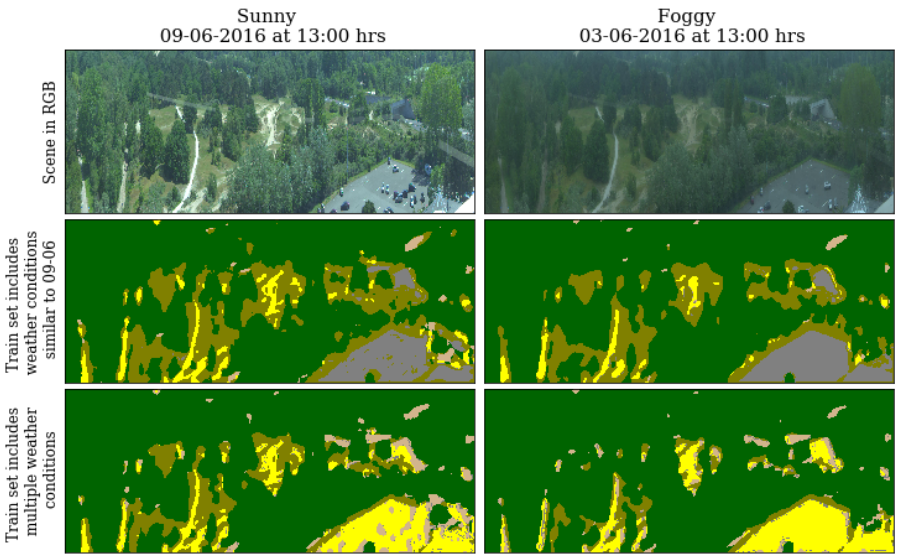

Figure E-2: Segmentations for a sunny and cloudy test set image; obtained with (1) a training data set comprising sunny images only and (2) a training data set comprising multiple weather conditions.

Extrapolation beyond the training data set. Accuracy, F-score per class and mean F-score

for test images in table E-1.

Test Image			Brd.leaf	Grass	Sand	Asphalt	Art. G	
Date	*Hour*	A[%]	F_1	F_2	F_3	F_4	F_5	$\bar{F}$
30-04	13	38	0.00	0.37	0.48	0.95	0.00	0.36
29-04	13	36	0.00	0.43	0.10	0.92	0.00	0.29
04-05	13	40	0.06	0.36	0.84	0.94	0.00	0.44
10-05	13	79	0.84	0.62	0.40	0.94	0.00	0.56
09-06	13	97	1.00	0.91	0.98	0.94	0.97	0.96
03-06	13	96	0.99	0.86	0.73	0.96	0.76	0.86
06-07	13	96	0.99	0.87	0.90	0.92	0.81	0.90
05-07	13	96	0.99	0.83	0.69	0.94	0.73	0.84
16-08	13	97	0.99	0.90	0.96	0.94	0.33	0.82
13-08	13	96	0.98	0.87	0.76	0.96	0.31	0.78

Table E-1: Accuracy A, F-score per class and mean F-score $\bar{F}$ of images from the test data set. Evaluation metrics correspond to segmentations in figure 6-4 (left to right, top to bottom).

E-3 Interpolating between one week in July

Segmented test data set images, network trained with data from a week in July (figure E-3).

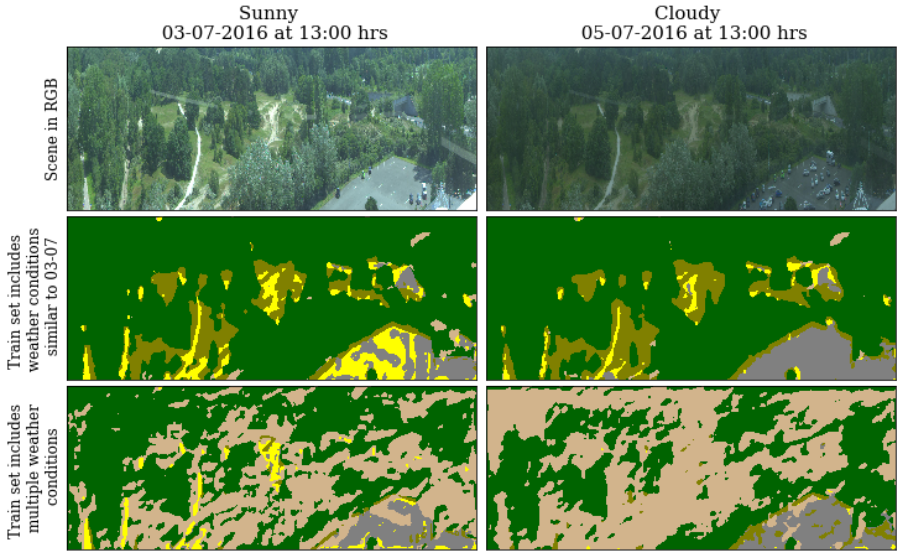

Figure E-3: Segmentations for a sunny and cloudy test image; obtained with (1) a training data set comprising sunny images only and (2) a training data set comprising multiple weather conditions.

Misclassification error increases in case the annotation mask does not fit the training data set due to vegetation growth, especially for the artificial grass class (figure E-4).

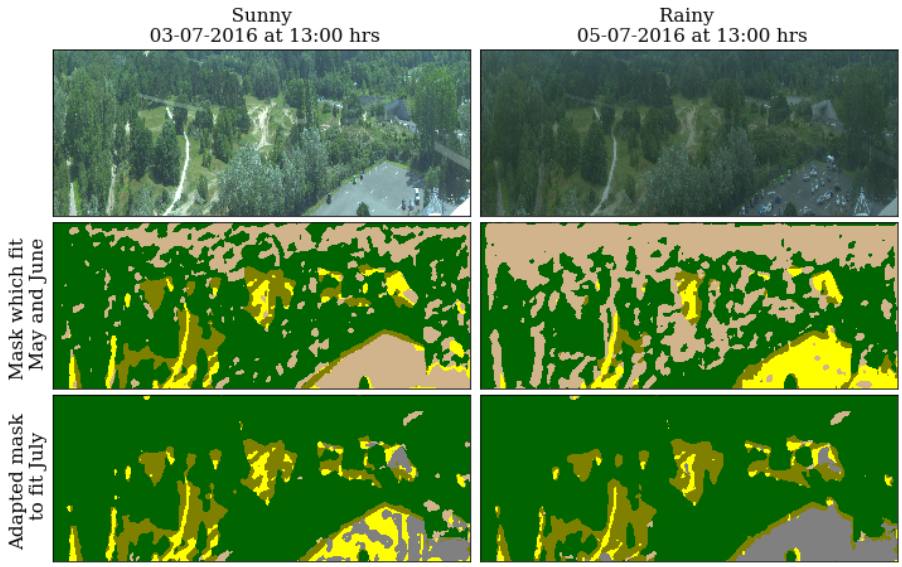

Figure E-4: Segmentations in case annotation mask does not fit artificial grass class (1), segmentations for adapted annotation mask (2).

Accuracy and F-Score for interpolating test data set images over time of the day (figure 6-2).

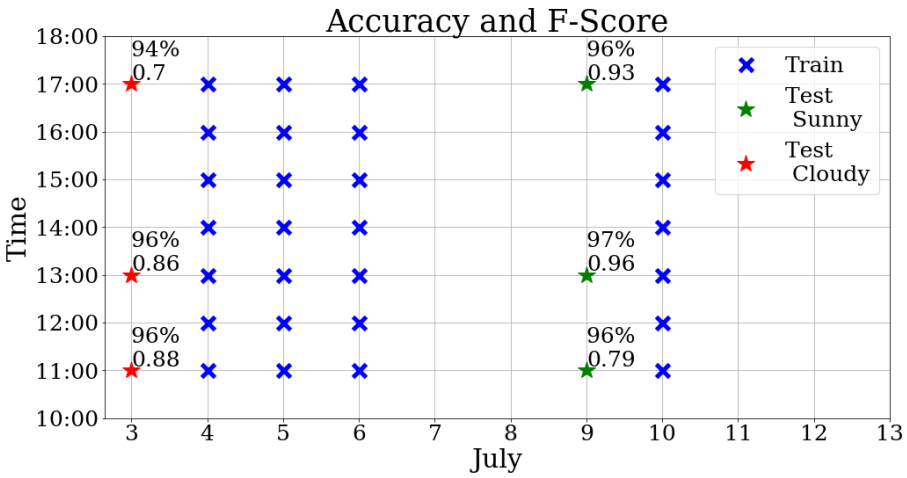

Figure E-5: Accuracy and F-score of test data set images over time (model trained with a single -sunny- weather type in July).

Extrapolating beyond the July train set to evaluate generalizing characteristics over seasons (figure E-6).

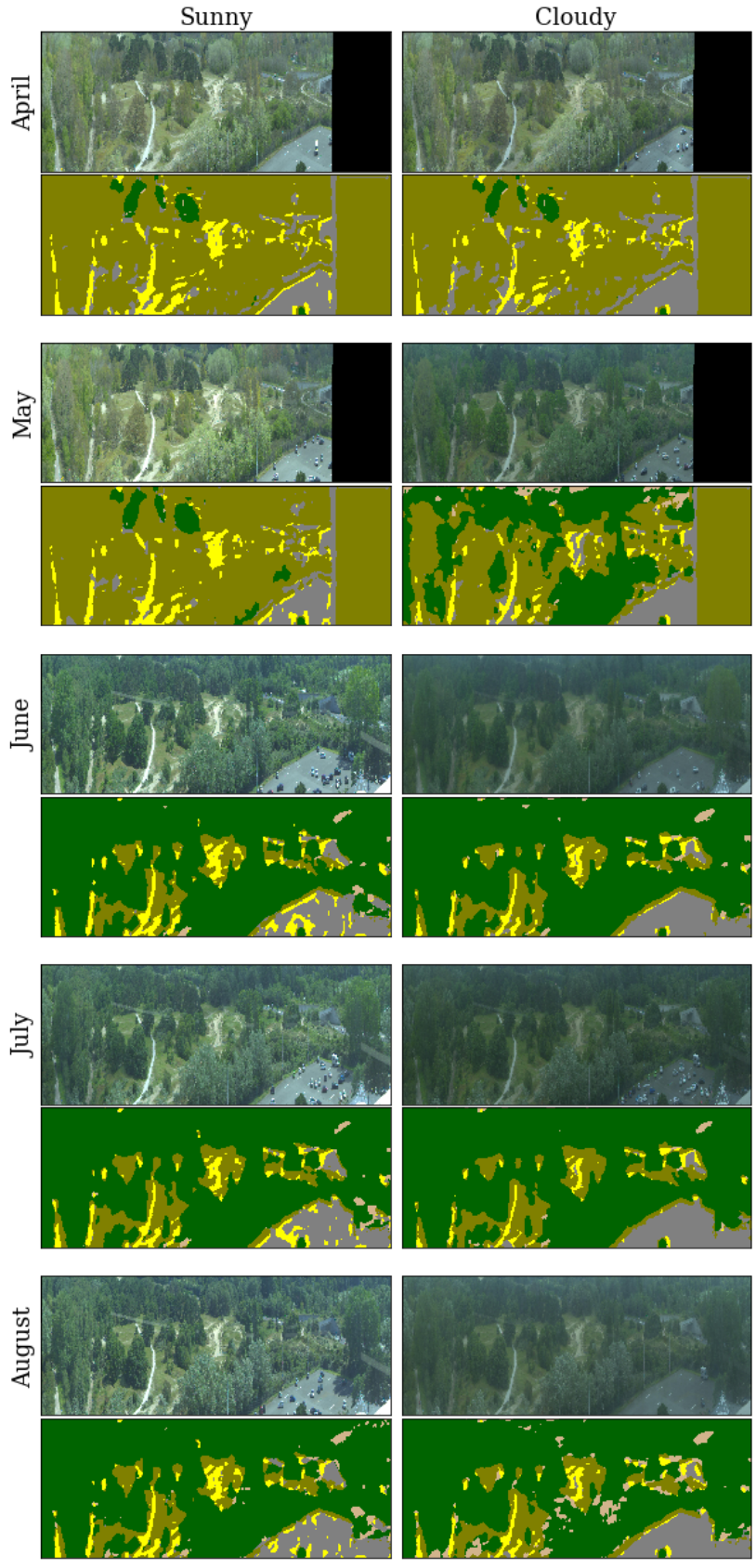

Figure E-6: Extrapolation beyond training data set; generated using a network trained on data from a single week in July (sunny weather conditions).

E-4 Interpolating between one week in August

Segmented test data set images, network trained with data from a week in August (figure E-7).

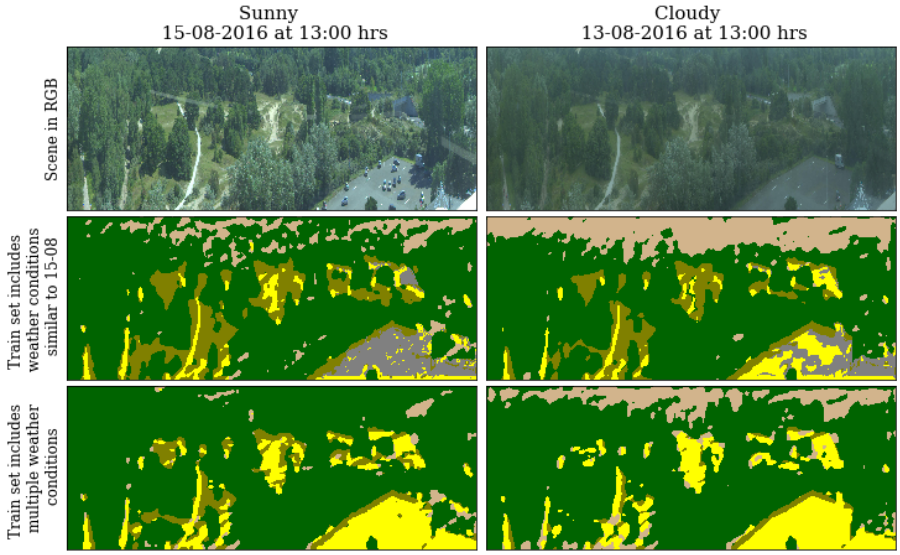

Figure E-7: Segmented test data set images from August, obtained with two differently trained networks.

E-5 Interpolating between days in four weeks in May and June

Segmented test images, by training a network over the course of four weeks in May and June (figure E-8 and E-10). Accuracy and mean F-score of test data set images are summarized in figures E-9 and E-11.

E-6 Interpolating between days in four weeks in July and August

Segmented test data set images, by training a network over the course of four weeks in July-August (figure E-12 and E-14). Accuracy and mean F-score of test data set images are summarized in figures E-13 and E-15.

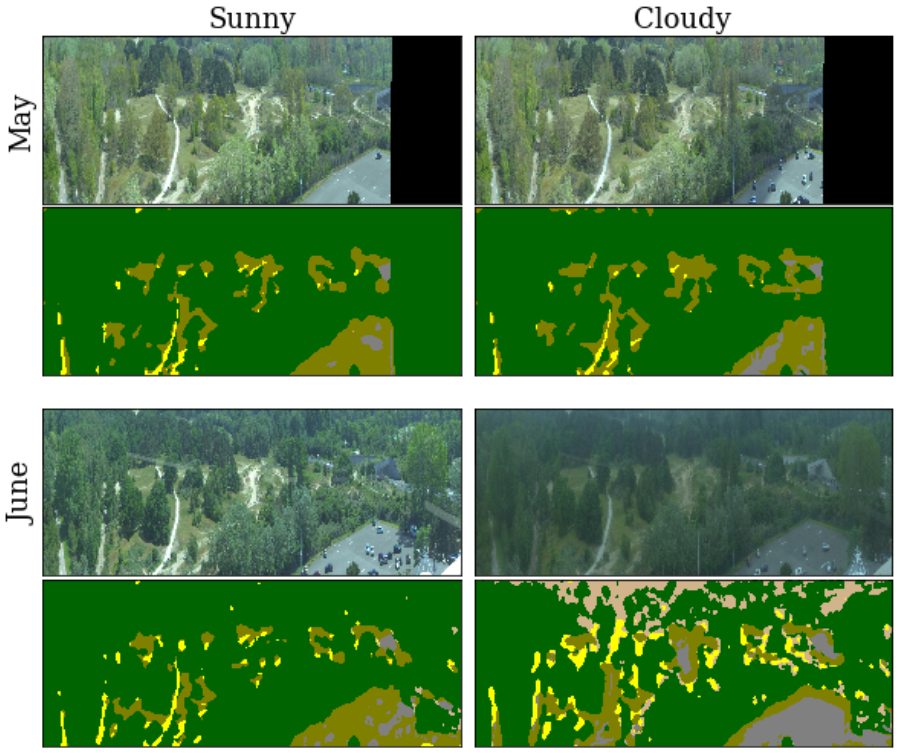

Figure E-8: U-Net trained with training data set under sunny weather conditions in May and June. Test data set images recorded at 06-05-2016, 03-05-2016, 09-06-2016 and 03-06-2016 (top to bottom, left to right).

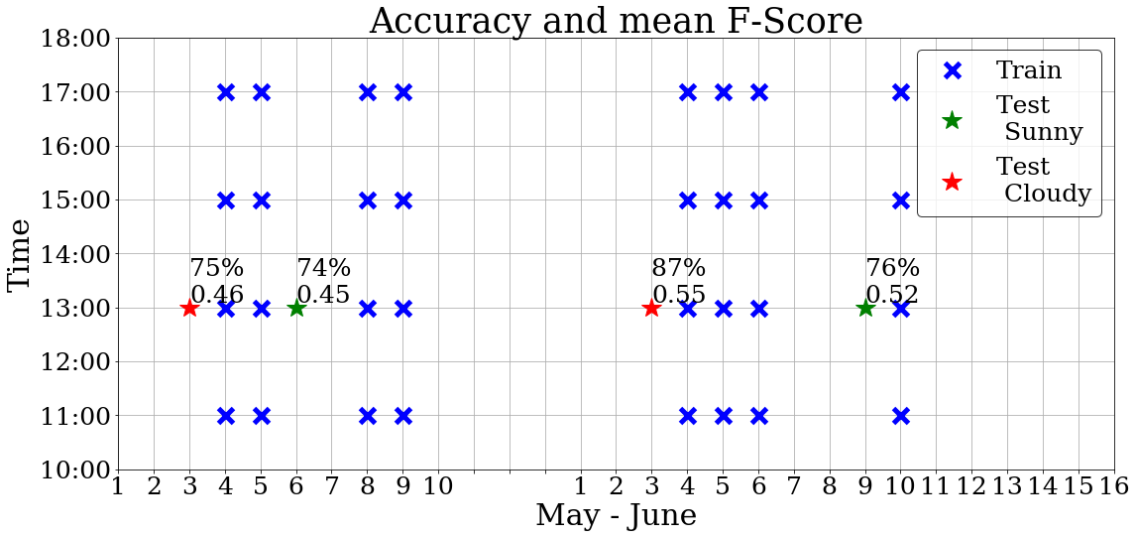

Figure E-9: Accuracy and mean F-score per test data set image for U-Net trained on a test data set under sunny weather conditions in May and June, scores correspond to images in figure E-8.

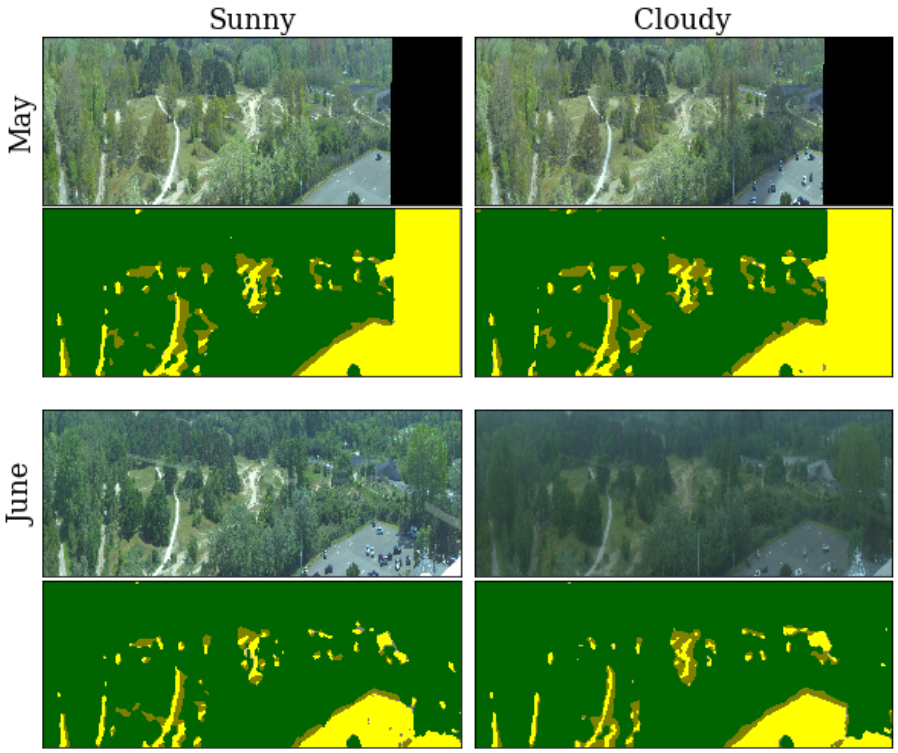

Figure E-10: U-Net trained with training data set under multiple weather conditions in May and June. Test data set images recorded at 06-05-2016, 03-05-2016, 09-06-2016 and 03-06-2016 (top to bottom, left to right).

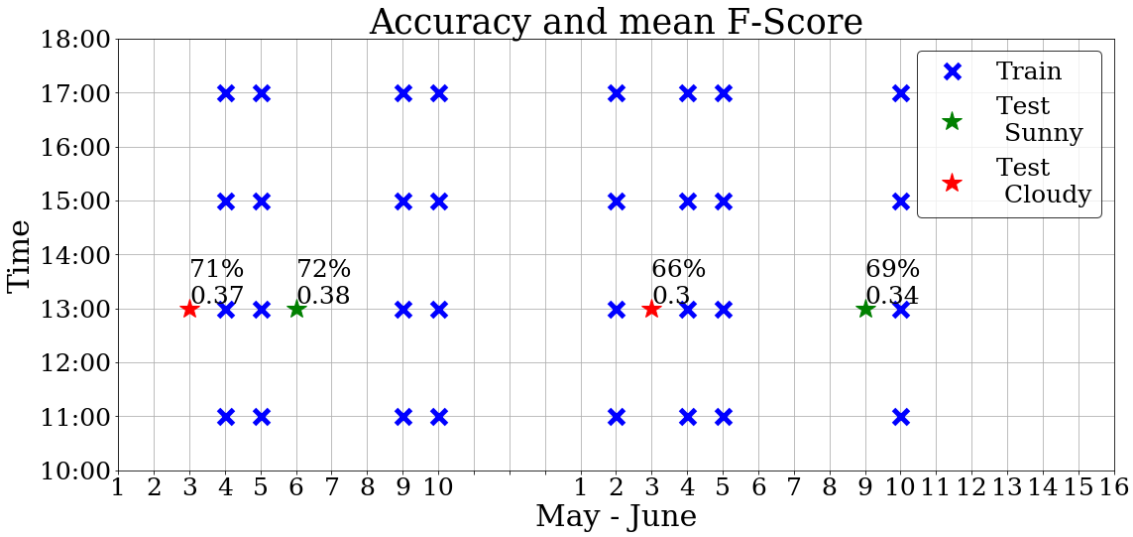

Figure E-11: Accuracy and mean F-score per test data set image for U-Net trained on a training data set under multiple weather conditions in May and June, scores correspond to images in figure E-10.

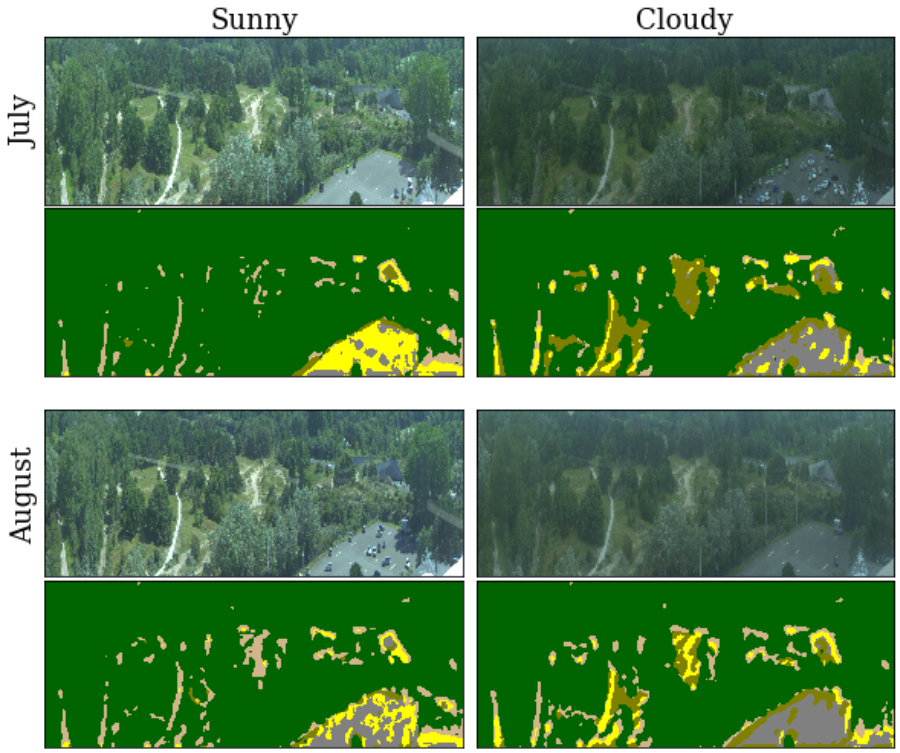

Figure E-12: U-Net trained with training data set under sunny weather conditions in July and August. Test images recorded at 03-07-2016, 05-07-2016, 15-08-2016 and 13-08-2016 (top to bottom, left to right).

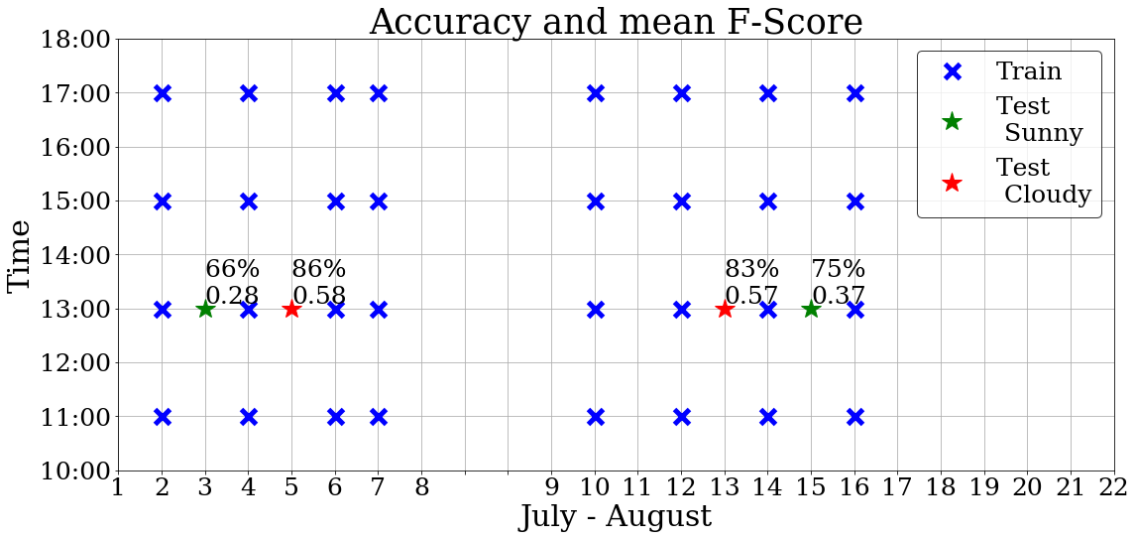

Figure E-13: Accuracy and mean F-score per test data set image for U-Net trained on a training data set under sunny weather conditions in July and August, scores correspond to images in figure E-12.

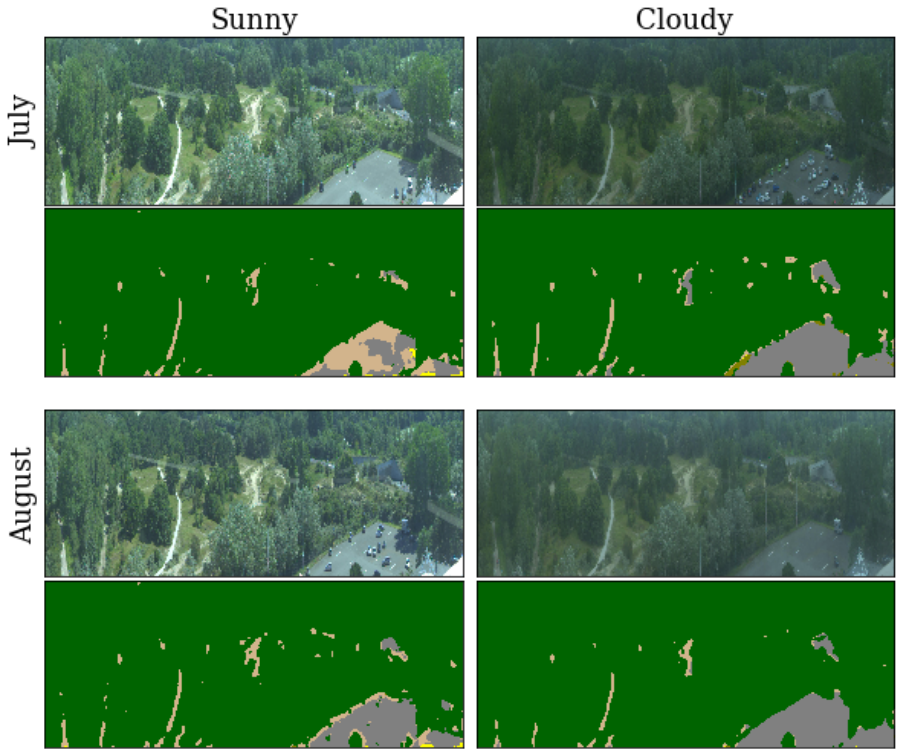

Figure E-14: U-Net trained with training data set under multiple weather conditions in July and August. Test data set images recorded at 03-07-2016, 05-07-2016, 15-08-2016 and 13-08-2016 (top to bottom, left to right).

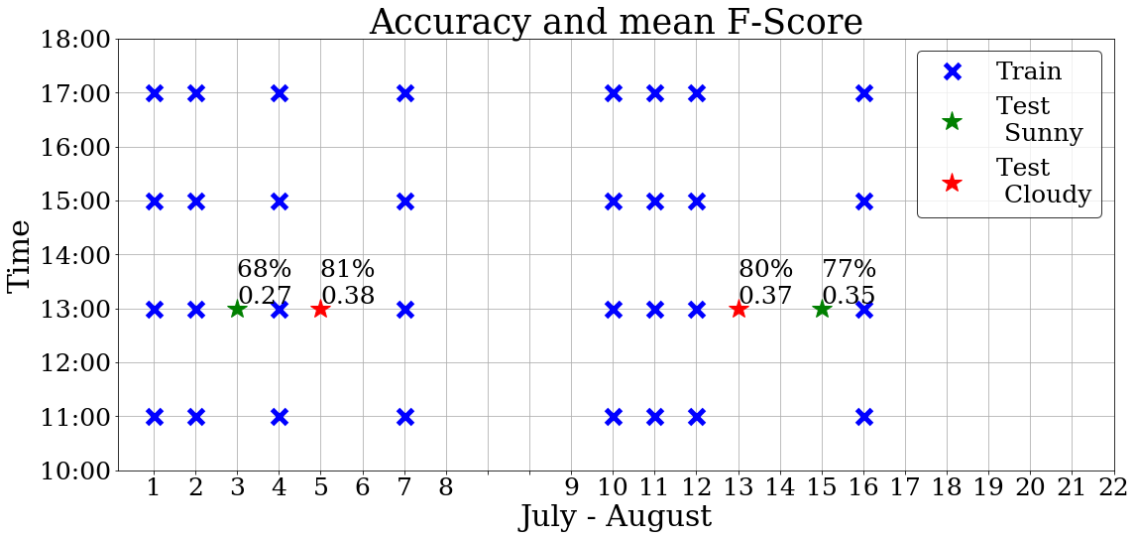

Figure E-15: Accuracy and mean F-score per test data set image for U-Net trained on a training data set under multiple weather conditions in July and August, scores correspond to images in figure E-14.

E-7 Interpolating between days in five months

Segmentations of the test data set images are obtained by training a network over a period of five months (April to August). Accuracy A, F-score per class and mean F-score $\bar{F}$ of the test set in displayed in table E-2.

Test Image			Brd.leaf	Grass	Sand	Asphalt	Art. G	
Date	*Hour*	A[%]	F_1	F_2	F_3	F_4	F_5	$\bar{F}$
30-04	13	90	0.94	0.82	0.76	0.92	0	0.69
29-04	13	90	0.94	0.84	0.56	0.90	0	0.65
04-05	13	92	0.96	0.86	0.71	0.89	0	0.68
10-05	13	93	0.98	0.91	0.22	0.87	0	0.60
09-06	13	89	0.96	0.79	0.59	0.80	0	0.63
03-06	13	89	0.94	0.62	0.40	0.94	0	0.58
06-07	13	87	0.95	0.66	0.57	0.83	0	0.60
05-07	13	87	0.93	0.51	0.39	0.93	0	0.55
16-08	13	90	0.96	0.79	0.64	0.86	0	0.65
13-08	13	91	0.95	0.70	0.43	0.94	0	0.61

Table E-2: Accuracy A, F-score per class and mean F-score $\bar{F}$ of images from the test data set. Evaluation metrics correspond to segmentations in figure 6-7 (left to right, top to bottom).

Segmentations of test images from October and November (figure E-16) are obtained with the same network (trained with images from April to August). It shows extrapolation capacity beyond the training data set.

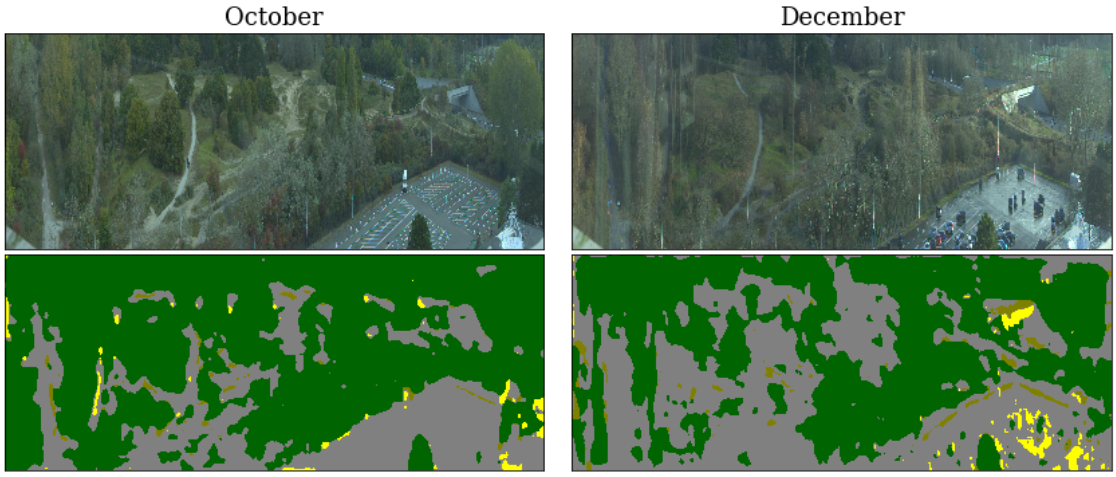

Figure E-16: Evaluating extrapolative properties. October (accuracy 65%) and December (accuracy 54%) segmented test data set images, yielded with a network trained on data from April to August.

Appendix F

Increasing Generalizability

Increasing neural network generalizability is carried out considering two different approaches; normalizing the hyperspectral (HS) input data (section F-1) and altering the number of feature maps in the standard U-Net architecture (section F-2).

F-1 Normalizing Hyperspectral Data

The training data set is subject to large variation, in a pre-processing step this is reduced by normalizing the raw HS data. First, experiments using per pixel normalized images are carried out (section F-1-1). The normalization step transforms all 25 bands per pixel so that they sum to unity. Secondly, Z-score normalization per band is used (section F-1-2).

F-1-1 Pixel sum to unity

Normalization has been carried out as an additional computational step; all pixels sum to unity. Experiments for a train set under both a single and multiple weather conditions have been carried out. On the most simple train set (single week in June, sunny weather conditions) the normalized train data (figure F-1) did not prove to be as accurate as training with raw train data. Furthermore, training with more complex train sets (single week in June, multiple weather conditions) showed no further use of this normalization method.

F-1-2 Z-score normalization for every band

As a pre-processing step, Z-score normalization has been carried out. The normalized image $\hat{x}$ is created using the mean μ_i and standard deviation σ_i of every band i from the raw image x (equation F-1).

$$\hat{x}_i = \frac{x_i - \mu_i}{\sigma_i} \quad \text{for } i = 1, 2...25 \tag{F-1}$$

Also this normalization method has not improved segmentation accuracy, results are similar to the method used in section F-1-1.

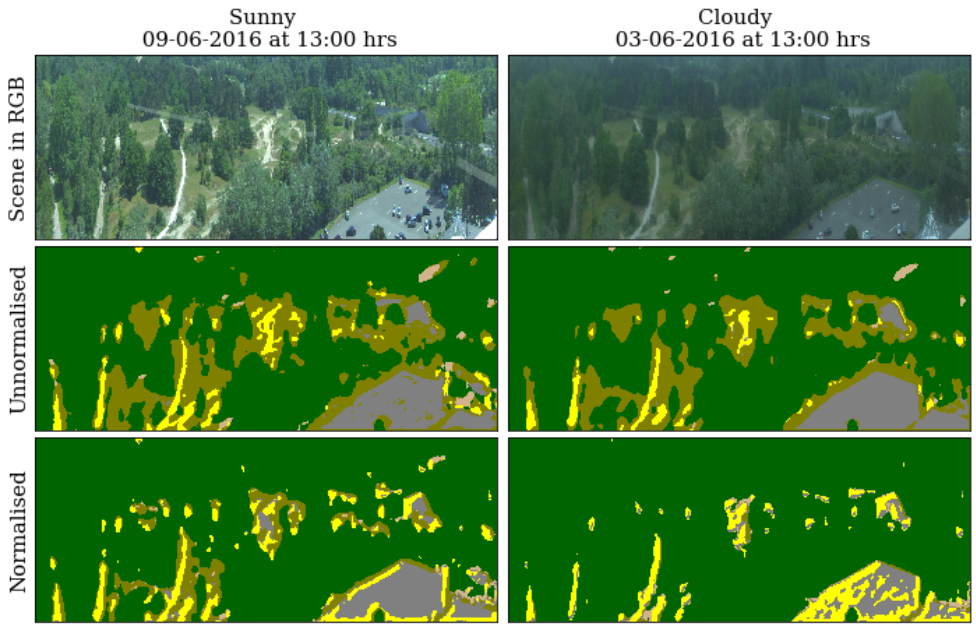

Figure F-1: Segmented test images for networks trained using raw (unnormalized) and normalized HS data.

F-2 Customized Architecture

In order to improve generalizing capabilities of the network, the trainable parameters in the convolutional layers are redistributed (table F-1). Parameters have been added to the shallow layers of the network, whilst parameters have been removed from the deepest layers. The total number of weights have roughly remained the same. The shallow layers generate features focused on edges and lines. In deeper network layers complex features are trained, aimed at high level concepts.

The network does not necessarily need a larger capacity or higher number of trainable parameters to enlarge generalization properties. The total number of parameters is kept approximately the same (i.e. in the same order of magnitude) while the number of feature maps per convolutional layers are changed. This customization targets the features to be formed, focusing more on the available spectral signatures.

F-2-1 Customized U-Net

The network which corresponds to the redistributed weights has a customized number of feature maps in each convolutional layer (figure F-2). The original U-Net architecture includes an exponentially increasing number of feature maps, starting with 32 feature maps in the first stack of layers. The customized U-net includes exponentially growing steps between layers, starting with a step of adding 20 feature maps. The following kernels grow with 40, 80 and 160 feature maps respectively.

Stack	Layer	Trainable parameters	
		U-Net	**Customized U-Net**
Decoder 1	Batch normalization	50	50
	Convolution (3×3)	7232	22.600
	Convolution (3×3)	9248	90.100
Decoder 2	Convolution (3×3)	18.496	108.120
	Convolution (3×3)	36.928	129.720
Decoder 3	Convolution (3×3)	73.856	172.960
	Convolution (3×3)	147.584	230.560
Decoder 4	Convolution (3×3)	295.168	345.840
	Convolution (3×3)	590.080	518.640
Decoder 5	Convolution (3×3)	1.180.160	864.400
	Convolution (3×3)	2.359.808	1.440.400
Encoder 6	Convolution (3×3)	1.769.728	1.382.640
	Convolution (3×3)	590.080	518.640
Encoder 7	Convolution (3×3)	442.496	576.160
	Convolution (3×3)	147.584	230.560
Encoder 8	Convolution (3×3)	110.656	302.520
	Convolution (3×3)	36.928	129.720
Encoder 9	Convolution (3×3)	27.680	198.100
	Convolution (3×3)	9248	90.100
	Convolution (1×1)	165	505
Total		7.853.175	7.352.335

Table F-1: Redistribution of the trainable weights in U-Net.

F-2-2 Hyperparameters

The customized network requires the hyperparameters to be redefined (table F-2).

Hyperparameter	Setting
Optimizer	Adam
Learning rate	$1 \cdot 10^{-6}$
Dropout	20%

Table F-2: Redefined hyperparameters, the remaining parameters have remained the same.

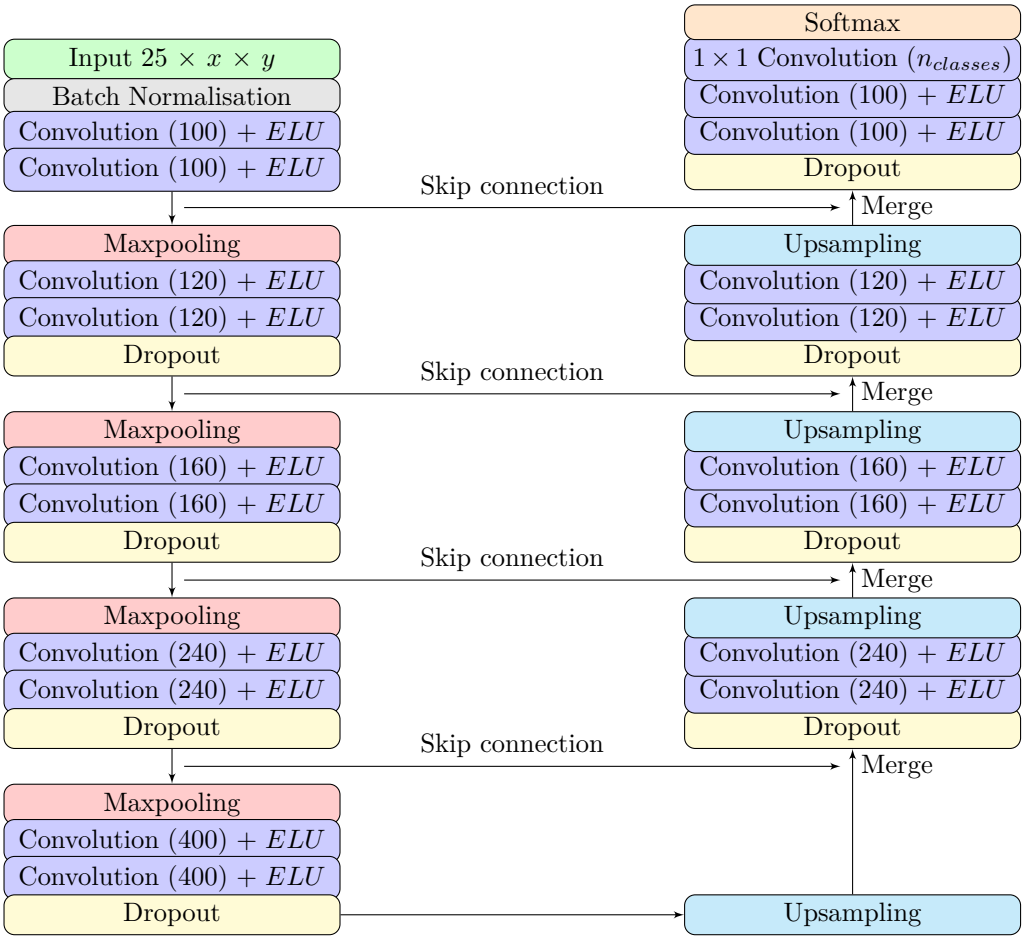

Figure F-2: Customized U-Net architecture.

Bibliography

[1] Y. Tarabalka, *Classification of Hyperspectral Data Using Spectral-Spatial Approaches*. PhD thesis, University of Iceland, Universite de Grenoble, 2010.

[2] F.-F. Li, "Stanford university computer science class cs231n: Convolutional neural networks for visual recognition." http://cs231n.github.io/convolutional-networks/, 2017. Accessed: 09-02-2017.

[3] M. Nielsen, "Neural networks and deep learning online book." http://neuralnetworksanddeeplearning.com, 2017. Accessed: 09-02-2017.

[4] P. Velickovic, "Cambride coding academy, deep learning for complete beginners: using convolutional nets to recognise images." http://online.cambridgecoding.com/notebooks/cca_admin/convolutional-neural-networks-with-keras, 2017. Accessed: 09-02-2017.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research 15*, 2014.

[6] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[7] M. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision*, 2013.

[8] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for segmantic segmentation," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[9] L.-C. Chen, G. Papandeou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation." 2017.

[10] U. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[11] CORDIS, "European commission; extended image sensing technologies project details." http://cordis.europa.eu/project/rcn/198017_en.html, 2015.

[12] M. K. Griffin and H. hua K. Burke, "Compensation of hyperspectral data for atmospheric effects," *Lincoln Laboratory Journal*, 2003.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, 2012.

[14] G. Cybenko, "Approximation by superposition of a sigmoidal function," *Math. Control Signals Systems*, 1989.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[19] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[20] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, 2011.

[21] M. D. Zeiler, "Adadelta: An adaptive learning rate method." 2012.

[22] G. Hinton, N. Srivastava, and K. Swersky, "Overview of mini-batch gradient descent." https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference for Learning Representations, San Diego*, 2015.

[24] D. Marmanis, J. Wegner, S. Galliani, K. Schindler, M. Datcu, and U. Stilla, "Semantic segmentation of aerial image with ensemble of cnns," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2016.

[25] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Convolutional neural networks for large-scale remote-sensing image classification," *IEEE Transactions on Geoscience and Remote Sensing*, 2016.

[26] M. Volpi and D. Tuia, "Dense semantic labeling of sub-decimeter resolution image with convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, 2016.

[27] O. Matan, C. Burges, Y. LeCun, and J. Denker, "Multi-digit recognition using a space displacement neural network," *Neural Information Processing Systems, volume 4*, 1992.

[28] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2017.

[29] G. Lin, A. Milan, C. Shen, and I. Reid, "Refinenet: Multi-path refinement network for high-resolution semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[30] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," in *International Conference on Learning Representations Workshop*, 2016.

[31] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[32] Kaggle, "Dstl satellite imagery competition." http://blog.kaggle.com/2017/04/26/dstl-satellite-imagery-competition-1st-place-winners-interview-kyle-lee/, 2017. Details of strategy.

[33] IMEC, "Specialty cmos image sensors." https://www.imec-int.com/nl/expertise/image-sensors-and-vision-systems/specialty-cmos-image-sensors, 2017.

[34] F. Chollet, "keras." https://github.com/fchollet/keras, 2015. GitHub repository.

[35] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Mï£¡ller, *Efficient BackProp*. Springer, 1998.

[36] S. Ioffe and C. Szegedy, "Batch normalisation: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[37] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *International Conference on Learning Representations (ICLR)*, 2016.

[38] A. Karpathy, "Convolutional neural networks for visual recognition." Course notes of Standford CS class CS231n.

[39] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

Glossary

List of Acronyms

API	Application Programming Interface
CMOS	Complementary Metal Oxide Semiconductor
CNN	Convolutional Neural Network
CNTK	Microsoft Cognitive Toolkit
CRF	Conditional Random Field
ELU	Exponential Linear Unit
EXIST	Extended Image Sensing Technologies
FCN	Fully Convolutional Network
GD	Gradient Descent
GPU	Graphics Processing Unit
HS	hyperspectral
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue
SGD	Stochastic Gradient Descent