Thesis for the degree of Master of Science in Engineering Physics

Classification of high resolution satellite images

Anders Karlsson

Laboratoire de Systèmes d'Information Géographique
Ecole Polytéchnique Fédérale de Lausanne
August 2003

Classification of high resolution satellite images

Anders Karlsson
Student of the Department of Engineering Physics
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Abstract

In this thesis the Support Vector Machine (SVM) is applied on classification of high resolution satellite images. Several different measures for classification, including texture measures, 1st order statistics, and simple contextual information were evaluated. Additionally, the image was segmented, using an enhanced watershed method, in order to improve the classification accuracy.

The Support Vector Machine was found to be flexible and powerful but still not perfectly suited for high resolution images. Classifying such images requires contextual information to be taken into consideration, and the SVM could not efficiently learn correct context from training examples. Without including contextual information, the SVM can still be usable for high resolution images in cases where only few land cover classes are used. Segmenting the image prior to classification did improve the results somewhat, at least visually, but a stronger reason for segmentation is that it provide a means for taking advantage of contextual information. The watershed method for segmentation proved to be efficient and the results thereof could be further improved by merging regions.

Acknowledgments

This diploma thesis is for the degree of Master of Science in Engineering Physics at Chalmers University of Technology in Gteborg, Sweden. It was carried out between March and August 2003 at the Laboratory of Geografic Information Systems (LaSIG) at the Swiss Federal Institute of Technology in Lausanne, Switzerland.

I would like to thank Professor Regis Caloz for inviting me to do my diploma thesis at LaSIG, and for his and my supervisor Abram Pointet's assistance throughout the work. Special thanks to the hospitable staff at the laboratory for being patient with my French and for the nice atmosphere they provided.

Göteborg, November 2003

Anders Karlsson

Contents

1

Introduction

1.1 Background

The launches of the IKONOS and Quickbird satellites in 1999 and 2001, respectively, have improved the resolution power of the previously most powerful commercially available satellite images by factors greater than 10. The Quickbird images provide 2.44 meter detail in multispectral mode (blue, green, red and near-infrared bands) and 0.61 meter in panchromatic mode. This detail is comparable to aerial photography, but satellite imagery has the advantage of being continuously updated without having to cover great areas using an aeroplane with expensive equipment. The availability of such images has given new possibilities to use satellite images in high detail GISs (Geografic Information Systems), including city-planning, real-estate inventory, natural risk management and any other kind of cartography.

Conventional analysis methods have often shown their limits when applied to images of such high resolution. Traditionally, each smallest component of the digital image, the pixel, has been classified individually according to its inherent spectral information only, possibly with the addition of information from neighboring pixels. This is often adequate when using lower resolution images, because land cover usually appears rather homogeneously in coarser scales. With finer resolution the images become less smoothed and more heterogeneous, and each object or region can contain components of many different spectral signatures. In such cases, the per-pixel analysis often render noisy results.

1.2 Objective

The aim of this thesis is to evaluate the traditional per-pixel classification of satellite images and the more recent region based classification. A central question is: does segmenting an image prior to classification aid in classifying its components, and if so, for what occasions is this approach more suitable than the traditional per-pixel classification. Additionally, we will use the recent method Support Vector Machine

(SVM) as the main classification algorithm and qualitatively evaluate its performance and user-friendliness.

Furthermore, a good solution on how to segment an image is sought. This is a large area of research and we do not attempt to develop a cutting edge algorithm for this purpose, but instead we experiment with existing algorithms in order to evaluate their suitability to remote sensing applications.

1.3 Method

Initially, the theory of the Support Vector Machine was studied which led to an implementation in Matlab®. The method was tested to gain understanding of the underlying algorithms and influence of its parameters. Thereafter, an image segmenting method was developed, using different ways to perform the initial partitioning of an image, as well as numerous ways of combining the initial segments into homogeneous regions. Some attempts to train an SVM to perform the segment merging were made. With these two algorithms at hand, both unsegmented and segmented images were classified and the results were analyzed.

1.4 Structure

Chapter 2 explains basic concepts of classification and describes possible attributes that can be used to recognize different types of land cover. A quite detailed introduction to the Support Vector Machine is given. In Chapter 3 the concept of split-and-merge segmentation is explained. Some common methods of initial image segmentation are described and thereafter we propose a machine learning approach of merging the segments from the splitting phase.

The classification results of both pixel-based and segment-based classification are presented in Chapter 4. Also the outcomes of the segmentation method is shown. Chapter 5 is dedicated to a discussion of the results and propositions of further developments and uses of segment-based classification. Chapter 6 concludes the thesis with a summary of the major findings.

The appendix contains larger sized versions of some of the images in the thesis, some data of the test images used in the experiments, and the main Matlab® code used for the experiments.

2

Classification

While beautiful to look at, the very reason for obtaining satellite images is of course to extract information that is relevant for an application. For many geographic information systems, the information about land use is of great importance, and in cartography the usefulness is evident. In this chapter the basic classification procedure for segmented, as well as unsegmented images is described. This is followed by a relatively elaborate description of the Support Vector Machine, based on material from [1] and [2].

2.1 Basic concepts of classification

In a statistical sense, a ***pattern*** is a combination of measured features that describe some phenomenon. The features could be just about anything, but for remote sensing purposes typical features would be first- and second order statistics of pixel intensities from different frequency bands or high-level features like shape or frequency measures of regions in the image. The pattern is usually represented by a vector $\mathbf{x} \in \mathbb{R}^N$, where $\mathbb{R}^N$ is called the ***feature space***.

A ***classifier*** assigns subspaces of the feature space to different classes so that a pattern can be classified according to the subspace it is located in. In remote sensing possible classes include water bodies, forests and urban land cover. The classifier function can be obtained using one of many pattern classification methods. Among the most common are the maximum likelihood discriminator, the k-nearest neighbor and neural networks which are all summarized on p.178-188 of [3]. In recent years the Support Vector Machine (SVM), which is similar to neural networks, has been found to perform well in different kinds of applications.

All of the mentioned methods above rely on ***training examples***, a set of patterns $\mathbf{x}_i \in \mathbb{R}^N, i = 1, 2, ..., m$ with a priori known labels $y_i \in \{1, 2, ..., k\}$ (assuming m training examples and k unique labels), to find the partitioning. Obviously, the training examples must be of sufficient quantity in order to reliably define proper subspaces, and the training set must be increased for higher dimensional feature spaces. A classifier that gives relatively good performance using few training examples or high dimensional

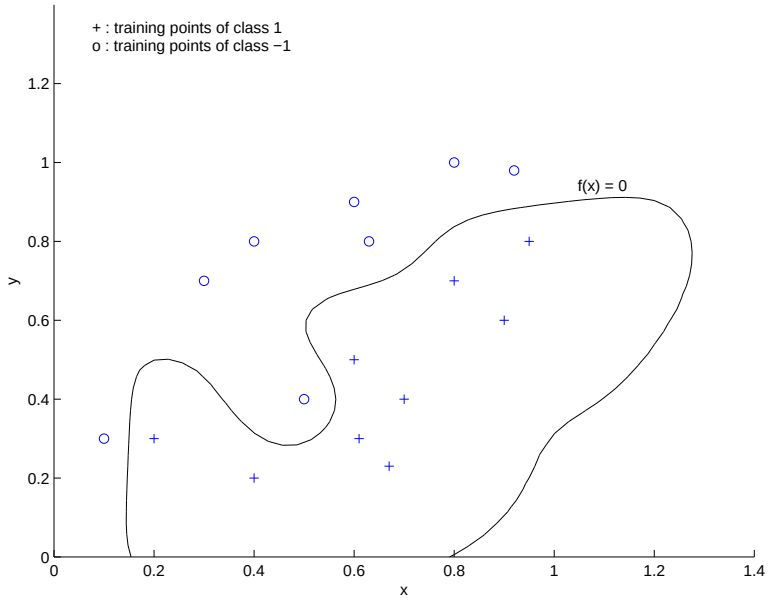

Figure 2.1: A 2-dimensional feature space partitioned into two subspaces by a classifier function according to a set of training points.

patterns is said to generalize well.

Ideally, the training patterns would form well separated clouds in the feature space. Unfortunately, even for correct training examples, there is usually an overlap between the training patterns of different classes, especially if there are many classes and few relevant features. There is a group of classification methods which output the probabilities that a pattern belongs to each of the classes, rather than selecting just one class. This partially addresses the issue of subspace overlapping, but while quite advantageous in many cases, such methods tend to fail for complicated training set distributions.

In the case of ***unsupervised classification***, the classification algorithm automatically finds the most distinct clusters of training examples in the feature space. The training examples will then be labelled according to the cluster it belongs to. In this thesis the training examples are manually labelled, which unsurprisingly is referred to as ***supervised classification***.

2.1.1 Land cover classification

The object of satellite image classification is to determine the land cover class that each unit of the image belongs to. For a non-segmented image, each individual pixel represents a unit, as opposed to a segmented image where the units are ***regions***, a group of neighboring pixels contained within a border. The units are said to be mixed if they are composed of several classes. In such cases it is possible to assign several labels to the same unit, possibly associated with some probability distribution. Herein, we will limit ourselves to one label per unit.

When the classification is supervised, the operator must identify the land cover classes that cover the actual area. A set of units, pixels or regions, are then selected as training examples, which are manually labelled by either visiting the real terrain of the image or judging directly from the image. The next step is to calculate the features of all the units in the image, and combine these into one pattern for each unit. Thereafter, a learning machine is used to train a classifier according to the training examples. The resulting classifier function can then be used to classify all the patterns corresponding to each unit of the image.

2.1.2 Curse of dimensionality

In principle, an infinite number of features could be used for classification, provided it was possible to find a classifier function that could correctly discriminate between all the classes in the entire feature space. The problem is that a training set that covers all possible pattern combinations is needed in order to give the perfect result, and the number of combinations grow exponentially with the dimension. This dilemma is usually referred to as the curse of dimensionality - although a high dimensional feature space has a better chance of separating the classes, the generalization performance will suffer due to need of extensive training input.

Instead of using all available features, one tries to select those who are reasonably uncorrelated and relevant in the way that the information they provide actually improves the possibility to discriminate patterns into subspaces. There are several schemes for finding out which features to pick, each method having both advantages and disadvantages. Usually a systematic trial and error procedure is applied, for example back- or forward analysis as described in [3] on p.205. Another common dimension-reducing technique is the Principal Component Analysis (PCA) which can lower the dimensionality of images that contain several highly correlated bands.

2.2 Features

The components of a pattern, the features, can be any attributes that describe a unit of the image. Among the features used in this project are pixel intensities, texture measures, and shape features. Some features, including 1st order statistics like mean and variance, require information from several pixels to be calculated. For small units, like pixels, we can include the neighboring pixels, creating a windowed feature. Such features work well as long as the texture is constant within the window, but they can be deceptive when calculated over natural edges of the image. In addition there are more or less sophisticated ways to improve the classification results using contextual information. Such information can be incorporated in the classification if it is transformed into some kind of features.

2.2.1 Intensity features

The simplest feature is the ***mean*** intensity of the pixels belonging to one band of a unit. Sometimes the ***median*** intensity is more representative of the unit, since it is

less sensitive to noise or other spurious irregularities. The ***variance*** of the unit's pixel intensities can often be used to discriminate textures. An alternative to variance is to use the ***entropy*** of the unit's pixels:

$$E = -\sum_{i=1}^{n} p_i log(p_i) \tag{2.1}$$

This formula assumes n possible levels of intensity and p_i is the probability that a pixel in the unit has the i:th intensity, estimated as

$$p_i = \frac{n_i}{n} \tag{2.2}$$

where n_i is the number of pixels with intensity i and n is the total number of pixels in the unit. Unfortunately, this measure is not suitable for small- or medium sized units because of the poor probability estimates.

2.2.2 Haralick features

In 1973 Haralick [4] introduced a number of 2nd order measures that can be used to characterize texture. In all 28 different measures were proposed in his work, all based on values from a gray level co-occurrence matrix. To calculate them, a window size and the number of gray levels must be selected. Refer to [4] for the details of the calculations. Of these measures, we use the four most common:

- ***Angular Singular Moment*** is a measure of the homogeneity of a unit.
- ***Contrast*** is a measure of local variation in the unit.
- ***Entropy*** is a measure of disorder in the unit but differs from the 1st order entropy in Section 2.2.1 in that it uses local differences of intensity instead of the intensities themselves to calculate entropy.
- ***Inverse difference moment*** is a measure of lack of variability.

Each of these measures are calculated in 4 directions - 0°, 45°, 90° and 135°. Consequently, directional textures can be distinguished, which can be used to decide wether neighboring regions belong to the same object. However, for normal, non-contextual classification, directional measures are not important since the direction of texture in regions tend to appear randomly, and usually the images are assumed isotropic. In such cases the values from the 4 direction can be added in order to decrease the feature space dimension.

2.2.3 Shape features

For larger units, shape features can provide valuable information. Only the simplest descriptors are used in this thesis, namely the region's ***area*** and its ***border length***. The border can be either the border between two regions or the whole boundary of one region. Both the area and the border length are calculated by simply counting the pixels

in the region and on the border, respectively. While the calculation could be done with more care, the outcome is correct enough to serve the purposes of our experiments.

2.2.4 Contextual features and averaging

When the features are extended to include the features of regions in a proximity to the object to be classified, we are talking about contextual classification. To discriminate a road from a building that has the same shape and color as a road, we would need some higher level knowledge like "if there is a shadow on the left side of the object it is more likely to be a building than a road". It is not necessary and perhaps not advantageous to use a pattern recognition approach. For example, a decision rule tree could be constructed but that is beyond the scope of this thesis.

Contextual information can be included in a statistical learning machine in a simple manner. The patterns are extended by one feature for each land cover class. The features contain the percentages of each units surrounding that is occupied by each class. The percentages are calculated in a window around the units and each pixel can be weighted according to its distance to the unit. This procedure requires an initial classification without the contextual features and then the procedure can be reiterated several times until the result stabilizes. Compared to simple averaging, the advantage of using this method is that the classifier is still minimizing the training error, as opposed to normal averaging where the training examples are abandoned after the first classification. This prevents pixels from being labelled in total disagreement with its spectral signature and the method is thus a bit more robust.

2.3 Support Vector Machines

A support vector machine, SVM, constructs a ***binary*** (two-class) classifier function

$$f_\alpha(\mathbf{x}) = sign(f(\mathbf{x})) : \mathbb{R}^N \rightarrow \{\pm 1\} \tag{2.3}$$

that divides the feature space into two subspaces, one for each class. Using training patterns, $\mathbf{x}_i \in \mathbb{R}^N$ with known labels $y_i \in \{\pm 1\}$, an SVM finds a function that separates the training examples as well as possible under certain constraints. In real problems there are normally more than two classes, and as will be shown, a set of binary classifiers, like those produced by an SVM, can be combined to solve a multi-class classification task.

What sets SVM apart from most other classifiers is that it is a kernel-based approach, which permits high-dimensional feature spaces and complex non-linear classifier functions while avoiding both severe running-time penalties and unwanted effects of high dimensional feature spaces. The resulting classifier is often similar to the ones produced by well-tuned neural networks, but in general an SVM will not need as much user input and tuning as its neural network counterparts. However, a disadvantage is that the training time can be relatively high for large training sets.

2.3.1 Hyperplanes and Margins

A linear binary classifier is based on a ***hyperplane***

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.4}$$

In the case of linearly ***separable*** training data with labels $y_i \in \{\pm 1\}$, a properly chosen hyperplane will divide the feature space into two parts, correctly classifying all training examples $\mathbf{x}_i$ as

$$y_i = sign(f(\mathbf{x}_i)) \tag{2.5}$$

The ***margin*** of a correctly classified point equals its euclidian distance to the hyperplane. In the case of an incorrectly classified point, the margin is the negative distance. A hyperplane is said to be in its ***canonical representation*** if $\mathbf{w}$ and b are scaled so that the condition

$$\min_{i=1,\ldots,m} y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1 \tag{2.6}$$

is fulfilled for both classes. In this case, the smallest margin of points in each class equals $1/\|\mathbf{w}\|$.

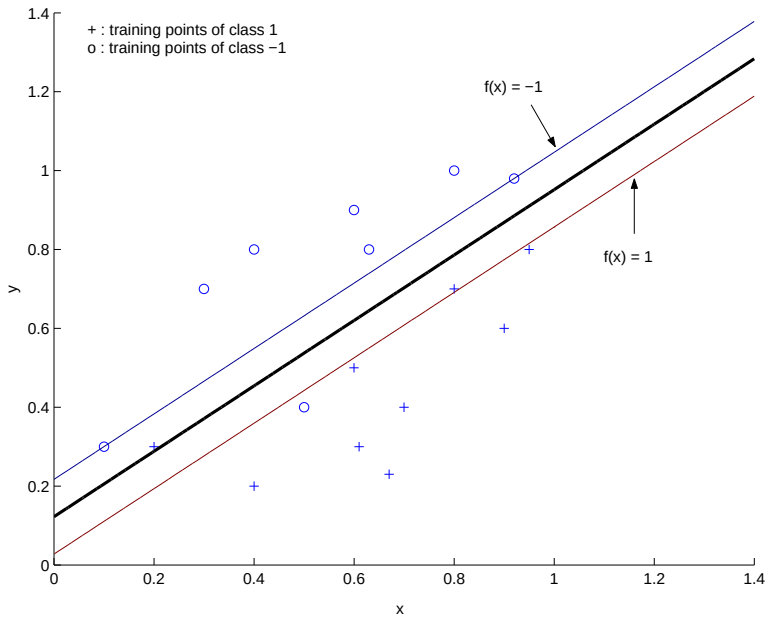

Figure 2.2: Training points, canonical hyperplane and margins in a 2-dimensional feature space. Note that the training set is not linearly separable.

2.3.2 Linear Support Vector Machines

In the separable case, a support vector machine constructs a canonical linear classifier by maximizing the minimum margins of the points of each training class. Since the smallest margin is inversely proportional to $\|\mathbf{w}\|$, maximizing the margin means minimizing $\|\mathbf{w}\|^2$. Accordingly, a maximum margin classifier hyperplane is achieved by solving the hard margin ***primal optimization problem***:

$$\begin{array}{ll} \min\limits_{\mathbf{w},b} & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} & y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1 \quad \text{for all } i=1,...,m \end{array} \tag{2.7}$$

In the non-separable case, a ***slack variable***, $\xi_i \geq 0$, must be introduced for each training example. The slack variable changes the constraints to

$$y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i \quad i=1,...,m \tag{2.8}$$

which allows for a certain degree of misclassification. However, by making ξ_i large enough, the constraints could always be met and would therefore be useless. To avoid large slack variables, they are penalised by adding the sum of all ξ_i to the expression to be minimized. The result is the soft margin primal optimisation problem,

$$\begin{array}{lll} \min\limits_{\mathbf{w},b,\xi} & \frac{1}{2}\|\mathbf{w}\|^2+\frac{C}{m}\sum\limits_{i=1}^{m}\xi_i & C>0 \\ \text{subject to} & \left\{\begin{array}{l} y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i \\ \xi_i\geq 0 \end{array}\right. & \text{for all } i=1,...,m \end{array} \tag{2.9}$$

The cost variable C provides the possibility to adjust the trade-off between the classification error and the complexity of the classifier function. This is illustrated in Fig. 2.3. A too complex classifier function will lead to over-fitting of noisy data, so usually a simple function that explains the data well is preferred to a complex one without classification errors.

2.3.3 The dual optimisation problem

In order to find a representation of the optimisation problem that is suitable for numerical solving, we introduce the ***Lagrangian*** for the soft margin problem Eq. (2.9):

$$L(\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})=\frac{1}{2}\|\mathbf{w}\|^2+\frac{C}{m}\sum_{i=1}^{m}\xi_i-\sum_{i=1}^{m}\alpha_i(y_i(\mathbf{x}_i\cdot\mathbf{w}+b)-1+\xi_i)-\sum_{i=1}^{m}\beta_i\xi_i \tag{2.10}$$

where $\alpha_i = 0$ and $\beta_i = 0$ are called Lagrange multipliers, one for each inequality constraint in Eq. (2.9). It can be shown that in order to solve Eq. (2.9), the Lagrangian should be minimized with respect to $\mathbf{w}$, b and $\boldsymbol{\xi}$, and maximized with respect to $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, which means that the sought solution is a saddle point. The Kuhn-Tucker theorem

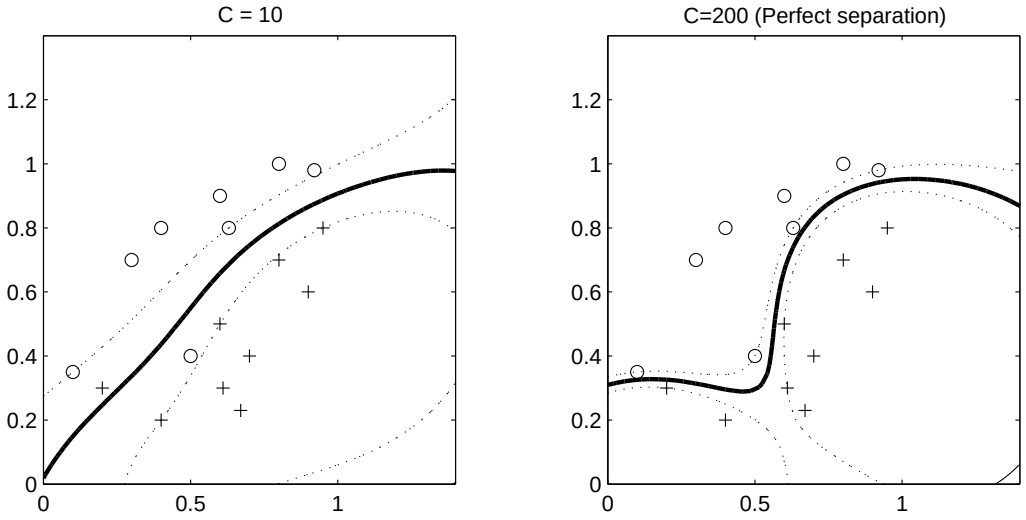

Figure 2.3: Showing the effects of different values of C with slack variables shown for misclassified points. In this case, the SVM is non-linear

(see [2]) concludes that a necessary and sufficient condition for a point $(\mathbf{w}, b, \boldsymbol{\xi})$ to be an optimum solution of Eq. (2.9) is the existence of $\boldsymbol{\alpha}$ such that

$$\left.\begin{aligned} \frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \mathbf{w}} &= 0 \\ \frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial b} &= 0 \\ \frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \boldsymbol{\xi}} &= 0 \end{aligned}\right\} \quad \text{(Saddle point equations)}$$

$$\left.\begin{aligned} \alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i) &= 0 \\ y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i &\geq 0 \\ \alpha_i \cdot \beta_i &\geq 0 \end{aligned}\right\} \quad \begin{array}{l} i = 1, ..., m \\ \text{(Kuhn-Tucker conditions)} \end{array} \tag{2.11}$$

The 4th relation implies that only training points on the smallest margin or with a negative margin will render non-zero α:s, and these points are called the ***support vectors***. Consequently, as long as the training set is reasonably separable, the number of support vectors will only be a small fraction of the training set.

Taking partial derivatives of Eq. (2.10), the saddle point equations yield the following expressions:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} = 0 \quad &\Rightarrow \quad 0 = \sum_{i=1}^{m} \alpha_i y_i \\ \frac{\partial L}{\partial b} = 0 \quad &\Rightarrow \quad \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \\ \frac{\partial L}{\partial \boldsymbol{\xi}} = 0 \quad &\Rightarrow \quad 0 = \frac{C}{m} - \alpha_i - \beta_i \end{aligned} \tag{2.12}$$

Inserting these expressions into the primal optimisation problem Eq. (2.9), and using the Kuhn-Tucker conditions from Eq. (2.11), result in the ***dual optimisation problem***,

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \boldsymbol{x}_j \\ \text{subject to} \quad & \begin{cases} 0 \leq \alpha_i \leq \dfrac{C}{m} & \text{for all } i = 1, ..., m \\ \displaystyle\sum_{i=1}^{m} \alpha_i y_i = 0 \end{cases} \end{aligned} \tag{2.13}$$

Eq. (2.13) is a quadratic programming problem, which can be solved for $\boldsymbol{\alpha}$ by using one of many possible numerical methods, preferably one that takes advantage of the sparsity of the $\boldsymbol{\alpha}$:s to speed up the calculations. A commonly used method is the ***Sequential Minimum Optimisation*** (SMO) as proposed in [5]. The bounds on the α_i:s are called the box constraints, as the cost parameter C forces $\boldsymbol{\alpha}$ to be constrained within a hypercube with sides $\frac{C}{m}$.

2.3.4 Extension to a non-linear SVM

The power of SVMs lies in the way it can perform the classification in a transformed, high dimensional feature space. The advantage of a higher dimension feature space is an increase in the number of possible ways to separate a set of training points by using a linear classifier, which means that a larger part of the training set can be correctly classified. Note that in the transformed feature space, the classifier is still a linear hyperplane, but in the original feature space it is non-linear.

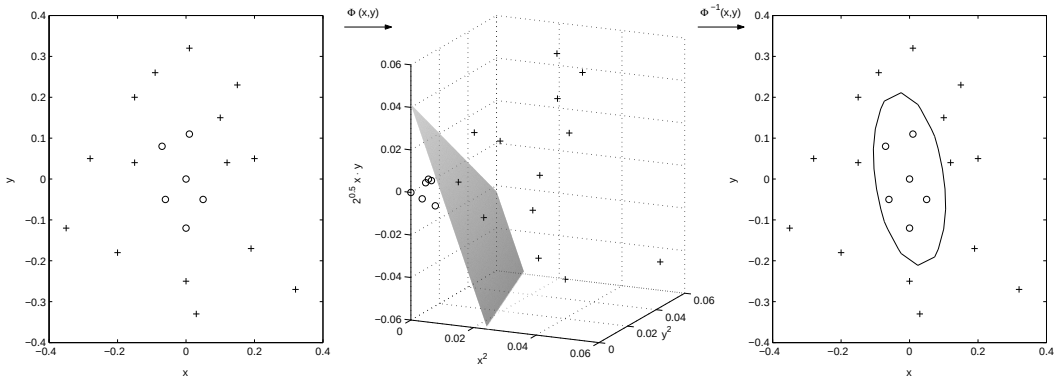

Figure 2.4: Illustrating how a linear plane can be used in the transformed space to produce a non-linear classifier in the original feature space.

The feature space is transformed by a mapping

$$\mathbf{x} \in \mathbb{R}^N \rightarrow \boldsymbol{\Phi}(\mathbf{x}) \in \mathbb{R}^M \tag{2.14}$$

The fact that the classifier in the obtained $\mathbb{R}^M$-space is linear implies that all the results found in the previous sections for linear SVMs can be used. The points $\mathbf{x}_i$ in Eq. (2.13) are simply replaced by the transformed points $\mathbf{\Phi}(\mathbf{x}_i)$, which changes the expression to be maximized to

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{\Phi}(\mathbf{x}_j) \tag{2.15}$$

The truly useful property of the dual formulation is that the mapping $\mathbf{\Phi}(\mathbf{x})$ only appears in the scalar product. The trick is now to replace the scalar product with a ***kernel*** $k(\mathbf{x}_i, \mathbf{x}_j)$, a function that returns a value that is taken to be the scalar product of two transformed vectors. It is not crucial to know exactly what mapping the kernel corresponds to, since the critical characteristic is the higher dimensional feature space that it implicitly introduces. However, the *category* from which the kernel is chosen will greatly affect the obtained classifier function.

Using kernels, only the computational cost of evaluating the kernel function is added to the linear problem. No calculations need to be done in the transformed space and consequently the use of a non-linear classifier function becomes feasible. The problem to solve comes down to a quadratic optimization problem, just as in the linear case, with the addition of calculating the kernel function for all combinations of two support vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. The resulting ***Support Vector Classifier*** (SVC) function can be expressed as a linear combination of the kernel functions as

$$f_\alpha(\mathbf{x}) = sign\left(\sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b\right) \tag{2.16}$$

Evidently, the classification function is a linear combination of kernel functions evaluated at different points in the non-transformed feature space. This can be helpful in getting an intuitive understanding for what classifier function a kernel is capable of building. Common kernels include **polynomials** of degree d

$$k(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x} \cdot \mathbf{x}_i)^d \tag{2.17}$$

and the **gaussian kernel** of width $\sigma > 0$

$$k(\mathbf{x}, \mathbf{x}_i) = e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{c}} \tag{2.18}$$

which is often referred to as a radial basis function since it is hyperspherically isotropic. Unlike a polynomial function, the gaussian is compact and as such a good basis functions, providing a rich set of classifier functions.

2.3.5 Multi-class classification

There are different ways to extend SVMs to be able to perform classification when there exists multiple classes. The simplest and most common ones involves several binary classifiers that are combined to function as a multi-class classifier. Two such methods are briefly explained below:

One Versus the Rest

For this method, a set of binary classifier functions $f^1(\mathbf{x}), ..., f^M(\mathbf{x})$, each trained to separate one class from *all* the others, is constructed. A pattern $\mathbf{x}$ is taken to belong to the class whose classifier function gets the largest value when evaluated at $\mathbf{x}$.

Pairwise Classification

In this case, a binary classifier is created for each possible pair of classes. There are then different strategies on how to use these classifiers to obtain the final result. One method is to test each classifier on the pattern, and for every test, a vote is added to the one of the two labels that is chosen by the classifier. The class that has accumulated the largest number of votes when all classifiers have been used is the class that is assigned to the pattern. Another way is to organize the binary classifiers in a directed acyclic graph as in Fig. 2.5. This way, impossible classes are ruled out early in the classification process and the number of necessary trials is reduced. The accuracy is usually similar to the voting procedure mentioned, as shown in (reference), while the speed of classification is often significantly reduced.

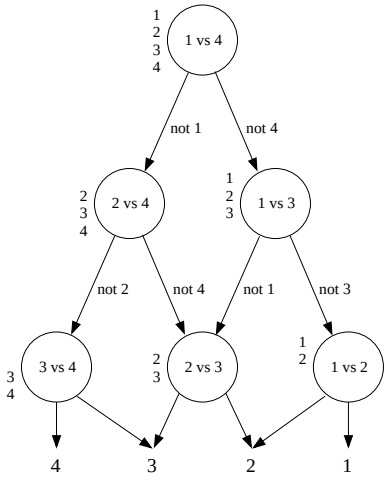

Figure 2.5: Each node represents one binary classifier. Only one path through the graph is traversed so only a fraction of all binary classifiers will be used.

2.3.6 Normalization and scaling

For most reasonable kernels, the subspaces will have the same possibility to fit to the training data in the whole feature space. This is generally a favorable characteristic, but sometimes one would want the subspaces to fit tighter in certain areas of the feature space and looser in others. Instead of trying to adjust the kernel, it is in such cases more reasonable to transform the features. A typical example of this kind of feature is the area of regions in the images. We might expect the subspaces to have a more complicated appearance around small areas than around bigger ones. In such cases we

can transform the feature to exhibit a balanced behavior all over the feature space. In the case of the area, taking the logarithms will be appropriate.

For the same reasons, all feature variables need to be scaled so that their variances are similar. Usually this is done by normalizing the feature, but it is also possible to simply scale the variables into some interval, for example $[0, 1]$.

3

Segmentation

While per-pixel classification of remotely sensed multi-spectral images may be a task that is very suitable for a computer, the human eye and brain is superior in extracting relevant objects from the image. Nevertheless, many algorithms dedicated to finding relevant segments in images in a human-like fashion have been developed in the last 20 years, and while they are still far away from performing perfect segmentations, they can often be of great value to the remote sensing operator. The segmentation of an image provides a whole new range of possibilities for semantic image understanding that is largely unexploited to this date. In this diploma thesis, some of the additional information that can be retrieved from regions rather than single pixels is used to improve classification results, but most possibilities are left for the future.

The simplest segmentation is the one resulting from a regular per pixel classification algorithm where regions are formed by connected pixels of the same label. However, the result of that method tends to be noisy, and it does not take advantage of the information that gradients in the image carry. To overcome these weaknesses other approaches are often used, normally some combination of merging and splitting the image. We make use of a split-first-then-merge method inspired by [6] where segmentation is performed in two main stages. First, the image is split into many small segments by some method, such as watershed or quadtree segmentation. Then these small segments are merged into larger regions with uniform texture and/or intensity, according to a homogeneity criterion.

3.1 Initial Segmentation

The beginning step of the split-and-merge segmentation may be the most important part of the process as it sets the ultimate constraints on the final regions. Each initial segment should be part of at most one object in the image. A logical assumption is that for such a segment, any strong edges should be on its border. More generally, the segments inner pixels should be rather homogeneous and as such fulfill some homogeneity criteria. A lack of strong edges will not automatically mean a segment is homogeneous, since the

intensities can change slowly spatially, but still cover a large range of intensities. To conclude, the ideal initial segment is

a) Part of only one object.

b) Rather homogeneous.

c) The majority of the gradient on its border is strong.

In practice we often need to compromise these requirements due to slowly changing intensities as described above. Since the segments will be merged into larger regions in the merging stage, the image should be over- rather than under-segmented to ensure that the requirement a) is met.

The simplest initial segmentation is to use the individual pixels as segments. However, this way the number of segments will be very large, and the merging process will be time-consuming. It is also difficult to find consistent edges when merging individual pixels since each pixel that is merged to a segment is merged independently of its surroundings. There are several more sophisticated methods that provide larger initial segments and avoid some of the drawbacks of using every pixel as starting segment. Two of those, the watershed segmentation and the quadtree segmentation are described in the following sections.

3.1.1 Watershed segmentation

This type of segmentation is inspired by topography concepts ***watersheds*** and ***catchment basins***. The watersheds are the lines which divide catchment basins and, using the analogy with topography, where water would be pulled down by gravity into surrounding catchment basins, as illustrated in Fig. 3.1. What we seek is the watershed lines from the gradient image, that is, the lines of pixels that locally have the highest gradient.

Watershed segmentation results in one segment from each local minimum in the gradient image which are precisely the catchment basins. Starting from a "water level" of zero, the algorithm applies a kind of successive flooding as illustrated in Fig. 3.2. The water level is increased one step at a time, and in each iteration the pixels that are on the same level as the water are added to the neighboring segment. Whenever two segments meet, the pixels on the border between them are marked as watersheds. The corresponding pixels in the gradient image are given the maximum gradient level, and a wall is thus built between the segments, preventing them from merging. The water level is increased until all pixels are either part of a segment or marked as a watershed.

A problem with the result is that the watersheds themselves will be 1 pixel wide, which will limit the possibility to resolve small detail in the image. A brute-force solution is to simply upscale the image by a factor two, using bi-linear or bi-cubic re-sampling. Resizing the image does of course not add any information; it simply gives room in the image for both the borders between regions *and* the regions themselves in case of fine detail. The downside is that the image will be 4 times larger. A more elegant solution would be to describe the watersheds mathematically by means of anchor points

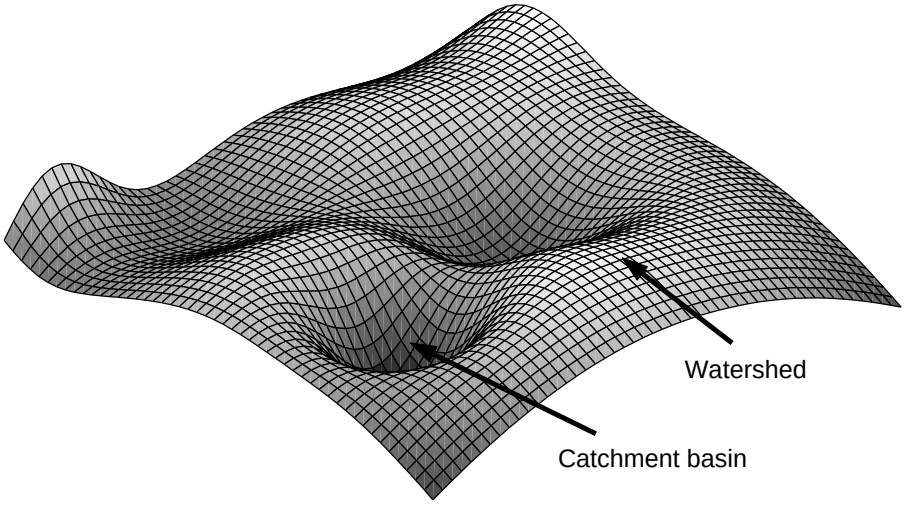

Figure 3.1: This 3-dimensional gradient image demonstrates the analogy to topography.

in order to make them infinitesimally thin. Presumably, such a process would be less running-time efficient.

Multi-spectral watershed segmentation

Remotely sensed images normally consist of several bands and including more than one band in the segmenting process can improve the result. This can be accomplished by calculating the gradients for each band and then combining them into one gradient image before the watershed algorithm is run. The different bands can be weighted to compensate for differences in contrast or to change the influence of certain bands. A reasonable way to combine the bands is using the length of the intensity vector formed by the bands at every pixel of the image. For the four bands in our test images, with weighings included, the combined gradient is then calculated as

$$I = \sqrt{\frac{(w_{\mathrm{B}}\mathrm{B})^2 + (w_{\mathrm{G}}\mathrm{G})^2 + (w_{\mathrm{R}}\mathrm{R})^2 + (w_{\mathrm{IR}}\mathrm{IR})^2}{w_{\mathrm{B}}^2 + w_{\mathrm{G}}^2 + w_{\mathrm{R}}^2 + w_{\mathrm{IR}}^2}} \tag{3.1}$$

were $w_i, i \in [B, G, R, IR]$ are the weights for each gradient band.

Reducing the number of segments

Since most gradient images will contain many local minima, watershed segmentation results in a great number of segments. One way to avoid the segments that result from small variations in the gradient image is to low-pass filter the image in order to remove these gradients. However, low-pass filtering will also decrease the precision, for example the two gradients from a thin road in a field could diffuse into one. We have found another method that removes false segments without low-pass filtering. The trick is to set all gradients below a threshold value to zero before doing the watershed

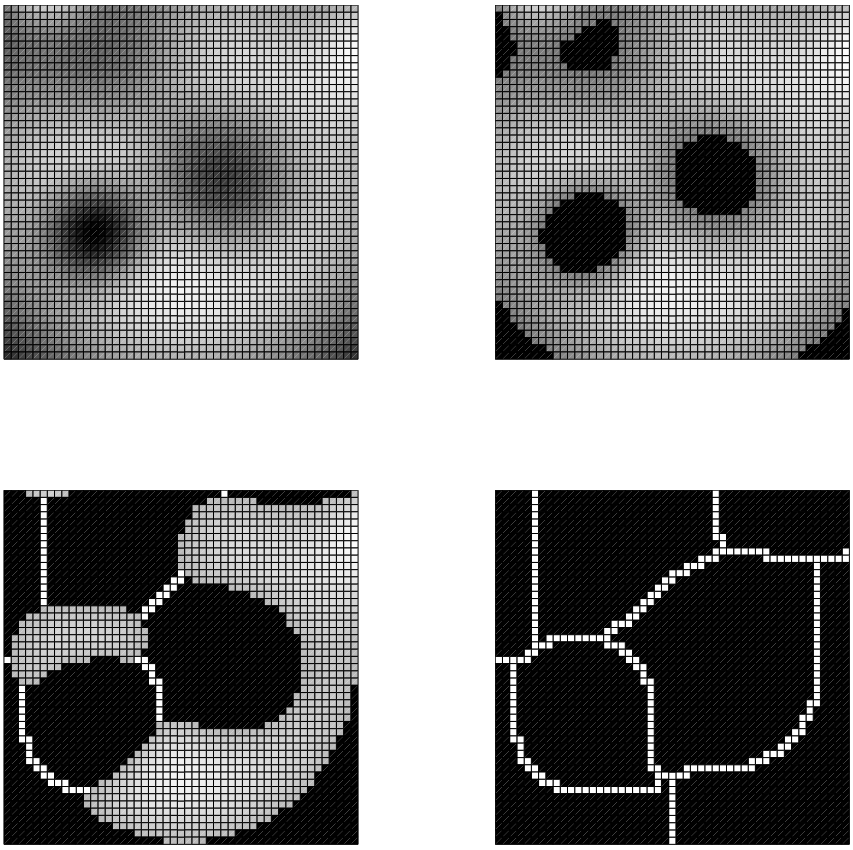

Figure 3.2: The gradient image from Fig. 3.1 is flooded until only the watershed lines remain.

segmentation. This results in large continuous fields of zeros that will automatically belong to the same catchment basin. Using this method, the number of segments can be greatly reduced without losing detail, as is likely when using low-pass filtering.

3.1.2 Quadtree segmentation

Unlike watershed segmentation the quadtree method uses the intensities of the image itself to find homogeneous regions. The image is tested according to some homogeneity criterion; for example, the image could be considered homogeneous if the difference between the highest and lowest intensity is less than a pre-selected threshold. If the homogeneity criterion is not met, the image is split into 4 equally sized squares that are again tested for homogeneity. This procedure is repeated for the resulting squares until every part of the image belongs to a square that is homogeneous or of the smallest possible size. A drawback is that the segments will be squares at predetermined positions. As a consequence, there will be many small squares along borders in the image, as it is

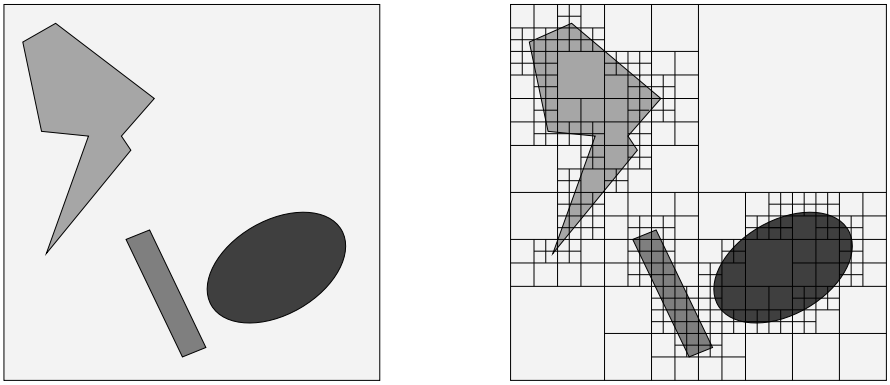

Figure 3.3: A typical quadtree segmentation of a toy image. The need of many small squares along the edges of the objects is evident.

unlikely that the borders would be only vertical or horizontal lines along the borders of bigger squares. However, by examining Fig. 3.3 we can see that two neighboring segments that share a long edge are likely to be merged, so many of the regions from the quadtree segmentation could be joined as an intermediate step before the real merging procedure.

Including information from more than one band in the segmenting can be achieved in a similar manner as for the multi-spectral watershed segmenting. A quadtree segmentation is done for each band, possibly with different homogeneity criterions, and the resulting squares are simply superimposed on each other. The result then becomes the combination of the finest segments from the decompositions of each band.

3.2 Merging

The merging of two regions should be carried out if a certain homogeneity criterion is met. In addition to this criterion, the order in which the regions are merged will also play a significant role for the final outcome. There are several possible heuristics for how to select two regions to be tested for merging, and the following that was used in this thesis is only one possibility:

1 Start with the smallest region that is not marked *unmergable* or *merged*.

2 Try to merge the region with its neighbors, and merge it with the neighbor that will best fit the merging criterion, unless it cannot be merged with any of its neighbors.

3 If no merge could be performed, the region is marked *unmergable*.

4 If merging succeeded, create the new region and reset any of this region's neighbors that were marked unmergable. Remove the two original regions.

Repeat this until all regions are marked unmergable.

It then remains to find a good merging criterion. The criterion can be formulated as a binary function whose two possible values represent either merge or no merge.

$$f_{merge} = sign(f(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p})) : \mathbb{R}^{2 \cdot N + P} \rightarrow \{\pm 1\} \tag{3.2}$$

The input variables are the features of the two regions ($\mathbf{x}_1$ and $\mathbf{x}_2$) and the properties of the border between them ($\mathbf{p}$). It would also be possible to use contextual information as input.

3.2.1 Fixed merging criterion

Because of the possibly high dimensional input of Eq. (3.2), it can be very difficult to find the function. A smaller amount of features can be selected in order to provide a simple merging function with acceptable performance. One possible criterion would be to use only the mean of the gradient values on the border pixels, and the mean intensity of one or two bands of the image. The merging would then be carried out if the mean gradient is low and the mean intensities do not differ significantly between the two regions. Using only simple relations like this, the merging function can be found manually through some intelligent trial and error approach.

3.2.2 A novel approach - intelligent merging

It might be nice to include more features than the ones in the previous section. For example, the areas of the regions could be used to make merging smaller regions easier compared to larger ones. Also various shape measures of the regions and the length of the border, as well as many other measures could be usable. Adding more features, it may still be possible to manually find a function, but it would require lots of work, and the outcome is likely to be less than optimal.

To address these issues, we invented a machine learning approach to find the merging function. What is sought is the function (Eq. (3.2)) that divides the function input variable space in two subspaces, one for the decision to merge and one for no merge. An SVM with proper training input should be able to provide such a merging function for a high dimensional input vector. How to determine which features to use, and especially how to construct a good training set that covers all possible combinations of two regions are difficult problems. One option is to collect the necessary information by manually selecting which regions to merge or not, and store the result (merge or not merge) together with the features of the regions involved in the merging trial. This is the procedure that was adopted for the experiments that are described in Chapter 4.

Another similar method is to manually segment the image, and then let the merging heuristic run, but instead of using a classifier for determining which segment pairs to merge, the a priori segmentation would be consulted to find out whether to merge or not. The features of every tested pair of regions and the test result would be collected and serve as a base for further analysis. A third approach would be to start with a fixed merging criterion and use the results thereof to train the SVM. Whenever the merging is

not satisfactory, new training examples could be added to customize the fixed merging criterion.

4

Results

In this chapter we describe the experiments and present the results from evaluating the algorithms in Chapter 2 and Chapter 3. The results of classification algorithms are commonly judged by the Kappa value, as described on p.207 of [3], and confusion matrices. We do not use these measures, since the purpose of this work is not to compare different methods of classification, but rather to investigate the properties of the Support Vector Machine, and to find out what impact segmenting the images has on classification. In this case, percentages of correctly classified training examples will provide the same or better information.

4.1 Classes and training examples

The set of classes was selected according to the content of the images. In all classification experiments in the following sections, the classes in the legend of Fig. 4.1 were used. It can be argued that they are quite few, but as will be seen from the results, this was an appropriate amount. Using more classes, we would risk getting random classification accuracy, since classifiers trained by training examples that overlap will not be consistent. It is thus preferable to use a reasonably separable training set, which means the number of classes should be chosen accordingly.

The pixels used as training examples were labelled by hand from the RGB- and IR-images. Mixed pixels were avoided but due to human errors, the labels cannot be guaranteed to be 100% correct. Consequently, the input may be a bit noisy, but experiments have shown that the training set is still rather well selected. Furthermore, the training set is certain to be incomplete for high dimensional feature spaces. Although limiting the possibility of achieving perfect results, the training input very well represents a real world scenario - no operator, no matter how skilled, could be expected to label as many training examples as feature spaces of very high dimensions require.

Land cover label	Quantity
Water	114
Buildings	503
Road/Railway	974
Crops	507
Broadleaf vegetation	1123
Evergreen	132
Grass	449
Others	239

Figure 4.1: The land cover classes and the number of training examples of each class.

.

4.2 Pixel-based classification

This section has somewhat dual purposes; it demonstrates the usability of SVMs in pixel-based classification of high resolution satellite images, and it also provides an analysis of the SVM properties in a more general fashion. The experiments were carried out in order of ascending complexity, starting with a minimum of features for classification. In addition to the features, the parameters of the support vector machine (kernel, cost-parameter C) were varied.

Unless otherwise mentioned, 1011 training examples out of 4041 labelled pixels were used for training. The remaining pixels were used to calculate the classification errors.

4.2.1 SVM parameters - kernel and cost variable

To gain some understanding of how the kernel and the cost variable C affect the classification, a 2-dimensional feature space composed of the relatively uncorrelated near-infrared and red bands was used. Although using only two features gave somewhat unsatisfactory classification results, this allowed to visualize and qualitatively judge the ability of the chosen kernel to partition the feature space into classes.

The classification error rates, using several combinations of SVM parameters, are presented in Table 4.1. According to these results, the majority of the tested kernels are capable of producing similar error rates with a properly chosen C. However, this should be considered as a side-effect of the low dimension that makes the classes inseparable - even a perfect classifier would render an error rate of approximately 20% with the same input. The large differences in the partitioning of the feature space, as shown in Fig. 4.2 are certainly more interesting.

Without taking generalization performance into consideration, the best results were achieved with gaussian kernels of widths σ between 0.05 and 0.5. This was expected since the training examples were scaled to the interval [0 1] and a typical cluster was then in the order of 0.1. It is logical that a radial basis kernel of the same width would be able to partition the space well. As for the polynomial kernels, we do not get the same resolution, but still very respectable performance. Since a polynomial kernel will

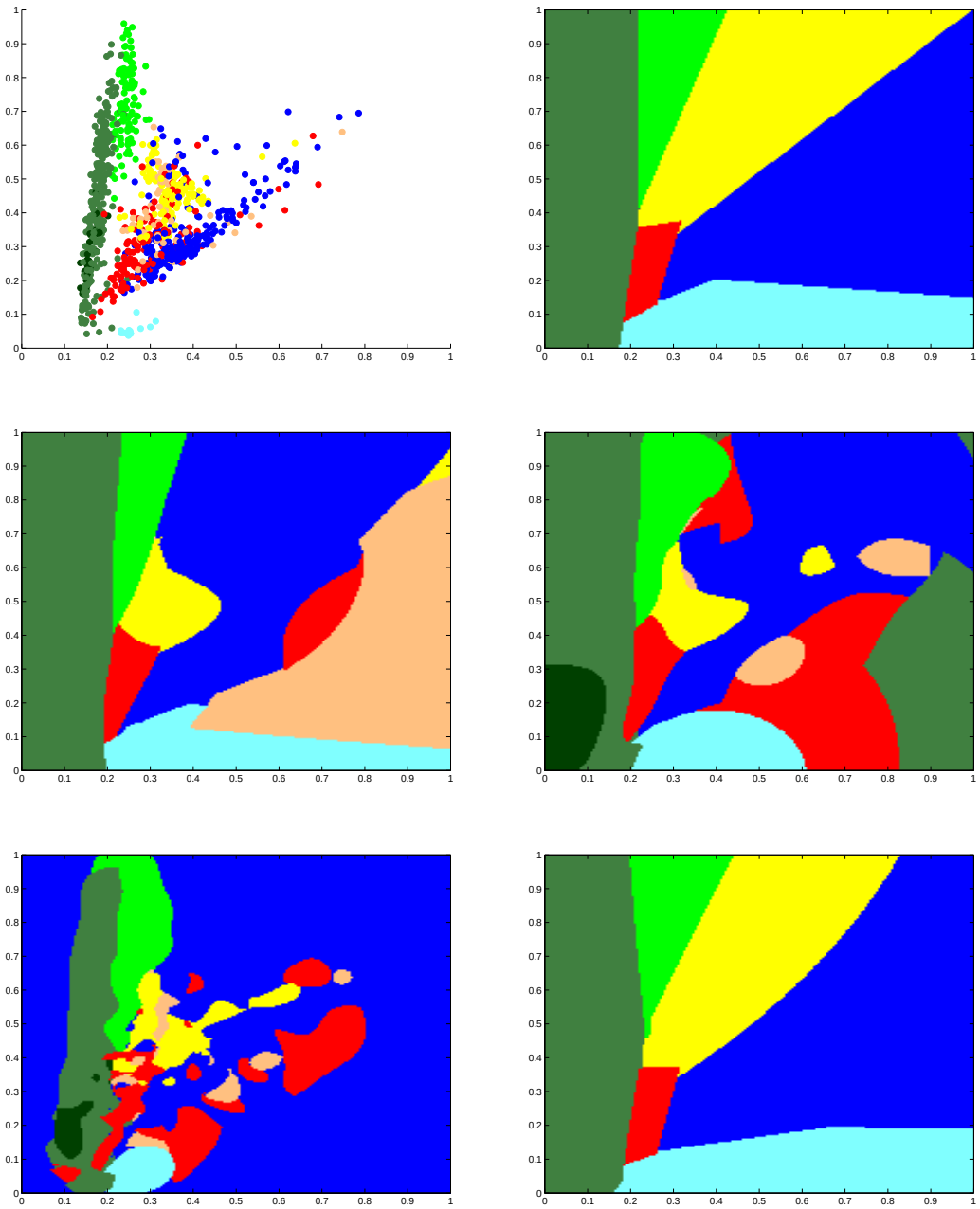

Figure 4.2: (a) the training examples in 2D, (b) linear kernel, C=1000 (c) polynomial kernel, degree=4, C=1000 (d) gaussian kernel, $\sigma = 0.2, C = 1000$) (e) gaussian kernel, $\sigma = 0.05, C = 10000$) (f) gaussian kernel, $\sigma = 2, C = 100$)

Kernel	C=1	C=10	C=100	C=1000	C=10000	C=100000
Linear	47.5%	31.3%	25.7%	24.9%	24.7%	24.7%
Polyn. degree=2	42.0%	27.1%	25.0%	23.8%	23.0%	22.8%
Polyn. degree=4	37.3%	25.7%	24.2%	23.0%	22.8%	22.5%
Gaussian $\sigma = 2$	47.5%	30.9%	25.5%	23.9%	23.0%	22.7%
Gaussian $\sigma = 1$	37.7%	25.8%	23.9%	23.0%	22.7%	22.5%
Gaussian $\sigma = 0.5$	26.6%	23.6%	22.8%	22.6%	22.6%	22.2%
Gaussian $\sigma = 0.2$	24.6%	22.9%	22.7%	22.0%	21.3%	21.3%
Gaussian $\sigma = 0.1$	23.5%	22.6%	21.6%	21.4%	21.1%	21.1%
Gaussian $\sigma = 0.05$	22.1%	21.1%	21.4%	21.4%	21.5%	21.8%
Gaussian $\sigma = 0.02$	21.5%	21.0%	22.3%	23.2%	24.0%	23.9%

Table 4.1: The error rates from classifying a 2D feature space using different combinations of SVM parameters

not provide the same flexibility, and thus possibility of over-fitting the classifier, it can be a good choice, especially for a high dimensional feature space. However, gaussian kernels of large widths, σ, produce classifiers that are very similar to the ones resulting from polynomial kernels. Because of this, only gaussian kernels will be used in the following experiments.

As shown, the kernel will greatly affect the resolution power of the classifier. At the same time, since the classes are severely overlapping, over-training the classifier would increase the amount of visual noise as is evident when studying the classified images in Fig. 4.3. The best classifier is a compromise between generalizing capabilities and ability to correctly classify the training examples. In the case of remote sensing, one would usually prefer a classifier where each subspace is a well defined, rather compact, space that explains the data well. Disjoint subspaces, belonging to the same class would be a sign of an over-trained classifier or of poorly chosen classes. On the other hand, too simple subspaces would give poor classifying correctness.

The number of support vectors was found to closely reflect the performance of the classifier. In the case of a thin gaussian the high number of support vectors indicates that the function might be over-fitted, since a complicated classifier function require a large set of basis functions. Another consequence of many support vectors is high error rates since all the training points that would be misclassified each correspond to one support vector. Accordingly, the ideal machine would use as few support vectors as possible without sacrificing accuracy.

4.2.2 All intensity bands

It is clear from figure Fig. 4.2a) that the training examples of different classes overlap in two dimensions, even when two uncorrelated features like the intensities from the red and near-infrared bands are used. Increasing the number of features could ameliorate the situation if the training examples contain relevant information in the new dimensions. Such features would lead to a decrease of the overlap between training examples,

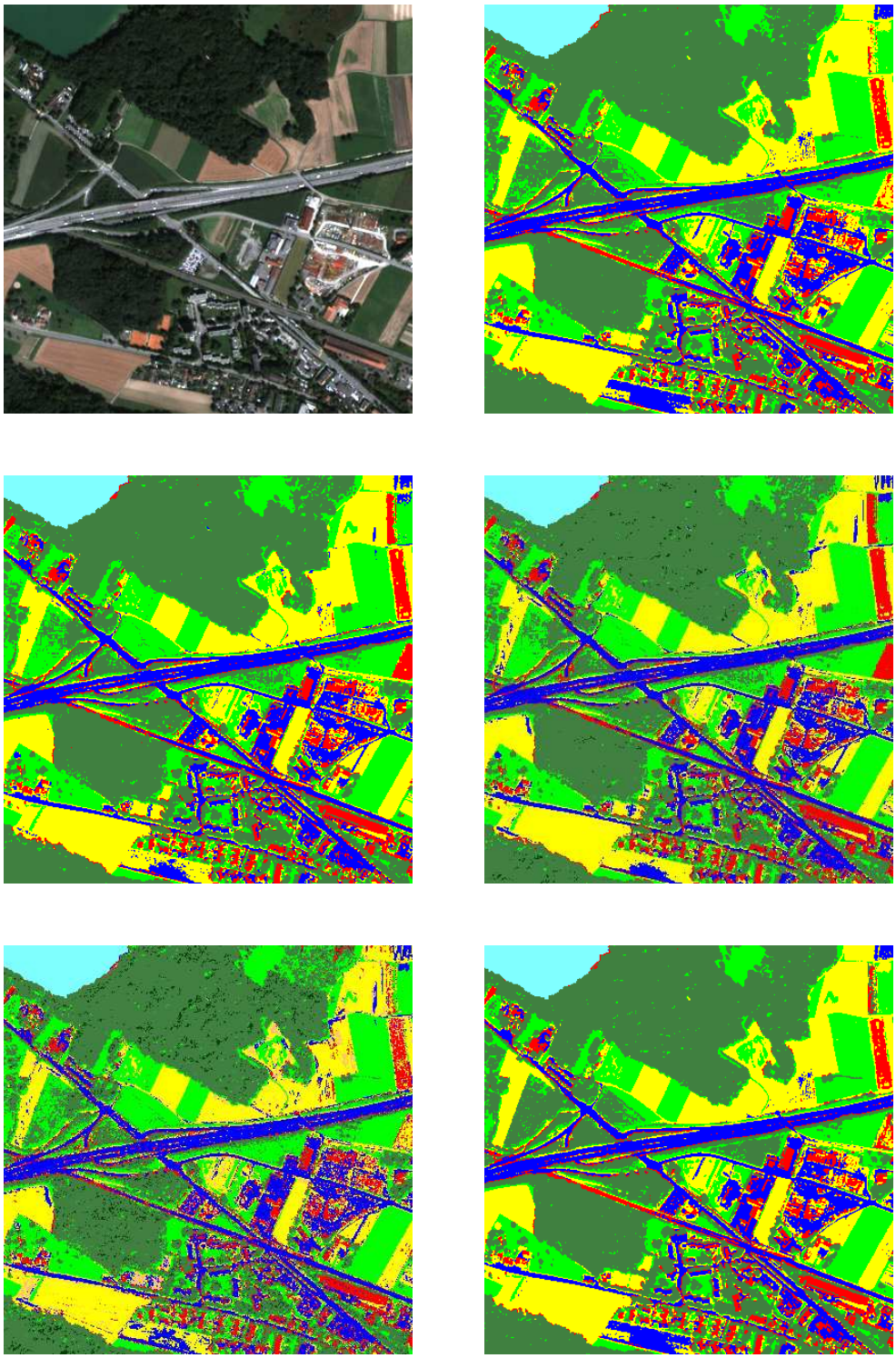

Figure 4.3: The visual results. (a) the original image (b) linear kernel, C=1000 (c) polynomial kernel, degree=4, C=1000 (d) gaussian kernel, $\sigma = 0.2, C = 1000$) (e) gaussian kernel, $\sigma = 0.05, C = 10000$) (f) gaussian kernel, $\sigma = 2, C = 100$)

improving the discriminating possibilities. Since a dimension higher than 3 makes visualization tedious and difficult to interpret, all results from multi-band classifications in this and following sections will be accounted for as percentages of correctly classified pixels and/or classified images.

Using the knowledge from the 2-dimensional classification, the combinations of parameters to be tested were narrowed down to produce the results in Table 4.2. Subjectively, the best compromise between classification accuracy and noise was achieved

Kernel	C=10	C=100	C=1000	C=10000
Gaussian $\sigma = 0.5$	20.7%	18.6%	17.8%	17.0%
Gaussian $\sigma = 0.2$	19.2%	17.9%	16.8%	16.4%
Gaussian $\sigma = 0.1$	17.6%	17.0%	16.6%	17.8%
Gaussian $\sigma = 0.05$	16.4%	17.3%	18.2%	18.9%

Table 4.2: The classification errors using the IR, R, G and B bands.

with the gaussian kernel, $\sigma = 0.2$, $C = 1000$ - the same parameter values as in the 2-dimensional case. There is still a good portion of noise in the image, which can be seen in the larger crop fields and pastures of Fig. 4.4 a). The classification of urban areas containing lots of detail are of course noisy since they are spectrally very heterogeneous. Although representing the ground use rather well, a less correct classification may be preferred as clearer segments then would be formed. The other image in Fig. 4.4 has worse accuracy, but also less noise. It does not use the evergreen label at all but the first classification does not classify the evergreen very well either, and it can be argued that the less noisy of the two would be the prefered one. There is no single answer to which classification is the best, since the demands will vary for each application.

4.2.3 Textural features

The four textural features described in Section 2.2.2 were calculated for the near infrared and green bands using 5×5 and 9×9 windows and 64 gray levels. To find out what the textural features would possibly improve in the classification, each permutation of window size, band and feature was tested together with each of the bands. The gaussian kernel of width $\sigma = 0.2$ and cost parameter $C = 1000$ was used.

Feature	5x5 G	5x5 IR	9x9 G	9x9 IR
ENT (Entropy)	15.1%	15.1%	15.4%	14.8%
CON (Contrast)	15.0%	15.1%	14.6%	14.8%
ASM (Angular second moment)	16.0%	15.7%	15.9%	16.1%
IDM (Inverse difference moment)	16.2%	14.7%	15.0%	14.9%

Table 4.3: The errors of each textural feature in combination with the near infrared (IR), red (R), green (G) and blue (B) bands.

Textural features from the near infrared band improved visual results more than

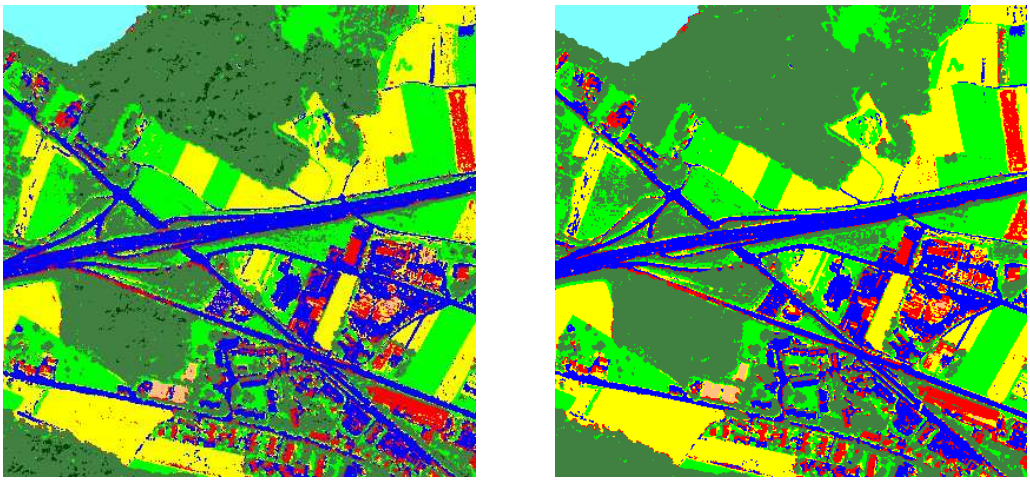

Figure 4.4: (a) gaussian kernel, $\sigma = 0.2, C = 1000$) (b) gaussian kernel, $\sigma = 0.5, C = 100$)

than those calculated from the green band which can be a consequence of the higher contrasts in the infrared image. A window size of 9×9 was almost always better than 5×5, although it is difficult to judge the improvements of a smaller window, as they may be subtle but correct. In general the 9×9 windows gave better error rates and de-noising than 5×5 windows which indicates that 5×5 pixels is not enough to characterize the textures in the images. For each of the textural features the general impressions were as follows:

- The entropy seems to aid in removing spurious pixels in homogeneous areas, although noise tends to be added to forests and other areas of higher disorder. It also made discriminating fields from buildings with similar spectral footprints possible.
- Likewise, Inverse difference moment removes noise from larger homogeneous areas, but it does not have a negative impact on forests to the same extent as the entropy.
- Unsurprisingly, the Contrast feature often added noise where contrast is high, especially using the 5×5 window in areas with high variability.
- The Angular second moment was a mixed bag, showing both improvements and artifacts. In combination with some other textural features it could in some cases render good results, removing single misclassified pixels.

Table 4.3 shows that using a textural feature always improved the error rate compared to using none at all. To find the optimal combination of these features, a forward-analysis was carried out. Starting with the feature that gave the best error according

to Table 4.3, the feature that improved the error the most was added until no more features remained. The result is shown in Fig. 4.5 together with the number of support vectors.

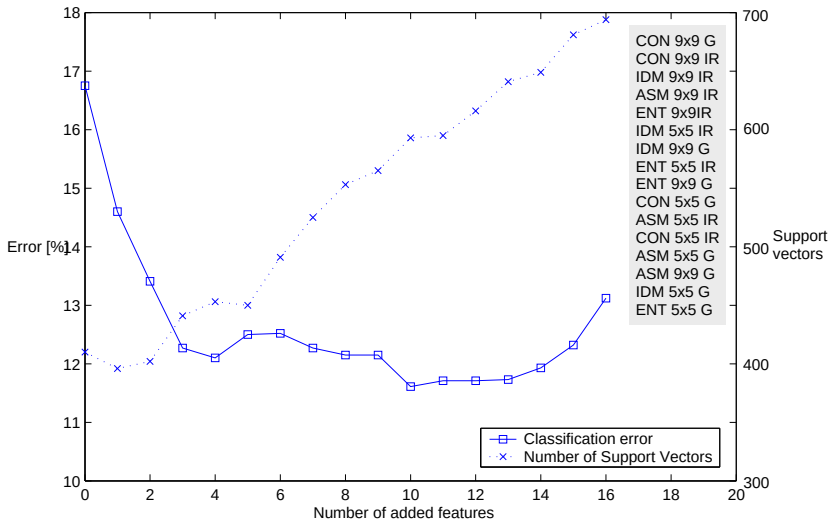

Figure 4.5: The error rates and number of support vectors in the forward analysis. The features in the legend are listed in the same order as the points on the curves.

The lowest error rate achieved was 11.6%, but a closer inspection (Fig. 4.6)a shows that the visual quality is degraded compared to using fewer or no textural features. Using too many features always resulted in strange artifacts which can be explained by the increasing number of support vectors that indicate that the classifier needs more training examples. A good classification error did hence not automatically mean that the visual result was satisfying. Trying different combinations showed that using only one textural feature was usually enough and more robust than using the combination that rendered the lowest classification error. For example, the inverse difference moment of the green band, using a 9×9 window (Fig. 4.6b) is a good choice that improves all homogeneous segments of the image without introducing artifacts in the areas with high variability.

4.2.4 Simple contextual features - smart averaging

Contextual features were implemented according to the simple method described in Section 2.2.4. The window used was sized 5×5 pixels with the weight 2 for the 8 inner pixels and 1 for the the outer pixels. In this experiment eight classes are discriminated and accordingly eight features were added to each pattern. As initial classifications, two different combinations of features were used; 1) the R, G, B and IR bands, and 2) the same bands with the addition of the inverse difference moment from a 9x9 window in the IR band. As usual, a gaussian kernel of width $\sigma = 0.2$ was used and the cost parameter C was set to 1000. The resulting classification error percentages are shown

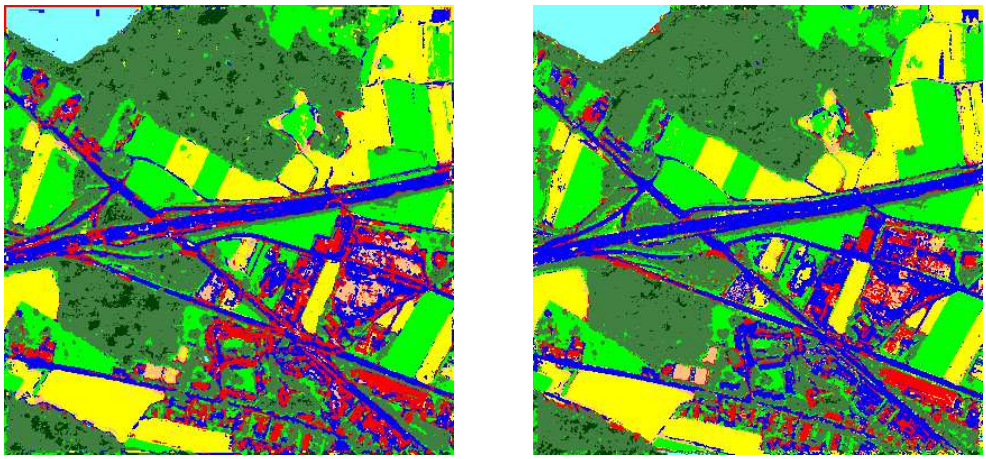

Figure 4.6: (a) best error percentage (b) best visual quality (IDM 9x9 G)

in Table 4.4.

Features / Iterations	0	1	2	3	4	5	6
R G B IR	16.8%	13.1%	12.1%	11.8%	11.7%	10.7%	10.6%
Support Vectors	410	408	356	305	308	298	289
R G B IR IDM9x9IR	14.9%	12.8%	13.1%	11.8%	11.8%	11.8%	12.0%
Support Vectors	401	394	350	317	310	311	305

Table 4.4: The error percentages using contextual information iteratively.

Table 4.4 shows improvements in error percentages, and the resulting image (Fig. 4.7) can also be considered rather good after a few iterations. The image stabilized after about 5 iterations and the results were similar for the two series, with the one using only the spectral features being slightly better. Obviously, adding textural information to contextual classification did not give a better result. The fact that the number of support vectors decreased from about 400 to 300, contrary to the increase seen when evaluating textural features, indicates that surrounding labels facilitate the classification of a pixel in a proper way.

Depending on what result the operator desires, this technique can be usable. The smallest detail information is lost, which means that noise is removed to the same extent. In Fig. 4.7 there is some evidence that contextual knowledge has been learnt by the machine. For example, the small roads are better detected between fields. However, small roads are also detected between fields where there are no roads, demonstrating the fact that correct context is not easily learnt by a machine.

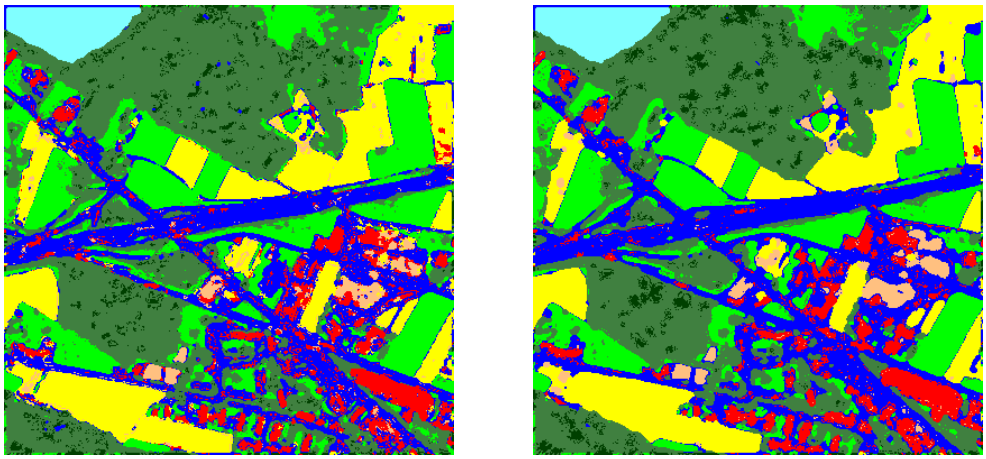

Figure 4.7: The classification result after (a) 2 iterations (b) 6 iterations.

Quantity of training examples

Two combinations of features were used to examine how the amount of training examples affect the performance in different feature space dimensions. The number of training examples was varied between 50 and 4000 and the SVM used a gaussian kernel, $\sigma = 0.2, C = 1000$, giving the error rates shown in

Fig. 4.8.

The resulting two curves behave somewhat differently. Using only the four spectral features, the error rate flattens out as the number of training examples grow. This is evidence that the classifier does not overfit to training data and that the training set is sufficient already at approximately 1000 training examples. The small improvements after this point is mainly due to the fact that a larger portion of the training examples are used also used in the classification and thus in calculating the training error, giving a somewhat unfair advantage of using many training examples.

Adding the three textural features changes the shape of the curve. It is now rather linear in the interval used; the error keeps improving as a higher number of training examples are used. This shows that the higher dimensional feature space, in combination with the chosen SVM parameters, allows separating most training examples and thus causes over-fitting. When using all possible training examples the resulting accuracy will be very good, but only because the same training examples are used for testing the classifier. Since the curve never flattens out it can be concluded that the training set does not cover the feature space well and the classifier can not be expected to generalize well. The visual results do not improve when using more than 1000 examples in both cases, so this would be a proper training set size to use for the actual test image.

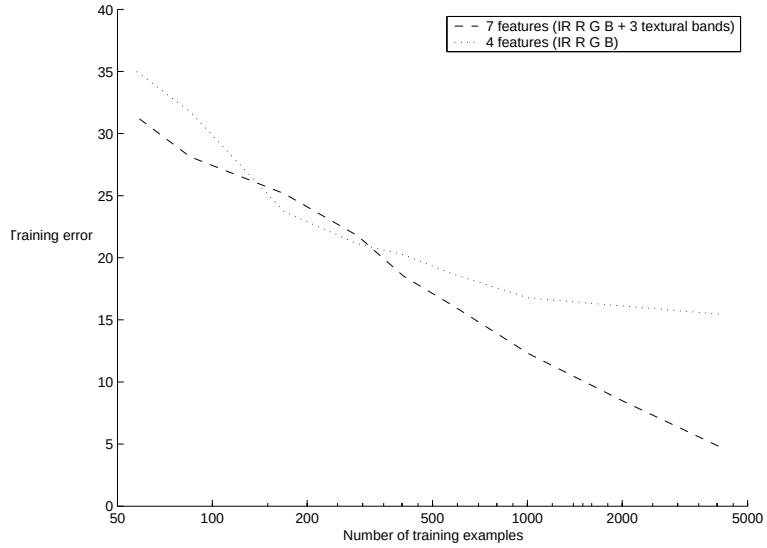

Figure 4.8: The error rates for two different feature configurations while varying the number of training examples.

4.3 Region-based classification

During initial experiments, different methods of splitting and merging the images were tried out and evaluated in order to find one to use for further experiments. The advantages of the watershed method is that it succeeds in finding almost every edge that may be the border of an object. Hence it is a good starting point for merging. On the other hand, the final results using quadtrees for initial segmentation proved to depend on an advanced merging algorithm in order to find object edges. For the reason of this higher complexity, the quadtree segmentation was not further evaluated.

4.3.1 Initial segmentation using watershed segmentation

To calculate the gradients, an edge detection algorithm from the image treatment software Adobe PhotoShop® was used. This algorithm is quite flexible, giving the user the possibility to adjust the size of the convolution window, as well as weighting the brightness of the resulting edges. A large convolution window has the effect of smoothing the gradients, making the edges more continuous. The downside is that some detail will be lost. When the image had been enlarged by a factor 2, a window size of 5×5 was found to give the best compromise between continuity and resolution power.

Following this step, the gradients were combined to include information from all bands in one single image. The initial approach was to use the largest gradient at each pixel of all the bands, but a few trials showed that better results were achieved by using the intensity vector length, Eq. (3.1). To further improve the resulting segmentation, the green band was given a weight of 1.4 and the near infrared band was weighted 0.7. A possible drawback compared to the maximum gradient method is that edges that

only appear in one band will not influence the final result as much as edges that are in all bands, but in reality the differences were not very obvious. A composite gradient image obtained through the described method is shown in Fig. 4.9.

Figure 4.9: A small part of the test image. (a) original, (b) composite gradient from a 5x5 window

Applying watershed segmenting directly on this gradient image results in a large amount of segments (Fig. 4.10a), since each minimum in the gradient image corresponds to one segment. The number of segments could be drastically reduced by setting all pixels having a value under a certain threshold to zero. This way the majority of the minima was removed without erasing important edges (compare Fig. **??**a and b). It was evident that it is important to find a suitable threshold, as setting it too low results in too many segments, and setting it too high will remove important edges. The optimal threshold will vary between different images; for the image in this test the best value was found at around 5% of the maximum composite gradient value, and this could be a reasonable starting point for other images as well.

By using thresholding, the number of segments was reduced by a factor 3 for the test image. For smoother images containing fewer objects the result would be even better. The advantage of this reduction is that memory usage is significantly lowered and that the merging phase will take much less time.

4.3.2 Merging

Finding a good merging criterion proved to be the real challenge. In order to get a statistical base for finding a merging function and for determining which features to use as its input, the merging heuristic described in Section 3.2 was run, manually determining which regions to merge or not in each step. The features involved in each attempt to merge two regions were collected to constitute a training set.

Manual merging is a tedious procedure that requires lots of time so out of necessity

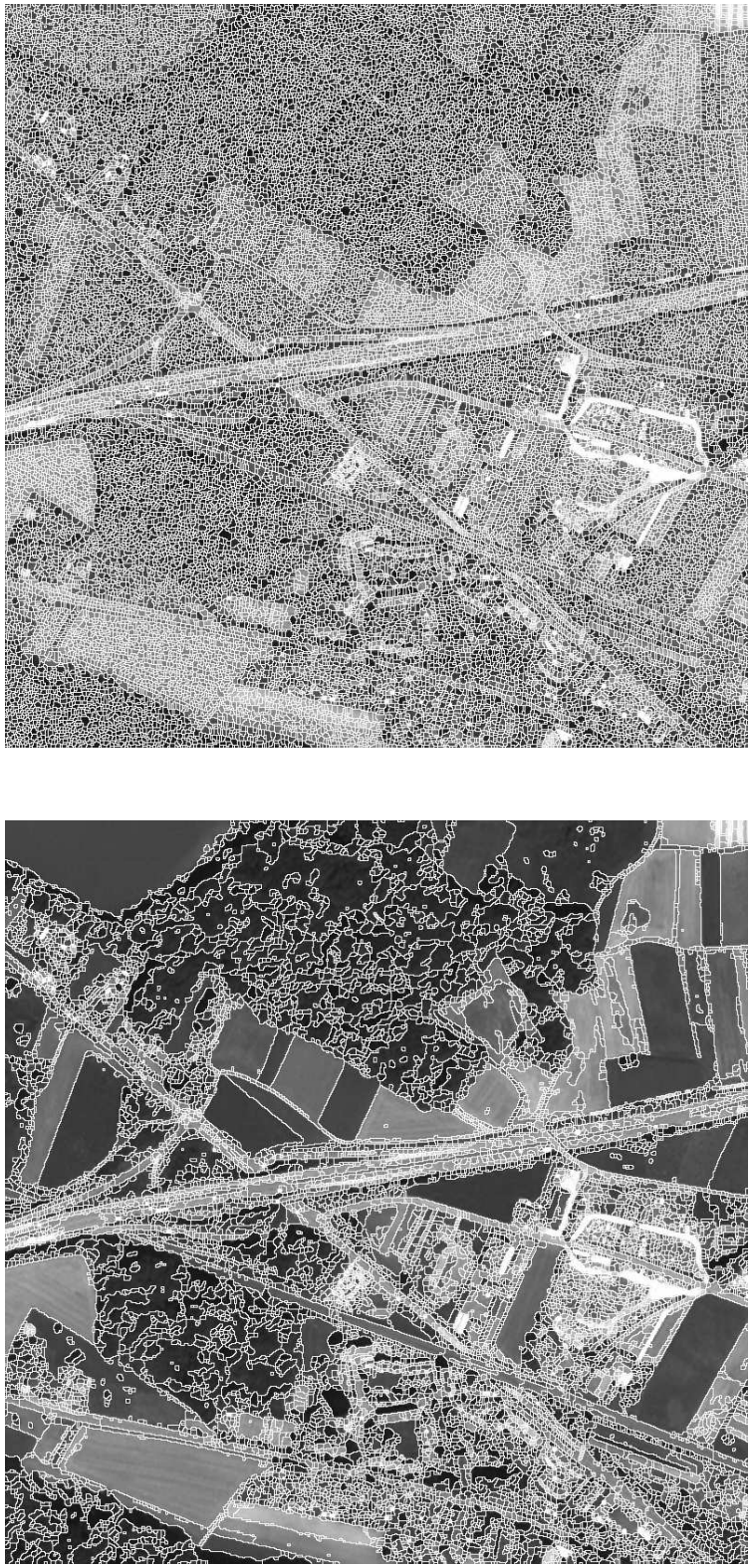

Figure 4.10: a) Watershed segmentation of the test image before thresholding. The total number of segments is 35867. b) Watershed segmentation of the test image after thresholding. The total number of segments is 12944.

the image used for this purpose was quite small, containing a total of only 776 regions after thresholded watershed segmentation. The number of collected merging examples were in all 3637 of which 1279 were merged while 2226 were not. The original segmentation as well as the resulting one after manual merging are shown in Fig. 4.11. Since the image used for collecting training samples is rather small, the training set can unfortunately not be considered complete, at least not for all but the smallest feature spaces. In other words, it does not cover all possible combinations of the features. However, the merging criterion we seek should be quite general and preferably not adapted specifically to the training data, and for this we do not need to know all local variations in the feature space.

Figure 4.11: (a) The initial test segmentation (776 segments), (b) the manually merged result (138 segments).

The Support Vector Machine proved to be usable, but far from perfect in finding a merging function. First, the machine was trained using only the spectral differences between the means of each pair of regions, the mean gradient of the border pixels, and the logarithm of the areas. In this configuration the merging function will behave the same throughout the color space, which means that it will not change the merging criterion due to the mean hue of the regions. Several gaussian kernels of different widths, σ, as well as a linear kernel were tested. The linear kernel produced almost as good results as the best of the gaussian kernels; around 85% of the pairs of regions were correctly merged for the best gaussian kernels and 83% for the linear one. The resulting segmentations, using linear and gaussian kernels, are shown in Fig. 4.12.

Adding more features to support the merging function increases the need of a good training set. In spite of the meager set in our disposal, adding the mean intensities and the variances to the patterns improved the results. In some cases the rather differently colored regions forming roads were correctly merged while avoiding to merge the similarly colored surrounding areas. However, the combinations of many features and few training examples also seemed to cause strange artifacts, like small regions in large homogeneous fields of grass. The overall impression was that using simpler patterns led to more consistent results.

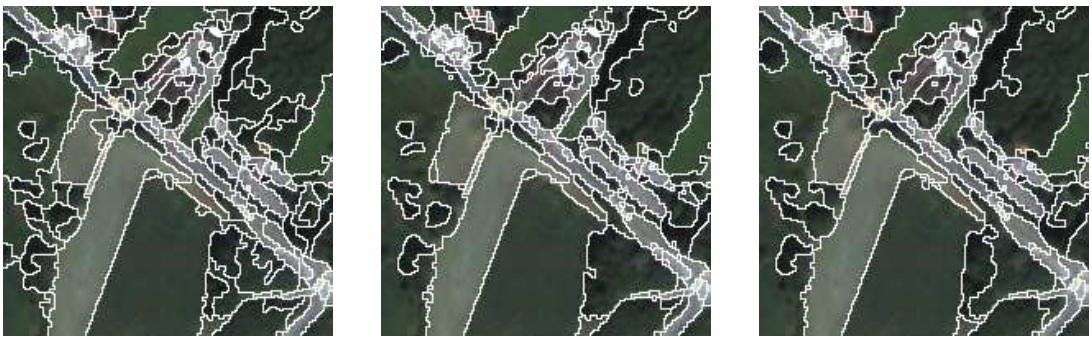

Figure 4.12: The test image merged using (a) a linear SVM with bias adjusted to produce 200 segments, (b) a gaussian kernel of width $\sigma = 0.2$ with bias adjusted to produce 200 segments. (c) the same gaussian kernel without bias adjusted, resulting in 156 segments

.

4.3.3 Classification

The segments of the test image was merged using a gaussian kernel of width $\sigma = 0.1$ and cost parameter $C = 1000$ resulting in 3637 regions showed in Fig. 4.13a, to be compared with the initial 35867 regions. These regions were classified using the region means and variances of each band and the best accuracy was achieved using a gaussian kernel of width $\sigma = 0.1$ and cost parameter $C = 1000$. Fig. 4.13b shows the resulting classification where the segment borders have been replaced by the label carried by the majority of its neighbors, for easy comparison with earlier results.

It is not evident that the results are either better or worse than per-pixel classification. The accuracy, measured in the same way as for per-pixel classification, is 12.3%. However, this percentage is only measured for the selected training points though, and it is difficult to conclude if the segmentation has improved classification of the remaining pixels, and those on or close to the borders in particular. Visually, the result is pleasing, with well defined borders and little noise.

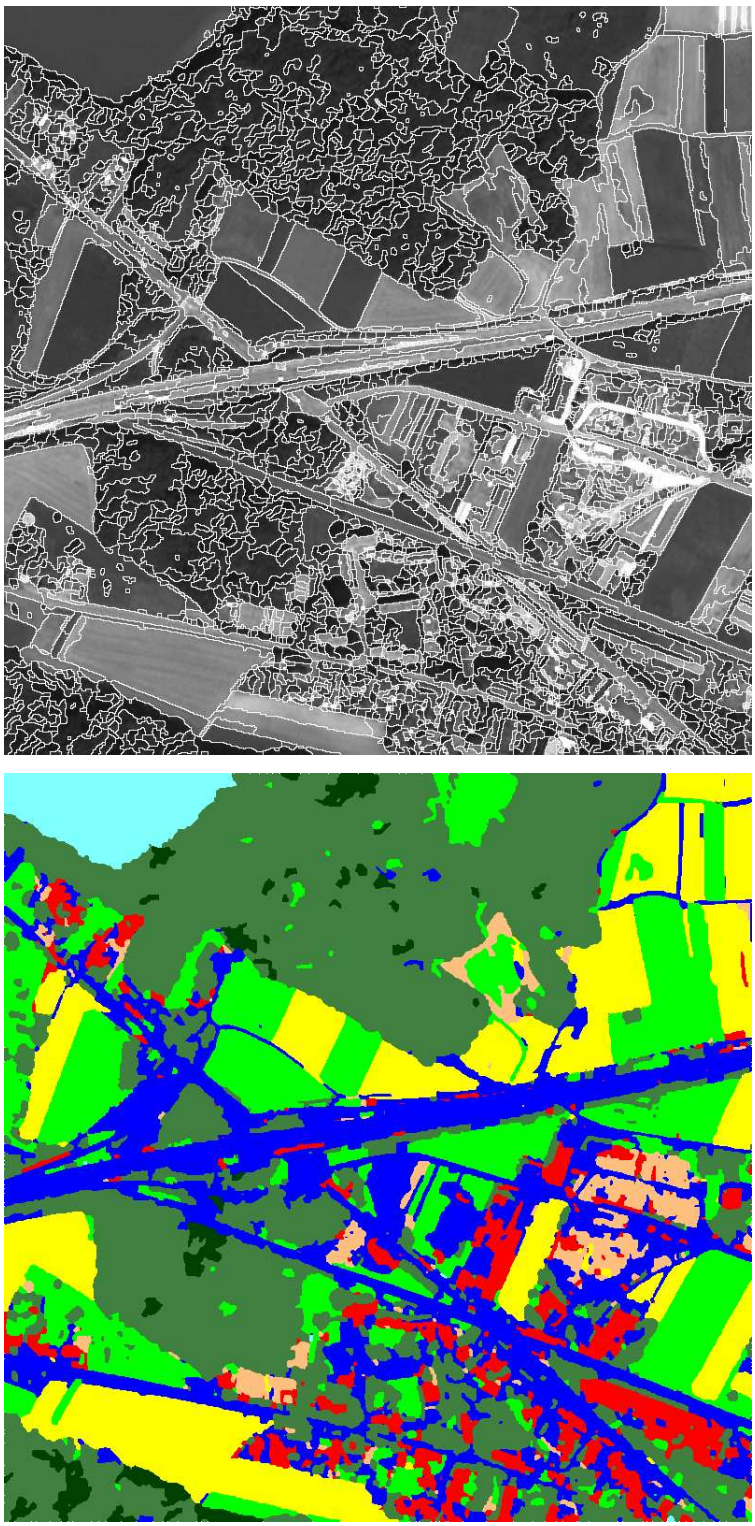

Figure 4.13: a) The final merged segmentation , now containing only 3637 segments. b) The classification result using the means and variances of the region as features.

.

5

Discussion

This chapter is dedicated to a summarization and discussion of the results. Future possibilities for segmented image analysis and understanding are also briefly discussed.

5.1 Classification of pixels and regions

Classifying high resolution is difficult due to the high level of detail. Most objects of the same class do not have a uniquely defined spectral signature, which is often the case when working with lower resolutions. The problem is made even more complicated by shadows and mixed pixels. It has been concluded that the information carried by a pixel is not enough to label it, even when the number of classes is low. The solution is to use some kind of contextual information in order to improve the classification.

Textural features improved the results somewhat, mainly by removing noise in larger homogeneous areas. In less homogeneous areas, the results were not significantly worsened, but definitely not improved. In many cases the running-time penalty of using the co-occurrence matrix is too large compared with the obtained improvements. Simple contextual averaging also reduced noise and smoothed the image. It was quite clear, however, that contextual knowledge is difficult for a statistical learning machine to learn. This is mainly because the training set must be extremely comprehensive to cover all possible pattern combinations. By simply glancing at a satellite image, one realizes that the number of combinations is close to infinity. For lower demands the method we used is still quite good, as it de-clutters the final classification, at the cost of some accuracy though.

There are certainly more advanced ways to carry out contextual classification but then a region based approach is logical, especially for high resolution images. For implementing such context we would probably avoid using a statistical learning machine because of the need of a training set. The problem with such an approach can be illustrated by an example: Let us assume we have a class *orchard* which can be described as,

"An orchard is many small broadleaf trees, quite regularly planted. the trees are sur-

rounded by grass or dirt"
The machine's knowledge of an orchard, based on training examples, may be something like,
"An orchard is an area where 60% is covered by broadleaf vegetation and there are 3 buildings in the neighbouring regions.
This example explains why a decision-tree or some other non-intelligent approach, incorporating hard-coded operator knowledge, would probably be more suitable. One could then pin-point exactly the wanted contextual knowledge in the machine.

In this thesis, we used only intensity statistics and simple shape information for classifying regions. Although this does not take full advantage of a segmented image, it produces classifications that are virtually noise-free, have clean borders, and whose classification accuracy is on par with the best pixel-based classification. These advantages can make segmentation worth the while, even for simpler classification tasks. The main conclusion of the region-based classification implemented in this thesis work is that segmentation is a step towards better classification but it is definitely not a solution to how to classify high resolution satellite images.

5.2 Support Vector Machine

We now turn to the main classification algorithm in this thesis and discuss the benefits of using it in the field of remote sensing. The factors that influence the performance of the Support Vector Machine are the dimensionality of the feature space, the training set, the kernel, and the cost parameter C.

The pattern dimension

A high dimensionality has proven to not significantly worsen the results but nevertheless one should take care in selecting features that provide uncorrelated information of the units to classify. A high dimensional feature space increase the risk of false learning and bad generalization capabilities. Still, it is not critical to find an optimal set of features since the Support Vector Machine tends to be rather forgiving. Thus, the operator will not need to spend lots of time optimizing the feature set, and dimension reducing techniques, such as principal component analysis are not vital.

The training set

The training set, on the other hand, is critical for the quality of the classifier, and the operator must find training samples that represent as many of the possible pattern combinations possible - a daunting task whenever complicated feature spaces are involved. Here to, the Support Vector Machine eases the burden a bit by not being overly sensitive to incomplete training sets. This is a consequence of the tolerance against high dimensional feature spaces. Still, any wanted variances in the classifier function must be represented in the training set and the SVM can never compensate for a lack thereof.

The kernel and the cost parameter

The kernel and the cost parameter C can be approached with different strategies depending on what one wishes to do with the resulting classifier. The linear or a polynomial kernel is easier to analyze and get an intuitive understanding for. However the classifier becomes much more limited than when using a gaussian kernel. Whether this is an advantage or not depends on the complexity and the quality of the training set. For a perfect training set, it would be advantageous to use a gaussian kernel of appropriate width to be able to better define the subspaces of the feature space. For an incomplete training set, generalizing performance may be preferred instead, and then a more restricted kernel can be used. In the latter case, the classifier will correct, to a certain extent, mistakes made by the operator when picking training examples.

However, a gaussian kernel can also be made less flexible by choosing a large width, so a natural recommendation is to stick to the gaussian kernel unless a simpler expression for the classifier function is needed for further analysis. The cost parameter and the training set is thus what remains for the operator to find, and it seems plausible that for a given application the cost parameter can be roughly pre-determined, further simplifying on the operator's part.

Benefits of using SVM in remote sensing

Using SVM:s in remote classification provides better possibilities of separating the classes compared to other similar methods. The benefits are not immediately visible when using few features, but for higher pattern dimensions it is a very powerful tool. An SVM handles such tasks both efficiently and with good results. The operator can keep the flexibility of the machine in check by means of the kernel and the cost parameter C so that the correct degree of fitting the classifier to training data can be found. A training set that is known to be of low quality can be somewhat compensated for by selecting a looser fit. If the training set is small, one can choose a less flexible kernel that will give better generalization performance. In many cases the operator could rely on preset values for both kernel and cost parameter, and the only task left would be to provide the best possible training set.

Positive and negative aspects of the SVM

+ Generalizes better than most other comparable classification algorithms (the curse of dimensionality is less evident).

+ The kernel allows complicated classifier functions to be constructed without severe running time penalties.

+ Classifying is fast.

+ Training has complexity $\mathcal{O}(\mathrm{n})$ using SMO.

+ The method is mathematically well defined, which gives less of the black-box behavior in for example neural networks.

- More difficult implementation than simpler methods such as maximum likelihood.
- Despite linear complexity, training time can be rather high for complex training sets.
- The input data needs to be normalized.

5.3 Segmentation

The segmentation of an image is an extremely complex task and the goal of this thesis was primarily to find a reasonable segmentation in order to try region-based segmentation. This subject turned out to be so interesting and important that its part was successively expanded. Still, we have barely touched this subject, which is currently a hot topic in the research world. As the main strategy we chose the split-and-merge segmenting scheme. This is a rather natural divide-and-conquer way of simplifying the algorithms.

Initial segmentation

For the splitting part of the algorithm we found that the watershed algorithm was better suited than the quadtree approach. Its main advantage is that the created segments correspond well to the objects in the image without post-processing which means that the resulting segments can be used directly for classification. As it turned out, the segment count could also be drastically reduced by a simple thresholding, making the watershed even more attractive. Still, this method is not perfect. Often the transition between objects are not sharp edges and in those cases the gradients are also less distinguished. Such borders can be missed and consequently areas of different hue or texture could be merged. For the quadtree, a smallest heterogeneity can be guaranteed, avoiding such problems, but instead it has no built-in edge detection, leaving the border definition to the merging step.

Merging

By using the heuristic described in Section 3.2 the merging problem is reduced into finding a binary function that determines whether or not to merge two regions, based on the features of both regions and the border between them. The approach with training a machine to perform the merging is appealing due to its conceptual simplicity. It works quite well too, without performing magic, as the results demonstrate. The problem lies in the difficulty of creating a reasonable training set because of the immense number of possible combinations of any two regions.

A compromise may be to use the machine learning approach to find a function that thereafter can be analyzed and corrected. With a support vector machine, this can be done by using a linear or polynomial kernel. The use of such a kernel also reduces the risk of over-fitting the function to the training data. This means it automatically avoids an overly complicated function. Although the results of using an intelligent merging machine are promising, it seems that a statistical learning machine is best used as a

starting point for some system of decision rules, especially if contextual information is to be included in the merging.

5.4 Possible improvements - Future work

It is clear that classification based on a statistical learning machine is powerful but limited. The heterogeneity of high resolution satellite images calls for alternative approaches, and during the work of this thesis some ideas for more advanced image understanding have emerged. A great reference, with many such methods thoroughly discussed, is chapter 8 of [6]. Herein, we will limit ourself to briefly discuss two topics; 1) improving object shapes and 2) using a data structure to organize the objects.

Active Contour Models

The resulting regions from the watershed segmentation, or most other segmenting methods for that matter, is of course very true to the original image. In many cases the shape of the objects can actually suffer from this since both the resolution and obstruction can cause artifacts on the object boundaries.

A way to improve the segmentation involves Active Contour Models, also called "snakes". Such a model fits a contour to an object by minimizing the contour energy functional. This functional is a weighted combination of the internal and external forces, where the internal forces emanate from the shape of the contour and the external come from the image and/or higher-level information.

Theoretically, we can use class-specific models to improve the shapes of already classified objects. For example, a snake that is optimised to enhance the shape of buildings would minimize its energy at the point where the contour consists of four straight edges that are orthogonal, at the same time as these edges lie on strong gradients and surround a region of rather homogeneous color. A sample of what could be possible is shown in [Figure].

Figure 5.1: A possible effect of applying specialized snakes on the watershed

Since the image is already segmented, the snakes would have a good starting point and the iterative process of finding the final shapes would be efficient. This is demonstrated by Park et al. [7] who apply ASMs on a watershed segmented image with good results. Nguyen et al. [8] describe a way to combine watershed segmentation with active contour models in order to include higher level knowledge directly in the watershed. Unfortunately, their so called watersnake rules out the possibility to use different snakes for differently labelled objects.

Hierarchical segmentation

Organizing the segmented objects in a data structure would facilitate contextual classification since the context is exactly what such a structure should contain. Furthermore, the data would be better organized and thus facilitate further post-processing or exporting to GIS:s. For the structuring of objects, it seems natural to borrow concepts from object oriented programming languages. Here, objects are of a class of objects and they can have, belong to, or know other objects. A house, for example, ***is*** a building, it can ***belong to*** a residential area, it can ***have*** a chimney and we can say it ***knows*** about its neighbouring buildings. For remote sensing purposes it would sometimes also be beneficial to rate each object with a level of significance, so that the user can select what level of objects not to show. The user could then retrieve just the displayable amount of information at each scale of visualization.

For the remote sensing operator, the construction of the data structure should of course be as automated as possible. How to do this is an open question, but a reasonable starting point would be to try and find natural delimiters, such as roads, rivers and distinct borders between regions of different land cover. In the regions formed by these delimiters the process would then be iterated in some fashion. Another possibility is provided by the watershed segmentation. By repeating the segmentation using different thresholds for the gradients, a kind of hierarchy can be built, by letting those segments that result from a high threshold be parents of those from a lower threshold segmentation. However, a quick evaluation showed that such a structure will not be satisfactory without further post-processing.

In the eCognition® software by Defiens Imaging, an object oriented structure is used for organizing the data. The construction of the structure requires substantial effort from the user, but the algorithms of the software greatly facilitates the task. Overall, the eCognition® approach makes sense in many ways and seems to be well adapted to the transition to higher resolution imagery.

6

Conclusions

The Support Vector Machine has been found to be a flexible and powerful statistical learning algorithm. In spite of this, it does not manage to reliably discriminate more than a few classes of land cover when using either per-pixel or region-based approaches. The problem lies in the heterogeneity of high resolution images - there is simply no way to be 100% sure what land cover a pixel or a small region represents by just using the information in that specific unit.

To take advantage of the high detail, the use of contextual information is required. However, our experiments have shown that correct classification from context cannot be efficiently integrated in a statistical classification process due to the need of an extensive amount of training. For pixel-based classification we saw some improvements using contextual averaging and textural features, but it would make more sense to manually create a set of rules for contextual classification than using a statistical classification method.

Segmentation of the image into regions did not significantly improve classification results. Instead, the real reason for segmentation is to facilitate contextual classification and to organize the objects of the image. The watershed method combined with a gradient threshold results in a nice segmentation that is directly usable. The Support Vector Machine can be used to further improve the segmentation by merging the watershed regions, but the difficulty of finding a representative training set makes other merging techniques preferable. Analogous to classification, contextual information would also be valuable for improving the quality of a merging algorithm.

A

Large versions of images

B

The test image

C

Code and descriptions

C.1 Per-Pixel classification

C.1.1 Description

C.1.2 Code

C.2 Region Growing

C.2.1 Description

C.2.2 Code

Bibliography

[1] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[2] B. Schlkopf and A. J. Smola, *Learning with Kernels*. The MIT Press, 2002.

[3] P. M. Mather, *Computer Processing of Remotely-Sensed Images*. John Wiley & Sons, 2 ed., 1999.

[4] K. S. Robert M. Haralick and I. Dinstein, *Textural features for image classification*, IEEE Trans. on System, Man, and Cybernetics **SMC-3**, 610–621 (1973).

[5] J. C. Platt, *Fast training of support vector machines using sequential minimal optimization*, in *Advances in Kernel Methods - Support Vector Learning*, C. J. C. B. B. Schlkopf and A. J. Smola, eds., pp. 185–208. MIT Press, Cambridge, MA, 1999.

[6] V. H. Milan Sonka and R. Boyle, *Image Processing, Analysis, and Machine Vision*. PWS Publishing, 2 ed., 1998.

[7] J. Park and J. M. Keller, *Snakes on the watershed*, IEEE Trans. Pattern Analysis and Machine Intelligence **23**, 1201–1205 (October, 2001).

[8] M. W. Hieu Tat Nguyen and R. van den Boomgaard, *Watersnakes*, IEEE Trans. Pattern Analysis and Machine Intelligence **25**, 330–342 (March, 2003).

[9] J. B. Campbell, *Introduction to Remote Sensing*. Taylor & Francis, 3 ed., 2002.