Coastline Detection in Satellite Images Using Machine Learning Techniques

Therese Kustvall Larsson, Lisa Silfversten
Coastline Detection in Satellite Images Using Machine Learning Techniques

Therese Kustvall Larsson
dic13tla@student.lu.se

Lisa Silfversten
kem12lsi@student.lu.se

May 25, 2018

Master’s thesis work carried out at Tactel AB.

Supervisors: Pierre Nugues, Pierre.Nugues@cs.lth.se
Jonas Neldeborn, jonas.neldeborn@gmail.com
Examiner: Jacek Malec, Jacek.Malec@cs.lth.se
Abstract

The growing number of affordable satellite services today has provided new opportunities for the use of space data. However, there are still numerous challenges in satellite image processing, and detecting diverse types of terrain can help in improving the data.

This thesis investigates three different techniques for segmenting land and water masses in satellite images to detect coastlines. The morphological active contours without edges algorithm is compared with a random forest and a U-Net convolutional neural network machine learning model.

We evaluate the different methods regarding execution time and accuracy, in addition to identifying difficulties and challenges with each technique. The resulting implementations of the three methods show that the convolutional neural network is the most preferable technique for this purpose, with a 95.0% classification accuracy and a faster execution time than the other methods.

Keywords: satellite data, segmentation, morphological ACWE, random forest, neural network, machine learning
Acknowledgements

We would like to thank our supervisor Pierre Nugues for his invaluable guidance and encouragement throughout the whole project. With his knowledge and expertise, he gave us the support to fulfill this Master’s thesis.

We would also like to express our sincere gratitude to our supervisor at Tactel, Jonas Neldeborn, and the entire maps team for their support, insights, and endless positivity.

And lastly, we would like to thank all Tactel employees for their open mind, solidarity, and for giving us the opportunity to develop as engineers.
Contents

1 Introduction 7
  1.1 Satellite Data ........................................... 7
  1.2 Map Data ................................................ 8
  1.3 Purpose .................................................. 9
  1.4 Research questions ....................................... 9
  1.5 Related Work .......................................... 9
  1.6 Contribution .......................................... 10

2 Theory 11
  2.1 Image Segmentation ....................................... 11
     2.1.1 Active Contours ....................................... 12
     2.1.2 Active Contours Without Edges ......................... 13
  2.2 Machine Learning ......................................... 15
     2.2.1 Classification ......................................... 16
     2.2.2 Decision Trees and Random Forests .................... 16
     2.2.3 Overfitting ........................................... 17
  2.3 Deep Learning and Neural Networks ....................... 18
     2.3.1 Activation ............................................ 18
     2.3.2 Backpropagation ...................................... 19
     2.3.3 Convolutional Neural Networks ......................... 20
  2.4 Evaluation ............................................... 20
     2.4.1 Jaccard Similarity .................................... 21
     2.4.2 Accuracy .............................................. 21
     2.4.3 Precision .............................................. 21
     2.4.4 Recall ................................................ 22
     2.4.5 F1-score ............................................. 22
     2.4.6 Support ............................................... 22
     2.4.7 Execution time ....................................... 22
## Approach

3.1 Methodology .................................................. 25
3.2 Utilities ......................................................... 26
3.3 Data Understanding ............................................ 27
   3.3.1 TIFF and GeoTIFF ........................................ 27
   3.3.2 OpenStreetMap Data ........................................ 28
3.4 Data Preparation .............................................. 28
3.5 Data Modeling .................................................. 29
   3.5.1 Morphological ACWE ...................................... 29
   3.5.2 Random Forest ............................................. 30
   3.5.3 Convolutional Neural Network ............................ 32
3.6 Model Evaluation ............................................... 34
   3.6.1 Morphological ACWE ...................................... 34
   3.6.2 Random Forest ............................................. 35
   3.6.3 Convolutional Neural Network ............................ 35

## Results

4.1 Output masks .................................................... 37
4.2 Morphological ACWE .......................................... 39
4.3 Random Forest ................................................ 40
4.4 Convolutional Neural Network ................................. 41

## Discussion

5.1 Satellite images and OSM data ............................... 43
5.2 Morphological ACWE .......................................... 44
5.3 Random Forest and Convolutional Neural Network ........ 45
5.4 Comparison of algorithms ..................................... 46
5.5 Improvements .................................................. 46

## Conclusions

6.1 ................................................................. 49

## Bibliography

Bibliography .......................................................... 51
Chapter 1
Introduction

1.1 Satellite Data

The technical evolution of modern time enables users to access high-resolution satellite imagery from various organizations free of cost. The growing number of affordable satellite services makes way for new uses of space data, and the increasing interest has attracted new businesses to invest in the area.

Copernicus, the earth observation mission developed by European Space Agency (ESA) ([ESA] 2018a), is the world’s largest earth observation mission, gathering data from up to 30 satellites. ESA’s newly developed family of satellites, called Sentinels, provide timely and accurate imagery (Figure 1.1) that can be used for many purposes within the areas of climate monitoring and research, digital applications (maps), and for surveillance.

Satellite images often need processing before being used, since they are captured in different conditions and with various levels of cloud coverage. Image segmentation is a technique that can be used for improving satellite data. Segmentation of different types of terrain can be used to detect nature changes for research purposes or in the development of map applications. By segmenting terrain in map images, different segments can be manipulated as desired, and unwanted segments (like clouds) can be removed. Important map segments, for example cities, can be replaced by segments with higher resolution. By segmenting land and water, changes in water levels can be observed, which provide data that can be valuable in climate research. In the development of map applications, there is also a need for segmentation. Image data for sea areas is not used, which requires a composition of different land segments and other data (e.g. blue fill color) for a complete world map.

There are various existing techniques for segmenting image data, and it can be troublesome to know what method to use for a certain purpose. This thesis investigates three different methods for segmenting land and sea to detect coastlines in satellite images, as a contribution to the knowledge of applicable methods for this and similar purposes.
1. Introduction

Figure 1.1: Example of satellite data from Sentinel satellites

1.2 Map Data

The organization OpenStreetMap (OSM) provides a free editable map of the world (OpenStreetMap [2018]). OSM is a collaborative project with over two million registered users, allowing anyone to add and remove data. OSM provides both map images and underlying map data, which both are available with free access. The company has recently gained popularity, and OSM map data is now used by big companies like Apple Inc., Snapchat and Flickr in their applications (Topf and Hormann [2016]).

The OSM data contains information about all ground objects, manually classified and marked by contributors. The data is structured by data primitives such as nodes, ways, relations, and tags, which together describe important properties and relations between objects. OSM also provides map data that includes descriptions of coastlines, land and water (Figure 1.2). These are represented as polylines and polygons, and accessed by so-called shapefiles, further describes in the chapter Approach.

In this Master’s thesis, OSM map data is used to obtain information about land and water geometry and coordinates.

Figure 1.2: Example of OSM map data. Water polygons (blue) and land polygons (green).
1.3 Purpose

This thesis investigates different techniques for segmenting land and water in satellite images to detect coastlines. Our focus is to evaluate the different methods regarding execution time and accuracy, in addition to identifying difficulties and challenges with each technique. By segmenting land and water areas with high accuracy, satellite data can be processed more easily and the quality of the data can be further improved. The research of this thesis contributes to the knowledge of which segmentation techniques to use for these kinds of problems.

1.4 Research questions

This report answers the following questions:

- How can coastlines in satellite images be detected? Investigate coastline detection with the three methods: morphological active contours without edges algorithm, random forest, and U-Net neural network machine learning techniques.

- Which method is the most efficient to detect coastlines? How does it compare in terms of accuracy and execution time with the other methods?

- Are there any challenges with the chosen methods?

1.5 Related Work

There are several existing research projects both for detecting coastlines and classifying terrain in satellite data. Many of them use edge detection algorithms or machine learning methods.

Liu and Jezek (2004) present a method based on Canny edge detection with an integration of locally adaptive thresholding techniques to determine water and land boundaries. The method provides a faster convergence towards the curve fitting process and the result is a more trustworthy local threshold for image segmentation. The article presents further image processing steps, such as labeling continuous regions into individual image objects, that results in a vector-based line coverage of the coastline.

Dandawate and Kinlekar (2013) compare the level set method (LSM) with modified Chan Vese algorithm (MCV) for the extraction of coastlines and rivers in multispectral satellite images. The methods are compared with respect to Dice coefficient, computation time and Hausdorff Distance, as well as being subjectively evaluated. The results found that the highest convergence speed is obtained with the LSM.

Peña et al. (2014) combine different machine learning techniques with object-based image analysis for classifying and monitoring summer crops. The machine learning methods used are decision trees, logistic regression, support vector machines (SVM), and multilayer perceptron neural network (MLP), both as single classifiers and hierarchical classifiers. As training data, both spectral and textural features were used, and the results
show that SVM and MLP prove best as single classifiers with a classification accuracy of 88%.

In previous research, it is common to compare different edge detection algorithms or machine learning methods separately. In this thesis, a morphological ACWE edge detection algorithm is compared with random forest and U-Net convolutional neural network machine learning methods together, to get a better understanding of what type of approach is best suited for this kind of segmentation. We altered the morphological ACWE algorithm in this project to work automatically, since no manual insertion of seeds is required, hence making it possible to run the algorithm on a larger dataset than in previous projects. In addition, convolutional neural networks and the U-Net architecture do not seem as explored as other methods when it comes to segmentation in satellite imagery.

1.6 Contribution

This thesis has been completed in full collaboration. Both students have participated in research, implementation and documentation of all parts of the study.
Chapter 2

Theory

2.1 Image Segmentation

Segmentation is a processing technique used for distinguishing groups of similar pixels in images. The technique is commonly used for object detection and recognition in areas such as machine vision (computer vision) and medical imaging. In computer vision, image segmentation is one of the oldest and most studied areas, with numerous known algorithms and methods (Szilárd, 2010). Some of the most commonly used segmentation techniques include thresholding, edge detection and region-based techniques and active contours (Anjna and Kaur, 2017; Baswaraj et al., 2012).

Thresholding is one of the simplest techniques, both in theory and implementation. The gray level intensity of all pixels is checked and compared to one or several threshold values. Pixels within different intervals in intensity are assigned different colors. The advantage of this method – apart from its simplicity – is that the algorithm is fast, even with bigger images. However, segmented objects may not be coherent, and the result may appear noisy with a lot of scattered pixels (Anjna and Kaur, 2017).

Image segmentation based on edge detection uses the derivative of an image function to discover discontinuities in intensity – which is often an edge. Thus, edge detection based segmentation works best if the image has sharp contrasts between edges and background, and little noise which can be mistaken as edges. To reduce noise, images are sometimes smoothed as a first processing step, even though the smoothing can sometimes cause loss of actual edge information as well. The computational cost for this method is quite low, making it suitable for larger images. However, besides being sensitive to noise, this method often produces discontinuous boundaries and detects a lot of internal object edges which can be undesirable in some cases when extracting object contours. Examples of common edge detection methods are the Sobel, Prewitt, Canny and Kirsch algorithms (Yuheng and Hao, 2017).

Region-based image segmentation separates pixels in different regions based on their
similarities. A pixel is grouped with neighboring pixels that share the same characteristics based on set rules, for example, gray level intensity \cite{Muthukrishnan2011}. The advantages of this approach are that the algorithm is quite simple, and yet it provides a satisfying segmentation result with only a few seeds (starting pixels from which the algorithm can grow from). One disadvantage is that the algorithm is computationally costly to execute. In addition, noisy images can lead to gaps in segmentation or over-segmentation \cite{Yuheng2017}.

\subsection{Active Contours}

Active contour models are frameworks used for detecting edges in images. They have become popular to use in different kinds of areas within computer vision such as object tracking, shape recognition, segmentation, and edge detection \cite{Marquez-Neila2014}.

Snakes

Kass et al. \cite{Kass1988} first introduced active contours in the form of snakes. As explained by Kass et al. \cite{Kass1988}, a snake is a contour defined by a function $f$ which, initiated in an image, will cling to lines and edges with the help of external constraint and image forces. Thus, a snake is initiated as a closed contour (circle, rectangle etc.), and then iteratively shrunk or expanded with a series of operations which is called contour evolution. The evolution is carried out by minimizing the energy function possessed by the contour, which is defined as the sum of three energy terms:

\begin{equation}
E_{\text{snake}} = E_{\text{internal}} + E_{\text{external}} + E_{\text{constraint}}
\end{equation}

$E_{\text{internal}}$ is defined by the smoothness and continuity of the contour, i.e. how the snake contour points are positioned compared to each other regarding distance and smoothness.

$E_{\text{external}}$ describes how well the snake contour matches the local image intensity data and is the energy term that attracts the snake to salient features in the image, as light or dark lines.

$E_{\text{constraint}}$ is the external force that is responsible for putting the initial contour near the object boundary, which can be done by a user through a user interface.

Without going into further details of the mathematics of the energy function, it can be said that these three terms are defined in such a way that the final position of the contour will yield the minimum energy of the snake.

This first implementation of snakes only converges to stop at an object boundary if the object is enclosed by and near the initial contour, requiring prior knowledge of the image, such as a user initializing the contour through a user interface. In addition, the snake cannot divide itself to detect several detached objects but is limited to stay as a single contour, nor can several initial snakes be combined. An additional limitation of snakes with a contour of parametric form is that the topology of the contour is not easily changed if the object boundary changes drastically, thus requiring reparameterization \cite{Baswaraj2012}.
2.1 Level Sets

The level set method, first introduced by Osher and Sethian (1988), is a way of representing a curve as a level set instead of a parametric function as the previous snakes. The idea is to represent a curve as the zero level set \( \phi(x, y, t = 0) \) in a higher dimensional function, called the level set function (Figure 2.1). This method makes it easy to track drastically changing object boundaries in both 2D and 3D images that the previous, basic snakes could not do (Jiang et al., 2012). However, since this method is dependent on image gradient, noisy images need to be smoothed, causing the boundaries to be smoothed as well (Chan and Vese, 2001).

![Figure 2.1: The level set function and its corresponding zero level set (Jiang and Chen, 2017).](image)

By defining a signed distance function (SDF) as \( \phi(x, y, t = 0) = d \), the distance \( d \) can be computed, which is the shortest distance between the point of \( x \) on the surface and the curve (Jiang et al., 2012). Examples of values of \( d \) are shown in Figure 2.2, where the sign of \( d \) is positive outside the contour and negative inside. These numbers constitutes the zero level set, and will be further discussed in the following sections.

![Figure 2.2: Zero level set (Yan and Kassim, 2006).](image)

2.1.2 Active Contours Without Edges

Taking snakes one step further, active contours without edges (ACWE) was introduced by Chan and Vese (2001). In previous models, the image gradient is used as a stopping
criterion for the snake’s boundary, thus detecting salient features regarding local image intensity. In his model, image gradient is not used as a stopping criterion but instead based on the Mumford-Shah segmentation technique (Mumford and Shah [1989]). This enables the snake contour to find objects boundaries that are not necessarily defined by gradient, for example very smooth or discontinuous boundaries. In this method, intensity averages of different segments (object and background) are calculated, and the contour evolves to keep similar intensity values inside and outside the contour, which works without the need to first smooth noisy images. In addition, the initial contour can be put anywhere in the image for the algorithm to work, and all internal edges will be found as well as object boundaries (Chan and Vese [2001]).

Morphological ACWE

This thesis will use morphological ACWE that is based on snakes and level sets to find edges in satellite images. It is an extension of ACWE, with similar results in image segmentation but with faster computation time. It is shown to perform well in pictures with poor contrast and provides a stable and fast contour evolution without the need of sophisticated numerical algorithms requiring a lot of computational power. In addition, the resulting contour is continuous and thus fits well the purpose of extracting coastlines. In this method, the SDF and zero level set are binary, leading to values of 0 outside the contour and 1 inside (Marquez-Neila et al., 2014). This matrix of 0’s and 1’s is used in the implementation of the morphological ACWE for this project, to represent the evolving curves. Other methods for edge detection were considered, but after studies and some testing on smaller sets of images, this method proved to be best one suited for the aim of this thesis.

Figure 2.3: Morphological ACWE algorithm contours and outputs masks. 10, 200 and 600 iterations.
To be able to evolve the snake contour, the algorithm needs a starting level set (which we will call seed) to be specified. A seed is represented by a circle with a specified radius. The morphological ACWE algorithm does not need the initial contour to be set near the object to work but will find all internal edges as well as boundaries if a level set surrounding the entire image is set at the start. However, only coastline borders and not internal edges on land are desired in this project. Therefore, it was decided that a seed with a small radius would be initialized only in water areas, which would stop the contour once it reaches land and thus avoid internal edges. If there are several closed water areas in one image, several seeds need to be initialized to enable the contour to evolve and enclose all areas. Figure 2.3 shows how the morphological ACWE contour implemented in this project evolves after 600 iterations, starting from one seed with a radius of 4 pixels.

2.2 Machine Learning

Machine learning (ML) is a field in computer science and a subfield of artificial intelligence (AI). In the 1950s, pioneer Alan Turing aroused the question if machines could learn and become artificially intelligent, and came to the conclusion that they could (Chollet, 2018). ML evolved from statistics and the study of pattern recognition, and soon became the most popular and successful subfield of AI (Witten and Hall, 2011).

A system that uses ML is trained rather than told explicitly how to act according to a set of rules. In classical programming, humans create a program that consists of a set of rules and inputs data to be processed according to these rules. In contrary to classical programming, ML systems takes input data and the expected outcome and learns a function that will optimally fit all input-output pairs. This mapping function is a way of representing the input data so that it will get closer to the output. By training ML algorithms on a relevant dataset, the output can be applied on other datasets by giving the program data as input. Chollet (2018) describes four broad machine learning categories:

**Supervised learning.** Supervised learning is the most common case. The agent is given a set of inputs and their desired outputs, often called labels, to derive a mapping function between input and output.

**Self-supervised learning.** Self-supervised learning is closely related to supervised learning. The difference is how labels are generated. In supervised learning, labels are in most case annotated by humans, but in self-supervised learning, labels are generated from input data.

**Unsupervised learning.** Unsupervised learning is often used for data visualization. The agent is given input without any desired output. Clustering is a well-known technique in unsupervised learning, where the input is clustered by similarity.

**Reinforcement learning.** Reinforcement learning is a researched area that so far mostly has practical success in games. The agent is given feedback based on its decisions and performance in a task. The model is improved by the given feedback.
2. Theory

This thesis will focus on supervised learning, as it will predict the probability of a certain class, by training on a large dataset containing samples and labels.

2.2.1 Classification

Classification is a learning problem in ML, where a dataset contains an input-output pair and the output is a finite variable, often called a label. Classification involves mapping a function between each input-output pair so that given an input, the model predicts its output (Russel and Norvig, 2010).

Another learning problem in ML is called regression. Regression is similar to classification, only the output contains a continuous value. Instead of predicting a class, the function predicts a value which could later be translated into a certain class (Russel and Norvig, 2010).

2.2.2 Decision Trees and Random Forests

A popular method in ML is called decision trees. Decision trees are commonly used both for classification and regression and are easy to visualize (Figure 2.4). Decision trees are constructed with nodes, where a question is asked about the input data in every node, e.g. if the input data is lower or bigger than a threshold value. Depending on the answers to these questions, the input propagates differently in the tree. This way, similar input ends up at the same end nodes (leaves) in the tree and can be assigned the same class in a classification problem, or a value in a regression problem (Chollet, 2018).

![Figure 2.4: Example of a decision tree.](image)

Random forest (RF) is an algorithm based on decision trees, that constructs a multitude of trees and outputs the most common result of the classification, or in case of regression, the mean value of the output. Random forests are applicable to many different ML tasks, which has made it one of the most popular algorithms. Regardless of which task, random forests often perform with high accuracy compared to other algorithms (Chollet, 2018).
To construct a decision tree, smaller subsets from the training data are randomly chosen for each tree. The best split of a node is calculated according to the input feature data of the subset. This step is iterated for child nodes, where the subsets increase in similarity for each split. Several decision trees are merged to create a RF (Figure 2.5), where each tree is trained with the same parameters, but on different subsets of training data for a more stable model. When predicting labels with a RF classifier, the input data is predicted in every decision tree and the final prediction is then averaged (regression) or based on majority voting (classification) (Wieland and Pittore 2014).

![Figure 2.5: The visualization of the random forest algorithm.](image)

### 2.2.3 Overfitting

A common problem that occurs in all types of learning is called **overfitting**. Overfitting is caused if too little data with relevant features are used when trying to predict on new data with more generalized features (Russel and Norvig 2010). The model is then well attuned to the training dataset but performs poorly on new, more generalized data. Common techniques for avoiding overfitting are cross-validation, using more training data and removing irrelevant features (Bilbao and Bilbao 2017). Overfitting becomes more likely as the number of attributes in input samples grows, and less likely with a smaller number of attributes. Figure 2.6 includes a comparison of an underfitted, overfitted, and correctly fitted model.

![Figure 2.6: The visualization of an underfitted, correctly fitted and overfitted model.](image)
To avoid overfitting, we use a large dataset containing a wide spread of different land terrains and water masses. If too similar terrains would be used for training, the model could fail to fit additional data or fail to predict reliably on new data.

### 2.3 Deep Learning and Neural Networks

Neural networks (NN) are models inspired by our understanding of the brain and a common method in deep learning, an approach in machine learning focusing on learning successive layers of data representation. A neural network is a collection of nodes, connected to each other by links. Each link has a weight associated with it and can transmit a signal, called activation, from node to node. A weight is a numeric value that determines the strength of the connection (Russel and Norvig 2010).

![Figure 2.7: Example of a neural network with an input layer, two hidden layers, and an output layer. Every input to the neural networks passes through the hidden layers.](image)

A neural network consists of a number of layers of increasingly meaningful data representations, which in turn are a collection of one or more nodes (Figure 2.7). Each layer is learned by exposure to examples, i.e. giving the model input-output pairs for training. Layers between the input and output layers are often referenced as hidden layers, as they are not visible as a network output. The deep in deep learning represents the idea of learning successive layers of data representation, rather than the reference to a deeper understanding than other machine learning techniques (Chollet 2018).

#### 2.3.1 Activation

To ensure that the network can represent a nonlinear function, an activation function is applied to each layer. Typically, an activation function is a logistic function, also known as the sigmoid function, calculated with the following formula:

\[ S(x) = \frac{1}{1 + e^x} \] (2.2)
Another commonly used activation function is the **rectified linear unit**, known as the ReLU function (Equation 2.3). The ReLU takes all negative values and turns them into zero \( f(x) = \max(0, x) \) (Chollet et al., 2015).

Figure 2.8 contains a visualization of the sigmoid and ReLU functions. By introducing non-linearity in the network with these functions, classification of data that is not linearly separable is possible.

![Figure 2.8: The visualization of equation 2.2 and 2.3](Chollet, 2018).

### 2.3.2 Backpropagation

Backpropagation is a central algorithm in neural networks. Together with a **loss function** and an **optimizer** it controls the network by propagating a signal from the output to previous layers (Figure 2.9) (Chollet, 2018).

The loss function is used to observe the performance of a model. It computes the error value between the predicted outcome and the correct target. The result is used as feedback to update the values of the weights in the direction to minimize the score of the loss function. The update of the weights is done by the optimizer, which uses the error values to calculate the gradient of the loss function (Chollet, 2018). The error of the hidden layers is calculated as follows:

\[
\text{HiddenError}_i = \sum (\text{OutputError}_i \cdot w_{ij}) \cdot F'(\text{HiddenOutput}_i)
\]  

(2.4)

Where \( \text{OutputError}_i \) and \( w_{ij} \) are respectively the error of next layer and the corresponding weights and the \( \text{HiddenOutput}_i \) is the activation function (Dolhansky, 2014). The variation of weights is calculated as a product between the hidden errors and the output of the input node:

\[
\Delta w_{ij} = \text{Output}_i \cdot \text{HiddenError}_j
\]  

(2.5)

When the variation of weights for all layers are calculated, they are accumulated and the weights are updated (Dolhansky, 2014). The update of weights is calculated as follows:

\[
w_{ij} = w_{ij} + (\Delta w_{ij} \cdot \text{LearningRate})
\]  

(2.6)
Where $w_{ij}$ are the current weights, $\Delta w_{ij}$ the accumulated weights and LearningRate is a constant that determine how fast the model will converge to a result (Dolhansky 2014).

![Figure 2.9: The visualization of backpropagation. Predictions are compared with its true labels and a loss function calculates an error score which is sent back to update the weights by an optimizer. When the error of the output layer is calculated, the error for each hidden layers is calculated going backwards, layer by layer.](image)

### 2.3.3 Convolutional Neural Networks

Convolutional neural networks (CNN) are models inspired by the animal visual cortex, that consists of a number of convolutional layers. These layers create a kernel, also known as filters, that is convolved with the input layer, a $M \times M \times R$ image, to detect features in the image. The kernel has a small receptive field that is slid across the matrix, calculating the dot product for each dimension of the input matrix. Different filters are learned to detect different features. If the result of the convolution is a high number, a feature has likely been detected in that part of the image. Compared to other machine learning methods, convolutional neural networks can thus learn and detect features anywhere in an image. The number of pixels that the kernel is moved for each calculation is defined by a numeric value called stride. The size of the image data shrinks with every convolutional operation (see Figure 2.10). Padding can be used to obtain an output with the same size as the input (Chollet 2018).

In this Master’s thesis, a CNN with an architecture called U-Net is used, further described in the chapter Approach.

### 2.4 Evaluation

This section describes all metrics that have been used to evaluate and compare the different segmentation methods for this project.
2.4 Evaluation

Figure 2.10: Example of a convolutional operation with an image size of 5×5 pixels and a kernel size of 3×3 pixels. The stride is set to one, resulting in an output decreased to the size of the kernel.

2.4.1 Jaccard Similarity

Jaccard similarity, also called intersection over union (IoU), is a way of calculating similarity between two datasets. The Jaccard similarity is calculated as follows:

\[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.7} \]

where A and B are the two compared datasets. A value close to 1 means that A and B are very similar, and a value of 0 means that they do not share any samples [Marquez-Neila et al. 2014]. In binary and multiclass classification, the Jaccard score is the same as the accuracy score (accuracy is explained in the next section) [Scikit-learn 2017b].

In this project, Jaccard score is calculated for each image used in the programs and saved in a list. When all images are segmented, mean and median Jaccard scores are calculated from this list, and used in the evaluation of the three methods.

2.4.2 Accuracy

Accuracy is a metric used to measure how good a classifier’s model is. It is simply interpreted as the ratio of correctly classified samples to the total number of samples. If 100 pixels were to be classified, and 80 of them are classified correctly, the accuracy of the model is therefore 80%.

In binary and multiclass classification, accuracy score is the same as Jaccard score [Scikit-learn 2017b]. If the dataset has an uneven distribution of different classes, the F1-score is a better performance metric to use (F1-score is explained further below) [Joshi 2016].

2.4.3 Precision

Precision is a metric used to evaluate classifier output quality. It represents the classifier’s ability to avoid labeling negative samples as positive, i.e. the percentage of all positively classified samples that are actually positive. Precision is calculated with the following formula:

\[ \text{Precision} = \frac{tp}{tp + fp} \tag{2.8} \]
where $tp$ is the number of true positives and $fp$ is the number of false positives. A good precision gives a value close to 1, and a bad precision close to 0 (Scikit-learn, 2017e).

### 2.4.4 Recall

Similar to precision, recall is a metric to estimate the classifier’s quality. It represents its ability to find all positive samples, i.e. how big part of all positive labels that are classified as positive. Recall is calculated with the following formula:

$$Recall = \frac{tp}{tp + fn} \quad (2.9)$$

where $tp$ is the number of true positives and $fn$ is the number of false negatives. As with precision, a good recall gives a value close to 1, and a bad recall close to 0 (Scikit-learn, 2017f).

### 2.4.5 F1-score

The F1-score is a weighted harmonic mean of the precision and recall metrics, with a good value close to 1 and a bad value close to 0 (Scikit-learn, 2017c). This metric shows the classifier’s accuracy, but takes both false positives and false negatives into account which makes it a bit different from the accuracy metric. It is recommended to use this score as performance metric if the distribution of the different classes are uneven (Joshi, 2016). F1-score is calculated with the following formula:

$$F1\text{-score} = \frac{2 \cdot (\text{precision} \cdot \text{recall})}{\text{precision} + \text{recall}} \quad (2.10)$$

### 2.4.6 Support

The support metric is the number of labeled samples of each class that occurs in a given dataset (Scikit-learn, 2017d). In this report, the support metric gives a value of how many instances of each class that occurs in the ground truth of the entire test dataset.

### 2.4.7 Execution time

Execution time is used as an evaluation metric for the morphological ACWE algorithm. It is measured as the time it takes to execute the program, which includes all of the stages from reading OSM shapefiles, creating truth masks, extracting seeds and evolving contours. All these steps are required in our implementation of the algorithm when segmenting new images, which means execution time serves as a measure of how fast this particular algorithm is. Evaluation time is not included since this step is not required when segmenting new images, only for evaluation of the result.

For the classifiers, time is measured a bit differently, since not only execution time of the programs are relevant. When using machine learning techniques, training and testing time are more relevant metrics to measure since other stages included in execution time,
2.4 Evaluation

e.g. preparation of datasets and evaluation, has nothing to do with the used techniques and thus are not relevant to compare. Training and testing time in this project are defined as:

**Training time** specifies how long it takes to train a machine learning model with an already finished training dataset. The time required to create the training dataset is not included here. For the RF classifier, training time includes the process in which features are extracted from the training data. This step is not used by CNN classifier.

**Testing time** specifies how long it takes to predict the whole testing dataset without evaluation, with an already finished test set. For the RF classifier, this time also includes feature extraction for the testing dataset, since this stage is required every time new images are predicted. The CNN testing procedure does not have this step.

Once the dataset is created, only training of the model and testing are the steps required when segmenting images, and thus training and testing time are compared with execution time for the morphological ACWE in this project.
3.1 Methodology

In this Master’s thesis, we decided to use a methodology called CRISP-DM, a well proven technique for data mining projects. CRISP-DM (Cross Industry Standard Process for Data Mining) is a model which offers a framework for the working process in data mining projects. The purpose of this methodology is to provide a standard approach that costs less, is more reliable, faster, more manageable, and easier to repeat. Regardless of which industry the project is carried out in, and what technology is used, the CRISP-DM can be utilized. In general, the approach is built as an iterative cycle with multiple stages that can be traversed as desired depending on the outcome from the previous phase. The following six stages are part of the CRISP-DM cycle [Wirth and Hipp 2000]:

**Business Understanding.** The initial phase, where the objective is to understand the requirements and project goals. A data mining problem is defined from these requirements and included in a project plan.

**Data Understanding.** An initial dataset is collected and explored. The purpose of this phase is to get familiar with the data in terms of what information is included, how to use it, and of what quality it is etc.

**Data Preparation.** In this phase, the final datasets to be used are constructed. Any transformation of the data to make it usable for the project is done here.

**Modeling.** The choice of modeling technique is made in this phase. Parameters used in the model are calibrated and the technique is implemented.
### 3. Approach

**Evaluation.** Evaluation of one or several chosen modeling techniques and the achieved results before moving on to the most suitable model.

**Deployment.** In this phase, the end result is generated to be used by end users, for example a report, a system or process.

### 3.2 Utilities

To carry out this Master’s thesis, we have chosen some suited utilities for our tasks:

**Hardware.** As hardware, we used a Macbook Pro laptop with 16 GB RAM and a 2.8 GHz quad core Intel i7 processor on which the algorithms were executed.

**QGIS.** QGIS is a free software for geographic information systems. It is an open-source platform for viewing, editing and analyzing geospatial data. QGIS has support for different layers, such as vector and raster layers (Wikipedia, 2018b).

We used this tool to vectorize output files from our program and compare them with OSM shapefiles. In addition, we used QGIS to generate simple train and test data to test our programs on small datasets.

**GDAL.** The geospatial data abstraction library (GDAL) is an open-source library for reading and writing geospatial data. GDAL supports both vector and raster formats and can be built with other utilities for data translation (GDAL contributors, 2018). We mainly used GDAL for reading and writing TIFF-files and rasterizing vector files (OSM shapefiles). GDAL also has support for reading and writing georeferencing information saved in a TIFF-file, which we used to save and set projections and coordinate system to map our data with OSM shapefiles of land and water polygons.

**Scikit-learn.** Scikit-learn is a machine learning library used for Python programming language. The library provides several different machine learning techniques such as classification, regression and clustering (Scikit-learn, 2017a). We used scikit-learn to create a RF classifier. We also used scikit-learn’s classification report to evaluate our models.

**Keras.** Keras is a high-level API for neural networks, written in Python. We used Keras on top of TensorFlow, a math library often used for machine learning tasks. Keras contains a wide range of commonly used building blocks for neural networks, such as different layers, activations, and optimizers (Chollet et al., 2015).

Keras and TensorFlow can be run on GPU instead of CPU with large datasets for faster execution. We used the CPU version of TensorFlow with Keras to build a CNN, using the U-Net architecture described in the next subsection.
3.3 Data Understanding

**U-Net.** U-Net is an architecture for neural networks used for image segmentation. It was originally designed as a CNN for biomedical image segmentation, focusing on using annotated data sampled more efficiently. U-Net also works for regular 2D image segmentation and has proven to be a fast network with high performance even on small datasets (Ronneberger and Thomas, 2015). The architecture of U-Net will be further explained in the section *Data Modeling*.

### 3.3 Data Understanding

We used satellite imagery collected from the Sentinel-2 satellites to create a dataset. Sentinel satellites provide high-resolution optical images for land services (ESA, 2018b). The data is constructed by three layer RGB data with a resolution of 512 x 512 pixels (Figure 3.1). The imagery consists of 13 different zoom levels, where each level contains images of the entire world, excluding oceans. Zoom level 13 and 1 contain respectively the most and least detailed images. We chose to use zoom level 9 in this project, since it shows a suitable number of details.

![Figure 3.1: Visualization of the RGB layers.](image)

#### 3.3.1 TIFF and GeoTIFF

The satellite images have a format called Tagged Image File Format (TIF/TIFF), which is a file format for storing raster images (Wikipedia, 2018c). GeoTIFF is used to store georeferencing information in all TIFF-files (Wikipedia, 2017). We used information such as projection and coordinate system to map our data with OSM shapefiles to extract land and water polygons, used as ground truth.
3.3.2 OpenStreetMap Data

As mentioned earlier, OpenStreetMap (OSM) provides free editable map data of the entire world. The OSM data has a topological structure with four types of data primitives (Wikipedia, 2018a):

- **Nodes** are points with a geographic position that represent map features without size, as mountain peaks. Nodes are described with latitude and longitude coordinates.

- **Ways** are ordered lists of nodes that form polylines or polygons. Ways are used to describe streets, rivers, or different areas of terrain.

- **Relations** are ordered lists of nodes, ways, and relations, which are used to describe the relation of existing nodes and ways, e.g. turn restriction on roads.

- **Tags** are key-value pairs that contain data about all map objects, for example type, name, or physical properties.

Coastlines are represented as polylines with the tag `natural=coastline`. OSM provides files containing all coastline polylines of the world (Topf and Hormann, 2016). These files are called shapefiles, which are basically a set of files in different formats that contain information about geodata, such as geometry, projection, and coordinates. A shapefile is a common standard for representing geospatial vector data. In a shapefile, the geometry is represented as points, polylines, and polygons. Files included in a shapefile vary, and usually include some files that are optional. These are some of the most common formats:

- `.shp` - shape format
- `.shx` - shape index format
- `.dbf` - attribute format
- `.prj` - projection format; describes projection and coordinate system.

OSM provides shapefiles derived from coastline polylines, with land and water polygons. We used these files in this project to create truth masks. Figure 3.2 shows a satellite image with its associated piece of the land and water polygons shapefiles.

3.4 Data Preparation

We used the same dataset containing 250 pictures in all three algorithms. We handpicked the images from the large Sentinel-2 dataset by choosing images containing coastline spread all over the world. We included both clear images without clouds and cloud/ice spotted images, as well as some images with only land or water to get a dataset with high diversity.

When comparing each satellite image to the OSM ground truth, all images are required to have a coordinate system, making it possible to fit the image to the correct place in the large shapefile. All the images from zoom level 13 were available with a coordinate
system, but images from other zoom levels were not. To get the coordinates and projection for the images from zoom level 9, we created a Python script that extracts coordinates from level 13 images and fits with images from level 9 (one level 9 image contains 256 level 13 images).

The 250 images were divided in two sets for training and testing (validation). To obtain an equal spread of different types of terrain in both sets, we created a Python script that picks images randomly with a split of 80/20 percent for training and testing. Each pixel is considered a sample, which gave us a training/test dataset of 52,428,800 vs. 13,107,200 samples.

### 3.5 Data Modeling

We used a RF classifier and a CNN as supervised machine learning algorithms to compare, as well as a third method, morphological ACWE. All the models are described in detail below.

#### 3.5.1 Morphological ACWE

The algorithm was executed on one test image at a time. We used the GDAL library to extract and save the coordinates from the image, which we then used to cut out the relevant piece of the OSM shapefile and rasterize it to obtain a (512, 512) matrix truth mask, containing the values 0 for land and 1 for water. Seeds for the algorithm were then extracted, which is explained in the next subsection.

#### Extracting seeds from OSM

To be able to find all closed water areas in the images, one seed per water area is required at least. To avoid having to set the seeds manually, we used the OSM shapefile to create a list of possible seeds in water areas. We started by dividing the image in a grid with squares the size of 16×16 pixels (Figure 3.3), and then searching the label of all pixels in each square in the OSM shapefile. If all pixels in a square were labeled as water, we
extracted the pixel in the middle of the square and appended it to the seed list. As a result, all squares that are entirely water contributed with one seed. One seed is represented by pixel coordinates \((x, y)\).

![Figure 3.3: Each image is divided in a 16×16 pixels grid.](image)

**Algorithm iteration**

We converted the original image to gray scale where the resulting data was a \((512, 512)\) matrix containing gray scale intensity values for all pixels in the image. We then retrieved the first seed in the seed list and used as the initial level set for the algorithm. As a base framework for the algorithm implementation, we used the code available in the Github repo by Neila [2015].

When evolving the contour, we used the averaged intensity values of pixels inside and outside the contour, \(c_1\) and \(c_2\), as stopping criterion instead of specifying a fixed number of iterations. When \(c_1\) and \(c_2\) stopped changing, a local minimum had been found to the energy function and the contour had evolved to its final position. The resulting level set matrix then contained the contour of the first grown seed, with values 1 as water (inside contour) and 0 for land (outside contour).

After evolving the first contour, we retrieved the next seed in the list and checked its coordinates \((x, y)\) against the level set matrix. If the value of the level set matrix in coordinates \((x, y)\) was already equal to 1, we discarded the seed and retrieved the next. This way, we made sure that all possible seeds were iterated through, but only if they were not already within the contour. When all seeds had been iterated through, we saved the final level set contour as a \((512, 512)\) matrix to be compared with the truth mask.

**3.5.2 Random Forest**

We used the scikit-learn library for Python to create a RF classifier. The model takes two parameters: \texttt{n_estimators}, which is the total number of trees in the model and \texttt{n_jobs}, the total number of jobs to run in parallel for both training and predicting.
Training and predicting

To train the model, we loaded one image from the training dataset at a time, using the GDAL library to read the RGB layers and stacking them in a (512, 512, 3) shaped matrix. We then extracted RGB intensity values for each pixel, and all directly neighboring pixels (9 pixels in total) from this matrix and used as features, resulting in a feature vector of 27 elements for each pixel. If a pixel was located in a corner or on the edge of the image, features for missing neighbor RGB values were set to 0. For each image, we also saved its coordinates to extract the corresponding data from the OSM shapefiles containing land and water polygons. We rasterized the OSM data and saved it as a (512, 512, 1) shaped matrix, were each value in the matrix represents a class, 0 or 1 (land or water). For each image, we concatenated the matrices with all prior loaded images, which eventually lead to two matrices: one containing the RGB values for all training pixels and its neighbors, and one containing all labels.

When all images in the training dataset were loaded, we trained the model using the two matrices as parameters.

![Image of RGB matrix and OSM matrix](image)

**Figure 3.4:** Example of an RGB image with 6×6 pixels in a (6, 6, 3) shaped matrix and their associated class, 0 or 1, in a matrix with shape (6, 6, 1). The pixel value for each layer in the RGB matrix at a specific coordinate, and all neighboring coordinates are matched with the corresponding value in the matrix containing labels at the original coordinates. For example, coordinate (1, 1) has value 10 for blue intensity and some other value for green and red. The resulting feature vector for coordinate (1, 1) contains RGB values for coordinates (1, 1), (1, 2), and (2, 1), and is padded with 0’s since the pixel is located in a corner. The corresponding label of 1 or 0 will be extracted from the matrix containing labels at coordinate (1, 1).

After training the model, we fed the test dataset to it. For each image in the test dataset, we used the GDAL library to read the RGB layers and stacking them, using the same approach as for the training data. We saved RGB values for each pixel and its neighbors in a common matrix. The model predicted the class for every pixel, based on the 27 feature...
3. Approach

values for that pixel, and we saved the result in an array. For each image, we also saved its coordinates to extract its true labels from the OSM shapefile. We then saved all true labels in a matrix, which in the end was used to calculate the Jaccard similarity score.

3.5.3 Convolutional Neural Network

We constructed the neural network with a high-level API called Keras. We used an architecture called U-Net, a model which has proven great performance for image segmentation. The U-Net model is built by a number of convolutional and pooling layers, stacked on top of each other as input. Each convolutional layer uses the activation function called ReLU. The output layer is a convolutional layer with sigmoid activation function, which returns a value between 0 and 1, corresponding the prediction of a certain class.

![Figure 3.5: Visualization of the U-Net architecture. Each grey box is a multichannel feature map, and the grey arrows pointed to the right is a convolutional operation, followed by a nonlinear activation. The downwards arrows are maxpooling operations, decreasing the input size for each operation. The white arrows and the upwards grey arrows are respectively concatenations and up-convolutions.](image)

**Convolutional layers**

We used code for the U-Net architecture from the blog by Åmdal Sævik [2018]. This model is built with two types of convolutional layers, convolution and convolution transpose. Every convolution layer has a kernel size of $3 \times 3$, and every other convolution layer increases the number of feature channels by 8 at a time, up to 128.
When the number of output filters has reached 128, the convolution transpose layers are added. They have a kernel size of $2 \times 2$ and decreases the number of output filters by 8 at a time. The convolution transpose layers are concatenated with the corresponding convolution layer, which is the one containing the same number of outputs. A visual representation of the U-Net architecture can be seen in Figure 3.5.

### Pooling layers

The model is built with a number of maxpooling layers. Maxpooling is a non-linear down sampling method that uses the maximum value of each cluster from the prior layer (Figure 3.6). In the U-Net model, each maxpooling layer have a size of (2, 2) corresponding a $2 \times 2$ filter with a stride of 2.

![Figure 3.6: Example of maxpooling with a 2×2 filter and stride 2.](image)

### Training and predicting

To train the model, we first loaded every image from the training dataset in an array. We used the GDAL library to read the images, extracting the RGB layers and stacking them in a $(512, 512, 3)$ shaped matrix. We also used the GDAL library to save coordinates and projection from the images, which we used to extract the corresponding piece of the OSM shapefile. Having the correct piece of the OSM shapefile, we rasterized it and obtained a $(512, 512, 1)$ shaped matrix containing the true labels for all pixels in the image.

We trained the model in batches of 8 with 30 epochs, which means that the model trains on 8 samples at a time and iterates over the entire dataset 30 times. The optimizer that we chose for this model has a learning rate of 0.001, and the loss function is for binary classification. The model also uses earlystopping, which is a function that stops the iteration if the score of the loss function is not improved. For each iteration, the model is saved and overwritten if the loss function is improved, resulting in obtaining the model with the lowest loss function.

After the model was trained, we used the test dataset for prediction. When reading the dataset, we used the same approach as for training, resulting in two arrays, one containing the RGB values and one containing true labels. When predicting, the model returns a matrix containing the prediction values of the class land. We formatted these predictions
into class values of 0 or 1, corresponding to land and water, making it possible to calculate the Jaccard similarity between the predictions and their true labels.

3.6 Model Evaluation

This section describes how we evaluated all models and how we chose parameters values as a result of the evaluation.

3.6.1 Morphological ACWE

As mentioned earlier, we used Jaccard similarity score and execution time when evaluating the models. For this algorithm, we also monitored the number of iterations and seeds required for one image. After running the algorithm, we compared the contour output with the truth mask obtained from the OSM shapefile. We saved the Jaccard score, execution time, number of iterations, and number of seeds in a list together with results from all other images. After running all images, we calculated mean and median values from these lists.

Algorithm parameters

The parameters required to run the algorithm are $\lambda_1$, $\lambda_2$ and $\mu$. $\lambda_1$ and $\lambda_2$ are constants specifying the relative importance of the inside pixels against the outside pixels. Since they are equally important, we set both constants to 1. $\mu$ is the smoothing strength, i.e. the number of repetitions of the smoothing step to carry out in each iteration. We tried different values of smoothing values on smaller datasets, and concluded that a smoothing strength of 0 gave the best results, hence used in this project.

Seed radius and square size

We agreed upon the size of the squares used when extracting seeds after a lot of testing on smaller datasets. When using small squares, there is a greater chance of finding small water areas, however the seed list will be longer and extend the execution time when being iterated through. A larger square size results in the smallest water areas not being found, but also results in a shorter seed list which is faster to iterate through. The tests concluded that a square size of 4 - 8 pixels gave the best results in accuracy, but for reasons accounted for in the discussion section, we increased the square size to 16 pixels.

In addition, we tested different radii of the seeds. Using a large radius caused the contour to grow over land areas if being initiated in a small water area, extending the execution time significantly and impairing the accuracy result. The tests concluded that a pixel size bigger than 1 but relatively small gave the best results, hence we chose a size of 4 pixels.

Gaussian blur

Even though a good result with the morphological ACWE does not depend on the image being smoothed beforehand, we tried applying different levels of Gaussian filter blur on
all images since we noticed worse performance on noisy images and images spotted with clouds or ice. However, it was evident that blurring the image was only obscuring the edges, thus causing the contour to evolve over land and impairing result accuracies. Thus, we executed the final program without any blur.

3.6.2 Random Forest

As mentioned earlier, in addition to Jaccard similarity score, we used training and testing time when evaluating the performance of the RF classifier. Both of these must be considered when being compared to the other algorithms, even if the training part only has to be executed once.

To evaluate the classifier internally, we also used the scikit-learn classification report. The classification report includes precision, recall, f1-score and support. As these measurements are evaluated for both land and water, it is useful for evaluating how accurately the classifier predicts on both classes.

Algorithm parameters

We primarily set the parameters required to run the RF classifier to default values. The parameter \texttt{n\_jobs} was set to 4, which factors the number of processes to run in parallel to be four, both for fit and predict. This number should be the same as the number of cores of the CPU. We set the parameter \texttt{n\_estimators} to the default value of 10.

3.6.3 Convolutional Neural Network

We evaluated the CNN was in the same way as the RF classifier, using scikit-learn’s classification report for local evaluation and Jaccard similarity score and training and testing time for comparison with the other two algorithms.

When we experimented with the CNN, it showed that the model also performed with high accuracy using a smaller training dataset, which both decreased the total execution time and made it possible to use the algorithm on a smaller dataset without impairing accuracy very much.

Algorithm parameters

We chose to use the U-Net model when creating the network, as it has proven great performance in image segmentation. Therefore, we did not experiment with the algorithm parameters, as this thesis focuses on evaluating the performance on different algorithms rather than experimenting with different models of neural networks.
3. Approach
In this chapter, we present the results of the algorithms. As mentioned earlier, execution time and Jaccard similarity are the metrics we chose to focus on when comparing the algorithms. Some individual metrics are also presented, but these have mainly been used to evaluate the plausibility of the algorithms.

### 4.1 Output masks

To visualize the result from the algorithms, we decided to implement a method that creates an output for each image segmentation. The output is saved as a black and white TIFF-file, making it possible for us to see the segmentation result for every image and algorithm. In this section, we present the result for all algorithms by three chosen satellite images with different quality. In Figure 4.1, the output masks are presented together with associated satellite images and truth masks extracted from OSM shapefiles. All output masks will be presented separately for each algorithm below.

As mentioned earlier, execution time or training and testing time will also be used as metrics to evaluate the three methods. These results are presented separately for each algorithm.
4. Results

(a) Satellite data

(b) OSM shapefile

(c) Morphological ACWE output

(d) Random forest output

(e) Neural network output

Figure 4.1: Comparison of satellite data, OSM ground truth and algorithm outputs.
4.2 Morphological ACWE

Tables 4.1 and 4.2 show the results from the morphological ACWE. A mean Jaccard score of 86.9% is quite good but leaves room for improvement. The median Jaccard score is significantly higher, over 4%, resulting in a median Jaccard score of 89.5%. These results show that at least half of images in the dataset achieved a Jaccard score of 89.5% minimum, which is somewhat a more acceptable result.

Morphological ACWE is supposedly a fast algorithm compared to other edge detection algorithms, but with a total execution time of approximately 5391 sec ≈ 1.5 h, this algorithm is tedious to use compared to the machine learning techniques presented below.

**Table 4.1:** Jaccard similarity and execution time results.

<table>
<thead>
<tr>
<th></th>
<th>Jaccard Similarity</th>
<th>Execution Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.869</td>
<td>Mean 123</td>
</tr>
<tr>
<td>Median</td>
<td>0.895</td>
<td>Total 5391</td>
</tr>
</tbody>
</table>

The average number of seeds used for each image is approximately 44, which means the algorithm has encountered some poor images with a lot of disturbance from e.g. clouds or ice. Normally, a pictures would not contain that many separate, closed water areas, which means that a large number of seeds has been required for each water area. Of course, a high number of seeds per image like this causes a much longer execution time. The median number of seeds per image is much lower, which tells us that there is a smaller number of images that has required a very large number of seeds, thus increasing the average number of seeds and extending the execution time significantly. If poor images were not used, the algorithm should have performed a lot better regarding both Jaccard score and execution time.

The number of iterations shows how many iterations the algorithm has required to grow in total for each image, with starting points at each placed seed. With a mean value of 541 iterations and median value of 629, one could conclude that the number of iterations is more balanced than the number of seeds in an image. This is understandable, as the number of iterations is not necessarily depended on the number of seeds but how big the water areas in the pictures are. In other words, it is difficult to draw conclusions from these numbers, as a high number of iterations does not imply a good result.

**Table 4.2:** Number of iterations and seeds.

<table>
<thead>
<tr>
<th>Number of Iterations</th>
<th>Number of seeds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>541</td>
</tr>
<tr>
<td>Median</td>
<td>629</td>
</tr>
<tr>
<td>Total</td>
<td>23,801</td>
</tr>
<tr>
<td>Mean</td>
<td>44</td>
</tr>
<tr>
<td>Median</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>1925</td>
</tr>
</tbody>
</table>

In Figure 4.2, the output masks from the algorithm can be seen. It is evident that the algorithm suffers in performance when ice or cloud cluttered images are encountered, which is understandable since these types of terrain induce salient edges in the satellite images. Since this is an edge detection algorithm and not a machine learning classifier, it does not differentiate between edges caused by ice and clouds, or actual coastlines. In clear images with smooth water areas, the algorithm performs really well.
4. Results

Figure 4.2: Morphological ACWE output.

4.3 Random Forest

The result of the RF classifier can be seen in Tables 4.3 and 4.4. With a training time of approximately 3049 sec $\approx$ 51 minutes, and a testing time of 170 sec, 3.5 sec per image, the RF classifier is a lot faster than the morphological ACWE. Since the training is only required once, and is not needed when segmenting new images, testing time will be used as a more crucial metric when comparing the execution time between the three methods.

The training time of the RF classifier is only 8 minutes faster than the U-Net CNN classifier, whose results will be presented in the next section, which is mostly due to the extraction of features which takes approximately 2200 sec $\approx$ 37 minutes. However, the RF classifier is much slower when predicting on the test data set. With a total testing time of 170 sec, it is over 9 times slower than the U-Net CNN classifier. Which is, as for training, mostly because the RF model needs to extract features before predicting.

A mean Jaccard score of 93.6% is a quite good result, and the even higher median Jaccard score of 95.7% shows that at least half of the dataset achieves a very high similarity.

As can be seen in the classification report (Table 4.5), the classifier is somewhat better in precision when classifying water than land. On the other hand, the recall is slightly better when classifying land. Since the distribution of classes is quite even, (45.3% land and 54.7% water), the Jaccard score is a better metric to use than the F1-score.

Table 4.3: Jaccard similarity results.

<table>
<thead>
<tr>
<th>Jaccard Similarity</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.936</td>
</tr>
<tr>
<td>Median</td>
<td>0.957</td>
</tr>
</tbody>
</table>

Table 4.4: Training and testing time.

<table>
<thead>
<tr>
<th></th>
<th>Training [s]</th>
<th>Testing [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>15.2</td>
<td>Mean 3.5</td>
</tr>
<tr>
<td>Total</td>
<td>3049</td>
<td>Total 170</td>
</tr>
</tbody>
</table>

As seen in Figure 4.3, the RF classifier is quite good at classifying clear images but suffers a bit in performance when given images spotted with clouds or ice. Also, a lot of
Table 4.5: Classification report results.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>land</td>
<td>0.92</td>
<td>0.94</td>
<td>0.93</td>
<td>5815448</td>
</tr>
<tr>
<td>water</td>
<td>0.95</td>
<td>0.93</td>
<td>0.94</td>
<td>7029608</td>
</tr>
<tr>
<td>avg/total</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
<td>12845056</td>
</tr>
</tbody>
</table>

pixels on land are regarded as water, probably due to RGB intensity similarity with water pixels in these spots. The result is a tolerable but noisy segmentation of land and water.

Figure 4.3: Random forest output.

4.4 Convolutional Neural Network

Results from executing on the test set can be seen in Tables 4.6, 4.7 and 4.8. The output masks can be seen in Figure 4.4.

With an average Jaccard similarity of 95.0% and a total testing time of 18 sec \(\approx 0.3\) minutes, the neural network outperforms the other two methods in both respects. The median Jaccard score of 97.9% show that a big part of the dataset achieves a Jaccard score that is extremely high, and that there are some images that lower the mean score. The training time for the U-Net CNN classifier is approximately 3523 sec \(\approx 59\) minutes, only 8 minutes slower than the RF Classifier.

As mentioned earlier, since the number of samples from the two classes are quite even, the accuracy score (Jaccard score) is considered to be a better performance metric to use than the F1-score. From the classification report it can be seen that the classifier’s precision when classifying land is slightly better than its precision when classifying water. On the other hand, the recall of land pixels is somewhat worse than the recall of water pixels.

Table 4.6: Jaccard similarity results.

<table>
<thead>
<tr>
<th>Jaccard Similarity</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.950</td>
</tr>
<tr>
<td>Median</td>
<td>0.979</td>
</tr>
</tbody>
</table>

It can be seen that the neural network classifies pixels with great accuracy even when dealing with images spotted with clouds, ice, or water containing different colored areas.
4. Results

Table 4.7: Training and testing time.

<table>
<thead>
<tr>
<th></th>
<th>Training [s]</th>
<th>Testing [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>18</td>
<td>0.4</td>
</tr>
<tr>
<td>Total</td>
<td>3523</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 4.8: Classification report results.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>land</td>
<td>0.97</td>
<td>0.95</td>
<td>0.96</td>
<td>26588297</td>
</tr>
<tr>
<td>water</td>
<td>0.96</td>
<td>0.97</td>
<td>0.96</td>
<td>27413367</td>
</tr>
<tr>
<td>avg/total</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>54001664</td>
</tr>
</tbody>
</table>

like the two last images in Figure 4.4. However, some land areas that are similar in color to water are mislabeled (see last image in Figure 4.4), but the segmentation result is not as noisy as the one in RF. The resulting output mask is a clear and continuous segmentation of land and water.

Figure 4.4: Neural network output.
Chapter 5
Discussion

5.1 Satellite images and OSM data

While examining some of the images with closer zoom levels, we discovered that the OSM coastlines were not correct at all times. Some details are missing, as bridges or piers, and some lines are located higher up on land than they should be (Figure 5.1). Since the OSM data is created by people drawing lines after image data, some mistakes are likely to occur. Especially if the images are taken with different tides, causing the coastlines to differ. In addition, some buildings might have been built or removed since the making of OSM data and the capture of these satellite photos. That is one of the reasons why we chose zoom level 9 and not closer. On this level, these kinds of mistakes are not as noticeable. It is hard to know if bigger mistakes are occurring, but this could impair the classifier’s result by giving inaccurate training labels for some pixels. Since OSM data is used as ground truth, the performance measure of the classifier output would also be slightly misleading, marking some pixels as mislabeled even if they are correctly labeled.

In the earlier versions of our implementation, the output masks from the RF classifier were used when extracting seeds for the algorithm. We chose bigger squares to extract seeds from to make sure that only certain areas of water would get picked. We later chose to use the RF classifier as a separate method to evaluate instead of connecting it with the morphological ACWE, since we did not want a poor performance of the classifier to affect the performance of the contour algorithm.

When the images were handpicked, we chose as many images containing land and water as possible. Some of these pictures might not display coastlines, but lakes instead. In the OSM shapefiles, lakes are not a part of the seas and are therefore regarded as land, which means that all lake pixels are labeled as land even though they are actually water. It is not likely that many of the handpicked pictures contain lakes, but at least some of them do (that we know of). If these pictures are included in the training dataset, the classifier will be trained wrong since OSM ground truth shows all in-land water as land. This would
5. Discussion

Figure 5.1: Detail image with zoom level 13. OSM lines are not entirely accurate (see circles). The pixels constituting the pier will be judged as incorrect if they are classified as land, according to the ground truth.

likely impair the classifiers’ performances when testing on new data. In addition, if these images are used in the test dataset, the classifiers will probably classify correctly for the most part but the resulting pixels labeled as water will be evaluated as incorrect anyway. To avoid this problem, we should have been more thorough when choosing images for the dataset to make sure no pictures of lakes were included, something we did not think of beforehand.

5.2 Morphological ACWE

As an attempt to improve images before running the morphological ACWE algorithm, we applied different levels of blur to the images. While test running on a smaller set of images with different levels of blur, we noticed that the result when using clear images was worse when using blur than without. However, results when using images spotted with clouds or ice were improved when using blur, since the edges of these areas got less salient. When running on the large dataset, the same level of blur was applied to all images, making good images worse and bad images better. Thus, we noticed that the best average result was achieved without blur, and we removed it entirely. To improve performance of the algorithm, one could have applied blur on bad images and no blur on the good ones,
5.3 Random Forest and Convolutional Neural Network

When running the Morphological ACWE algorithm, the seeds are taken from OSM data to make sure the seeds are put in water areas. As mentioned earlier, the OSM data is not always correct close to the coastline areas, and may display a coastline further up on land than is visible in the images. This may cause problems if seeds are extracted in these water areas that are actually land. If a seed is put on land area, the contour will continue to grow over land and eventually cover most of it at worst. In a best case scenario it does not reach very far because of the more irregular intensity pattern on land. If the contour grow on land, the resulting output will have a much lower similarity to the truth, and this affects the end result similarity.

Mislabeled areas in OSM are fortunately quite small (that we have seen). As explained earlier, all pixels in a square need to be labeled as water in the OSM shapefile, if a seed from that square is to be extracted. Thus, the risk of extracting seeds from a mislabeled area is higher if a smaller square size is used (since mislabeled areas are usually small). To avoid this problem, we therefore made the squares in which a seed is to be retrieved from 16×16 pixels in size, which is big enough to avoid seeds from mislabeled areas. As an example in Figure 5.1 a seed could never be put on the pier by mistake, since the pier is much thinner than 16 pixels. One immediate effect of this is that water areas that are smaller than 16×16 pixels will not be discovered and no seeds will be put there. We decided that the gain from not putting out seeds on land with risk to mislabel a large number of pixels is greater than the loss of mislabeled pixels in small water areas.

5.3 Random Forest and Convolutional Neural Network

The dataset used for the classifiers consists of three bands, the RGB layers, giving us three values for each pixel. If multispectral satellite images, that contains useful bands such as infrared, near-infrared and coastal were used instead, the models could be further improved by a more complex dataset.

Another issue with the models is the previous mentioned incorrect labels, extracted from the OSM shapefiles. Thus, resulting in the classifiers training on samples that are labeled incorrectly. This obviously decreases the performance of the classifiers, which has to be considered when evaluating the results.

Even if the RF classifier considers neighboring pixels’ RGB values when training and predicting, the final result is quite noisy, with a lot of mislabeled, single pixels both on land and in water. The classifier does not consider the predicted labels on neighboring pixels, which makes it classifying single pixels as land even if all surrounding pixels are water. To improve the result, an iterative contextual pixel classification (ICPC) can be made (Loog and van Ginneken, 2002), where original features (RGB values) are used for a first classification, and each pixel is then reclassified using the contextual information of neighbor labels in addition to the original features.

However, the CNN seems to tackle the problem better as it produces a much better visual result than the RF classifier, with a less noisy output. When experimenting with the implementation of the CNN, we discovered, and as described in previous research, that it performs quite well with a much smaller dataset for training as well.
The execution time of both classifiers consists mainly of the training time, but since training is only done once and is not needed when segmenting new images, testing time is considered a more crucial metric when evaluating the methods.

5.4 Comparison of algorithms

As seen in the result, the U-Net CNN is the best algorithm to use as it achieves a better performance in terms of both accuracy score and execution (testing) time.

The training time for both machine learning algorithms achieves a very similar result: 51 minutes for the RF classifier compared to 59 minutes for the U-Net CNN. However, the testing time for the U-Net CNN is much faster than the RF classifier. Since 27 features need to be extracted for the RF classifier for each sample, both training and testing becomes more time-consuming than if using fewer features.

As seen in the output images, the U-Net CNN performs very well even on images that are hard to analyze with human eyes, such as noisy images with clouds or images with unclear land and water areas consisting of pixels with almost the same color and intensity. The segmented result is continuous and without noise, even though some segments may be mislabeled. The RF classifier performs worse on these kind of images, with output images containing a lot of noise and also some segments of incorrectly classified pixels.

The morphological ACWE is the algorithm with the weakest performance as it achieves an accuracy of almost 10% lower and slower execution time than the other two, even when also considering training time of the models. However, experiments that we performed during this project showed that the morphological ACWE performs very well on images with clear water areas, with an accuracy score of 99% and an output mask without any noise. In comparison to the RF classifier, the morphological ACWE performs better on these kind of images as the RF classifier outputs some noise even on clear images. As the morphological ACWE does not need any training time and is relatively fast to execute on clear images, this algorithm could be the most preferable in case of a small dataset with clear images.

5.5 Improvements

There are some possible improvements that could have been made in this project. However, due to limited time and resources, these improvements are left for others to implement.

In the morphological ACWE, several improvements could be made to reduce execution time, which is also proposed by Marquez-Neila et al. (2014). They consist of the use of GPU, several threads or the narrow band technique. When using the narrow band technique, only the pixels surrounding the level set contour are included in the calculations, since they are the only ones changing, instead of including all pixels outside and inside the contour. These improvements were not implemented since we did not have the time when implementing and evaluating three different techniques.

The segmentation result of the RF classifier is quite good but contains a lot of mislabeled pixels, mostly on land. To reduce this noise, a morphological transformation could be made that is called closing. When applying this transformation, an output mask is first
5.5 Improvements

dilated and then eroded (adding border pixels to the contours and then removing a border from the contour), which would remove all single, mislabeled pixels on land. Since we wanted to compare the classifiers’ performances without post processing, we did not include this transformation. Another way to reduce noise is, as mentioned earlier to feature neighboring pixel labels as input data in an iterative contextual pixel classification, and not only consider RGB intensity of neighboring pixels.

When it comes to neural networks and Keras, Tensorflow could be run with GPU capabilities on larger datasets when a capable GPU is available. Since the morphological ACWE and RF were run on CPU, this version was not installed to make the comparison of all methods more fair.

The satellite images used only contain the three bands of RGB intensity values. However, there is multispectral satellite data available with numerous bands beyond RGB, such as an near infrared, mid-infrared and far-infrared bands which are very useful when detecting different kinds of vegetation. If this additional data would have been used, the classifying result could have been improved further.

The RF classifier could have benefited from using a larger dataset of images, since we noticed a greater difference in results when varying the size of the training dataset than with the U-Net CNN classifier. In the beginning of the project, we used a much larger dataset that ended up being too heavy for the computer to process when all 27 features of the RF were added. Thus, we settled for a smaller dataset.
5. Discussion
The following section concludes the answers to the research questions defined for this thesis and any improvements to be considered for future projects.

**How can coastlines in satellite images be detected?**

There are a lot of possible techniques to use when segmenting different types of terrain. Some common edge detection methods are: thresholding, edge detection and region based techniques, and active contours. Edge detection based algorithms will highlight all internal edges and not just the terrain desired to segment, which leaves machine learning methods as a better option, especially if the dataset contains images with a lot of noise. These methods take RGB intensity values and other features into account when segmenting areas in satellite images, thus omitting undesired edges.

**Which method is the most efficient to detect coastlines?**

The U-Net CNN classifier is the preferred method to use when detecting coastlines, since it has a high accuracy score when classifying land and water, in addition to being very fast when predicting on new images. Moreover, it classifies well even on poor images spotted with ice or clouds. The morphological ACWE performs very well on clear images and does not need any training before executed on desired images, which make this method a good alternative if a small dataset with clear images is used.

**Are there any challenges with the chosen methods?**

The morphological ACWE suffers in performance when using poor images spotted with clouds or ice, since these types of terrains induces edges in the images. These poor images causes the need of a large number of seeds to evolve the contour in water areas, which
turn causes a long execution time. This method is to be used on clear images without obstacles like clouds or ice in water areas. Improvements that could speed up execution time are: including GPU compatibility, using narrow band technique or using several threads.

The RF classifier performs quite well even in poor images, but the resulting segmentation is noisy from mislabeled pixels mostly on land but also in water. An improvement that could be made is using some kind of morphological transformation like closing, where the output mask is first dilated and then eroded to fill in mislabeled pixels on land. In addition, an iterative contextual pixel classification can be made for a less noisy classification.

The U-Net CNN classifier is both fast and accurate, which leaves few challenges with the method. Some areas are mislabeled, but the result is not as noisy as with the RF method but provides a continuous segmentation, which is desirable when detecting coastlines.

Hopefully, the research of this Master’s thesis provides new insights and ideas for similar projects to be implemented in the future.
Bibliography


Detektion av kustlinjer med hjälp av maskininlärning

Det finns ett behov av att segmentera olika typer av terräng i satellitbilder, till exempel för att upptäcka naturförändringar i forskningssyfte eller vid utveckling av kartapplikationer. Genom att segmentera land och hav kan förändringar i vatten- och havsnivå observeras, som ger data som kan vara värdefull inom klimatforskning. Vid utveckling av kartapplikationer finns också behovet av segmentering. Bilddata för havsområden används inte och därför krävs det komplettering av data vid sammansättning av segment till en fullständig karta.

European Space Agency (ESA) driver ett projekt som kontinuerligt tillhandahåller data från Sentinel-satelliter. Genom att kombinera Sentinel-data med geografisk vektordata från organisationen OpenStreetMap (OSM), har vi skapat ett dataset med kustbilder från olika världsdelar. Den vektordata som vi har använt innehåller en geometrisk beskrivning av land- och havsområden över hela världen.

Vi har jämfört tre algoritmer, varav två maskininlärningsalgoritmer: en random forest (RF) klassificerare och ett neuralt nätverk (NN), och en algoritm för kantdetektion: morphological active contours without edges (MACWE). En större del av datasetet har används för att träna maskininlärningsalgoritmerna. Resterande del har sedan använts för att utvärdera samtliga metoder och avgöra vilken av dessa som presterar bäst med avseende på precision och exekveringstid.


Med ett resultat på 95% precision och snabbast exekveringstid är det neurala nätverket den metod som presterar bäst.