NTNU

Norwegian University of
Science and Technology

Map Creation from Semantic Segmentation of Aerial Images Using Deep Convolutional Neural Networks

Utilizing publicly available spatial data to
make an aerial image labeling dataset

Ruben Schmidt Mällberg
Valdemar Edvard Sandal Rolfsen

Master of Science in Engineering and ICT
Submission date: June 2018
Supervisor: Terje Midtbø, IBM

Norwegian University of Science and Technology
Department of Civil and Environmental Engineering

Abstract

For centuries cartographers have segmented and labeled the surface of the earth onto analog and digital maps. Today, map-making is still a time-consuming, manual process that requires large amounts of work.

In this thesis, we investigate the possibility of using publicly available spatial data to train deep convolutional neural networks into performing accurate semantic segmentation of aerial images. We collect spatial data from the Norwegian Mapping Authority and propose a dataset consisting of aerial photographs and high-quality labels from five cities in Norway. The dataset consists of four different spatial classes: Buildings, Roads, Water and Vegetation. We present detailed statistics and highlight issues in the dataset and show that the technique for creating the dataset can be expanded to include data from all of Norway.

To investigate the possibility of creating maps automatically, we adopt two deep convolutional neural networks and train them on the proposed dataset. We show that training one network for each semantic class yield better results than training one network on all the classes simultaneously.

Finally, we present maps produced by the networks and assess its quality and usability. The results show that our method can produce useful maps without the need for pre- and postprocessing of the data.

Keywords: High-quality aerial images, deep learning, convolutional neural networks, automatic map creation

This page intentionally left blank.

Sammendrag

Kartografer har i flere århundrer segmentert og tegnet jordens overflate på analoge og digitale kart. Å lage et kart er fortsatt en manuel og tidkrevende prosess som krever store mengder arbeid.

I denne masteroppgaven, undersøker vi muligheten for å bruke offentlig tilgjengelige data til å trene dype konvolusjonelle nevrale nettverk for å gjøre nøyaktig segmentering av flyfoto. Vi samler inn data fra Kartverket og foreslår et dataset som består av flyfoto og høy kvalitets masker fra fem byer i Norge. Datasettet består av fire ulike romlige klasser: Bygning, Veg, Vann og Vegetasjon. Vi presenterer detaljert statistikk og fremhever problemer i datasettet og viser at teknikken som brukes for å lage datasettet kan utvides slik at datasettet inneholder data fra hele Norge.

For å undersøke muligheten for å lage kart automatisk, tilpasser vi to dype konvulsjonerende nevrale nettverk og trener de på det foreslåtte datasettet. Vi viser at det å trene ett nettverk for hver semantisk klasse gir bedre resultater enn å trene ett nettverk på alle klassene samtidig.

Til slutt presenterer vi kart produsert av nettverkene og vurderer kvalieten og brukervennligheten på disse. Resultatene viser at metoden vår kan produsere brukbare kart uten noen form for pre eller postprosessering av dataene.

This page intentionally left blank.

Preface

This paper is a master thesis written for the Department of Civil and Transport Engineering at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. It is part of the study program Engineering and ICT, with a specialization in Geomatics. The thesis was written in the spring of 2018.

We would like to thank our supervisor, Terje Midtbø, for his assistance during the writing of this thesis. We would also like to thank our coworkers, fellow students, and family for valuable input and discussions. Lastly, we would like to thank Anne Sofie and Nora. Your love and support has helped us keep our motivation and focus during these long and hard working days.

Trondheim, 2018-06-11

Ruben Schmidt Mällberg
Valdemar Edvard Sandal Rolfsen

This page intentionally left blank.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the background and motivation and state our goals for this master thesis. We continue with explaining our method and discuss the limitations of our work before we give an outline of the rest of the thesis.

1.1 Background and Motivation

Feature extraction and semantic segmentation of images, using supervised machine learning algorithms, have become the leading field of research within computer vision since Krizhevsky, Sutskever, and Hinton [52] won the ImageNet LSVRC-2010 contest with their Deep Convolutional Neural Network. Since then these algorithms have been used in the development of technologies such as self-driving cars, face recognition and natural language processing [5, 32]. The remote sensing community applies machine learning algorithms in numerous ways to extract meaningful information from aerial images [97, 96, 36, 66, 35].

One of the most significant challenges for supervised machine learning is to acquire enough training data. A neural network, for example, often require millions of training examples to generalize well beyond the data initially seen. One branch of machine learning that has proven to require a substantial amount of data is semantic image segmentation [84].

A problem that arises when trying to create large datasets is the amount of manual work required to label the data. In semantic segmentation problems, each pixel in the raster label has a color value corresponding to its respective class, as seen in Figure 1.1. Hence, creating such datasets can be both difficult and time-consuming [60].

High-quality aerial imagery combined with geographical information provides an ideal source for creating semantic segmentation datasets since the purpose of the geographical information is to represent the position and extent of spatial objects.

Figure 1.1: Training example and corresponding label from city scene [100]

In Norway, the task of producing, storing and maintaining a digital database for geographical information, is the responsibility of the Norwegian Mapping Authority and the Geovekst [71] initiative. This database, referred to as FKB (Felles

Kartdatabase), contains large quantities of information, ranging from the use of land to cadastre data, and the information within this database is considered to be of high quality.

New machine learning contests such as the SpaceNet challenge [90] and DSTL [43] shows that there is an increasing focus on semantic segmentation of remote sensing data, but the amount of high-quality training datasets is still limited compared to other branches within machine learning.

1.2 Research Goals

We base the work conducted in this thesis on two research goals:

G1: Create a high-quality training dataset based on publicly available data from the Norwegian Mapping Authority where the label creation process is automatable.

As machine learning becomes more prominent within the remote sensing community, the need for high-quality training data grows. Because of the significant workload required to label such training data manually, exploitation of already existing official spatial data can be a better approach. We also want to focus on creating the labels for the dataset in a way that can be easily automated, without the need for human intervention.

G2: Train different configurations of convolutional neural networks on the proposed dataset.

If the datasets are to bring any value to the research field, they have to be of a certain quality. Therefore, there is a need for a detailed analysis of the performance of different machine learning implementations when being trained on datasets created from publicly available spatial data.

We combine our goals to create an overall question that we aim to answer in this thesis:

Can deep convolutional neural networks learn to create maps based on publicly available spatial data? If we are to answer our research question, it is apparent that we need to reach both **G1** and **G2**.

1.3 Research Methods

To reach the specified goals, we first dive into the relevant theoretical background and conduct a series of literature reviews to gain a better understanding of the technology and methods that exist today. Afterward, we use what we have learned to evaluate relevant techniques by applying them to our problem.

1.3.1 Study of Theory and Related Work

We start by looking into the theoretical background that is relevant to the thesis. Then we conduct literature reviews for the creation of different machine learning datasets, different methods for semantic segmentation and lastly automatic map generation and feature extraction from remote sensing data.

1.3.2 Analysis, Implementation and Experiments

We start by giving a definition of our map and analyzing the FKB dataset to determine its magnitude and level of detail. Then we make a qualitative decision on which of the FKB classes to include in the dataset.

We use the defined classes to create a geospatial database that can be used to query geographical information from different areas in Norway. We combine this database with tiled, aerial images from Norge i Bilder (NIB) to generate training examples and labels. Using this approach, we aim to solve **G1**.

These examples and labels are used to train different implementations of deep convolutional neural networks. We evaluate their predictions and compare them with state-of-the-art segmentation results from other, previous work, to address **G2**.

1.4 Limitations

The machine learning community is, at its current state, developing at a very high pace, continually publishing improved algorithms. Even though this thesis is trying to present the performance of the dataset on the most advanced neural networks currently existing, we see examples of new findings continually appearing [53].

Furthermore, while most of the networks presented in the reviewed research papers train on large machines with high computational power, our work is limited by

available resources.

1.5 Outline

We structure the remaining chapters of this thesis into the following parts: *Related Work*, *Evaluation* and *Discussion and Conclusion*.

1.5.1 Theory and Related Work

Chapters 2-5 concern the early research phase of this thesis, which includes looking into the relevant theoretical background material and reviewing relevant literature to acquire a deeper understanding of the subject.

- **Theoretical background** (chapter 2): The chapter gives a theoretical background of the methods and theory used in the thesis. We start by looking at FKB data in general and how it is commonly retrieved through photogrammetric cartography. Furthermore, we present the basic concepts of artificial neural networks, before we present convolutional neural networks.
- **Datasets in Computer Vision**: (chapter 3): In this chapter, we present the previous work related to creating datasets for training machine learning algorithms. It describes earlier work of the creation of computer vision datasets for object classification, object detection, and semantic segmentation.
- **Semantic Segmentation Methods** (chapter 4): This chapter includes a description of standard methods for image segmentation, an overview of the advances in convolutional neural networks and how one can use these networks for semantic image segmentation.
- **Feature Extraction in Remote Sensing** (chapter 5): The last chapter in the first part of this thesis describes different approaches of map generation, through semiautomatic and automatic feature extraction methods.

1.5.2 Implementation and Assessment

In chapter 6-9, we present the methodology based on our findings from investigating the related work. We also conduct experiments and show the results of these.

- **Methodology** (chapter 6): The chapter describes the methods we use in this thesis, including a definition of our map, a technical description of how

we generate the dataset, what classes of the FBK dataset we use and how we implement the different artificial neural networks.

- **Proposed Dataset** (chapter 7): This chapter first presents the generated dataset with examples of labels and statistics explaining the different features. We expose some errors present in the dataset. Lastly, we show the full potential of the dataset.
- **Algorithmic Experiments and Results** (chapter 8): In this chapter, we present all experiments performed on the generated dataset. The experiments involve finding the proper configuration for the networks and the training of them on the dataset. We both train the networks to perform binary and multiclass predictions on the defined classes.
- **Results from the Predictions** (chapter 9): The chapter presents the final predictions of our work, including examples of binary predictions from both the networks and multiclass maps for larger areas.

1.5.3 Discussion and Conclusion

In chapter 10 and 11, we discuss and conclude our findings from the conducted research within the scope of this thesis.

- **Discussion** (chapter 10): We discuss different approaches and choices taken during the writing of the thesis, including the quality and basis of the dataset, the chosen configuration and the accuracy of the predictions after training on the dataset.
- **Conclusion** (chapter 11): In the last chapter, we present our conclusion together with suggestions for further work.

Part I

Theory and Related Work

This page intentionally left blank.

Chapter 2

Theoretical background

In this chapter, we identify and give the theoretical background needed for this thesis. The theory includes concepts, definitions, methods and regulations/key standards to explain the methods and techniques used in the rest of the thesis.

2.1 The Definition of a Map

To define what a map is, we use the definition presented by Anson and Ormeling [2]:

> *"A convetionalized image representing selected features or characteristics of geographical reality designed for use when spatial relationships are of primary relevance."*

According to Heywood, Cornelius, and Carver [29], the mapping process consists of several steps:

1. Establish the purpose of the map.
2. Define the scale.
3. Select the features which must be portrayed on the map.
4. Choose a method for representation of these features (points, lines, areas).
5. Generalize these features for representation in two dimensions.
6. Adopt a map projection for placing these features onto a flat surface.
7. Apply a spatial reference system to locate these features relative to each other.
8. Annotate the map with keys text legends to facilitate the map.

2.2 Map construction using photogrammetry

In general, photogrammetry denotes the process of deriving metric information of different objects from photographs by measuring distances. By using the principles of collinearity, we can calculate the real-world distance between two points by knowing the distance between the points on the image, the projection angles of the image and the image scale. In geomatics, photogrammetry is used to create elevation models, *orthophotos* and maps.

FKB data is, as mentioned in section 2.3, mostly derived from photogrammetric map production. The process involves taking aerial photographs of a selected area and georeferencing these photographs using aero-triangulation and block adjustment techniques (see subsection 2.2.1) to create accurate stereo models and use these models for mapping.

2.2.1 Georeferencing aerial images

We divide the process of georeferencing images using aero-triangulation and block adjustment into two parts; Interior Orientation and Outer Orientation.

Interior Orientation The purpose of interior orientation is to reconstruct the bundle of beams that formed the image at the time of projection. The reconstruction is done by finding the parameters that define the inner geometry of the camera at that specific moment.

Outer Orientation The purpose of outer orientation is to find a connection between the two-dimensional image coordinates and three-dimensional coordinates in a terrain model. The connection can be found by following the collinearity constraint, which states that every point that is part of an orientation (three points in the outer orientation of an aerial photo) should all be on the same line. Shown in Figure 2.1.

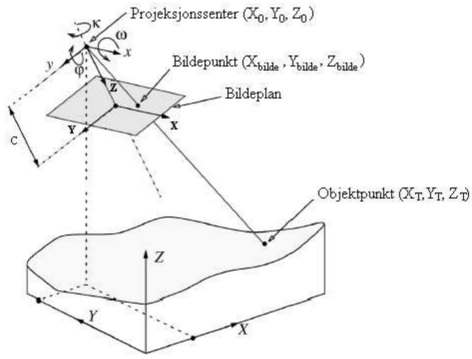

Figure 2.1: Outer Orientation using the bundle method

2.2.2 Vectorizing georeferenced images into maps

By using georeferenced stereo images, it is possible to extract features such as buildings, roads and contour lines directly from the images, using a digital photogrammetric workstation.

Digital Photogrammetric Workstation (DPW) A DPW is a high-performance computer connected to a screen with stereoscopic capabilities and an input device for 3D-navigation. The machine has to have installed photogrammetric software

that is capable of handling large digital images. Even though there are many applications for the DWP, the primary usage is 2D feature extraction or vector mapping [58].

Feature extraction is used to label different properties of aerial images such as houses, rivers, roads, fences, and poles. As seen in Figure 2.2. Feature extraction today is mainly done manually, but there exist attempts to automate parts of the process.

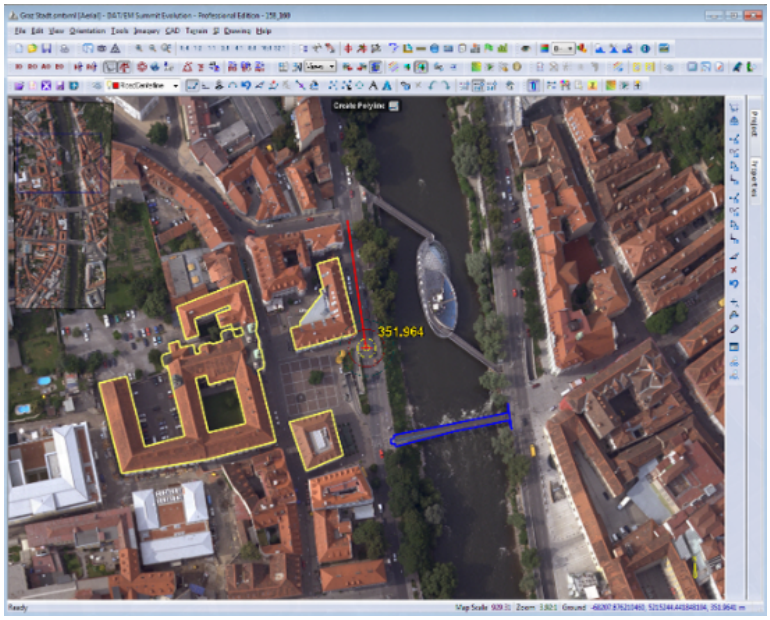

Figure 2.2: Feature extraction using a DPW

In addition to creating maps from stereo photos, it is possible to use the aerial images for feature extraction without the use of specialized equipment, if the pictures go through orthorectification.

Orthophotos A single image is a central projection where the scale is not uniform and thus contain geometric distortion. Maps are what we call an orthogonal projection, where all points are projected with parallel lines and have a uniform scale. To use a single image as a background map or as an accurate description of the true distances in the real world, we need to orthorectify it. Rectification is the process of removing the distortion of each point in the image. Instead of viewing the stereo model as described above, the stereo model is used to create a digital elevation model of the area. The elevation model is then used to calculate the distortion at each point in the image and shift the point accordingly.

In standard orthophotos, we correct the scale on ground level. However, the photos still maintain the vertical relief displacement, e.g., the sides of a building would still have distortions. Orthophotos where the vertical relief displacement also has been corrected, are called *true orthophotos*.

2.3 Felles kartdatabase (FKB)

FKB is a collection of map data that makes up for the public map base of Norway. It consists of data collected by the Norwegian mapping authority, the Norwegian municipalities and other agencies in an initiative called Geovekst [71]. It includes detailed data on contour lines, water, land cover, land use, buildings, roads and more. We see an example of the data in Figure 2.3.

Figure 2.3: FKB data visualized in a geographical information system

FKB focuses on the same cartographical disciplines as the old map series ØK (Økonomisk Kartverk) and TK (Teknisk Kartverk). Its purpose is to aid in engineering tasks, administrative case management in municipalities, analysis and presentation in geographical information systems and production of maps.

FKB delivers the data in four quality categories, based on detail, localization accuracy and the location of the data. The four categories are shown in Table 2.1, the locations the categories are based on are shown in Figure 2.4 and the detail levels of each category can be seen in Figure 2.5.

Although parts of the FKB dataset is retrieved from other sources than the Norwegian Mapping Authority, such as the Traffic Authority, municipalities and others in the Geovekst initiative, most of the data in FKB is gathered through photogrammetric map construction, as explained in section 2.2.

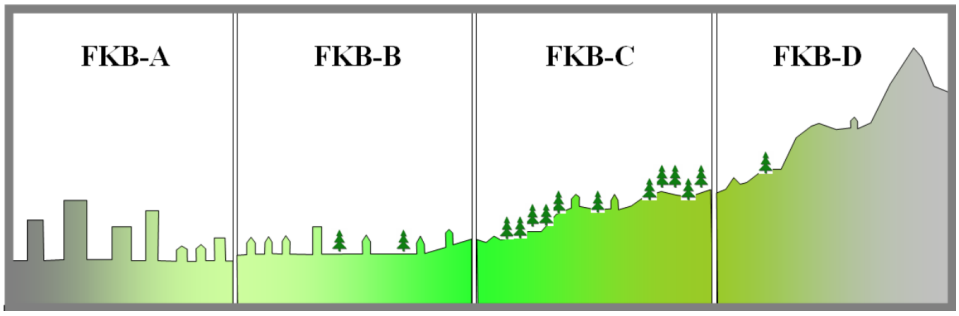

Figure 2.4: Areas of different levels of detail [47]

Category	Description
FKB-A	Used for areas covering larger cities with a high degree of utilization. The level of detail in the A-standard is very high. Objects such as buildings have features such as ridgelines, porches, and stairs. Furthermore, it includes heights for objects such as fences, walls, and masts.
FKB-B	The B-standard covers areas of smaller cities, towns, larger roads, railroads and developing regions. The level of detail is very similar to that of the old technical maps (Tekniske Kart).
FKB-C	Used for other populated areas. It has the same level of detail like the traditional economic maps (Økonomiske kartverk). The level of detail of, i.e., buildings can vary based on where the data is collected.
FKB-D	Used in rural areas such as mountains. Most of the data is collected from N50. Thus the level of detail is close to that of N50. Some roads and houses are however collected from the official road database and cadastre data respectively.

Table 2.1: The different categories in the FKB data

2.4 Artificial Neural Networks

In the later years, there has been increasing use of artificial neural networks (ANNs) for feature extraction in remote sensing images [55]. The fundamental principle behind ANNs is that it is built up by a network of many simple units, called neurons, with no centralized control unit [70]. The ANNs primary means of long-term storage is the weights between the individual units, and updating the weights is the most important way the network learns new information. An ANN is defined by the number of neurons, layers and the connections between the layers.

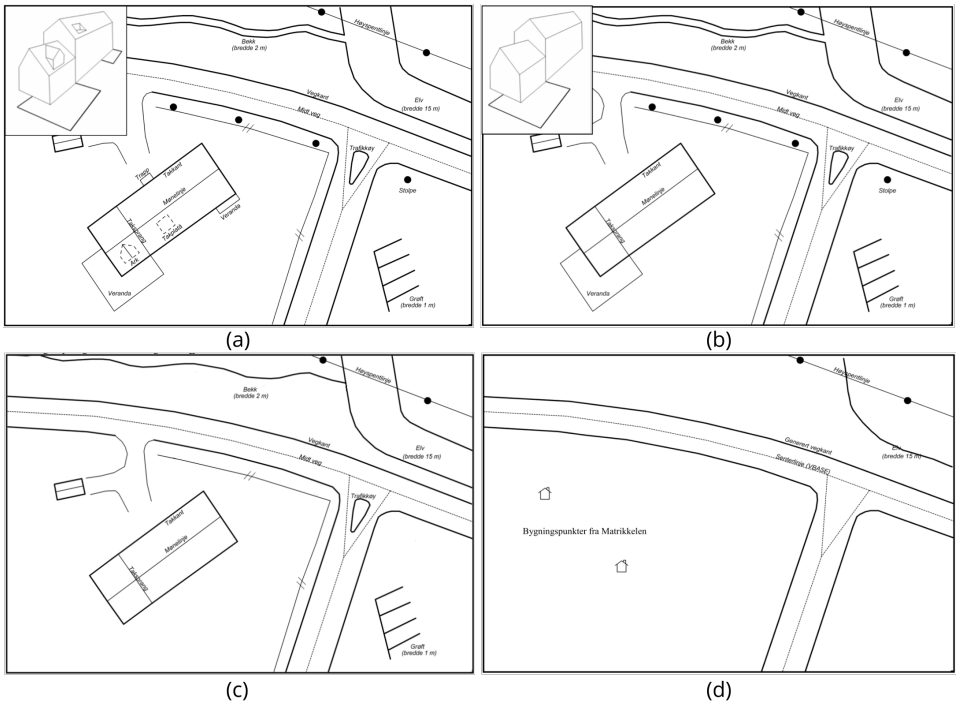

Figure 2.5: Different levels of detail in the FKB data. FKB-A (a), FKB-B (b), FKB-C (c) and FKB-D (d) [47]

2.4.1 Artificial neurons

Each layer in the ANN consist of one or more artificial neurons also called nodes or units. An artificial neuron is a mathematical representation of a biological neuron and consist of inputs with weights and bias, a transfer function and an activation function. The weights are what scales, either amplifying or decreasing, the input to the node. The bias is a constant scalar value per layer that is added to ensure that at least some of the nodes in the layer are activated, that is, forwarding a non-zero value to the next layer. The transfer function takes the weighted sum of the input variables and transfers it to the *activation function*. A model of an artificial neuron compared to a real neuron, can be seen in Figure 2.6.

2.4.2 Activation functions

Activation functions are scalar-to-scalar non-linear functions that define the output of a node based on the inputs, weights, and bias. They are what enables the network to model non-linear functions. Note that sometimes the identity function $f(x) = x$

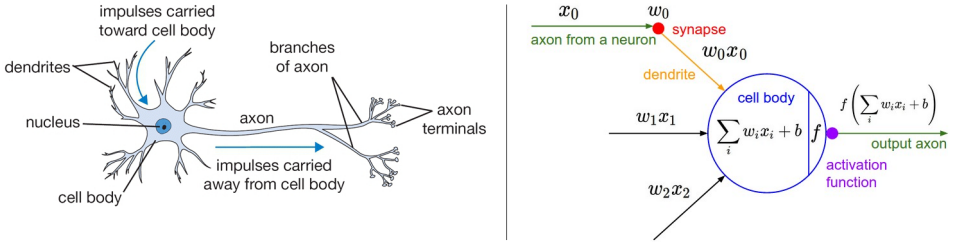

Figure 2.6: Structure of a real neuron compared with an artificial one [44]

is referred to as an activation function, it is however only used in the input layer of the network, and is not one of the non-linear activations used in neurons. The most common non-linear activation functions are:

Sigmoid The sigmoid activation function belongs to the class of logistic activation functions. It coverts independent variables of near infinite range into probabilities in the range [0, 1]. Expressed mathematically as:

$$a = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

Tanh The tanh function represents the ratio between the hyperbolic sine to the hyperbolic cosine that outputs values in the range [-1, 1]. Expressed mathematically as:

$$a = \sigma(x) = \tanh(x)$$

Softmax The softmax function also referred to as the normalized, exponential function, is a generalization of the logistic function that is most often used to weight the largest value in a set and dampen values that are considerably smaller. It returns the probability distribution over mutually exclusive output classes. For example, if the softmax activation function was applied to the vector [1, 2, 3, 4, 1, 2, 3], the result would be [0.024, 0.064, 0.175, **0.475**, 0.024, 0.064, 0.175,]. We can express Softmax as follows:

$$a = \sigma(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

where j indexes the input vector so $j = 1, 2, ..., K$.

Rectified Linear The rectified linear unit (ReLU) function only activates a node if it is above some threshold (usually zero). When the input is above the threshold, the output has a linear relation to the input:

$$a = \sigma(x) = \max(0, x) \tag{2.1}$$

ReLUs are considered the state-of-the-art because of their proven usefulness in many situations, and their ability to train better in practice than sigmoids [52]. ReLUs does not have the problem of *vanishing gradients*, where networks lose the ability to learn. The reason is that significant changes in the value of parameters in the early layers do not have a significant effect on the output since some activation functions "squash" the input space into small regions.

While removing the problem of vanishing gradients, ReLUs introduce another problem. *Dying ReLU* [44] occurs when a weight update in a ReLU node happens in such way that the neuron will never activate again, making the gradient passing through the neuron always to have a zero-value.

2.4.3 Feed-forward neural networks

The most common ANN architecture is the feed-forward neural network. The network is built up of one input layer, one or more hidden layers and an output layer. A network that has no hidden layers is called a perceptron, and these types of networks are only able to model linear functions [65]. A feed-forward network with one or more hidden layers (with non-linear activation functions), is shown to be a *universal approximator* and can thus model any continuous function [16].

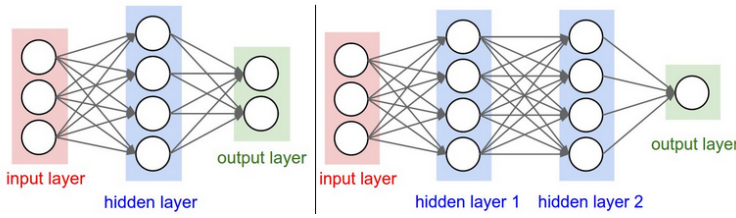

Figure 2.7: Example structures of feed forward ANNs [44]

Input Layer The input layer is the layer that feeds the information into the network. The number of neurons is typically equal to the number of features in the data.

Hidden layer The hidden layers with non-linear activation functions are what enables the network to learn and model nonlinear functions. It is the weights on the connections between the different layers that enables the network to encode the information extracted from the training data.

Output layer The output layer extracts the answer or prediction from the model. Depending on the setup of the neural network, the output value can either be a real value or a set of probabilities. The output type is dependent on the activation function that we choose for the layer.

A feed-forward network can either be fully or partly connected. The difference is that in a fully connected network (FCN), all the neurons in each layer have a connection to all neurons in the previous the next layer.

2.4.4 Training a neural network

There are different forms of learning: Supervised learning where we show the network the correct answer, unsupervised learning where the network itself decides how to label the data, and reinforcement learning where the network does not get to know the answer but learns by reward or punishment. We will only focus on supervised learning in this thesis since it is the form of learning we use when doing semantic segmentation with ANNs.

The primary purpose of training an ANN is to adjust its weighted connections to allocate significance to some features and removing it from others. In supervised learning, the network learns by training on a set of inputs and desired outputs. As inputs pass through the network and outputs are generated it learns by adjusting weights and biases, causing some neurons to become smaller and some to grow larger. The larger a neuron's weight is, the more it affects the network and vice versa.

By adjusting the weights and biases, the network reduces the errors, also called loss. The loss is defined by a *loss function* that quantifies the correctness of the output compared to the ground truth. By using a loss function, we reclass the learning problem as an optimization problem, where we try to minimize the loss.

The most common learning algorithm associated with neural networks is the *back-propagation learning algorithm*.

Backpropagation learning

The backpropagation algorithm adjusts the weights by first taking a forward pass through the network to compute an output value. If the output produces a significant loss, the weights are updated accordingly. We do the weight update with the following equation:

$$W_{j,i} \Leftarrow W_{j,i} + \alpha * a_j * Err_i * g'(input\_sum_i) \tag{2.2}$$

Equation 2.2 is called the weight update rule for the connection between neurons in layers j and i. Furthermore, α is the learning rate (discussed in subsection 2.4.6), a_j is the incoming activation function, Err_i is the error in i and $g'(input\_sum_i)$ is the gradient of the activation function over the input sum

$$a_j = g(input\_sum_j) \tag{2.3}$$

where the $input\_sum_j$ is defiend as:

$$input\_sum_i = W_i * x_i + b \tag{2.4}$$

where W_i is the current weight, x_i is the input variable and b is the bias.

The backpropagation algorithm traverses the network backward, updating the weights of the connections between each layer until it reaches the input layer. This way it reduces the influence of the weight connections that are assigned the blame, while it strengthens the connections that support the correct answer.

2.4.5 Loss functions

We describe the ideal state of the network as the state that classifies all the examples correctly. The loss function is a way of quantifying how close a network is to this ideal state. This quantification is done by aggregating the errors produced by the network's prediction over the entire training dataset and average this value to get a single number that represents how close the network is to its ideal state. We will now go through some common loss functions.

Categorical Cross Entropy Loss

In machine learning, cross entropy is often used to calculate the loss between the true probability distribution p and the predicted probability distribution q. For multinomial classification problems, we can use the categorical cross entropy loss function. For the categorical cross entropy loss function to work, the labels in the dataset have to be one hot encoded. Meaning that if the dataset consists of 5 classes, we use a 5-dimensional array to represent the labels where all values are zero, except at the index of the corresponding class where the value is one (see Figure 2.8).

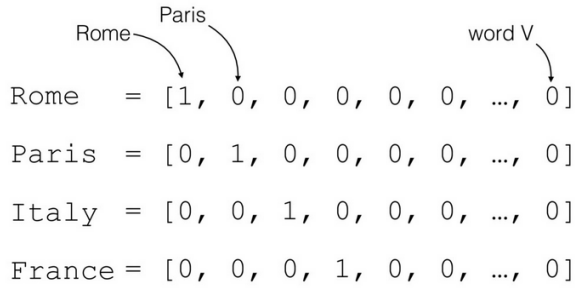

Figure 2.8: Example of one hot encoded labels

Let p be the predictions in the range [0, 1] from a function such as Softmax and t be the targets represented as a one-hot encoded vector. We define the categorical cross-entropy between the predictions and targets as:

$$L(p,t) = -\sum_{x} p(x) log(t(x))$$

Binary cross entropy

Binary cross-entropy is the particular case where the number of classes is two. In this type of problem, we can model the network output to be in the range [0,1] after being passed through a Sigmoid activation. Here the two classes are represented by a single, binary output value and not by separate probability distributions. The intuition behind only using a single output is that the probability distribution of class A is equal to one minus the probability distribution of class B ($p(classA) = 1 - p(classB)$). Therefore having two distributions, as in the one-hot encoded version, is unnecessary. We define the cross-entropy from the single output as:

$$L(p,t) = -tlog(p) - (1-t)log(1-p)$$

2.4.6 Hyperparameters

In machine learning, hyperparameters are values that we tune to make the network train faster and better. The selection of these parameters is made to ensure that the network neither overfits nor underfits the data.

We say that a model overfits its training data when there exists an alternative model with the same or higher training error that generalizes better. In other words, overfitting refers to a model that fits its training data too well. It learns the detail and noise of the training data to the extent that it negatively impacts the performance of the network on new data. Meaning that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize. On the contrary, we say that our model is under fitted if it generalizes too much and is not able to fit the training data. We see the terms illustrated in Figure 2.9.

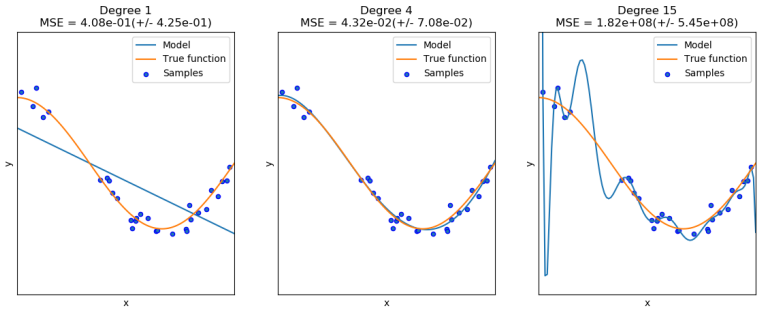

Figure 2.9: Left: Underfitted model. Middle: Appropriate fit. Right: Overfitted model

Learning rate The learning rate, as seen in Equation 2.2 is a coefficient that scales the size of each weight update step. The learning rate decides how much of the computed gradient it uses when updating the weights in the network. A large learning rate will update the weights more at each step but can make the optimizer overshoot the minimum of the loss function. A small learning rate will update the weights less at each step and should help reach the minimum, but it can take a significant amount of time.

The learning rate is one of the most challenging hyperparameters to tune, because of its significant impact on learning. A conventional technique that helps lower the significance of the initial learning rate is to *anneal* the learning rate over time. The intuition behind annealing the learning rate comes from the fact that with a large learning rate the optimizer can jump around in the loss space, causing it to not settle down in the deeper narrower parts of the loss space. There are different forms of annealing. The most common is *step decay* annealing, where we lower the learning rate by some factor after a few epochs or when for instance the validation error stagnates. Smith [82] presents a new method for adjusting the learning rate, named *cyclical learning rates*, where the learning rate cyclically varies between two reasonable boundary values. This method lowers the need to experimentally find a good learning rate since the two boundaries are simple to obtain using a learning rate test proposed in the paper.

We will later see in subsection 2.4.7 that some of the optimizers have built-in learning rate decaying to eliminate the need to tune the learning rate before training correctly.

Regularization Regularization is essential to control what is called out-of-control parameters. We do this by controlling the trade-off between finding a good fit on the training data and limiting the weight of features with high polynomials since these tend to overfit the training examples. A common form of regularization is L2, where the term $\frac{1}{2}\lambda w^2$ is added to the weight [44]. Another form of regularization is *dropout* [83]. Dropout works by randomly dropping some of the neurons during training, causing it to train on a thinned version of the network. Dropout has shown major improvements over the other regularization techniques [83].

Momentum We often see Momentum described as the learning rate of the learning rate. It is a method that helps accelerate stochastic gradient descent (SGD) algorithms in the relevant direction. We can view SGD with momentum as pushing a ball downhill. It increases its speed when constantly going downhill. The same happens to the gradients. They get increased speed when going in the same direction.

A problem with momentum is that when the gradients get a big accumulated speed, the momentum algorithm takes a big jump in the direction of the gradient, making the algorithm not follow the steepest descent, and in the worst case, to overshoot the minimum. *Nesterov Accelerated gradient* (NAG) [67] tries to solve this problem by making the momentum algorithm smarter. Instead of taking a big jump in the direction of the current gradient, it takes a big leap in the direction of the previous

gradient and measures where it ends up. Making it able to make corrections before doing the jump. We can see both methods illustrated in Figure 2.10

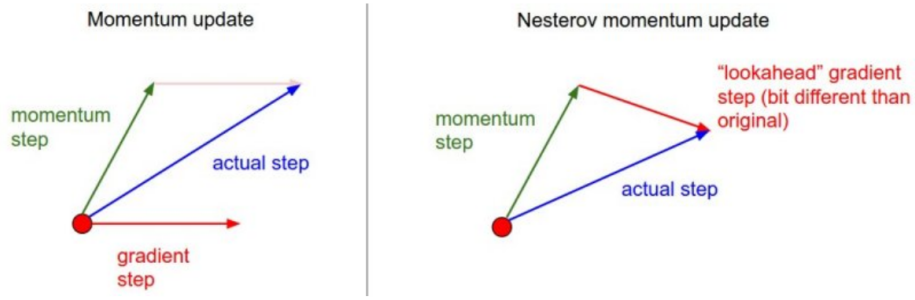

Figure 2.10: Standard momentum updated and Nesterov momentum update.

2.4.7 Optimizers

The loss functions quantify the quality of the weights in neural networks. The goal of gradient descent optimization is to find the weights that minimize the loss function. Mathematically this can be described as minimizing the loss function $L(\theta)$, that is parameterized by the parameters $\theta \in \mathbb{R}^d$, in the direction of the gradient of the loss function $\nabla_\theta L(\theta)$.

We present some of the most common gradient descent optimizers used to train neural networks below.

Stochastic gradient descent

Standard gradient descent computes the gradient over the entire training set for each update. Stochastic gradient descent (SGD) is the same as standard gradient descent, except that it calculates the gradient for each *mini-batch* of n training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta; x^{i:i+n}; y^{i:i+n})$$

Where η is the learning rate, x is the training example and y is the training label. All of the optimizers described next, also does *mini-batch* gradient descent.

Adagrad

Duchi, Hazan, and Singer [20] propose the adaptive gradient algorithm (Adagrad) that adapts the learning rate during training. Adagrad uses a different learning rate for each of the parameters θ_i at every step t and is a part of the *per-parameter adaptive learning rate methods*. The parameter update then becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Where $g_{t,i}$ is the gradient of the loss function w.r.t the parameters θ at each step t, G_t is the diagonal matrix of the sum of the square of the past gradients, ϵ is a smoothing term to avoid zero-division.

A benefit with Adagrad is that the built-in adaptive learning rate adjustment eliminates the need to tune the learning rate manually. The problem is, however, that since G_t contains the sum of the square gradients, this term becomes larger and larger, causing the learning rate to become very small, thus making the algorithm unable to learn anymore.

Adadelta

Adadelta [98] tries to fix the problem of Adagrad, by reducing the decreasing learning rate. Adadelta does this by making the term G_t contain only the sum of the past gradients for a smaller window, defining the sum of gradients as a decaying average of all the past gradients, essentially a running average $E[g^2]_t$. The denominator term of Adagrad, $\sqrt{G_{t,ii} + \epsilon}$, thus turns into the root mean square (RMS) error of the gradient, $\sqrt{E[g^2]_t + \epsilon}$. The RMS term causes a mismatch of units in the equation (see proof in the paper citeZeiler2012), the authors fix this by adding the RMS term to the numerator making the update equation become:

$$\nabla\theta_t = -\frac{RMS[\nabla\theta]_{t-1}}{RMS[\nabla\theta]_t} g_t$$
$$\theta_{t+1} = \theta_t + \nabla\theta_t$$

As we can see, the learning rate is not a parameter in the update rule anymore.

RMSprop

Root Mean Square propagation (RMSprop) is an unpublished method proposed in [30]. It is similar to Adadelta in regards to solving Adagrads diminishing learning rates. The update rule is also the same expression as before the introduction of the RMS term in the numerator:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

Adam

Adaptive Moment Estimation (Adam) [50] is also in the family of per-parameter adaptive learning rate methods. It stores an exponentially decaying average of the past squared gradients v_t, like RMSprop and Adadelta. In addition to this, Adam saves the exponentially decaying average of the past gradients m_t, similar to what momentum does. It is designed to combine RMSprop and Adagrad. The decaying averages are computed as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$$
$$v_t = \beta_2 m_{t-1} + (1 - \beta_2)g_t^2$$

m_t and v_t are estimates for the mean (first moment) and the uncentered variance (second moment) of the gradients respectively. β_1 and β_2 are hyperparameters that control the exponential decay rate of the moving averages. They are initialized as zeros, causing the estimates to be biased towards zero in the initial timesteps, to counteract this, the authors use bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

This gives the update rule:

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

Nadam

Nesterov-accelerated Adaptive Moment Estimation (Nadam) [19] combines Adam with NAG. The momentum term m_t is changed in order to do the Nesterov update:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} (\beta_1 \hat{m}_t + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t})$$

2.5 Convolutional neural networks

In recent years, convolutional neural networks (CNN) have become recognized as very suitable for tasks involving object recognition and semantic segmentation of images [70]. The name comes from the networks use of convolutions, a mathematical operation on two functions to produce a third.

A well-known problem when analyzing image data using conventional feed-forward multilayer neural networks is that they do not scale well with increasing image sizes. Take for example a series of 400x400 pixel images, used as the input for an ANN. Each image, represented as a vector, would create $400 * 400 * 3 = 480000$ different weight connections for each neuron in the first hidden layer. For a fully connected network, this would be the case for all layers to come, which would create a tremendous amount of weight connections.

Convolutional neural networks solve this issue by representing the images in a three-dimensional structure, meaning that they represent the input data as a three-dimensional matrix with image width, image height, and color channels.

Architecture

As seen in Figure 2.11 the network can be divided into three sections; Input, Feature-extraction and Classification layer(s). The most interesting part of this structure is the feature-extraction layers, which are used to identify features in the images, and from these construct higher-order features. The strategy of constructing high-order features is one of the key aspects of deep learning.

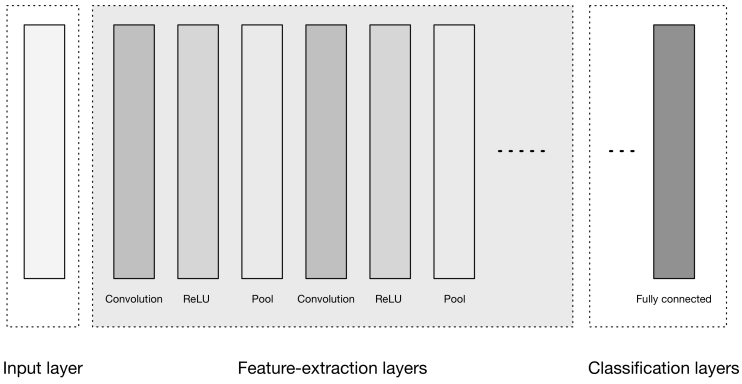

Figure 2.11: A general presentation of the architecture of CNNs [70]

Convolutional layers

The convolution layers, seen in Figure 2.11, detect features in an image through what is called the convolution operation. The layers consist of a set of learnable filters, also called *kernels*. Each of the filters is small in regards to input width and height, but are always the same size as the input in regards to depth. The convolution operation applies the filter to the input by sliding, or *convolving*, the filter across the width and height of the input, seen in Figure 2.12. At each position, the dot product, expressed in Equation 2.5, between the filter and the input is calculated. The output is a two-dimensional map called the *activation map*.

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1} \tag{2.5}$$

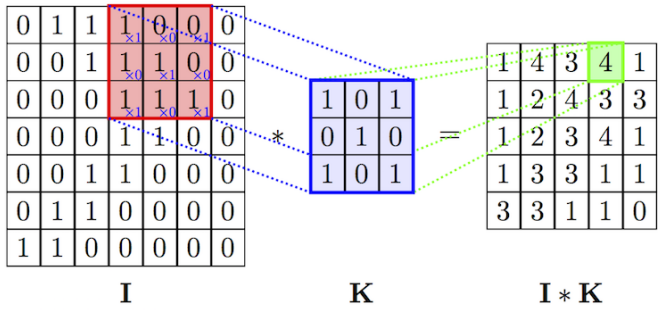

Figure 2.12: The convolution operation (applying a kernel filter) [8]

The network will learn filters that cause the node to activate when specific visual

features are seen, for instance, an edge. Deeper into the network we will see filters that become more global in term of the input and recognizes nonlinear combinations of features. An example of what the filters look like in a deep CNN can bee seen in Figure 2.13 where we see filters learned in the first convolutional layer in an eight-layer network [52].

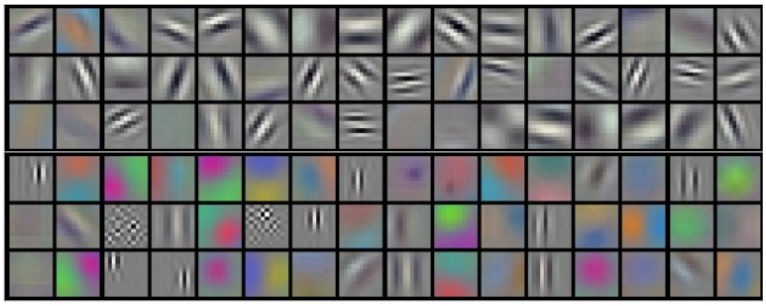

Figure 2.13: 96 learned filters in the first convolutional layer [52]

Pooling layers

The pooling layers are another essential part of the CNN. They help to reduce the number of parameters in the network and prevent overfitting, by reducing the size of the input data. The reduction is done by downsampling the input with different pooling functions. The most common operation is *max pooling*.

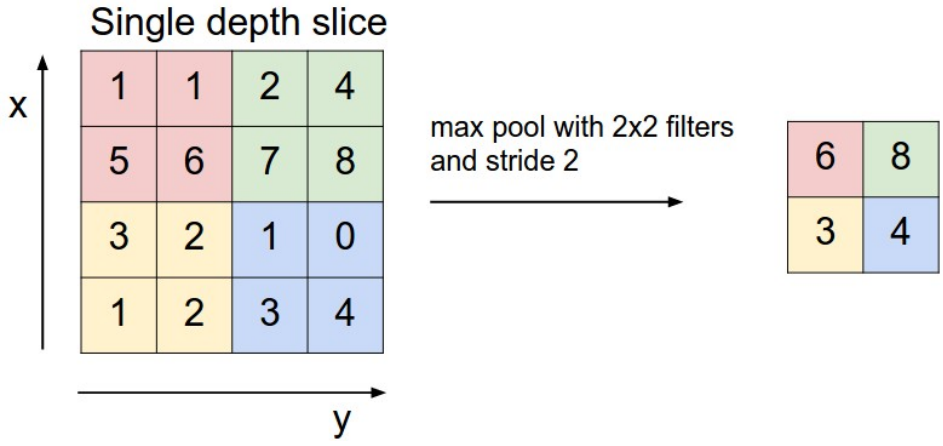

Figure 2.14: Example of max pooling operation [44]

Figure 2.14 shows a max pooling layer with a 2x2 filter size, and a stride of 2, meaning that it compares 2x2 pixels and that the sliding window moves 2 pixels

for each comparison. In practice this means that the 75% of the activations from the previous layer is discarded, thus reducing the number of parameters.

Upsampling

With the use of convolutional and pooling layers, the spatial resolution of the input is lowered. To make dense predictions that have the same resolution as the input image, a method to increase the resolution of the coarse heatmaps deep in the network is needed. This operation is called *upsampling*. The simplest way to do this is using resampling and interpolation, where the input image is rescaled to the desired size and pixel values are calculated using an interpolation method such as bilinear or nearest neighbor. *Transposed Convolution* refers to the operation of going in the opposite direction of a standard convolution. Transposed convolutional layers work the same way as standard convolutional layers, but with the forward and backward pass swapped around, thus training the filters to upsample an image during backpropagation. We can also refer to the operation as deconvolution [99]. However, since the operation is not a proper mathematical deconvolution, we will use the term transposed convolution in the rest of this thesis.

2.6 Object Recognition and Semantic Segmentation

2.6.1 Object recognition

Object recognition describes the science of finding and identifying objects within a two-dimensional image or video stream. Over multiple decades many approaches have been made to discover an efficient algorithm for detecting objects, but very few have been successful [94].

We can divide object recognition into several different categories. The most important of these are:

Model-based Object Recognition In model-based approaches, a 3D-model of the object being recognized is available for analysis. This way the algorithm can gain spatial knowledge about the object it is trying to identify, concerning the relationship between different parts and the shape of the object. An obvious issue with this approach is the difficulty of creating 3D-model examples that are generic enough for the algorithm to generalize well.

Appearance-based Object Recognition This approach uses a set of highly correlated training images of the desired object, mixed with a collection of random photos. By compressing this set of images using dimensionality reduction techniques, a lower dimensional Eigenspace is obtained. Object recognition is then performed by projecting new pictures onto the Eigenspace and measure the correspondence.

Feature-based Object Recognition Instead of evaluating an image as a whole, feature-based methods recognize objects based on different features, such as colors, patterns, and edges. By scanning an image for specific features and comparing the results with a database of various features, the algorithms can determine the probability of an image containing a particular object.

Pattern matching In pattern matching small parts of an image is matched against a database of image templates. By summing the difference in pixels between the image patch and the image template, and providing an acceptable threshold, the algorithm can decide if an object was recognized or not.

Artificial Neural Networks In later years Artificial Neural Networks (see section 2.4) have grown in popularity for object recognition in images. By passing labeled training examples through the network, a feature space is created within the network. After a substantial amount of training, the goal is that the network has defined its feature space in a way such that it can generalize beyond its initial training data.

2.6.2 Image segmentation

Image segmentation means to divide a picture into a set of visually coherent parts. Relatively good image segmentation can be achieved in many different ways by doing border detection, compare colors within a specific area and measure color gradients. Some typical algorithms for image segmentation are:

- Felzenszwalb's efficient graph based segmentation [22]
- Quickshift image segmentation [34]
- SLIC - K-Means based image segmentation [1]

An example of image segmentation using an efficient graph-based segmentation algorithm is shown in Figure 2.15.

Figure 2.15: Image segmentation using an Efficient Graph based algorithm

2.6.3 Semantic segmentation

As seen in Figure 2.15, there exists algorithms that can segment an image into coherent parts accurately. However, what if we want to understand the image?

In many use cases, for example, when a self-driving car is trying to understand the current city scene, it is not enough to only be able to segment an image. We need algorithms that can both create and classify the segments. These algorithms are called semantic segmentation algorithms.

In comparison to segmentation problems, semantic segmentation is a lot more difficult. The algorithms have to perform dense predictions on a pixel level, dividing an image into regions of a limited number of classes as seen in Figure 2.16.

Figure 2.16: Semantic segmentation [89]

Before the era of deep learning popular approaches used to solve semantic segmentation problems were algorithms based on concepts such as conditional random fields and hidden Markov models [61] [40].

Conditional Random Fields CRFs are statistical models that fall within the category of sequence modeling algorithms. The algorithms take the pixels and areas around the pixels it is trying to classify into account. For example, nearby

pixels are more likely to belong to the same class, pixels with similar color are more likely to belong to the same class, and the pixels above the pixels classified as "chair" are more likely to belong to the class "person" than the class "plane."

Hidden Markov Models A Hidden Markov Model (HMM) is a probabilistic temporal model in which a single discrete random variable describes the state of the process. The possible values of the variable are the possible states of the world.

Deep Convolutional Neural Networks In later years the use of deep learning networks has dominated the field of computer vision [78]. Deep Convolutional Networks can learn deep relationships between features in an image, thus helping them to perform very well in semantic segmentation tasks.

2.6.4 Evaluation metrics

To compare the different approaches to semantic segmentation, the need for evaluation metrics arise. We will now go through some of the standard metrics used for semantic segmentation results. We will use the notation from the confusion matrix, seen in Figure 2.17, for the equations.

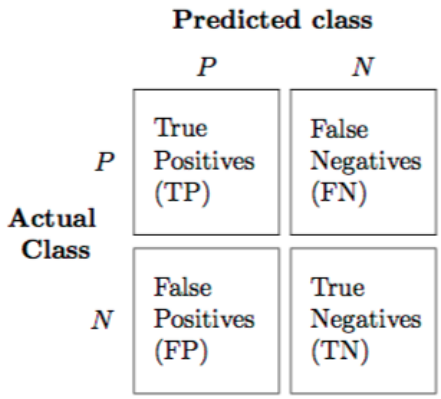

Figure 2.17: Confusion matrix

Accuracy Accuracy refers to the ration of correctly classified pixels over all the available pixels. The accuracy can be calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision The precision refers to the relation between true positives and all the pixels that are classified as positives, and is calculated as follows:

$$Precision = \frac{TP}{TP + FP}$$

Recall The recall is the relation between true positives and all the positive pixels and is calculated with the following formula:

$$Recall = \frac{TP}{TP + FN}$$

F_1 score The F_1 score (also known as F-measure, F-score or Sørensen-Dice coefficient) is the harmonic mean of precision and recall and is calculated as follows:

$$F_1 = \frac{2TP}{2TP + FN + FP}$$

Mean Intersection Over Union (mIOU) The *intersection over union* (IOU) and its mean value (mIOU), is the de-facto evaluation metric for semantic segmentation tasks used in a wide variety of research papers [78, 74, 10, 4, 38, 77, 100]. It can be seen as a similarity metric between two sets A and B. The general formula is:

$$J(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

With the use of our notation this becomes:

$$J = \frac{TP}{TP + FN + FP}$$

The overall perfomance is typically measured by taking the mean value of the mIOU for all classes except background.

2.7 Predicting large photos

CNN's train on image patches with sizes that are limited by the computing power available. Sometimes the size of the input images is much larger than the patches. If we want to predict a large photo with smaller patches, the predictions generate

a tiled output map in total, where each of the predictions is concatenated together. The concatenation makes it obvious where the patch borders are since the network lack context at the borders, thus making the predictions worse.

One way to fix this is to merge predictions done on overlapping patches [74, 42]. In addition to the overlap, we can rotate and mirror the large photos to make up the *dihedral group D4* [92] as we see in Figure 2.18. We can then predict all the rotated and mirrored photos with overlapping patches.

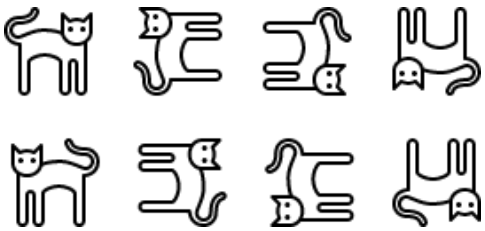

Figure 2.18: Dihedral group D4

The patch size might not be divisible with the aerial photo's size, which can cause the last patch in vertical and horizontal directions to not fit. We solve this by padding the image so that the patches divide the aerial photo correctly, as we see in Figure 2.19. We remove the padding when the predictions are merged to get the same size as the input image.

We can merge the predictions by using methods such as taking the geometric mean or with a window function to weight the predictions.

Figure 2.19: Padded image with overlapping patches in red. (a) The patches do not divide the image evenly and overshoot the image. (b) The image is padded so that the patches fit correctly.

This page intentionally left blank.

Chapter 3

Datasets in Computer Vision

Datasets play a critical role in computer vision research. There is a broad consensus that the current state-of-the-art, in convolutional neural networks, is due to large datasets and increased computational power [84].

This chapter presents the previous work related to the creation of datasets for computer vision algorithms. We can divide the datasets for object recognition into three main categories: object classification, object detection, and semantic segmentation.

3.1 Object classification

Object classification is the task of determining if an object exists within an image or not. Some examples of such datasets are the MNIST handwritten digits dataset [57], CIFAR-10 and CIFAR-100 [51] with 10 and 100 categories from small images (32x32) and ImageNet [39] with more than 15M classified images in more than 22k categories. ImageNet is unique, in the sense that it contains many more images than the other previous datasets. The size has made ImageNet the enabler for significant advances in object classification [52][28].

3.2 Object detection

Object detection is the task of not only stating that an object is present in the image but also give the precise location of the object. We often represent the position of the object with a bounding box. Early research focused on the localization of faces [32], but short after pedestrian detection also became popular with the Caltech Pedestrian Dataset [18] containing 350K labeled pedestrians with bounding boxes. A significant effort was made in the years 2005 to 2012 with the PASCAL VOC [21] datasets and challenges. Today the dataset contains 20 object categories in more than 11K images, with more than 27k bounding boxes. The PASCAL VOC challenges have become a standard way of evaluating the general performance of object detection algorithms. In 2011 a subset of ImageNet was used to create a dataset with 200 categories in 400K images [75]. The ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) used this dataset. The object detection dataset contains 620k bounding box annotations.

Microsoft introduced the Microsoft Common Objects in Context (MS COCO) in 2015 [60], a dataset that contains 328K images and 2.5 million labeled instances for classification, detection, and semantic segmentation. The dataset strives to include pictures that provide a non-iconic view of the labeled instances, where they appear in the background, partially occluded, amid the clutter and among multiple instances from different categories.

3.3 Semantic Segmentation

The task of semantic segmentation requires that we assign a class to each pixel in an image. We do not care to distinguish between the instances of the same type the same way we do in object detection. Many datasets with semantic labels have been

created in recent times [7, 101, 14]. PASCAL VOC added segmentation to their challenge in 2009 with 7K images and 3K segmentations. The CamToy dataset [7] was in 2008 the first dataset with semantic labels for video with 32 classes. MS COCO as described above also provides semantic labels for a subset of classes for their 328K images. The ADE20K dataset [101] with 20K annotated images from 2.6K classes, provide pictures from scenes, objects, parts of objects and parts of parts. The Cityscapes dataset [14] provides a dataset for complex urban street scenes from 50 cities with 30 classes. There are 5K images with high-quality labels and 20K with coarse labels.

3.4 Aerial Image Segmentation

Within aerial and satellite images, a few datasets have been created. The SpaceNet Challenge [90] is a contest provided by DigitalGlobe, Nvidia and CosmiQ Works. There have been three challenges in total: Two with building footprints, where the contestants are supposed to extract segmentations of buildings within images, and one with road extraction, where the contestants are supposed to retrieve line segments that correspond to roads within images. For the first contest, a dataset over Rio de Janeiro was provided, containing 4M building labels over an area of 2400 km^2 with a 50cm ground resolution for the images contributed by the Digital Globe WorldView-2 satellite. For the second contest, they expanded the dataset to multiple cities including Las Vegas, Paris, Shanghai and Khartoum with a ground resolution of 30cm from the Digital Globe WorldView-3 satellite. The total number of building footprints was over 970K. The roads network extraction challenge is currently ongoing at the time of writing. The images are over the same areas as in the second building contest and contain more than 8000km of roads. We note that they did not release the photos with an open source license and we can therefore only use them within the competition.

The Defence Science and Technology Laboratory (DSTL) Satellite Imagery Feature Detection competition [43] provide users with satellite images from the Digital Globe WorldView-3 satellite. The whole dataset contains 450 1x1km images, but they only label 25 images for training. They label the photos with ten different classes. They do not give the location of where the photos are taken, however, with inspection, it is clear that they are from a rural area. The images have a ground resolution of 0.3m. We can see a sample label and image in Figure 3.1. They do not give any information about how they create the labels, and they state that the dataset only can be used in the competition.

Figure 3.1: Ground trouth labels and arial image from the Dstl Satellite Imagery Feature Detection

The Inria Aerial Image Labeling dataset [62] is a dataset covering an area of 810 km^2 with building footprints from different regions of the world, where 405 km^2 of the data has labels used for training, and the rest of the data is unlabeled for testing. They focus on covering different cities, ranging from densely populated areas to alpine towns, as an experiment whether it is possible to learn aerial segmentation covering multiple cities and landscape types. The cities in the training set are Austin, Chicago, Kitsap County, Western Tyrol and Vienna. They gather the data from multiple sources of official data contributions, such as USGS's National Map service in the US, that is publicly available. The aerial images have a resolution of 0.3 m. As seen in Figure 3.2 the quality of the images and labels are high.

Figure 3.2: Ground trouth labels and arial image from the Inria Aerial Image Labeling dataset

DeepGlobe Satellite Image Understanding Challenge [17] is a competition where there are three different satellite image understanding tasks: Road extraction, building detection, and land cover classification. The competition is currently running at the time of writing, and there is little information about the dataset publicly, without registering for the contest, other than that they use DigitalGlobe satellite imagery.

Chapter 4

Semantic Segmentation Methods

In this chapter, we present related work for semantic segmentation methods. First, we look at regular image segmentation, before we dive deeper into semantic segmentation and how machine learning has played an essential role in the research field in recent years.

4.1 Image segmentation

People have discussed the importance of perceptual grouping of objects in the world around us for many decades. As early as in 1938, Wertheimer [93] presented several essential elements, such as similarity, proximity, and good continuation, that are important for semantic segmentation of objects.

Important works are later done by Shi and Malik [79] which tries to define coherence in images using low-level cues, such as brightness, color, texture, and smoothness of boundaries. By building a hierarchical tree structure based on these cues, their goal is to divide an image into different segments (see Figure 4.1).

Their approach is to look at the problem as a graph partitioning problem and to use unbiased, normalized cut criteria to create a segmentation of the image.

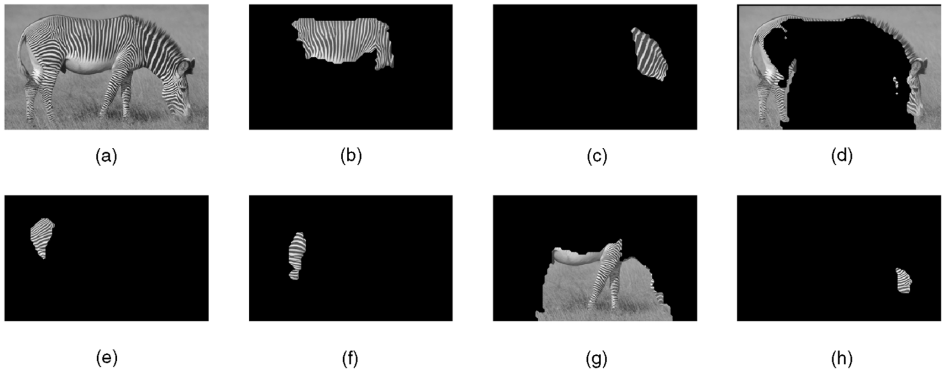

Figure 4.1: Segmentation of an image using normalized cuts [79]

Other attempts to automatically divide images into meaningful segments is the Simple Linear Iterative Clustering (SLIC) algorithm [1]. The SLIC algorithm divides the image into superpixels, where all pixels inside a superpixel has some coherence. While other segmentation algorithms have no limitations to their search space, SLIC limits the search space of each superpixel, thus making the search speed much faster.

One issue that occurs with these algorithms is that, even though they can divide an image into meaningful segments accurately, they are not able to classify the different segments. Meaning that the algorithms can segment out two houses in an aerial photograph successfully, but does not know what it is trying to find, or even that the two houses are belonging to the same class.

4.2 Semantic Segmentation with Convolutional Neural networks

In recent years, many semantic image segmentation algorithms have become increasingly dependent on convolutional neural networks (CNN's), because of their strong ability to yield hierarchies of features [77].

The use of convolutional operations in neural networks first appeared in the late 90's [56], but it was not until 2012, when Krizhevsky, Sutskever, and Hinton [52] won the ImageNet 2012 competition, that CNN's were acknowledged as one of the leading approaches for image computer vision [52].

Since then many advances have been made within the research area, such as the introduction of inception architectures and smaller convolutional operations [85, 81].

A problem that arises with deep convolutional neural networks is the degradation problem. Meaning that when the depth of the networks increases the accuracy of the predictions decreases, as seen in Figure 4.2.

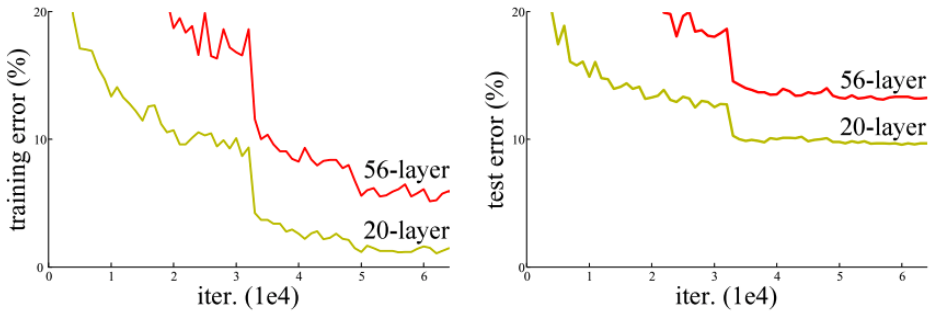

Figure 4.2: The problem of degradation in deep convolutional neural networks [95]

To solve the degradation problem, Wu, Zhong, and Liu [95] introduces residual blocks with shortcut connections, which sets the foundation for their residual neural network (ResNet). The concept behind ResNet is that any deeper network should not produce a higher training error than its shallower counterpart, only by seeing the layers that differentiate the two networks as identity mappings ($f(x) = x$). However, this does not seem to be the case in practice, suggesting that networks have problems learning identity mappings between multiple, non-linear layers. The authors propose a solution to this problem, by introducing two new concepts:

Residual mappings The paper hypothesizes that it is easier to optimize residual mapping, thus introducing the residual mapping function

$$F(x) := H(x) - x$$

where H(x) is the desired, underlying mapping function after two layers.

Shortcut connections Shortcut connections are connections that perform identity mappings (Figure 4.3) and are added to the output of the mapping function

$$F(x_n) = F(x_{n-1}) + x_{n-1}$$

where x_n is the input at layer n. The residual neural network has played an essential role in the recent development of very deep convolutional neural networks.

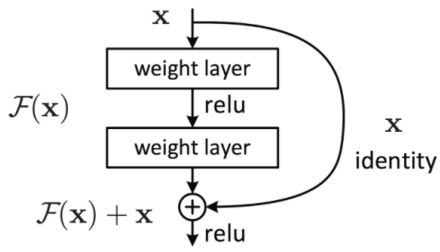

Figure 4.3: Residual block with a shortcut connection [95]

One of the first attempts to use convolutional neural networks for pixel-wise, semantic segmentation of images was made using a fully convolutional network (FCN) [78]. By adopting modern classification networks, such as AlexNet [52], VGG net [81] and GoogLeNet [85], into fully convolutional neural networks and using them for segmentation tasks, the network was able to exceed the state-of-the-art solutions. Shelhamer, Long, and Darrell [78] argues that the computational efficiency of the convolutional operation makes CNN's a natural choice for dense problems, such as semantic segmentation of images.

By analyzing the prediction results of the FCN, Zhao et al. [100] identifies several common issues for complex-scene parsing, and present their Pyramid Pooling module to solve them. The common problems identified are:

- **Mismatched Relationship:** The lack of ability to collect contextual information. The network should know that a car is seldom above water.

- **Confusion Categories:** Similar categories are often predicted within the same object.
- **Inconspicuous Classes:** Small objects can often be ignored if they are contained within another object.

We see examples of these issues in Figure 4.4.

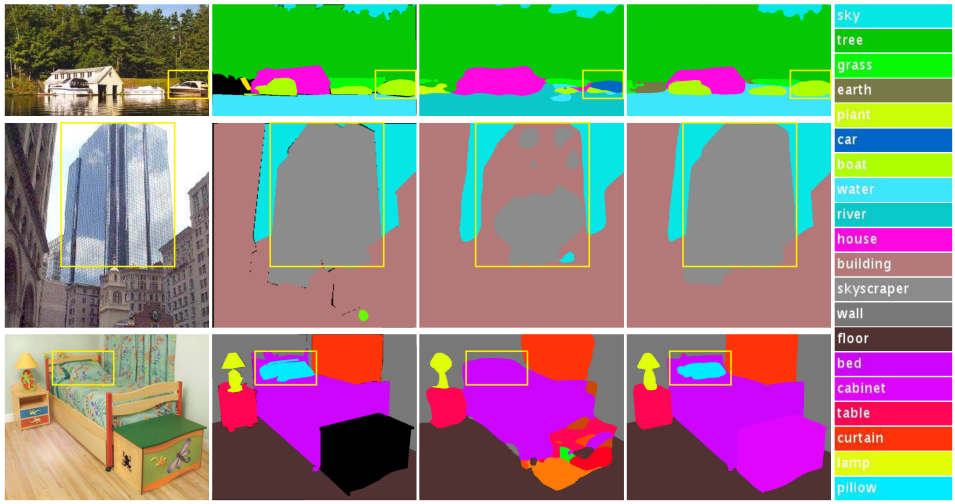

Figure 4.4: Common issues in for semantic segmentation seen in FCN [100]. The first row shows mismatched relationship when the network labels a boat as a car. The second row shows that the "building" is confused with "skyscraper." The third row shows inconspicuous classes where the pillow is classified as "bed" by FCN.

The Pyramid Pooling module is a hierarchical global prior that contains different scales and varying filter sizes among different sub-regions. By applying this module, Zhao et al. [100] was able to score a mIOU of 0.4168 and a pixel accuracy of 80.04 on the ImageNet Scene Parsing Challenge Zhao et al. [100].

There is, as mentioned earlier, a high consensus in the machine learning community, that to successfully train a very deep convolutional neural network a vast number of training examples is required. This is especially the case for semantic segmentation problems, because of their detailed, pixel-wise labeling. Ronneberger, Fischer, and Brox [74] present a training strategy that, rather than relying on a large set of training examples, uses large amounts of data augmentation and a different network structure. The augmentations consist of rotations and elastic transformations of the input images. This way they can successfully train a CNN into performing semantic segmentation of neuronal structures in electron microscopic stacks [1]. They call their

[1] A microscopic stack is a 3-dimensional stack of images taken by an electron microscope.

network a U-Net, because of its U-shaped structure, seen in Figure 4.5.

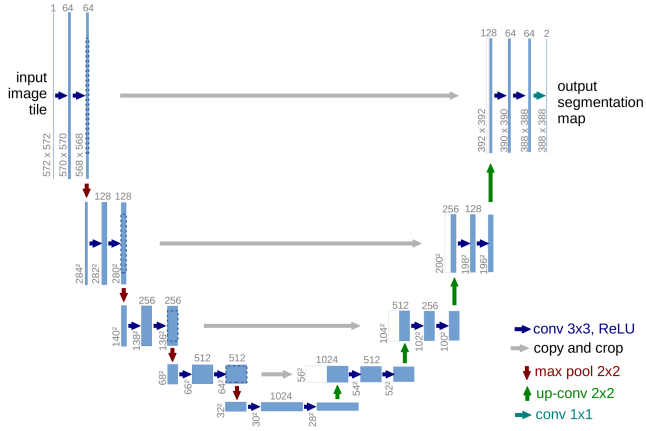

Figure 4.5: U-Net architecture [74]

The network is an extension of the Fully Convolutional Neural Network that has been modified to work with very few training examples and still yields precise, semantic segmentation. The networks U-shaped form comes from the large number of feature channels in the upsampling path of the network. The feature channels allow the network to propagate context information from higher resolution layers.

As presented earlier, an issue with very deep convolutional networks is the degradation problem caused by vanishing gradients. While one successful solution to this problem has been identity connections [95], Huang et al. [31] proposes a dense convolutional network, DenseNet, where they connect all layers in a dense-block to every other layer in a feed-forward fashion. [31] (see Figure 4.6).

Jegou et al. [38] presents a DenseNet modified to be fully convolutional, FC-DenseNet, for the task of semantic segmentation. The downsampling path in the network is the same as in the original DenseNet paper. In conventional fully convolutional networks the spatial dimension of the input is recovered by convolution, upsampling and skip connections. In FC-DenseNet, they replace the convolution operation with a dense block and an operation they call *transition up*. The transition up operation is made up by a transposed convolution on the previous feature map. A concatenation of the transition up module and the skip connection then forms a new input to the next dense block. The high-level architecture is similar to U-Net. The network achieves state-of-the-art results on the CamVid dataset with fewer trainable parameters than other networks and without pretraining on large datasets.

A new architecture recently introduces by Sabour, Frosst, and Hinton [76] is cap-

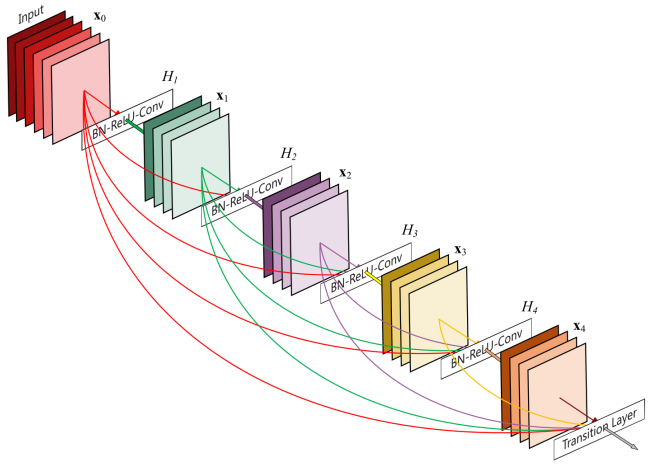

Figure 4.6: Dense connections between layers [31]

sule networks with dynamic routing. The multi-layer capsule network has shown state-of-the-art achievements on the MNIST dataset, especially when it comes to recognizing overlapping digits. LaLonde and Bagci [53] expands the use of capsule networks to the task of semantic segmentation by modifying the original dynamic routing algorithm to act locally when it is routing child capsules to the parent capsules and to share transformation matrices across capsules. They show that the SegCaps network can perform better than the U-Net while containing 95.4% fewer parameters.

This page intentionally left blank.

Chapter 5

Feature Extraction in Remote Sensing

As mentioned in section 2.2 the process of producing detailed and meaningful maps can both be resource intensive and time-consuming. For nearly four decades automatic feature extraction from remotely sensed data has been an active field of research [80]. Both semiautomatic and automatic methods have been developed, mainly focusing on extracting one specific feature. In this chapter, we present the most relevant research, concerning the extraction of buildings and roads.

5.1 Roads

Neuenschwander et al. [68] and Vosselman and Knecht [91] present attempts for semiautomatic tracking of roads. Their approaches take a starting point and a direction and use techniques such as snakes [48] and recursive Kalman filters to aid cartographers when manually detecting roads in aerial images.

Signal, Ssing, and No [80] uses Marr's theory for vision [63] with low-level processing for edge detection, mid-level processing for detection of road structures and high-level processing for road recognition, to automatically extract roads from the images. They claim that it is necessary to define a road model containing the definition of road parts, relationships between roads and geometric/photometric properties of a general road, to recognize it automatically. We see their results in Figure 5.1.

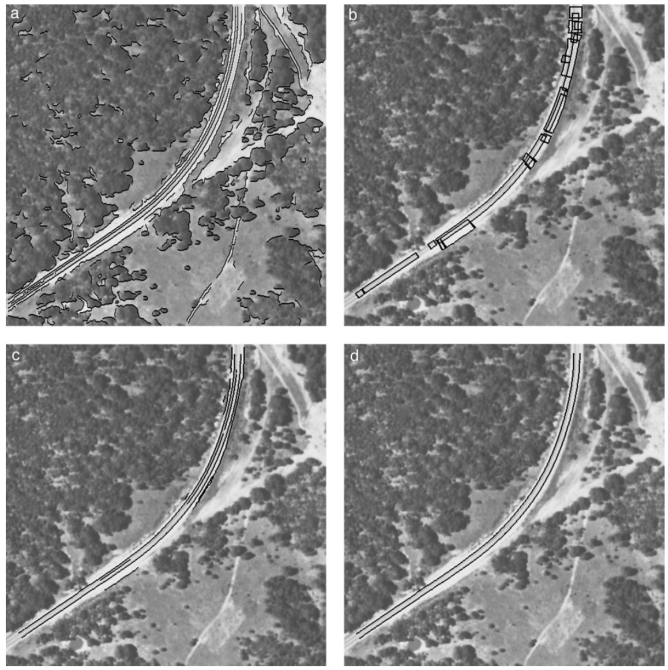

Figure 5.1: Road detection: (a) Detected edges, (b) Generatet antiparallel pairs, (c) Features formed after grouping and (d) Generated hypothesis of road segment [80]

Laptev et al. [54] make another attempt at road extraction. They propose an approach that can bridge different road segments, which have been separated by shadows or cars, using the heavily disturbed evidence in the aerial images.

To detect road segments Laptev et al. [54] uses a concept called Ribbon snakes, which is an extension of the snakes introduced by Kass, Witkin, and Terzopoulos [48], that takes the width of the element into account.

Satellite constellations, such as Quickbird, IKONOS, SPOT-5, and WorldView [33][35] have in recent years provided high-resolution aerial images. Higher resolution gives richer spatial information, which allows for more detailed analysis of the ground, thus making the task of road detection more relevant. However, it also reveals small objects such as cars and trees, which act as noise for the algorithms. Therefore, the algorithms need to become smarter.

Mokhtarzade and Zoej [66] present the first approach using artificial neural networks for road detection in aerial images, where they apply a simple feed-forward network with one hidden layer and a single output neuron. They train their network on high-resolution images taken by the satellite constellations IKONOS and Quick-Bird.

Cheng et al. [12] conclude that the use of convolutional neural networks in road detection is beneficial because of the complexity of the backgrounds, and the noise produced by the occlusions of trees and cars. They present a cascading convolutional neural network, CasNet (see Figure 5.2), consisting of two relatively small networks with eight layers in each. The network aims not only to classify the area covered by the roads but also detect the center line of the road, which is stated by many of being of equal importance [33][11][37][64].

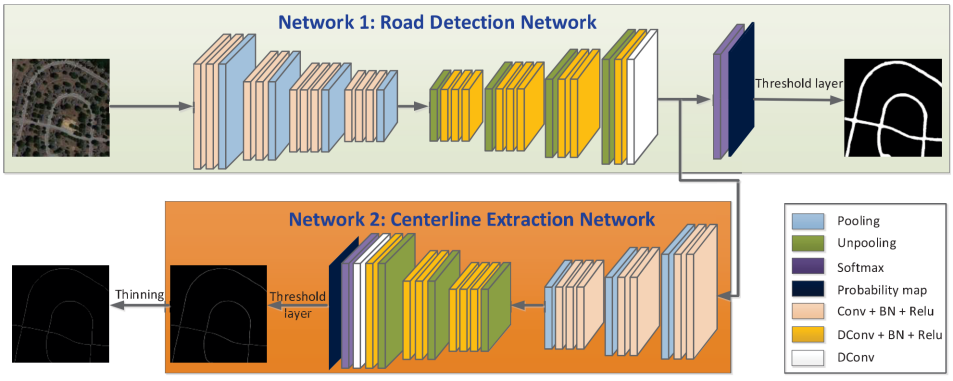

Figure 5.2: CasNet contains two convolution networks for road detection and center line extraction [12]

5.2 Buildings

Buildings are an essential part of the understanding of urban areas, and techniques to extract them have numerous applications in urban mapping, urban planning, and geoinformation engineering. Automatically extracting buildings from aerial have been a research field for many decades [97].

Kim and Muller [49] propose a method for automatic building extraction using a line detection approach that utilizes graph structures to map line relationships. Although being able to detect basic building formations, the presented approach is not able to generalize well over more advanced constructions.

Other fully automatic approaches have been feature-extraction using corner detection and variational level set evolution [15] and Conditional Random Fields [59]. Li et al. [59] points out that even though there have been many advances in the field of building detection in aerial images, the challenging task of developing generic algorithms that can generalize well remains unsolved. Using a higher-order CRF, they try to perform a pixel-wise classification of the images into four different classes, as seen in Figure 5.3. The blue node nodes represent pixels in the image, purple dots represent segments formed by the pixels, and the last node represents the four classes: White (rooftops), black (shadows), green (vegetation) and gray (unknown). Using their method, they achieve a mean F1-score of 0.786 on six datasets.

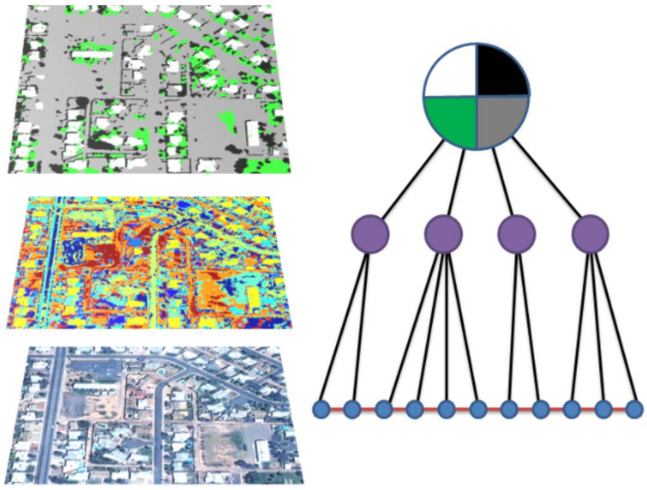

Figure 5.3: High-order conditional random field classifying aerial images into four classes. [59]

Yuan [97] presents a solution for building extraction similar to the solution pre-

sented in this thesis. By using different GIS databases, they create a training dataset for building extraction and uses a convolutional neural network to make building predictions. Their network is built up by seven convolutional layers.

Bischke et al. [6] criticize earlier deep learning approaches because of their inability to preserve segmentation boundaries between the different classes. They address the problem by introducing a new cascaded multi-task loss

$$L_{total}(x;\theta) = \sum_{i=1}^{T} \lambda_i L_i(x_i;\theta)$$

where T is the number of tasks, L_i is the corresponding loss function to be minimized by the optimizer with regards to the network parameters and λ_i is the weight for each loss function. The goal of this loss function is to incorporate two different terms when evaluating the performance of the network. Besides the regular semantic term, the researchers also use a geometric term indicating the distance from a pixel to a building boundary. This way they can bias the network to learn per pixel information about the location of the boundary and capture implicitly geometric properties. We can see an example of how they calculate the distance in Figure 5.4.

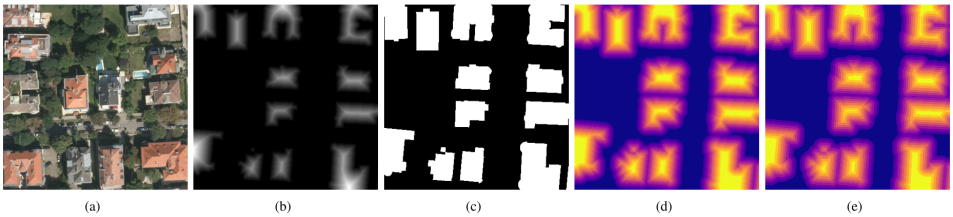

Figure 5.4: Training example with (a) RGB image, (b) Distance transform, (c) Segmentation mask, (d) Truncated distance mask and (e) Truncated and quantized distance mask [6]

Xu et al. [96] uses deep residual neural networks combined with a guided filter to perform building extraction. By first training the network using high-resolution aerial imagery, and then preprocessing the building predictions they achieve state-of-the-art segmentation results as we see in Table 5.1.

Dataset	OA	prc. B	F1 B	Recall B	prc. C	F1 C	Recall C
Postdam	0.9691	0.9634	0.9390	0.9158	0.9709	0.9793	0.9878
Vaihingen	0.9771	0.9621	0.9515	0.9412	0.9816	0.9850	0.9883

Table 5.1: Precision measurements for the cities Postdam and Vaihingen. Here B stands for buildings, and C represents clutter, and OA represents the overall accuracy.

5.3 Maps

Iglovikov, Mushinskiy, and Osin [35] presents an approach for semantic segmentation of aerial imagery in the DSTL Satellite Imagery Feature Detection challenge [43]. By using a U-Net, as we see in Figure 5.5, with augmentation techniques and border enhancement, they get an overall third place in the challenge with accuracies compared with the first and second place.

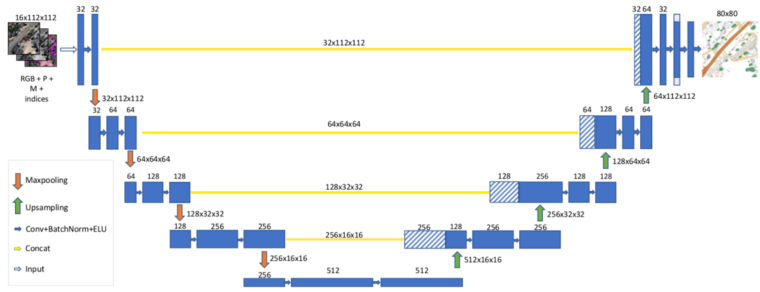

Figure 5.5: U-Net architecture with five downsampling layers and five upsampling layers [35]

The network uses an exponential linear unit (ELU) [13] as the activation function, the evaluation metric is the Jaccard Index, and the loss function is a smooth Jaccard loss function. They highlight the issue that prediction quality decreases, as seen in Figure 5.6, as we move away from the center of the tiles. They argue that the reason for this is that the network is structured so that the number of ways to get from any input pixel to the center part of the output is much higher than to get to the edges.

A solution to this problem is to make predictions on overlaying patches and crop out the edges. The authors present an alternative solution where they add a cropping layer to the output layers of the network. Later [42], they also describe a way of predicting, where they average multiple rotations of predictions with the geometric mean. They also propose to use the dihedral group D4 for the predictions that are

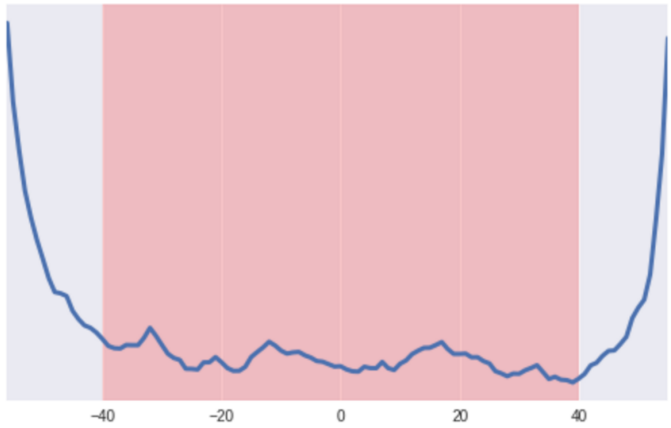

Figure 5.6: Accuracy of predictions decrease when mooving away from the center [35]

averaged, using all eight possible rotations on the input image.

This page intentionally left blank.

Part II

Implementation and Assessment

This page intentionally left blank.

Chapter 6

Methodology

In this chapter, we present our methods. First, we show our approach for creating the dataset, including what parts of the FKB dataset we use and how we match NIB and FKB data to create valid training examples. Then we review the chosen networks and present our implementations of their architectures.

6.1 Defining the properties our map

Per the mapping process defined in section 2.1, We aim to create a map to represent urban areas, consisting of four spatial features: Roads, Buildings, Water and Vegetation. The map should be able to aid users to navigate urban scenes. We represent these features as polygons. The input data defines the scale, map projection, and spatial reference system. We choose not to annotate the map, as it is outside the scope of this thesis.

6.2 Creating the Dataset

As stated earlier, one of the goals of this thesis is to explore the possibility of using automatically created segmentation labels based on publicly available spatial data to train CNN's. The dataset used to generate the labels is Felles Kartdatabase (FKB) downloaded from GeoNorge [27]. We use high-quality aerial images from the service Norge i Bilder (NiB) [26] with a ground resolution of 0.2 m to serve as example images. These are made available by The Norwegian Public Roads Administration, The Norwegian Institute of Bioeconomy Research and The Norwegian Mapping Authority. FKB and the aerial photos from NiB are not free to use for the public, but anyone can buy them. The data is available free-of-charge for academic use.

We will now explain the overall process used to create the datasets automatically. We have chosen to focus on some of the largest cities in Norway for the dataset, while at the same time making sure that the cities are geographically spread out, to assure proper coverage of different environments.

6.2.1 Collecting and storing the spatial data

Aerial images from the cities of Oslo, Bergen, Trondheim, Stavanger, Bodø, and Tromsø are downloaded from NiB in GeoTIFF format. These are aerial images uploaded by *Norway Digital* partners [45], which are large-scale vendors using or creating spatial data in Norway. The photos are taken at different times and by different vendors and uploaded project wise, making the seasonal conditions and image quality slightly different across local regions. We see an example of the quality difference in Figure 6.1.

The images are orthophotos, that is, they have been geometrically corrected in such way that the scale is uniform. However not all the photos are true orthophotos

Figure 6.1: Differences in image quality and seasonal condition between two projects

in the sense that vertical features, such as buildings, are reprojected. This causes some of the images to contain tall oblique features where the footprint is at the correct coordinates and the top of the feature somewhat leaning to one side, such as in Figure 6.2. The labels will always correspond to the footprint of the features, and from a mapping perspective, this is the correct way to label a feature. Since NiB does not deliver all the aerial images are delivered as true orthophotos, we choose to have both types in the dataset. The reason for this being that it might help the network generalize better, as the real world not only consists of true orthophotos. We tile the images into 512 x 512 px images corresponding to an area of $10485.76m^2$ per image.

We download corresponding spatial vector data from FKB from GeoNorge for all the cities. Since we are focusing on cities, we get the highest quality FKB data, FKB-A, as described in section 2.3 for our dataset. This is then uploaded to a PostGIS database (see section A.2). We consider all the 26 datasets in FKB, with multiple classes, when selecting what data to upload. Having them all in the dataset would create a very challenging segmentation problem. Therefore, we limit the problem to four main classes (excluding background) from five of the datasets. We see the classes, and the belonging datasets in Table 6.1.

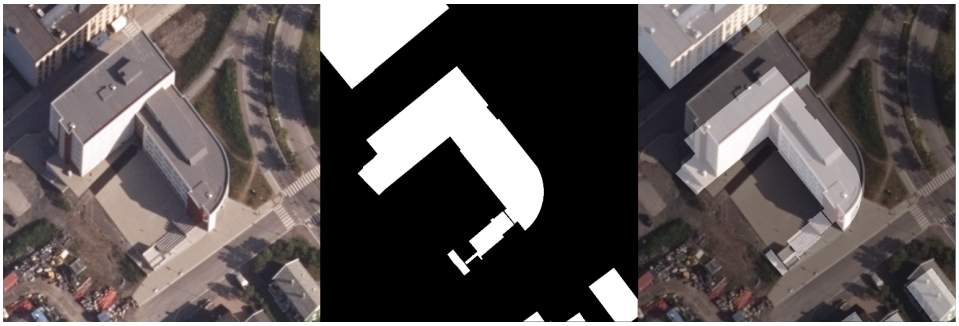

Figure 6.2: Leaning building due to the images not being true orthophotos.

Class	Datasets	Original Name
Background		
Roads	FKB Road	FKB Veg
Water	FKB Water	FKB Vann
Grass	FKB Landcover FKB Area Usage	FKB AR5 FKB arealbruk
Buildings	FKB Buildings	FKB Bygg

Table 6.1: Classes and belonging datasets

Using the defined classes, we build four binary datasets for individual segmentation of the different classes and one multiclass dataset that combines the classes from the binary datasets. We illustrate the overall process for creating the dataset in Figure 6.3.

6.2.2 Binary datasets

We represent each class in a separate binary dataset, where the area the feature is located (foreground) has a value of one, and the rest (background) has a value of zero. We create a dataset for each of the classes described Table 6.1.

Buildings The binary buildings dataset is, as seen in Table 6.1, generated using the dataset FKB Buildings. The dataset consists of three different object types, as seen in Table 6.2. Out of the three "Bygning" and "Takoverbygg" is selected for generating the dataset. Furthermore, we also consider FKB Building facilities (FKB BygnAnlegg), but after analyzing the 19 different object types, we select none of them. We consider three object types: "Silo," "Tank" and "Fundament."

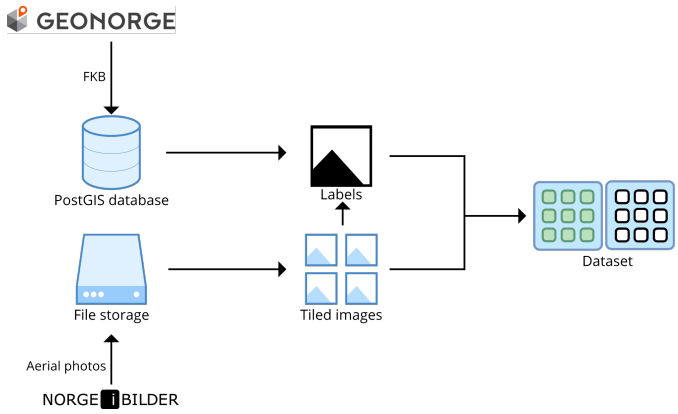

Figure 6.3: The overall dataset creation process. FKB vector-data is downloaded from GeoNorge. Aerial images are downloaded from NiB. The aerial images are tiled and used as bounding boxes when we query the PostGIS database. Both label and original query image is saved in the dataset.

However, all of them contain too many objects that could pollute the dataset, such as statues, heat tanks, and other storage devices. We, therefore, omit them from the dataset generator.

ObjType	Description
Bygning	Building
AnnenBygning	Other Building
Takoverbygg	Roof over building

Table 6.2: Object Types in the FKB Buildings dataset

Many of the FKB datasets contain an attribute called *medium*, which describes the location of different spatial objects on the surface of the earth. The possible values are displayed in Table 6.3. In the part of the FKB Buildings dataset that we are working with, only two of the mediums appear. These are "T," and "U." Out of these, we remove "U" from the generator. We remove "U" because it represents building structures that are beneath the ground. Including these labels in the dataset would, therefore, pollute the results, since the learning algorithm would learn to label, e.g., segments of parking lots and streets as buildings.

Roads In the binary road dataset, we label the area that is covered by roads, as a road. We could have worked with the center lines such as in [12], but since FKB Road contains accurate road areas and we are focusing on semantic segmentation, the area came out as the most correct to use. We base the dataset on FKB Road,

Description	Code
Always in water	V
In building / building construction	B
In air	L
On glacier	I
On the sea floor	S
On the terrain / at ground level (default)	T
On the water surface	O
Occasionally under water	D
Under Glacier	J
Under the sea floor	W
Under the terrain	U
Unknown	X

Table 6.3: Medium types in the FKB Datasets [46]

Figure 6.4: Label for building structure hidden under ground

which consists of five different object types (see Table 6.4).

ObjType	Description
GangSykkelVeg	Cycling and walking Road
Parkeringsområde	Parking Area
Trafikkøy	Traffic Island
Traktorveg	Tractor road
Veg	Road

Table 6.4: Object Types in the FKB Road dataset

Out of these five, we use "Road," "Tractor road," and "Cycling and walking road" in the generated dataset. Also, we consider the object type "Trail" as a potential candidate, but because of the amount of occlusion caused by trees over large parts of the dataset, we omit it from the generator. The FKB Roads dataset also contain the *medium* attribute. In the FKB Roads dataset, only the mediums "B," "J," "L," "T," and "U" are present. Out of these, we omit "J," "U," and "B" from the generator, to remove the labels for underground tunnels.

Vegetation The binary vegetation dataset is generated using two of the FKB datasets: FKB Landcover and FKB Area Usage. The FKB Landcover dataset consists of 11 area types (as seen in Table 6.5).

ArType	Name	Description
21	Fulldyrka jord	Cultivated land
22	Overflatedyrka jord	Surface Cultivated land
23	Innmarksbeite	Cultivated pastures
30	Skog	Forest
60	Myr	Swamp
50	Åpen fastmark	Open solid ground
80	Vann	Water
70	Bre	Glacier
11	Bebygd	Inhabited
12	Samferdsel	Transport and Communications
99	Ikke kartlagt	Not mapped

Table 6.5: Areal types in the FKB AR5 dataset

Out of the area types in Table 6.5, we select 21, 22, 23, 30, 50 and 60 for the dataset. Furthermore, we also use some parts of the FKB Area Usage dataset, as it includes areas such as parks, football fields, alpine slopes and more (see Table 6.6).

ObjType	Description
Alpinbakke	Alpine slope
Anleggsområde	Construction Site
Campingplass	Caravan park
Fyllplass	Landfill
Golfbane	Golf course
Gravplass	Cemetery
Grustak	Rock dump
Industriområde	Industrial Area
Lekeplass	Playground
Park	Park
Skytebane	Shooting range
SportIdressaPlass	Outdoor sport fields
Steinbrudd	Quarry
Steintipp	Gravel tip

Table 6.6: Object Types in the FKB Area usage dataset

Out of all the object types in Table 6.6, we select "Alpine slope," "Golf course," "Cemetery," "Park" and "Shooting range" to supply a higher degree of detail to the vegetation dataset. These are selected because they contain a stable amount of

vegetation, while for example "Playground" and "Caravan park" often can contain asphalt and features that look like buildings.

Water We use the FKB Water dataset to generate the water dataset. FKB Water does not have any object types associated with it, but the whole dataset is considered relevant and is therefore used in its whole to represent the water dataset. It does, however, contain the medium attribute, wherein the dataset we use polygons with medium class T and U. We remove the U class from the generator, as it represents rivers and other water inventories beneath the ground.

6.2.3 Multiclass dataset

In the multiclass dataset, we represent all the classes described in subsection 6.2.2 in the same dataset. Each raster image consists of multiple labels with color values ranging from 0 to 4. The connection between the classes and the color values are displayed in Table 6.7.

Class	Color Value
Background	0
Water	1
Vegetation	2
Roads	3
Buildings	4

Table 6.7: Color values for the different classes

The color value does not only represent a class but also works as a ranking of the importance of the different classes. What this means is that the class with the highest color value will be displayed in the raster image if there is a conflict between the classes. For example, if a pixel is within both vegetation and a building feature, it will be labeled as a building.

6.2.4 Querying the database

Since the tiled, aerial images are georeferenced, it is possible to use the extents of the image to query the database for the geospatial features that lie within the image bounding box. We can, therefore, create raster labels that contain the features corresponding to an image by querying the database with its bounding box. We only use the labels where at least one feature is visible in the related image.

The five datasets are generated using the queries described in Appendix B. We filter out the different geospatial features contained in the database using an attribute called "color." If the "color" attribute of a feature matches one of the defined class ranks seen in Table 6.7 we include the feature in the dataset, else we omit the feature from the dataset.

A problem when querying the PostGIS database for raster labels using a bounding box, is that they might not get the same extent as the bounding box, as explained in Figure 6.5. If the extent is different, the labels are useless since it is impossible to align them correctly with the input image. The reason for this is that PostGIS's *ST_AsRaster* function generate rasters with the extent of the geometries from the query and not the query bounding polygon [73]. To make this work, we would need all of the area bounded by the input image to be covered by geometries. This might be the case when using data such as land cover, but it is not the case when using other data types such as roads and buildings. We solve this by forcing Postgis to always create an empty background layer in the raster with the exact bounds as the input image. The geospatial properties of each pixel then cause the input and output rasters to align correctly.

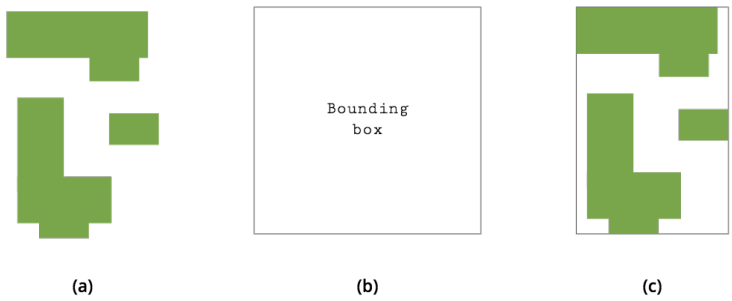

Figure 6.5: PostGIS raster rendering problem where (a) is a visualization of the geospatial features in the database, (b) is a bounding box and (c) is the resulting raster generated by PostGIS

6.2.5 Generalization Test Areas

To assert the generalization capabilities of the algorithms for new areas, we download aerial photos and labels for the city of Drammen and Grorudalen, a valley in

eastern Oslo. The aerial photos from Grorudalen come from the same aerial photo project in NiB as the Oslo dataset and therefore shares the same characteristics. The aerial photos from Drammen come from a new NiB project and have entirely new characteristics compared to the other aerial images.

The Grorudalen images let us assert the generalization capabilities for new areas within the same aerial photo project, whereas Drammen let ut assert how a different NiB project affects the generalization of the algorithms.

6.3 Network architectures

When choosing the network architecture, we consider a few different factors: General performance, performance without pre-trained datasets, model complexity and model availability.

We choose to go forward with two networks: U-Net and FC-DenseNet. Both networks have good performance without much data or pretraining, and they have open source implementation available in Keras (see section A.5).

All of the code is made available in our GitHub repository [1].

6.3.1 Review of U-Net

For the task of semantic segmentation of aerial images, we have seen increasing use of U-Nets, or slightly modified U-NETs, in recent competitions [41, 86]. The reason for this is the excellent baseline performance of end-to-end trained models with little data, relative simple architecture, and scalability. These properties make it easy to experiment with, thus making it a sound choice to start with for our thesis.

U-Net is designed as an autoencoder. An autoencoder learns to compress input data into a shortcode and then decompresses the code to create a close representation of the input. The compression is done in the *encoder* part, and the decompression is done in the *decoder* part. The U shape of the network comes from the connection between the encoder and the decoder part of the network. The encoding and decoding part of the network is referred to as the contracting and expanding path respectively.

The contracting path follows a typical architecture for a CNN. It consists of multiple layers of two convolutional operations, ReLU activation, and max pooling. At each downsampling step, the number of feature channels is doubled. Every step in

[1] https://github.com/valdemarrolfsen/master-thesis-code

the expanding path consists of up-convolutions that halve the number of feature channels, a concatenation with the corresponding layer in the contracting path and two convolutional operations.

The two convolutional operations after the concatenation help the network learn a more precise output based on the high-resolution features in the contracting path and the upsampled output.

The authors focus on the ability to learn accurate segmentations without the need for large datasets since the medical image dataset they work with only contains 30 images. Data augmentation solves the problem. The augmentation consists of shifting and rotating the images at random. The augmentation makes the network robust in regards to deformations and variations in the input images and also has the benefit of making the dataset larger.

6.3.2 Implementing U-Net

When implementing U-Net, we experiment with several configurations based on the networks presented in the literature combined with our modifications. We base the final configuration on the evaluation of the original architecture, a version with batch normalization and dropout, a deeper version with an additional down block with batch normalization and dropout and a deeper version with transposed convolution instead of upsampling.

The original architecture

We base our first implementation, on the architecture presented by Iglovikov, Mushinskiy, and Osin [35], which we see in Figure 5.5. The network has a depth of four, where the blocks in the downstream path have the architecture of a standard convolutional block with Maxpool and batch normalization layers (see Figure 6.6), except for the first block, where the Maxpool layer is replaced with an input layer.

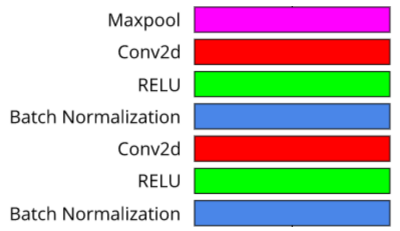

Figure 6.6: A standard convolutional block with Maxpool.

The blocks in the upsampling path have a similar structure to the blocks in the downsampling path. However, the Maxpool layers are replaced with Upsampling layers, as seen in Figure 6.7, to map the generated feature map back to the original size of the image (see section 2.5).

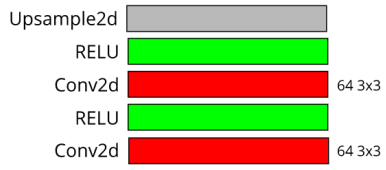

Figure 6.7: A block from the upsampling path in the U-Net architecture.

In the last block in the upsampling path, the Upsample2d layer is replaced with a Softmax/Sigmoid activation function to map the output values to probabilities.

Implementation with dropout

The next implementation is very similar to the original architecture, but a dropout layer is added to each block in the upsampling path to reduce overfitting.

A deeper version

In the third implementation, we try to increase the depth of the network by one level, to create an even more complex feature space.

Transposed convolution instead of upsampling

The regular unpooling layer used in the implementation on U-Net (Upsample2d), multiplies the rows and the columns of the past layer with a specified factor, which in our case is 2, and then interpolates the pixels to smoothen the result. Although upsampling layers are useful at mapping the learned feature space back to the original input size, we lose the advantage of convolutional filters, as explained in section 2.5.

We implement a version of U-Net where transposed convolutions replace the Upsample2d layers.

The different architectures are visualized in Figure 6.8.

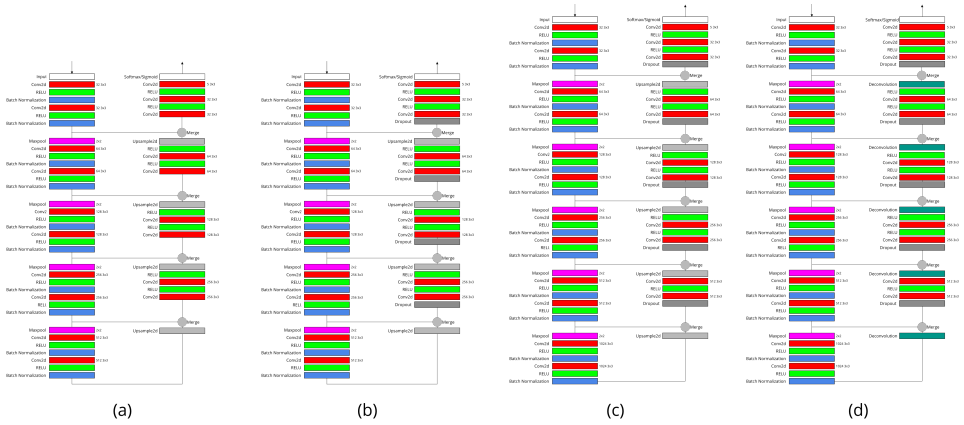

Figure 6.8: Our unet architectures.

6.3.3 Review of Fully Convolutional DenseNet

Fully Convolutional DenseNet (FC-DenseNet) can provide very accurate semantic segmentations for different tasks without pretraining. The high-level architectural design extends U-Net by adding fully Convolutional *dense blocks*, that share feature maps in a feed-forward fashion. Skip connections between the dense blocks are added, enabling the network to share features between the dense blocks.

To get the intuition of how DenseNets compare to traditional CNN's, we need to look at the way the layers in the networks differ. In a traditional CNN, we compute the output of a layer by applying a non-linear transformation of the input. This transformation is often a convolution followed by a non-linear activation such as ReLU. If we let x_n denote the output at layer n and F the non-linear transformation function, we write the standard transformation mathematically as:

$$x_n = F_n(x_{n-1})$$

The next important inspiration for DenseNets come from ResNets and their residual blocks [95], where the *shortcut connections* map the input of the layer to the output of the transformation F to deal with the *degradation problem*. We then expand the transformation function to:

$$x_n = F_n(x_{n-1}) + x_{n-1}$$

DenseNets take this idea even further and create concatenations between the output

from all the previous layers in a dense block, thus expanding the transformation function even further into

$$x_n = F_n([x_{n-1}, x_{n-2}, ..., x_0])$$

where ... is the concatenation operation of the layers.

The authors present three different architectures in the paper: FC-DenseNet56, FC-DenseNet67, and FC-DenseNet103. The numbers refer to the number of layers in the network, and all the architectures consist of five dense blocks (five down, one at the bottom and five up) with different configurations. The architectures are different in two aspects: The number of layers in each dense block and the *growth rate* of the dense blocks. The growth rate describes the number of feature maps, K, at each layer. The growth rate in the implemented versions is either 12 or 16. The feature maps grow linear with each layer, meaning that layer L will have $K \times L$ feature maps.

The network with 56 layers has four convolutional blocks (convBlocks) per dense block and a growth rate of 12. The network with 67 layers has five convBlocks per dense block and a growth rate of 16. The network with 103 layers has 4, 5, 6, 10, 12, 15 convBlocks per dense block (including the bottom one) respectively.

6.3.4 Implementing FC-DenseNet

We implement all three standard architectures as described in the paper [31]. We base the implementations on the same structure displayed in Figure 6.9. From the figure, we see that the general structure of the network is very similar to the U-Net with a U-shape and skip connections at every layer.

We present the actual implementation used in this thesis in Table 6.8. We base our implementations on the naive, memory inefficiencies version of the network presented by Huang et al. [31] because Tensorflow (section A.4) did not support shared memory allocation, until recently. Meaning that the model size and, also the maximum possible batch size, is limited by the GPU memory [72].

6.4 Training the Networks

We will now explain how we train the networks and the hardware we use. All networks are trained using a dataset that is split into training, test and validation

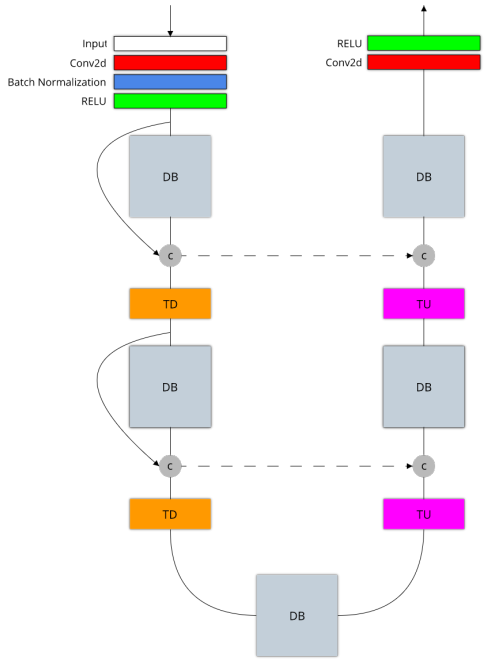

Figure 6.9: The FC-DenseNet architecture

FC-DenseNet-56		FC-Densenet-67		FC-Densenet-103
3x3 Convolution		3x3 Convolution		3x3 Convolution
DB (4 layers) + TD		DB (5 layers) + TD		DB (4 layers) + TD
DB (4 layers) + TD		DB (5 layers) + TD		DB (5 layers) + TD
DB (4 layers) + TD		DB (5 layers) + TD		DB (7 layers) + TD
DB (4 layers) + TD		DB (5 layers) + TD		DB (10 layers) + TD
DB (4 layers) + TD		DB (5 layers) + TD		DB (12 layers) + TD
DB (4 layers)		DB (5 layers)		DB (15 layers)
TU + DB (4 layers)		TU + DB (5 layers)		TU + DB (12 layers)
TU + DB (4 layers)		TU + DB (5 layers)		TU + DB (10 layers)
TU + DB (4 layers)		TU + DB (5 layers)		TU + DB (7 layers)
TU + DB (4 layers)		TU + DB (5 layers)		TU + DB (5 layers)
TU + DB (4 layers)		TU + DB (5 layers)		TU + DB (4 layers)
1x1 Convolution		1x1 Convolution		1x1 Convolution
Softmax		Softmax		Softmax

Table 6.8: Architecture of different FC-DenseNet implementations

segments with sizes equivalent to that of 70%, 20%, and 10% respectively. When choosing the size of the different segments, we take mainly two competing concerns into account. First, it is crucial that the network is presented with a rich selection of training examples so that it can see and learn all the commonalities that define

the different classes in the dataset. Secondly, enough examples should be left out to assure that the network learns universal concepts.

To evaluate the performance during training, we use mIOU and F_1 scores. During training, we predict and evaluate the entire validation set after each epoch. After training, we do the same procedure for the test set. Lastly, we assert the generalization performance of the trained algorithms by predicting the aerial photos of Drammen and Grorudalen.

We train the networks on a Linux server with two Nvidia GTX 1080 GPU's with 8GB of memory each, i7-7700K CPU and 32 GB of RAM.

6.5 Predictions on the test data

To measure the performance of the networks, we measure mIOU and F1 score of the predictions on the test dataset. The test dataset has been held out of the training process so that the networks have never seen the images.

When predicting on larger areas, we try two different approaches. The first approach, later referred to as the regular approach is to divide the large image into image tiles with the same size as the network has trained on. Then the networks predict each of tiles individually before they are concatenated back together. The second approach later referred to as the merged D4 (mD4) patching approach, is where we use the technique presented in section 2.7.

Chapter 7

Proposed Dataset

In this chapter, we present the dataset we have created through examples and relevant statistics. We also present known errors and show the full potential of the dataset if we use aerial images and FKB data from all of Norway in the process. As described in section 6.2, we focus on five cities in our work; Oslo, Bergen, Trondheim, Bodø, Tromsø and Stavanger.

7.1 Overview

The tiling process explained in section 6.2, produce 49,387 images covering an area of 517.8km^2, with distribution among the cities as seen in Table 7.1.

City	Tiles	Area (km^2)
Bergen	13,260	139.0
Bodø	1,302	13.7
Oslo	18,952	198.7
Stavanger	3,417	35.8
Tromsø	4,008	42.0
Trondheim	8,448	88.6
Total	49,387	517.8

Table 7.1: Dataset statistics

Looking at the distribution, Oslo and Bergen represent 65.21% of the dataset with a total area of 337,7 km^2. We can divide Norway into two parts: The northern part with a total area of 112,951 km^2 and 484 647 inhabitants, and the southern part with a total area 210,802 km^2 and 4 768 525 inhabitants [88, 3]. Out of the six cities, we locate two of them in the northern part of Norway, while we locate the rest of the cities in the southern part of Norway. The two cities in the northern part, Tromsø and Bodø, represent 10.75% of the dataset with a total area of 55.7 km^2, and the remaining four cities represent 89,25 of the dataset with a total area of 462,1 km^2.

After matching the tiled images with the features in the geospatial database, the number of tiles included in each dataset is determined based on how many of the tiles that contain the respective feature of the dataset. The generated training datasets contain images from all of the cities randomized between the training, validation and test dataset to have samples that represent all the cities. The number of training, validation and test examples for each of the datasets are presented in Table 7.2.

Dataset	Training images	Val images	Test images
Buildings	17,821	5,183	2,570
Roads	19,395	5,611	2,830
Vegetation	20,433	5,788	2,952
Water	9,329	2,692	1,352
Multiclass	31,273	8,989	4,525

Table 7.2: Number of training, validation and test images in the datasets

7.2 Examples and Statistics

7.2.1 Binary datasets

In the binary datasets, the pixels in the raster images either have a value of zero or one, indicating whether the pixel belongs to an object or background. We present samples from the four datasets in Figure 7.1. Because of the spatial nature of the different object classes, the amount of area covered by each class will vary to some extent. Looking at Table 7.3 the area covered by road labels is considerably smaller than the area covered by the other class labels, while the area covered by vegetation labels is considerably larger.

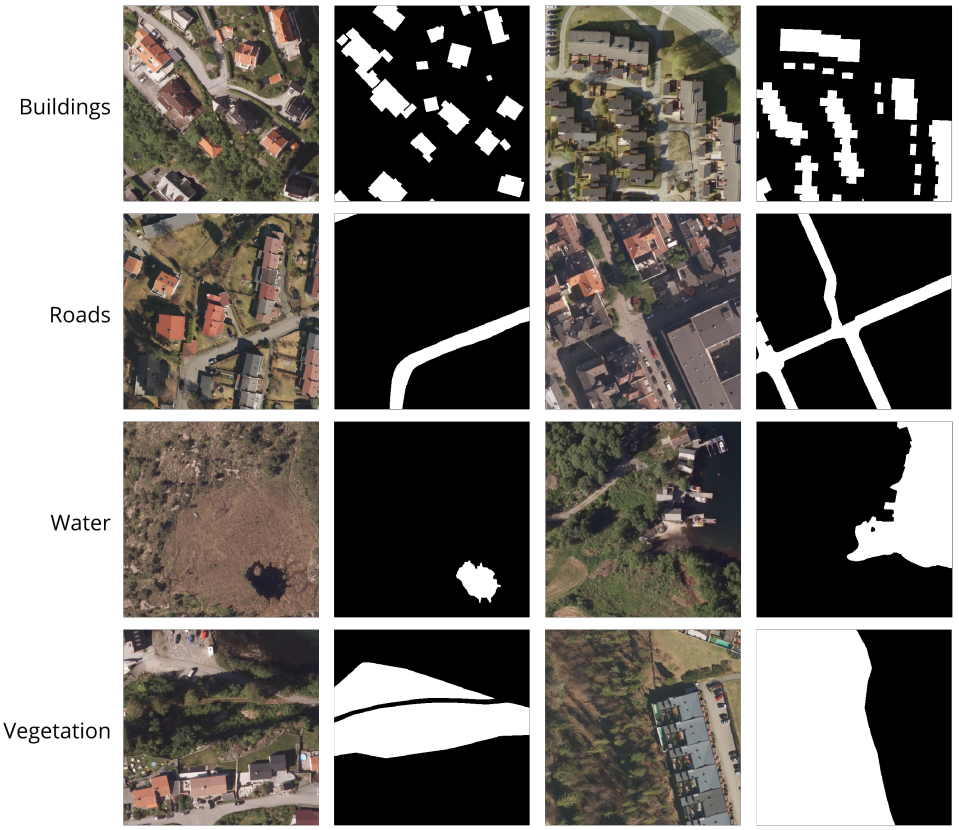

Figure 7.1: Sample images from binary datasets.

City	Buildings (km^2)	Roads (km^2)	Vegetation (km^2)	Water (km^2)
Oslo	17.47	9.93	64.73	15.71
Bergen	11.96	7.12	56.13	26.92
Tromsø	2.45	1.66	11.20	18.73
Trondheim	6.93	5.13	40.52	12.61
Stavanger	4.13	2.84	6.32	9.32
Bodø	1.43	0.96	4.97	1.69
Total	44.37	27.64	183.87	84.98

Table 7.3: Coverage statistics from the different cities.

Label distribution

The distribution of label types in the cities are not equal, as we can see in Table 7.3. We see in Figure 7.2 that there is a correlation between the percentage of building and road labels in the cities. Oslo, Bergen, and Trondheim show a similar percentage of building and road labels. Stavanger has the highest proportion of road and building labels, together with Bodø, whereas Tromsø has the lowest. For vegetation, we see that Stavanger has the smallest proportion and Trondheim has the largest. For water, we see that Tromsø has much more water than any other of the datasets and Oslo has least.

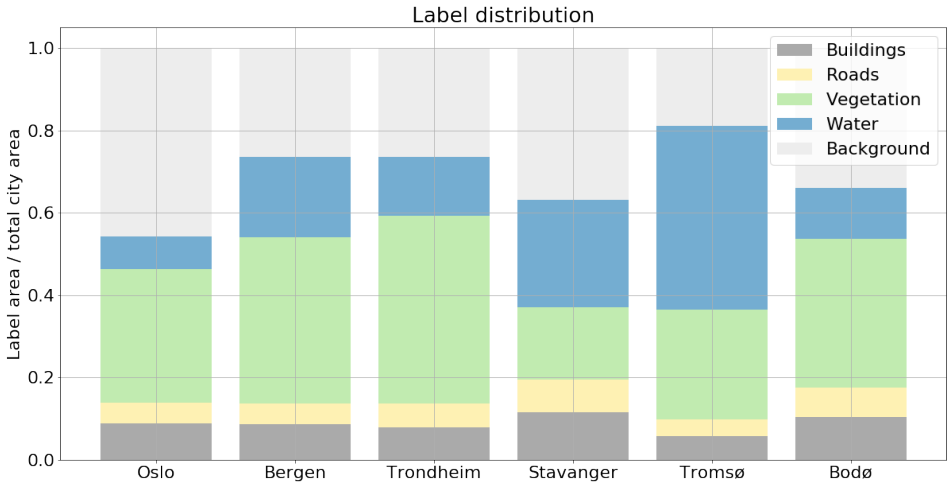

Figure 7.2: Distribution of labels for each of the cities.

Pixel density

We present the pixel density for each of the binary datasets in Figure 7.3. It is evident that there are some imbalances in the datasets. The pixel density of the labeled pixels in the road and building datasets are very low, whereas for vegetation and water the number of labels and background pixels is almost the same.

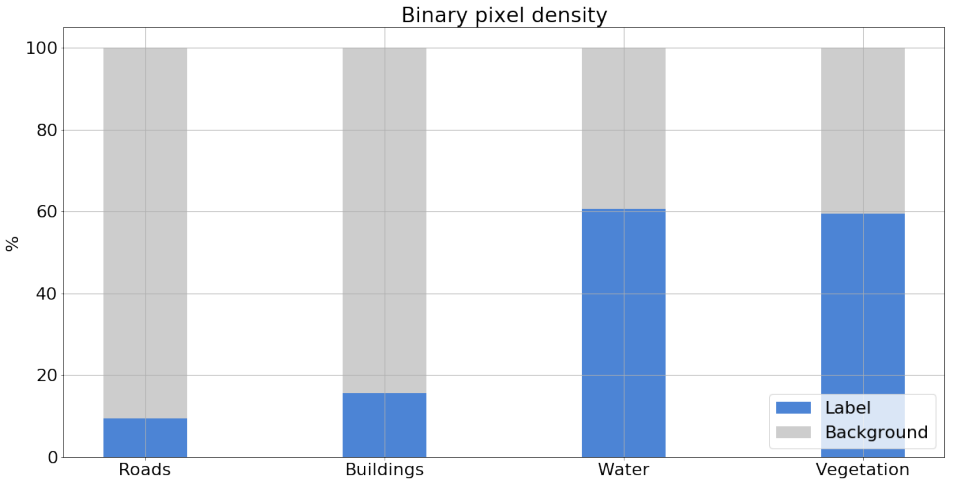

Figure 7.3: Pixel count for each of the binary datasets.

7.2.2 Multiclass

The multiclass dataset is a single dataset where the labels contain at least one of the four feature classes, Figure 7.4. When used for training, the raster labels have color values ranging from zero to five, which would be hard to distinguish by the human eye. Therefore, in Figure 7.4 the colors have been altered to display the different features in the labels.

We can see all five classes represented in at least one of the examples in Figure 7.4. The complexity of the labels varies from having one class covering the whole image to representing multiple classes in advanced structures where similar parts of the aerial images can have different labels.

Pixel distribution

We can see from Figure 7.5 that the multiclass dataset is imbalanced between the different classes. It follows the same distribution as the binary dataset, where

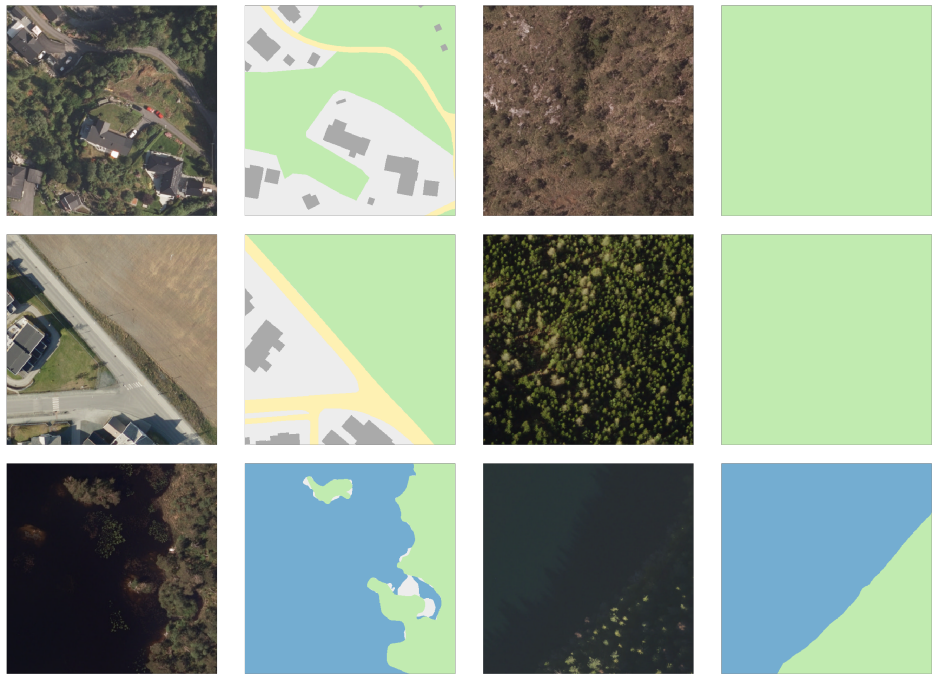

Figure 7.4: Sample images from multiclass datasets.

vegetation and background is the most common class followed by water, buildings, and roads respectively.

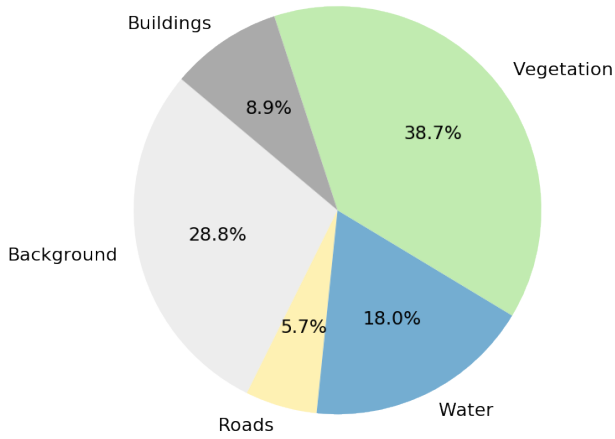

Figure 7.5: Label distribution in multiclass dataset.

7.3 Errors

As stated in research goal **G1**, we want the label creation process to be automatable. Therefore, we do no manual inspection of the datasets.

FKB is not free from errors, and these errors will, therefore, be transferred by the process to the generated dataset. We have found three primary sources of errors in the dataset: Missing and excessive features, miss-labeled features and occluded features.

Missing and excessive features Since the images we download from NiB and FKB are created at different times, some features may not yet have been labeled and some labels might exist for features that are no longer present.

Miss-labeled features Some features are incorrectly labeled either because of human or software errors.

Occluded features In some aerial photographs, occlusions caused by trees and shadows cover the labeled features making them impossible to detect.

In Figure 7.6 we see examples of all the errors mentioned. In (a) we see miss-labeling where suddenly there is a gap in the vegetation label where water labels and road labels mix up unnaturally. In (b) we see an occluded river and a residence labeled as vegetation. Missing buildings and an occluded river can be seen in (c) and (d), respectively.

7.4 Potential

In the proposed dataset we only use a small part of FKB and NiB that is available, but the technology developed has the potential of generating a much larger dataset. This section will discuss the full potential of the FKB and NiB datasets.

Since NiB contains aerial images from all of Norway, the total area of Norway is 385180 km^2 and the generated tiles have an area of $0.010485km^2$ (512x512px), we can, in theory, create a total of 36,736,289 tiled images. However, this might not be the correct number, since the coverage of FKB limits us. By looking at the statistics for FKB for rest of Norway, we can get an idea of the accurate size that the dataset can have. The data presented comes from email correspondence with the Norwegian Mapping authority (Kartverket).

Figure 7.6: Typical errors in the dataset.(a)Miss-labeling with sudden gap in vegetation, (b) Building labeled as vegetation and occluded river, (c) Missing building features and (d) Occluded river

Dataset	Our usage	Total available	Usage percentage
FKB Buildings (km^2)	44.37	527.96	8.40%
FKB Roads (km^2)	28.23	1071.76	2.63%
FKB Water (km^2)	104.19	167,081.00	0.06%
FKB Landcover (km^2)	183.87	385180.00	0.048%

Table 7.4: Available data in FKB

As we can see in Table 7.4, the amount of data available is tremendous and we have only utilized a small amount of it. The numbers for water include marine areas which are under Norwegian control. FKB Landcover is the most extensive dataset because it thoroughly covers Norway with polygons and thus has the same area as the total area of Norway.

The possible utilization amount for the dataset varies because of the requirement that all training images should contain at least one feature. For buildings or roads, this makes it impossible to use all of the 36,736,289 images for the dataset. However, for FKB Landcover one would be able to utilize all of the image tiles.

Chapter 8

Algorithmic Experiments and Results

We will in this chapter present the experiments conducted by using the proposed dataset to train different implementations of deep convolutional neural networks as described in chapter 6.

8.1 Configuration of the network

In order to get the best results it is important to chose and properly configure the network, which includes setting the right hyperparameters (subsection 2.4.6), chosing an optimizer (subsection 2.4.7) and a loss function (subsection 2.4.5).

Learning rate adjustment

As seen in subsection 2.4.6, there are different strategies to how we can adjust the learning rate. In our experiments, we focus on two of them; Annealing learning rate adjustment and Cyclic learning rate adjustment.

Annealing learning rate adjustment The first strategy we test is to reduce the learning rate when the learning stagnates. In our experiments we choose patience of 3 on the validation mIOU and an adjustment factor of $\sqrt{0.1}$, meaning that if the validation mIOU does not improve within three epochs, we reduce the learning rate by a factor of approximately 0.3. Figure 8.1 shows three examples of how the learning rate adjusted during training for three different networks.

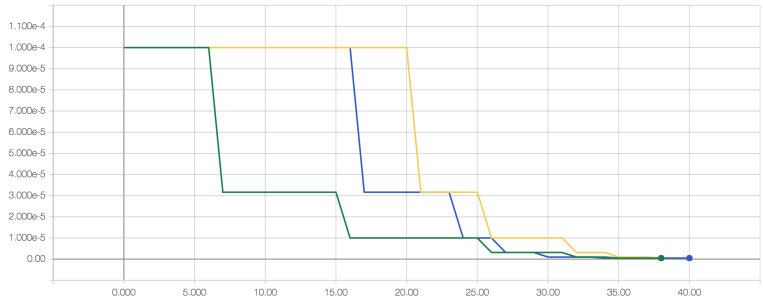

Figure 8.1: Examples of plateau learning rate adjustment

Annealing the learning rate makes the network make large adjustments to the weights in the beginning, and then as the accuracy increases, the adjustments become smaller and smaller.

Cyclic learning rate adjustment We also apply the cyclic learning rate adjustment strategy, presented by Smith [82], as it has proven to yield good results. To decide the base and maximum values for the cyclic learning rate adjustment, we conduct a learning rate test on both the networks, as described in the paper.

During the test, the network is run for ten epochs, while linearly increasing the learning rate form 0 to 0.1. From Figure 8.2 we see that the accuracy proliferates up to approximately 0.9, where it stagnates and neither improve nor decrease before the learning rate approaches 0.1.

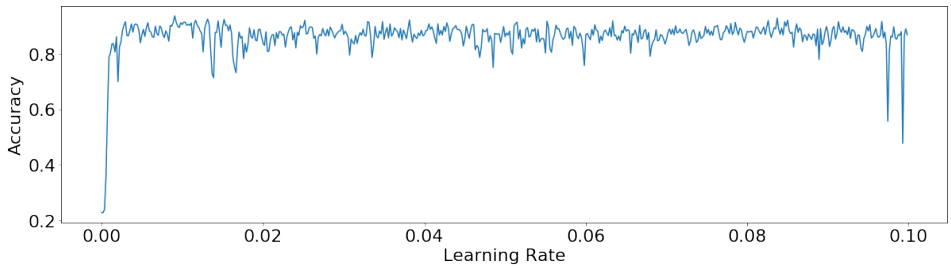

Figure 8.2: Learning rate test for U-Net

By analyzing the graphs displayed in Figure 8.3 and Figure 8.4, we can pick out the most suitable learning rate values. According to Smith [82] the ideal values for the base and maximum learning rates are the points on the test where the learning starts to improve drastically and where it begins to stagnate.

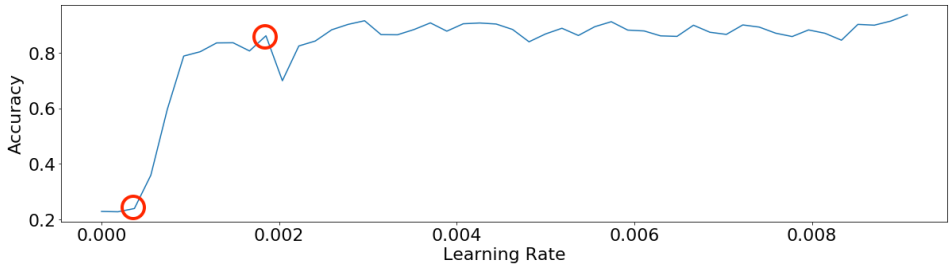

Figure 8.3: Indication of the ideal values for base and maximum learning rate from the U-Net learning rate test

Figure 8.4: Indication of the ideal values for base and maximum learning rate from the FC-DenseNet learning rate test

We base the learning rate values chosen for the networks, on our analysis of what values the accuracy starts increasing and stagnates. We can see the values in Table 8.1.

Network	Base	Maximum
U-Net	0.0002	0.002
DenseNet	0.00002	0.00055

Table 8.1: Values after learning rate test

We apply two different policies for how the cyclic learning rate should behave; The standard *triangular* policy and the *triangular2* policy where we cut the learning rate difference in half after each cycle. We set the step size to 3 times the number of training iterations in an epoch, which means that after six epochs the learning rate has been through one complete cycle.

Regularization adjustment

We test two dropout rates for regularization in the network; A dropout rate of 0.2 and a dropout rate of 0.5. These dropout rates are selected because they are the most common rates used when training convolutional neural networks [52, 81, 31, 4, 12, 69, 38, 9].

8.1.1 Choosing the optimizer

We consider two optimizers for testing; Adam and Nadam. The reason for this choice, as explained in subsection 2.4.7 is that all the different optimizers are part of a developing research field where newer versions solve the older versions problems. Karpathy [44] also recommends Adam as a starting point for testing different networks.

8.1.2 Choosing the loss function

We consider three different functions to evaluate the loss of the predictions. We present two of these in subsection 2.4.5; Categorical Cross Entropy loss and Binary Cross Entropy loss. The last is an experimental loss function called the Soft Jaccard Distance loss. Since we use the Jaccard index as the evaluation metric, it would be ideal to model it as a loss to get the best possible performance. It is however not differentiable and can therefore not be used with backpropagation. We, therefore,

use the *Soft Jaccard loss* [35] which is the combination of cross entropy and the Jaccard index defined as:

$$J(p,t) = \frac{1}{n}\sum_{i=1}^{n}\frac{t_i \cdot p_i + \epsilon}{t_i + p_i - t_i \cdot p_i + \epsilon}$$

$$L = H - logJ$$

Where J is the batch-wise Jaccard index, t_i is the true label, p_t is the prediction, n the batch size, ϵ is a smoothing term in order to cope with zero division and H denotes binary cross entropy and categorical cross entropy for binary and multiclass segmentation problems respectively. This loss helps to maximize the probabilities for right pixels and maximize the intersection between the masks and predictions.

8.2 Testing the different configurations

To estimate which configurations that yield the best results in the final experiments, we conduct a performance test. Using the configuration of U-Net (d) in Figure 6.8, we run the network with all 24 combinations of the parameters described above, for 20 epochs each.

We chose U-Net for these experiments because of its very efficient computation time. When running with a batch size of 20, the average computation time for one epoch is 7.5 minutes, thus making it possible to run all 24 configurations in 60 hours.

As seen in Figure 8.5, the performance of the different configurations is quite similar. The configuration parameters, max F1 score, and max mIOU score for each run are also displayed in Table 8.2.

Looking at the results from Table 8.2, configuration number 4 stands out as the best candidate for further experiments, with both the highest F1 and mIOU scores. The configuration uses the Adam optimizer with a dropout rate of 0.2, the experimental Soft Jaccard Loss function and the learning rate is regulated using the cyclic triangular2 strategy. In Figure 8.5, where we mark the best run with a solid line, we see that difference between the configurations is not very large, except for some outliers, such as run 20 that hit a local minimum.

Nr.	Optimizer	Dropout	Loss func.	rr	Max F1	Max mIOU
0	Adam	0.2	Binary Cross	Annealing	0.87794	0.73200
1	Adam	0.2	Binary Cross	triangular2	0.88251	0.74234
2	Adam	0.2	Binary Cross	triangular	0.88125	0.73782
3	Adam	0.2	Soft Jacc	Annealing	0.88083	0.73891
4	Adam	0.2	Soft Jacc	triangular2	**0.88406**	**0.74505**
5	Adam	0.2	Soft Jacc	triangular	0.88327	0.74309
6	Adam	0.5	Binary Cross	Annealing	0.87866	0.73534
7	Adam	0.5	Binary Cross	triangular2	0.88049	0.73790
8	Adam	0.5	Binary Cross	triangular	0.88037	0.73682
9	Adam	0.5	Soft Jacc	Annealing	0.87829	0.73469
10	Adam	0.5	Soft Jacc	triangular2	0.88283	0.74104
11	Adam	0.5	Soft Jacc	triangular	0.88271	0.74227
12	Nadam	0.2	Binary Cross	Annealing	0.82395	0.65351
13	Nadam	0.2	Binary Cross	triangular2	0.80168	0.61835
14	Nadam	0.2	Binary Cross	triangular	0.88148	0.74070
15	Nadam	0.2	Soft Jacc	Annealing	0.88123	0.74070
16	Nadam	0.2	Soft Jacc	triangular2	0.88216	0.74171
17	Nadam	0.2	Soft Jacc	triangular	0.81964	0.64558
18	Nadam	0.5	Binary Cross	Annealing	0.86058	0.70833
19	Nadam	0.5	Binary Cross	triangular2	0.87848	0.73494
20	Nadam	0.5	Binary Cross	triangular	0.81730	0.64697
21	Nadam	0.5	Soft Jacc	Annealing	0.80656	0.63840
22	Nadam	0.5	Soft Jacc	triangular2	0.88116	0.73912
23	Nadam	0.5	Soft Jacc	triangular	0.88381	0.74377

Table 8.2: Results from the performance test on the different configurations for U-Net.

8.3 Selecting the Number of layers for FC-DenseNet

As explained in subsection 6.3.3, we base our implementation of FC-DenseNet on the memory-inefficient version available at the time of writing. Since the number of training parameters grows when the depth of the network increases, deep implementations of the network will have a high memory consumption. In practice, this means that the amount of GPU memory limits other configurations such as the input and batch size. Looking at the three configurations of DenseNet displayed in Table 8.3, we see that the computation time of one epoch increases with a factor of approximately 1.4 between each configuration, while the maximum batch size decreases with a factor of approximately 0.7.

To get a right balance between computation time and accuracy, we chose to continue the testing with DenseNet-67.

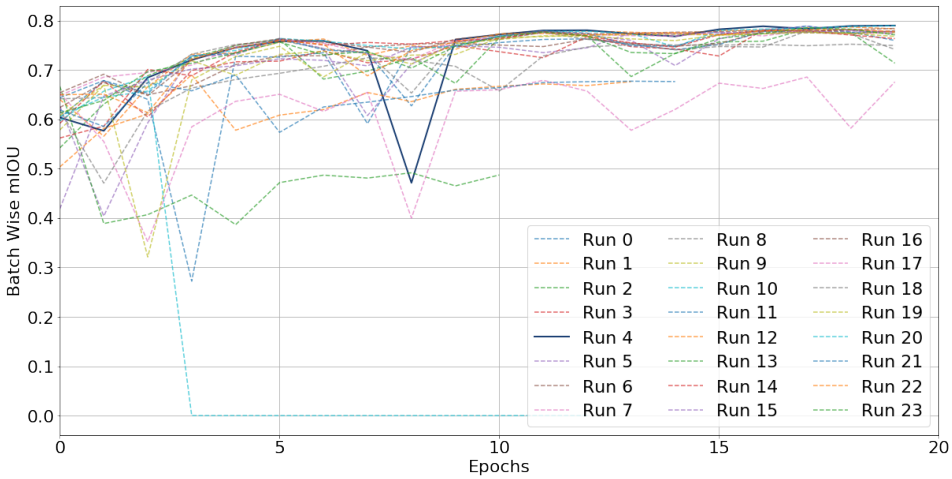

Figure 8.5: The batch-wise mIOU score for the test configuration runs.

Name	Input size	Max batch size	Avg. time per epoch
DenseNet-56	256	6	28m
DenseNet-67	256	4	38m
DenseNet-103	256	2	53m

Table 8.3: Tested input sizes, batch sizes and average computation times for 1 epoch, of different DenseNet

8.4 Training the best configurations

We train the best configuration on each of the datasets. During training, we monitor the mIOU on the validation dataset and stop training once it stagnates. We only save the weights when the performance after an epoch has finished, measured in mIOU, increases.

8.4.1 U-Net

For the U-Net we train the network with batch size 20, images of size 320x320px, the triangular2 cyclic learning rate with a base learning rate of 2e-4 and a max learning rate 2e-3. We then fine tune each of the network on full-size images (512x512px) with a batch size of 12, a base learning rate of 2e-5 and a max learning rate of 2e-4.

In Figure 8.6 we see that the water network has a higher batch-wise mIOU score during training than any other of the datasets.

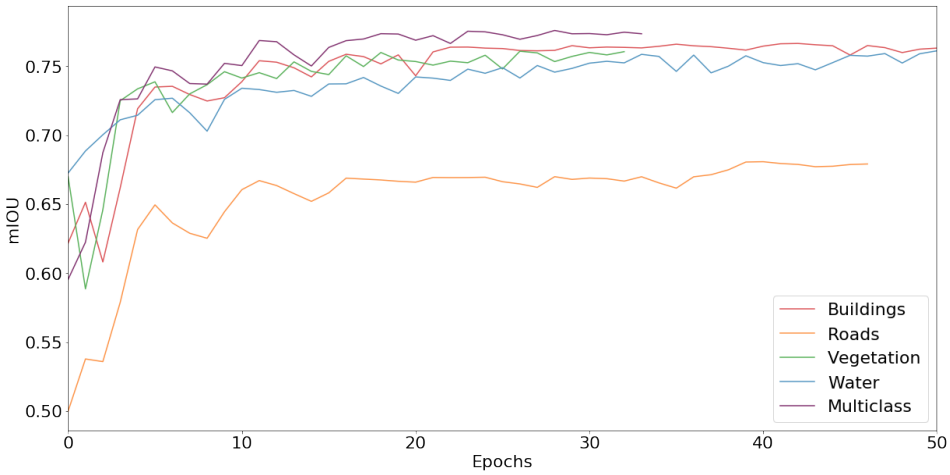

Figure 8.6: Training results for U-Net on all classes

8.4.2 FC-DenseNet

To train the FC-DenseNet, we used the FC-DenseNet-67 configuration with a batch size of 4, an image size of 256x256px and the triangular2 cyclic learning rate with base and maximum learning rate of respectively 2e-5 and 5.5e-4. We then fine-tune the network with a batch size of 2, an image size of 320x320px and the triangular2 cyclic learning rate with base and maximum learning rate of respectively 2e-6 and 5.5e-5.

We see the mIOU during training in Figure 8.7, and the resulting mIOU scores test dataset in Table 9.1.

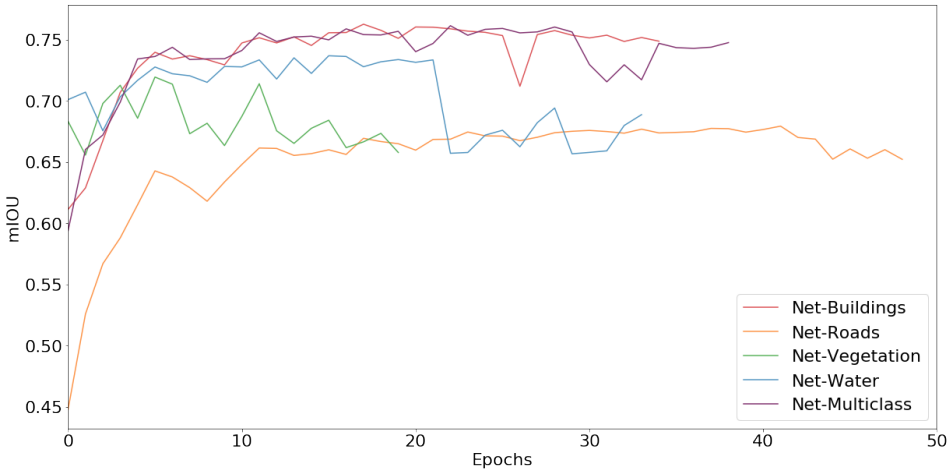

Figure 8.7: Training results for FC-DenseNet-67 on all classes

This page intentionally left blank.

Chapter 9

Results from the Predictions

In this chapter, we present the results for each of the networks and datasets. We show the predictions made on the test set for each of the proposed datasets. We also predict the two generalization test areas from Drammen and Grorudalen.

9.1 Mean IOU and F1 score on the test datasets

We run the two networks on the test dataset to measure the performance of their predictions. We use both the mIOU and the F1 score, see subsection 2.6.4, to measure the performance, and the results are displayed in Table 9.1, where we see the mIOU and F1 scores for both the networks.

Network		Buildings	Roads	Water	Veg.	Multi.
U-Net	mIOU	0.76355	0.66337	**0.77238**	0.77157	**0.77267**
	F1	0.89491	0.84156	**0.98525**	0.93930	0.83316
U-Net	mIOU	**0.77581**	0.67212	0.77164	**0.77458**	0.77258
Finetune	F1	**0.90023**	**0.84502**	0.98462	**0.93953**	**0.83327**
FC-DenseNet-67	mIOU	0.75784	**0.67375**	0.75088	0.72971	0.75720
	F1	0.89150	0.84491	0.98111	0.93118	0.93791
FC-DenseNet-67	mIOU	0.75447	0.66305	0.71365	0.69244	0.74545
Finetune	F1	0.88830	0.83565	0.94849	0.90802	0.81400

Table 9.1: Mean IOU and F1 scores after predictions on the test dataset

The best results are mainly given by the U-Nets, which give the highest F1 score for all the classes, and the highest mIOU score for all classes except Roads. We see in Figure 8.7 that FC-DenseNet-67 outperform its own fine-tuned version in all classes. U-Net, on the other hand, performs better after fine tuning, except for the water class where both the mIOU and F1 scores on the predictions slightly drop. The U-Net gives the best mIOU score for multiclass before fine-tuning, which reflect the fact that the mIOU score did not improve while fine-tuning. The F1 score increases marginally during fine-tuning.

9.2 Examples of Binary Predictions

The scores presented in Table 9.1 does not indicate the qualitative properties of the predictions. In this section, we present selected examples and try to enlighten the qualitative properties of the predictions.

9.2.1 Buildings

We show examples of the predictions on the binary building dataset in Figure 9.1. The four examples are from random locations in the five cities used to create the dataset.

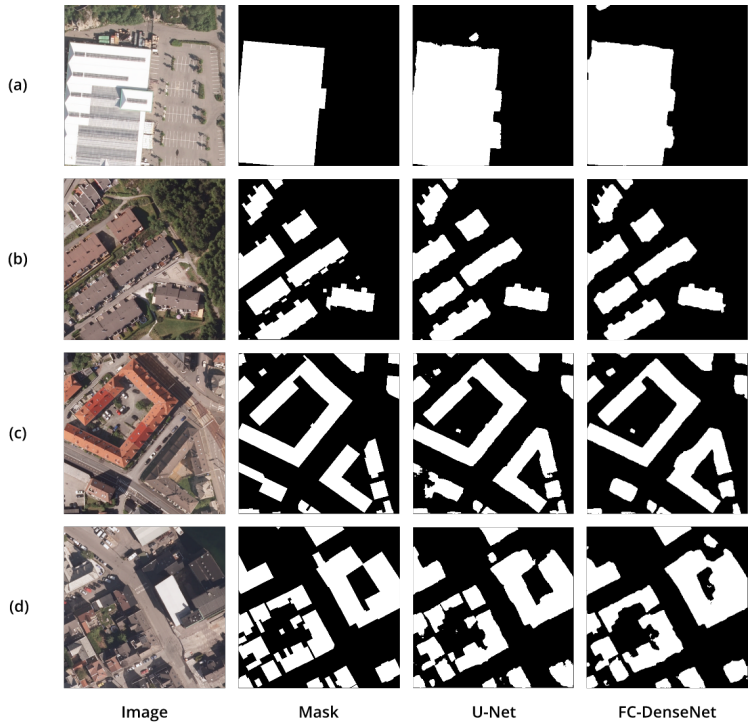

Figure 9.1: Examples of binary building predictions

In example (a) a storage building is visible in the image, and both of the networks seem to label the building correctly. An interesting observation is that the roof over the entrance of the building is not labeled in the mask, but predicted by both the algorithms. In example (b) all the labeled houses are discovered by both the networks. In example (c) there is an error in the mask, where the corner of the bottom-right apartment building is not labeled, but the networks identify the building corner correctly as a building. Example (d), shows multiple complex building structures that both of the networks classify correctly. U-Net has slightly better qualitative predictions.

9.2.2 Roads

In Figure 9.2 we display selected predictions from the binary roads dataset. The input images show examples from different road types in different areas.

Example (a) shows a T intersection with a cycling path along the cross street. FC-DenseNet gives a better and cleaner prediction, where it locates the center top road and have more straight edges. In example (b) we show a highway with a junction

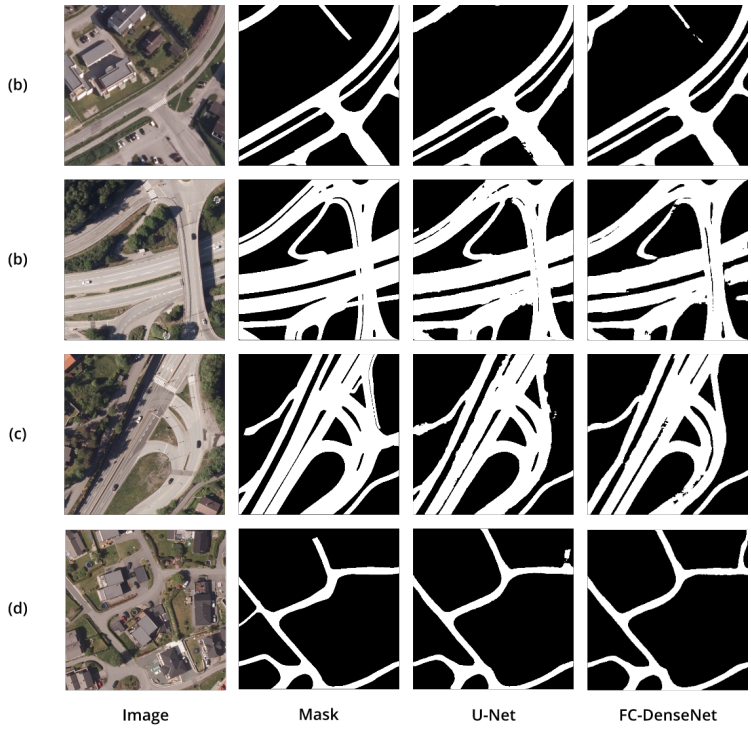

Figure 9.2: Examples of binary road predictions.

and a crossing bridge. Both of the networks manage to predict this complicated scenario well. Example (c) shows a similar situation where the networks show similar performance and are better than each other in different areas of the image. Example (d) shows a residential area, where FC-DenseNet classifies one road at the top-right corner, that U-Net fails to classify.

9.2.3 Vegetation

Figure 9.3 display examples of predictions on the binary vegetation dataset. We pick the selected examples from both residential and non-residential areas.

We see that U-Net is performing better than FC-DenseNet in all the examples. Example (a) is typical in the binary vegetation dataset, where it is unclear what parts of the image that should be labeled vegetation. Both the networks chose to label the top-left corner as vegetation, while the label does not. In example (b) the label lack detail and the networks give a more accurate prediction of the vegetated area. Example (c) shows an example of a more straightforward situation that both the networks manage to predict. Example (d) show a vegetated area

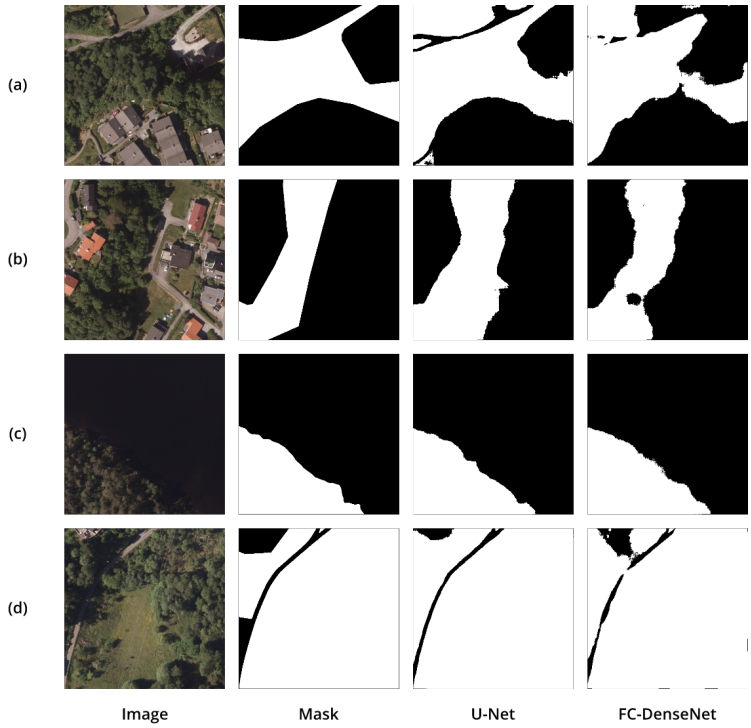

Figure 9.3: Examples of binary vegetation predictions.

with a road going through it. Both the networks identify the road, but the U-Net gives a cleaner prediction.

9.2.4 Water

In Figure 9.4 we show examples of the predictions on the binary water dataset. The selected examples represent different settings where water is present.

With many occluded features and areas that are hard to spot, water labels can be difficult to predict. Example (a) show an area covered by open water where both networks manage to predict the label satisfactorily. Example (b) also shows an open area, but in this example, the water has many tiny islands. We see that none of the networks manage to predict the islands correctly and that FC-DenseNet predicts a too large water area. In example (c) the river is entirely hidden by the trees, and while U-Net predicts that there is no water present in the image, FC-DenseNet predicts false positives (see Figure 2.17). In example (d) there is a small river running alongside the road and through a building complex. Both networks can predict parts of the river, but FC-DenseNet has the same issue as in (c) with

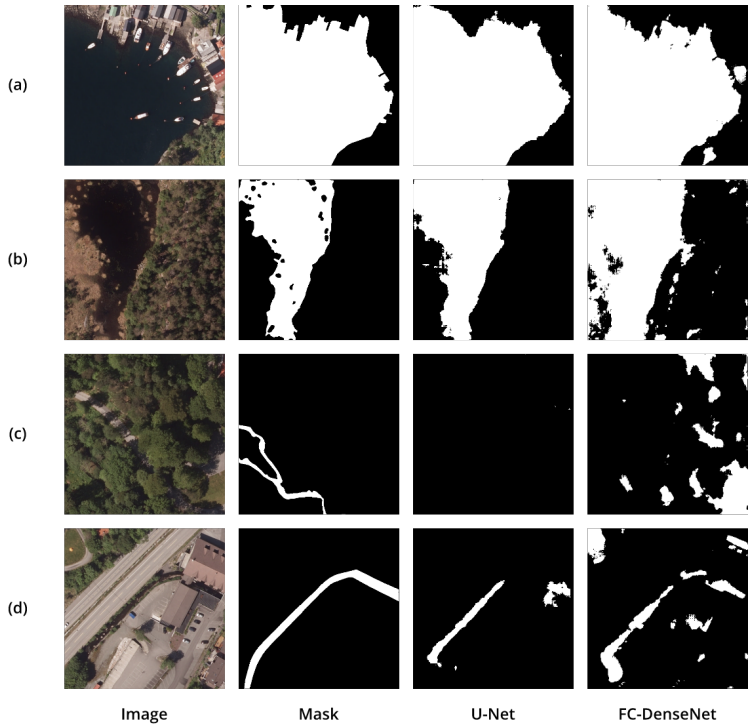

Figure 9.4: Examples of binary water predictions

multiple false positives.

9.3 Examples of multiclass predictions

In Figure 9.5 examples of the multiclass predictions from both the networks are presented. All the four spatial classes are represented in at least one of the examples.

In example (a) both the networks can recognize and connect all the roads in the image, while some of the buildings are not detected. In (b) there are three buildings in the mask, but two of them are very hard to see in the aerial image. None of the networks can detect these two hidden buildings. In example (c) and (d) the predictions are very close to the image masks.

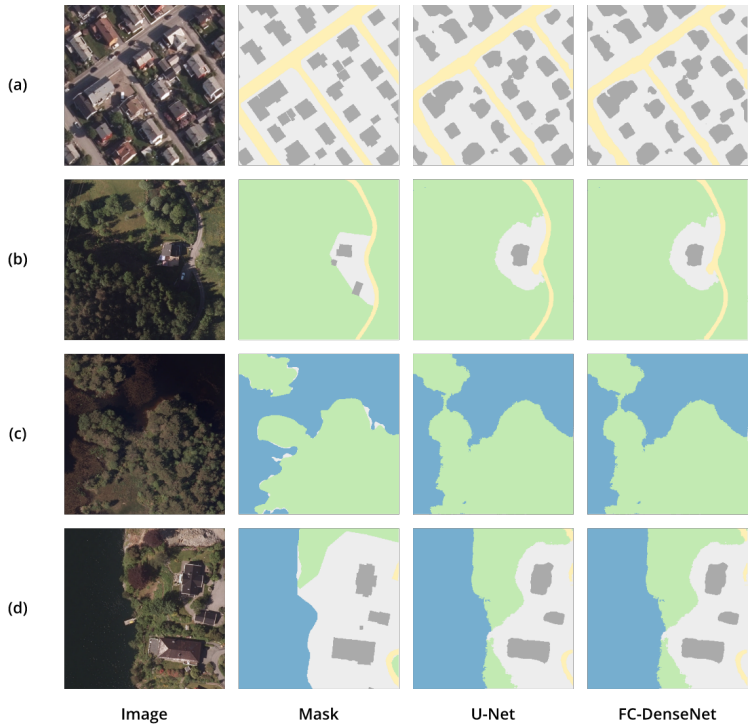

Figure 9.5: Examples of multiclass predictions

9.4 Mean IOU and F1 score of the test areas

The scores collected after predicting the two generalization test areas, using both concatenated patches and merged D4 (mD4) patches, are shown in section 9.4.

From section 9.4 we see that using the mD4 patching approach explained in section 2.7, improves the mIOU and F1 scores of the predictions. A qualitative analysis of the predictions also gives the impression that the technique improves the results, as seen in Figure 9.6.

9.5 Created maps

We will now look at the created maps from Grorudalen and Drammen. We create all maps by using the merged mD4 patches. We construct the maps using both multiclass predictions and by merging binary predictions of all four classes. We handle conflicts in the merging process by using the ranks presented in Table 6.7.

Network		Buildings	Roads	Water	Veg.	Multi.
U-Net Grorudalen (Regular)	mIOU	0.77321	0.74920	0.00134	0.57240	0.49913
	F1	0.87210	0.85662	0.00268	0.72806	0.59785
U-Net Grorudalen (mD4)	mIOU	**0.77827**	**0.77186**	7.87504e-5	0.61323	0.50309
	F1	**0.87532**	**0.87124**	0.00016	0.76025	0.60104
FC-DenseNet Grorudalen (Regular)	mIOU	0.75343	0.73120	**0.00433**	0.65648	0.50520
	F1	0.85938	0.84473	**0.00862**	0.79262	0.60274
FC-DenseNet Grorudalen (mD4)	mIOU	0.76765	0.74430	0.00161	**0.69494**	**0.54346**
	F1	0.86855	0.85341	0.00321	**0.82002**	**0.63023**
U-Net Drammen (Regular)	mIOU	0.64272	0.52823	0.42923	0.24088	0.49654
	F1	0.78251	0.69130	0.60064	0.38824	0.64183
U-Net Drammen (mD4)	mIOU	0.69169	0.54982	0.46973	**0.25275**	0.52486
	F1	0.81775	0.70953	0.63920	**0.40352**	0.66501
FC-DenseNet Drammen (Regular)	mIOU	0.67381	0.58304	**0.60488**	0.12450	0.61962
	F1	0.80512	0.73661	**0.75380**	0.22143	0.74864
FC-DenseNet Drammen (mD4)	mIOU	**0.73585**	**0.64382**	0.60020	0.11414	**0.66329**
	F1	**0.84783**	**0.78332**	0.75015	0.20490	**0.78221**

Table 9.2: Mean IOU and F1 scores for U-Net and FC-DenseNet for Grorudalen and Drammen.

9.5.1 Groruddalen

Figure 9.7 show the binary predictions used for generating the merged binary map. When comparing U-Net and FC-DenseNet, we can see that FC-DenseNet has a cleaner vegetation area compared to U-Net. The road predictions are also slightly better, containing more combined roads sections. We see that even though there is no water in the image, water is predicted multiple places by both networks.

We see that both multiclass and merged binary maps created by U-Net and FC-

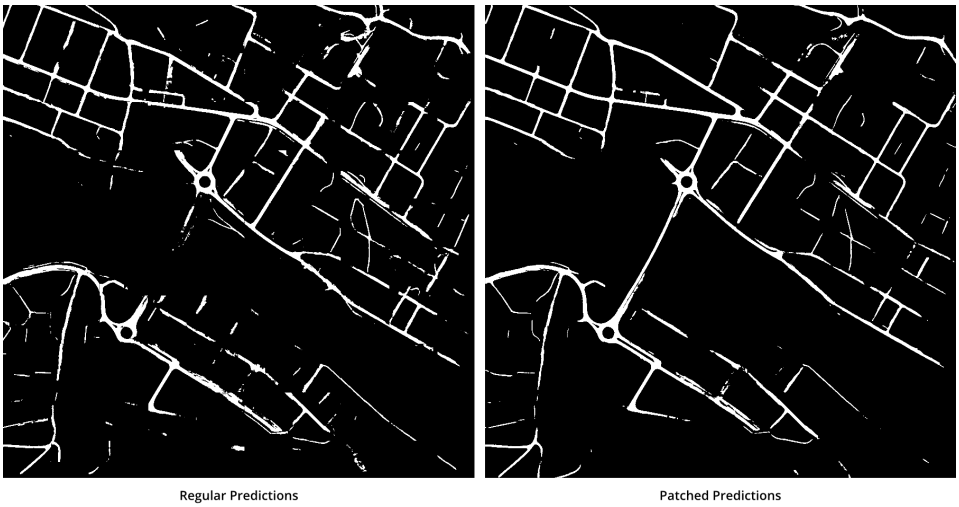

Figure 9.6: Example of difference between regular and mD4 predictions.

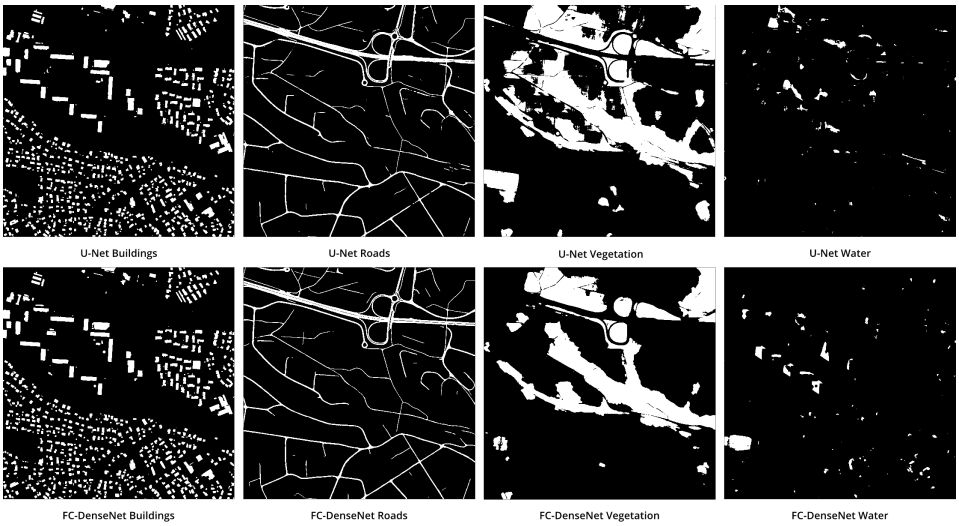

Figure 9.7: Binary predictions of Grorudalen by U-Net and FC-DenseNet

DenseNet in Figure 9.8. We see that the multiclass water predictions are better than binary for both networks. FC-DenseNet has no false positives for water in the multiclass case, whereas U-Net predicts a football field as water. It appears that the predictions done by the binary networks give a better accuracy for roads and buildings, while they struggle with the water class.

Image

U-Net Multiclass Prediction

U-Net Merged Binary Prediction

FC-DenseNet Multiclass Prediction

FC-DenseNet Merged Binary Prediction

Figure 9.8: Map of Groruddalen from U-Net by merged binary and multiclass predictions

9.5.2 Drammen

Drammen portrays a city scene with a river crossing in the middle and a railroad track crossing in the bottom part of the image. In Figure 9.9 we see the binary predictions from U-Net and FC-DenseNet. We see that FC-DenseNet perform better than U-Net for all the classes, with overall more persistent predictions. Both networks struggle with the water class, where they both predict the river in the center correctly, but predicts a lot of false positives for water.

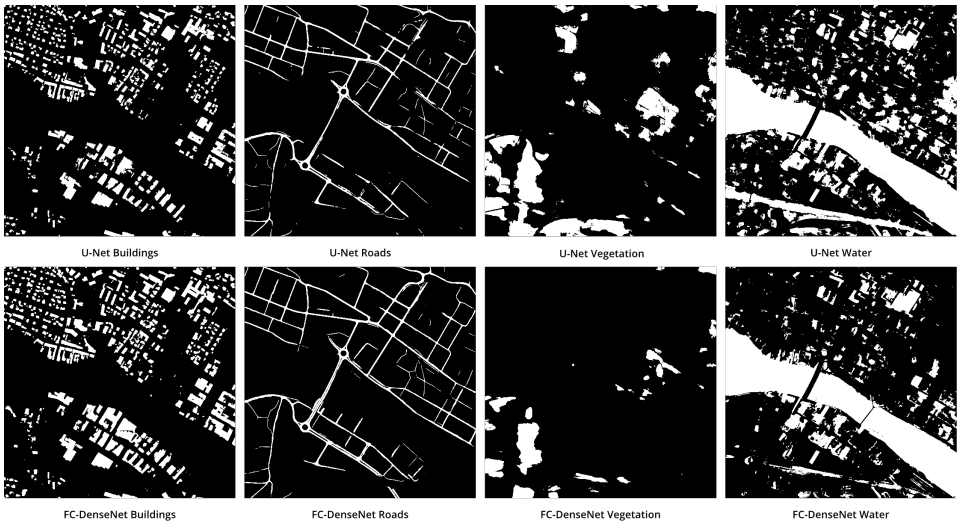

Figure 9.9: Binary predictions of Drammen by U-Net and FC-DenseNet

From the predictions in Figure 9.10 we see the same situation as in the created map of Groruddalen: The merged binary predictions are more accurate for buildings and roads, but false positives in the water predictions occlude the results. The multiclass predictions from both networks outperform the binary for the water class. FC-DenseNet performs better than U-Net with almost no false positives in the multiclass prediction.

Figure 9.10: Map of Drammen from U-Net and FC-DenseNet by merged binary and multiclass predictions

Part III

Discussion and conclusion

This page intentionally left blank.

Chapter 10

Discussion

In this chapter, we discuss our approach and the results of our experiments. First, we look at the generated dataset, and then we evaluate the performance of the network on it.

10.1 The proposed dataset

Even though we decided to only focus on four different classes, the level of detail in the dataset is much broader. We could for example label the dataset using the object type of each FKB Dataset, potentially giving a large variety of classes. This is particularly the case for FKB Landcover, because it is a complete segmentation of Norway, containing 11 different areal classes. For more specific training tasks, such as detailed feature extraction from houses or object detection of features such as chimneys, we could use the FKB Buildings dataset in much more detail.

We use both orthophotos and true orthophotos in the dataset. For tasks such as building extraction, using only true orthophotos might be a better choice because then the network does not need to learn what the correct footprint of a leaning building is. On the other hand, using only true orthophotos limits the amount of input data, since many aerial imagery providers do not provide true orthophotos. There are far fewer projects in NIB that have true orthophotos compared to regular orthophotos. We would, however, need more experiments to conclude on this matter.

As mentioned in section 6.2 the FKB dataset is publicly available, but not free of charge. It is, therefore, worth discussing the possibility of creating a training dataset using only free data. In Norway, the most detailed dataset that is free and publicly available is the N50 dataset.

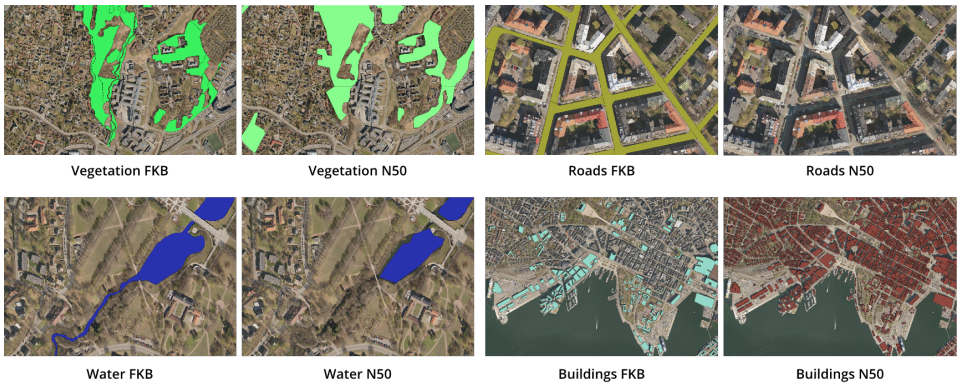

Figure 10.1: Different quality of N50 and FKB datasets taken from different areas in Oslo, Norway

Figure 10.1 show that the level of detail in the labeled data is considerably worse for vegetation, water, and buildings. For roads, the level of detail is almost the same for the two datasets, but N50 does only contain line representations of the roads. If the task were to do center line detection of roads, the N50 dataset would probably

be sufficient to create a training dataset. From the examples in Figure 10.1, we can see that even though it would be possible to generate a training dataset from N50, the quality of it would be inferior compared to the dataset presented in this thesis.

The five cities we select for the proposed dataset are geographically spread out across Norway to ensure diversified training examples. In section 7.1 we state that only two cities, which constitutes 10.75% of the dataset, belong to the northern parts of the country, even though the area of northern Norway is one-third of the total area of Norway. One can argue that we underrepresent the northern regions regarding the area in the dataset. On the other hand, only 10% of Norway's inhabitants live in the northern part of the country. Choosing a proper, geographical distribution of the cities in the dataset comes down to what geographic and demographic factors that represent the output map we want. For us, areas with more inhabitants give better training data for the networks, since we want them to learn to create maps of urban areas.

Our dataset has a small size compared to the full potential of FKB. In this thesis, we chose to only focus on some of the larger cities in Norway. The main reason why we decided not to include more of the available data is that it would have been very time-consuming to train the networks. Furthermore, using the current dataset, we can prove the quality of our method.

As seen in section 7.4, the potential for the dataset is much more significant than what we present in this thesis. In theory, we could use the whole country as a basis. Moreover, we believe that our work shows that it is possible to generate very detailed datasets using the FKB data, which can contribute to further research within fields such as computer vision and remote sensing.

The errors presented in section 7.3 are difficult to fix without the use of manual labor. We might be able to better the number of errors by using very recent aerial images to make sure that they match the labels better. Another solution is to use up-to-date satellite images, as they are updated much more frequently than the images from NIB. The downside with this approach is the price of such high-quality satellite images with licensing that allow deriving features from them.

The water labels contain many rivers that are impossible to see on the aerial photos. From a quality perspective, this is good, but it causes problems since we are only predicting features that are visible in the aerial photos. We can solve this problem if the Norwegian Mapping Authority can provide more information about the datasets. For example which aerial images they use during drawing or the data attribute called *visibility* that is present in the product specification for the FKB

datasets but not in the downloaded files. The *visibility* attribute should describe the degree of visibility of spatial objects, and if correctly used, would probably solve many of the problems with invisible features.

One issue that we encountered when training the networks on the binary water dataset was that the FKB Water dataset covers human-made structures such as bridges. However, this is not the case for FKB Area usage, which also contains an area type for water (see Figure 10.2). We can then ask the question: What is the correct way to label water when we are trying to predict water in aerial images? The way we see it, the correct classification is the classification that resembles the FKB Water dataset the most. The binary water classifier should, therefore, learn to label roads that span over water as water. When we later combine the results from water with the binary road classifier, the ranking of the road classification is higher. The resulting class for these pixels will, therefore, be road. In the multiclass dataset, this is not a problem since the road pixels will appear on top of the water pixels because of the ranking in the labels.

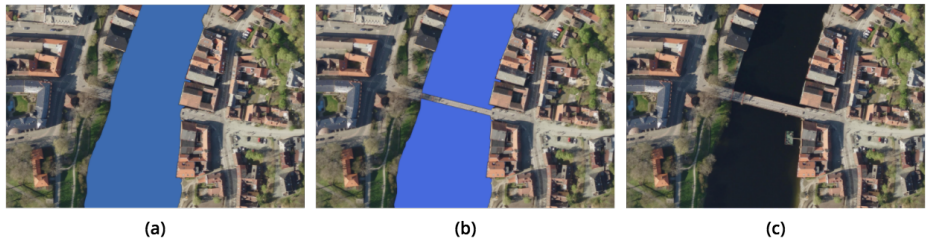

Figure 10.2: Different labels for water. (a) Is the FKB Water dataset covering a bridge, (b) is the FKB AR5 with artype 82 (water) and (c) is the image without any labels

10.2 Training the networks

The configuration of the network was, as seen in section 8.2, based on a test running 24 different combinations of network parameters. Using this approach, we were able to provide a sound basis for the following experiments.

An issue with this approach is that it does not take into account the differences in the four training tasks. Neither does it give an optimal answer as to which configuration the FC-DenseNet should run with since the best configuration will vary based on the architecture of the network and dataset. Therefore, the ideal approach would have been to run the configuration test for all the four tasks with

both the networks. This way we would get the optimal configuration for both the networks on all the tasks, but looking at Table 10.1 the total runtime for all tests would be 2400 hours. Given the time limitations of the thesis, and the fact that the results of the U-Net configuration test on the buildings dataset were almost equally good, we did not prioritize to run all the tests.

Network	Approx. Runtime for 20 epochs
U-Net	2h 30m
DenseNet-67	22h 30m

Table 10.1: Approximate runtimes for 20 epochs for the network configurations.

From Figure 8.7 we see that the mIOU scores decrease after we start fine tuning the FC-DenseNet. We hypothesize that the reason for this is the adjustment of the batch and image size since decreasing the learning rate should not impact the prediction accuracy to that extent. Before fine tuning, the FC-DenseNet is already training with a low batch size because of the memory issues explained in subsection 6.3.4. To halve the batch size and only slightly increasing the image size, probably aggravates the results.

In this thesis, we have focused on testing the proposed dataset on two different network architectures: U-Net and FC-DenseNet. One can argue that we should try more architectures to get a better basis for concluding. However, if we were to train more network architectures on the hardware available, we would need to lower the number of parameters in the configuration test by a significant amount. On the opposite side, it might have been better to only focus on a single network architecture to be able to test more configurations, including a larger number of optimizers.

10.3 Final predictions and the created maps

Table 9.1 shows the final scores of the trained networks after predicting on the test dataset. The mIOU is a good, quantitative measure of how close the predictions match the FKB dataset. However, it is hard to conclude on the performance of the networks solely based on the quantitative results since there are no other attempts at training on the proposed dataset. The results are, however, within the range of what is considered reasonable compared to similar works [35, 6].

Comparing the qualitative properties of the predictions might be more suitable to address the presented research question. From the figures in section 9.2 we

see examples of the final predictions of the trained networks. The building, road and vegetation predictions comply with our minimal map definition and show the relationship between entities in a satisfying manner, making it possible to navigate in urban areas. Both networks are capable of predicting open water but struggle with small rivers that are more or less occluded by objects. The networks trained on binary water, generally predict many false positives compared to the multiclass networks. When navigating urban scenes, buildings and roads represent the most relevant features, because they represent the infrastructure that people must take into account when navigating. We, therefore, value the accuracy and correctness of the road and building predictions higher than the water and vegetation predictions.

Figure 9.8 and Figure 9.10 show that the quality of the water predictions on new areas are not as good as they should be. Water appears in places where there is no indication of water. There can be many reasons to why the networks fail to predict water in some areas. First, a problem with binary predictions is that the networks do not know about other classes than the one they are trying to predict. While the multiclass network can compare the probabilities of several classes, the binary networks can only compare two. Also, as we stated in subsection 6.2.4, all datasets have labels present in all images, which means that it has trained on very few areas that commonly does not contain water. Being presented with such areas might, therefore, result in week predictions.

By examining the predictions on Grorudalen and Drammen from both U-Net and FC-DenseNet, we see that even though FC-DenseNet has a consistently lower mIOU on the training datasets, the predictions on the generalization areas are sometimes better than U-Net's. This shows the importance of validating the network on the task we ultimately want it to perform during training. It would have been better to set up a validation case where we predict the two large test areas with the mD4 patching approach during training instead of just predicting the validation dataset.

There is a significant difference in both qualitative and quantitative results for Grorudalen compared to Drammen. The networks consistently produce lower quality results on Drammen. The main difference between the photos is that they are from different NIB projects. Although the differences in the photographs are small, we have to expect that there are some aspects, such as the camera used, the lighting or the time of year which have been learned by the network while training. The differences in results between Grorudalen and Drammen shows us that the networks have lower generalization capabilities across NIB projects and photo characteristics. A problem with this is that we can not assess the generalization capabilities across different geographical areas before we have the same consistent photo characteristics across all images. We can solve this problem by using satellite photos

from the same satellite. We can also use data augmentation to make the photos have more common characteristics than before.

Data augmentation can help improve the generalization capabilities of the network by altering the quality, lighting, and colors of the images. These alterations can help the network learn the universal properties of the objects it is trying to extract. These properties can, for example, be that a road usually has the same width, independent of the time of year and the quality or color of the road. From chapter 4 we learn that there are many ways to augment the data before we feed it to the neural network, such as enhancing the borders [35, 96] or first segmenting the image using superpixels [1]. In our thesis, we have limited the testing of different augmentation techniques, both because of our time limitations and also because it is not entirely within the scope of what we are trying to prove. However, we think there is little doubt that preprocessing is very important to improve the accuracy of the predictions and to make the networks be able to generalize better for unfamiliar domains.

Even though the road predictions are satisfactory, they have some issues with connectivity. Viewing road detection as a semantic segmentation problem, might not be the best approach if we want valid road networks as output. Through our approach, the networks can learn to extract complex road segments, as seen in Figure 9.2, but they do not learn the importance of connectivity between the segments. Therefore, a better approach might be to use another metric that emphasizes the importance of valid graph structures. We can also apply postprocessing methods on the road segments, as seen in section 5.1, to find the correct connections.

10.3.1 Use cases

During the last decades, the resolution and update frequency of satellite imagery has improved drastically, and as satellite technology improves the need for automatic methods for information extraction increases. If the extraction methods become good enough, the number of possibilities are many.

By allowing machines to draw up maps can help the government agencies of developing countries to get a better understanding of their demography and infrastructure, without the need of extensive manual labor.

The use of fast automatic extraction methods in situations of environmental disasters can be crucial to get a quick overview of damaged areas so that that disaster response teams can act as quickly as possible. Displacement of houses, locating roads that have been blocked and measuring the amount of water in flooded areas

are all examples of such.

As seen in section 9.2, the building predictions are sometimes more correct than the label. We can use CNN's for quality checking already existing spatial data. It is a difficult job to keep track of new buildings and roads, and by using automatic extraction methods and satellite imagery, cartographers can get frequent updates on the change in infrastructure over time.

Chapter 11

Conclusion

In this chapter, we conclude our research and experiments in light of our research goals. We also highlight future work that could be interesting to investigate further.

Our thesis aimed to reach **G1** and **G2** by answering the following question:

Can deep convolutional neural networks learn to create maps based on publicly available spatial data?

We show how to automatically transform publicly available data into high-quality training examples for the use of machine learning algorithms. By using data from FKB and NIB, we create a dataset covering an area of $517.8km^2$ in five cities from the northern and southern parts of Norway. We carefully select four semantic classes and create two types of datasets: Binary, where each of the classes is represented in separate datasets and multiclass, where we represent all classes in the same dataset. We highlight issues in these datasets and show that, while the overall quality is high, there are some significant problems with the data, especially in the case of water where trees occlude many rivers. These are problems with the underlying data and can be hard to solve. We look at the total available data and show that the potential size of the dataset is many times larger than the dataset we propose.

We use the datasets to train two deep convolutional neural networks: U-Net and FC-DenseNet-67. We test different configurations of U-Net and experiment with a deeper network, cyclic learning rates, and soft Jaccard distance loss. We find that cyclic learning rates with the triangular2 configuration and the soft Jaccard distance loss produce the best results. We conclude that we obtain the best overall quantitative results when the networks train on the binary datasets, and their predictions are merged. For water predictions, we conclude that the multiclass networks perform better on new, unseen areas. Our quantitative results are similar to the highest ranking results in challenges such as Inria [62] and DSTL [43].

We increase the mIOU scores when predicting Groruddalen and the city of Drammen by merging overlapping predictions on patches from images rotated and mirrored in the dihedral D4 group. This technique also increases the usability of the created maps since it removes border artifacts. We conclude that slight changes in image characteristics, such as in the comparison between Grorudalen and Drammen, considerably affects the predictions from the networks.

Through our research, we have shown that it is possible to train deep convolutional networks to do semantic segmentation of aerial images, to automatically create maps. In fact, we have shown that for features such as buildings, the CNN's sometimes have a more accurate description of the enclosing area of the features than the label. Showing that it is possible to use our result for quality checking the FKB dataset.

We conclude that the created maps for unseen areas are of such quality that it

is possible to use them for navigating urban scenes and it is, therefore, possible for deep convolutional neural networks to learn to create maps based on publicly available spatial data. However, the created maps are simple, and their use cases are limited. Consequently, there is still a need for high-quality manual cartography.

11.1 Future Work

In this research, we created a relatively small dataset compared to the data available. Future work should utilize more of the data to create a more extensive dataset, reaching a size similar to the ImageNet dataset, by expanding the size and number of features. The dataset should contain images from all the recent aerial photo projects in NIB for the networks to learn diverse characteristics of the photos. It is also possible to expand the dataset to include spatial data from other countries, such as in the Inria dataset.

Future works should strive to combine the more extensive dataset with more computing power to train state-of-the-art networks and their proposed configuration within reasonable times. We show in chapter 9 that even though the mIOU scores of DenseNet is lower than U-Net's, the predictions are sometimes cleaner for roads and buildings. Utilizing a more memory efficient version of DenseNet to be able to train with larger images or batch size is also an exciting topic to pursue in the future. Other implementations that are interesting to try is the Pyramid Scene Parsing module presented by Zhao et al. [100] because of its promising results on the ImageNet Scene Parsing Challenge and the ability to collect contextual information in images. It would also have been fascinating to see the performance of the SegCaps Network presented by LaLonde and Bagci [53], as the use of capsules for dynamic routing is an active field of research.

Data augmentation of the input images should also be examined in order to solve the problem of bad generalization across photo characteristics.

This page intentionally left blank.

Bibliography

[1] Achanta, Radhakrishna et al. "SLIC superpixels compared to state-of-the-art superpixel methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2274–2281. ISSN: 01628828. DOI: 10.1109/TPAMI.2012.120.

[2] Anson, R.W. and Ormeling, F.J. *Basic Cartography: For Students and Technicians*. 1991, pp. ix–x. ISBN: 9781483257129. DOI: 10.1016/B978-1-4832-5712-9.50004-3. URL: http://www.sciencedirect.com/science/article/pii/B9781483257129500043.

[3] Askheim, Svein. *Sør Norge*. 2016. URL: https://snl.no/S%7B%5Co%7Dr-Norge. (visited on 05/25/2018).

[4] Badrinarayanan, Vijay, Kendall, Alex, and Cipolla, Roberto. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2481–2495. ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2644615. arXiv: 1511.00561.

[5] Bhandare, Ashwin et al. "Applications of Convolutional Neural Networks". In: *International Journal of Computer Science and Information Technologies* 7.5 (2016), pp. 2206–2215. URL: http://ijcsit.com/docs/Volume%207/vol7issue5/ijcsit20160705014.pdf.

[6] Bischke, Benjamin et al. "Multi-Task Learning for Segmentation of Building Footprints with Deep Neural Networks". In: (2017), pp. 1–7. arXiv: 1709.05932. URL: http://arxiv.org/abs/1709.05932.

[7] Brostow, Gabriel J, Fauqueur, Julien, and Cipolla, Roberto. "Semantic Object Classes in Video: A High-Definition Ground Truth Database". In: *Pattern Recognition Letters* October 2007 (2008), pp. 1–21.

[8] Cambridge. *Convolutional Neural Networks with Keras*. 2017. URL: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html (visited on 11/14/2017).

[9] Chen, C. et al. "Learning Oriented Region-based Convolutional Neural Networks for Building Detection in Satellite Remote Sensing Images". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-1/W1.June (2017), pp. 461–464. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLII-1-W1-461-2017. URL: http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-1-W1/461/2017/.

[10] Chen, Liang-Chieh et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: (2016), pp. 1–14. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2017.2699184. arXiv: 1606.00915. URL: http://arxiv.org/abs/1606.00915.

[11] Cheng, Guangliang et al. "Accurate urban road centerline extraction from VHR imagery via multiscale segmentation and tensor voting". In: *Neurocomputing* 205 (2016), pp. 407–420. ISSN: 18728286. DOI: 10.1016/j.neucom.2016.04.026. arXiv: 1508.06163.

[12] Cheng, Guangliang et al. "Automatic Road Detection and Centerline Extraction via Cascaded End-to-End Convolutional Neural Network". In: *IEEE Transactions on Geoscience and Remote Sensing* 55.6 (2017), pp. 3322–3337. ISSN: 01962892. DOI: 10.1109/TGRS.2017.2669341.

[13] Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: (2015), pp. 1–14. ISSN: 09226389. DOI: 10.3233/978-1-61499-672-9-1760. arXiv: 1511.07289. URL: http://arxiv.org/abs/1511.07289.

[14] Cordts, Marius et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: (2016). ISSN: 10636919. DOI: 10.1109/CVPR.2016.350. arXiv: 1604.01685. URL: http://arxiv.org/abs/1604.01685.

[15] Cote, Melissa and Saeedi, Parvaneh. "Automatic rooftop extraction in nadir aerial imagery of suburban regions using corners and variational level set evolution". In: *IEEE Transactions on Geoscience and Remote Sensing* 51.1 (2013), pp. 313–328. ISSN: 01962892. DOI: 10.1109/TGRS.2012.2200689.

[16] Debao, Chen. "Degree of approximation by superpositions of a sigmoidal function". In: *Approximation Theory and its Applications* 9.3 (1993), pp. 17–28. ISSN: 10009221. DOI: 10.1007/BF02836480.

[17] DeepGlobe. *DeepGlobe Satellite Image Understanding Challenge*. 2018. URL: http://deepglobe.org/index.html (visited on 04/24/2018).

[18] Dollár, Piotr et al. "Pedestrian detection: An evaluation of the state of the art". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.4 (2012), pp. 743–761. ISSN: 01628828. DOI: 10.1109/TPAMI.2011.155.

[19] Dozat, Timothy. "Incorporating Nesterov Momentum into Adam". In: *ICLR Workshop* 1 (2016), pp. 2013–2016.

[20] Duchi, John, Hazan, Elad, and Singer, Yoram. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159. ISSN: 15324435. DOI: 10.1109/CDC.2012.6426698. arXiv: arXiv:1103.4296v1. URL: http://jmlr.org/papers/v12/duchi11a.html.

[21] Everingham, Mark et al. "The pascal visual object classes (VOC) challenge". In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338. ISSN: 09205691. DOI: 10.1007/s11263-009-0275-4.

[22] Felzenszwalb, Pedro F and Huttenlocher, Daniel P. "Efficient graph-based image segmentation". In: *International Journal of Computer Vision* 59.2 (2004), pp. 167–181. ISSN: 09205691. DOI: 10.1023/B:VISI.0000022288.19776.77.

[23] GDAL. *GDAL Data Model*. 2018. URL: http://www.gdal.org/gdal%7B%5C_%7Ddatamodel.html (visited on 05/02/2018).

[24] GDAL. *GDAL Raster Formats*. 2017. URL: http://www.gdal.org/formats%7B%5C_%7Dlist.html (visited on 03/13/2018).

[25] GDAL. *OGR Architecture*. 2018. URL: http://www.gdal.org/ogr%7B%5C_%7Darch.html (visited on 05/02/2018).

[26] Geodata. *Norge i Bilder*. 2015. URL: http://norgeibilder.no/ (visited on 04/25/2018).

[27] Geonorge. *FKB Geonorge*. 2018. URL: https://kartkatalog.geonorge.no/metadata/uuid/d145d8bf-5564-492c-b704-9bfe4f1a053a (visited on 04/25/2018).

[28] Girshick, Ross et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2014), pp. 580–587. ISSN: 10636919. DOI: 10.1109/CVPR.2014.81. arXiv: 1311.2524.

[29] Heywood, Ian, Cornelius, Sarah, and Carver, Steve. "An Introduction to Geographical Information Systems". In: *Coraput Indus Engng* 1.6 (2006), p. 295. DOI: 10.1680/icien.1996.28911. URL: http://www.amazon.co.uk/Introduction-Geographical-Information-Systems/dp/0131293176.

[30] Hinton, Geoffrey and Tijmen, Tieleman. *Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning*. 2012. URL: http://www.cs.toronto.edu/%7B~%7Dtijmen/csc321/slides/lecture%7B%5C_%7Dslides%7B%5C_%7Dlec6.pdf (visited on 05/08/2018).

[31] Huang, Gao et al. "Densely Connected Convolutional Networks". In: (2016). ISSN: 0002-9645. DOI: 10.1109/CVPR.2017.243. arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[32] Huang, Gary B et al. "Labeled faces in the wild: A database for studying face recognition in unconstrained environments". In: *University of Massachusetts Amherst Technical Report* 1 (2007), pp. 07–49. ISSN: 1996756X. DOI: 10.1.1.122.8268.

[33] Huang, Xin and Zhang, Liangpei. "Road centreline extraction from high-resolution imagery based on multiscale structural features and support vector machines". In: *International Journal of Remote Sensing* 30.8 (2009), pp. 1977–1987. ISSN: 01431161. DOI: 10.1080/01431160802546837.

[34] Ibrahim, Abdelhameed, Salem, Muhammed, and Ali, Hesham Arafat. "Automatic Quick-Shift Segmentation for Color Images". In: 11.3 (2014), pp. 122–127.

[35] Iglovikov, Vladimir, Mushinskiy, Sergey, and Osin, Vladimir. "Satellite Imagery Feature Detection using Deep Convolutional Neural Network: A Kaggle Competition". In: (2017). arXiv: 1706.06169. URL: http://arxiv.org/abs/1706.06169.

[36] Inglada, Jordi. "Automatic recognition of man-made objects in high resolution optical remote sensing images by SVM classification of geometric image features". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 62.3 (2007), pp. 236–248. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2007.05.011.

[37] J., Debayle. "An Integrated Method for Urban Main Road Centerline Extraction from Optical Remotely Sensed Imagery". In: *IEEE Transactions on Geoscience and Remote Sensing* 52.6 (2014), p. 3359. ISSN: 01962892. DOI: 10.1109/TGRS.2013.2272593.

[38] Jegou, Simon et al. "The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* 2017-July (2017), pp. 1175–1183. ISSN: 21607516. DOI: 10.1109/CVPRW.2017.156. arXiv: 1611.09326.

[39] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 248–255. ISSN: 1063-6919. DOI: 10.1109/CVPRW.2009.5206848. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5206848.

[40] Jiten, Joakim and Merialdo, Bernard. "Semantic Image Segmentation with a Multi- dimensional Hidden Markov Model". In: *Advances in Multimedia Modeling* (2007), pp. 616–624.

[41] Kaggle Team. *Dstl Satellite Imagery Competition, 1st Place Winner's Interview: Kyle Lee*. 2017. URL: http://blog.kaggle.com/2017/04/26/dstl-satellite-imagery-competition-1st-place-winners-interview-kyle-lee/ (visited on 04/13/2018).

[42] Kaggle Team. *Dstl Satellite Imagery Competition, 3rd Place Winners' Interview: Vladimir & Sergey*. 2017. URL: http://blog.kaggle.com/2017/05/09/dstl-satellite-imagery-competition-3rd-place-winners-interview-vladimir-sergey/ (visited on 05/22/2018).

[43] Kaggle Team. *Dstl Satellite Imagery Feature Detection*. 2017. URL: https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection (visited on 04/24/2018).

[44] Karpathy, Andrej. *Convolutional Neural Networks for Visual Recognition*. 2017. URL: http://cs231n.github.io/convolutional-networks/ (visited on 05/06/2018).

[45] Kartverket. *Norway Digital*. 2010. URL: https://www.geonorge.no/en/infrastructure/spatial-data-infrastructure/norway-digital/ (visited on 04/25/2018).

[46] Kartverket. "Produktspesifikasjon FKB-Veg". In: (2016).

[47] Kartverket. "SOSI Del 3 Produktspesifikasjon for Felles KartdataBase (FKB)". In: (2013).

[48] Kass, M., Witkin, a., and Terzopoulos, D. "Snakes: Active contour models". In: *International Journal of Computer Vision* 1.4 (1988), pp. 321–331. ISSN: 09205691. DOI: 10.1007/BF00133570.

[49] Kim, Taejung and Muller, Jan-Peter. "Development of a graph-based approach for building detection". In: *Image and Vision Computing* 17.1 (1999), pp. 3–14. ISSN: 02628856. DOI: 10.1016/S0262-8856(98)00092-4. URL: http://linkinghub.elsevier.com/retrieve/pii/S0262885698000924.

[50] Kingma, Diederik P. and Ba, Jimmy. "Adam: A Method for Stochastic Optimization". In: (2014), pp. 1–15. ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[51] Krizhevsky, Alex. "Learning Multiple Layers of Features from Tiny Images". In: ... *Science Department, University of Toronto, Tech.* ... (2009), pp. 1–60. ISSN: 1098-6596. DOI: 10.1.1.222.9220. arXiv: arXiv:1011.1669v3. URL: http://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:Learning+Multiple+Layers+of+Features+from+Tiny+Images%7B%5C#%7D0.

[52] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances In Neural Information Processing Systems* (2012), pp. 1–9. ISSN: 10495258. DOI: http://dx.doi.org/10.1016/j.protcy.2014.09.007. arXiv: 1102.0183.

[53] LaLonde, Rodney and Bagci, Ulas. "Capsules for Object Segmentation". In: Midl (2018), pp. 1–9. arXiv: 1804.04241. URL: http://arxiv.org/abs/1804.04241.

[54] Laptev, I. et al. "Automatic extraction of roads from aerial images based on scale space and snakes". In: *Machine Vision and Applications* 12.1 (2000), pp. 23–31. ISSN: 0932-8092, 1432-1769. DOI: 10.1007/s001380050121.

[55] Lary, David J. et al. "Machine learning in geosciences and remote sensing". In: *Geoscience Frontiers* 7.1 (2016), pp. 3–10. ISSN: 16749871. DOI: 10.1016/j.gsf.2015.07.003. URL: http://dx.doi.org/10.1016/j.gsf.2015.07.003.

[56] Le Cun, Yann et al. "Gradient-Based Learning Applied to Document Recognition". In: (1998). ISSN: 00189219. DOI: 10.1109/5.726791. arXiv: 1102.0183.

[57] LeCun, Y, Cortes, C, and Burges, C J C. *The MNIST dataset of handwritten digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/ (visited on 04/19/2018).

[58] Lemmens, Mathias Johannes Peter Maria. *Geo-information : Technologies, applications and the environment*. Springer, 2011, p. 349. ISBN: 9789400716667. URL: https://www.gim-international.com/content/article/digital-photogrammetric-workstations.

[59] Li, Er et al. "Robust rooftop extraction from visible band images using higher order CRF". In: *IEEE Transactions on Geoscience and Remote Sensing* 53.8 (2015), pp. 4483–4495. ISSN: 01962892. DOI: 10.1109/TGRS.2015.2400462.

[60] Lin, Tsung Yi et al. "Microsoft COCO: Common objects in context". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8693 LNCS.PART 5 (2014), pp. 740–755. ISSN: 16113349. DOI: 10.1007/978-3-319-10602-1_48. arXiv: 1405.0312.

[61] Liu, Ziwei et al. "Deep Learning Markov Random Field for Semantic Segmentation". In: (2016), pp. 1–14. ISSN: 10636919. DOI: 10.1007/978-3-319-46723-8. arXiv: 1606.07230. URL: http://arxiv.org/abs/1606.07230.

[62] Maggiori, Emmanuel et al. "Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark". In: *International Geoscience and Remote Sensing Symposium (IGARSS)* 2017-July (2017), pp. 3226–3229. DOI: 10.1109/IGARSS.2017.8127684.

[63] Marr, D. "Vision". In: *Freeman, San Francisco* (1982).

[64] Miao, Zelang et al. "Road centerline extraction from high-resolution imagery based on shape features and multivariate adaptive regression splines". In: *IEEE Geoscience and Remote Sensing Letters* 10.3 (2013), pp. 583–587. ISSN: 1545598X. DOI: 10.1109/LGRS.2012.2214761.

[65] Minsky and Papert. *Perceptrons*. 1969.

[66] Mokhtarzade, M. and Zoej, M. J Valadan. "Road detection from high-resolution satellite images using artificial neural networks". In: *International Journal of Applied Earth Observation and Geoinformation* 9.1 (2007), pp. 32–40. ISSN: 15698432. DOI: 10.1016/j.jag.2006.05.001.

[67] Nesterov, Y. "A method of solving a convex programming problem with convergence rate {O}(1/kˆ2)". In: *Soviet Mathematics Doklady* 27.2 (1983), pp. 372–376. URL: http://www.core.ucl.ac.be/%7B~%7Dnesterov/Research/Papers/DAN83.pdf.

[68] Neuenschwander, W et al. "From Ziplock Snakes to Velcroˆ{\rm TM} Surfaces". In: *Ascona Workshop on Automatic Extraction of Man-Made Objects from Aerial and Space Images* (1995), pp. 105–114.

[69] Park, Sungheon and Kwak, Nojun. "Analysis on the dropout effect in convolutional neural networks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10112 LNCS (2017), pp. 189–204. ISSN: 16113349. DOI: 10.1007/978-3-319-54184-6_12.

[70] Patterson, Josh and Gibson, Adam. *Deep Learning - A Practitioner's Approach*. 2017.

[71] Paule, Torbjørn. "Geovekst – et vellykket samarbeid om digital kartlegging". In: (2012). URL: https://kartverket.no/globalassets/om-kartverket/geovekst/geovekst-jubileumshefte-2012.pdf.

[72] Pleiss, Geoff et al. "Memory-Efficient Implementation of DenseNets". In: (2017). arXiv: 1707.06990. URL: http://arxiv.org/abs/1707.06990.

[73] Postgis. *ST_AsRaster*. 2018. URL: https://postgis.net/docs/RT%7B%5C_%7DST%7B%5C_%7DAsRaster.html (visited on 04/26/2018).

[74] Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015), pp. 1–8. ISSN: 16113349. DOI: 10.1007/978-3-319-24574-4_28. arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[75] Russakovsky, Olga et al. "Detecting avocados to Zucchinis: What have we done, and where are we going?" In: *Proceedings of the IEEE International Conference on Computer Vision* (2013), pp. 2064–2071. ISSN: 1550-5499. DOI: 10.1109/ICCV.2013.258.

[76] Sabour, Sara, Frosst, Nicholas, and Hinton, Geoffrey E. "Dynamic Routing Between Capsules". In: Nips (2017). arXiv: 1710.09829. URL: http://arxiv.org/abs/1710.09829.

[77] Shelhamer, Evan, Long, Jonathan, and Darrell, Trevor. "Fully Convolutional Networks for Semantic Segmentation". In: (2016), pp. 1–12. ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2572683. arXiv: 1605.06211. URL: http://arxiv.org/abs/1605.06211.

[78] Shelhamer, Evan, Long, Jonathan, and Darrell, Trevor. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651. ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2572683. arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038.

[79] Shi, Jianbo and Malik, Jitendra. "Normalized cuts and image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8 (2000), pp. 888–905. ISSN: 01628828. DOI: 10.1109/34.868688. arXiv: 0703101v1 [cs].

[80] Signal, Digital, Ssing, P Roce, and No, Article. "Automatic Road Extraction fromAerial Images". In: 224.1998 (1998), pp. 215–224.

[81] Simonyan, Karen and Zisserman, Andrew. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: (2014), pp. 1–14. ISSN: 09505849. DOI: 10.1016/j.infsof.2008.09.005. arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[82] Smith, Leslie N. "Cyclical Learning Rates for Training Neural Networks". In: April (2015). DOI: 10.1109/WACV.2017.58. arXiv: 1506.01186. URL: http://arxiv.org/abs/1506.01186.

[83] Srivastava, Nitish et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. ISSN: 15337928. DOI: 10.1214/12-AOS1000. arXiv: 1102.4807.

[84] Sun, Chen et al. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: (2017). DOI: 1707.02968. arXiv: 1707.02968. URL: http://arxiv.org/abs/1707.02968.

[85] Szegedy, Christian et al. "Going Deeper with Convolutions". In: (2014), pp. 1–9. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842.

[86] The DowlinQ. *2nd SpaceNet Competition Winners Code Release*. 2017. URL: https://medium.com/the-downlinq/2nd-spacenet-competition-winners-code-release-c7473eea7c11 (visited on 04/24/2018).

[87] The PostgreSQL Global Development Group. *PostgreSQL: Contributor Profiles*. 2017. URL: https://www.postgresql.org/community/contributors/ (visited on 02/10/2018).

[88] Thorsnæs, Geir. *Nord Norge*. 2017. URL: https://snl.no/Nord-Norge (visited on 05/25/2018).

[89] TINGWU WANG. "Tutorial of Semantic Segmentation". In: ().

[90] TopCoder. *SpaceNet on Amazon Web Services (AWS)*. 2016. URL: https://spacenetchallenge.github.io/datasets/datasetHomePage.html (visited on 04/23/2018).

[91] Vosselman, George and Knecht, Jurrien de. "Road tracing by profile matching and Kalman filtering". In: *Automatic Extraction of Man-Made Objects from Aerial and Space Images I* 1976 (1995), pp. 265–275. ISSN: 1098-6596. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

[92] Weisstein, Eric W. *Dihedral Group D_4*. 2018. URL: http://mathworld.wolfram.com/DihedralGroupD4.html (visited on 05/13/2018).

[93] Wertheimer, Max. "Laws of organization in perceptual forms." In: *A source book of Gestalt psychology*. 1923 (1938), pp. 71–88. ISSN: 03400727. DOI: 10.1037/11496-005. arXiv: arXiv:1011.1669v3. URL: http://content.apa.org/books/11496-005.

[94] Wong, Yuk-Man. "A Study of Approaches for Object Recognition". In: *Proceedings of the IEEE* 77.12 (2005), pp. 1797–1815. ISSN: 15582256. DOI: 10.1109/5.48824.

[95] Wu, Songtao, Zhong, Shenghua, and Liu, Yan. "Deep residual learning for image steganalysis". In: *Multimedia Tools and Applications* (2015), pp. 1–17. ISSN: 15737721. DOI: 10.1007/s11042-017-4440-4. arXiv: 1512.03385.

[96] Xu, Yongyang et al. "Building extraction in very high resolution remote sensing imagery using deep learning and guided filters". In: *Remote Sensing* 10.1 (2018). ISSN: 20724292. DOI: 10.3390/rs10010144.

[97] Yuan, Jiangye. "Automatic Building Extraction in Aerial Scenes Using Convolutional Networks". In: (2016). ISSN: 0162-8828. DOI: 10.1109/TPAMI.2017.2750680. arXiv: 1602.06564. URL: http://arxiv.org/abs/1602.06564.

[98] Zeiler, Matthew D. "ADADELTA: An Adaptive Learning Rate Method". In: (2012). ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. arXiv: 1212.5701. URL: http://arxiv.org/abs/1212.5701.

[99] Zeiler, Matthew D. et al. "Deconvolutional networks". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010), pp. 2528–2535. ISSN: 10636919. DOI: 10.1109/CVPR.2010.5539957. arXiv: 1302.1700.

[100] Zhao, Hengshuang et al. "Pyramid Scene Parsing Network". In: (2016). DOI: 10.1109/CVPR.2017.660. arXiv: 1612.01105. URL: http://arxiv.org/abs/1612.01105.

[101] Zhou, Bolei et al. "Semantic Understanding of Scenes through the ADE20K Dataset". In: (2016). arXiv: 1608.05442. URL: http://arxiv.org/abs/1608.05442.

Appendix

This page intentionally left blank.

Appendix A

Open-Source projects used

An open-source project is a term used for computer programs where the owners have made the source code available under a license, such as the Apache 2.0 or the MIT License[1].

Open source projects are important because they enable collaborative development between multiple sources, creating better products for the end user in terms transparency and maintenance. The work done in this thesis would not have been possible without the use of open source projects.

A.1 PostgreSQL

PostgreSQL (often referred to as postgres) is an object-relation database management system (ORDBMS) which has been developed and maintained by an open source community since 1997 [87].

The postgres system is fully ACID compliant, meaning that it can be characterized by the following properties:

Atomicity Atomicity says that either all operations in a database transaction occurs or none. This way there are no "dangling" operations that can create inconsistency in the database.

[1] Link: https://opensource.org/licenses

Consistency Consistency ensures that all transactions will bring the database from one valid state to another.

Isolation This property makes sure that if more transactions occur concurrently, the result of the system state would be equal to a scenario where the transactions were to be executed sequentially.

Durability A database system is said to be durable if when a transaction has been committed it will remain so even if the whole system goes down.

PostgreSQL uses GiST (Generalized Search Tree) indexing, which also serves as the foundation for the open source project PostGIS. GiST is a height-balanced tree structure with tree nodes that contains predicate (p) and pointer (ptr) pairs. The predicate is used as the search key, while the pointer points to the database record associated with the key. GiST has the following properties:

- Each node contains between min and max index entries unless it is the root
- For each index entry (p,ptr) in a leaf node, p is true when instantiated with the values from the indicated tuple (i.e., p holds for the tuple).
- For each index entry (p, ptr) in a non-leaf node, p is true when instantiated with the values of any tuple reachable from ptr.
- The root has at least two children unless it is a leaf
- All leaves in the tree appear on the same level

A.2 PostGIS

PostGIS is an open source extension for postgres which adds support for geographic objects and spatial queries to the PostgreSQL database system that was initially released April 19, 2001. Geographical features that are supported are: Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, MultiPolygons and GeometryCollections.

Furthermore, spatial predicates for determining interaction of geometries and spatial operators, such as area, distance, union and difference measurements are provided.

A.3 Geospatial Data Abstraction Library (GDAL)

GDAL is a software library used to read, write and process both raster and vector formats that have been georeferenced. GDAL is used by a large quantity of computer programs, such as ArcGIS and QGIS, and holds support for 154 raster and 93 vector geospatial data formats [24].

In order to support the large variety of formats the library converts the data into a single raster abstract data model [23] or a single vector abstract data model [25], before it is presented to the calling application.

Typical use cases for GDAL are:

A.4 Tensorflow

Tensorflow is an open source software library used for dataflow programming, and it is used for easy implementation of machine learning applications such as neural networks. Tensorflow was developed by the Google Brain team for internal use at Google, but was later released under the Apache 2.0 open source license[2] in 2015.

Tensorflow has the ability to run on multiple CPUs and GPUs, and its computations are expressed as a stateful dataflow graph, which is suitable for numeric processing.

A.5 Keras

Keras is an open source neural network library written in Python, designed to enable fast experimentation with deep neural networks. The library runs on top of Tensorflow and contains numerous implementations of commonly used neural network building blocks.

[2]Link: https://www.apache.org/licenses/LICENSE-2.0

This page intentionally left blank.

Appendix B

SQL queries

This appendix present the SQL queries that is used to generate the different datasets.

B.1 Binary Buildings

```
WITH area AS (
SELECT st_asraster(st_intersection(geom, st_makeenvelope({
    min_x}, {min_y}, {max_x}, {max_y}, 25833)),
ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {max_y
    }::FLOAT, {x_scale}, {y_scale}, 0, 0, 25833),'8BUI', {
    color_attribute}::INTEGER, 0) as rast
FROM bygning_flate
WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y},
     {max_x}, {max_y}, 25833)) AND color = 4
),
empty as (
SELECT st_asraster(
st_makeenvelope({min_x}, {min_y}, {max_x}, {max_y}, 25833,
    ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {
    max_y}::FLOAT, {x_scale}, {y_scale}, 0, 0, 25833),
'8BUI', 0, 0) as rast
)
SELECT ST_AsGDALRaster(st_union(foo.rast, 'max'),'GTiff')
FROM (
```

```
SELECT rast FROM area
UNION SELECT rast FROM empty) foo
```

B.2 Binary Roads

```
WITH area AS (
SELECT st_asraster(st_intersection(geom, st_makeenvelope({
    min_x}, {min_y}, {max_x}, {max_y}, 25833)),
ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {max_y
    }::FLOAT, {x_scale}, {y_scale}, 0, 0, 25833),
'8BUI', {color_attribute}::INTEGER, 0) as rast
FROM veg_flate
WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y},
     {max_x}, {max_y}, 25833)) AND color = 1
),
empty as (
SELECT st_asraster(
st_makeenvelope({min_x}, {min_y}, {max_x}, {max_y}, 25833),
ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {max_y
    }::FLOAT, {x_scale}, {y_scale}, 0, 0, 25833),
'8BUI', 0, 0) as rast
)
SELECT ST_AsGDALRaster(st_union(foo.rast, 'max'),'GTiff')
FROM (
SELECT rast FROM area
UNION SELECT rast FROM empty) foo
```

B.3 Binary Vegetation

```
WITH area AS (
SELECT st_asraster(st_intersection(geom, st_makeenvelope({
    min_x}, {min_y}, {max_x}, {max_y}, 25833)),
ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {max_y
    }::FLOAT, {x_scale}, {y_scale}, 0, 0, 25833),
'8BUI', {color_attribute}::INTEGER, 0) as rast
FROM ar5_flate
WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y},
     {max_x}, {max_y}, 25833)) AND color = 3
),
area2 AS (
SELECT st_asraster(st_intersection(geom, st_makeenvelope({
    min_x}, {min_y}, {max_x}, {max_y}, 25833)),
ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {max_y
    }::FLOAT, {x_scale}, {y_scale}, 0, 0, 25833),
'8BUI', {color_attribute}::INTEGER, 0) as rast
FROM arealbruk_flate
WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y},
     {max_x}, {max_y}, 25833)) AND color = 3
),
empty as (
SELECT st_asraster(
st_makeenvelope({min_x}, {min_y}, {max_x}, {max_y}, 25833),
ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {max_y
    }::FLOAT, {x_scale}, {y_scale}, 0, 0, 25833),
'8BUI', 0, 0) as rast
)
SELECT ST_AsGDALRaster(st_union(foo.rast, 'max'),'GTiff')
FROM (
SELECT rast FROM area
UNION SELECT rast FROM area2
UNION SELECT rast FROM empty) foo
```

B.4 Binary Water

```
WITH area AS (
SELECT st_asraster(st_intersection(geom, st_makeenvelope({min_x}, {min_y
ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {max_y}::FLOAT, {x
'8BUI', {color_attribute}::INTEGER, 0) as rast
FROM vann_flate
WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y}, {max_x}, {m
),
empty as (
SELECT st_asraster(
st_makeenvelope({min_x}, {min_y}, {max_x}, {max_y}, 25833),
ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT, {max_y}::FLOAT, {x
'8BUI', 0, 0) as rast
)
SELECT ST_AsGDALRaster(st_union(foo.rast, 'max'),'GTiff')
FROM (
SELECT rast FROM area
UNION SELECT rast FROM empty) foo
```

B.5 Multiclass Dataset

```
WITH area AS (
  SELECT st_asraster(st_intersection(geom, st_makeenvelope
      ({min_x}, {min_y}, {max_x}, {max_y}, 25833)),
        ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT
            , {max_y}::FLOAT, {x_scale}, {y_scale}, 0, 0,
            25833),
        '8BUI', {color_attribute}::INTEGER, 0) as rast
  FROM ar5_flate
  WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y
      }, {max_x}, {max_y}, 25833)) AND color = 3
),
areatype AS (
  SELECT st_asraster(st_intersection(geom, st_makeenvelope
      ({min_x}, {min_y}, {max_x}, {max_y}, 25833)),
        ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT
            , {max_y}::FLOAT, {x_scale}, {y_scale}, 0, 0,
            25833),
        '8BUI', {color_attribute}::INTEGER, 0) as rast
  FROM arealbruk_flate
  WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y
      }, {max_x}, {max_y}, 25833)) AND color = 3
),
roads AS (
  SELECT st_asraster(st_intersection(geom, st_makeenvelope
      ({min_x}, {min_y}, {max_x}, {max_y}, 25833)),
        ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT
            , {max_y}::FLOAT, {x_scale}, {y_scale}, 0, 0,
            25833),
        '8BUI', {color_attribute}::INTEGER, 0) as rast
  FROM veg_flate
  WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y
      }, {max_x}, {max_y}, 25833)) AND color = 1
),
buildings AS (
  SELECT st_asraster(st_intersection(geom, st_makeenvelope
      ({min_x}, {min_y}, {max_x}, {max_y}, 25833)),
        ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT
```

```
            , {max_y}::FLOAT, {x_scale}, {y_scale}, 0, 0,
            25833),
        '8BUI', {color_attribute}::INTEGER, 0) as rast
  FROM bygning_flate
  WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y
      }, {max_x}, {max_y}, 25833)) AND color = 4
),
water AS (
  SELECT st_asraster(st_intersection(geom, st_makeenvelope
      ({min_x}, {min_y}, {max_x}, {max_y}, 25833)),
        ST_MakeEmptyRaster({x_res}, {y_res}, {min_x}::FLOAT
            , {max_y}::FLOAT, {x_scale}, {y_scale}, 0, 0,
            25833),
        '8BUI', {color_attribute}::INTEGER, 0) as rast
  FROM vann_flate
  WHERE st_intersects(geom, st_makeenvelope({min_x}, {min_y
      }, {max_x}, {max_y}, 25833)) AND color = 2
),
empty as (
  SELECT st_asraster(
              st_makeenvelope({min_x}, {min_y}, {max_x},
                  {max_y}, 25833),
              ST_MakeEmptyRaster({x_res}, {y_res}, {min_x
                  }::FLOAT, {max_y}::FLOAT, {x_scale}, {
                  y_scale}, 0, 0, 25833),
              '8BUI', 0, 0) as rast
)
SELECT ST_AsGDALRaster(st_union(foo.rast, 'max'),'GTiff')
FROM (
SELECT rast FROM area
  UNION SELECT rast from areatype
  UNION SELECT rast from roads
  UNION SELECT rast from buildings
  UNION SELECT rast from water
  UNION SELECT rast from empty) foo
```