Vectorization of Scanned Raster Maps Using Deep Learning

Ruben Schmidt Mällberg

Fall 2017

TBA4560 - Specialization Project

Department of Civil and Transport Engineering
Faculty of Engineering Science and Technology
Norwegian University of Science and Technology

Supervisor 1: Terje Midtbø

Supervisor 2: Carl Christensen

Abstract

Old scanned zoning regulations are currently being vectorized using manual techniques in Norway. With the recent advances in image recognition and semantic segmentation using convolutional neural networks, there should be a possibility to use them for automatic vectorization of the regulations thus, lowering the cost and time spent doing manual work.

This paper gives an overview of the state-of-the-art regarding image segmentation using convolutional networks and vectorization techniques for automatic vectorization of raster maps. We also look at how the dataset needed to train a convolutional neural network can be automatically created by using already vectorized zoning regulations.

We show that the current techniques of vectorization need simple raster maps with uniform looks and clear polygons, qualities the zoning regulations do not have, and that the automatic creation of the dataset is not so simple as initially though because of the lack of information about the position of the vectorized polygons within the scanned zoning regulations.

Preface

This paper is the results of the specialization project TBA4560 as part of the study program Engineering and ICT with specialization in Geomatics. The project was conducted fall 2018 in cooperation with Spacemaker, a company that works with AI technology to maximize the value of building sites.

The project has given the author a thourough look at the state-of-the-art regarding automatic vectorization of raster maps using traditional approaches and deep convolutional networks that will serve as the basis for a master thesis in spring 2018.

Trondheim, 2017-12-20

Ruben Schmidt Mällberg

Contents

Chapter 1

Introduction

Raster to vector or vectorization is a central part of what GIS specialists do. Vectorization is the task of extracting vector layers from raster maps so that they can be used for further analysis, and is often a time consuming manual process. With the vast amount of raster maps available online, we are losing valuable information because we are unable to process them automatically.

There are a couple of software products on the market today concerning the problem of converting a raster map to a vector image such as GDAL Polygonize [29] and R2V [46]. Hovewer these require either very simple and well defined polygons or human intervention to successfully vectorize the raster map.

Problems occur when the rasters not only consists of isolated objects that are easy to distinguish but contains a spatial structure, overlapping geometries, and background layers. The multilayered nature of raster maps in addition to varying image quality makes automatic vectorization a really hard problem.

Deep convolutional neural networks (DCNN), are top performers of semantic image labeling [21] and can, therefore, be a possible solution to the vectorization problem. DCNN require large amounts of properly labeled training data. This is often generated manually for each case with, for instance, crowd-sourcing tools such as Amazon's Mechanical Turk [21].

In this paper, we will look at the state of the art in vectorization of raster maps and feature extraction with DCNN, look at the data needed to train the networks sufficiently and investigate the considerations one would have to take into account when implementing a network.

Chapter 2

Motivation

Digitizing and vectorization is a time consuming and expensive process [45].

In 2009, the Norwegian government made it statutory for all municipalities in Norway to have a digital zoning registers [19]. This has many benefits to society, both for the municipalities, the government, and the private sector. Benefits such as faster insights and sped up proceedings for building projects.

In June 1st, 2017, 354 out of 426 municipalities are registered to have digital zoning registers. The law does however not force the municipalities to vectorize the zoning regulations in detail, only to have them scanned in a digital format such as Portable Document Format (PDF) with the bounding limits of the plan vectorized. The detailed vectorization is then another step that has to be done in order to fully utilize the data in more advanced use cases, such as in GIS. See Figure 2.1 for an example of a zoning regulation in PDF format.

For the municipalities, the digitalization and vectorization is a process that is often done by external contractors. In a project in Telemark and Vestfold, the cost was estimated to be around 3000NOK for each plan, where the whole project consisted of 100 plans [2]. The cost savings in automating the vectorization of digitized zoning plans are thus economically significant.

Since the zoning regulations has to follow strict quality standards, the acccuracy of the output from the DCNN also has to follow these standards. With many zoning regulations already digitized and vectorized, we potentially have a lot of validated, accurate data that can bee used to generate training data for the DCNN. This gives us a great oportunity to investigate the performance of DCNN for vectorization of scanned maps.

There are a lot of approaches and network architectures that need to be examined and reviewed for this specific problem. This research will be the basis of the author's master thesis, where a practical implementation of the theory found in this research will be done.

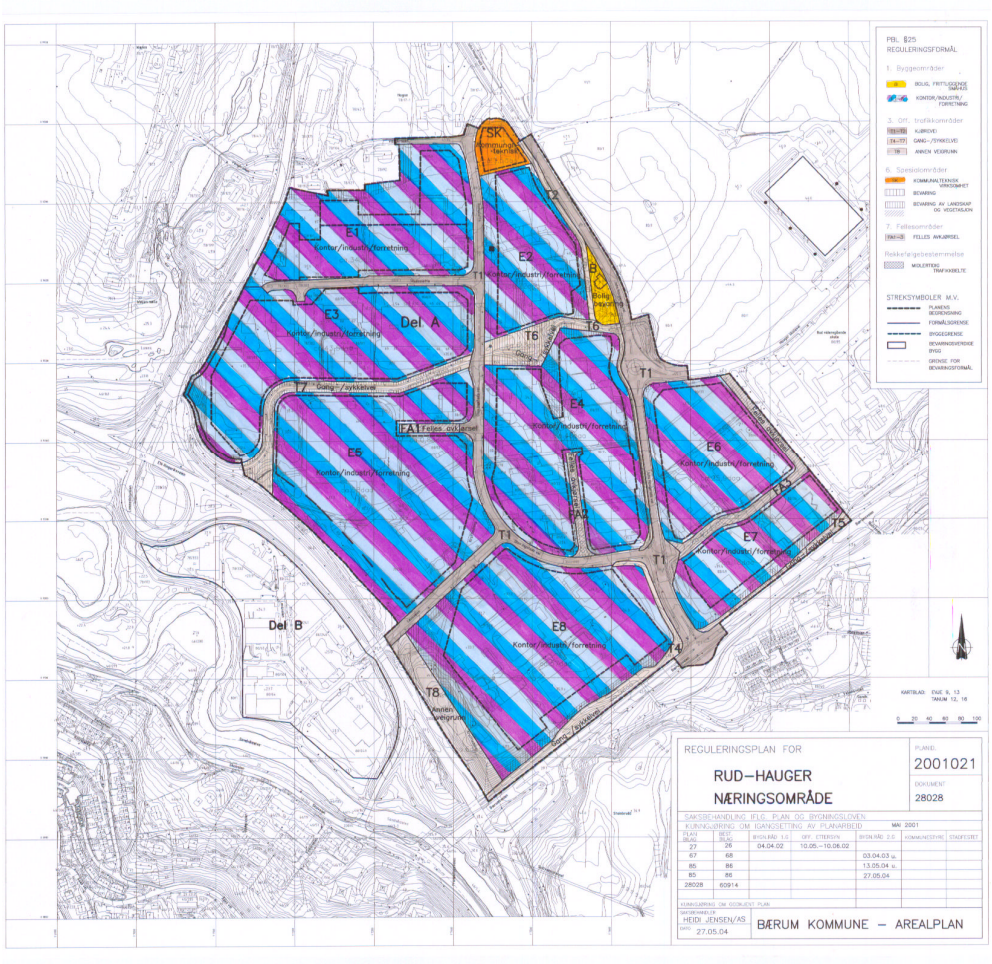

Figure 2.1: Scanned zoning regulation in PDF format from Bærum municipality.

Chapter 3

Vectorization

Vectorization, raster-to-vector conversion or image tracing is the conversion of raster graphics to vector graphics. To understand the reason for converting from raster to vector we need to look at some of the properties of both storage techniques.

Raster data is structured as an array of grid cells, also referred to as pixels. Each cell in a raster can be addressed by its position in the array, by row and column number. Since each pixel has it's own value, a raster can represent a range of spatial objects. A point can be represented by a single pixel, an arc represented by a sequence of pixels and an area as a collection of continuous pixels. Vector data is structured as a finite straight line segment defined by its endpoints. The location, or coordinates, of the endpoints, are given with respect to a coordinatization of the plane. The vector representation is not discretized in a grid space the same way as a raster but does follow an implicit grid structure as a result of the nature of computer arithmetic. Like the raster, the vector structure can represent multiple spatial structures. A point is given by its coordinates, an arc represented as a sequence of line segments, each consisting of start and end coordinates and an area represented by its boundary consisting of a collection of vectors.

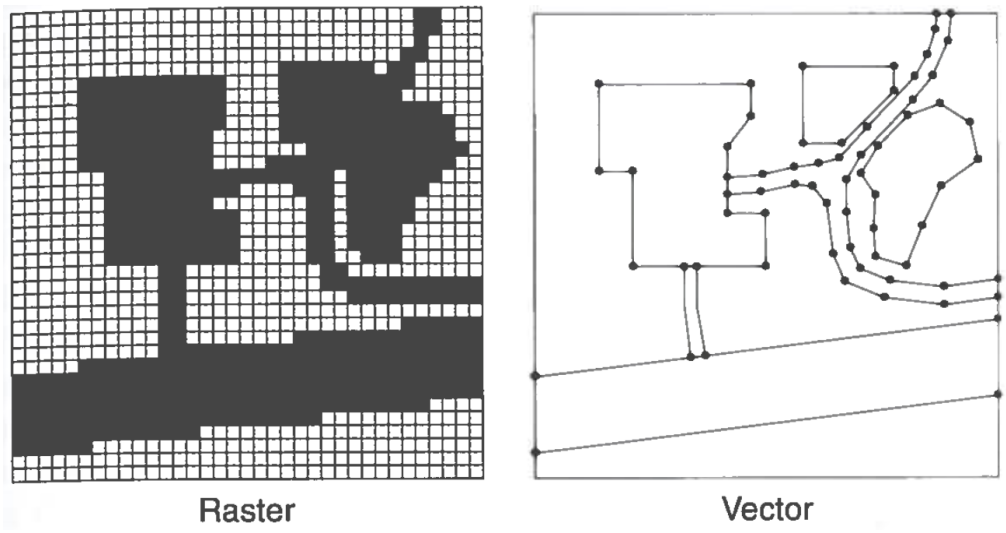

Figure 3.1: Raster and vector data
Source: Worboys [45]

There are multiple reasons for storing data in vector format: The geographic accuracy is higher since it is not dependent on grid size, it allows for efficient encoding of topology, an important aspect when doing analysis that utilizes topologic relations, such as proximity and network analysis and it allows for storage of attributes in the data, giving us another dimension of information and the storage size is smaller.

There are multiple different techniques for vectorization. We will now look at some of the well-known ones.

3.1 Hough Transform

The Hough Transform can isolate features of particular shape within images. The user must describe the desired feature in a parametric form, therefore the most common application of Hough Transform is to detect regular curves such as lines and circles. The technique is known to be tolerant of image noise and gaps in the feature boundary.

Hough Transform works by transforming each of the pixels into a straight line in a parameter space. The parameter space is described by the parametric form of features we are looking for. In the simplest case of a Hugh Transform, we want to detect straight lines. For this the normal form $r = xcos(\theta) + ysin(\phi)$ is used to represent the lines. When iterating the pixels in the image the algorithm looks if there is enough evidence that there is a straight line at that pixel, if there is, it calculates the parameters θ and ϕ at that point and adds it to an array. The array accumulates

all the points that belong to each specific θ and ϕ, thus the more points that are on the same parameters the more likely they belong to a line. The algorithm visits each pixel one time and is therefore linear with the number of pixels in the image.

3.2 Thinning

Thinning is a morphologic operation that is used to remove foreground pixels from an image, resulting in a simplified image with the same topological relations as the input image. It works by successively removing pixels from the boundary until there is not possible to remove more. The skeleton that is left contains the centerlines of the objects. This technique is mostly used on binary images. The iterative nature of the algorithm makes it computationally expensive.

3.3 Contour

This method aim at lowering the computational cost compared to thinning. The main idea is to find edges before calculating the middle point that is between two opposite parallel edges. A chain of middle points represents the medial axis. It has to use an edge detection algorithm before extracting the middle points between the edges. A problem with the contour based methods is how they deal with junctions. It is likely to miss junctions that are crossing at a small angle and it has difficulties with cross intersections where four lines meet [43]. It is thus not suited for vectorization of curves and crossing lines.

3.4 Run-graph

Run-graph is a semi-vector representation of the raster image [27]. It examines the raster in either row or column direction in what is defined as horizontal and vertical *runs*. A horizontal run is a maximal horizontal sequence of black pixels. A vertical run is a maximal vertical sequence of black pixels. After running, the run-graph is composed of edge areas that represent line segments and node areas and touching points that correspond to endpoints and junctions. The line extraction from the graph attempts to minimize the area of the node and maximize the length of the connected edges. Run-graph based methods are vulnerable to image noise and can cause unsatisfactory results at junction areas [17].

3.5 Mesh pattern

Mesh pattern was introduced by Lin et al. [24]. The idea is to divide the image into a mesh and check the distribution of black pixels at the border of each mesh unit. Using the patterns along the border, a control map is created. Lines are then extracted from the control maps by comparing them to characteristic patterns in a pattern database. Unknown patterns are labeled with question marks. These areas are analyzed pixel by pixel to determine where the line goes. In Figure 3.2 we can see the process of generating a mesh, extracting the control maps and filling the lines based on the analysis of control maps.

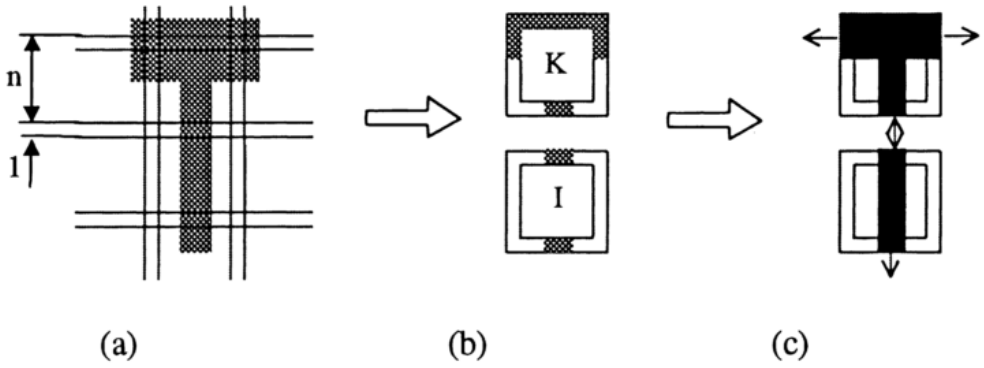

Figure 3.2: Mesh pattern line extraction. (a) image and mesh, (b) mesh pattern labels and control map, (c) lines extracted by analyzing the control map.
Source: Wenyin and Dori [43]

3.6 Sparse-pixel

The first of these algorithms was the Orthogonal Zig-Zag (OZZ) introduced by Dori [8]. The basic idea of the algorithm is to track a single pixel wide "beam of light" which turns orthogonally each time it hits an edge, the same way light travels in a fiber-optic cable. In the algorithm, the beam changes direction if it encounters white pixels or the length of the beam exceeds a predefined threshold. If the threshold is hit, two new beams are emitted orthogonally from the point. This can be seen in Figure 3.3. The midpoint of the runs are also recorded, used to reconstruct the line, corner correction and merger of crossing lines. Wenyin and Dori [44] improved the OZZ method with Sparse Pixel Vectorization (SPV). There were three main improvements: The procedure begins with a reliable medial axis point found by a separate procedure for each black area, a general following procedure is used for all cases of OZZ, i.e. Vertical, horizontal and diagonal following and a junction recovery procedure is applied whenever a junction is encountered during following.

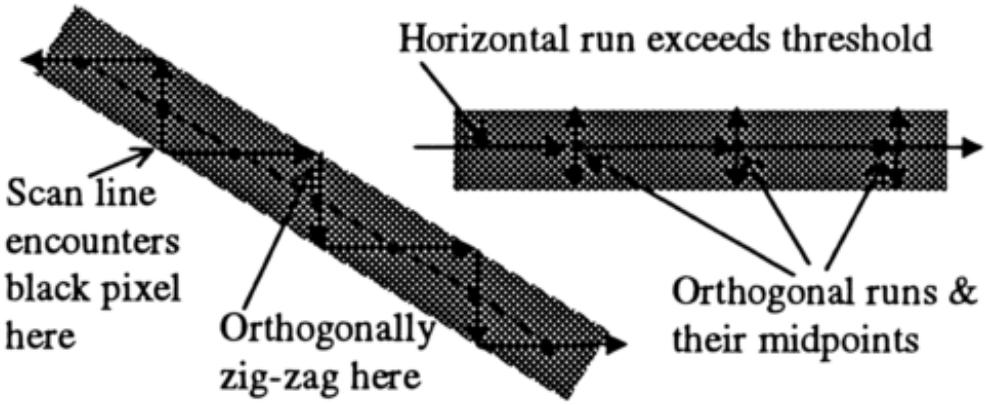

Figure 3.3: Orthagonal Zig-Zag
Source: Wenyin and Dori [43]

Chapter 4

Image segmentation with Deep Learning

In this chapter, we will look at the problem of image segmentation and the state of the art regarding segmentation and object detection using Deep Learning. We will look at both semantic and instance segmentation.

Semantic segmentation of images is one of the key problems in the field of computer vision. It is about making dense predictions inferring labels at the pixel level, assigning a class to each pixel with its enclosing object [10]. Taking it a step further, we get to instance segmentation, where we want to associate the classes with a physical instance of an object.

Both semantic and instance segmentation can be seen as giving us an understanding of an image at a higher level. This fine-grained control of an image greatly helps with scene understanding which is becoming more and more relevant with the increasing number of applications, such as self-driving cars and augmented reality.

We can see an example in Figure 4.1 where we see the difference between the two approaches. In the middle photo, all the chairs have the same classification, as chairs. In the right photo, we see that the chairs now are classified as chairs, but with a different class for each of them. We can see that instance segmentation is the combination of both object detection and semantic segmentation.

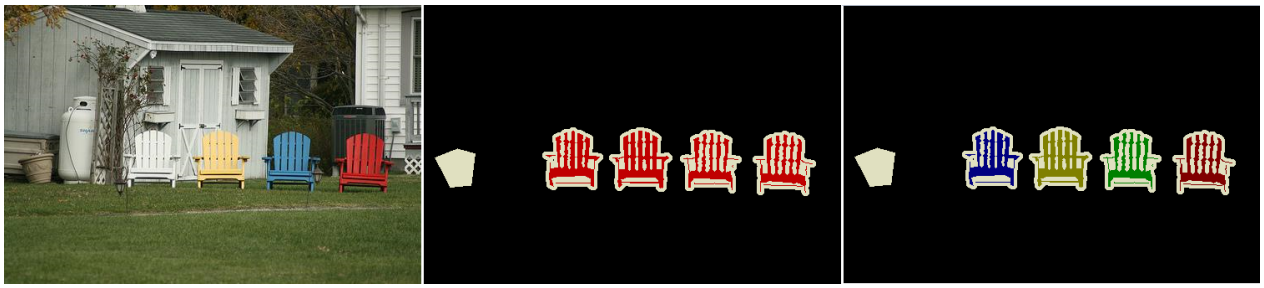

Figure 4.1: Left: input image. Middle: Semantic segmentation. Right: Instance segmentation. Source: PASCAL VOC [31]

4.1 Neural networks

To get a deeper understanding of how neural networks perform segmentation of images, we need to take a look at the fundations behind them and how they operate.

Neural networks are a computational model that shares som properties with the animal brain in which simple units called neurons are working in parallel with no centralized control unit [33]. The primary means to long-term information storage is in the weights between the units and updating them is the primary way the network learns new information.

A network is defined by the number of neurons, number of layers and the connections between the layers. One of the easiest architectures to understand is the feed-forward multilayer architecture viewed in Figure 4.2. It is a neural network with an input layer, one or more hidden layers and an output layer. The input layers feeds input, in the form of vectors, to the rest of the network. The number of neurons at the input layer often reflects the size of the input vector.

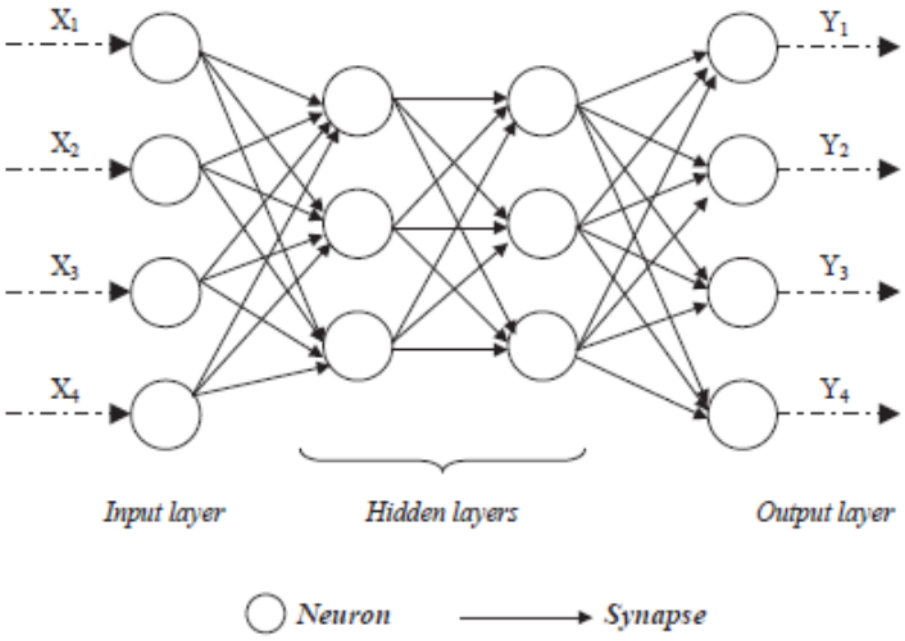

Figure 4.2: Structure of a multilayer feed forward network.
Source: Zangeneh, Omid, and Akram [49]

4.1.1 Artificial neurons

Each layer consist of one or more artificial neurons, also called nodes. 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. The activation functions are scalar-to-scalar functions that defines the output of the node based in the inputs, weights and bias. A model of an artificial neuron compared to a real one, can be seen in Figure 4.3.

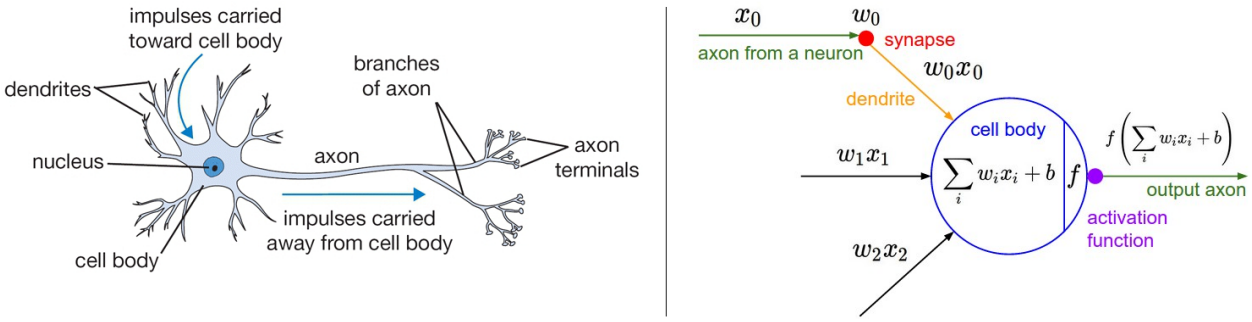

Figure 4.3: Structure of a real neuron compared with an artificial one.
Source: Karpathy [18]

4.1.2 Activation functions

The use of activation functions in the hidden layers add the ability for the network to learn non-linear functions. We will now take a look at some of the usefull activations functions that are used today.

Sigmoid

The sigmoid activation function has a characteristic "S"-shaped curve and can take variables of near infinite range and convert them to values between 0 and 1. It is good at reducing outliers and extreme values in the dataset. Expressed mathematical as:

$$a = \sigma(x) = \frac{1}{1 + \exp(-x)} \tag{4.1}$$

Tanh

A trigonometric hyperbolic function. Tanh can normalize input to the range of -1 to 1 and can therefore deal with negative numbers better than the Sigmoid. Expressed mathematically as:

$$a = \sigma(x) = \tanh(x) \tag{4.2}$$

Rectified Linear

Rectified Linear only activates a node if it is above some threshold. When the input raises above the threshold it has a linear relation to Equation 4.3. Nodes that use the rectifier are called Rectified Linear Unit or ReLU.

$$a = \sigma(x) = \max(0, x) \tag{4.3}$$

ReLUs are the state-of-the-art because of their proven usefulness in many situations and their ability to train better in practice than sigmoids. ReLU does not have the so-called problem of vanishing gradients either. Vanishing gradients is a problem that occurs when using gradient-based methods (explained in subsection 4.1.3) for learning, where large changes in the value of parameters from the early layers, does not have a big effect on the output, making the network lose its ability to learn. The reason for this happening is that some activation functions, such as sigmoids or tanh, forces the input space into small regions.

While removing the problem of vanishing gradients, ReLU introduces another one and that is the problem of "dying ReLU" [18]. This is a problem that occurs when a large gradient passes through the neuron causing the weight update to be so large that it causes the neuron to never activate again, that means that the gradient passing through the neuron will be forever zero.

4.1.3 Training

There are different forms of learning such as supervised, where we show the network what the correct answer is. Unsupervised, 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 paper as this is the type of learning we use when doing image segmentation.

In supervised learning, the network learns by training on a set of inputs and desired outputs. As inputs are passed through the network and outputs are generated, it learns by adjusting weights and biases causing some neurons to become smaller and some to become 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 some loss function that quantifies the correctness of the output from the network in regards 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 algorithm for the weight adjustment in neural networks is called *backpropagation*.

Backpropagation Learning

When the output from a neural network produces a large loss, we need to update the weights accordingly. A problem with multilayer neural networks, however, is that there are many weights connecting an input with an output, so it becomes difficult knowing what weights that affect the output. We need a clever way of finding what specific weight that contributes to the output. This is the problem backpropagation tries to solve. A high level understanding of backpropagation is that we use the chain rule to iteratively calculate the gradients for each layer. The steps of the algorithm is as follows:

1. Initialize network with random weights
2. Loop trough the training examples
3. Compute the network output for the current training example
4. Compute the loss with the loss function.
5. Compute the weight update for the output layer with the weight update rule:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times \alpha_j \times \Delta_i$$

   Where

$$\Delta_i = Err_i \times g'(input\_sum_i)$$

   and g' is the derivative of the activation function
6. Loop trough all the layers in the network all the way to the input layer and:
7. Compute the error at each layer with the propagation rule:

$$\Delta_j \leftarrow g'(input\_sum_i) \sum_i W_{j,i} \Delta_i$$

8. Update the weights leading in to the hidden layer with the update rule:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times \alpha_k \times \Delta_j$$

The term α is the learning rate, and belongs to the family of what we call *Hyperparameters* in machine larning.

Hyperparameters

The hyperparameters are what we tune to make the network train faster and better. The selection of these parameters are done to ensure that our network does not *overfit* or *underfit* the data. We say that our model is overfitted if it fits our training data too well but does not generalize enough over the entire dataset. We say our model is underfitted if it generalizes too much and is not able to fit the training set. The terms are illustrated in Figure 4.4. In the left image we see that the model does a bad job at approximating the function, it is underfitted. In the middle image we see that the model approximates the function well. In the right image we see that the model fits the training samples very good, but does a bad job at approximating the function, the model is overfitted.

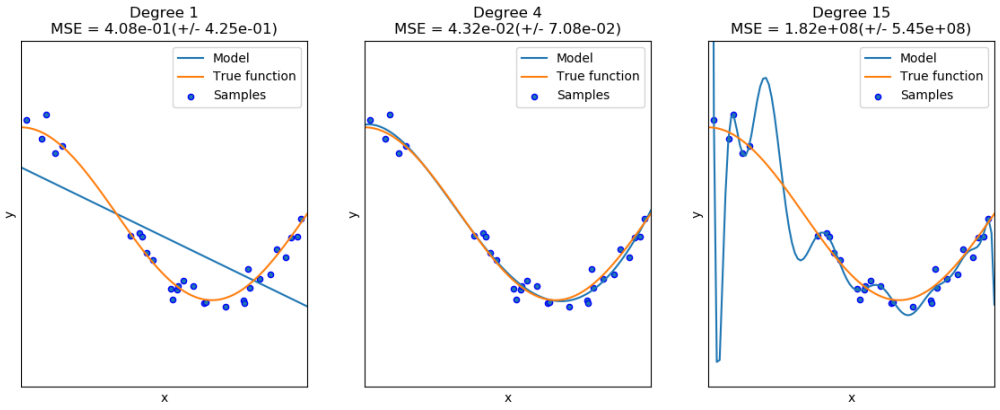

Figure 4.4: Left: Underfitted model. Middle: Appropriate fit. Rigth: Overfitted model.

The learning rate affects the amount we adjust the parameters during training. A large learning rate will make our parameters take large steps, thus saving time, but can cause us to overshoot the minimum of our loss function causing us to never find a minimum. A small learning rate causes us to take smaller steps and should help us reach the minimum, but can take a very long time to do so.

Another important hyperparameter is *regularization*. Overfitting often occurs when some weights have become very large and regularization is about reducing the effects of the large weights in the network. The perhaps most common form of regularization is L2 and is often implemented

as the term $\frac{1}{2}\lambda w^2$ that we add to the weights [18]. Another regularization, introduced in Srivastava et al. [38] is *dropout*. Dropout works by randomly dropping some of the neurons during training causing it to train on a "thinned" version of the net. Dropout has shown major improvements over other regularization tehniques [38].

4.2 Convolutional neural networks

Fully connected multilayer neural networks take inputs as a one-dimensional vector. When using an image as input, these vectors become very large. The reason for this is that we represent each pixel in the image as one value in the vector. If we are working with color images represented with 3 channels of RGB information, each of these also needs to be mapped. For a single 200x200 image this means $200 \times 200 \times 3 = 120000$ connections in only the first layer. This illustrates how bad the fully connected neural networks scale.

Convolutional neural networks, or CNNs, tackle the scaling problem by assuming inputs as images and model them as three-dimensional objects with image width, image height and color channels as the dimensions. At a high level, the architecture consist of an input layer, convolutional layers, pooling layers and fully-connected layers [33].

4.2.1 Convolutional layers

The convolutional layers, or CONV layers, are the core building blocks of a CNN. The layers consist of a set of learnable filters also called kernels. Each of the filters are small in regards to width and height but are always the same size as the input in regards to depth. The filter is applied to the input by sliding, or *convolving*, the filter accross the width and height of the input. At each position, the dot product between the filter and the input is calculated. The output is a two dimensional map that is called *activation map*. This process is illustrated in Figure 4.5. For each of the filters in the CONV layer we get such a map and stack them in the depth direction, this in turn represents the output of the layer.

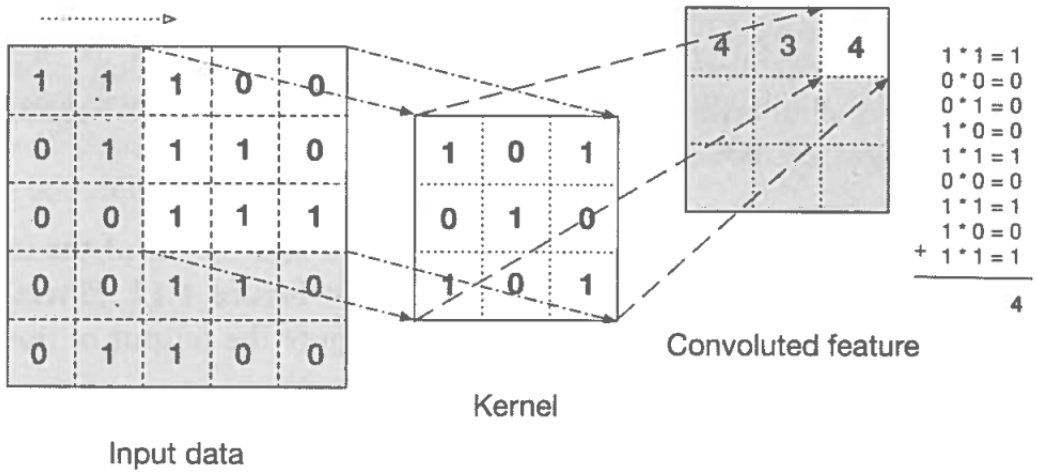

Figure 4.5: The convolution step.
Source: Patterson and Gibson [33]

The network will learn filters that causes the node to activate when certain visual features are seen, for instance an edge. Deeper into the network we will se 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 DCNN can bee seen in Figure 4.6 where we see filters learned in the first convolutional layer in an eight layer network [21].

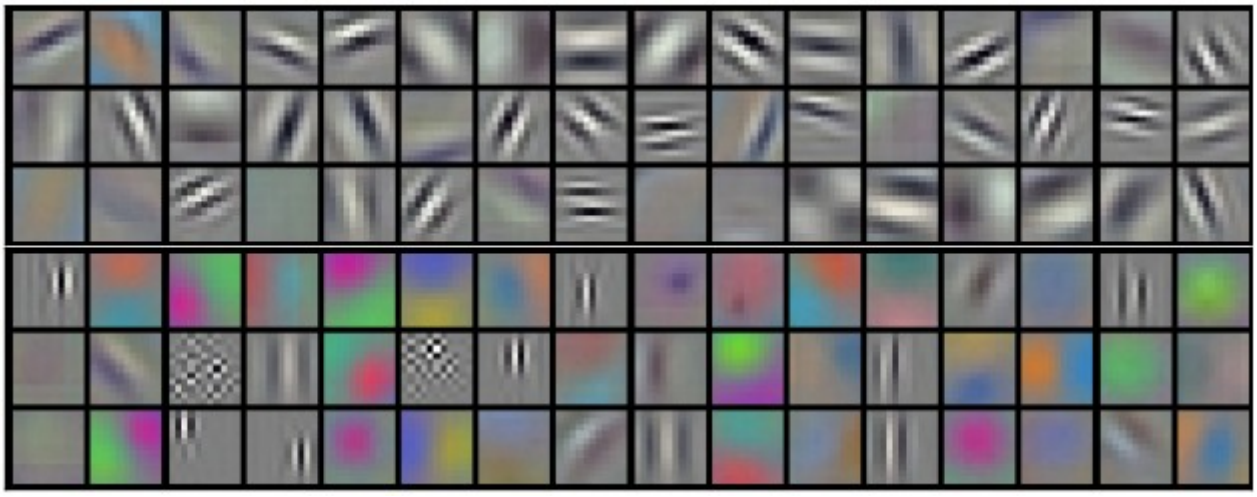

Figure 4.6: 96 learned filters in the first convolutional layer.
Source: Krizhevsky, Sutskever, and Hinton [21]

If we imagine that we freeze the filter as it is convoluted accross the input, a single step and its

calculation, can be viewed as the output from a neuron. The activation map then represents a sheet of neurons with each of the neurons looking at a small part of the input, not knowing anything about the rest of the image. This feature is called *local connectivity* and is an important part of how the network keeps the number of parameters smaller then a regular neural network. All the neurons in the sheet also share parameters, since it is the same filter that did the calculation. This is the concept of *shared paramters* and is the other important part of how CNNs keep the number of paramters low.

4.2.2 Pooling layers

Another way to reduce the number of parameters in the network, is the use of pooling layers. Pooling layers essentially reduces the input size by downsampling the input with different pooling functions. The most common operation is *max pooling* with a 2x2 filter with a stride of 2 that reduces the size by two in the height and width dimension [18].

Chapter 5

Related work - Image segmentation

In this chapter, we will look at some of the previous work done in the field of image segmentation with neural networks.

During the last ten years, we have seen many important advances when it comes to the architecture of deep networks for image segmentation. AlexNet [21], VGGNet [37], GoogLeNet [40], ResNet [13], ReNet [42] and the very recent CapsNet [36] are all examples of such advances. In the next section, we will look at the main points from each of these networks.

5.1 Important architectures

5.1.1 AlexNet

Achieving first place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [35] in 2012 with a top-5 test accuracy of 84.6% by a margin of 10% to the next competitor, AlexNet pioneered DCNN in image classification. The network consisted of a total of eight-layer, five convolutional layers with max-pooling and three fully-connected layers. All the layers used ReLU as activation. To reduce overfitting they used dropout. Figure 5.1 shows the architecture of the network.

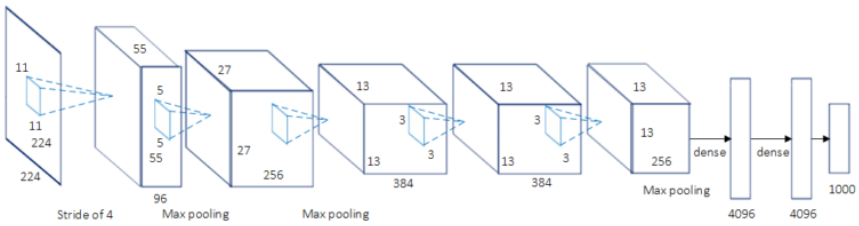

Figure 5.1: AlexNet architecture
Source: Krizhevsky, Sutskever, and Hinton [21]

5.1.2 VGG

The Visual Geometry Group (VGG) model was introduced by the Visual Geometry Group at oxford university. In their paper, they propose multiple different architectural configurations with weight layers ranging from 13 - 16 layers deep. The most interesting is the model with 16 weight layers. It was submitted to ILSVRC 2013 and managed to get a top-5 test accuracy of 92.7%. In Figure 5.2 we can see that the architecture makes us of more layers with small receptive fields rather than a few layers with large receptive fields.

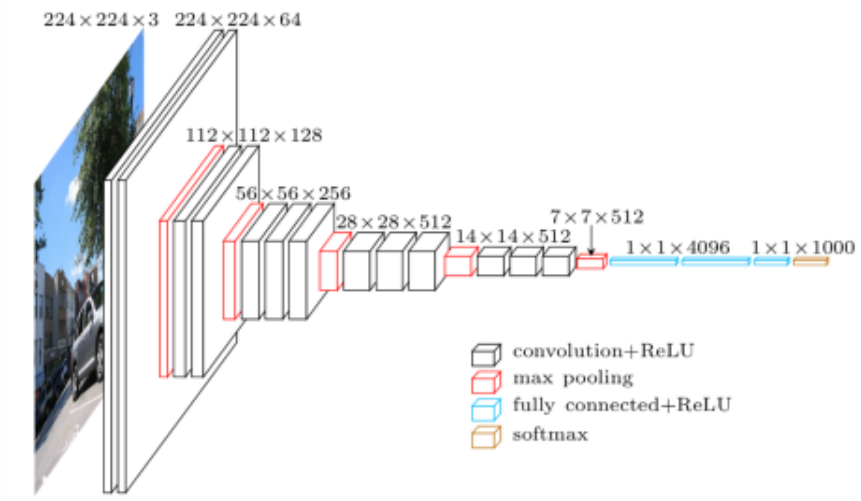

Figure 5.2: VGG 16 architecture
Source: Cord, Durand, and Thome [7]

5.1.3 GoogLeNet

This network won the 2014 ILSVRC challenge with a top-5 test accuracy of 93.3%. The network introduced a new architectural concept called the *inception* model (see Figure 5.3). The model is essentially a new mini-network with a pooling operation, large convolution layers, and smaller convolution layers. They proposed the use of small 1x1 convolution layers to reduce the complexity before the large convolution layers to keep the parameters and computational cost under control. This showed an increase in speed ranging from 3-10x faster than similar networks without the inception module.

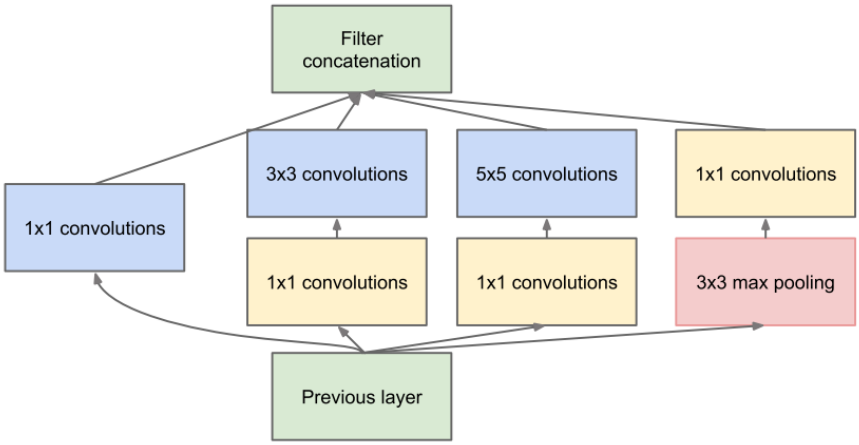

Figure 5.3: Inception module
Source: Szegedy et al. [40]

5.1.4 ResNet

ResNet won the 2015 ILSVRC, with a top-5 test accuracy of 96.4%. The network is known for its depth of 152 layers and a new kind of building block called residual block. The residual block contains two paths between the input and the output where one of the paths serve as a shortcut connection to the output (see Figure 5.4) essentially copying the input to the output layer. A big problem with very deep networks is that they are hard to optimize. When the depth of the network increases, the accuracy gets saturated. This is called *degradation* and is the problem that the residual blocks are addressing by forcing the network to learn on top of already available input.

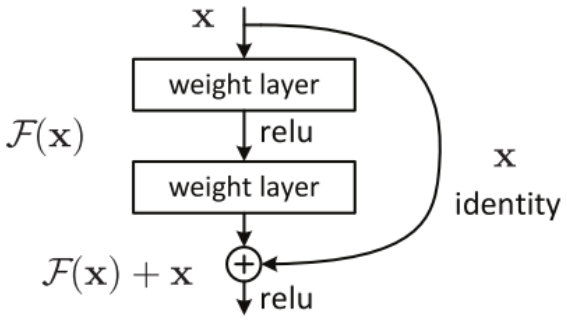

Figure 5.4: Residual block.
Source: He et al. [13]

5.1.5 CapsNet

Released November 2017, this is a very recent advancement in neural networks. It introduces a new type of neural network based on *capsules*. A capsule can be seen as a new type of neuron that instead of working on scalar values, encapsulates all the information about the state of the feature they are detecting in vector form.

It addresses the issue that CNNs are not good at generalizing new viewpoints. They are good at generalizing to translation, but other affine transformations have shown to be difficult to learn. In the paper the network is testen on the MNIST dataset. The results show a significant perfomance increase in the case of overlapping digits.

Another important discovery is that the use of capsules makes the network capable of learning to state-of-the art perfomance with a fraction of the data that a traditional CNN need.

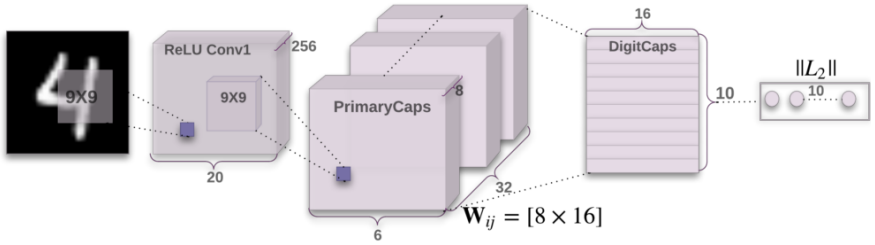

Figure 5.5: CapsNet with 3 layers.
Source: Sabour, Frosst, and Hinton [36]

5.2 Image segmentation

Many of the previous network architectures described in section 5.1 are predicting labels of what the images contain and not where and what part of the image the labels are to be found in. Image segmentation is about assigning a class to each pixel with its enclosing object, also called dense predictions, so we need output from the networks that are spatial maps instead of classification scores. In this section, we will review important networks that are specialized in image segmentation. A benchmark often used for image segmentation is the PASCAL Visual Object Classes Challenge (VOC) [9], that provides standardised image data sets for object class recognition. We will therefore evaluate the networks with the scores in VOC when applicable.

5.2.1 FCN

Fully Convolutional Network (FCN) by Long, Shelhamer, and Darrell [25] can be seen as a common forerunner for semantic segmentation with convolutional networks [10]. FCN adopted the contemporary deep classification nets AlexNet, VGG and GoogLeNet architectures we saw in section 5.1 to make dense predictions at the pixel level. It is important to note that they not only reused the architecture but used the pre-trained classification models as a starting point. The network replaced the fully-connected layers with convolutional ones, noting that the fully-connected layers could be seen as convolutional ones with kernels (filters) that cover the entire input region. This allowed segmentation maps to be generated from images of any size. Because of all the pooling operations in CNNs, a technique called *deconvolution* [50] was used to upsample the coarse output to dense pixels. Deconvolutional layers can learn interpolation functions the same way the network learns weights. Skip connections similar to the ones we saw in ResNet, are also included to give the deeper layers higher resolution feature maps. It reached a score of 62.2 in the VOC2012 challenge.

5.2.2 SegNet

SegNet by Badrinarayanan, Kendall, and Cipolla [1] uses an Encoder-Decoder architecture. One can also view FCN as this type of architecture, where the downsampling is the encoding part and the deconvolution is the decoder part. The difference between these networks is in the decoding part of the architecture. In SegNet more shortcut connections are added, however instead of copying the input to the output of one layer, each upsampling layer corresponds to a max-pooling layer in the encoding part and the indices from the max-pooling layer is copied to the upsampling layer. This can be seen in Figure 5.6 where we see the blue lines as shorcut

connections to the upsampling layers. The network was not benchmarked on VOC2012 in the paper but the leaderboard [32] shows that it reached a score of 59.9.

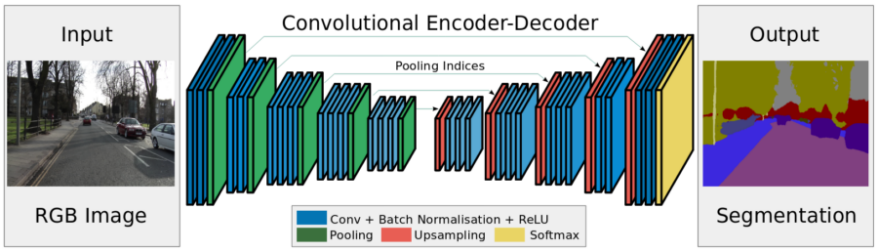

Figure 5.6: SegNet architecture.
Source: Badrinarayanan, Kendall, and Cipolla [1]

5.2.3 Dilated Convolutions

A problem with pooling layers is that they remove more context and resolution from the image the deeper we get into the network. For segmentation we need contextual reasoning and full-resolution output and Yu and Koltun [48] try to solve this problem with dilated convolution layers. Dilated convolution layers can increase the receptive field of a convolutional filter exponentially while the number of parameters grow linearly. This is illustrated in Figure 5.7 where we see that the receptive field, indicated in green, grows expoentially compared to the parameters, indicated as red dots. The network showed state-of-the-art perfomance with a simpler architecture, based on VGG-16 [37], than the competition, scoring 71.3 in VOC2012 with their basic network.

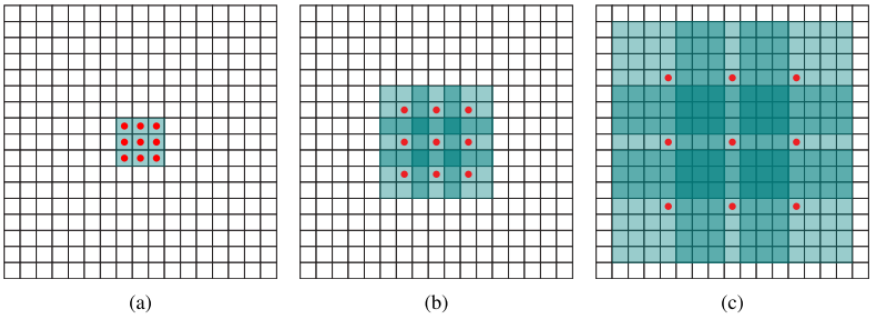

Figure 5.7: Diluted filters. (a) 1-dialted, (b) 2-dialated, (c) 3-dialated.
Source: Yu and Koltun [48]

5.2.4 DeepLab (v1 & v2)

DeepLab v1 (2014) [5] and DeepLab v2 (2016) [3] both used dilated convolutions though they refer to them as *atrous convolutions*. They use fully-connected Conditional Random Fields (CRF) to capture fine-grained details in images as proposed by Krähenbühl and Koltun [20]. CRF can be used to combine the class scores from deep CNNs with the low-level information captured by the local interactions of pixels and edges [5] and is used in DeepLab v1 as a post-processing method. As we can see in Figure 5.8 the effect of using CRF is significant in regards to detecting details in the image. DeepLab v1 used VGG-16 as a starting point for their architecture and got a score of 71.6 at VOC2012 beating the runner-up by a margin of 7.2%.

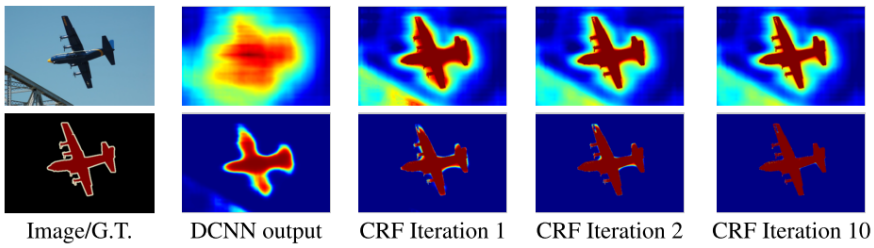

Figure 5.8: Score map (input before softmax) and belief map (output after softmax).
Source: Chen et al. [5]

Deep CNNs can represent different object scales by training on datasets that contain objects of varying size. However explicitly accounting for scale can improve the networks ability to handle large and small objects [30]. DeepLab v2 introduces a new technique to handle multivariate scale, called *atrous spatial pyramid pooling* (ASPP). It uses multiple parallel atrous convolutional layers with different sampling rate/direction to improve the networks ability to deal with objects of different scale. DeepLab v2s best architecture used a pre-trained ResNet-101 (ResNet seen in subsection 5.1.4, that has 101 layers) with atrous convolutions, ASSP, and CRF that got a score of 79.7 at VOC2012.

5.2.5 DeepLab v3

In DeepLab v3 ,Chen et al. [4] introduce an improved model of ASSP that involves concatenation of image-level features, a 1x1 convolution and three 3x3 atrous convolutions with different rates as seen in Figure 5.9.

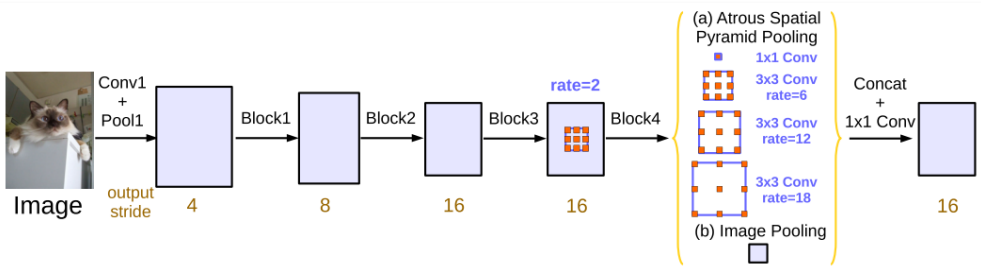

Figure 5.9: Improved version of ASSP, augmented with image-level features.
Source: Chen et al. [4]

Cascading modules was also proposed. A technique where the last ResNet block is duplicated multiple times and modified with atrous convolution Each block consists of three atrous convolution layers as seen in Figure 5.10.

Both methods were tested, but the new ASSP performed better and was selected for use in the final model. Using a ResNet-101 pre-trained on ImageNet dataset, DeepLabv3 reached a perfomance of 85.7 on VOC2012. Using a ResNet-101 pre-trained on both ImageNet and JFT-300M dataset (internal dataset at Google with over 300M images [14][39]) , DeepLabv3-JFT got a score of 86.9 on VOC2012.

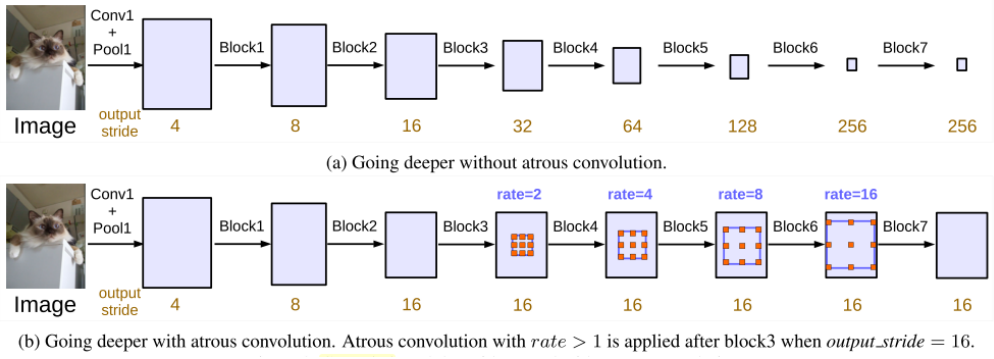

Figure 5.10: Cascaded modules.
Source: Chen et al. [4]

Chapter 6

Related work - Rastermap Vectorization

Image segmention has come a long way on images of normal everyday objects with DCNN. However none of the reviewed papers use their networks on raster maps. Not a lot of work has been done in regards to vectorization of raster maps using neural networks, and the author only found one paper on the subject. However there has been a fair amount of research done on other image analysis techniques.

6.1 Non-artifical intelligence methods

6.1.1 Color image segmentation in historical topographic maps based on homogeneity

Leyk and Boesch [22] presents a color image segmentation of raster maps from the 19th century suffering from poor quality with a clustering technique using the local image plane, frequency domain an color space. The goal of the color image segmentation is to reduce the color values to fit the original colors used when printing the map. The method managed to segment lines, symbols and areas that belong to different color layers, however, there were still some minor classification errors that had to be solved manually. Results from the color segmentation can be seen in Figure 6.1. Note that the output is not a raster image, but a raster image with a lower number of colors.

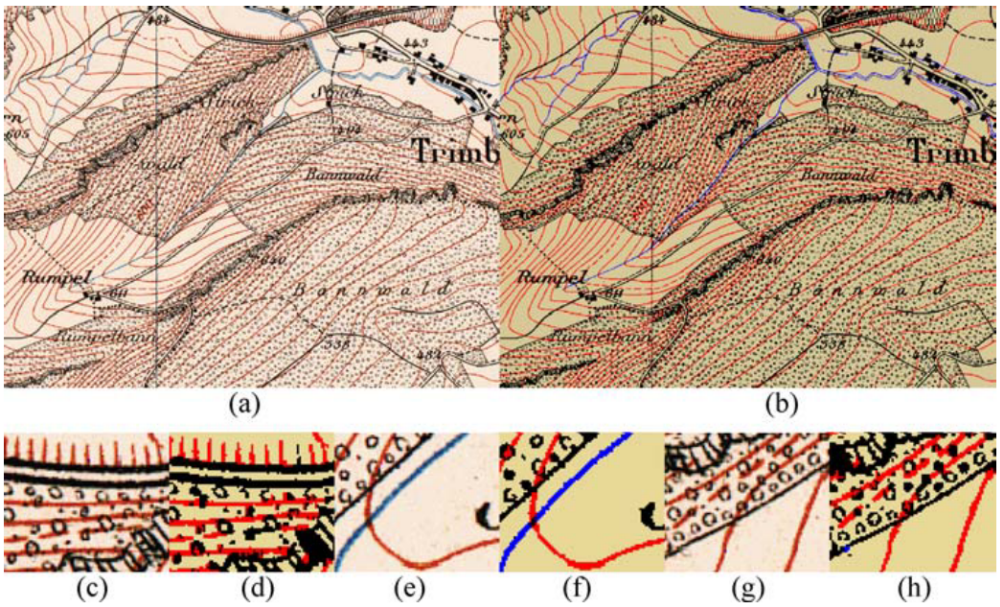

Figure 6.1: Color segmentation. (a) original map, (b) segmented map, (c-h) highlighted areas showing critical areas of the original.
Source: Leyk and Boesch [22]

6.1.2 Towards a comprehensive methodology for automatic vectorization of raster historical maps

Iosifescu, Tsorlini, and Hurni [16] use open-source solutions to vectorize historical maps from the 19th century. Their procedure consists of five steps: Scanning of the map, georeferencing the map, pre-processing of an image to clean artifacts, automatic vectorization, automatic cleaning of the results. The authors note that the pre-processing step and scan quality are the most crucial for the performance of the vectorization and have to be customized for each set of raster maps. The pre-processing consist of RGB channel processing, conversion to binary images and cleaning. By processing the RGB channels in the image, different features on the map can be highlighted.

The pre-processed image is then converted to vector format with Geospatial Data Abstraction Library [29] (GDAL)'s polygonize and contour methods. The results are acceptable when taking the quality of the maps into consideration as seen in Figure 6.2.

Figure 6.2: The final result of the vectorization of buildings, rivers and contours.
Source: Iosifescu, Tsorlini, and Hurni [16]

6.1.3 Historical map polygon and feature extractor

Giraldo Arteaga [12] vectorize polygons from old historical maps of building footprints in New York by using a procedure consisting of multiple open source tools. The tools in use are: GDAL [29], OpenCV [28], GIMP [11], R [41] and ImageMagic [15]. They propose a process consisting of three main steps: Raster image thresholding, rough polygon extraction and polygon analysis and simplification. There is a manual configuration step for each uniform set of map sheets where one has to enumerate the distinct colors in the map. The maps they are working on in the paper look very similar and therefore they only need to do this process one time for all the sheets. The input rasters can be seen in Figure 6.3 where we also see the result obtained from the process.

They state that the results will decrease the time spent on vectorizing the maps, but that the results do contain errors. These errors they propose to solve via crown-sourcing tools.

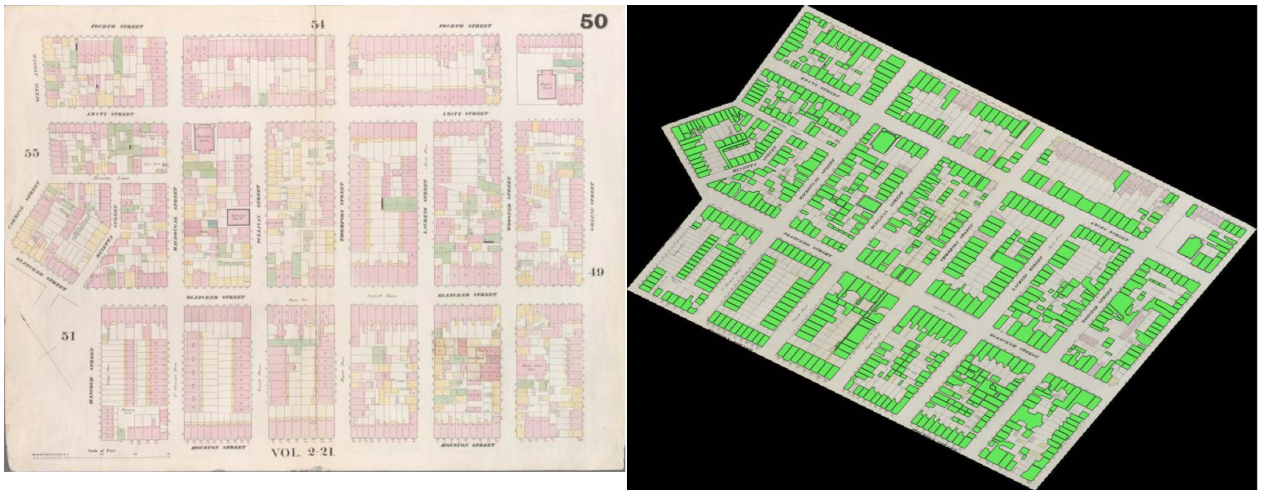

Figure 6.3: Left: Input maps. Right: Output from the process
Source: Giraldo Arteaga [12]

6.1.4 A general approach for extracting road vector data from raster maps

Chiang and Knoblock [6] present a semi-automatic technique for road extraction from raster maps with a system they call *Strabo*. The system consists of two components: Road layer extraction and road layer vectorization. In their components, they use techniques such as mean-shift, k-means and Hough Transform. They compare they systems performance against R2V [46]. Strabo performed better in 58.3% of the cases. Generally, their road vector lines manage to stay in the center of the roads more than R2V. This can also be seen in Figure 6.4. A limit of their system is that it struggles to correctly label roads that are very thin on the map. Their technique works on multiple different types of maps.

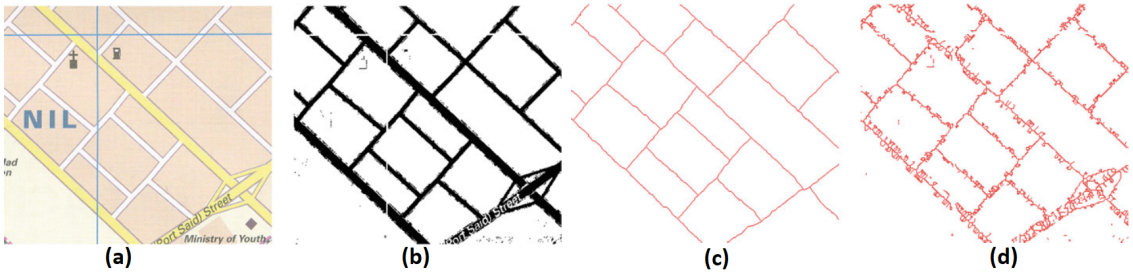

Figure 6.4: Vectorization results. (a) input map, (b) extracted road pixels, (c) Strabo results, (d) R2V results.
Source: Chiang and Knoblock [6]

6.1.5 Guided Superpixel Method for Topographic Map Processing

Miao et al. [26] propose a *superpixel*[34] approach to extract height curves from raster maps. Their method, named *Guided Superpixel Method in Topographic Map*(GSM-TM), consists of three parts: Linear feature extraction, boundary detection, and guided watershed transform. For the linear feature extraction, they merge two negative and two positive Gaussian filters. For the boundary extraction, they use a technique of color boundary detection proposed by Yang et al. [47]. The guided watershed transform was introduced since standard watershed is very sensitive to weak boundaries. The guided part consist of modifying the boundaries obtained in the boundary detection step with the lines obtained in the linear feature extraction to make the boundaries clearer before the watershed transform. The results show that the proposed GSM-TM method performs better than the other superpixel algorithms they compare with, also seen in Figure 6.5.

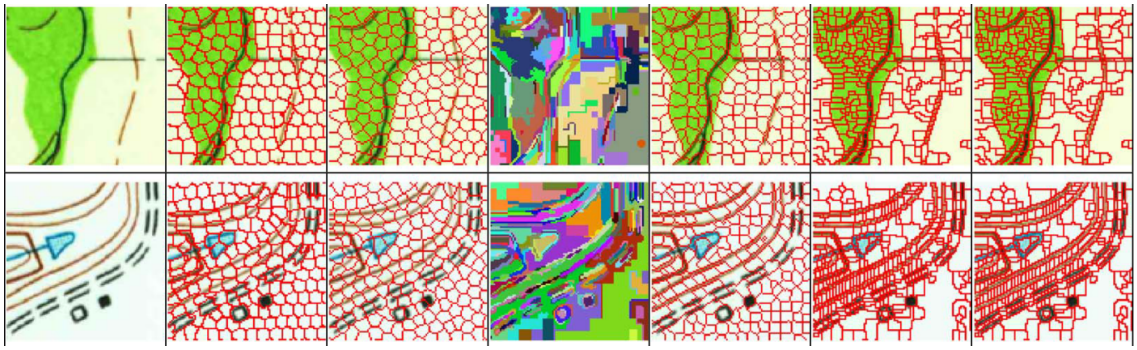

Figure 6.5: Results of compared methods. From left to right are the original map and the results from SLIC, NC, GS, Turbopixel, WT and GSM-TM
Source: Miao et al. [26]

6.2 Artificial intelligence methods

6.2.1 VecNET

VecNet proposed by Karabork et al. [17] in 2008 is one of few examples of vectorization of cartographic raster maps using a neural network. The authors present a three-step process consisting of thinning, line tracking with ANN and simplification. The main goal of the network is to find the critical points of objects, that is, to find breakage points of lines. They use an ANN with an input layer, a hidden layer and an output layer to classify. The training set is very small with only 16 samples. The output layer is a single vector with size 12, where the 8 first places represent an

8-way chain code of directions (the direction the line is following) and the last four represent a prediction of where the next pixel is going to be. It evaluates if the point is critical by checking if the last 8-way direction is different from the one currently calculated. If the point is critical, they store it. The algorithm is tested on a single raster map only consisting of lines and does not perform better than sparse pixel SPV, but manages to get acceptable results.

Chapter 7

Implementation

7.1 Creating the dataset

DCNN make use of vast amounts of data for training. The MNIST dataset contains 60.000 training examples and 10.000 test examples, the Pascal VOC 2012 segmentation dataset [31] has a total of 9993 segmented images, the Microsoft COCO dataset [23] has over 200.000 segmented images and Google's internal JFT dataset [14] has over 300 million images with labels. There is a general agreement in the field on DCNNs that the revolution in semantic segmentation is a product of the large datasets available [39].

Most of the reviewed networks build on pre-trained models to save computational time and to focus on training the network at the task at hand. Using pre-trained networks also lowers the need to have a large dataset available. In the case of zoning regulations, there does not exist a dataset that has been used or can be used to train the network. This must, therefore, be created.

As stated in chapter 2, 354 out of 426 municipalities are registered to have digital zoning registers. This implies that they have PDF's with scanned raster regulations openly available and published online. The PDF files have varying quality and some of them can be very old as seen in Figure 7.1. Some of these municipalities also have vectorized their regulations. It can, therefore, be possible to use the already vectorized regulations to create a dataset that can be used for training the neural network.

Figure 7.1: Samples of the zoning regulations.

A problem with this is that the PDFs are not georeferenced, that is, we do not know the coordinates of the corners of the PDF. A procedure that connects the boundary of the vectorized zoning areas to the same area within the PDF is therefore needed. The problem can be seen in Figure 7.2 where we see the vectorized bounding area on the left and the scanned zoning regulation PDF on the right. A possible approach is to use a computer vision framework such as OpenCV [1] and its feature detection library to find the translation between the images. After the translation is found it would be possible to generate a training image the size of the PDF and with proper labels. This procedure does however add a possible source of error that has to be examined thoroughly.

[1] Link: https://opencv.org/

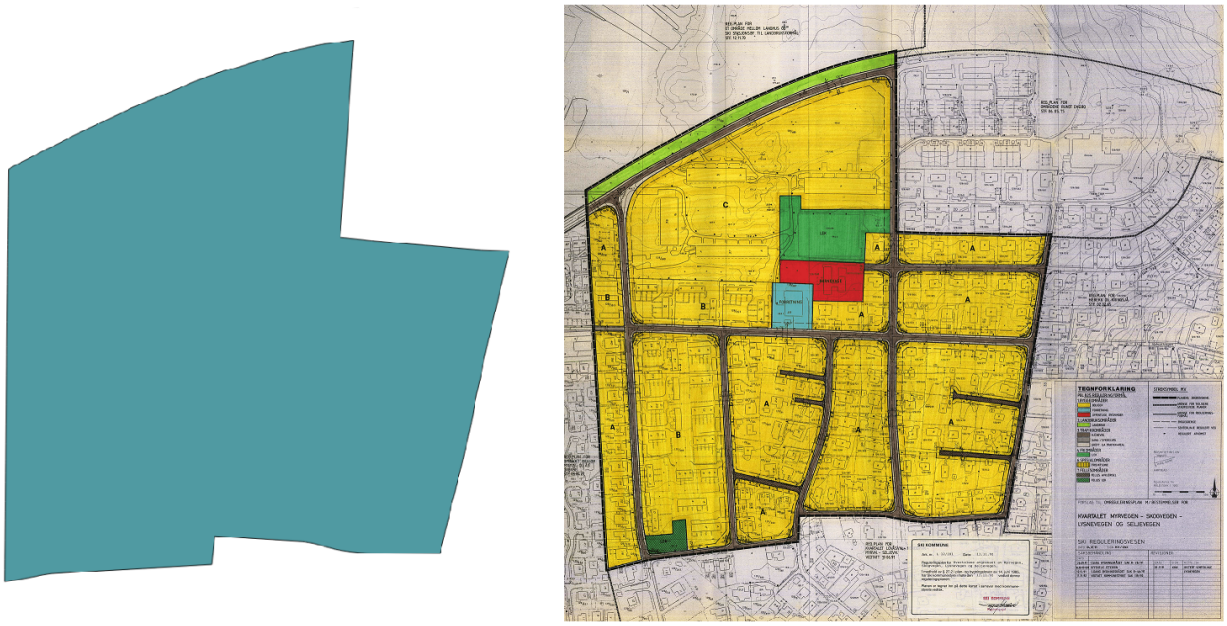

Figure 7.2: Vectorized bounding area on the left and scanned zoning regulation og the right.

Another approach is to create the dataset with a micro-tasking tool, such as Amazon Mechanical Turk [2], that allow you to outsource the work on the internet. This is for instance how the Microsoft COCO and ImageNet datasets were created [23]. This can however be a very time consuming process as seen in the creation of COCO, where the segmentation required 22 worker hours per 1000 images. One could argue that the time spent creating the training data, could be used to vectorize the rest of the regulations instead of generating training data. When outsourcing the work, one would also need to do quality checks of the data and prepare the data so that it is an easy task for the participants to do, as they do not neccesarily have any knowledge or skills regarding vectorization.

7.1.1 Feature matching with OpenCV

There was done some initial testing of feature detection with OpenCV. The method tried was brute force matching with Oriented FAST and Rotated BRIEF (ORB) descriptors with greyscale images and greyscale thresholded, contoured images. Since proper testing of multiple alternatives is outside of the scope of this paper, this can not be viewed as absolute results.

The results from the greyscale images can be seen in Figure 7.3 and as we can see, the algorithm

[2] Link: https://www.mturk.com/mturk/welcome

did not manage to find any matches. The same method was tried when using contour detection first, as seen in Figure 7.4 the results did not become better.

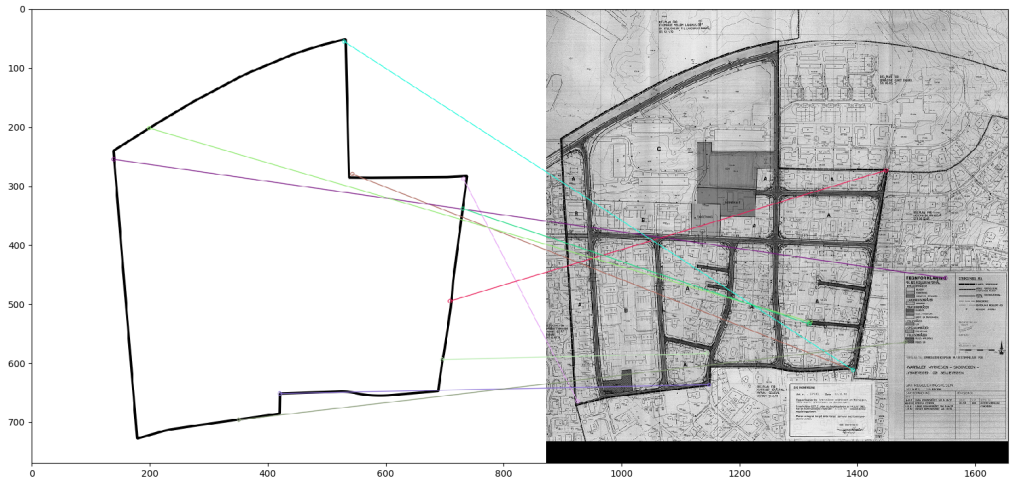

Figure 7.3: Vectorized bounding area on the left and scanned zoning regulation og the right.

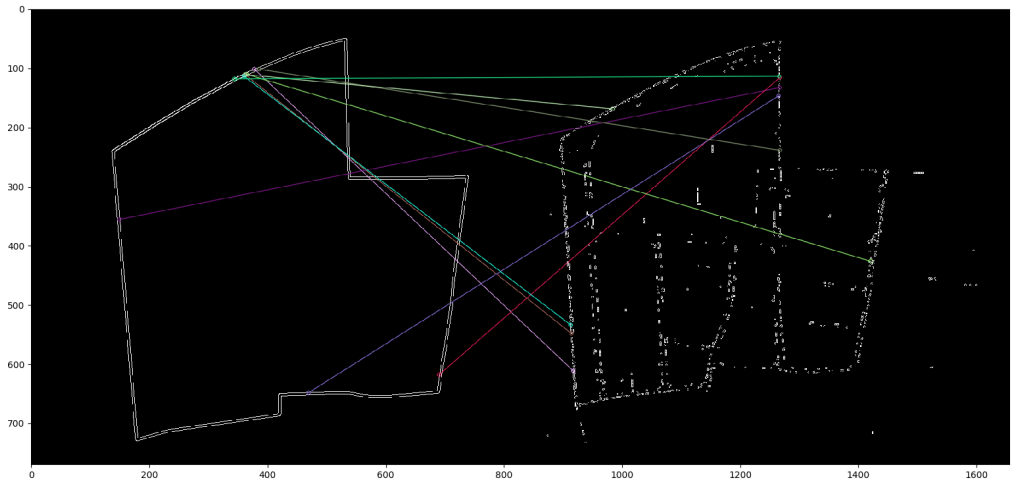

Figure 7.4: Vectorized bounding area on the left and scanned zoning regulation og the right.

7.2 Deep learning frameworks

There exist multiple deep learning frameworks that could be used to implement the network such as Tensorflow [3], Keras [4], CNTK [5], Caffe [6] or Torch [7]. Every framework is different, some are written for a specific programming language others provide lower or higher levels of abstraction. One of the frameworks that provide the highest level of abstraction is Keras. Keras was developed with a focus on enabling fast experimentation and offers a simple API written in python. It has the ability to use either Tensorflow, CNTK or Theano (recently retired) as the underlying computation engine. With Keras, the creation of a DCNN can be done in a few lines of code and a wide variety of pre-trained networks and example architectures comes prebuilt with the framework. This makes it a good candidate for the framework to be used when implementing the network proposed in this paper.

7.3 Computational power

Training large networks demand a considerable amount of computing power. In the early days one had to consider the cost of buying specific hardware, but with the emerge of deep learning machines in the cloud with large amounts of memory and computing power, this is not neccesary anymore. As an example, the latest generation smallest computational instances on Amazon Web Services, P3.2xlarge, cost $1.3 per Hour at the time of writing. These instances has 1 Nvidia Tesla v100 GPUs with 16 GB of GPU memory, 8 virtual CPU cores and 64 GB of ram. These instances should fit the work proposed in this paper very well.

[3] Link: https://www.tensorflow.org/
[4] Link: https://keras.io/
[5] Link: https://github.com/Microsoft/cntk
[6] Link: http://caffe.berkeleyvision.org/
[7] Link: http://torch.ch/

Chapter 8

Discussion

As we can see from the previous work, little research has been done in regards to using DCNN to vectorize scanned raster maps. Many of the traditional algorithms used for vectorization seen in chapter 3, specialize in the recognition of lines or known shapes. Most of the related work is also focusing on the extraction of linear features, such as contour lines and roads. DCNN show great ability to learn and segment images if trained properly and should, therefore, be able to vectorize raster maps if one could generate a dataset with training data.

Looking at the more traditional approaches not using neural networks to solve the problem, we see some interesting results. The benefit of the image analysis approach is that it does not require any training data and relies solely on the analysis itself. The downside, as seen in many of the reviewed papers is that one often needs to tune the algorithm and images for each specific case. For instance one of the steps in Iosifescu, Tsorlini, and Hurni [16], is to convert the image to a binary one, that is, converting it to contain only two colors. We see similar steps in the other papers, where they more than often convert the problem to a simpler one with pre-processing or binarization.

Giraldo Arteaga [12] propose maybe the most probable version that could work on the zoning regulations by using GDAL Polygonize to do the work. The map sheets they use are however a lot simpler than the ones for the regulations and they still see errors that need to be fixed manually. The biggest problem with this approach is the configuration step for the process that needs to be done for each subsequent type of maps, which in turn creates the need to segment the maps according to quality. As we have seen the quality of the zoning regulations are very different so this might not even be possible.

For the zoning regulations, the zone color is one of the important aspects of the plan that cannot be ignored, as it describes what the allowed usage of that zone is. The plans are also of different

quality as seen in section 7.1, so the method needs to generalize well beyond the training samples in order to extract the polygons from different quality PDFs. This is not something we have seen in the previous work focusing on traditional methods. On the opposite side, generalizing beyond training samples is where DCNNs have shown great ability. In addition to vectorizing the polygon in the plan, with the perfomance of DCNN making dense predictions and assigning classes to objects, it should be possible to try to label the specific zoning type for the polygon.

As seen in section 7.1 there is no simple way to know the location of the zoning area within the PDF, so a procedure needs to be done in order to fully automate the creation of the dataset. The initial tests with OpenCV did not yield good results. This procedure needs to be investigated further in order to draw a conclusion regarding the feasibility of matching the images, but we can see that there might be a significant amount of work to get the matching algorithms to work. If one manages to match the images, the heterogeneous nature of the regulations might make it neccesary to customize the process for each plan. Causing the process to not be automatic anymore. An approach by using Mechanical Turk might be better.

Chapter 9

Conclusion

DCNNs have shown great ability to do dense predictions of semantic labels in images with accuracy up to 86.9% on the VOC2012 dataset. Huge datasets have been made for the training of these nets and data availability seems to be the biggest factor in success nowadays, but recent approaches such as CapsNet [36] try to lower the demand for data.

Some of the approaches using existing open source tools show interesting results in vectorization, but the need for uniform and similar maps does however limit the application when working with maps of different quality.

With the performance seen in the results of the DCNNs reviewed in this paper, there is no reason to believe that they should not be able to vectorize the zoning regulations as proposed. However, one of the biggest challenges found is the need to create a specific dataset. There was however no straight forward way to create the dataset automatic.

If such dataset is created, either manually, automatically or by crowd sourcing, the work proposed in this paper could greatly speed up the process of vectorizing old zoning regulations by automating the whole process. Automation could yield substantial economic savings and an effort should be given to evaluating different methods in order to do so.

Bibliography

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: (2015), pp. 1–14. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2644615. arXiv: 1511.00561. URL: http://arxiv.org/abs/1511.00561.

[2] Tore Bø. *En veileder basert på praktiske erfaringer fra Telemark og Vestfold*. 2009. URL: https://kartverket.no/globalassets/plan/digitalisering-av-reguleringsplaner-telemark-vestfold.pdf.

[3] Liang-Chieh Chen 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.

[4] Liang-Chieh Chen et al. "Rethinking Atrous Convolution for Semantic Image Segmentation". In: (2017). arXiv: 1706.05587. URL: http://arxiv.org/abs/1706.05587.

[5] Liang-Chieh Chen et al. "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: (2014), pp. 1–14. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2017.2699184. arXiv: 1412.7062. URL: http://arxiv.org/abs/1412.7062.

[6] Yao Yi Chiang and Craig A. Knoblock. "A general approach for extracting road vector data from raster maps". In: *International Journal on Document Analysis and Recognition* 16.1 (2013), pp. 55–81. ISSN: 14332833. DOI: 10.1007/s10032-011-0177-1.

[7] Matthieu Cord, Thibaut Durand, and Nicolas Thome. *Deep learning and weak supervision for image classification*. URL: http://thoth.inrialpes.fr/workshop/thoth2016/slides/cord.pdf (visited on 11/13/2017).

[8] Dov Dori. "Orthogonal Zig-Zag: An algorithm for vectorizing engineering drawings compared with Hough Transform". In: *Advances in Engineering Software* 28.1 (1997), pp. 11–24. ISSN: 09659978. DOI: 10.1016/S0965-9978(96)00035-X.

[9] Mark Everingham 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.

[10] Alberto Garcia-Garcia et al. “A Review on Deep Learning Techniques Applied to Semantic Segmentation”. In: (2017), pp. 1–23. DOI: 10.1007/978-1-4471-4640-7. arXiv: 1704.06857. URL: http://arxiv.org/abs/1704.06857.

[11] GIMP. *GIMP - GNU IMAGE MANIPULATION PROGRAM*. 2017. URL: https://www.gimp.org/ (visited on 12/16/2017).

[12] Mauricio Giraldo Arteaga. “Historical map polygon and feature extractor”. In: *Proceedings of the 1st ACM SIGSPATIAL International Workshop on MapInteraction* (2013), pp. 66–71. DOI: 10.1145/2534931.2534932.

[13] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Multimedia Tools and Applications* (2015), pp. 1–17. ISSN: 15737721. DOI: 10.1007/s11042-017-4440-4. arXiv: 1512.03385.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. In: (2015), pp. 1–9. ISSN: 0022-2488. DOI: 10.1063/1.4931082. arXiv: 1503.02531. URL: http://arxiv.org/abs/1503.02531.

[15] ImageMagick Studio LLC. *ImageMagic*. 2017. URL: http://www.imagemagick.org/script/index.php (visited on 11/12/2017).

[16] Ionut Iosifescu, Angeliki Tsorlini, and Lorenz Hurni. “Towards a comprehensive methodology for automatic vectorization of raster historical maps”. In: *e-Perimetron* 11.2 (2016), pp. 57–76.

[17] H Karabork et al. “A neural network algorithm for vectorization of 2D maps”. In: *[APRS'08] International Archives of Photogrammetry and Remote Sensing* XXXVII.B2 (2008), pp. 473–480.

[18] Andrej Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/neural-networks-1/ (visited on 11/10/2017).

[19] Kommunalt Planregister. *§ 2-2. Kommunalt planregister*. 2009. URL: https://www.regjeringen.no/no/dokument/dep/kmd/veiledninger%7B%5C_%7Dbrosjyrer/2009/lovkommentar-til-plandelen-i-/kapittel-2-krav-om-kartgrunnlag-stedfest/-2-2-kommunalt-planregister/id556741/.

[20] Philipp Krähenbühl and Vladlen Koltun. “Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials”. In: (2012), pp. 1–9. ISSN: 9781618395993. arXiv: 1210.5644. URL: http://arxiv.org/abs/1210.5644.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "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.

[22] Stefan Leyk and Ruedi Boesch. "Colors of the past: Color image segmentation in historical topographic maps based on homogeneity". In: *GeoInformatica* 14.1 (2010), pp. 1–21. ISSN: 13846175. DOI: 10.1007/s10707-008-0074-z.

[23] Tsung Yi Lin 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.

[24] Xinggang Lin et al. "Efficient diagram understanding with characteristic pattern detection". In: *Computer Vision, Graphics and Image Processing* 30.1 (1985), pp. 84–106. ISSN: 0734189X. DOI: 10.1016/0734-189X(85)90020-9.

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: (2014). ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2572683. arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038.

[26] Qiguang Miao et al. "Guided Superpixel Method for Topographic Map Processing". In: *IEEE Transactions on Geoscience and Remote Sensing* 54.11 (2016), pp. 6265–6279. ISSN: 01962892. DOI: 10.1109/TGRS.2016.2567481.

[27] G. Monagan and M. Roosli. "Appropriate base representation using a run graph". In: *Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR '93)* (1993), pp. 623–626. DOI: 10.1109/ICDAR.1993.395659.

[28] OpenVC Team. *OpenCV*. 2017. URL: https://opencv.org/ (visited on 10/12/2017).

[29] OSGeo. *GDAL - Geospatial Data Abstraction Library*. 2017. URL: http://www.gdal.org/ (visited on 11/01/2017).

[30] George Papandreou. "Modeling Local and Global Deformations in Deep Learning _ Epitomic Convolution,Multiple Instance Learning, and SlidingWindow Detection". In: *Cvpr* (2015), pp. 390–399.

[31] PASCAL VOC. *PASCAL VOC2011*. 2012. URL: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/segexamples/index.html (visited on 11/08/2017).

[32] PASCAL VOC. *Segmentation Results: VOC2012*. 2012. URL: http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11%7B%5C&%7Dcompid=6%7B%5C#%7DKEY%7B%5C_%7DSegNet (visited on 11/12/2017).

[33] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, 2017.

[34] Xiaofeng Ren and J. Malik. "Learning a classification model for segmentation". In: *Proceedings Ninth IEEE International Conference on Computer Vision* 1.c (2003), 10–17 vol.1. ISSN: 0769519504. DOI: 10.1109/ICCV.2003.1238308. URL: http://ieeexplore.ieee.org/xpl/login.jsp?tp=%7B%5C&%7Darnumber=1238308%7B%5C&%7Durl=http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=1238308.

[35] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. ISSN: 15731405. DOI: 10.1007/s11263-015-0816-y. arXiv: 1409.0575.

[36] Sara Sabour, Nicholas Frosst, and Geoffrey Hinton. "Dynamic Routing between Capsules". In: *Nips* Nips (2017). arXiv: 1710.09829. URL: https://research.google.com/pubs/pub46351.html.

[37] Karen Simonyan and Andrew Zisserman. "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.

[38] Nitish Srivastava 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.

[39] Chen Sun 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.

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

[41] The R Foundation. *R*. 2017. URL: https://www.r-project.org/ (visited on 11/12/2017).

[42] Francesco Visin et al. "ReNet: A Recurrent Neural Network Based Alternative to Convolutional Networks". In: (2015), pp. 1–9. ISSN: 10450823. DOI: 10.1109/CVPR.2016.399. arXiv: 1505.00393. URL: http://arxiv.org/abs/1505.00393.

[43] Liu Wenyin and Dov Dori. "From Raster to Vectors: Extracting Visual Information from Line Drawings". In: *Pattern Analysis & Applications* 2.1 (1999), pp. 10–21. ISSN: 1433-7541. DOI: 10.1007/s100440050010. URL: http://link.springer.com/10.1007/s100440050010.

[44] Liu Wenyin and Dov Dori. "Sparse pixel tracking: A fast vectorization algorithm applied to engineering drawings". In: *Proceedings - International Conference on Pattern Recognition* 3 (1996), pp. 808–812. ISSN: 10514651. DOI: 10.1109/ICPR.1996.547280.

[45] Michael F. Worboys. *GIS: A computing perspective*. 2003, p. 376. ISBN: 0-7484-0065-6.

[46] Yecheng Wu and Able Software Corp. *R2V*. 1999. URL: http://ablesw.com/r2v/rasvect.html (visited on 12/16/2017).

[47] Kaifu Yang et al. "Efficient color boundary detection with color-opponent mechanisms". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2013), pp. 2810–2817. ISSN: 10636919. DOI: 10.1109/CVPR.2013.362.

[48] Fisher Yu and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". In: (2015). ISSN: 00237205. DOI: 10.16373/j.cnki.ahr.150049. arXiv: 1511.07122. URL: http://arxiv.org/abs/1511.07122.

[49] M Zangeneh, M Omid, and A Akram. "A comparative study between parametric and artificial neural networks approaches for economical assessment of potato production in Iran". In: *Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA) Spanish Journal of Agricultural Research* 9.3 (2011), pp. 661–671. ISSN: 2171-9292. DOI: 10.5424/http://dx.doi.org/10.5424/sjar/20110903-371-10. URL: www.inia.es/sjar.

[50] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. "Adaptive Deconvolutional Networks for Mid and High Level Feature Learning". In: (2011), pp. 2018–2025.