Image-based terrain characterization for autonomous vehicles, based on deep learning

Andreas Hagen

Thesis submitted for the degree of
Master in Electronics and Computer Technology
Program option: Cybernetics
30 credits

Department of Physics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2019

Image-based terrain characterization for autonomous vehicles, based on deep learning

Andreas Hagen

Image-based terrain characterization for autonomous vehicles, based on deep learning

http://www.duo.uio.no/

Abstract

For an autonomous vehicle to interpret and understand the scene in front of itself, it relies on several types of different sensors. A camera may be one of these sensors, and through implementing a convolutional neural network (CNN) it is possible to extract all the necessary features from the images. In order to execute a successful feature extraction the network needs ground truth for a lot of images to train itself with a supervised learning method

This thesis investigates the opportunity to partly automate the process of assign labels for all images in two datasets from a more rural environment. To generate a label for a single image may take from two minutes to over an hour depending on the scene and the total number of classes. This thesis is considering two classes i.e. road and background. In this project, two datasets without labeling containing 439 and 2040 images are applied. From each dataset, 45 and 164 images are manually labeled, and these act as ground truth for the training images in each dataset. The network's mission is further to predict the labeling for the remaining 2270 images automatically after it has executed the training.

The residual network implemented in this thesis manages to some extent to provide most of the images with approximately close to an accepted ground truth. The exceptions are the most comprehensive scenes where unidentified objects arrive in the scene. This problem may be solved by adding more classes of the objects with the highest probability for arriving in the scene. The residual network does therefore lack some robustness to predict accurate in all types of scenes.

Sammendrag

For at et autonomt kjøretøy skal kunne tolke og forstå miljøet foran seg selv, trenger den informasjon fra ulike typer sensorer. Et kamera kan være en slik sensor, og gjennom implementering av et konvolusjonelt nevralt nettverk er det mulig å trekke ut alle nødvendige egenskaper fra bildene. For å trekke ut de ønskede egenskapene, trenger nettverket fasiten fra utvalgte bilder, så det kan trenes opp gjennom ledet læring.

Denne masteroppgaven undersøker muligheten til å delvis automatisere prosessen med å generere fasiten for alle bildene fra oppgavens datasett i et terrengbasert miljø. Å generere fasiten til ett enkelt bilde kan variere fra to minutter til over en time avhengig av antall klasser og hvordan scenen er utformet. Fasiten i denne oppgaven består av to klasser bestående av av vei og bakgrunn. Det er benyttet to datasett uten fasit i denne oppgaven, hvor det første inneholder 439 bilder og det andre 2040 bilder. Videre er det generert fasit manuelt for 45 bilder fra det første datasettet og 164 bilder fra det andre datasettet, hvor disse bildene er plassert ut i treningssettet. Nettverkets oppgave er dermed å forutsi fasiten til de resterende 2270 bildene automatisk etter de gjennomførte treningene.

Det nettverket med mest nøyaktig resultat i denne oppgaven, vil kunne forutse en tolkning som kan aksepteres som en tilnærmet fasit til de fleste bildene. Unntakene er de mest krevende miljøene hvor det forekommer forskjellige objekter i scenen. Dette problemet kan løses ved å legge til flere klasser av de objektene som har størst sannsynlighet for å dukke opp i scenen. Det mest presise nettverket i oppgaven mangler dermed tilstrekkelig robusthet for å kunne forutse fasiten til bildene i de mest utfordrene scenene.

Preface

This thesis is the end of a two years master degree in electronics and computer technology, under the cybernetics program at University of Oslo. The work has been carried out during the spring of 2019.

I would like to thank my supervisor Idar Dyrdal for his excellent guidance, super-fast response time and his availability during the whole thesis period.

I would also like to thank my family and my girlfriend for all the help and support I have received during this project. It has meant the world for me!

Lastly, I would thank all the fellow cybernetics students at Kjeller for making such a great student environment during our degree, these are two years that I will never forget.

Kjeller, May 27, 2019

Andreas Hagen

Contents

List of Figures

List of Tables

Abbreviations

AI artificial intelligence

ANN artificial neural network

CNN convolutional neural network

CPU central processing unit

DL deep learning

FCN fully convolutional network

GPU graphics processing unit

GUI graphical user interface

IoU intersection over union

mIoU mean intersection over union

ML machine learning

RGB red, green, blue

UGV unmanned ground vehicle

Chapter 1

Introduction

This section will provide an introduction to the thesis starting with the motivation before covering the problem formulation and describing the goals of the thesis. Lastly, the thesis outline is explained.

1.1 Motivation

An unmanned ground vehicle (UGV) needs several different sensor inputs to understand the entire scene in front of the vehicle, and further provide its software with the necessary information to make the right decisions regarding drivable surfaces. One of these sensors can be in the form of a red, green, blue (RGB)-camera mounted on the vehicle. A method for making the UGV interpret the scene is applying a network that trains on the data from similar scenes or alternative correlated scenes, making it able to predict the semantic scene pixel by pixel. To be able to train on the scene, the network needs ground truth for the images, and producing a label from an image manually is time consuming.

There exist multiple robust methods to perform semantic segmentation on different types of data with high accuracy [18] [11] [26]. Most of the state-of-the-art networks perform their training and inference on popular urban environment datasets with ground truth, like for example Cityscapes [3] or KITTI [20]. It is less common to find work where the datasets include scenes from a more rural environment.

This thesis will take two unlabeled terrain datasets from a RGB-camera, and investigate whether it is possible to automatically determine the ground truth for the images by classifying each pixel in the image as either road or background.

1.2 Problem formulation

Convolutional neural networks (as described in section 2.4.9 on page 17) are today able to perform both object detection and semantic segmentation with an accuracy similar to the human level [6]. Performing on that level, the CNN is a robust choice of a network for image-based data. Most of the CNNs have the same thing in common; They train with supervised learning on large datasets, containing several images with ground truth. However, in some cases the

dataset at hand may only consist of unlabeled images. To automate the task of producing ground truth for the images would save both time and costs. The problem formulation in this thesis can be expressed as follows:

Will the use of CNNs provide useful test set predictions in a unlabeled terrain-image dataset, given only a minor portion of manually annotated images?

Two different CNNs will be implemented in order to automate the task of labeling the datasets. This thesis will investigate if the predictions made by these networks can be used as ground truth for the images in the datasets. Unlike in most other datasets, this problem formulation seeks to only use a minor part of manually labeled images in the training set. Even though most CNNs use large datasets with many annotated images in the training set, François Chollet argues in his book *"Deep Learning with Python"* that deep learning (DL) can be applicable to fairly small datasets as well [2].

1.3 Contribution and goals

The implementation of the the residual network in this thesis is inspired by the contribution from Ronneberger, Fischer, and Brox in their invention of the U-Net [26]. The U-Net was initially a CNN meant for biomedical image

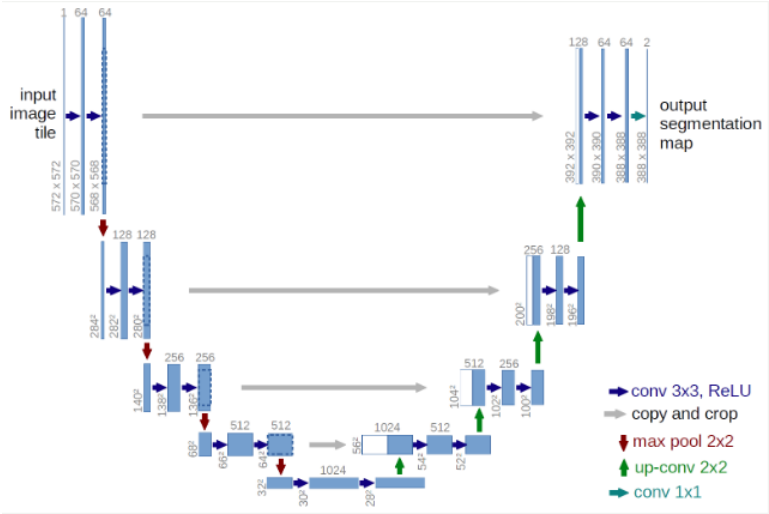


Figure 1.1: The structure of the U-Net [27]

segmentation. The main features in the network consists of its *"Skipping connections"* marked with grey arrows as *"copy and crop"* in fig. 1.1. Imagine the *"U-shape"* being split at the shortest grey arrow in the bottom, the part to the left would then be the contraction part, while the part to the right would be the expansion part. The *"Skipping connections"* concatenates the features from the contraction part with the corresponding up-convolutions in the expansion part. The U-Net architecture makes it possible to gain accurate results based on a few training samples, and that is exactly the type of feature this thesis aspire to produce.

The goals and aims of this thesis are as follows:

1. Perform a literature review of existing CNNs and select a robust network to implement
2. Implement a standard CNN as baseline
3. Investigate whether the CNNs predictions have the necessary accuracy to be used as ground truth for the images in the custom-made datasets
4. Obtain a more generalized network with transfer learning from a similar dataset
5. Testing the networks, then presenting and analyzing the results

1.4 Thesis outline

- **Chapter 1:**
  Presents the thesis introduction including the addressed problems, the contribution and the goals
- **Chapter 2:**
  Covers the necessary background theory and the related work
- **Chapter 3:**
  Describes the methods used and the experiments carried out
- **Chapter 4:**
  Presents the training results, the quantitative results and the qualitative results
- **Chapter 5:**
  Covers the discussion of results
- **Chapter 6:**
  Concludes the thesis and present suggestions for further work

Chapter 2

Theoretical background

This chapter will initially provide a short overview of the basics of AI, ML, computer vision, and deep learning (DL). A DL framework is used for implementing the network in the thesis, while a few computer vision techniques is used for processing the images further after the predictions. The reasoning behind choosing DL will be briefly discussed. Further is the necessary data preprocessing described, before the basics of different artificial neural network (ANN) is covered. Lastly, this section will go through the theory of manually generating ground truth labels with a suitable program, transfer learning and related work.

2.1 Artificial intelligence

AI is a field where machines are able to demonstrate intelligence through mathematics, statistics and logic. AI has the ability to tackle many complex problems, which are intellectually difficult or impossible to solve for a human being (with natural intelligence). Even though AI could solve complex problems, it still had a few challenges in the early days. Some intuitive tasks for humans, such as recognizing a cat or a dog in an image, or the context of a written text, proved to be a true challenge. A solution to these problems was allowing computers to learn to approximate logical rules from experience, which is where ML and DL comes into the picture [12].

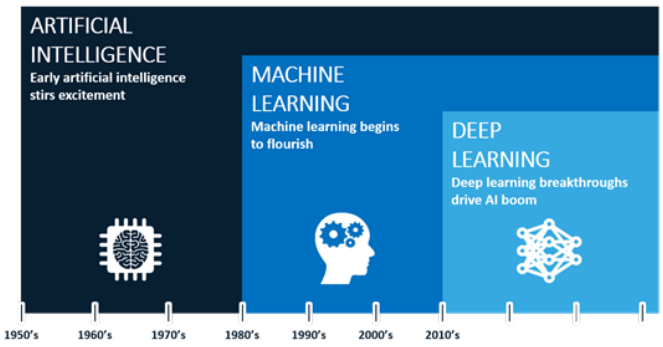


Figure 2.1: A provisional timeline for AI, ML, and DL [14]

2.1.1 Machine learning

Figure 2.1 illustrates that ML is a subset from AI. The term is further well defined by ML pioneer Tom M. Mitchell:

"Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience." [15].

ML algorithms will explicitly be programmed to improve their performance on a task. Humans would provide the data, along with the answers in order to search for and produce the rules. This is visualized in fig. 2.2 [2].

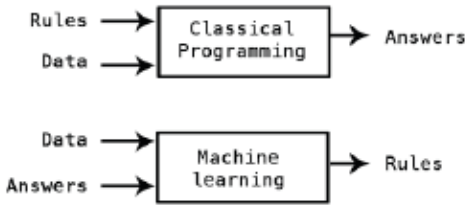


Figure 2.2: Classical AI illustrated on the top and ML illustrated on the bottom part of the image [2]

2.1.2 Deep learning

As visualized in fig. 2.1, DL is a subset of both AI and ML. The main difference between traditional ML algorithms and DL is that DL is able to learn data representations from datasets instead of manually extracting features. The whole network is in other words able to solve the problem from start to end without using external feature extraction methods as in ML. DL has its networks (called ANN) slightly based on the same principle as the neuron system from a human brain. This thesis will use DL as foundation for solving the task described in section 1.2 on page 1.

Supervised learning

Supervised learning is the most commonly used technique in DL [2]. The word supervised refers to known targets or annotations in the form of a labelled dataset. With that knowledge a function can learn how to map input data to the targets.

2.1.3 Benefits using DL with CNN vs traditional methods

DL has become increasingly more popular during the last few years. One of the reasons for this increased popularity is DLs ability to provide higher accuracy when trained with large amounts of data. Even though DL is usually known for predicting good results from large datasets, it does also contain methods for providing good results from networks trained with smaller datasets as well. As is

the case in this thesis. This benefit over traditional ML methods alone would be enough to consider taking DL as preferred method.

Another argument for employing DL over traditional methods is the feature extraction from images. While the traditional algorithms need to manually implement different computer vision techniques in order to extract the desired features before classification, this is not the case with DL. With the use of convolution layers from CNN the features are extracted automatically from the layers. The first layer will detect and learn small edges. Then the second layer will learn larger patterns made from the features from the first layer, and this concept will repeat itself in the further layers. The patterns learned are translation invariant, which means it can recognize the learned pattern if it appears anywhere else in the image. This advantage are exclusively for the CNNs, and makes the networks able to generalize better with less training samples compared to a densely-connected network [2].

2.2 Computer vision

Computer vision is a field whose main purpose is to make machines able to interpret and understand features from images or video. In other words, "Teaching computers how to see" [28]. This section will only go through a small number of computer vision concepts, as they are used in the thesis.

2.2.1 Semantic segmentation

Segmentation is a concept where the input is in the form of an image, and the output consists of regions and structures based on the input. Normal segmentation will in most cases only provide a basic scene understanding. If the goal is to understand what is in the image more thoroughly, semantic segmentation is the next step. The idea behind semantic segmentation is that instead of regions, every pixel in the image are classified. This means that it will be possible to gain a broader scene understanding of the environment in the image, making it easier to recognize different elements [32].

2.2.2 Morphological operations

A common binary image operation is called *morphological operations*, since they change the shape of the underlying binary objects [19]. These operations are typically used to clean up binary images. The two binary morphological operations used in the thesis are:

- **Erosion**
- **Dilation**

Erosion thins the object and *dilation* thickens the object. Using these operations in this specific order (erosion + dilation) results in *opening*. This operation tends to smooth boundaries, while also removing some noise from the image. It is

common to experiment on how much iteration that should be implemented in each the erosion and the dilation operation.

The structuring element used is normally a rectangular kernel of desired size. It can also take the shape of an circle if necessary. This depends on what the regions in the image looks like. Sometimes a circle shaped kernel may be a better option to smooth the curves than the normal rectangular kernel.

2.2.3 Connected component analysis

The connected component analysis is a tool which can be used to filter out noise from a binary image. It labels all the connected regions in the image automatically in an iterative manner. This way makes it possible to mask out every label which contains less pixels than a given threshold. If the region after the noise filtering contain holes surrounded by a complete polygon, these can be sealed with a binary object filler. If both operations are implemented successfully, the improvement from the original prediction will be significantly better.

2.3 Data preprocessing

This section will cover data processing from an image-based point of view. Before the dataset can be fed into the network, it would in most cases be inevitable to perform several preprocesses. This is necessary in order to make the data prepared for training.

A common step with an image-based dataset is to resize the images to a slightly lower resolution. This is especially necessary if the computer does not have a state-of-art graphics processing unit (GPU), as the model otherwise might be very slow. The down scaling will help the model to be more effective and less time consuming, at the prize of losing some features from the original resolution. It is therefore important to test different scales to find the perfect fit between keeping enough key features and have an effective model.

A RGB image contains integer values in the range from 0-255 in each of its three channels. Because the values of the weights in a neural network are relatively small, it is common practice to normalize the image-array to values between 0-1. Doing so will prevent slowing down the learning process, as the values from the weights and the array now are in closer range. Casting the array from int to float before normalizing would increase the accuracy even further. This is due to the float division resulting in a more accurate number than int division.

Another process is to check whether the dataset has the correct shape or not. This is necessary because the network needs to know which input shape to expect. The input layer in *Keras* is a tensor which is passed to the first hidden layer. If that input layer does not correspond with the shape of each element in the dataset, the network will not be able to execute. In the case where "Conv2D" layers are used in a *Keras* framework, an input array needs to have the following structure:

(height, width, channels)

Where height and width refer to the x- and y-coordinates in the image, and channels refers to if the image is binary (channels=1) or RGB (channels=3).

2.3.1 Data augmentation

Data augmentation is a tool which helps increase the data distributions variance, which in return could increases the network's generalization. It is a method for applying transformations to the training data. When the images are pre-processed with the methods described in this section, the network learns how to cope with slightly different images than the original training set. This is the reason the model has a higher chance of predicting the test images (images that the model never have seen before) with more accuracy. In addition to potentially higher accuracy, the model also has lower risk of overfitting. This sub-section will only cover the most used augmentation techniques.

Random cropping

A popular method in data augmentation is random cropping. Random cropping is sampling a random chosen square box from the original image, and then resize to the original size. As seen in fig. 2.3 the image focus on different areas

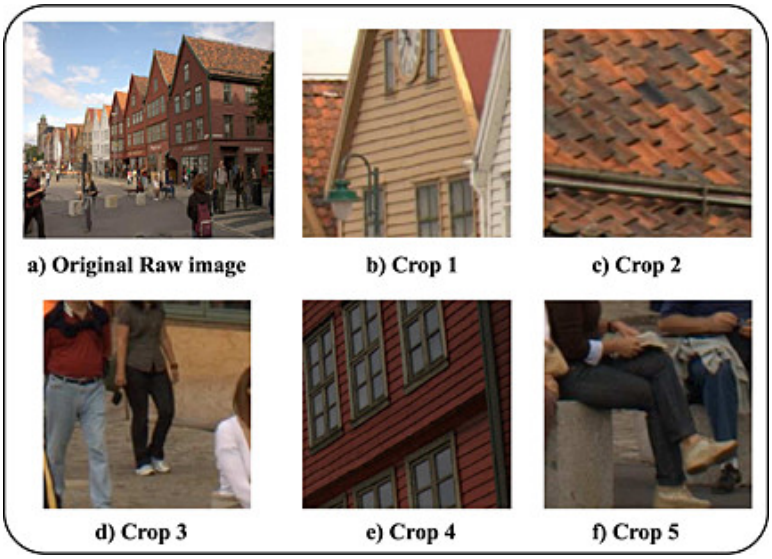


Figure 2.3: Random cropping [4]

from the original image, due to the random chosen box. This operation must be included with the ground truth images. When an operation changes the geometry in the image, the same operation must be done in the ground truth image in order to still be a valid target.

Flipping images

Another method is to flip the images in either vertical or horizontal order. Even when the images are flipped, are they recognizable for the model. The ground

truth must also go through the same operation in order to keep the correct geometry in both images.

Color changes

The idea behind color changes is to make the model more robust and generalized for new unseen data. Since the geometry in the images are the same after applying color changes, is it not necessary to do any operation on the ground truth images.

2.4 Artificial neural networks

To provide the necessary understanding of how the networks implemented in this thesis works, this section starts by explaining the fundamental pieces of ANN before diving deeper into CNN in section 2.4.9. Most of the theory in this section is based on the Stanford University course *"CS231n: Convolutional Neural Networks for Visual Recognition"* [5], and from the course *"INF4490 – Biologically Inspired Computing"* [9][10] at University of Oslo.

An ANN is a computing system which is vaguely based on the same principle as biological neurons in a human brain. It has the ability to learn different tasks and data representations. To fully understand the concept, the basics of a neural network is divided into different parts and further described in this section.

2.4.1 Single- and multi-layer neural network

In 1943 McCulloch & Pitts designed a much simplified version of biological neurons [33]. With their design, they are widely known as the inventors of the first ANN. Their ideas of a threshold in the activation function and combining many basic units in order to increase computational power are still being used today. The illustration of the neuron and its activation function in fig. 2.4 can be

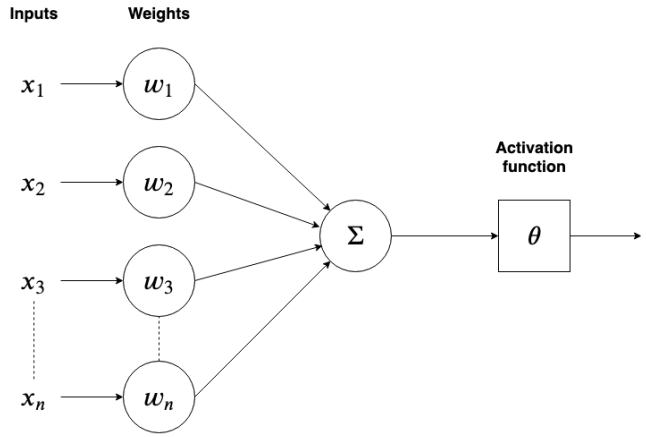


Figure 2.4: Illustrates the McCulloch & Pitts design of a simplified neuron

mathematically explained with eq. (2.1).

$$h = \sum_{i=1}^{n} x_i \cdot w_i \quad , \qquad o = \begin{cases} 1 & \text{h} \geq \theta \\ 0 & \text{h} < \theta \end{cases} \tag{2.1}$$

Where the neurons function (h) is denoted in the form of a dot product between the inputs (x_i) and the weights (w_i). The neurons activation function *"fires"* when the dot product of the input and its weight respectively are higher than a given threshold θ. Meaning the output (o) becomes 1 when h is equal to or higher than the threshold, and 0 when h is lower than the threshold value.

If many McCulloch & Pitts neurons are put together, the structure of a single-layer neural network appears. A single-layer perceptron is able to learn linear problems. When the task is to learn non-linear problems, the solution is to add one or more hidden layers, as done in multi-layer perceptron. The difference

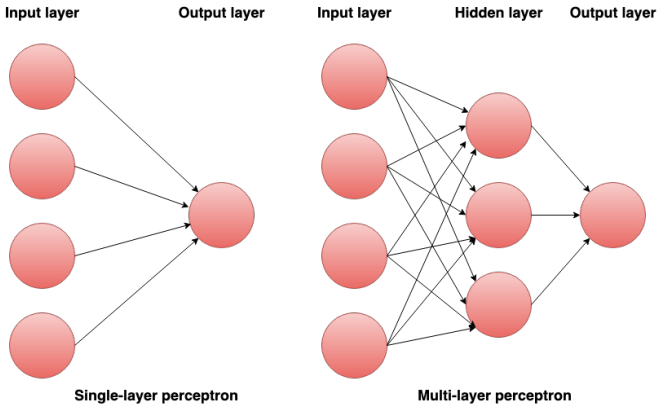


Figure 2.5: Single-layer perceptron to the left, and multi-layer perceptron to the right

can be seen visually by fig. 2.5, where the single-layer perceptron only has an input and an output layer, while the multi-layer perceptron includes at least one hidden layer.

2.4.2 The learning rule

In order for the single-layer neural network to learn, it has to adjust the weights accordingly. This is where the perceptron learning rule become relevant.

$$w_{ij} \longleftarrow w_{ij} + \Delta w_{ij} \tag{2.2}$$

Equation (2.2) shows how the weight (w_{ij}) updates. The goal of the learning rule is to minimize the error at the output, such that $\Delta w_{ij} = 0$. When the weights reach that state, they are tuned correctly. The weights can be both positive and negative, and how they adjust can be explained with the next equation.

$$\Delta w_{ij} = \eta * (t_j - y_j) * x_i \tag{2.3}$$

In eq. (2.3) η is referred to as the learning rate. It is a scalar which decide how much the weight value in each iteration will change. Finding the right balance

in the choice of learning rate is therefore crucial. A high value (e.g. 1) can create an unstable net. With a high learning rate will the weights change a lot every time they update. Choosing a low value will make a stable network, but will require much more learning time, because the weights uses more time to tune into correct values.

η is further being multiplied with the error ($(t_j - y_j)$, where t_j is the target output and y_j is the actual output). Before finally being multiplied with the inputs (x_i). As stated above, the goal is to minimize this error. Which is done during training where the weights are adjusted with eq. (2.3).

2.4.3 Bias

In the case where all inputs are zero, the weights will have no effect since they are multiplied with the inputs. The solution for that particular case is to have an adjustable threshold, which can be applied with a bias node. The bias node should be added to each neuron. Then eq. (2.1) will become eq. (2.4), where b is the bias.

$$h = \sum_{i=1}^{n} x_i w_i + b \quad , \qquad o = \begin{cases} 1 & \text{h} \geq \theta \\ 0 & \text{h} < \theta \end{cases} \tag{2.4}$$

2.4.4 Backpropagation in multi-layer neural networks

When learning in a single-layer neural network, it is possible to gain knowledge about which weight who contribute to reducing the loss. In multi-layer perceptron there is at least one hidden layer between the input and the output. Hence, it is impossible to know which weights are correct, and which activations being correct for the neurons in the hidden layer. Without knowing which weight or activation is correct, it is impossible to learn the weights or train the network. The problem of not being able to train a multi-layer neural network was solved in 1986 with an algorithm called backpropagation [22]. The

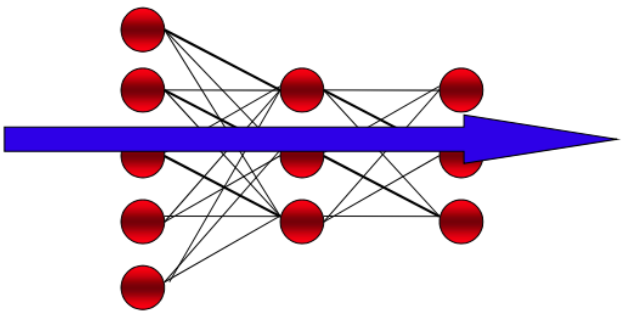


Figure 2.6: The forward pass in backpropagation [10]

backpropagation algorithm consists of two main steps. The first is the forward pass, which has the following structure illustrated in fig. 2.6. After the input layer has received its inputs, the activations of the hidden nodes in the middle

layer is calculated. Lastly the activations of the output nodes in the last layer are calculated. The second step in the backpropagation is called the backward pass,

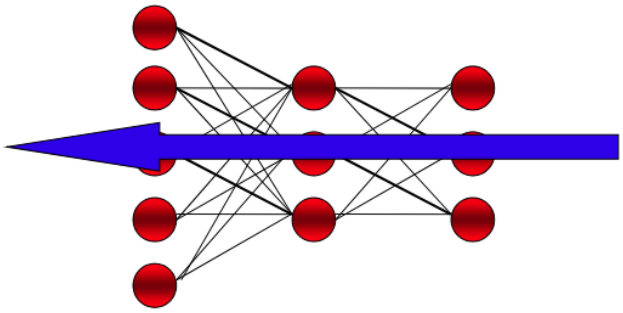


Figure 2.7: The backward pass in backpropagation [10]

and is illustrated in fig. 2.7. This step starts by calculating the output errors in the last layer, before it updates the same layers weights. Then the error is being propagated backwards, and the hidden weights in the middle layer are updated. This process is repeated until the first layer is reached.

2.4.5 Gradient descent learning and momentum

When training the network with backpropagation, the goal is to minimize the errors in the network. As described with the backward pass, after being calculated, the errors from the output layer are propagated backwards in the network. The tool used is a form of gradient descent.

$$E(w) = \frac{1}{2}\sum_k (t_k - y_k)^2 = \frac{1}{2}\sum_k (t_k - \sum_i w_{ik} x_i)^2 \tag{2.5}$$

It differentiates the sum-of squares error in eq. (2.5), showed with the eq. (2.6).

$$\Delta w_{ik} = -\eta \frac{\delta E}{\delta w_{ik}} \tag{2.6}$$

Even though gradient descent algorithm is a good method for finding the minimum value, it has a potential risk of getting stuck in a local minimum as visualized in fig. 2.8. There are two alternatives to avoid being stuck in a local minimum. The first one is to initialize the training several times with random weights. The other method is to use momentum. If the gradient descent algorithm reaches a local minimum, the momentum keeps the algorithm going further uphill for a while, until the descending starts again and hopefully a global minimum will be found instead. Momentum is described mathematically in eq. (2.7).

$$w_{ij} \longleftarrow w_{ij} - \eta \Delta_j z_i + \alpha \Delta w_{ij}^{t-1} \tag{2.7}$$

2.4.6 Activation functions

The task of an activation function is to decide whether a neuron should *"fire"* or not. In other words, the activation function takes a number and performs

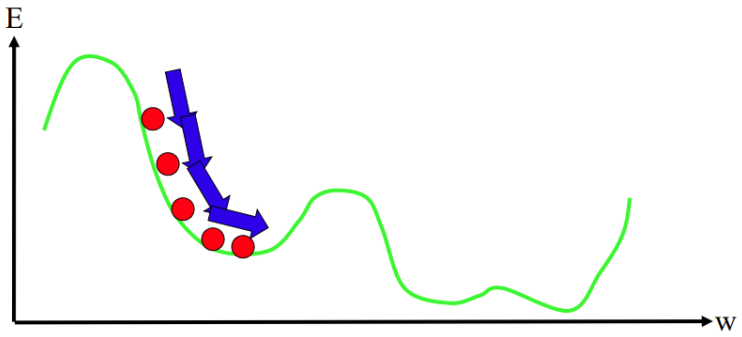


Figure 2.8: Gradient descent [10]

a mathematically operation on it. There exist several activations functions, all with different pros and cons. The ones being used in this thesis will be covered here.

The sigmoid function

The sigmoid function was a historically often used activation function. It is mathematically described with eq. (2.8).

$$\sigma(x) = \frac{1}{(1+e^{-x})} \tag{2.8}$$

The function transforms the input numbers into a range between 0-1 as shown in fig. 2.9. This means large negative numbers become 0, while large positive numbers become 1. Today, the popularity of the sigmoid function has decreased

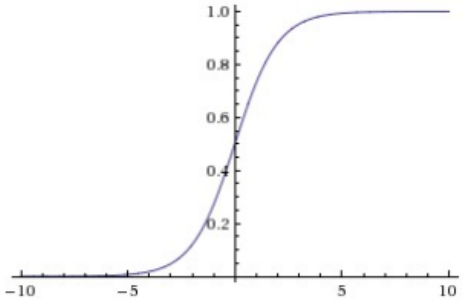


Figure 2.9: Sigmoid activation function [16]

due to the following drawbacks:

- *Vanishing gradients at values close to 0 or 1*
- *The neurons can be saturated if the initial weights are too large*
- *The outputs are not zero-centered*

The ReLu function

The ReLu function (Rectified Linear Unit) has become increasingly more popular in the last few years. It is proven much faster than the sigmoid or tanh (which is a scaled sigmoid function) functions in a paper, due to its linear, non-saturating form [13].

$$f(x) = max(0, x) \tag{2.9}$$

$$f(x) = \begin{cases} 0 & \text{for x} < 0 \\ x & \text{for x} \geq 0 \end{cases} \tag{2.10}$$

The ReLu function is described mathematically in two ways in this thesis, both being illustrated in eq. (2.9) and eq. (2.10) for a better understanding of the function.

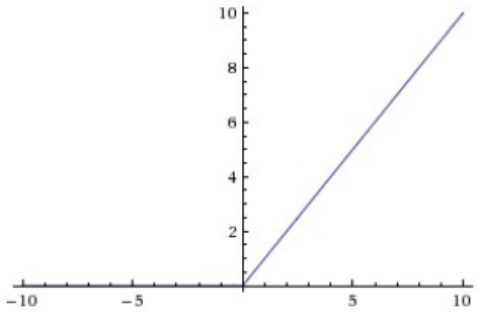


Figure 2.10: ReLu activation function [16]

As illustrated in fig. 2.10 the ReLu activation is thresholded at zero. This makes ReLu a favored choice over the sigmoid/tanh functions. It is a less computationally expensive method, because it does not have the exponential implementation. There is, however, one disadvantage using ReLu. The units can be fragile during training and as much as 40% of the network may end up *"dead"*. This can happen if a large gradient is flowing through the ReLu neuron. It may cause the neuron to update in such a way that the neuron never will activate on a data point again and end up *"dead"*. With a proper learning rate (not too high), the problem tends to be avoided in most cases.

2.4.7 Loss and optimizers

The task of a loss function is to measure the distance between the predictions being made by the network and the actual ground truth. In this manner there can be computed a distance score, controlling how well the network did with its prediction [2]. This is illustrated in figure fig. 2.11.

The loss score computed by the loss function, is further being used as a feedback signal for adjusting the weights slightly. The weights are updated in a direction which lower the loss score, in order to make better future predictions. This is shown in fig. 2.12. The job of adjusting the weights is executed by what is

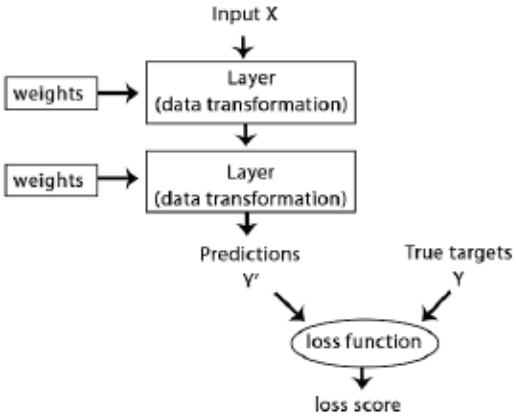


Figure 2.11: Loss function [2]

called an optimizer. The optimizer implements the backpropagation algorithm earlier described in this section [2].

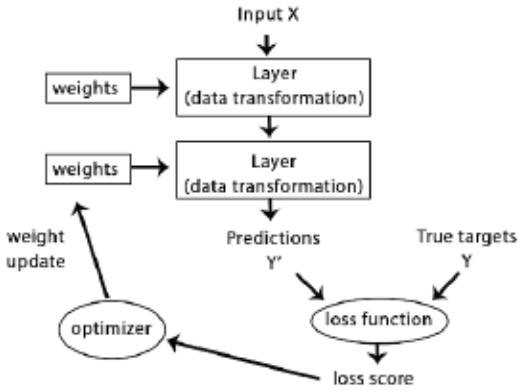


Figure 2.12: The visualization of an optimizer [2]

2.4.8 Regularization

To avoid overfitting (explained in page 95 & 96 [2]), the implementation of regularization will be helpful. Regularization refers to regulating the weights, by constraining them to only accept small values. It is implemented by adding a *cost* to the loss function if it has too large weights. The cost comes in two different forms:

- ***L1 regularization***
- ***L2 regularization***

Where *L1 regularization* adds the costs proportionally to the absolute value of the weights coefficients, and *L2 regularization* adds the costs proportionally to the square of the absolute value of the weights coefficients [2].

Another popular regularization method is *dropout*, developed by Hinton and his students at the University of Toronto [8]. It consists of randomly zeroing out (dropping out) a number of output features of the layer during training. The term *"dropout rate"* refers to the fraction of the features which is being dropped out, and is usually put to a number between 0.2-0.5. When the algorithm is ready for testing, the output values are scaled down with a factor equal to the dropout rate instead of being dropped out. This is done in order to balance for the amount of more active units during testing. To use dropout as a regularization method is both very common and efficient [2].

2.4.9 Convolutional neural networks

A CNN is a sub class of ANN. It is a type of network which automatically extracts several types of features from images, and are further used for making different sorts of predictions based on the given task. The main difference between a CNN and an ordinary neural network is how the input is interpreted. In a CNN the inputs are assumed to be images.

Unlike a CNN, an ordinary neural network with fully connected layers will keep all parameters connected from the input until the output. The CNN, with the use of convolutions, is able to reduce the numbers of parameters vastly from each layer while keeping the key features. This difference makes CNN a better tool for processing images, as images often contains a large number of parameters.

The following example will demonstrate why it is necessary to reduce the parameters when processing images. Imagine an ordinary neural network with an image (width, height, depth) as an input. If the size of the input is (224x224x3) this means the number of weights will become $224*224*3 = 150528$ weights. Adding more similar neurons will escalate the number of parameters considerably, which then will result in an unnecessary overfitting and a poor network.

The characteristics of a CNN is described below, providing a general understanding how the parameters are reduced and CNNs unique features.

Convolutional Layer

Being the core building block of a CNN, the convolutional layer does most of the computationally expensive work necessary for the network to perform well. It consists of learnable filters, which slides through the whole input image bit by bit. Equation (2.11) illustrates the general expression of a 1D convolution. Where ω is the filter being convolved with the input $f(x,y)$, providing the output $g(x,y)$.

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t) \tag{2.11}$$

To understand the process even better, the Stanford course *"CS231n: Convolutional Neural Networks for Visual Recognition"* has provided an intuitive visual model showing a 3D convolution explicitly in fig. 2.13. This illustration consists of following parameters and configurations:

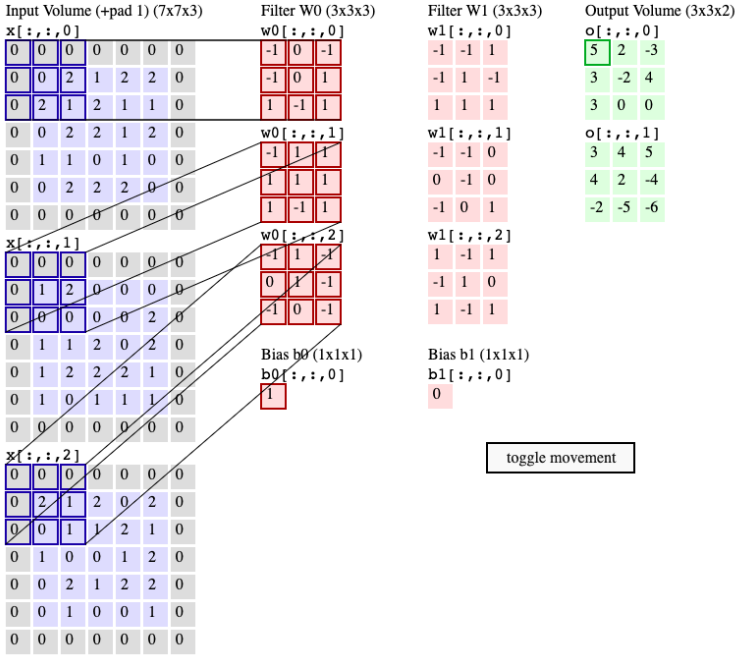


Figure 2.13: 3D Convolution step by step [5]

- *Input volume = (7x7x3)*
- *Stride = 2*
- *Number of zero padding = 1*
- *Two weight filters = (3x3x3)*
- *Two biases = (1x1x1)*
- *Output volume = (3x3x2)*

Where the first weight filter W_0 is sliding over each part of the input in its three channels. This gives the output volume O[:,:,0], while the convolving of weight filter W_1 provides O[:,:,1]. The stride is set equal to two, which means the filter can slide in three positions both in x- and y-direction. This operation makes the output dimension into width and height equals to 3. The last dimension in the output volume is set by the number of filters (W_0 and W_1) convolving over the input volume. The number of chosen filters is a hyperparameter and decides the depth of the output. In this example there are two filters, which means the final output volume becomes (3x3x2). The implementation of zero padding helps us to control the spatial size of the output. It is also a hyperparameter, and in this example it is set equal to one, which gives us the pad (marked in grey in fig. 2.13) around the input volume filled with zeros.

The equation to compute the spatial size of the output volume is illustrated in eq. (2.12)

$$O = \frac{(W - F + 2P)}{S} + 1 \tag{2.12}$$

Where O is the output volume. W is the input volume, F is the receptive field (the weight filter), P being the amount of zero padding, and S for the number of strides.

Parameter sharing and local connectivity

In contrary to ordinary neural networks, CNNs have neurons set up in three dimensions:

- *Width*
- *Height*
- *Depth*

These neurons may have various levels of connectivity between the layers. The two concepts controlling and reducing the number of connections between the neurons are called parameter sharing and local connectivity. As earlier described, one of the main reasons for the CNNs to out perform ordinary neural networks (when processing images), are their properties to reduce the number of parameters while keeping key features.

Parameter sharing means to share some weights and biases in order to control the number of parameters. This can be done by assuming that if a feature is useful to calculate at a specific spatial location (x_1, y_1), it should be useful to compute it at a different location (x_2, y_2) as well. In practice that means constraining the neurons in each depth dimension to use the same weights and bias. Parameter sharing will greatly reduce the amount of total weights, and is an important contribute to make an efficient CNN.

Local connectivity connects each neuron only to a small region of the previous layers input, unlike connecting the neurons to all the neurons from the previous layer as done in ordinary neural networks. This small region is a hyperparameter, and is called the receptive field of the neuron. It is the same as the weight filter from the example above. The depth slice in the weight filter is always the same as the depth slice from the previous layers input. This means that we have local connections along width and height, and full connection along the depth of the input layer.

2.4.10 Pooling layer

Another method to reduce parameters is to add a max pool layer. Similarly to the convolution layer it contains a filter (F) and stride (S).

But as seen in fig. 2.14, the filter takes the max value in each frame instead of convolving through the input like the convolution layer does. The most common values in the max pooling layer are F =2 or 3, and S = 2. Increasing these values will result in a destructive layer. It is a common practice to implement a max pool layer periodically between convolutional layers.

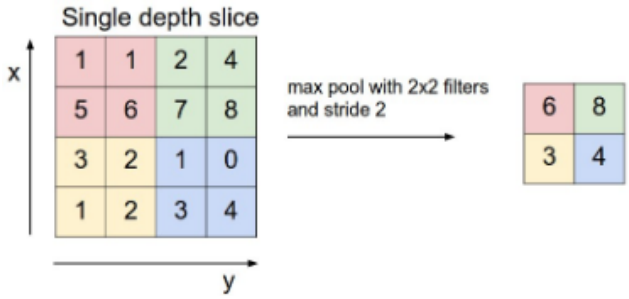
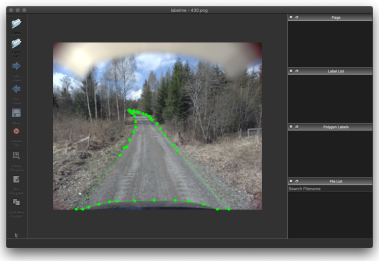


Figure 2.14: Max pooling [5]

2.5 Generating the ground truth labels manually

There exist several good programs for generating labels for datasets, all with different benefits and shortcomings. The program used in this thesis is a free program, which offer offline annotation. It is called *"LabelMe"*, and it can be installed directly from GitHub [31]. The program is easy to learn and contains annotation examples in the GitHub folder. *LabelMe* has a clean programmed GUI regarding the working space, which is illustrated in fig. 2.15a.



(a) Manually annotating an image with LabelMe

(b) The annotation superimposed on the original image

Figure 2.15: Labeling an image

When the polygon is drawn, another script is executed from the LabelMe directory, and the complete annotation ends up like in fig. 2.15b.

2.6 Transfer learning

Transfer learning applies already learned knowledge from saved weights and applies it to a new problem with a new dataset. It is sometimes referred to as using a pre-trained network. Using a pre-trained network is usually a highly efficient way to gain better results when using small datasets. The spatial features learned by the pre-trained network might prove useful for the original problem the network was designed for, because it may transfer essential knowledge from for example a well made dataset. The results of implementing transfer learning will in most cases provide a much higher accuracy to the original problem, in contrary to only train on a small dataset [2]. The implementation of transfer learning is therefore used in this thesis due to the very small number of annotated images available for the training set.

2.7 Related work

There are numerous examples where CNNs have been successfully used for semantic segmentation problems, some of them will be mentioned and briefly described in this section.

DeepLab [18] is a state-of-the-art network which provides high accuracy on semantic segmentation problems. It uses dilated convolutions (illustrated in fig. 2.16) as a tool to adjust the filter's field-of view.

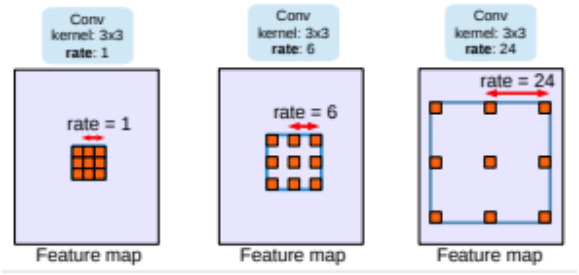


Figure 2.16: Dilated convolution illustrated with different dilation rates [18]

When the $dilation\ rate = 1$ in fig. 2.16, this represents a normal convolution. If this dilation rate increases, the field-of-view increases accordingly and the resolution will in this manner be controllable because an increase in the dilation rate will decrease the numbers of features computed.

Another widely known network is the *Fully convolutional network (FCN)* [11]. This network could take an image input of arbitrary size, and provide the same size as output. The authors behind FCN claimed to have one of the first network which was trained end-to-end and had pixelwise prediction, meaning the predictions were semantically annotated. When their paper was published in 2014 they exceeded the state-of-art in semantic segmentation. Figure 2.17 illustrates the process from the input image to the output prediction.

This thesis loosely bases one of its network implemented on the *U-Net* architecture [26]. The *U-Net* is already briefly covered in section 1.3, and will be further explained in section 3.1.4. *U-Net* proved its qualities by winning the

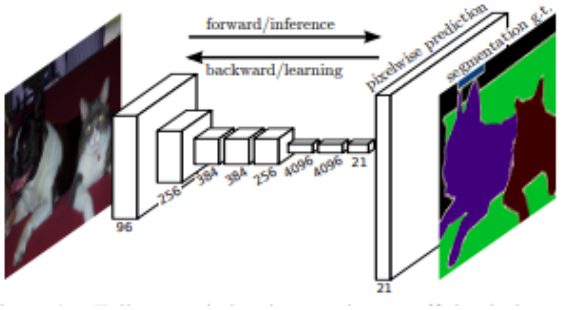


Figure 2.17: The FCN process from input to pixelwise prediction [11]

ISBI cell tracking challenge 2015 [17], some of the network's features is having a short inference time and being able to receive accurate results on a relatively few training samples.

For the networks to be able to make accurate predictions, they need datasets made with high quality. Most of the networks nowadays which carries out the training on road-based images does so in popular semantic segmentation datasets like for example the *Cityscapes* [3] or the *KITTI* dataset [20]. These datasets helps most of the networks providing accurate results when predicting road and objects in urban city areas. In this thesis the datasets consists of images from a more rural scene. Even though there is much less research provided in prediction of images from rural environments, there exist an off-road dataset called the *Freiburg Forest* dataset which is described in section 3.3.1 and in section 3.1.5. This dataset is used for transfer learning in this project also explained in section 3.1.5.

Chapter 3

Method

3.1 Implementation

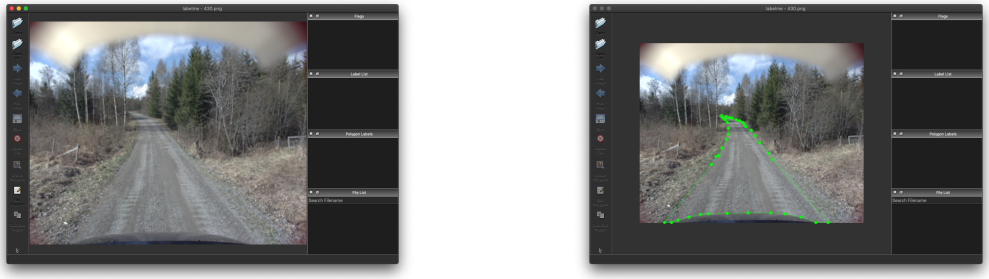
This chapter will cover the implementation of the program in the thesis and discuss the choices of methods. It will initially describe how the ground truth from the training images is made. Then cover which framework is used, before describing both implementations of CNN, with pros and cons. Furthermore the advantage of using Tensorboard will be discussed. The mid section in this chapter will cover predictions made from the network, and a few computer vision techniques used in order to maintain a clean output with minimal noise. The last part of this section will cover the experiments done in this thesis, in the form of describing the structure of the datasets and the training configuration.

3.1.1 Annotating the training set

Before the work with the network can start, there must exist a prepared dataset. As supervised learning is the applied method, some of the images needs ground truth in order for the algorithm to maintain a reliable loss function. Only ten percent of the images are manually annotated. This is due to the intention of this thesis to automatically predict the test images ground truth with as little manually annotated images in the training set as possible. The images picked out for manual annotation starts from image number one in the dataset, and continues in an iterative manner every tenth image. The following commands is used in the annotating program *LabelMe* in order to start and convert the annotation from *.json* to *.png* file format:

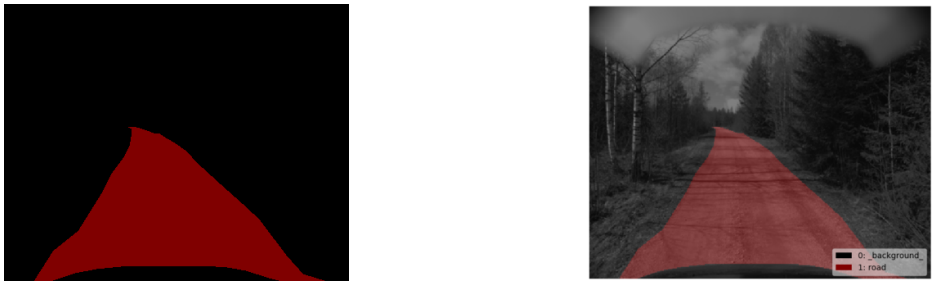
```
$ labelme <input_file.png> -O <output_file.json>
$ labelme_json_to_dataset <input_file.json> -o <output_folder>
```

When the first line is executed, the GUI for *LabelMe* shows up in an external window. From there it is possible to start drawing the annotation by pressing the *"Create Polygons"*. Figure 3.1b illustrates a finished drawn polygon. After the polygons are drawn and the save button is pressed, the second line is ready to be executed. This command will take all the polygon positions saved in the *.json* file and convert the ground truth into the final *.png* format. The second command will also make a directory which consist of the original image, the ground truth superimposed on the original image, and the ground truth image.



(a) The GUI for LabelMe (b) Drawing a polygon

Figure 3.1: The LabelMe GUI



(a) (b)

Figure 3.2: The ground truth in its original form in fig. 3.2a, and superimposed into original image in fig. 3.2b

The finished ground truth for the image consists of two classes. Where class 0 is assigned to the background, and class 1 to the road. The program *LabelMe* was chosen due to its quick setup, its possibility to annotate data offline, and a clean GUI making it easy to draw polygons.

3.1.2 Keras

Keras is an user friendly DL library developed in Python. It was originally made for researchers as a way to do quick experimentation, with an easy to use implementation. *Keras* has quickly gained popularity among its users and is one of the most popular frameworks in DL projects nowadays [2]. The main reasons for the popularity is the user friendliness and that *Keras* can run with the same code seamlessly on central processing unit (CPU) or GPU [2]. Both implementations of CNNs in this thesis are done in *Keras* and it became the chosen framework due to its easy implementation of networks and powerful tools.

3.1.3 The sequential network

The sequential network implemented is made as a basic fully convolutional network (FCN). This means the network is composed without using any fully connected layers at all. The FCN has therefore learning filters placed everywhere, even in the decision making layers at the end of the network. This

will save computation time and reduce the number of parameters compared to using fully connected dense layers.

```
# Initializing
model = models.Sequential()

# Input layer
model.add(layers.Conv2D(16, 5, strides=(2, 2), padding='same', activation='relu',
input_shape=input_shape))

# Conv layers
model.add(layers.Conv2D(16, 5, strides=(2, 2), padding='same', activation='relu'))
model.add(layers.Conv2D(32, 5, strides=(2, 2), padding='same', activation='relu'))
model.add(layers.Conv2D(32, 5, strides=(2, 2), padding='same', activation='relu'))
model.add(layers.Conv2DTranspose(32, 5, strides=(2, 2), padding='same',
activation='relu'))
model.add(layers.Conv2DTranspose(32, 5, strides=(2, 2), padding='same',
activation='relu'))
model.add(layers.Conv2DTranspose(16, 5, strides=(2, 2), padding='same',
activation='relu'))
model.add(layers.Conv2DTranspose(16, 5, strides=(2, 2), padding='same',
activation='relu'))

# Output layer
model.add(layers.Conv2D(1, 1, strides=(1, 1), padding='same', activation='sigmoid'))

# Prints the network structure summary
model.summary()
# Adding loss and optimizer
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

As seen in the code snippet above, this is the implementation of the sequential network used in the thesis. The input tensor (which is (224x224x3)) goes from the input layer through each layer until the output layer is reached. There is both a downsampling and an upsampling process in the network. In the downsampling process consisting of four downsampling layers, the height and width is reduced from (224x224) to (14x14). Each step in the following list represents a layers downsampling and has the depth included:

- *(112x112x16)*
- *(56x56x16)*
- *(28x28x32)*
- *(14x14x32)*

Then the upsampling process also with four layers continues sampling from (14x14) to (224x224) in the height- and width-size of the tensor. Every step in the list represents a layers upsampling and the depth is included here as well:

- *(28x28x32)*
- *(56x56x32)*
- *(112x112x16)*

- *(224x224x16)*

The last layer is the output layer containing sigmoid activation. It has the shape (224x224x1).

The sequential network is included as the baseline in the thesis. It is done in order to benchmark it against the state-of-the-art based implementation of the residual network (U-Net). In that way it will be possible to compare the differences between a basic sequential and a state-of-the-art residual-network, trained on the same data.

Loss

There are simple guidelines which can be used for choosing the right loss. If there is a two-class classification problem, a loss called *"binary_crossentropy"* will be a natural choice [2]. This is therefore the loss used in the sequential network, and it is described mathematically in the eq. (3.1).

$$L_{BCE} = -\sum_{i=1}^{C=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1-t_1) log(1-f(s_1)) \tag{3.1}$$

$$f(s_i) = \frac{1}{1+e^{s_i}} \tag{3.2}$$

Where BCE is short for binary cross entropy. C_i is the classes, t_i and s_i is the target and the score respectively. The score goes first through a sigmoid activation described in eq. (3.2), before the loss is further computed with a cross-entropy loss.

3.1.4 The residual network

The residual network in this thesis is loosely based on the state-of-the-art U-Net, which is further described in the paper [21]. The actual implementation is as stated in the code, based on a tutorial from Kjetil Åmdal-Sævik [1], and is illustrated in the following code snippet.

```
# U-Net model
# The implementation is based on:
# "https://www.kaggle.com/keegil/keras-u-net-starter-lb-0-277"

inputs = Input((int(config['data_processing']['x_pic']),
int(config['data_processing']['y_pic']), 3))

# Layers
conv_1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(inputs)
conv_1 = Dropout(0.1)(conv_1)
conv_1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_1)
pool_1 = MaxPooling2D((2, 2))(conv_1)
conv_2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(pool_1)
conv_2 = Dropout(0.1)(conv_2)
```

```
conv_2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_2)
pool_2 = MaxPooling2D((2, 2))(conv_2)

conv_3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(pool_2)
conv_3 = Dropout(0.2)(conv_3)
conv_3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_3)
pool_3 = MaxPooling2D((2, 2))(conv_3)

conv_4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(pool_3)
conv_4 = Dropout(0.2)(conv_4)
conv_4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_4)
pool_4 = MaxPooling2D(pool_size=(2, 2))(conv_4)

conv_5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(pool_4)
conv_5 = Dropout(0.3)(conv_5)
conv_5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_5)

up_6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(conv_5)
up_6 = concatenate([up_6, conv_4])
conv_6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(up_6)
conv_6 = Dropout(0.2)(conv_6)
conv_6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_6)

up_7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (conv_6)
up_7 = concatenate([up_7, conv_3])
conv_7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(up_7)
conv_7 = Dropout(0.2)(conv_7)
conv_7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_7)

up_8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(conv_7)
up_8 = concatenate([up_8, conv_2])
conv_8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(up_8)
conv_8 = Dropout(0.1)(conv_8)
conv_8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_8)

up_9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(conv_8)
up_9 = concatenate([up_9, conv_1], axis=3)
conv_9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(up_9)
conv_9 = Dropout(0.1)(conv_9)
conv_9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(conv_9)

outputs = Conv2D(1, (1, 1), activation='sigmoid')(conv_9)

```

```
model = Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

When implementing the network, the choice of loss and optimizer are set to be the same as for the sequential network. The implementation in this thesis is as earlier described based on the U-Net. It will further be referenced to as the residual network.

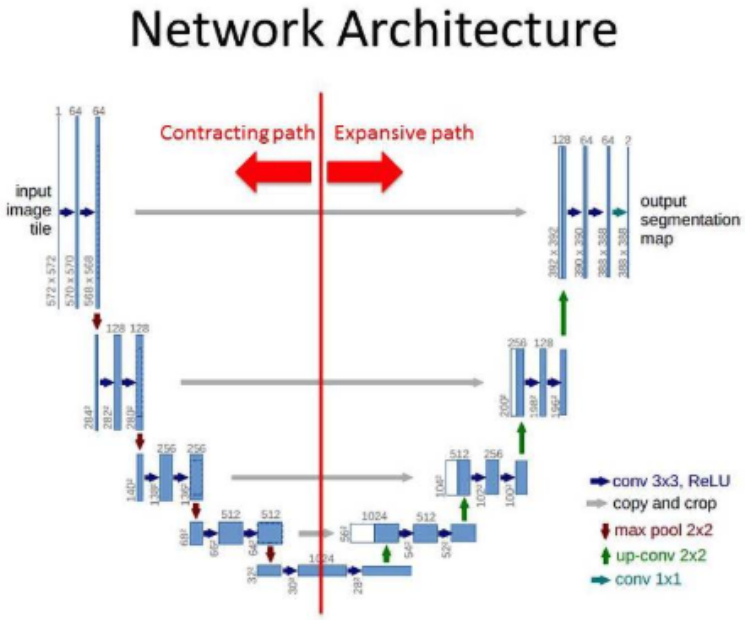


Figure 3.3: The U-Net architecture illustrated with skip connections [25]

When testing the performance in the residual network, it proved to be able to provide quite accurate predictions from few training samples. Gaining accurate predictions with few training images are the main priorities when implementing the network architecture. Having it successfully implemented will save a lot of time because it is sufficient to only manually annotate a few images. To have accurate predictions means the network performs well and can be able to provide better results in the form of good predictions.

An important difference between the sequential and the residual network, is the residual networks capability to provide skipping connections. These skipping connections transfers information from each downsampling layer, directly to its upsampling layer respectively, as illustrated with the gray arrows in fig. 3.3. The upsampling layers concatenates its existing information with the information from the skipping connections. As seen in fig. 3.4, each layer in a CNN extracts different types of features. When the U-Net architecture concatenates the local information provided from the skipping connections, with the global information from the upsampling layers; this results in an improved and efficient feature extraction.

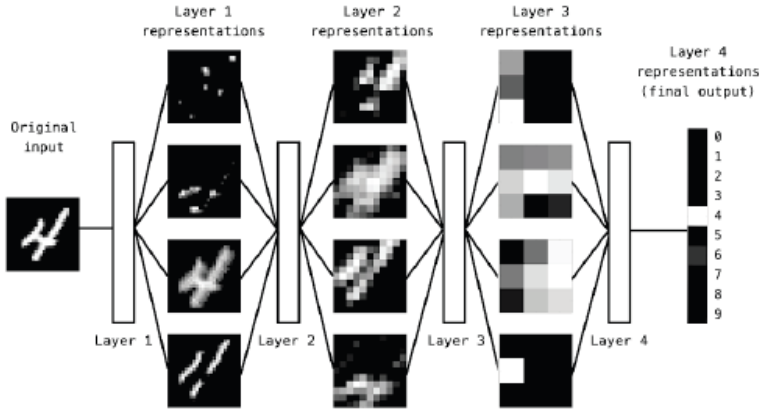


Figure 3.4: The features extracted from different layers in CNN [2]

3.1.5 Transfer learning

Before the transfer learning can start, the network needs to train on a dataset and save the weights respectively. This is done by training on the *Freiburg Forest* dataset [30]. The *Freiburg Forest* dataset is developed in the University of Freiburg in Germany. It consists of images from off-road environments taken from the Black Forest area in Schwarzwald. The reason why this dataset is such an interesting choice to use, is its similarities to the custom-made datasets used in this thesis. The custom-made datasets does also have images from an off-road scene.

When the training is finished the weights are saved and the fine tuning can begin after one last step. The last step is to change the training set to the original dataset used in this thesis.

```
1  # Transfer learning
2  if config['train/test/debug'].getboolean('transfer') is True:
3  model = load_model('models/' + config['train/test/debug']['weights'])
4  for layer in model.layers[:int(config['train/test/debug']['layers'])]:
5  layer.trainable = False
6  # Train model
7  model.fit(x_train, y_train, batch_size=10,
8  epochs=int(config['train/test/debug']['epochs']),
9  shuffle=True, validation_data=(x_val, y_val),
10 callbacks=callbacks_model)
```

As seen in the code snippet above, the model retrieves the saved weights from the training with the *Freiburg Forest* dataset in line three. Further it freezes all the layers up the chosen networks last layers, which means when the training starts again the weights from the frozen layers will not change. Then the network starts training with the original dataset in line seven. Since all the layers up to the last layers are frozen, only the layers after the frozen ones will change its weights during this training. This is called fine tuning.

3.1.6 Tensorboard

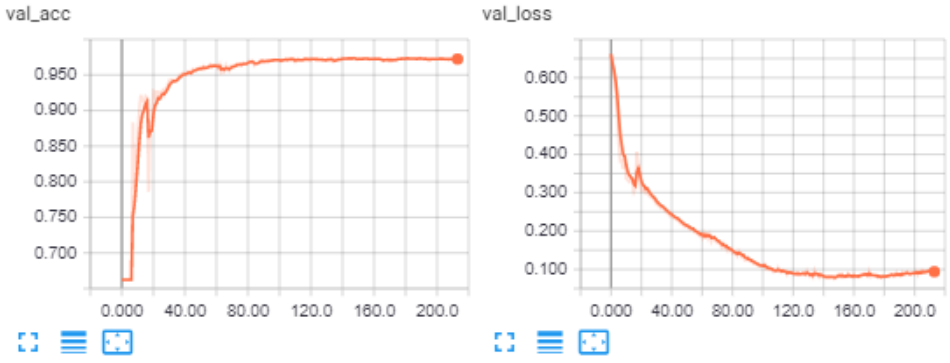
A handy way to gain control over the training, in order to tweak parameters and control the models accuracy, is to use Tensorboard. Tensorboard is a visual tool developed by TensorFlow providing full overview of the training, while training. It is easy to set up in the terminal. As illustrated in the code snippet below, Tensorboard is up and running with the following command:

```
tensorboard --logdir=<log>
```

The *"<log>"* must be replaced with the address to the log folder the algorithm provides.

When using Tensorboard with *Keras*, it is necessary to use *callbacks* in the code. A callback can provide a view on the internal states and statistics of the model, while training. Using callbacks, it is possible to decide which information/process to include and when it should be executed during the training process [29].

(a) Training accuracy and loss



(b) Validation accuracy and loss

Figure 3.5: Accuracy and loss for both training and validation visualized in Tensorboard

Two standard graphs used to check the quality of the DL model is *"accuracy"* and *"loss"*, which are found under the tab *"SCALARS"*. To have the opportunity to visualize these graphs during training, is an advantage. This is due to the possibility of easily spotting errors like *overfitting* or *underfitting*, and if the

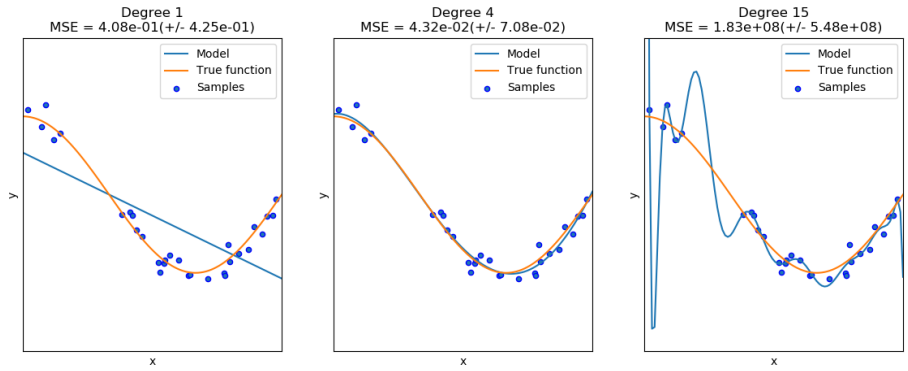


Figure 3.6: Underfitting, perfect sampling and overfitting [24]

accuracy in the model is fulfilling user expectations. Examples of *underfitting* from the left image, perfect sampling in the middle and overfitting in the right image, can be seen in fig. 3.6. As seen in fig. 3.5 the accuracy and loss in both training and validation fulfil expectations. By observing that the loss function steadily decreases into a smooth curve indicates that there is neither *overfitting* nor *underfitting* in the model.

Another tab in Tensorboard called *"GRAPHS"* shows the current networks architecture. The architecture from the sequential network is visualized in fig. 3.7. Being able to see all the layers and connections, helps the user see the whole network in a visualized perspective. There is possible to rename each layer in the model according to the user's own preference.

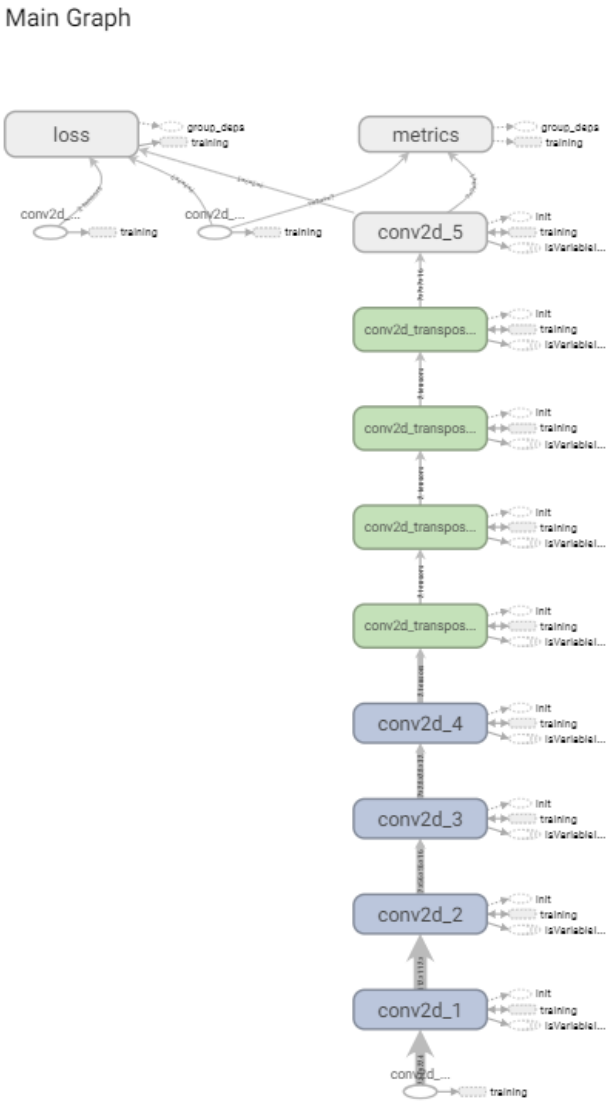


Figure 3.7: The sequential network architecture visualized in Tensorboard

3.2 Predictions

A prediction in this context is when the networks in this thesis produce a qualified guess of what value each pixel in the image shall be assigned, from either one of the two classes. This is the basis for what will become the automatically annotated images. After the network has updated its internal state satisfactorily, the training can stop, and it is ready to be fed new images it has never seen before. The network will then predict an output given the input of the new data, which will use the finished trained internal state of the network to predict. The predictions with sufficiently good results will in the end become the automatically annotated images this thesis tries to produce. When all the predictions are made, they will in the most cases contain some adverse features like different types of noise or choppy edges in the classification. To avoid these

types of artifacts several techniques from computer vision can be applied.

3.2.1 Thresholding

The first method used on the predictions is called thresholding. It removed the predicted pixels in the image which has a value below the given threshold. When the networks predict pixel values, this happens in the range between 0-1. The thresholding will convert that fuzzy logic to values which are either 0 or 1. Where class 0 is background and class 1 is road.

```
# Filter out values with less certainty than 65 %
prediction = np.where(prediction > 0.65, np.ones_like(prediction),
np.zeros_like(prediction))
```

$$f(x_i) = \begin{cases} 1 & x_i > 0.65 \\ 0 & x_i \leq 0.65 \end{cases} \tag{3.3}$$

What the code snippet above does, is to filter all values where x_i are the predicted values and assign them to either 0 or 1. All values over 0.65 are assigned to 1 and all values up to 0.65 are assigned to 0. This is described in eq. (3.3). The thresholding will therefore remove some noise and assign values up to the threshold as background instead of road.

3.2.2 Morphological operations

The next method uses is morphological operations. In this thesis it is applied on the road class prediction. In the cases where the road is a bit spiky, erosion is applied to trim down the edges slightly. The erosion operation is followed by a dilation which fills the boundary slightly more smoothly than it was before the morphological operations. The number of times erosion and dilation is applied is chosen in *iterations* as illustrated in the code snippet below. In this thesis every prediction gets three iterations of erosion followed by two iterations of dilation.

```
# Processing with morphological operations
kernel = np.ones((3, 3), np.uint8)
prediction = cv2.erode(prediction, kernel, iterations=3)
prediction = cv2.dilate(prediction, kernel, iterations=2)
```

As a bonus to trimming down spiky edges, the three iterations of erosion may be enough to remove very tiny bits of falsely classified road in the background, if it exists at all in the prediction. The kernel size used in this thesis is a 3x3-square.

3.2.3 Connected component analysis

The next operation used in order to remove larger sections of noise predicted in the image is called connected component analysis. When implemented in this thesis it removes all sections in the images which contains less than 5000 pixels. This means in the most cases that all of the remaining noise in the image will be removed. The connected component analysis has one disadvantage which is

when the road predicted is less than the threshold value of 5000 pixels. Then the whole prediction will consist of only background. This only happens in a very limited number of the images. The reason for setting the pixel threshold to 5000 became clear after testing different values. Lower values did not remove all the noise in the image, while much larger values could remove the correct predicted road.

3.2.4 Binary hole filling

The last computer vision technique used in this thesis is to fill all holes remaining in the prediction of the road. That operation is simply done by the following line of code:

```
predictions = scipy.ndimage.morphology.binary_fill_holes(predictions)
```

The operation is described in the Scipy documentation [23], and briefly explains that the algorithm uses binary dilations to find the boundary of the objects in the image, and further fills the holes inside the object.

3.3 Experiments

This last part of the method chapter will cover the experiments done in this thesis. In the form of describing the structure of the datasets, and the choices made regarding choosing the number of images in the train, test and validation sets. Further this last part of the section will go through the training configurations chosen during training.

3.3.1 Datasets

This thesis uses three datasets. Two of them contains only unlabeled images, while one of them has existing ground truth. The one with existing ground truth is the *Freiburg Forest* dataset briefly described above. This is a public dataset and suits this thesis due to its similar off-road environment.

As described in the website [7], this dataset was collected with a mobile robot platform capturing multi-spectral and multi-modal images from several cameras. The dataset contains images from three different days, providing the dataset with various lighting conditions. The dataset consists of pixel-wise ground truth for six classes, since this thesis wants to investigate a two-class problem containing road and background, the classes other than road will be masked as background. The second dataset is named *Custom 1*, and the last

Dataset	**Training-**	**Testing-**	**Validation-images**
Freiburg Forest dataset	207	136	23
Custom 1 dataset	40	394	5
Custom 2 dataset	147	1472 + 404	17

Table 3.1: Dataset structures

is titled *Custom 2*. As earlier stated these are two datasets originally consisting

of only unlabeled images (no ground truth). This thesis wants to investigate if there is an opportunity to automatically predict the annotations to these images given only a small portion of manually annotated images. The task of manually annotate lots of pictures is both time consuming and repetitive, and having this task automated by a network would be an advantage. The few images which are manually annotated in these two datasets are the ones in the training and validation sets respectively. The dataset structure is illustrated in table 3.1. In the table, it is possible to see explicitly the number of images in each directory. One can notice from each datasets directory distribution that the number of training images are significantly higher in the *Freiburg forest* dataset than in both the custom-made datasets. This means there is a higher probability that the *Freiburg Forest* dataset will provide a more generalized network than any of the two custom-made datasets, even though the residual network is known to provide good results with few training samples [26]. As discussed earlier, this thesis investigates the possibility of retrieving many automatically annotations from few training samples. The experimentation will consist of using the more generalized features gained from the *Freiburg Forest* dataset combined with the residual network implementation. This is done in order to hopefully retrieve a certain percentage of automatically generated annotations, with decent accuracy. The process is partly described above in the transfer learning section in this chapter.

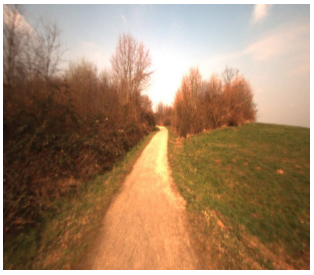


Figure 3.8: An image from the *Freiburg Forest* dataset

Figure 3.9: An image from the *Custom 1* dataset

Figure 3.10: An image from the *Custom 2* dataset

To get a visual understanding of which type of scene is included in each dataset an example image from each dataset is provided in fig. 3.8, fig. 3.9, and fig. 3.9. It is possible to see some similarities between all datasets, hence understand why the *Freiburg Forest* dataset is used as the dataset for the transfer learning. The *Custom 1* dataset is an off-road dataset contains 439 images. The scene consists of similar images like the one illustrated in fig. 3.9. The 45 manually annotated training and validation images are chosen as every tenth image ranging from start to end in the dataset. This means that small fractions from the full sequence may be included in the training set. This way of choosing training images is not a normal method in DL, but is used since the goal of this thesis is to mostly predict highly correlated data. The cost of choosing training data like this may reduce some of the network's generalization capability. The hypothesis is that since the data is very correlated, the network will still be able to predict the test images with satisfactorily high accuracy.

The last dataset used in the thesis is *Custom 2*. This dataset consists of 2040 images in total. It contains 164 manually annotated training and validation images that are chosen every tenth image ranging from start until number 1636 in the dataset. An example image from *Custom 2* is illustrated in fig. 3.10. The difference from *Custom 1* apart from having a slightly different type of scene, consists of a second test set containing 404 images. This test set is taken from the last sequence in the dataset and nothing of its sequence has been included in either the training or the validation set. This is done in order to check if the network is able to generalize well on completely new unseen data, which in theory should be completely uncorrelated from the images in the training set.

The datasets need to be structured in order to gain control over which data to use when training, or when testing. The structure used in this thesis is illustrated in fig. 3.11. This structure is equal for all three datasets with one exception. The *Freiburg Forest* dataset does also include ground truth images for the test set.

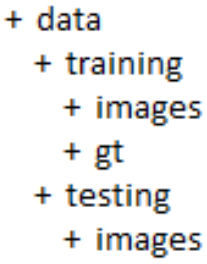


Figure 3.11: This thesis directory structure for the datasets

Note that from both custom-made datasets, the only images containing ground truth are the training images. Optimally all the images included in these datasets would have ground truth. In that way, the algorithm can be able to measure accuracy and loss in predictions both during training and testing. The reason why the ground truth only is included in the training set for both custom datasets is as earlier described due to the high cost of manually labeling an image. To check how good the predictions provided by the network are, this can only be done visually by comparing the road predictions to where the road actually is in the original image.

The training directory is further split into following list variables:

- *x_train*
- *y_train*
- *x_val*
- *y_val*

Where *x_train* and *y_train* refers to the training images and its ground truth respectively. *x_val* refers to validation images, and *y_val* is the ground truth for the validation images. The network will only train on the training set, then check accuracy on the validate set and predict values on the testing set. The validation and test set should on a general basis be data that the network never

has seen before. As described in the section above, only a few of the images were manually annotated. This means the dataset only has a small number of images to train on. Hence, the percentage of images split into the validation set is put to ten percent. Optimally should this number be higher but being able to train on most available images got a higher priority in this thesis. Another reason for experimenting with a low percentage of training images is to check how many images could be automatically annotated satisfactory with as few manually annotated training images as possible.

3.3.2 Training

Specification	**Description**
CPU	i7-6700k
GPU	GeForce GTX 1070
GPU-memory	8 GB
RAM	16 GB
Operating system	Windows 10

Table 3.2: System specifications

The computer used for the training has the following system specifications described in table 3.2. When the training is executed the different runs consists of a set of common parameters which are described in table 3.3.

Description	**Value/Name**
Loss	Binary cross entropy
Optimizer	adam
Resolution	224x224
Epochs	5000
Batch size	10
Activation function	ReLu
Output layer activation	Sigmoid

Table 3.3: The hyperparameters used in training

The experiment training cycles either utilize the sequential or the residual network in order to compare the two different networks in the end. The first two runs are with the *Freiburg Forest* dataset. After the training runs, the best weights are saved as h5 files ready for transfer learning. The next step is to load the weights and fine tune them for both the *Custom 1* and *Custom 2* dataset respectively. After the fine tuning is finished, the final weights must be saved, and the networks are prepared for the prediction of the unseen data.

Chapter 4

Results

This chapter will cover the results provided by the two networks implemented. Starting with the training results, before presenting the quantitative results from the *Freiburg Forest* dataset, and an indication of how the quantitative results could be in the *Custom 1* and *Custom 2* datasets. The last part of this chapter presents the qualitative results for some predictions in the form of visual presentation of the predicted images next to the original images.

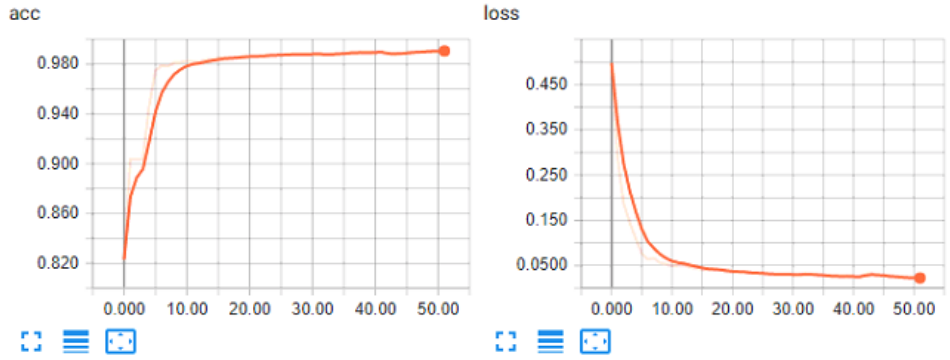
4.1 Accuracy and loss results from the training

This section will provide the accuracy and loss results from both the training and the validation set. The presentation of these training results will be divided into three parts, covering each dataset separately. Each figure presented will show through the accuracy on the y-axis in the left graph, and loss in the right graph. The x-axis represents the number of epochs [34] in all graphs.

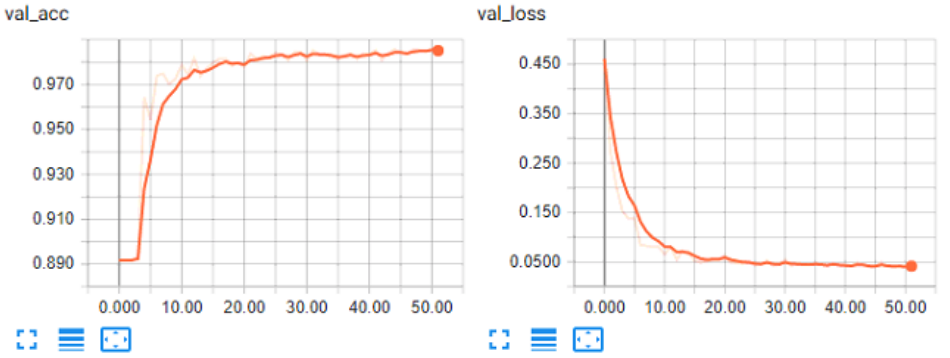
The two networks start training on the *Freiburg Forest* dataset. As presented in table 3.1 this dataset consists of 230 images split into 207 training images and 23 validation images. The training results from the residual network are visualized in fig. 4.1. Where *acc* (accuracy) and *loss* in fig. 4.1a belongs to the training images, and *val_acc* (validation accuracy) and *val_loss* (validation loss) in the fig. 4.1b to the validation images. The rest of the training results is presented in the same structure. The graphs in fig. 4.1a contains a high accuracy and a low loss, which is desirable. The same can be said about the validation accuracy and the validation loss in fig. 4.1b, which means the training did not do any overfitting. Summarized is this a successful training for the residual network from epoch 0 to epoch 50.

As illustrated in fig. 4.2 the sequential network follows almost the same lines as the residual networks graph, meaning this training is also considered successful. The epochs in this graph stretch from 0-160, and as seen in the validation loss around epoch 100, the curve is starting to ascent. The ascending indicates the start of possible overfitting because the training loss starts descending on approximately the same time. Therefore the training must be stopped, and the weights saved before that epoch.

The next dataset the two networks use in their training is the *Custom 1* dataset. The table 3.1 shows that this dataset consists of 40 training images, five



(a) Training accuracy and loss



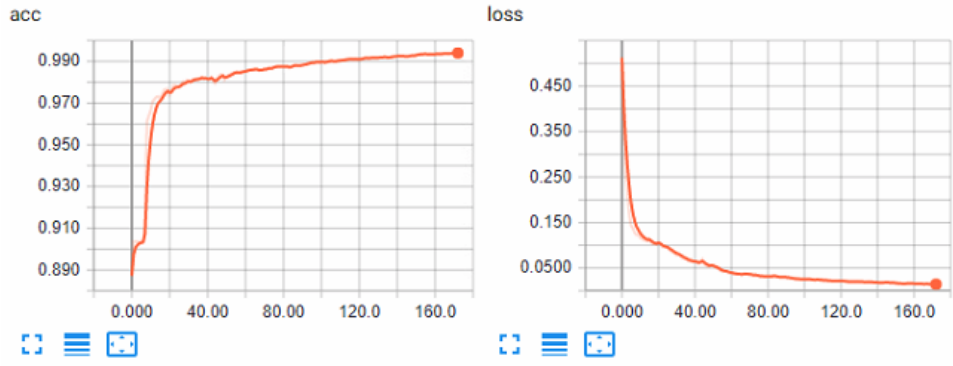
(b) Validation accuracy and loss

Figure 4.1: The training and validation set results from the residual network, trained on the *Freiburg Forest* dataset

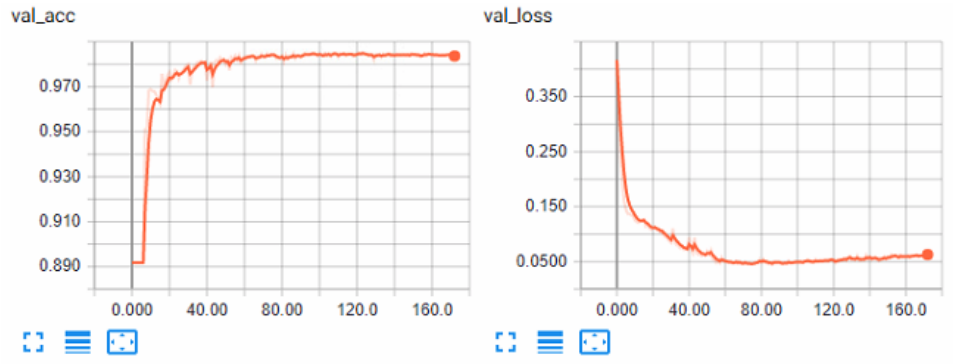
validation images, and 394 test images. Before each training starts, the weights saved from the previous run on the *Freiburg Forest* dataset are loaded into the current network. Then the most layers until the last few are frozen as described under the transfer learning section, and the network starts the training by fine tuning the weights of the last layers. As seen in fig. 4.3a this training provides an accuracy which converges close to 100%, and a low loss converging close to zero percent. The validation loss in fig. 4.3b is a little spiky but converge at around ten percent. The training loss in fig. 4.3a converges as well at approximately zero percent. In other words, this training does not look like potential overfitting.

The training results in fig. 4.4 are not as smooth as in fig. 4.3. The curves are a little bit spikier in general in this particular training. Despite the tendency to spiky graphs, no major overfitting is observed during this training of the sequential network.

The residual network training with the *Custom 2* dataset is also considered successful. The graphs in fig. 4.5a and fig. 4.5b illustrates a high accuracy in both training and validation images. The loss converges at around ten percent in the training images, and approximately 15 % in the validation pictures and no overfitting is observed. This training result is slightly worse than in the residual network training with the *Custom 1* dataset, but the *Custom 2* dataset consists of more images in both the training and the validation set.



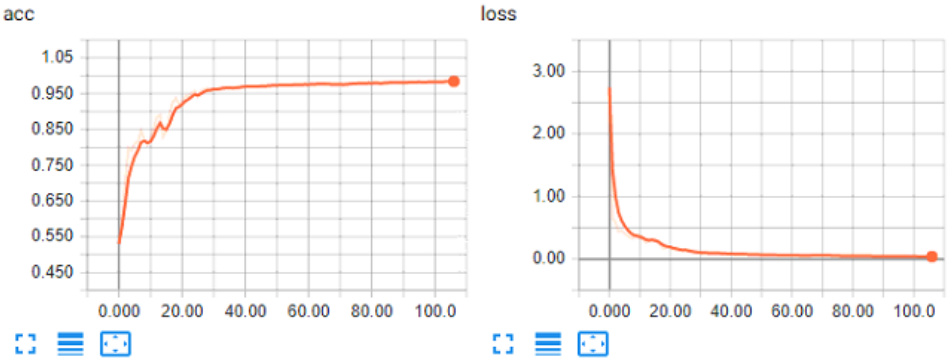
(a) Training accuracy and loss



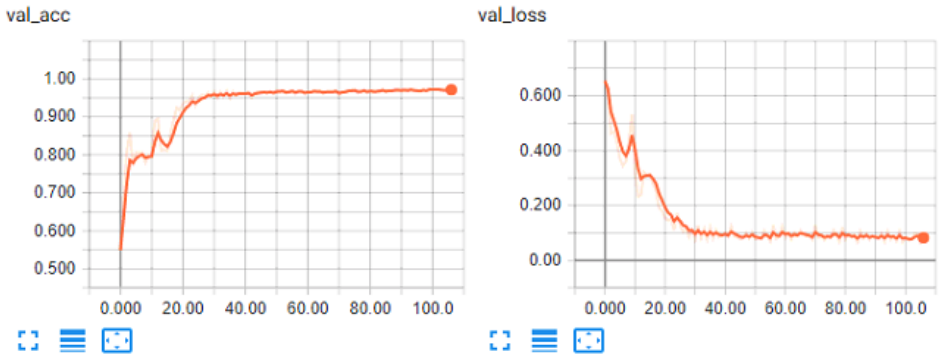
(b) Validation accuracy and loss

Figure 4.2: The training and validation set results from the sequential network, trained on the *Freiburg Forest* dataset

The last training is the sequential network with the *Custom 2* dataset, illustrated in fig. 4.6. This training is quite similar to the previous one except the accuracy is slightly lower, and the loss is barely higher.



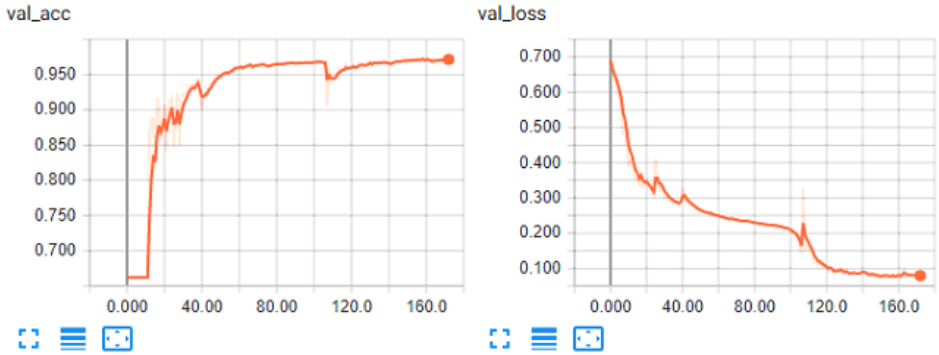
(a) Training accuracy and loss



(b) Validation accuracy and loss

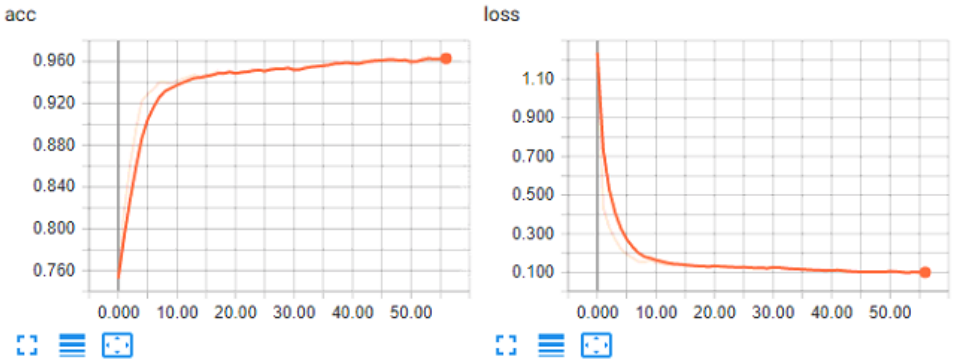
Figure 4.3: The training and validation set results from the residual network, trained on the *Custom 1* dataset

(a) Training accuracy and loss

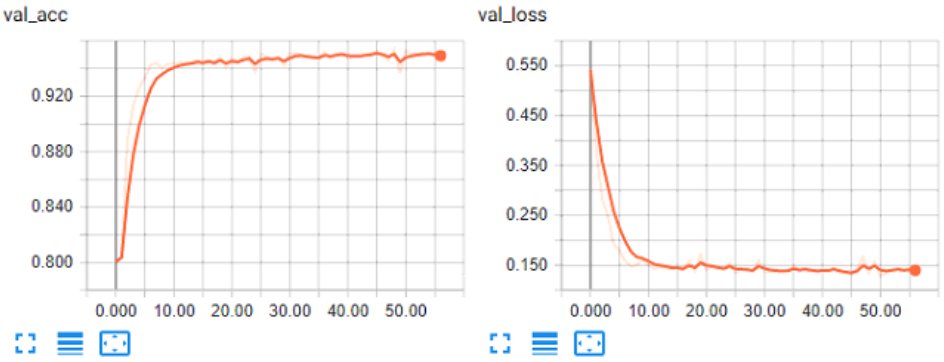


(b) Validation accuracy and loss

Figure 4.4: The training and validation set results from the sequential network, trained on the *Custom 1* dataset

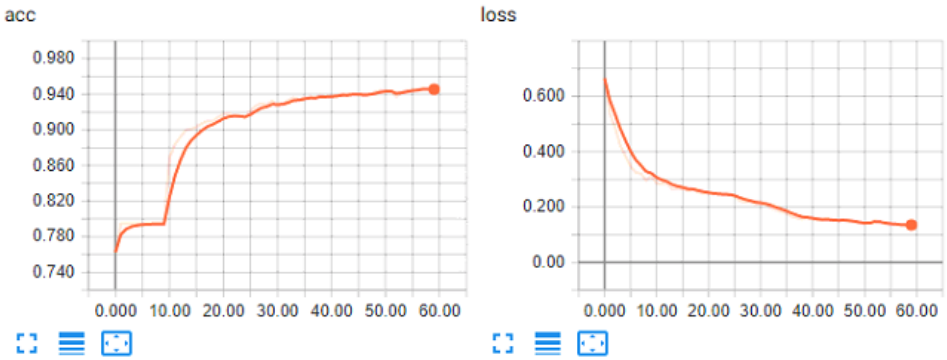


(a) Training accuracy and loss

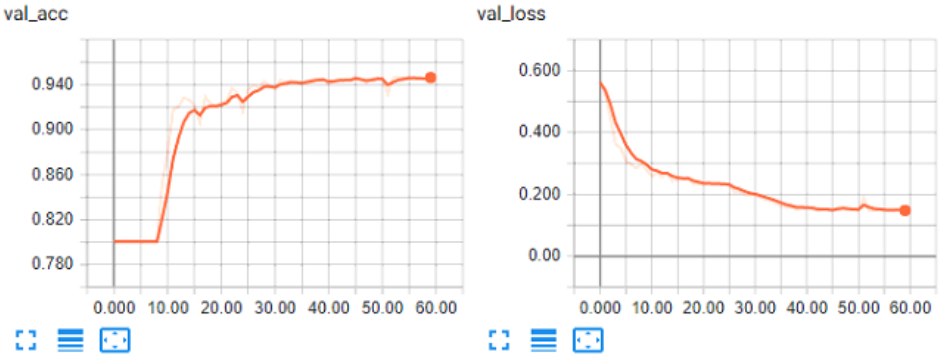


(b) Validation accuracy and loss

Figure 4.5: The training and validation set results from the residual network, trained on the *Custom 2* dataset



(a) Training accuracy and loss



(b) Validation accuracy and loss

Figure 4.6: The training and validation set results from the sequential network, trained on the *Custom 2* dataset

4.2 Quantitative results

This section will provide quantitative results from the three datasets. The result from the *Freiburg Forest* dataset provides the quantitative result from all of the test images in the dataset. As earlier described, a test image is an image the networks have never seen before. The results from the *Custom 1* and the *Custom 2* datasets will only be an indication of the quantitative results. The reason for this is the lack of ground truth in the 2270 test images in total, coming from the two datasets. To be able to make an indication, five uncorrelated images from *Custom 1* dataset, and ten uncorrelated images from *Custom 2* dataset are manually annotated. Then the accuracy, the intersection over union (IoU) for both classes and the mean intersection over union (mIoU) are calculated for each test set. The results are presented in table 4.1, table 4.2, table 4.3 and table 4.4. The accuracy is described mathematically in eq. (4.1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Here TP, TN, FP and FN are true positive, true negative, false positive and false negative respectively. This quantity has a potential weakness in some segmentation problems. If for example one of the classes in a two-class problem, is much larger than the other class, the accuracy might turn out to be higher than it should be. The reason is that both classes are in the equation, and if the much larger class does a useful classification while the other class does not, the accuracy will still be high. To prevent this potential problem, the IoU metric is included in the tables. The metric is given by eq. (4.2):

$$IoU = \frac{TP}{TP + FP + FN} \tag{4.2}$$

Opposite to the accuracy metric, the IoU only includes one of the two true classifications in a two-class problem. Having only true positives (TP) in the numerator means as briefly described above that this metric might describe each class with a more accurate accuracy in segmentation problems. This is especially applicable when one of the classes is much larger than the other. Imagine the two-class problem in the thesis, where the classes predicted is road and background. If the background is 90 % of the image and receives a good classification while the road classification receives a bad classification, the accuracy might end up incorrectly too high.

$$Confusion\,matrix = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \tag{4.3}$$

For a broader understanding on the eq. (4.2), this thesis confusion matrix is introduced in fig. 4.7 and eq. (4.3). In this notation the *non-road* represents the background, and the *actual class* is the ground truth. In eq. (4.3) the contents of the matrix are substituted. Where c_{11} denote the true positive, c_{12} the false positive, c_{21} is false negative and c_{22} the true negative. In this thesis, the true positive of the confusion matrix is when both the prediction and the ground truth agree that the pixel is the road. The true negative is when both the prediction and the ground truth agrees that the pixel is the background. The

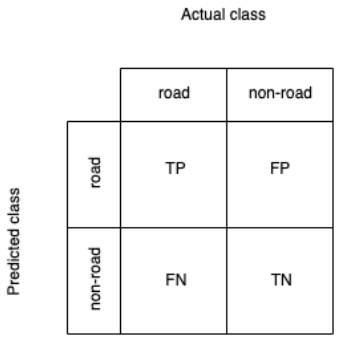


Figure 4.7: This thesis confusion matrix

correct classifications from the road and the background will be shown in the diagonal of the matrix respectively (c_{11} and c_{22}).

$$IoU_{road} = \frac{c_{11}}{c_{11} + c_{12} + c_{21}} \tag{4.4}$$

$$IoU_{background} = \frac{c_{22}}{c_{22} + c_{21} + c_{12}} \tag{4.5}$$

Equation (4.4) and eq. (4.5) describes the IoU for each of the two classes explicitly. These two equations are the ones that present the IoU in the following tables. The equations consist of the notation substituted from the confusion matrix.

$$mIoU = \frac{IoU_{road} + IoU_{background}}{2} \tag{4.6}$$

The last metric used for presenting the quantitative results is the mIoU. As described in eq. (4.6) it takes the mean of both the IoU classes.

	Accuracy	IoU_{road}	$IoU_{background}$	mIoU
Residual network	98.2%	75.9 %	98 %	86.9 %
Sequential network	89.4 %	47%	89%	67.9%

Table 4.1: The test set result from the *Freiburg Forest* dataset

The first quantitative results presented in this thesis provide the results from what the two networks managed to predict from the test images with the *Freiburg Forest* dataset. The results are presented in table 4.1. As this is the only quantitative results presented in this thesis which is not just an indication, these results should be considered most accurate in showing the complete accuracy of the dataset. As earlier mentioned *Freiburg Forest* dataset is the only one that contains ground truth for the whole test set. As seen in table 4.1, the residual network does an overall better performance compared to the sequential network.

The next quantitative results presented are from the predictions on the *Custom 1* dataset. Since these measurements only come from five uncorrelated images in the test set, the results can only be interpreted as an indication of

	Accuracy	IoU_{road}	$IoU_{background}$	mIoU
Residual network	97.7%	92.5%	96.6%	94.5%
Sequential network	98%	93.6%	97.1%	95.3%

Table 4.2: The test set result from the *Custom 1* dataset

what the actual quantitative result would be. In this indication, the sequential network performs slightly better than the residual network. Both networks have an overall high accuracy with its predictions on the *Custom 1* dataset. Why the sequential network had a slightly better performance metrics than the residual network will be further discussed in chapter 5.

	Accuracy	IoU_{road}	$IoU_{background}$	mIoU
Residual network	94.1%	77.7%	92.7%	85.3%
Sequential network	94%	77.4%	92.6%	85%

Table 4.3: The test set result from the *Custom 2* dataset

	Accuracy	IoU_{road}	$IoU_{background}$	mIoU
Residual network	90.5%	65.2%	88.6%	76.9%
Sequential network	91%	66.3%	89.3%	77.7%

Table 4.4: The second test set result from the *Custom 2* dataset

The last indication of the quantitative results is presented in table 4.3 and table 4.4, and consists of predictions from the *Custom 2* dataset. The overall results from these two tables are also considered a good result. This dataset has combined 1876 test images, while the number of training and validation images is 164. The first table shows slightly better results overall, but this is as expected due to the last test set (404 images) being completely independent from the training and validation images.

4.3 Qualitative results

This part will consist of a few qualitative results from both network predictions on all datasets. The results presented in this section consist of some good and some poor predictions, where most of the poor predictions comes from more challenging scenes which includes different types of objects in the road. A more extended image sequence from each dataset which can be used to observe the results from the predictions can be found in the appendix. The extended image sequence is added so the reader may get a better overview of the result from the predictions.

The first qualitative result comes from the *Freiburg Forest* dataset, and is illustrated in fig. 4.8. Comparing the two networks from fig. 4.8a and fig. 4.8b, the residual network gives a more precise prediction of the road than the sequential network. This single result corresponds to the metrics in table 4.1, where the residual network did get better results overall as well.

(a) Residual network

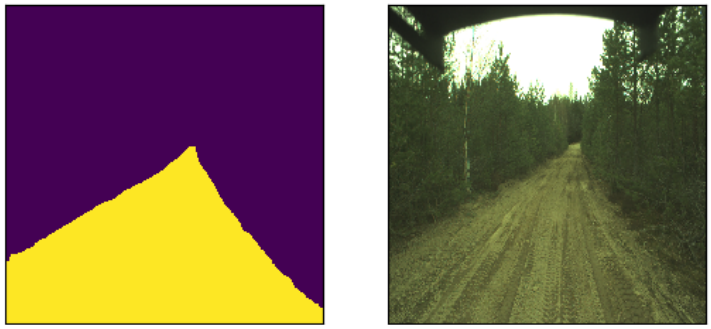
(b) Sequential network

Figure 4.8: Two samples from the test set predictions on the *Freiburg Forest* dataset

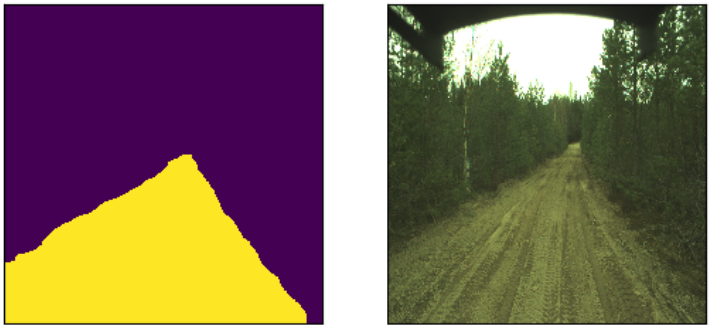
The next two predictions come from the *Custom 1* dataset. Here, the residual network provide a better prediction on the test set sample than the sequential network. The sequential network stops at predicting the road before the residual network does, in the tip of the road. A slight color change can be seen in that small area, which might explain why. In the indication of the quantitative results shown in table 4.2, the sequential network gave a slightly more accurate result than the residual network, and as seen in fig. 4.9 this does not correspond to the qualitative result were the residual prediction was better. Some theories about this will be discussed in chapter 5.

Figure 4.10 illustrates two predictions with an object (car) placed in the road. Both networks perform less than normal every time unknown objects arrive in the scene, but the residual network makes a surprisingly good prediction in fig. 4.10a compared to the sequential network. From other examples, it seems like both networks finds it challenging at first when new objects arrive at the scene before they quickly get back to good predictions. The residual network is slightly faster and more robust than the sequential network when it comes to resume good predictions after challenging scenes.

Figure 4.11 shows two examples of a challenging scene containing shadows, where the networks predict rather well. The large shadow part in the bottom of the images does not seem to concern any of the networks. The overall best prediction in this example comes from the residual network.



(a) Residual network



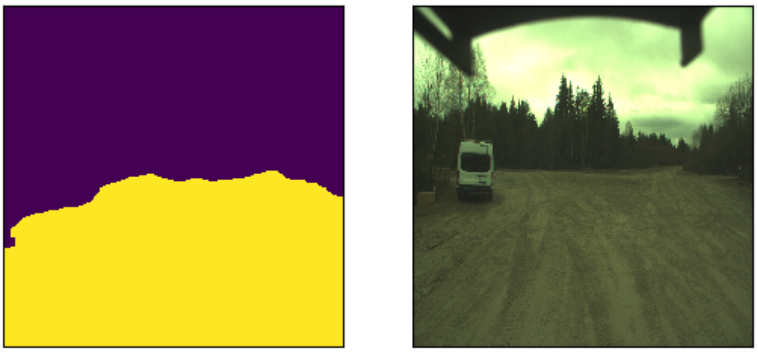
(b) Sequential network

Figure 4.9: Two samples from the test set predictions on the *Custom 1* dataset

Figure 4.12 illustrates a similar challenging scene as the example above in fig. 4.11. The main difference between the two results is which test set they come from. These samples are taken from the test set which only consists of 404 images and does not have any correlation to the training and validation images at all. As earlier described, the training and validation images are chosen iteratively from the start to the end of the dataset, meaning, for example, every 100'th in the dataset is chosen for training and validation. This method is chosen due to the desire for automatically predicting the rest of the dataset as annotation for the test images. Unlike the other two test sets, this test set has been left alone and consists of an untouched sequence. Being able to predict a good prediction on such an untouched scene as illustrated in these two images, means the network is somewhat generalized and has the opportunity to predict well on more completely unseen and uncorrelated data as well.

These two predictions illustrated in fig. 4.13 are examples of a challenging scene for the networks. The scene is from a U-turn made by the vehicle, and there is not a structured line in the road as it consists of frequently small areas with grass. The accuracy of the prediction in these particular scenes is, therefore, more reduced than in the rest of the datasets. The difference between the residual and the sequential network is also here noticeable. Once again, the residual network shows slightly better results.

(a) Residual network



(b) Sequential network

Figure 4.10: Two samples from the test set predictions on the *Custom 1* dataset. Containing a car in the image

(a) Residual network

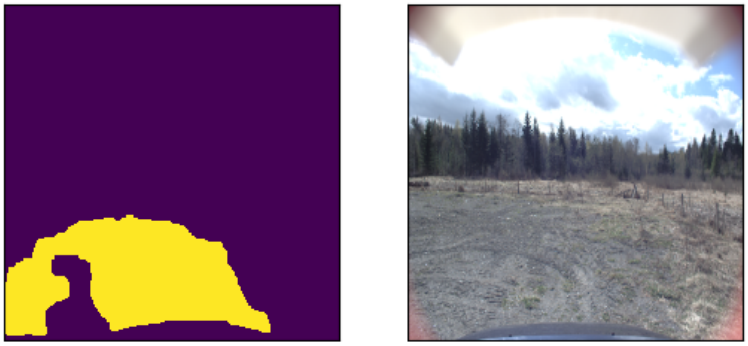
(b) Sequential network

Figure 4.11: Two samples from the test set predictions on the *Custom 2* dataset (The first test set with 1472 images)

(a) Residual network

(b) Sequential network

Figure 4.12: Two samples from the test set predictions on the *Custom 2* dataset (The second test set with 404 images)



(a) Residual network

(b) Sequential network

Figure 4.13: Two poor test set prediction from the *Custom 2* dataset (The first test set with 1472 images)

Chapter 5

Discussion of results

This chapter will first examine the quantitative and then the qualitative results presented in the thesis. The next section will provide an analysis of the use of transfer learning and discuss challenging scenes from the images. Then the next part will discuss the quality of the manually annotated images and the generation of the training and the validation sets. Lastly, this chapter will conclude the thesis and provide suggestions for further work.

The chapter will answer the goals which not yet has been answered in chapter 3 on page 23. The goals which also are described in the introduction are as follows:

1. Perform a literature review of existing CNNs and select a robust network to implement
2. Implement a standard CNN as baseline
3. Investigate whether the CNNs predictions have the necessary accuracy to be used as ground truth for the images in the custom-made datasets
4. Obtain a more generalized network with transfer learning from a similar dataset
5. Testing the networks, then presenting and analyzing the results

5.1 Quantitative results

It is necessary to have ground truth for all of the test images to calculate the quantitative results. The only dataset with ground truth for all test images is as earlier described the dataset from the University of Freiburg. Consequently, the *Freiburg Forest* dataset is the only one that gets the actual quantitative results from the networks in this task, and therefore should these quantitative results be emphasized most. The results were presented in section 4.2 on page 46 from table 4.1 on page 47. As seen in table 4.1 on page 47, the residual outperforms the sequential network in all metrics, as expected. The scene in the *Freiburg Forest* dataset contains several challenging areas to predict, like for example narrow roads, sunlit areas, and much grass in the path. These challenging scenes appear more often in the *Freiburg Forest* dataset than in the *Custom 1*

and *Custom 2* dataset and may be one of the reasons for the residual networks greater performance.

As earlier mentioned, the results in section 4.2 on page 46 from the *Custom 1* and the *Custom 2* dataset will only be an indication of the quantitative result. The reason for only providing an indication is due to the time cost of producing manually labeled images, as none are included in the test set. Five uncorrelated images are chosen from the test set to receive ground truth, and the metrics are then calculated in a small script. The results presented in table 4.2 on page 48 describes the two networks quantitative results from the test images in the *Custom 1* dataset. Compared to the *Custom 2* with its 2040 images, the *Custom 1* dataset has only 439 images in total. The quantitative result from table 4.2 on page 48 shows an overall high accuracy from both networks. The high accuracy from both networks is the desired result. As seen in table 4.2 on page 48, the sequential network has overall slightly higher accuracy than the residual network, and this is not as excepted since the residual network should be the better one. A reason for the slightly better accuracy in the sequential network may be that the images selected from the test set contained the best predictions from the sequential network, while the best from the residual network was not included. Another theory can be that the FP and the FN values vary between the networks. Low FP and high FN values from the first network, and opposite values in the form of correlated high FP and low FN values in the second network might provide similar IoU accuracy. FP is the predictions where the network falsely think it is road, and FN is the predictions where the network falsely think it is the background. The most unwanted scenario is where the FP values are too high because it is not desired to have a falsely predicted road where it is no road. The residual network had $FP = 156\ pixels$ and $FN = 994\ pixels$, while the sequential network had $FP = 534\ pixels$ and $FN = 439\ pixels$ in the *Custom 1* dataset. The mean values from the actual road are $TP = 14828\ pixels$, while the mean actual background values are $TN = 34215\ pixels$. Put in context, the values from FP are low compared to TP, which means there is only a tiny fraction of the image, which is falsely predicted as the road. Since the FP values from the residual network are lower than from the sequential network, it may indicate that the results from the residual network are better, despite the slightly higher accuracy in table 4.2 from the sequential network.

The two last indications of quantitative results are presented in table 4.3 on page 48 and table 4.4 on page 48, and comes from two different test set in the *Custom 2* dataset. The metrics are calculated from ten manually annotated images in the first test set, and five manually labeled images from the second test set, all images without correlation to each other. The results in table 4.3 on page 48 and table 4.4 on page 48 are slightly lower than in table 4.2 on page 48, but this is as expected since the *Custom 2* dataset consists of totally 2040 images with occasionally a more challenging scene.

As seen in the quantitative results from table 4.1 on page 47, the residual network was better than the sequential network. If it had been ground truth for all test images in the *Custom 1* and the *Custom 2* dataset, may the likelihood exist for seeing the same pattern in the networks quantitative results from these datasets as well. The reason to anticipate this outcome is supported further

in the qualitative results presented, where it is illustrated that the predictions from the sequential network are poorer than the residual network's predictions in more challenging scenes.

5.2 Qualitative results

The most expedient results to interpret in this thesis are the qualitative results since they provide a clear, explicit illustration of the quality of the predictions from the networks. As there may be several reasons for the quantitative results to provide falsely high accuracy in some occasions, the qualitative results presented as images with predicted ground truth gives a better understanding if the prediction fulfills the users' requirements. As illustrated in fig. 4.9 on page 50, the residual network makes overall better predictions than the sequential network, and similar examples of the same outcome will be illustrated in the appendix under section A.1 on page 69. Providing overall better predictions than the sequential network, the residual network is the most robust network in this thesis. Its predictions have high accuracy in the most scenes in the *Custom 1* and *Custom 2* datasets testing images.

5.3 Transfer learning

Before implementing the transfer learning from the networks training on the *Freiburg Forest* dataset, the predictions from both networks were a little inadequate on the two custom datasets. These predictions missed a little bit more of the road areas close to the trenches. The features the two networks learned from the *Freiburg Forest* datasets 207 training images from a similar environment as in the *Custom 1* and *Custom 2* dataset, helped improved the overall accuracy in the predictions. For that reason, transfer learning from a similar dataset may be considered a good idea, and it proved to give slightly more generalized networks in this thesis.

5.4 Challenging scenes

The *Custom 1* and *Custom 2* dataset contains several images with a challenging scene for the semantic segmentation problem in this thesis. The first one is presented in fig. 4.10 on page 51, where a car arrives at the top left position in the image. Since this thesis only handles a two-class problem, it was uncertain how the networks would react to a car in the scene. Figure 4.10 on page 51 shows that the residual network managed to get a good prediction, while the sequential network predicted the image with slightly lower accuracy.

Another comprehensive scene is presented in fig. 4.11 on page 52, where the road consists of shadows. In traditional ML problems, a scene like this may cause a poorer prediction due to the color change from the shadows, but both the residual and the sequential network managed to predict a reasonably good ground truth for the image. The residual network also has a slightly more accurate prediction in this case.

The last type of challenging scene which was presented in the result chapter was fig. 4.13 on page 54. The image illustrates a U-turn made by the vehicle were the road is partly filled with grass. This type of environment proved to be the most comprehensive for both networks. Even though the residual network had the best prediction of the two networks, was this not a particularly accurate road prediction. Some of the reasons for this poor prediction may be the lack of either a similar image, including ground truth in the training set or the weak lines in the road due to the grass in the scene.

Even though the result chapter illustrates some images from challenging scenes with variable accuracy on the predictions, did the two networks provide overall good results in the most predictions as will be illustrated in chapter A on page 69. These examples show that the residual network is slightly more robust than the sequential network in both the more challenging scenes and the ordinary scenes. A method which may increase the robustness and accuracy in all scenes even more, is implementing several classes. If there would be classes for the most common objects which arrive in the dataset's scene, this could help the networks considerably to predict more accurate when these objects arrive. It will require more work on the manual annotation part in the images from the training and validation set, but it is likely to result in better predictions in scenes with these objects.

5.5 The few manually annotated images from the *Custom 1* and *Custom 2* datasets

As illustrated in table 3.1 on page 34 the *Custom 1* dataset consists of 45 manually annotated images from the training and validation set, while the *Custom 2* dataset consists of 164 images. That makes the total number of manually labeled images used for training and validation, 209 images. These are the images that are used to train the two networks, and further potentially automatically produce ground truth for up to 2270 test images from the two datasets. To have these manually annotated images labeled correctly is therefore essential to have the highest chance of successful predictions. The average time spent on manually label each of these training and validation images is around two minutes, which implies that some of the annotations might have the opportunity for improvement. Since the result from the residual network's predictions has decent accuracy, it suggests that the manually labeled images are to some extent accurate enough.

5.6 Generating the training and validation set

Since the two custom datasets only contain unlabeled images initially, each dataset needs to be split into a training, a validation and a test set. The dataset structure is briefly described in section 3.3.1 on page 34, and the training set is the first one which is generated. Depending on the networks goals, it is several ways to generate the training set. In this thesis, the goal is to make the networks predict the ground truth for the test images. Therefore, an uniform sampling of the training images approximates the entire test set. This is why every tenth

image ranges from the beginning to the end of the dataset (in the *Custom 1* and the *Custom 2* datasets) is chosen as training images. The remaining images in both datasets are chosen to be in the test set. The validation set is generated by taking ten percent of the images from the training set. Having a training set which is slightly correlated to the test images may increase the chances for better predictions in the two custom datasets. Figure 4.9 on page 50, fig. 4.10 on page 51, fig. 4.11 on page 52 and fig. 4.13 on page 54 are examples of predictions from test sets which is slightly correlated to the training set. All these result with the exception of the image in fig. 4.13 on page 54 are considered decent predictions from the residual network, and slightly poorer but still okay predictions from the sequential network. These results indicate that generating a training set this way works if the goal is to find the ground truth in a test set with slightly correlated images to the training set.

There is created a second test set in the *Custom 2* dataset consisting of 404 images, which are split from the dataset before the training set was created. Having the second test set split before the training set was made means it may be completely uncorrelated from the training images. The reason this second test set is created is to check if the networks are generalized and can perform well on completely uncorrelated data. A result from this test set is presented in fig. 4.12 on page 53 and the prediction from the residual network is surprisingly decent, considering that the test set is entirely uncorrelated with the training set.

Chapter 6

Conclusion

CNNs with supervised learning is a powerful tool for gaining desired features from images. When applying supervised learning to a semantic segmentation problem, the dataset should include ground truth for the images. Providing ground truth is normally a time consuming task, so many datasets consists only of unlabeled images. This project applies two unlabeled datasets with images from a rural scene, and seeks to partly automate the task of assigning the ground truth for most of these images. The idea is to manually label a minor portion of the unlabeled images which further is used to fine tune the networks features learned from the *Freiburg Forest* dataset, and then assign the remaining labels for the unlabeled images.

In this thesis, two networks are implemented which purpose is to make the road and background predictions from the test images in such a detailed level that it can be used as ground truth for the images. The benefits of successfully providing such accurate predictions, is a potentially enormous amount of time saved instead of manually annotating all the images in the dataset.

After testing the two networks and presenting some of the prediction results in chapter 4 on page 39 and chapter A on page 69, the conclusion is that most of the residual network's predictions can be used as ground truth for the test images, which answers the problem addressed in the section 1.2 on page 1. However, there is potential for further improvements on the predictions for achieving a more accurate result. A small portion of the images from the most comprehensive scenes gave predictions which was not sufficient for further use as ground truth. In general the residual network performed consistently better compared to the sequential network, thus it was implemented as a baseline. Even though the sequential network delivered similar or slightly better results with respect to the indicated quantitative performance, the qualitative results revealed that the residual network was more accurate and robust. The result from this thesis will hopefully be useful for other institutions/projects which seek to automate the task of producing ground truth for the images in their unlabeled dataset.

Even though the residual network managed to provide many useful predictions, there is always room for further improvement. The first suggestion for improving the predictions and making the networks more robust would be to include more classes in the annotated images. As earlier described, including

other classes, which have a high probability for appearing in the scene, can contribute to making the networks more robust in these challenging scenes. It will take increased time to annotate each image used for training manually but the benefit may be worth the extra time. Spending slightly more time in general on each manual annotation can also be beneficial, and by that make sure the ground truth for the training images is more accurate.

Another task would be to perform more tweaking of the hyperparameters such as the number of epochs during training, the momentum, resizing the images, test other optimizers than adam, the learning rate, and the regularization. Testing different hyperparameters can be a repetitive task, but may provide increasingly better results instantaneously, which is rewarding. Experimenting with different types of kernels in the morphological operations on the predictions could also be another hyperparameter which would possible provide better results. The square morphological kernel used in this thesis could for example be replaced by a circle.

The most central suggestion for future improvements is to test if the implementation of an active learning algorithm would be able to improve the results further. The idea would be to use the stream based selective sampling method. This method takes each unlabeled image and predicts each one at a time, while the algorithm evaluates the informativeness of each image against their query parameter. Further, the user can decide whether to keep or reject the predictions and rerun the process with the accepted predictions added to the training set as ground truth. The networks would then have more images at each run for training, which hopefully could improve the accuracy even further on the next predictions.

Bibliography

[1] Kjetil Åmdal-Sævik. *Keras U-Net starter - LB 0.277*. URL: https://www.kaggle.com/keegil/keras-u-net-starter-lb-0-277 (visited on 04/30/2019).

[2] François Chollet. *Deep Learning with Python*. Manning Publications, 2017. ISBN: 9781617294433.

[3] *Cityscapes dataset*. URL: https://www.cityscapes-dataset.com/ (visited on 05/23/2019).

[4] *Color Images SteganalysisUsing RGB Channel Geometric Transformation Measures*. URL: https://www.researchgate.net/publication/293043690_Color_images_steganalysis_using_rgb_channel_geometric_transformation_measures (visited on 04/22/2019).

[5] *Convolutional Neural Networks: Architectures, Convolution / Pooling Layers*. URL: http://cs231n.github.io/convolutional-networks/ (visited on 04/04/2019).

[6] *Deep Networks Can Resemble Human Feed-forward Vision in Invariant Object Recognition*. URL: https://www.nature.com/articles/srep32672.pdf (visited on 05/14/2019).

[7] *DeepScene*. URL: http://deepscene.cs.uni-freiburg.de/ (visited on 05/09/2019).

[8] *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. URL: http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf (visited on 04/22/2019).

[9] Kai Olav Ellefsen. *Intro to machine learning and single-layer neural networks*. URL: https://www.uio.no/studier/emner/matnat/ifi/INF3490/h18/timeplan/slides/lecture5-6pp.pdf (visited on 04/08/2019).

[10] Kai Olav Ellefsen. *Multi-Layer Neural Networks*. URL: https://www.uio.no/studier/emner/matnat/ifi/INF3490/h18/timeplan/slides/lecture6-6pp.pdf (visited on 04/08/2019).

[11] *Fully Convolutional Networks for Semantic Segmentation*. URL: https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf (visited on 05/22/2019).

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[13] *ImageNet Classification with Deep Convolutional Neural Networks*. URL: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf (visited on 04/16/2019).

[14] Nikola Latinovic. *The significant difference between AI, ML and Deep Learning*. Feb. 2018. URL: https://www.usoft.com/blog/difference-between-ai-ml-and-deep-learning.

[15] Tom Mitchell and McGraw Hill. *Machine Learning*. http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html. 1997. ISBN: 0070428077.

[16] *Neural Networks Part 1: Setting up the Architecture*. URL: http://cs231n.github.io/neural-networks-1/ (visited on 04/15/2019).

[17] *Our U-net wins two Challenges at ISBI 2015*. URL: https://lmb.informatik.uni-freiburg.de/people/ronneber/isbi2015/ (visited on 05/23/2019).

[18] *Rethinking Atrous Convolution for Semantic Image Segmentation*. URL: https://arxiv.org/pdf/1706.05587.pdf (visited on 05/22/2019).

[19] G. X. Ritter and J. N. Wilson. *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, Boca Raton, 2nd edition, 2000.

[20] *Road/Lane Detection Evaluation 2013*. URL: http://www.cvlibs.net/datasets/kitti/eval_road.php (visited on 05/23/2019).

[21] O. Ronneberger, P.Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. LNCS. (available on arXiv:1505.04597 [cs.CV]). Springer, 2015, pp. 234–241. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1." In: ed. by David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning Internal Representations by Error Propagation, pp. 318–362. ISBN: 0-262-68053-X. URL: http://dl.acm.org/citation.cfm?id=104279.104293.

[23] *scipy.ndimage.morphology.binary_fill_holes*. 2015. URL: https://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.ndimage.morphology.binary_fill_holes.html (visited on 05/02/2019).

[24] *Underfitting vs. Overfitting*. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html (visited on 03/29/2019).

[25] *U-Net*. URL: http://www.deeplearning.net/tutorial/unet.html (visited on 05/01/2019).

[26] *U-Net: Convolutional Networks for Biomedical Image Segmentation*. URL: https://arxiv.org/pdf/1505.04597.pdf (visited on 05/06/2019).

[27] *U-Net: Convolutional Networks for Biomedical Image Segmentation*. URL: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/ (visited on 04/21/2019).

[28] *UNIK 4690 – Maskinsyn, Introduction*. Jan. 2018. URL: https://www.uio.no/studier/emner/matnat/its/UNIK4690/v18/lectures/lecture_00/lecture_0_introduction.pdf (visited on 03/14/2019).

[29] *Usage of callbacks*. URL: https://keras.io/callbacks/#tensorboard (visited on 03/20/2019).

[30] Abhinav Valada et al. "Deep Multispectral Semantic Scene Understanding of Forested Environments using Multimodal Fusion." In: *International Symposium on Experimental Robotics (ISER)*. 2016.

[31] Kentaro Wada. *Image Polygonal Annotation with Python (polygon, rectangle, circle, line, point and image-level flag annotation)*. URL: https://github.com/wkentaro/labelme (visited on 04/03/2019).

[32] Tingwu Wang. *Semantic Segmentation*. URL: http://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf (visited on 04/22/2019).

[33] Walter H. Pitts Warren S. McCulloch. "A Logical Calculus of the Ideas Immanent in Nervous Activity." In: *Bulletin of Mathematical Biophysics, Vol. 5, 115-133* (1943).

[34] *What is an epoch in deep learning?* URL: https://www.quora.com/What-is-an-epoch-in-deep-learning (visited on 05/11/2019).

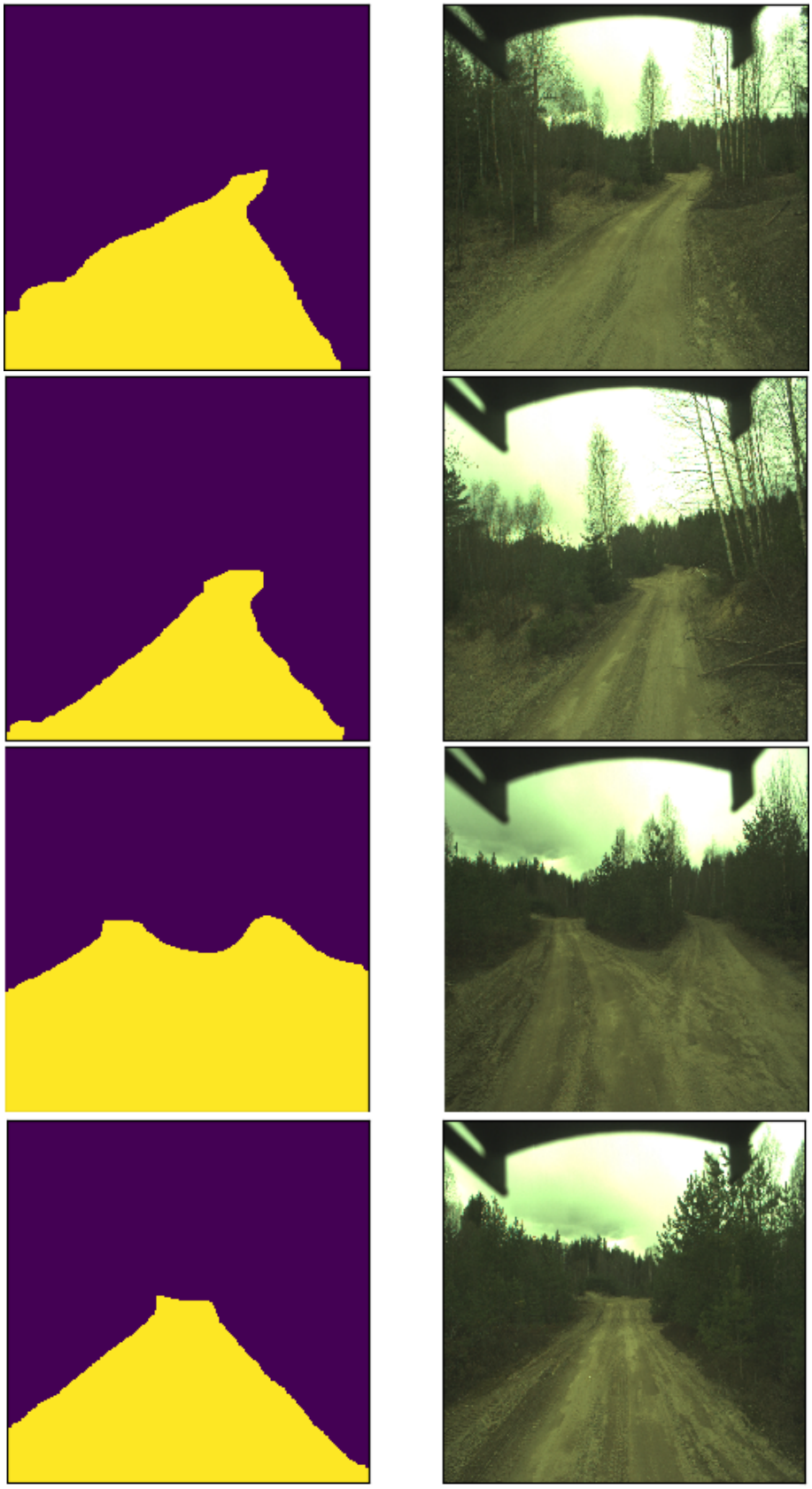
Appendices

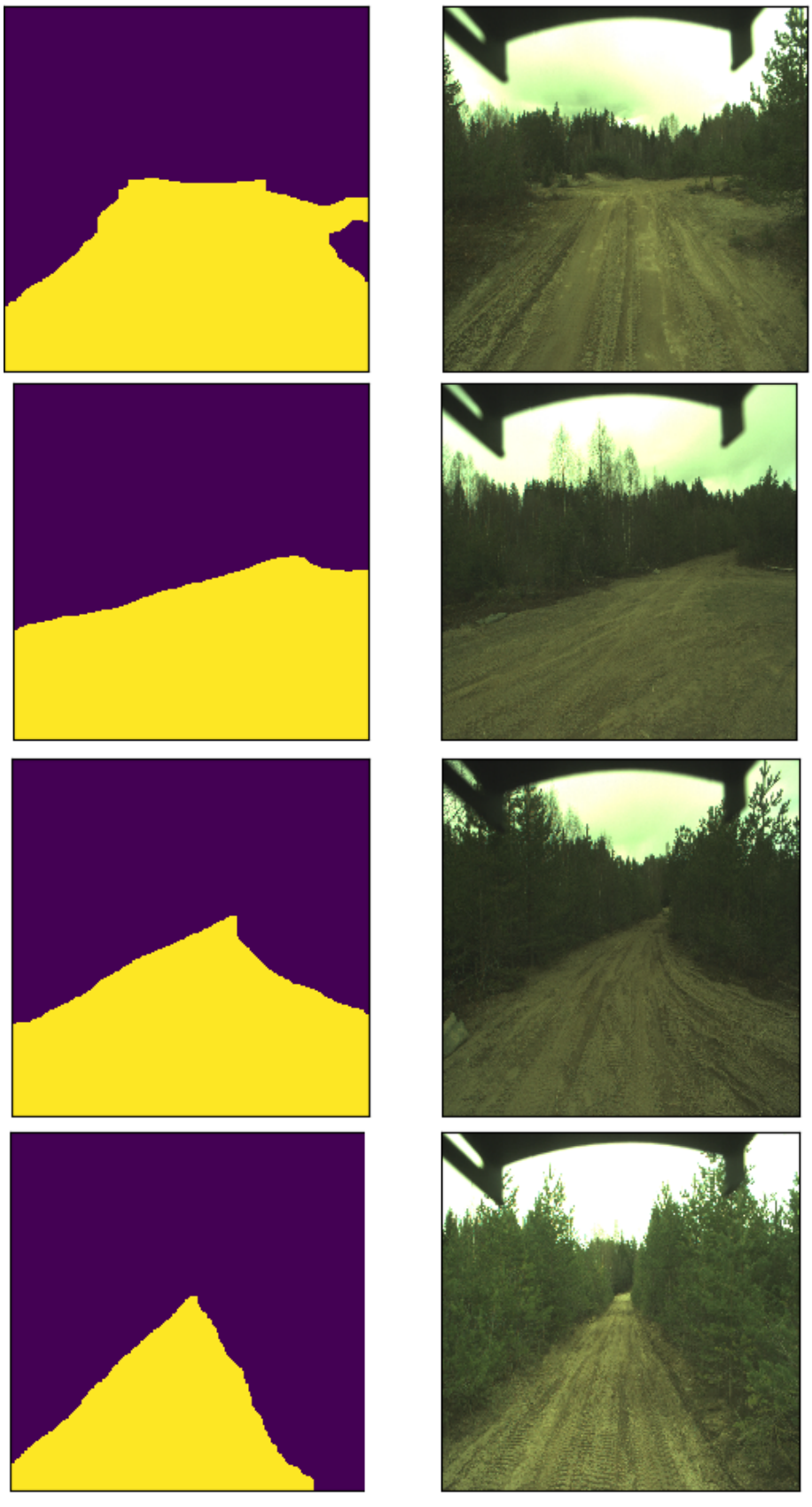
Appendix A

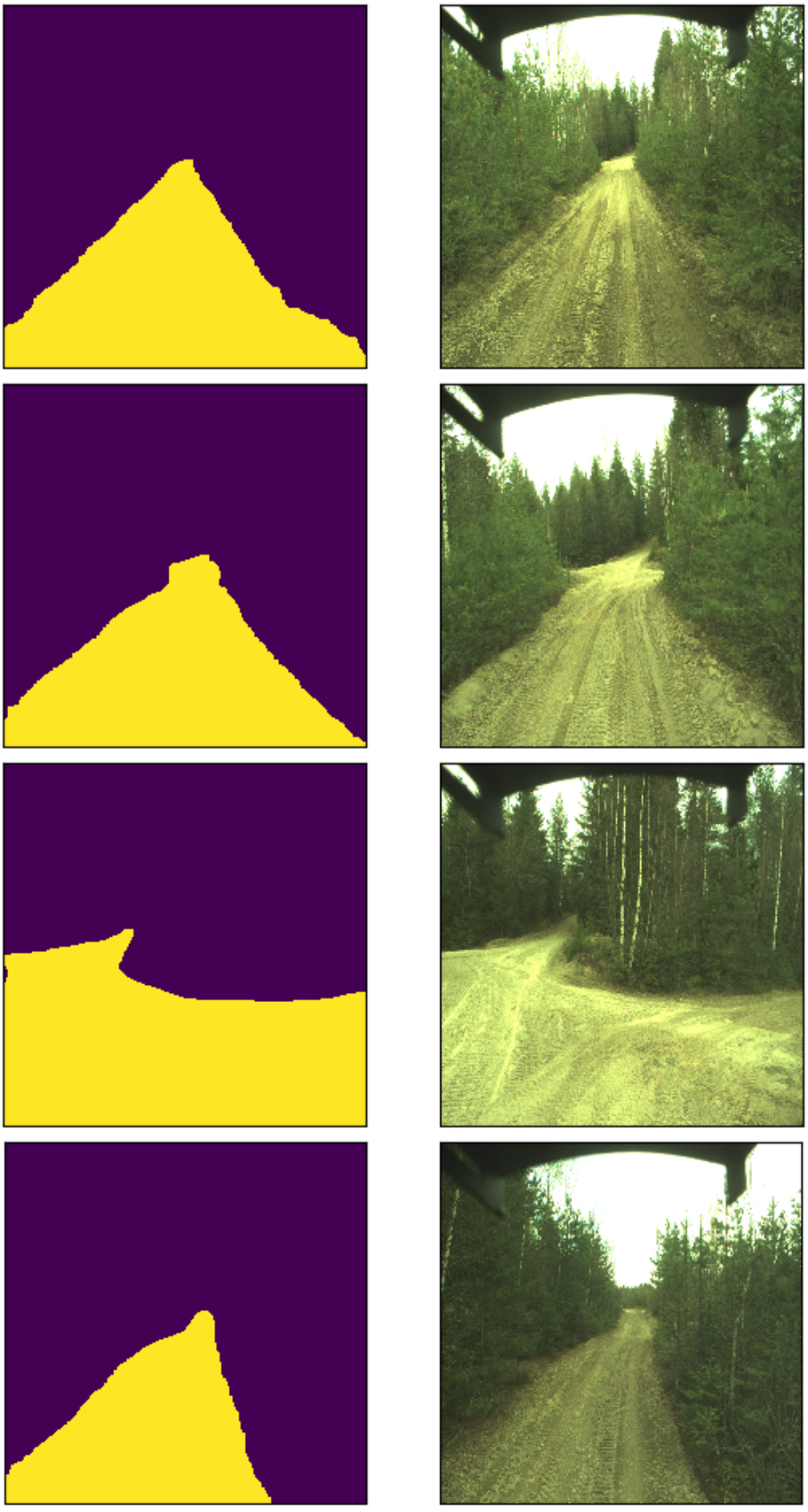
Image sequences from the predictions in each dataset

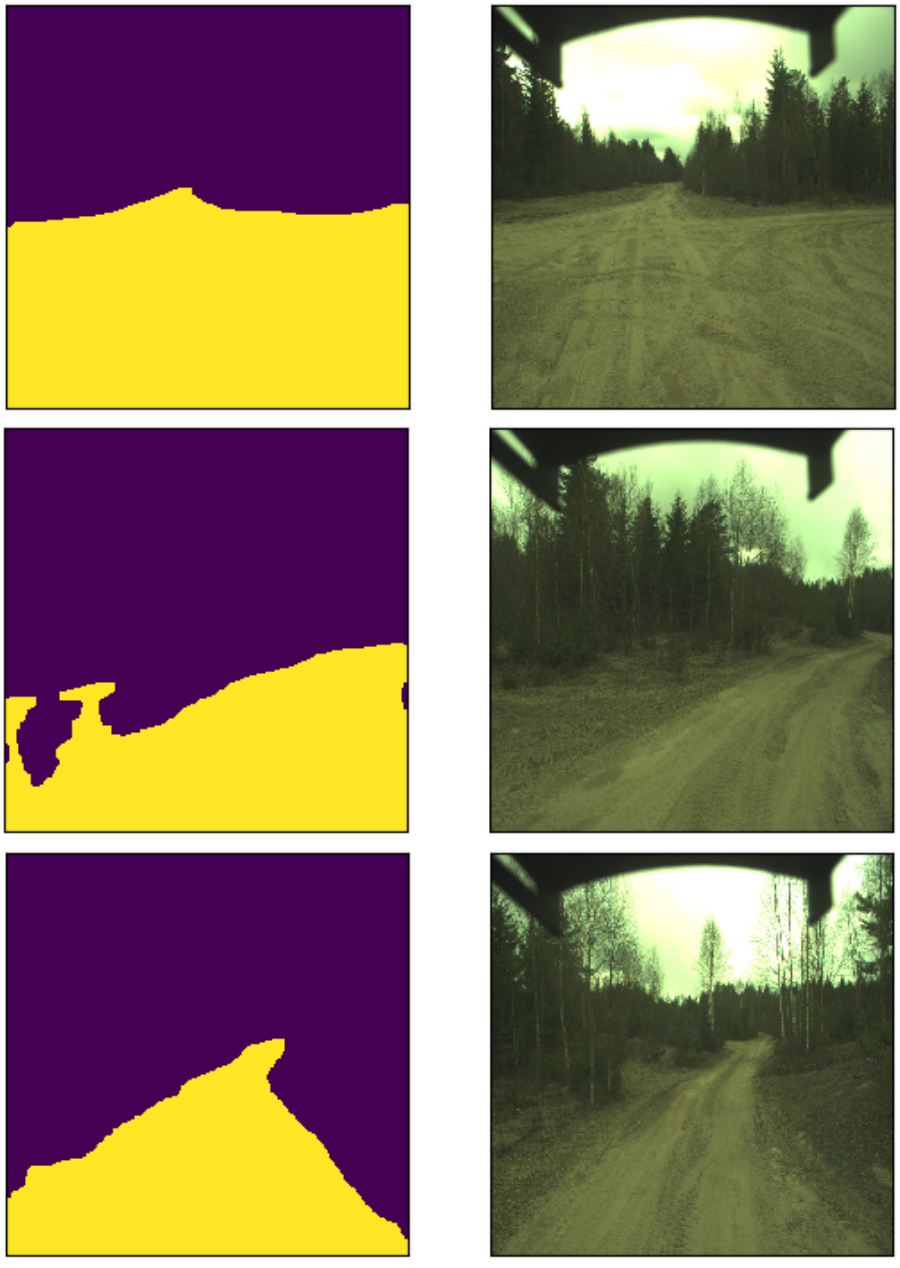
A.1 Predictions from the *Custom 1* dataset

This section presents 15 original images with their corresponding auto-annotated image to the left, uniformly sampled across the *Custom 1* dataset.









A.2 Predictions from the *Custom 2* dataset

This section presents 15 original images with their corresponding auto-annotated image to the left, uniformly sampled across the *Custom 2* dataset.