Predicting annotation difficulty using Monte Carlo dropout

Master's thesis in Complex Adaptive Systems

JENS WILHELMSSON

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Predicting annotation difficulty using Monte Carlo dropout

JENS WILHELMSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2019

Predicting annotation difficulty using Monte Carlo dropout

JENS WILHELMSSON

Supervisor: Daniel Langkilde, Annotell
Examiner: Giovanni Volpe, Department of Physics, University of Gothenburg

Master's Thesis 2019
Department of Physics

Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Heat map of 10 annotators idea of which pixels belong to the class road.

Typeset in LaTeX
Gothenburg, Sweden 2019

Predicting annotation difficulty using Monte Carlo dropout
JENS WILHELMSSON
Department of Physics
Chalmers University of Technology

Abstract

Developing products based on machine learning algorithms require relevant and accurate datasets. In particular when it comes to supervised learning algorithms whose performance is directly related to the quality and amount of training data. Within the field of computer vision, classification is a task that require training data in the form of annotated images. Annotating images is a manual task and I propose that the annotation difficulty of an image should be interpreted as the likelihood of someone else annotating an image differently. Knowing in advance which images are hard to annotate would facilitate the distribution of work between annotators with varying experience. In this thesis, it is shown that the uncertainty derived from Monte Carlo dropout resembles the variance of a group of persons annotations of the same image. This finding indicates that the level of agreement between persons can be predicted, and thus enable for better distribution of work between annotators. Furthermore, the finding could also be used to order images during training by prioritizing harder images higher.

Keywords: Annotations, Deep learning, Convolutional neural networks, Annotation difficulty, Monte Carlo dropout.

Acknowledgements

I want to express my appreciation to my supervisor Daniel at Annotell for providing the idea for the project and for guiding me along the way. I would also like to thank the rest of the team at Annotell for your much appreciated assistance and discussions.
I also want to thank the ever so patient Jakob for his invaluable advice and constructive feedback throughout the project.
Finally, thank you to my proofreader Malin.

Jens Wilhelmsson, Gothenburg, May 2019

Contents

List of Figures

1

Introduction

The number of fields in which machine learning algorithms have been implemented has increased in recent years. Along with the decreasing cost of computing power, lots of products based on supervised learning algorithms have become feasible. A prerequisite for supervised learning algorithms is however the access to quality training data, which has become the bottleneck of creating reliable products.

Depending on the purpose of the product, different kinds of training data are needed. In this project, the purpose of the products will be image recognition which means that the training data will consist of annotated images. Annotated images are in practice produced by marking objects in images with corresponding labels that specify what kind of objects they are. This process is manual and hundreds of thousands annotated images are often needed, explaining why it can be the bottleneck in a product development process. The fact that it is a matter of large volumes of manual work indicates that there is room for a lot of optimizing in the annotation process. If the difficulty of annotating an image is known before it is manually assessed would mean that easy images could be given to less experienced annotators, and hard images to more experienced annotators. Annotation difficulty is what will be focused on in this thesis, especially how the difficulty of annotating an image can be predicted without manual assessment.

1.1 Background

The lack of consistent training data was the spark that led to the founding of the company Annotell, which has made providing reliable and consistent training data its business. The company has created an annotation platform which their employees use to annotate images. Annotell is developing policies to assign annotation tasks to the most appropriate person. For example hard images should be given to experienced annotators. If the difficulty of annotating an image would be known before an image has been manually assessed, the images can be better distributed with respect to difficulty of the image and experience of the annotator.

1.2 Aim

The project aims to investigate the possibility of predicting the annotation difficulty before an image has been assessed manually.

1.3 Limitations

Due to the time limit, the project will be subject to some limitations as follows:

- The model shall be thought of as a prototype and the aim is not to implement it into Annotell's annotation platform.
- The model will only predict the annotation difficulty of single class semantic segmentation.

1.4 Problem formulation

The project aims to answer the following questions

- What does it mean that an image is difficult to annotate?
- How do you predict the difficulty of annotating an image?

2

Theory

This chapter will cover basic neural networks, why annotations are needed and more advanced neural networks that can be used to predict the annotation difficulty.

2.1 The annotation process

To ensure annotations are of high quality, annotations are done in three steps; annotation, review and correction. An annotation can be either accepted or rejected in the review step - if it's accepted it is finished, and if it's rejected, it goes to correction.

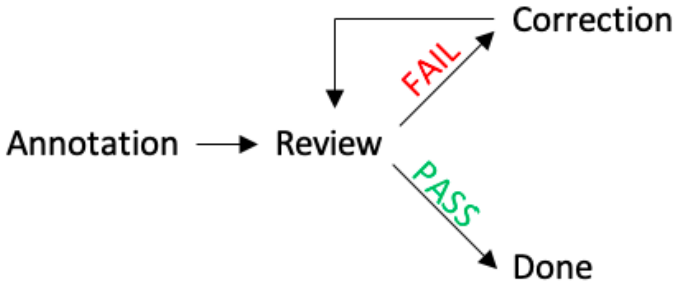

Figure 2.1: The iterative annotation process. If the reviewer does not approve the annotation, it is sent in a correction-review loop until it is passed by the reviewer.

2.1.1 Annotation difficulty

Annotations are performed by humans and what humans perceive as difficult varies from person to person. In other words, the difficulty of a task is a subjective measure.

I define difficulty as proportional to the probability that a group of people provide the same annotation for a specific image. The harder an image is to annotate, the less likely a group of people are to provide the same annotation.

In other words, a task is assumed to be easy if a large amount of people give the same answer to it, and hard if the variance among the answers is large.

To put this in context of image annotations, a hard image to annotate is recognized as one with large variances in annotations. For example, if 10 annotators are asked to annotate an image by classifying which pixels contains the class road and which pixels that doesn't. If a single pixel is classified as road by 5 of the annotators and as not road by the remaining 5 annotators, then it exists an large ambiguity about that pixel and it would be interpreted as hard in this thesis.

This is quantified by calculating the variance V at position (x, y) of the annotation A by annotator n as

$$V(x,y) = \frac{1}{N-1}\sum_{n=1}^{N} |A(n,x,y) - \mu(x,y)|^2 \tag{2.1}$$

where $\mu(x, y)$ is the average annotation of pixel (x, y).

The annotation difficulty of an image is then measured as

$$\sum_{x=1}^{W}\sum_{y=1}^{H} V(x,y) \tag{2.2}$$

where H and W is the width and height of the image. In other words, summing the variances for all pixels.

Note that all pixels are weighted equally, meaning that a corner pixel can add equally to the difficulty as a pixel in the middle of the image.

2.2 Artificial neural networks

The current state of the art-methods for semantic segmentation is based on artificial neural networks, inspired by the way the brain operates.

The goal of a simple artificial neural network with multiple inputs and a single output is to approximate some function f so that it maps the input vector $\mathbf{x}$ to the desired output category y. Such a neural network can be seen in figure 2.2.

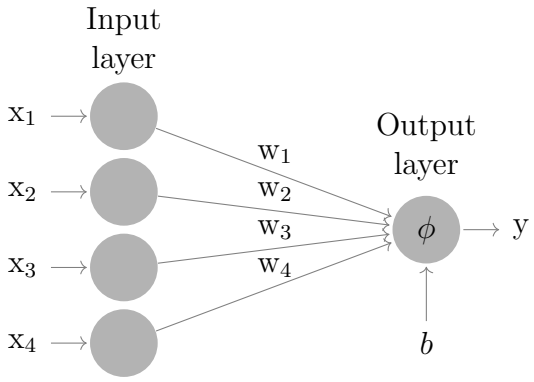

Figure 2.2: A simple neural network with input $\mathbf{x}$, output y, weights $\mathbf{w}$, threshold b and activation function ϕ.

The output in the example is calculated as

$$y = \phi\left(\sum_{i=1}^{4} w_i x_i + b\right) \tag{2.3}$$

where ϕ is the activation function, w_i is the weights, x_i is the inputs, b is a threshold and y is the output.

To make the neural network map the input to the desired output, the weights are updated until the output is close enough to the desired output. A *cost function* is defined to measure how well the output matches the desired output.

2.2.1 Cost function

The purpose of the cost function is to quantify the difference between the output and the desired output. A common cost function for image segmentation is the cross entropy function

$$H(\mathbf{y}, \mathbf{y}') = -\sum_i y_i \log(y_i') \tag{2.4}$$

where $\mathbf{y}$ is the desired output and $\mathbf{y}'$ is the actual output. Large deviations between $\mathbf{y}$ and $\mathbf{y}'$ generate large values, which means that a poor output is punished with a high cost.

The ambition is to make the cost function as small as possible, so to make the output match the desired output, the cost function is minimized.

2.2.2 Optimization algorithms

The weights and threshold in figure 2.2 are the parameters to adjust so that the cost function reaches its minima. Attempting to solve this minimization problem

analytically quickly becomes unfeasible since it is a non-linear problem due to the presence of non-linear activation functions. Instead, iterative methods are used.

For artificial neural networks, gradient descent is the most fundamental algorithm. It works as follows

1. Calculate the gradient of the cost function, with respect to the current significant parameter, for example the weights

$$\mathbf{G} = \frac{\partial \mathbf{H}}{\partial w} \tag{2.5}$$

2. Update the weights according to

$$w \leftarrow w - \eta \sum_{j}^{N} G_j \tag{2.6}$$

where η is the *learning rate*, which needs to be set to a good level, and N is the number of training examples. If the learning rate is too small, it will take the network too long time to converge and if it is too big, the network might miss minima since it "jumps" in the direction of the gradient.

However, equation (2.6) becomes very computationally heavy when the number of training examples are large. So instead, *stochastic* gradient descent is generally preferred. The stochasticity refers to the fact that all training examples not are included when calculating the weight update, as in equation (2.6). Instead, a *batch* of training examples are randomly selected from which the weight updates are calculated [1].

This introduces two parameters required to be set to some value, the learning rate η and the batch size. These two are included in the family of *hyperparameters* that are required to be specified prior to training a neural network. The hyperparameters are usually required to be adjusted gradually in the beginning of a training procedure, to find a combination that works well given the current task and computing power limitations.

Stochastic gradient descent can be further extended with additional hyperparameters such as *momentum* and *weight decay*. Momentum refers to maintaining a small part of a weight update to the next iteration, so that the update direction is calculated using not only the current gradient, but also using a part of the previous gradient. The momentum parameters refers to how much the previous gradient should impact the weight update.

Weight decay is a method that limits the freedom of the model's learning by penalizing large weights. By adding weight decay, equation (2.6) changes to

$$w \leftarrow w - \eta \sum_{i}^{N} G_i - \lambda \eta w \tag{2.7}$$

where λ is the weight decay parameter.

A further extension of stochastic gradient descent is the optimizer *Adam*. With Adam, the learning rate is individually estimated based on the first and second moments of the gradients [2]. Adam has proven to be superior of stochastic gradient descent, especially when it comes to time to convergence [3].

2.2.3 Activation function

As mentioned in equation (2.3), an activation function can be used in the output layer of a neural network. The reason for having an activation function there is to map the neural network output to possible answers of a problem. For example if the network is to define what pixels of an image are road and which pixels are not road. Then a suiting activation function would be one that maps the network output to either 1 (road), 0 (not road) or something in between. The *sigmoid function* is an example of such a function,

$$S(x) = \frac{1}{1 + e^{-x}} \in (0, 1) \tag{2.8}$$

where x is the network output and the resulting S ranges from 0 to 1.

Another common and useful activation function is the *rectified linear unit* (ReLU) which is defined as

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

where x is the input. As can be understood from the equation, it simply removes all negative values by setting them to zero. The reason behind using the ReLU activation function is that its gradient is always 0 or 1, in contrary to the sigmoid activation function where the gradient goes to zero for large values. In the context of neural networks, a gradient that goes to zero means that the learning process slows down, or even stops, which is preferably avoided.

2.3 Convolutional neural networks

Images can be broken down into parts. They are made up of motifs which in turn are made up of scenes, which are made up of objects, which are made up of parts, which are made up of edges and so on until the smallest level are reached - pixels. A good way of interpreting images would be to have different stages of recognition, relating to the stages that an image is broken down into. This is what is achieved in *convolutional neural networks* and is the reason why they are commonly used in image classification tasks. The thing that enables this has given the convolutional neural networks its name - the convolution operator. The process of interpreting different stages of an image is referred to as feature extraction, where features can be everything from a sharp edge, sized no bigger than a few pixels, to a car. Convolution is also suiting for feature extraction thanks to it being a translational invariant

operation. This means that it can recognize a feature independent of the feature's position in an image [4, 5].

Convolution

Convolution works by sliding a filter across an image and multiplicating the filter with the values currently being covered by the filter and then adding the resulting terms together. The operation is displayed graphically in figure 2.3. To control the size of the output, zero values are added around the input. This technique is called *padding*. A larger filter calls for more padding, if the size of the image is meant to be preserved.

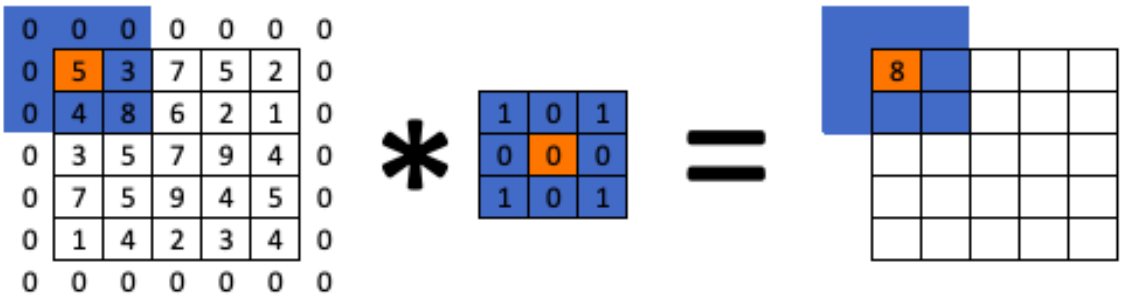

Figure 2.3: Convolution example. The operation displayed in the figure is performed across the whole input matrix, so that all cells in the right matrix is filled. In this example, padding is also added to the input, so that the output preserves the input size.

Typically in a convolutional neural network, many filters such as the one in figure 2.3 are used in the convolutional layer, which means that the number of output channels from a convolutional layer is bigger than the number of input channels.

The values of the filters (a 3×3 filter is shown in figure 2.3) are the parameters that are updated when training a convolutional neural network.

Pooling

Pooling is a form of non-linear down-sampling that can be done using different functions. The most common function used to perform pooling is max pooling, demonstrated graphically in figure 2.4. In the case of max pooling, a layer is split up into non-overlapping sub-regions, where the maximum value is extracted from each sub-region. While convolution increases the number of channels, the number of channels stays the same after pooling but the size of each channel is reduced [6].

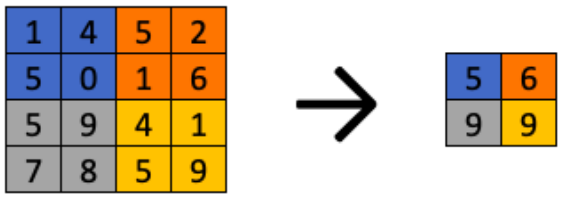

Figure 2.4: How max pooling is performed. A layer is split up into smaller sub-regions that do not overlap and the maximum value of each region is extracted.

Transposed convolution

As seen in figure 2.4, pooling reduces the size of each image. This means that some kind of upsampling is needed to bring the images back to their original size. Instead of using regular upsampling methods such as bilinear interpolation, transposed convolution is used since it comes with the advantage of containing trainable parameters. It works similarly to a regular convolution but as can be seen in figure 2.5, it adds extra padding to the input and also calculates output with the filter centered outside the input matrix.

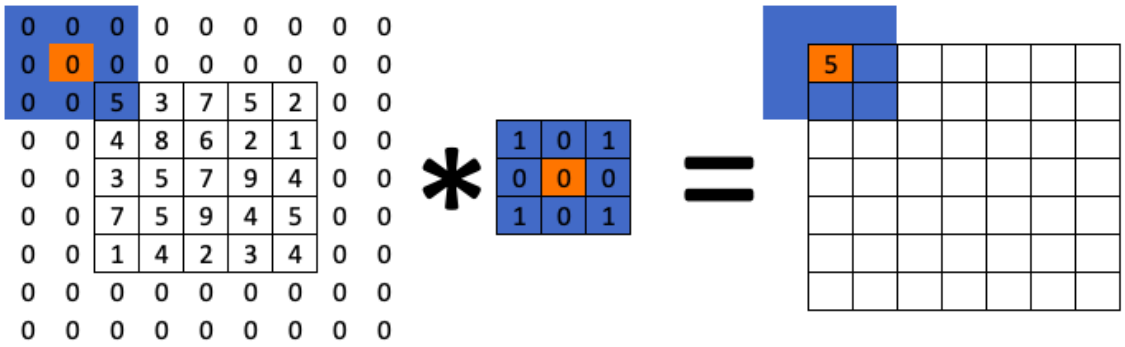

Figure 2.5: Example of a transposed convolution for upsampling. As with convolution, the filter is moved across the whole input matrix and the whole right matrix is filled with numbers. In this example, the transposed convolution changes the dimension of the input from 5×5 to 7×7.

Dropout

A common problem when training neural networks is *overfitting*, which means that the network learns the training data too well and thus will perform badly on validation data. To avoid this, dropout can be applied, meaning that the output from neurons in specific layers will be discarded with a *dropout probability* p.

2.3.1 Architecture

A convolutional neural network can often be split up into two parts, an *encoder* that performs downsampling and a *decoder* that performs upsampling. The idea with the encoding is to extract different inherent features of the input. By performing convolution with a larger number of output channels than input channels, the encoder produces multiple feature maps that represent different characteristics of the image. Each layer of the encoder usually consists of concatenations of convolutions and max pooling. The convolutions produce an arbitrary amount of channels, each channel is produced by a single convolving filter. More output channels in a layer in other words come with more parameters to be trained, which relates to complexity and training time of the network. Max pooling reduces the dimensionality, extracting only the sharpest details of sub-regions, which enhances the feature extraction performed by the convolution [7].

The decoder part of a network is the natural opposite to the encoder. By transposed convolution or other upsampling methods, all features extracted by the encoder are taken into consideration and an output is produced. The output is then through some activation function mapped to an, in the case of an input image, pixel-wise prediction.

Different types of architectures perform well on different problems and for the task of semantic segmentation, fully convolutional neural networks have proven to be reliable [8]. It is not the current best performing according to the Pascal Visual Objects Challenge leaderboard for semantic segmentation [9], but since the performance of the neural network not is crucial in this thesis, it is a convenient thanks to its relatively simple implementation.

Fully convolutional network

The Fully Convolutional Network (FCN) was proposed in a paper in 2015 [8] and the idea is to make maximal use of the powerful convolution operator, both in the downsampling and upsampling part of the network. Three different variants of a fully convolutional network is proposed by Long, Shelhamer and Darrel; FCN-32, FCN-16 and FCN-8.

FCN-8 is the one that performs best, according to the authors. The architecture FCN-32 is presented in Figure 2.6, which also is the base for FCN-16 and FCN-8. In FCN-16, the architecture is identical to FCN-32, but with the predictions (upsampled via transposed convolution) from layer 4 is added to the final output. In FCN-8, both the predictions from layer 3 and 4 are added to the final output. By adding the predictions from earlier layers, the network can produce output of finer detail [8].

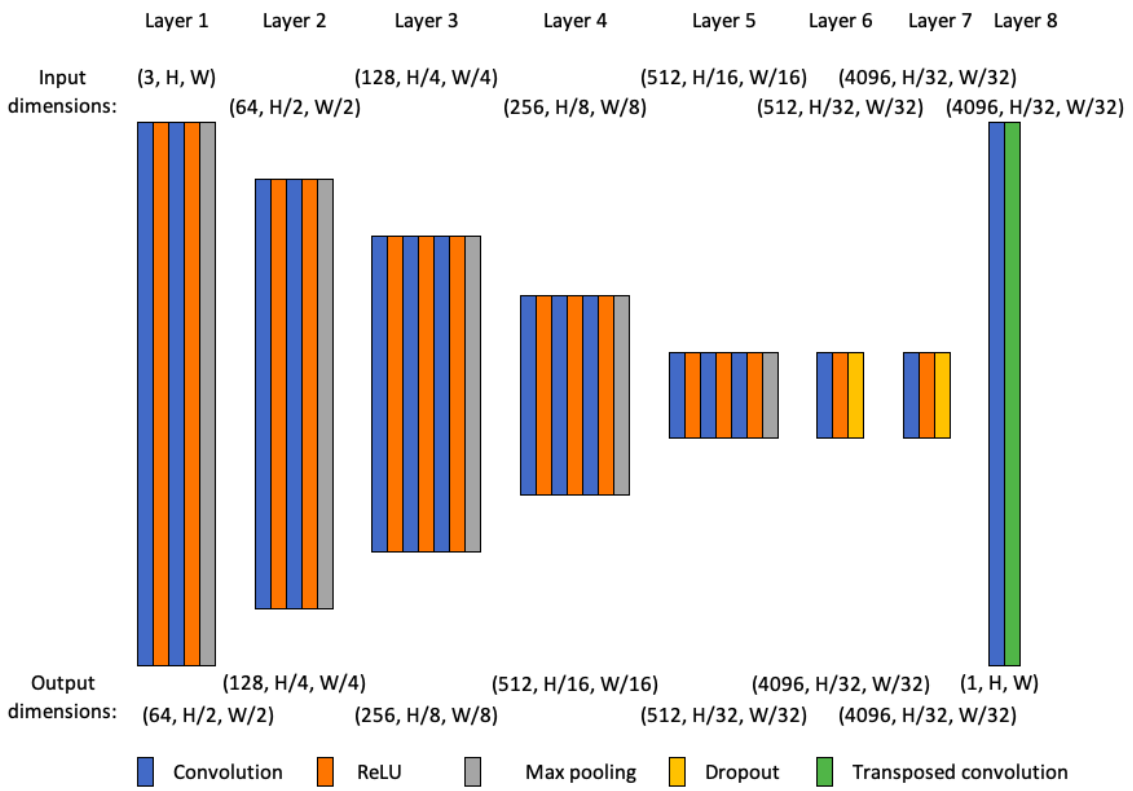

Figure 2.6: FCN-32 architecture. The architecture FCN-8 is the same as FCN-32 with the only difference being that the predictions (upsampled via transposed convolution) from layer 3 and 4 are added to the final output.

2.4 Model uncertainty

The output of a semantic segmentation neural network with a sigmoid activation function is a matrix of the same size as the input image with values between 0 and 1. It is easy to interpret this as a probability, that the higher the value, the more probable it is that it is actually road and not background. With a perfectly calibrated network, this could be the case. However, such a network is not plausible in practice.

Think of a prediction of 0.5. Instinctively, it should mean that the network is absolutely clueless whether the pixel is road or background, which is a correct observation. But what if the same image was processed again by the same network, and the output of that same specific pixel is now 0.4? This would also point to some other kind of uncertainty - how then should the total uncertainty of the model be quantified? It should be noted that neural network predictions in general are deterministic and can not produce a different output if fed with the same image multiple times. This is achieved through Monte Carlo dropout, described in section 2.4.3.

These two types of uncertainty are categorized as *aleatoric* and *epistemic* uncertainty.

2.4.1 Aleatoric uncertainty

Aleatoric uncertainty is the type of uncertainty that is impossible to foresee and therefore is unavoidable [10]. For example a coin toss, the probability of getting heads is 0.5. No matter how many times the coin is tossed, the probability will still be 0.5 of getting heads. This is an example of an purely aleatoric uncertainty [11].

The neural network predictions being between 0 and 1 is the aleatoric uncertainty in this context - a prediction close to 0.5 has high aleatoric uncertainty.

The predictions are 1 (road) with a probability $p \in (0, 1)$, making it a Bernoulli variable for which the variance is

$$\mathrm{Var}(p) = p(1 - p). \tag{2.10}$$

The Bernoulli variance is used to quantify the aleatoric uncertainty.

2.4.2 Epistemic uncertainty

Generally, the epistemic uncertainty is the type of uncertainty that can be predicted and reduced by refining a model or expanding a dataset.

For example a bag of poker chips. Assume that it is known that there are black and red poker chips in the bag, but the proportions of red and black are not known. When pulling out the first chip, the probability is 0.5 of getting a red chip since the distribution between red and black chips are unknown. Assume that the first chip turns out to be red, and after drawing another 9 chips, it turns out that 7 out of 10 are red. This indicates that there are more red chips than black chips in the bag. It also means that the epistemic uncertainty has decreased with the increasing knowledge of the contents of the bag. In clear, with more knowledge comes a smaller epistemic uncertainty. When all poker chips are drawn, the share of red and black poker chips are known and the epistemic uncertainty is zero.

In the context of this thesis, the epistemic uncertainty refers to the fact that a network can give different predictions of the same input image only due to the presence of active dropout during inference [10, 11].

2.4.3 Monte Carlo dropout

Normally, neural networks are deterministic. This means that they will return the same answer if asked several times. Unfortunately, this behaviour does not give any information concerning uncertainty.

Approximating the uncertainty of deep learning models is a relatively new field where Monte Carlo dropout has gained a lot of attention [12]. It uses dropout, which is often already implemented in neural networks to mitigate overfitting, to approximate the epistemic model uncertainty. By having dropout activated during inference and performing inference multiple times, random predictions are generated which can be interpreted as samples from a probabilistic distribution [12]. The network is then not deterministic anymore, it is instead probabilistic. Meaning that it will produce different results each time the same image is fed to it.

Gal and Ghahramani name this approach Monte Carlo dropout and show its validity in [12] when having dropout applied at each weight layer. In [13], Gal shows through examples that it is possible to have dropout applied at only a subset of weight layers and still get a measure of uncertainty. This means that Monte Carlo dropout can be used to estimate uncertainty of already existing network architectures that have dropout implemented. One example of such a network architecture is the fully convolutional network FCN-8 which is the architecture used in this thesis. The architecture FCN-8 is explained in detail in section 2.3.1.

3

Method and Data

In this chapter, the methods of the two parts of the project is presented first, followed by an explanation of the data used.

The project was written in Python and the neural networks were implemented using the package PyTorch [14]. An implementation of FCN-8 in PyTorch by Kentaro Wada[1] was used as a starting point for the project.

The training of the neural networks was executed using a Google-powered virtual machine provided by Annotell. The GPU used was a Nvidia Tesla P100 along with 32 GB RAM. With this setup, along with Nvidias parallel computing platform Cuda, a training session as described below, with 50000 iterations, took around 24 hours.

3.1 Multiple network model

The idea behind using multiple networks was to simulate the behaviour of an annotator. This was achieved by training multiple networks on different data. The networks were all of the same architecture and a network was only trained on annotation data created by a single person to make the networks represent one annotator each. This way, each network would capture underlying behaviour of the different annotators.

A total number of 10 models were each trained for 50000 iterations. The number 50000 were used based on training a network with the same configuration for 100000 iterations and evaluating where the network reached a acceptable loss. Each model was only trained on data annotated by a single annotator, so that each model would represent an annotator.

By feeding the same image to each of the trained networks, a number of output values for each pixel were created, as visualized in figure 3.1.

[1] https://github.com/wkentaro/pytorch-fcn

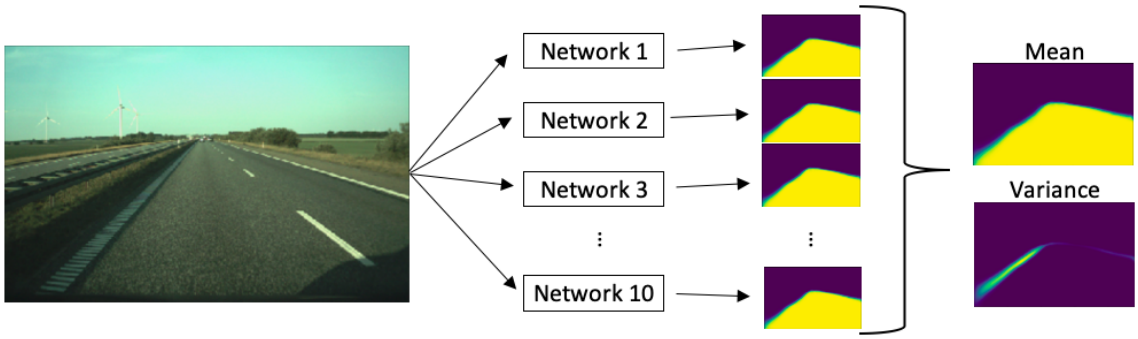

Figure 3.1: Overview of how the mean and variance were calculated using the multiple networks trained on different data.

At the output layer of each network, a sigmoid activation function were applied, which put the output pixel values in the range from 0 to 1. This method resulted in 10 different values between 0 and 1 for every pixel in each image. As seen in figure 3.1, the mean and variance of these values were calculated. The mean and variance are used in section 3.3 to quantify the aleatoric and the epistemic uncertainty respectively.

3.2 Monte Carlo dropout model

For dataset A, the network aimed to perform Monte Carlo dropout was trained using the same network architecture and parameter configuration as in the previous section, but with annotations from all annotators. All parameters were set to the same values. The only difference in the training was that this time, it was trained for 100000 iterations and trained on more data. Since it did not matter who annotated the data, the model could be trained using all available data. But due to computational limitations, the number of training images to be loaded simultaneously was limited to around 5000-6000 (out of 15000 totally). Roughly 5500 images were therefore randomly selected as training data from the dataset.

The same network architecture and configuration were used for training with the data from dataset B. However, only around 800 images were available for training, which limited the training since overfitting started occuring after around 70000 iterations (indicated by increasing validation error). Therefore, training was stopped after 60000 iterations.

Monte Carlo dropout was performed with 1000 iterations. Each time, the output pixel values varied due to the presence of dropout in the network which gave rise to a distribution for each pixel. These distributions are used to provide information about the epistemic uncertainty of the model. If a distribution is wide (large variance) the epistemic uncertainty is large for that specific pixel. See figure 3.2 for an overview of how the mean and variance were created using Monte Carlo dropout.

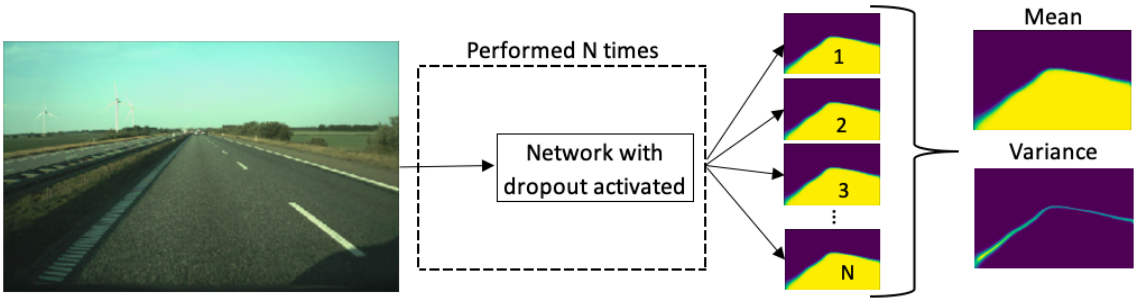

Figure 3.2: Overview of how the mean and variance are calculated with the model using Monte Carlo dropout.

3.3 Annotation difficulty

In this section, it is described how the predictions in section 3.1 and 3.2 were used to calculate a level of annotation difficulty for an image.

In addition to the epistemic uncertainty which is measured through the variance of the predictions, the mean of the predictions were used to quantify the aleatoric uncertainty of each image. Since each pixel output is 1 (road) with probability $p_1 \in (0, 1)$, the prediction y' is a Bernoulli variable with parameter p_1. The variance is consequently

$$\mathrm{Var}(p_1) = p_1(1 - p_1). \tag{3.1}$$

Call the prediction for pixel (x, y) by network/iteration n (network for multiple network model and iteration for Monte Carlo dropout model) $P(n, x, y)$.

The epistemic uncertainty U_e for each pixel was then estimated by calculating the pixel-wise variance,

$$U_e(x, y) = \frac{1}{N - 1} \sum_{n=1}^{N} |P(n, x, y) - \mu(x, y)|^2 \tag{3.2}$$

where $\mu(x, y)$ is the mean prediction for a pixel and N is the number of networks for the multiple network model and the times an image was fed to the network for the Monte Carlo dropout model.

With the Bernoulli variance $\mathrm{Var}(x)$, the aleatoric uncertainty U_a was calculated as

$$U_a(x, y) = \mathrm{Var}(\mu(x, y)). \tag{3.3}$$

Since the difficulty of an whole image is of interest, a summation over all pixels was done for both the epistemic,

$$S_e = \sum_{x=1}^{W} \sum_{y=1}^{H} U_e(x, y) \tag{3.4}$$

and the aleatoric uncertainty:

$$S_a = \sum_{x=1}^{W} \sum_{y=1}^{H} U_a(x, y). \tag{3.5}$$

A complete measure of difficulty for an image could then be a created by adding together S_e and S_a. Unfortunately, this calls for some scaling. The scaling factors C_e and C_e was derived by calculating the mean S_e and mean S_a for 100 images.

Consequently, the difficulty of an image could be computed by combining the aleatoric and epistemic uncertainty:

$$D = \frac{S_e}{C_e} + \frac{S_a}{C_a}. \tag{3.6}$$

The classic way of evaluating a prediction is by comparing it to some kind of ground truth. The ground truth in this case would be images annotated by multiple annotators, which would give rise to a real variance of annotations that could be compared to the predicted variances and difficulties.

For dataset A, such data was unfortunately not available. For dataset B however, 8 annotators were asked to annotate the same 100 images to create a ground truth to compare with.

Using the Monte Carlo dropout model, annotation difficulties were predicted for these 100 images. The method for calculating prediction difficulties described in section 3.3 were applied also to the ground truth variances to produce a ground truth annotation difficulty for each image. To evaluate the performance of the model, the correlation between the predicted annotation difficulties and the ground truth annotation difficulties were measured.

3.4 Data description

Two different datasets were used in the project. The first dataset will be referenced as dataset A and the second dataset B. The advantage of dataset A was that it contained a lot of data, 15000 annotated images, all taken while driving on a highway. The downside was however that it lacked a relevant and accurate ground truth to evaluate the predicted difficulties.

Dataset B on the other hand, contained 100 images annotated by 8 annotators each which means that the predicted difficulties could be evaluated in a quantitative fashion. Unfortunately, dataset B only contained 800 images in total. Another downside is that the images, unlike the images in dataset A, were taken in a lot of different settings and not only on a highway, which lowered the performance of the trained network.

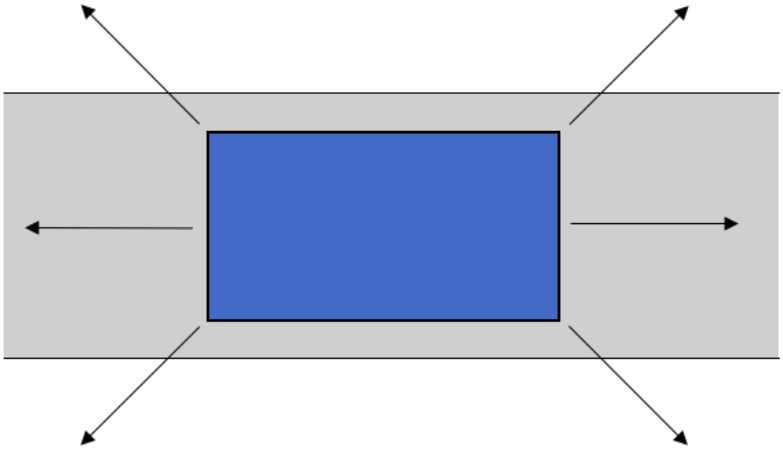

Figure 3.3: Birds view of a car in a car lane. The arrows represent the six camera views from where the images in dataset A were taken.

Figure 3.4: Example of an image from dataset A along with the corresponding bitmap. White pixels in the bitmap represent parts of the road that belong to the own vehicle lane. This image is taken from the rear of the vehicle, opposite of the driving direction.

3.4.1 Dataset A

Dataset A consisted of images taken with cameras attached to a car. The images were taken from six different camera views, shown in figure 3.3. All images were taken while the vehicle, to which the camera was attached, was driving on a highway.

Each image had an corresponding annotation bitmap of the same size as the image (1920×1208) where own road lane pixels were set to 1 (white) and background pixels to 0 (black). An example of an image and a bitmap can be seen in Figure 3.4. The whole dataset contained approximately 15000 images with annotations. The images were annotated by 10 different annotators (each annotator annotated roughly 1500 images).

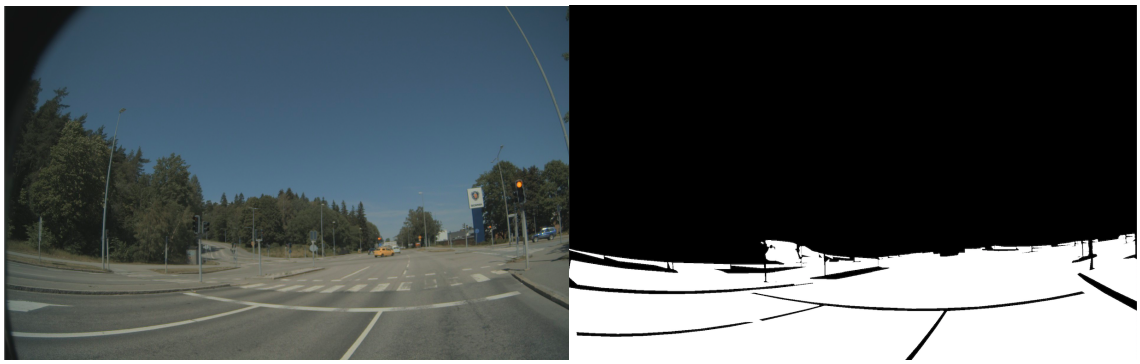

Figure 3.5: Example of an image from dataset B along with the corresponding bitmap. White pixels in the bitmap represent all paved surfaces. This image is taken from the front of the vehicle. Road markings were not included in the class paved surfaces, explaining why they not are included in the bitmap.

3.4.2 Dataset B

Dataset B consisted of about 800 images taken from a truck. While the images in dataset A only were from highway driving, the images in dataset B were from varying scenes and conditions. The camera angles, in relation to the truck, were not specified in the data.

As in dataset A, each image had a corresponding annotation bitmap of the same size as the image (1920×1200). Originally, the dataset contained annotations for a lot of classes and not only road as in dataset A. To create a dataset as easy as possible to learn for the network, only annotations for the class paved surfaces were extracted. An artifact from this is that some road markings are not annotated, since they belonged to another class in the original dataset. This can be seen in figure 3.5.

3.5 Neural network setup

Semantic segmentation was chosen as the type of classification, meaning that the network architecture had to be one that could learn this task. To minimize unnecessary work, the architecture had to be one for which it existed a PyTorch implementation. In addition, the architecture should preferably have performed well in the past. Originating from these conditions, the network architecture FCN-8 was chosen from the paper *Fully Convolutional Networks for Semantic Segmentation* [8]. It was chosen because it is in the Pascal Visual Objects Challenge leaderboard for semantic segmentation [9], along with having an existing PyTorch implementation[2].

[2]https://github.com/wkentaro/pytorch-fcn

3.5.1 Configuration and training

The optimizer Adam was used, along with the cross entropy cost function (2.4) which, for only 2 classes, takes on the form

$$H(y, y') = -y\log(y') - (1 - y)\log(1 - y') \tag{3.7}$$

where

$$\begin{cases} y' \in (0, 1) \\ y \in \{0, 1\} \end{cases} \tag{3.8}$$

The annotation value for a pixel, y, is 1 for road and 0 for background. The variable y' is the output from the network, a value between 0 and 1. This equation is often referred to as the *binary cross entropy*.

Before starting the training, a majority of the parameters were initialized based on a pre-trained model called VGG-16 which has reached good results in classification and localization contests [15]. It was possible thanks to the fact that the architectures are the same through the whole encoding parts of the networks. The decoder of the networks differed slightly, the architectures did not have the same number of output classes. The parameters that were not initialized based on the pre-trained VGG-16 model were initialized randomly.

To speed up the training, the images were scaled down by a factor of 16, from 1920×1208 to 480×302 for dataset A and from 1920×1200 to 480×300 for dataset B. The parameters used for training can be seen in table A.1.

4

Results

The results are divided into two parts which correspond to the two different approaches described in the method. 100 randomly selected images from dataset A are used to show the performance of the two different models used.

For dataset B, the 100 images annotated by 8 different annotators are used to evaluate the Monte Carlo dropout model.

4.1 Multiple network model

Along with the results of the training, a selection of images from dataset A are shown in this section. They are meant to be representative of different regions of predicted annotation difficulties by the multiple network model.

The training and validation loss during the 10 different training sessions are shown in figure 4.1. This is to ensure that the performance of the different networks are aligned. If the validation error would have started to increase sometime during training, it would have indicated that the model is overfitted. The constantly decreasing validation loss confirms that this is not the case.

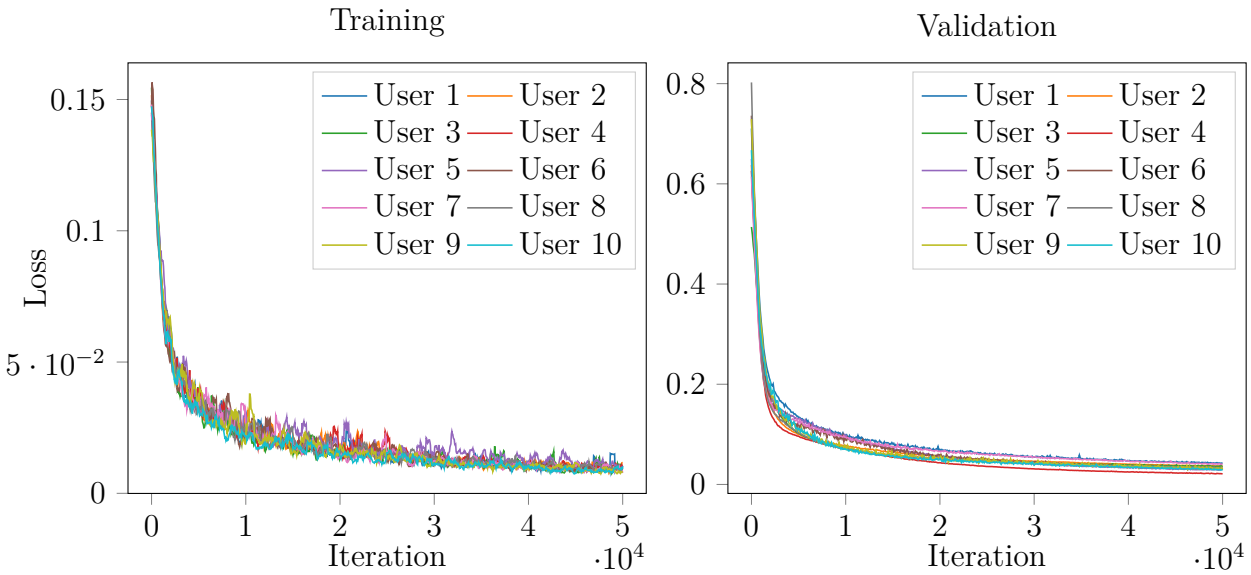

Figure 4.1: The loss during training for the training and the validation dataset when training networks to use in the multiple network model. The validation dataset consists of a very small number of images extracted from the training dataset. A user refers to an annotator.

4.1.1 Dataset A

Three images ranked 3, 51 and 94 out of 100 where 1 is the easiest possible predicted annotation difficulty are shown below. For each image, the prediction variance of the 10 networks is displayed in one image, and the prediction mean of the 10 networks in another. The variance and mean are normalized independently so that the maximum value is 1.

These three images are selected out of the 100 randomly selected images based on the fact that they are predicted as relatively easy (3/100), medium (51/100) and hard (94/100).

Under each pair of images is a histogram of predicted annotation difficulties of the 100 randomly selected images. The position of the current image in the histogram is denoted by a dotted line.

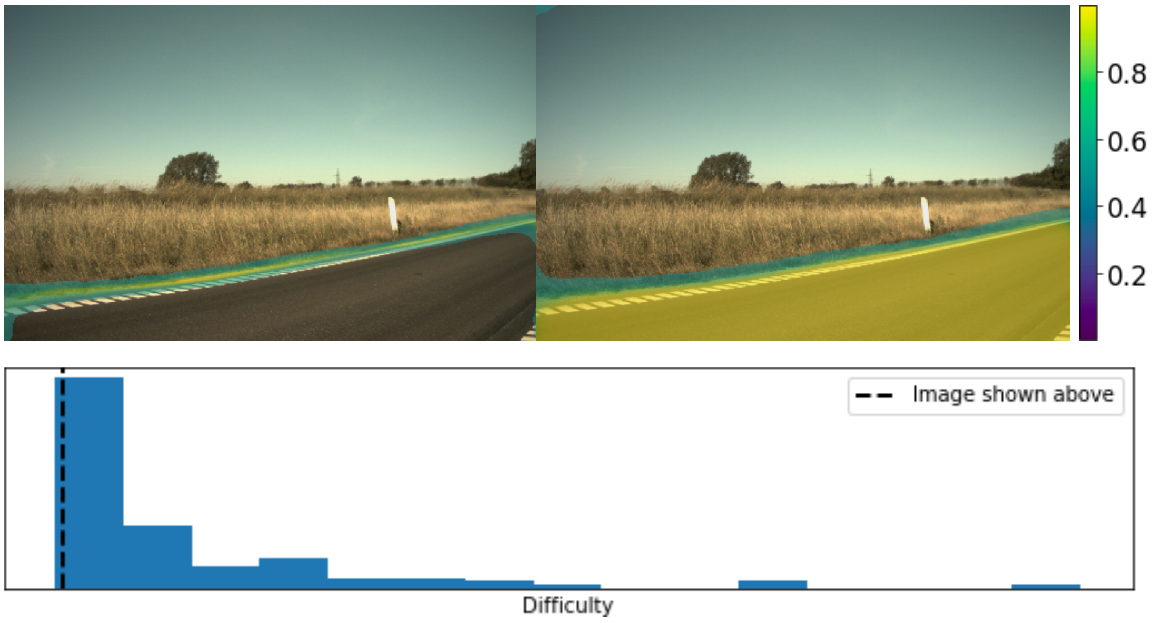

Figure 4.2: Heat maps of the multiple networks prediction variance and mean for an image from dataset A. This image is ranked 3/100 by the model, where 1 is the easiest. Zero values (purple) are hidden. **Top from left to right:** Prediction variance, prediction mean. **Bottom:** Histogram that shows the distribution of predicted difficulties, the dotted line shows the position of the image above in the distribution.

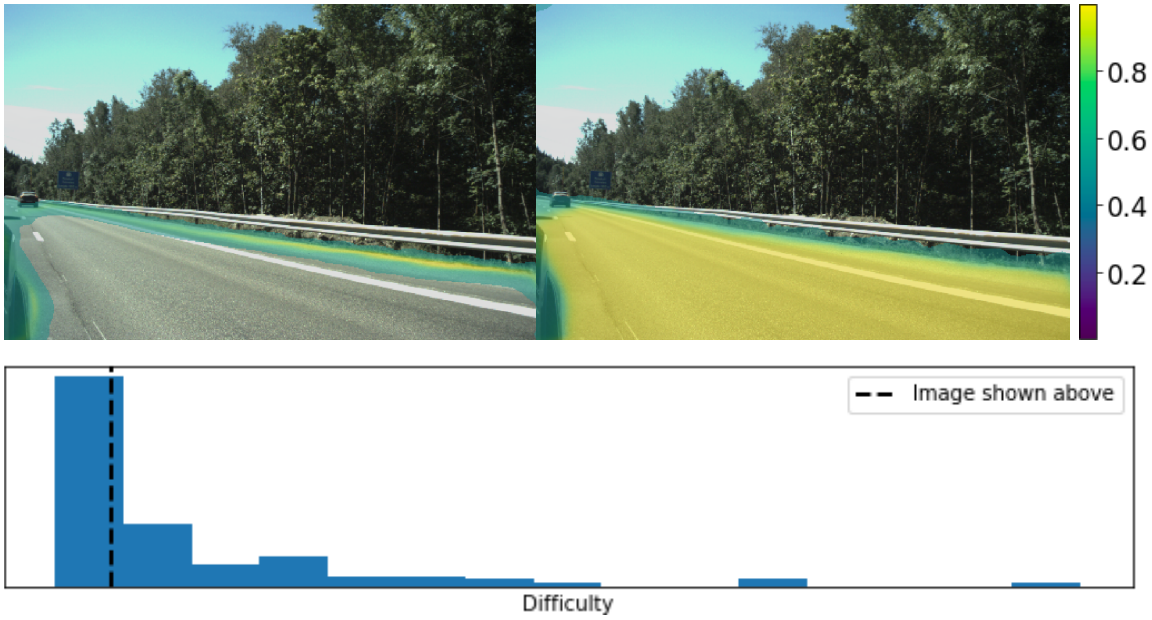

Figure 4.3: Heat maps of the multiple networks prediction variance and mean for an image from dataset A. This image is ranked 51/100 by the model, where 1 is the easiest. Zero values (purple) are hidden. **Top from left to right:** Prediction variance, prediction mean. **Bottom:** Histogram that shows the distribution of predicted difficulties, the dotted line shows the position of the image above in the distribution.

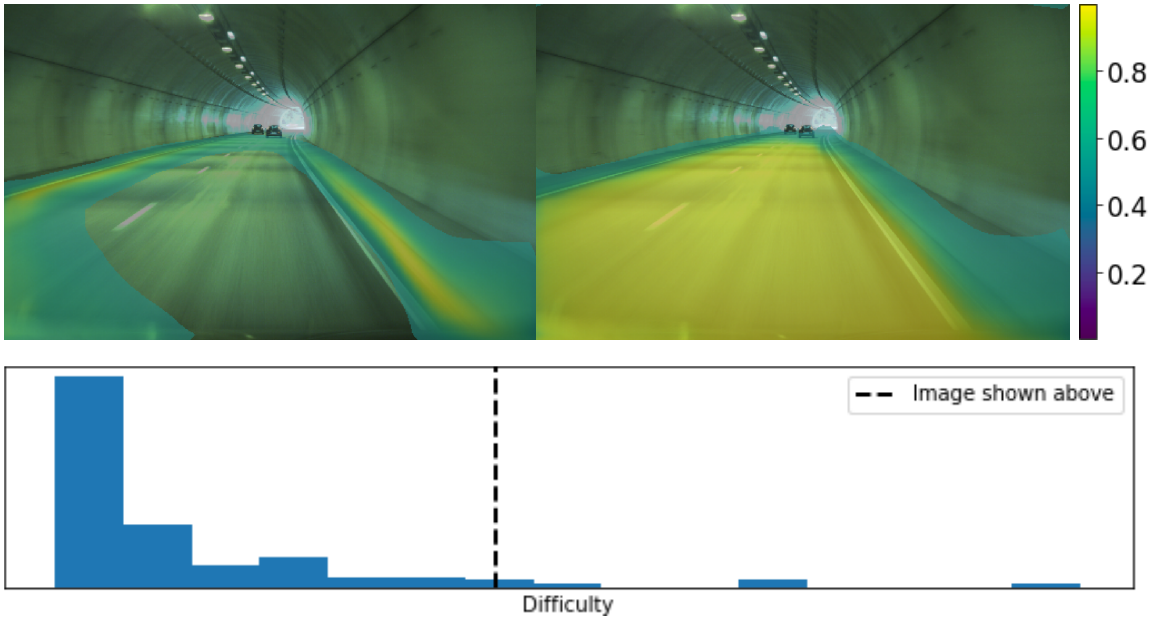

Figure 4.4: Heat maps of the multiple networks prediction variance and mean for an image from dataset A. This image is ranked 94/100 by the model, where 1 is the easiest. Zero values (purple) are hidden. **Top from left to right:** Prediction variance, prediction mean. **Bottom:** histogram that shows the distribution of predicted difficulties, the dotted line shows the position of the image above in the distribution.

4.2 Monte Carlo dropout model

This section shows the results of the Monte Carlo dropout model, applied to both dataset A and dataset B. The training and validation loss from both training sessions can be seen in figure 4.5.

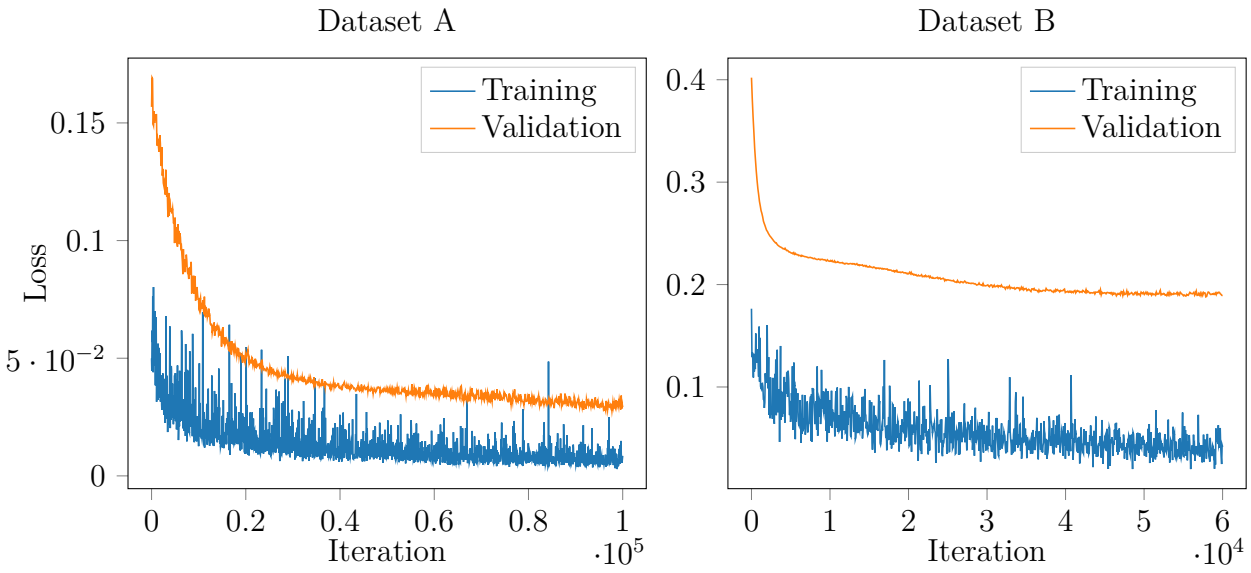

Figure 4.5: The loss during training for the training and the validation dataset when training for the Monte Carlo dropout model. The validation dataset consists of a very small number of images extracted from the training dataset.

4.2.1 Dataset A

The same images used for evaluation with the multiple network model are used again. They have been ranked as 12/100, 39/100 and 93/100 according to the Monte Carlo dropout model.

As with the results for the multiple network model, each image with both its prediction variance and its prediction mean as overlay in the image. Along with the prediction, each image is accompanied by a histogram that shows the distribution of the predicted annotation difficulties with the current shown image denoted by a dotted line.

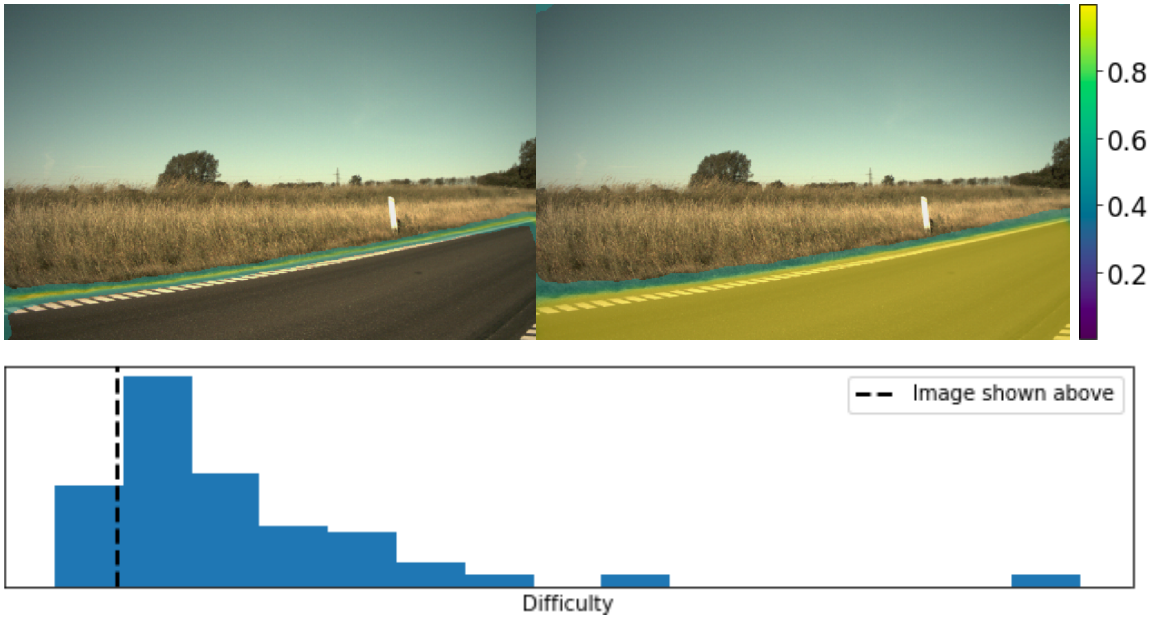

Figure 4.6: Heat maps of the Monte Carlo dropout model prediction variance and mean for an image from dataset A. This image is ranked 12/100 by the model, where 1 is the easiest. Zero values (purple) are hidden. **Top from left to right:** Prediction variance, prediction mean. **Bottom:** Histogram that shows the distribution of predicted difficulties, the dotted line shows the position of the image above in the distribution.

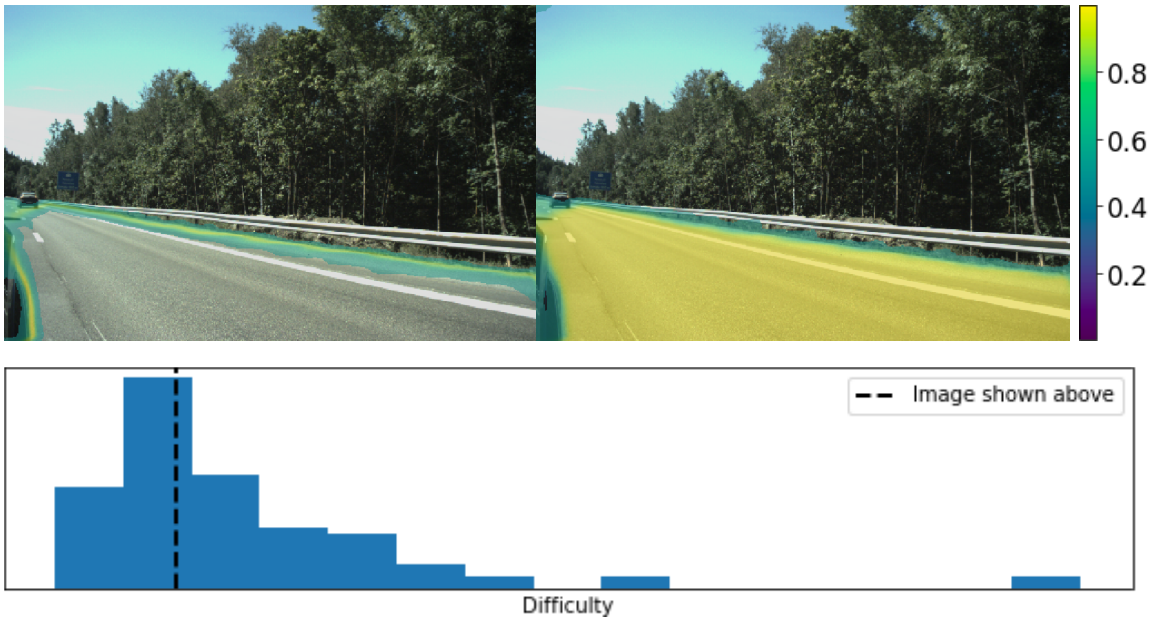

Figure 4.7: Heat maps of the Monte Carlo dropout model prediction variance and mean for an image from dataset A. This image is ranked 39/100 by the model, where 1 is the easiest. Zero values (purple) are hidden. **Top from left to right:** Prediction variance, prediction mean. **Bottom:** Histogram that shows the distribution of predicted difficulties, the dotted line shows the position of the image above in the distribution.

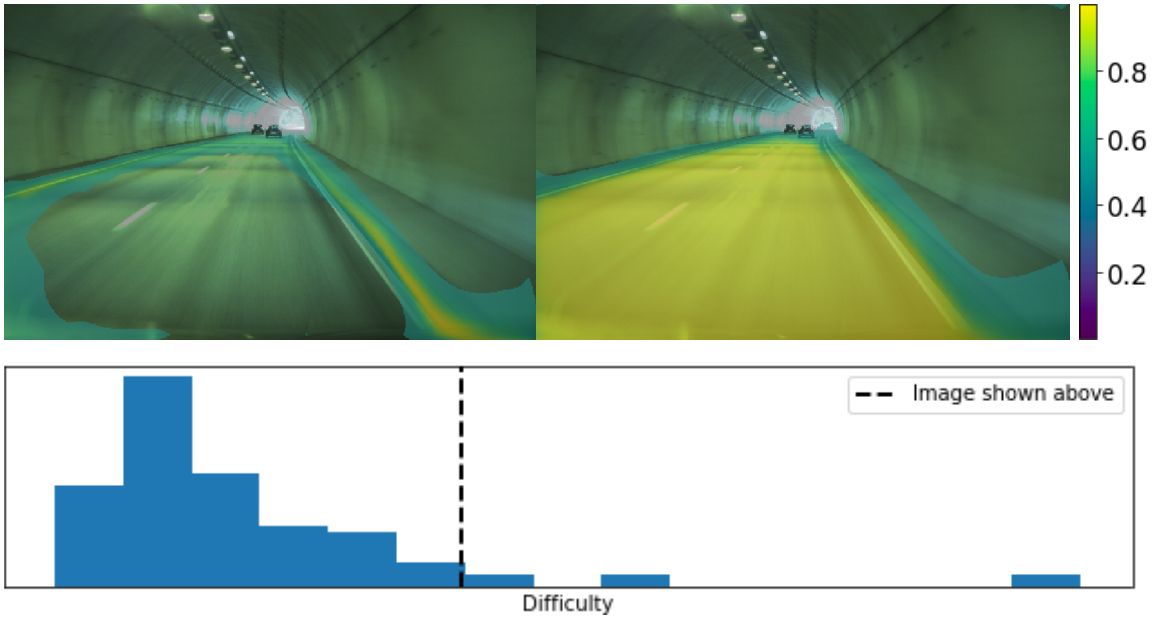

Figure 4.8: Heat maps of the Monte Carlo dropout model prediction variance and mean for an image from dataset A. This image is ranked 93/100 by the model, where 1 is the easiest. Zero values (purple) are hidden. **Top from left to right:** Prediction variance, prediction mean. **Bottom:** Histogram that shows the distribution of predicted difficulties, the dotted line shows the position of the image above in the distribution.

4.2.2 Dataset B

Three images from dataset B are presented in this section. They have been selected with the only condition being that the image should contain road. For each of the three images, two columns that represent the predictions (left column) and the ground truth (right column) are shown. From top to bottom, the columns contain mean, variance and a histogram that shows the distribution of difficulties with the current image denoted by a dotted line. For clarity, the left histogram shows the distribution of predicted difficulties and the right histogram shows the distribution of the difficulties calculated based on the ground truth data.

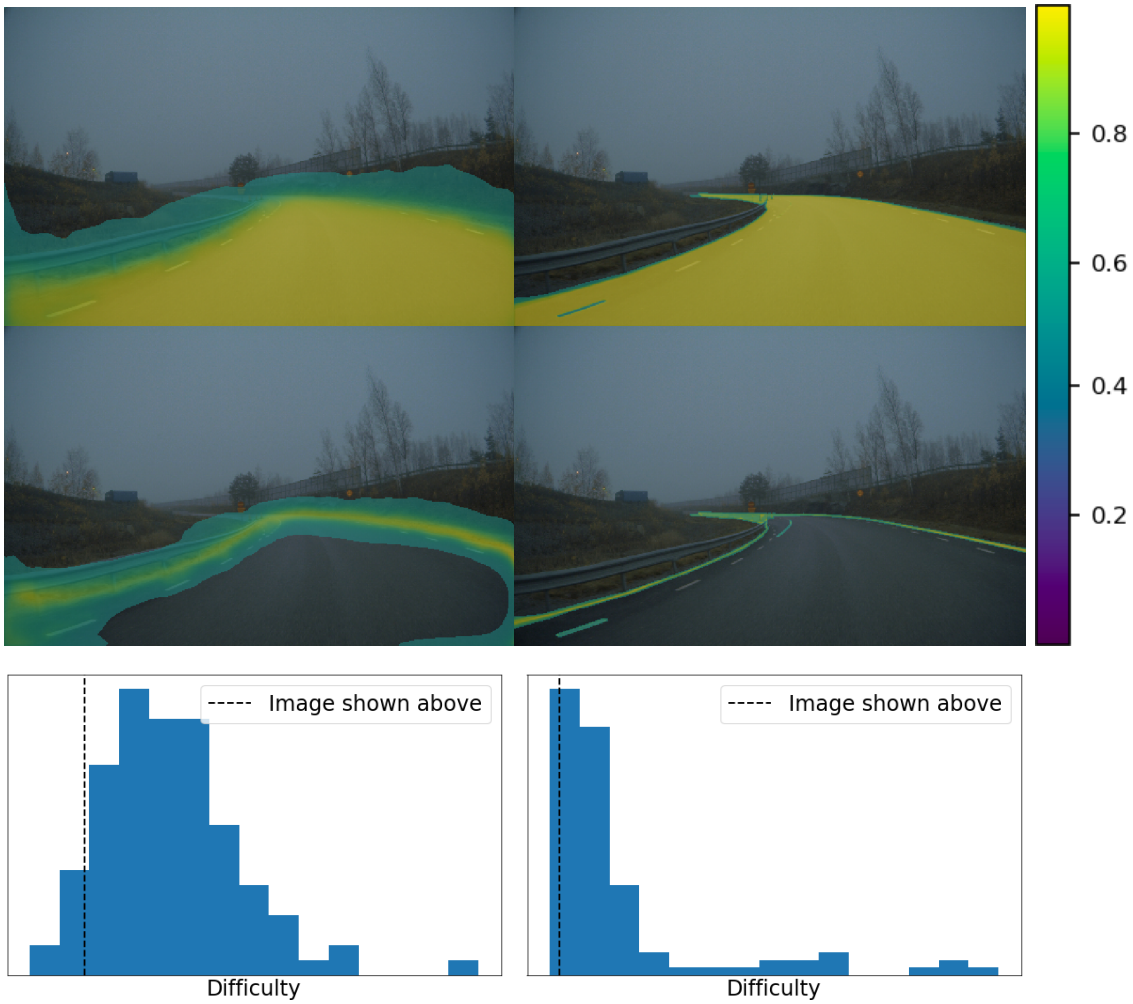

Figure 4.9: Heat maps of the Monte Carlo dropout model prediction variance and mean for an image from dataset A along with ground truth variance and mean. This image is ranked 6/100 by the model and 19/100 based on the ground truth data. Zero values (purple) are hidden. **Left column from top to bottom:** Prediction mean, prediction variance and histogram over the predicted difficulties with the image shown above marked with a dotted line. **Right column from top to bottom:** Ground truth mean, ground truth variance and histogram over the ground truth difficulties with the image shown above marked with a dotted line.

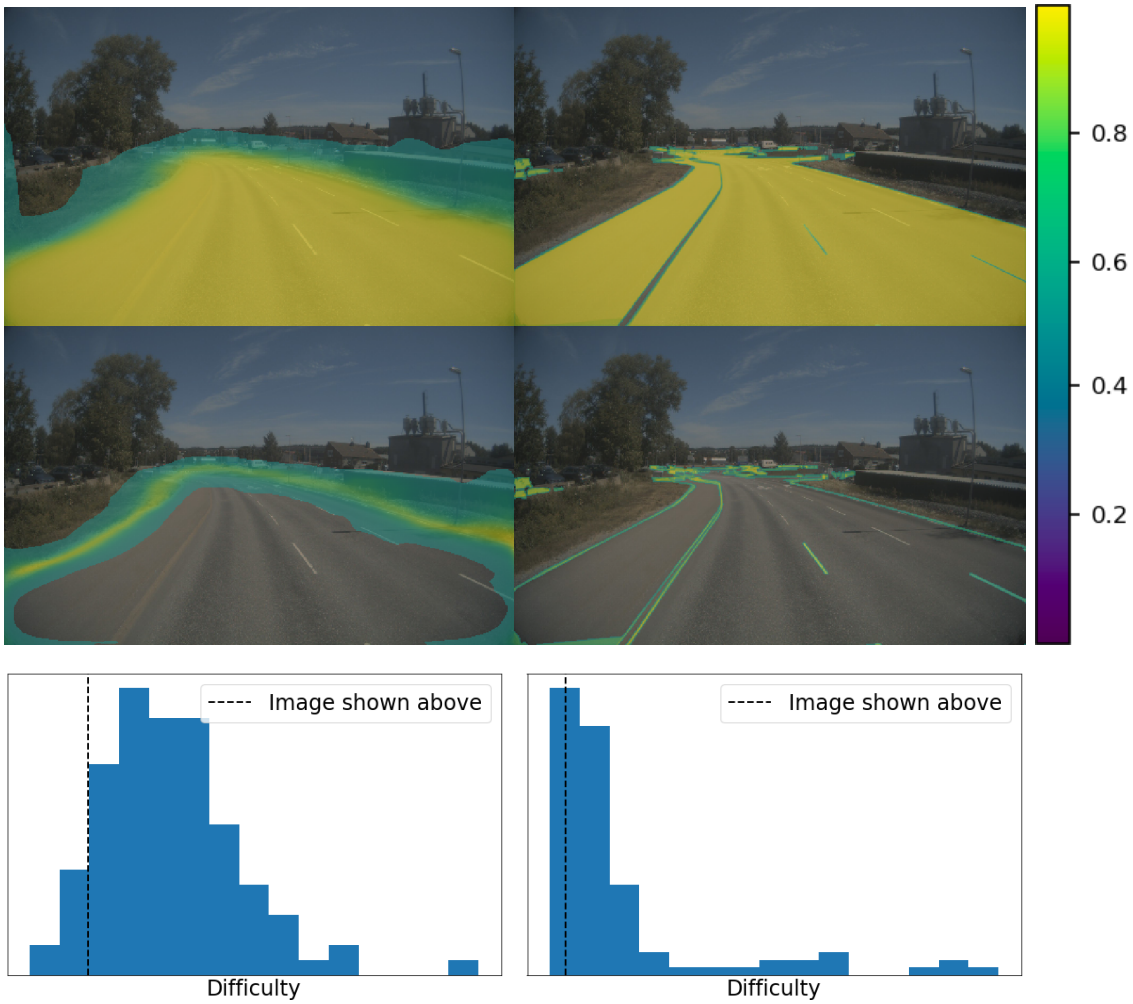

Figure 4.10: Heat maps of the Monte Carlo dropout model prediction variance and mean for an image from dataset A along with ground truth variance and mean. This image is ranked 8/100 by the model and 27/100 based on the ground truth data. Zero values (purple) are hidden. **Left column from top to bottom:** Prediction mean, prediction variance and histogram over the predicted difficulties with the image shown above marked with a dotted line. **Right column from top to bottom:** Ground truth mean, ground truth variance and histogram over the ground truth difficulties with the image shown above marked with a dotted line.

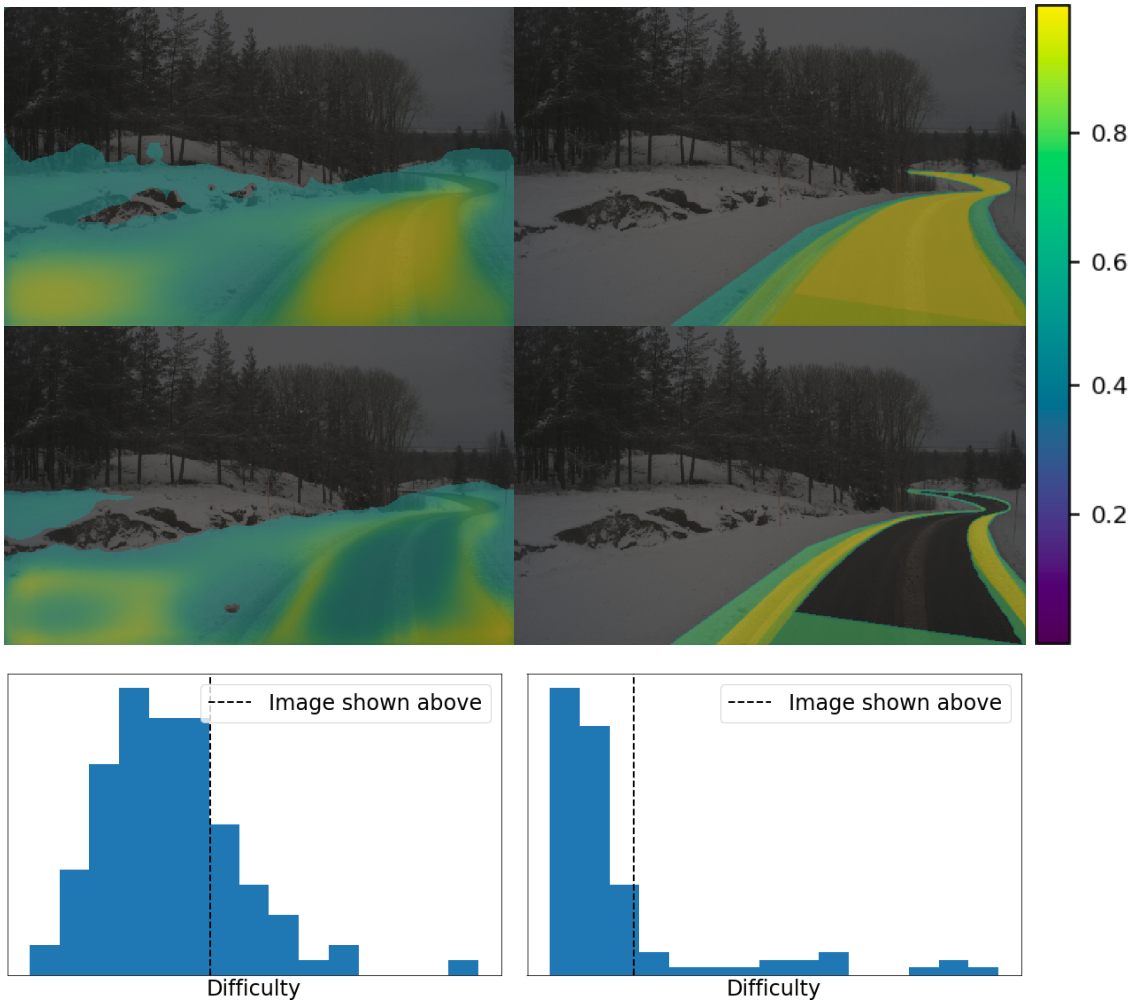

Figure 4.11: Heat maps of the Monte Carlo dropout model prediction variance and mean for an image from dataset A along with ground truth variance and mean. This image is ranked 76/100 by the model and 78/100 based on the ground truth data. Zero values (purple) are hidden. **Left column from top to bottom:** Prediction mean, prediction variance and histogram over the predicted difficulties with the image shown above marked with a dotted line. **Right column from top to bottom:** Ground truth mean, ground truth variance and histogram over the ground truth difficulties with the image shown above marked with a dotted line.

4.2.3 Prediction and ground truth correlation

The predicted difficulty ranks and the ranks calculated using the ground truth are used to create the scatter plot in figure 4.12. It displays the correlation between the predicted ranks and the ground truth ranks. The Pearson correlation coefficient (ranging from -1 to 1, where -1 is total negative linear correlation, 0 is no linear correlation and 1 is total positive linear correlation) for the two variables are 0.11, indicating no significant correlation. It should be noted that there are 14 images that have the rank 0 in the ground truth rankings. This is due to the fact that 14 images, according to the annotators, do not contain any road which give a difficulty value of 0.

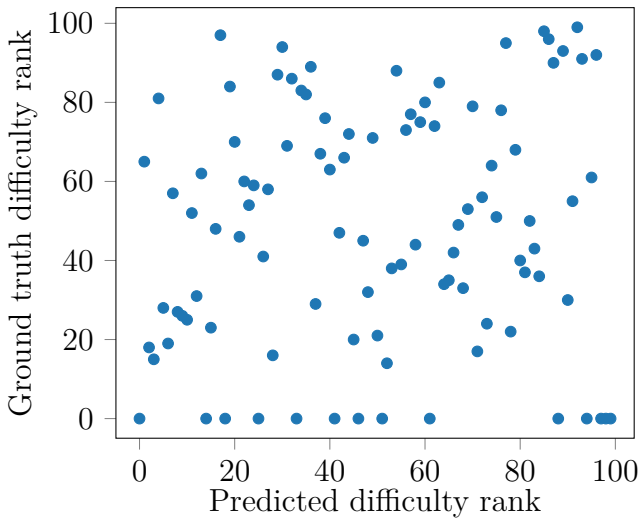

Figure 4.12: Scatter plot of the ground truth difficulty ranks and the predicted difficulty ranks. Note that 14 images among the ground truth rankings have the rank 0. This is due to the fact that these images, according to the annotators, do not contain any road which give a difficulty value of 0.

5

Discussion & Conclusion

In this chapter, the methods used are discussed and compared. The achieved results with the different methods are interpreted and analysed. Lastly, the conclusions for the project are presented and possible extensions and future work are discussed.

5.1 Multiple network model

The idea with the multiple network model was that each network should capture the underlying behaviour of the single annotator that created the training data for each network. However, the data used for training had been annotated according to the annotation process described in section 2.1 and all the used data was passed by the process. This means that both reviews and corrections of the annotations had taken place which arguably remove the annotation characteristics belonging to a single annotator. A possible solution to this could have been to use data that was only annotated once and not corrected at all for training.

Another relevant question concerning the multiple network model is if it mattered at all that the networks were trained on different data. Maybe the same result could have been achieved by training 10 networks on mixed data from all annotators, especially since the data was reviewed and corrected prior to training as mentioned above.

Unfortunately, the only way to evaluate the results from the multiple network model was through visual inspection. Simply put, does it look like the images ranked as hard could give rise to large variance among annotators? And does it look like the images ranked as easy could give rise to low variance?

Looking at the image ranked as 3/100 in figure 4.2, it contains no vehicles or other objects and it seems to be quite clear where the road ends, the road edge seem to be close to lane markings. It also has good lighting (no backlight).

The second image, ranked as 39/100, shown in figure 4.3 contains one other car. The side of the own vehicle is also slightly visible, which however has a very clear edge so it will probably not cause any disagreement between annotators. Even though a

barrier is visible on the side of the road, it is a somewhat wide region between the road edge and the breakdown lane. The lighting is good.

The third image, figure 4.4, is taken while driving in a tunnel and is ranked among the hardest to annotate (93/100). The image contains two other vehicles, but the part that seem to confuse the model is the road edge, it has a very large area of uncertainty. However, it is actually a curb on the side of the road so a real annotator would probably not have any problems with defining road edge. If the network would have been trained on more images of curbs and tunnels, it is quite easy to imagine that this image then not would have been ranked as particularly difficult since the curb acts as a very specific road edge indicator.

To summarize the analysis of the results, the ranking of the two easiest images seem to make sense while the image ranked as hardest out of the three probably not would be particularly difficult to annotate.

5.2 Monte Carlo dropout model

It is important to mention that there is no single correct way of measuring model uncertainty, Monte Carlo dropout is only one idea presented by Gal et al [12]. But obviously, using dropout is a very convenient way.

For dataset A, as with the multiple network model, the only available way of evaluating the result is through visual inspection. The three images are ranked 12/100, 39/100 and 93/100, in the same order as with the multiple network model. The results for these images are very similar for both models, one difference is however the wider areas of uncertainty for the multiple network model. This could be an artifact of more training, the networks used in the multiple network model were trained for 50000 iterations while the network used for Monte Carlo dropout was trained for 100000 iterations.

Another interesting detail to notice is the shape of the distributions, the Monte Carlo dropout model produces a wider distribution than the multiple network model. The fact that the distribution of the Monte Carlo dropout model seems to look somewhat like a Gaussian distribution give the impression that it is the desirable distribution since a lot of distributions tend to be Gaussian. But, looking at the ground truth distribution in figure 4.9 (for another dataset), it actually is more similar to the distribution of the multiple network model.

Dataset B opens for a more in-depth analysis thanks to the availability of ground truth data. The two figures 4.9 and 4.10 clearly illustrate that predicted mean and variance are very similar to the ground truth mean and variance. As expected, it is the areas at the edge of the road that causes a variance among the annotators, probably because there is a disagreement of where the road actually ends.

Figure 4.11 prove that snow is a challenging condition, it is very hard for both the

annotators and the model to predict where the road is when it is partly covered by snow. Even though the model has a lacking performance for this image, it is notable that it predicts large variance in the same areas as the ground truth, along the edges of the road.

Thanks to the available ground truth, a statistical analysis is also possible for dataset B. The predicted ranks and the ground truth ranks are scatter plotted in figure 4.12. It does however not look like there is a correlation between the ranks and the Pearson correlation coefficient verifies this.

I believe that the poor correlation can be derived from the lacking performance of the network which in turn can be derived from the small and irregular dataset. 800 images to train on is not enough to reach a decent performance, especially when the dataset contains a very wide range of scenarios and angles. This hypothesis is strengthened by the fact that in images such as the one in figure 4.9, it can be seen that the predicted variance clearly resemble the ground truth variance but with big areas of uncertainty.

5.3 Conclusion

The following research questions were asked in the beginning of this thesis:

- What does it mean that an image is difficulty to annotate?
- How do you predict the difficulty of annotating an image?

The proposed answer to the first question is that an image is difficulty to annotate if the image, when given to several people, yields a big variance among annotations. In other words, the image is hard to annotate if the level of disagreement between annotators is large.

A possible answer to the second question is that it can be predicted by quantifying the uncertainty of a suitable neural network trained on a satisfactory amount of data. Two methods are proposed in this thesis, the multiple network model and the Monte Carlo dropout model. Both models give similar results, but the Monte Carlo dropout model works with a fraction of the training that the multiple network model requires. Furthermore, it also saves time at inference since only a single network needs to be loaded. Therefore, the Monte Carlo dropout model is the recommended model for predicting annotation difficulties. And even though the results did not prove the method entirely, there are intuitive reasons to motivate further investigation.

5.4 Future work

A good next step would be to evaluate the Monte Carlo dropout model further, for example by creating a ground truth for dataset A and perform the same correlation analysis for dataset A as was done for dataset B. This would be necessary to ensure that the model works in the expected way.

The approach is also very dependent on a well-trained neural network, so a natural development would be to increase the performance of the network. It could be achieved by training on more data and augmenting data (cropping, mirroring etc.). Furthermore, if time and computing power not is a limitation, tuning of hyperparameters can be done to enhance the training procedure and eventually the performance of the network.

Bibliography

[1] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[3] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[4] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 253–256, IEEE, 2010.

[5] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[9] "Pascal voc segmentation results: Voc2012." `http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=6`. Accessed: 2019-01-31.

[10] A. Der Kiureghian and O. Ditlevsen, "Aleatory or epistemic? does it matter?," *Structural Safety*, vol. 31, no. 2, pp. 105–112, 2009.

[11] T. O'Hagan, "Dicing with the unknown," *Significance*, vol. 1, no. 3, pp. 132–133, 2004.

[12] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, pp. 1050–1059, 2016.

[13] Y. Gal, *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

[14] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

A

Appendix 1

Parameter	
Learning rate	10^{-7}
Weight decay	0.0005
Momentum	0.99
Batch size	4
Dropout probability	0.5

Table A.1: Parameters used for training. All networks were trained with the same parameters.