CHALMERS
UNIVERSITY OF TECHNOLOGY

Machine assisted semantic instance segmentation with interactive improvement

Master's thesis in Computer science and engineering

Elin Nordström
Karl Strigén

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Machine assisted semantic instance segmentation with interactive improvement

Elin Nordström
Karl Strigén

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

Machine assisted semantic instance segmentation with interactive improvement

Elin Nordström
Karl Strigén

Supervisor: Lennart Svensson, Department of Electrical Engineering
Advisor: Isak Hjortgren & Adam Brinkman, Annotell
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2020
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2020

Machine assisted semantic instance segmentation with interactive improvement

Elin Nordström
Karl Strigén
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Semantic segmentation is a supervised learning problem, where each pixel in an image is labelled with its category. This is of great importance when designing autonomous vehicles. Supervised learning problems require plenty of labelled data with a high level of detail. Annotating such data is currently done by human annotators to meet the high quality requirements, which is why the annotation process takes a lot of time. Having machine learning algorithms handle simple problems and let the human annotator focus on annotating only the more difficult details could therefore speed up this process, making it less costly.

We investigate the machine learning assisted annotation process, where the idea is that an annotator interacts with a neural network to produce the segmentation. The initial annotator input to the network is in the form of clicks on the object's extreme points, which allows the network to produce a first instance segmentation. Given this suggestion, the annotator may provide additional input to the network to further improve the segmentation.

In this thesis we simulate these user interactions and investigate their impact. We also investigate different types of user input to improve the segmentation, such as object contour clicks, positive clicks on the object pixels and negative clicks on the background pixels. In order to evaluate the impact, we train one network based on the ideas of a network model called DEXTR and another network with a network module called PointRend.

The key findings in our experiments are that additional clicks improve performance, but that most of the gain is achieved from the extreme clicks. Also, the type and amount of additional clicks has a marginal impact on the final performance. This suggests that user guidance to a network in the form of clicks is promising for speeding up the annotation process while maintaining high quality.

Keywords: Computer vision, autonomous vehicles, semantic segmentation, instance segmentation, assisted annotation, deep machine learning.

Acknowledgements

We would like to thank our supervisor Lennart Svensson for his guidance, interesting discussions and tons of great ideas.

Thanks also to our advisors at Annotell, Isak Hjortgren and Adam Brinkman for taking their time to answer all of our questions, even at inconvenient times. Special thanks to Isak for more GPU power when we needed it the most. On that note we would like to thank Daniel Langkilde and the Annotell team for believing in us and letting us use your data for this thesis.

Finally we want to thank Niklas Gustafson, Gustav Rödström, Amanda Belfrage and David Berg Marklund for their feedback on the drafts of this report.

Elin Nordström, Karl Strigén, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1
Introduction

Autonomous vehicles is currently a hot topic around the world, since they have the possibility of increasing safety and decreasing environmental impact of the automotive sector. The safety requirements of a self-driving car are very high, which means that its decision making systems need to be as accurate as possible. In order to train these systems to a degree where they might be trusted with the responsibility of human safety, an extensive amount of labeled training data is required. This is because the machine learning algorithms for autonomous driving are often supervised learning algorithms.

One type of supervised learning problem for perceiving the world is semantic segmentation. The task is to label each pixel in the image with its category, e.g. pedestrian, car or road. In addition to distinguishing each category of objects in the surroundings, it is also important to keep track of each object instance of a category. This is called semantic instance segmentation and it provides a high level of detail. This pixel accuracy of the perception of the surroundings is very important for autonomous vehicles, in order to make the correct decisions when driving.

In recent years, semantic segmentation algorithms have improved their performance remarkably. Many new challenges to measure the performance has also been designed, such as the Cityscapes [8] dataset. It consists of city street images where the task is to label each pixel in the image. The top performers currently reach an accuracy of approximately 84% [31]. Although the accuracy of the machine learning solutions for traffic scenes is increasing, it is still not accurate enough. This means that the algorithms alone are not enough to produce training data of high enough quality for autonomous vehicles. Because of the necessary high quality standards of the data, it is currently created manually by human annotators that mark each object in an image. Creating the data manually is a tedious process that requires a lot of attention to detail and it takes a lot of time. Since it is highly time consuming, it is also expensive to create the vast amounts of needed data.

If the human annotator could be assisted with an algorithm that provides suggestions of the annotation, the process takes less time per image and becomes less expensive. An intuitive way to accomplish this is to combine the strengths of a human annotator with the strengths of an algorithm for semantic segmentation. This could mean to let the machine learning algorithm take care of more straightforward problems and to let the annotator focus on more difficult areas and detecting the objects. Ideally, the combination of a human and an algorithm then could produce

data faster but still with the same high quality.

On the topic of machine learning assisted annotation, there exists some recent work [1][17][20] where the authors incorporate user input to let an algorithm make a suggestion. When provided with the suggestion, the annotator may interact with the system in some way to hopefully guide the network into making better predictions. Although these techniques reach good performance, they all use different methods of incorporating the annotator input. As the input form varies between the solutions, so does the resulting segmentation from the algorithm. Because of this, it is difficult to find out *what kind* of user input affects the algorithm in a desirable way and *how* it affects it. More work is therefore needed to decide definitely on which form of annotator interactions are optimal for the performance of the algorithm and to establish the respective impact of these interactions. We believe that there is a lot of possible and necessary research left to be done on how the human and algorithm can work together. This thesis will therefore focus on investigating two types of user input, their impact on the resulting semantic segmentation and how to train networks that can make use of the annotator input. In order to study the impact more thoroughly, the annotator input is also tested on networks not previously examined with it.

1.1 Problem statement

This thesis aims to investigate different ways of incorporating user input into a neural network with the intention of improving the process of creating training image data for autonomous vehicles. The thesis is done in collaboration with Annotell, a company providing annotated training data for autonomous driving. A challenge with annotating high quality training data is that it requires extensive manual work and thus it is time consuming and costly to produce. The goal, from a business perspective, is to improve on this process both in means of time spent on each data sample but also to make the annotation experience more convenient for the annotator.

Manual annotation of objects is done by labeling groups of pixels in the image as either an object of a specific class or as background. Just detecting the objects is generally not enough level of detail, but instead a classification of each object pixel is needed. The problem of labeling each pixel in an image is in machine learning referred to as *semantic segmentation*, and to also differentiate between instances of the same class is referred to as *semantic instance segmentation*.

It is worth noting that when a human does this work, the difference between the two problems of semantic segmentation and semantic instance segmentation is smaller since a human generally has no problem distinguishing between different instances of the same object class in an image. In the current workflow in Annotells application the human detects the object by drawing its contour in full as a polygon. In this thesis it is the user who detects the object instance by providing initial input

to the algorithm, and the neural network does semantic segmentation of that object instance.

The user input is easier and faster to provide than to manually label the whole image. It will be used to provide guidance to a machine learning algorithm. After this initial input is provided, the next goal is that it should be easy for the annotator to improve the segmentation provided by the algorithm by providing further input. That process is what we will often refer to as "interactive improvement" of the result and it is needed due to the high quality requirements of the data.

Research questions

The research questions that the thesis attempts to answer are:

- How can the user input be incorporated to the algorithm input in order to guide the segmentation?
- What kind of user input is best for interactive improvement of the segmentation.
- How could training data be created in order to simulate user behaviour? Specifically, how can we create training data such that the network learns to do interactive improvements to the segmentation?

We aim to examine existing techniques for machine assisted semantic instance segmentation and extend them to better be able to incorporate user input and interactive improvements. This includes implementing different machine learning algorithms with different network architectures. Some will be very similar to already existing solutions while the interactive improvement methods might be more novel in their approach. Different ideas on how to implement the interactive improvements and how to construct training data that is useful for this process will be discussed further on in the report.

1.2 Scope

This thesis builds upon the ideas for user input that was presented in DEXTR [20] and how to extend them to make more use of interactive corrections to the provided segmentation. We will also examine the PointRend [17] module and its potential in combination with the user input data. Due to time concerns, the initial input from the user will only be of one type and the network backbone will be the same for all architectures. To simplify the use of the PointRend module, instead of applying it as a separate final step to the architecture used for the other tests, it will be implemented via the package made public by FAIR, which also builds on ResNet, albeit with a somewhat different overall architecture.

As for annotated data, it could be of many types, but in this thesis the data is 2D RGB-images of road environments. One image can contain multiple objects, but for each sample we will only consider one object at a time. This simplifies the problem and utilizes the human annotator to detect the object.

The final delimitation is that there is no time to implement our work into an actual application. This means that our findings can not be examined regarding the actual decrease in annotation time and whether or not the quality improves in practice. This is unfortunate, but hopefully the findings will warrant further study regarding the impact of these algorithms on the work of the annotators.

1.3 Contributions

Our contributions are:

- Trying out more than one click on the contour as well as positive/negative clicks as extended input for the network architecture presented in DEXTR.
- Using extended input data in combination with a PointRend module.
- We investigate some possibilities of creating training data to mimic the behavior of a human annotator and its impact on the segmentation.
- We show that extending the input generates better segmentation, while the differences in how much we extend the data gives a rather small impact on the segmentation.
- We show that the PointRend module can also make use of extended input, and that it generates a very good segmentation when doing so. This would merit using the PointRend module as a standalone module in networks used for interactive annotation.

The results regarding the PointRend module are significant because it has not previously been evaluated with input extended in any of these ways. We also confirm the findings of DEXTR regarding the impact of the extreme clicks and of adding one contour click as well as showing that positive/negative clicks can be used in a similar fashion.

Finally, our strategies on creating training data that tries to mimic the work of a human annotator does not increase the accuracy of the segmentation. Our thoughts on this will be presented more in depth in chapter 5.

1.4 Related work

Many existing techniques for semantic segmentation and semantic instance segmentation build on convolutional neural networks(CNN). Current approaches that generate good results for semantic segmentation with deep CNNs are for example the encoder-decoder architecture proposed by Chen et. al. in DeepLab v3 [7] and the work by Yuan, Chen and Wang [31] where they make use of object-contextual representation for their classification layer. These works are noteworthy since they are some of the current state of the art solutions for semantic segmentation.

Another approach that makes use of the context in the image is the region-based CNNs (R-CNNs)[11] where focus lies on a few regions of interest (RoI) for *object detection* [24] [23]. Most current approaches for object detection builds on this technique. This idea was extended in Mask R-CNN [14] where they combined the

approach of object detection with *mask prediction*. This architecture is central in the PointRend module [17] presented below and used frequently in this thesis. Other approaches have replaced the CNN with a recurrent neural network (RNN) [25] to show that this is a viable option. Another idea is to extend the CNN with a Conditional Random Field (CRF) [2] to improve on predictions by getting more information from the context. In DeepLabv2 [6], Chen et. al. included a fully connected CRF to improve localization and also endorse parallel dilated convolutions capturing contexts of various scale. Another work to capture multi-scale context is the PSPNet [32] by Zhao et. al. which will be used as the classification module of many of the networks in this thesis.

On the topic of *interactive annotation*, where a user is meant to adjust or edit the segmentation in some way in order to improve the accuracy, there are a few different approaches. Some take input by a user to guide the network and others create polygons around the object for the user to adjust. An example of the latter is the work by Acuna et. al. [1] where they have an encoding CNN, an RNN and a gated graph sequence neural network (GGNN) model. The CNN extract image features as an encoder and then the RNN decodes one polygon vertex at a time. Lastly, the GGNN upscales the polygon to the required resolution. Their idea is to let an annotator drag-and-drop a bounding box around the object and then possibly adjust the resulting polygon vertices. Another solution, instead of sequentially predicting each edge of the polygon, predicts all vertices at the same time by using a Graph Convolutional Network [19]. On the topic of polygons, the PolyTransform [18] instance segmentation algorithm by Liang et. al. performs well. They use a segmentation network to extract the instance masks in an image and then convert them to a set of polygons. These are then fed to a *deforming* network that transforms the polygons to better fit the object contour.

One interactive approach that does not use polygons and is more similar to our work is to use clicks of the user in order to guide the network mask prediction. DEXTR [20] took user clicks as additional input in the form of heatmap channels to enhance performance. The extreme points of the object was used as an alternative to bounding boxes, based on the idea from Papadopoulos et. al [21] and are marked by an annotator on the four extreme coordinates of the object. This is the inspiration for the extreme click model implemented in this thesis. Another approach is to take two different types of clicks placed either on object pixels or non-object pixels [29]. This, in turn, is the inspiration for the positive/negative click model implemented in this thesis. Another idea is to combine CNN models with level set evolution as described by Wang et. al. [26]. This improves the accuracy even more while still using the same extreme clicks as input from the annotator.

A rather different approach is to look at image segmentation as a rendering problem and to use ideas from the field of computer graphics. In the recent work called PointRend [17], their module takes a feature map output from a CNN and extracts a feature representation for certain points where there is a need for a finer level of detail in the segmentation and predicts a label from the point-wise feature representation

for each of these points. The idea is partially based on the computer graphics approaches of subdivision [27] to refine a coarse pixel grid where the values have higher variance.

2
Theory

In this chapter are descriptions of the key theoretical concepts for the work presented in this thesis. Most central is the use of deep convolutional neural networks, a machine learning architecture used for most modern approaches to computer vision tasks. The loss function we use to solve these problems will be highlighted, and its key difference compared to other loss functions.

We will describe how our work relates to the most similar computer vision problems, and what the key differences are. The differences are highlighted by the use of extended input data, as described below, and how we can incorporate this into our algorithms. We will also describe the theory behind the PointRend module that can be used to improve the initial segmentation.

2.1 Semantic segmentation and semantic instance segmentation

Semantic segmentation is the task of deciding, for every pixel in an image, what class it belongs to. Formally: consider an image with a set P of size $|P|$ pixels p, such that:

$$p_{i,j} \in P, \tag{2.1}$$

where i and j are the coordinates of said pixels. We have the set of labels:

$$\mathcal{L} = \{l_1, l_2, \ \dots \ , l_n\}. \tag{2.2}$$

From that we wish to find the labelling function $\mathcal{F}$ such that:

$$\forall p \in P, \mathcal{F}(p_{i,j}) = l_x. \tag{2.3}$$

Semantic instance segmentation aims to accomplish the same classification of the pixels but to also distinguish between different instances. The task is not only to label what class each pixels belongs to, but also which instance of that class. Formally, this compares to the case described above such that each $l \in L$ can exist in many different instances. The sought after function $\mathcal{F}$ is still the same but the set of labels can be defined as

$$\mathcal{L} = \{l_{11}, l_{1m}, l_{2m}, \ \dots \ , l_{nm}\}, \tag{2.4}$$

where n is the number of classes and m is the number of objects of that class. m is not necessarily the same for all n.

If the human annotator takes care of the *object detection*, the developed algorithms can focus on semantic segmentation. Another part of the problem of semantic segmentation that humans are quite good at solving is *classification*. If we also let the human decide what type of object it is annotating, the algorithm only needs to distinguish between foreground and background. This is known as *class agnostic* instance segmentation and it corresponds to setting $n = 1$ in equations 2.2 and 2.4. This also works well in combination with the annotation software of Annotell since it gives the annotators the possibility to denote the class of the object they are annotating. Thus, as the network solves semantic segmentation, the annotator can solve object detection and classification.

2.2 Deep convolutional networks

Convolutional neural networks are commonly used in computer vision and throughout the years there has been a trend of deepening the networks, adding more layers, with the intention of learning harder problems. This is also the case for semantic segmentation, with the recent top performers making use of deep network models. In this thesis, deep networks is used for all the developed models and as such some theory on how they work is beneficial.

Due to the sheer amount of learnable parameters of deep neural networks, it is not feasible to have them operate on images of the same size as modern cameras produce. Images today tend to be thousands of pixels in both width and height and no commercially available computers have the computational power to calculate semantic segmentation for images of such sizes in a reasonable time frame. The simple solution to this problem is to downsample both the input images and their ground truth segmentations. All of the current state of the art methods require downsampling in such fashion.

A technique that is of great importance when it comes to training deep networks is transfer learning. A term first discussed by Pratt et. al. in 1991 [22], which simply refers to the concept of initializing one network with parameters from another network that has trained for a long time on large amounts of data. Its importance increases with the network depth, since training deeper networks takes longer time. To "kick-start" the training with weights from another model is very beneficial. The most common models from which to transfer weights are usually trained on the ImageNet [9] data, used by for example [7], [14] and [20].

It stands clear that deep networks should be better at solving computer vision problems than their shallow counterparts. However, achieving the sought after network depth by simply stacking layers upon each other is often followed by a network that is harder to train. The difficulty arises due to the gradient being back-propagated to earlier layers and becoming very small, and the issue is therefore called the *vanishing*

gradient problem, identified by Hochreiter in 1991 [16].

The vanishing gradient problem has been dealt with by batch normalization and various other techniques, depending on the field of the application. This allows the deeper networks to converge, but then there is still a problem of *degradation* [15], which is a problem for deep networks where they perform worse than their shallow counterparts. Such differences in performance are not caused by overfitting, but instead the stacking of more layers leads to a higher *training error*. The phenomena is not intuitive since there should be a way for the shallow network to be encompassed in a deeper network, with all other connections being identity mappings. Such a constructed example would theoretically mean that a deeper network should never produce a higher training error than its shallow counterpart. The problem arises from the underlying mapping being hard to learn for the network. Simply put, it is theoretically possible to create a deep network in the way described above but an actual network seem to never behave this way. This problem is discussed in length, together with a proposed solution, by He et. al. in their paper on the network architecture ResNet [15].

2.2.1 ResNet

ResNet [15] is one of the most widely used backbones for deep convolutional networks today. In the architectures presented further on in this chapter, a ResNet backbone is always used, and the aim is to give an understanding of its merits and why it is used for so many network architectures.

The main idea behind ResNet [15] is to use a shortcut connection and therefore avoid the degradation that happens when stacking layers. He et. al. [15] notes that previous algorithms seem unable to improve on this issue, and instead they propose a solution to this by aiming the stacked layers to map towards a residual function rather than the original one. The way that this works is by having residual building blocks that typically have double- or triple-layer skip connections.

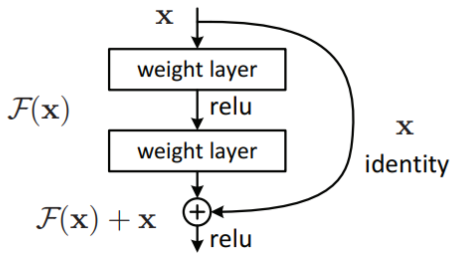

Figure 2.1: Residual block. The function F represents the residual mapping and the operation $F + x$ is performed by a shortcut connection and then element-wise addition. Image taken from [15].

Formally, consider that one wishes the stacked layers to converge towards the underlying mapping $H(x)$. One can then construct the mapping $F(x) := H(x) - x$

and we have the original mapping as $H(x) = F(x) + x$. He et. al. prove [15] that this greatly reduces the impact of the degradation problem.
The building block is formally defined as

$$y = F(x, W_i) + x, \tag{2.5}$$

where x is the input vector and y is the output vector of the layers. The function F represents the residual mapping and the operation $F + x$ is performed by a shortcut connection and then element-wise addition, as seen in figure 2.1.

2.2.2 Pyramid scene parsing

Semantic segmentation is pixel-wise classification task. When performing such a task, the context of the image is highly relevant. Mostly because most pixels will be of the same class as their neighbours. A neural network can make use of this via the Pyramid Pooling module presented by Zhao et. al. [32].

The key point here is that the label of a pixel is highly dependent on its context, i. e. the pixels around it. If this contextual information is incorporated in the learning algorithm, the predicted labels tend to be better than when such information is not used [32] [31]. The same thing goes for when the problem is not class agnostic, for example it is very rare to find an object of the class *car* surrounded by pixels classified as *water*. The correct label for the first object is then perhaps *boat*, which shows the importance of information from the surrounding pixels.

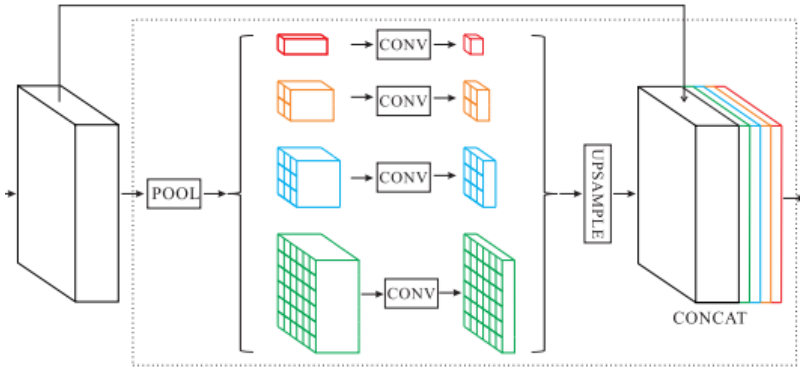

Figure 2.2: The pyramid pooling module from the PSPNet. Image taken from the paper Pyramid Scene Parsing Network by Zhao et. al. [32].

The scene parsing problems, which are based on semantic segmentation, are often solved with deep convolutional neural networks (DCNN). The *receptive field* of a DCNN is an indication of how much contextual information we make use of in every layer, and is smaller for these types of deep networks in the high-level layers. This means that the network may not take the global context into account and therefore make mistakes that come from insufficient scene understanding.

In order to utilize global features, the pyramid scene parsing network (PSPNet) was proposed by Zhao et. al [32]. The module of interest in the PSPNet is the

pyramid pooling module which is applied to an output feature map of a CNN. The module combines the features under four different scales, where the coarsest level, highlighted in red in figure 2.2, generates a single output and the finer levels divides the feature map into different sub-regions. They apply a $1x1$ convolutional layer after each pyramid layer to reduce the dimension of the context representation and therefore maintain the overall feature weight. The low-dimension feature maps are then up-sampled to the same size as the original feature map and concatenated. The final pyramid pooling global feature, which is the output of the PSPNet, is the concatenated feature levels. An image of the pyramid pooling module can be seen in figure 2.2.

2.3 Loss function

When a neural network "learns" something, what happens is that the weights of the network is updated. This is done by *backpropagation*. When calculating the values in this way, a loss function is needed to steer the weights in a direction that leads to better output from the network. The choice of loss function generally depends on the problem, and the choice is then between those usually used for semantic segmentation.

For the task of class-agnostic instance segmentation we have considered the same loss functions as examined by Maninis et. al. in the DEXTR paper [20]. The first one is the standard cross-entropy loss which is described by Goodfellow et. al. [13] (also by many others). However, since the occurrence of object and background pixels is not necessarily equal in the images, especially not after data augmentation, class balanced cross-entropy loss can be used to give importance to less frequent classes. The difference between these loss functions can be noted by looking at their equations,

$$L_s = -\beta \mathbf{Y}^* \log \hat{\mathbf{Y}} - (1-\beta)\,(1-\mathbf{Y}^*) \log(1-\hat{\mathbf{Y}}). \tag{2.6}$$

Compare this to the loss function of the regular cross-entropy loss:

$$L_s = -\mathbf{Y}^* \log \hat{\mathbf{Y}} - (1-\mathbf{Y}^*) \log(1-\hat{\mathbf{Y}}). \tag{2.7}$$

In both equations, $\mathbf{Y}^*$ signifies the prediction and $\hat{\mathbf{Y}}$ is the ground truth. It is important to note that $\mathbf{Y}^*$ is not yet a binary classification but a value between 0 and 1 defining the prediction score that this specific pixel is part of an object. The loss functions are also averaged over all samples when used in training.

We clearly see that the difference between equation 2.6 and equation 2.7 is the β-term. It is defined as:

$$\beta = 1 - \frac{\sum_{y^* \in \mathbf{Y}^*} y^*}{|\mathbf{Y}^*|}. \tag{2.8}$$

This signifies the number of negative samples, i. e. the number of background pixels, divided by the total number of samples. It helps with the class imbalance problem

by generating a smaller loss value when the dominant class is evaluated and thus reduces its effect on the output.

2.3.1 Logits

Since the classification step is often a softmax function, it is common to have the final representation of the output of the network in the form of *logits*. Logit is a mathematical function mapping probabilities, that are in the range $[0, 1]$ to the range $[-\infty, \infty]$. This mapping is done according to equation 2.9 where L is the logit and p is the probability,

$$L = \ln \frac{p}{1-p}, \quad p = \frac{1}{1+e^{-L}}. \tag{2.9}$$

This is of importance because translation between logits and probabilities might be needed to get correct values from the loss functions. The binary cross entropy loss and the class-balanced cross entropy loss operates on the probabilities and not the logits.

2.4 Extending the input data

To produce data such that the algorithm can do semantic segmentation for one annotated instance, a method for denoting objects in an image is needed. In this thesis we aim to utilize the user input to guide the network to a better segmentation, and in order to understand that process we explain the founding solutions upon which this thesis is built in this section .

2.4.1 Extreme clicks

When annotating objects in an image, the typical process is to draw a bounding box around the object. Manual annotation in this case means that the annotator needs to carefully draw a box around an object and match the x- and y-axis to the object limits. The crowd-sourcing framework by Papadopoulos et. al. propose an alternative way of annotating the object by instead of drawing a box around it, clicking on the object's extreme coordinates [21]. The extreme coordinates of an object in an image are the top-, bottom-, left- and rightmost pixel coordinates that belong to the object.

2.4.2 DEXTR

Exploring the previously mentioned extreme points as annotator input to a model was done by Maninis et. al. [20]. In their solution DEXTR, they take advantage of the fact that the extreme points lie on the boundary of the object and utilize them as a guiding signal for the network in order to obtain more accurate object segmentation in images.

The additional input to the image contains a heatmap with activations on the extreme point locations. Each point has a 2D Gaussian around its center and the heatmap containing all four points is concatenated to the RGB image as an extra channel.

The extreme clicks were also used to crop the image to fit the bounding box created by the points. In order to include some context to the image, the crop border is then widened.

Architecture of DEXTR

The backbone of DEXTR is ResNet-101 [15], but without the fully connected layers and the max pooling layers in the final two stages of their network. Instead, the authors included atrous convolutions [6] in the last two stages in order to preserve the receptive field. After the last stage of the ResNet-101 backbone, its output feature map is input to a pyramid pooling module in order to make use of the context of the object. These changes are all to the final step of the network, the actual classification.

The network output is a probability map that represents whether a pixel belongs to the object or not, and the network is trained to minimize the class balanced cross entropy loss previously explained in section 2.3. This balanced loss function worked well for them since they used a centered crop of the image which had a higher number of foreground pixels than background pixels.

Use cases for DEXTR

The authors of DEXTR list some of its use cases, among which class-agnostic instance segmentation and annotation pipeline are the ones most promising for the goals in this thesis. Class-agnostic instance segmentation is when the classes of the objects do not matter, and in the case of DEXTR, this generalisation applies to clicking on the extreme points of an object of *any* class. By annotation pipeline, we refer to the workload and annotation time of an annotator. The authors of DEXTR show that the annotation time decreases when using the extreme clicks as annotator input instead of manually drawing the polygon around the object. Although in this thesis we will not measure the actual annotation time for our methods, it is still of importance to consider methods that decrease the annotation time since our problem statement relies around that.

2.5 Interactive improvements to the segmentation

Having a human annotator that produce the data gives new possibilities on how to construct the algorithm to make use of this. To guarantee data of sufficient quality, it is important that the segmentation given by the network is evaluated by the annotator. If it is deemed not good enough, there needs to be a way for the annotator

to tell the algorithm that a better segmentation is needed.

The trivial solution to this is to let the annotator simply mark the pixels deemed to be wrong, and manually change their label. Since the initial segmentation is produced by a machine learning algorithm, perhaps there are ways to, once again, take input from the annotator that is more easily produced. Examples of that are contour clicks or positive and negative clicks, which would be more easy to produce than to manually correct the segmentation. This input is then used with the goal of improving on the segmentation.

Adding input with the goal to improve the segmentation, can be included in the training data. Improving on the segmentation can then in practice be performed by having the annotator provide clicking in any of the forms described below and then do a forward pass of the network with this, further extended, data. This produces a new segmentation, hopefully of a higher quality due to more information being included in the data.

2.5.1 Contour click

Taking further input from the annotator with the intention of improving a segmentation is tested to some extent in DEXTR by Maninis et. al. [20]. They gathered samples where the segmentation was not deemed to be good enough and sampled a click on the contour in the erroneous area. They shift the click location a little in order to simulate user behaviour and then let the network train with the extreme clicks plus the one contour click, with the result of improved accuracy for those samples.

2.5.2 Positive and negative clicks

Another idea is to have the annotator give different input whether or not the segmentation mask should include more or less pixels in the noted area. This idea is examined by Xu et. al. [29] where they label the different input formats as “positive” or “negative” clicks, which is where the semantic segmentation should be extended and decreased respectively.

2.6 Segmentation as rendering

In order to produce the best possible segmentation from the network, there are a lot of different modules and tweaks that can be applied to the ResNet backbone. One such module is the pyramid pooling module mentioned previously. Another such module is the PointRend module, developed by Kirillov et. al. [17], based on the idea of describing segmentation as a rendering problem.

Semantic instance segmentation performed by CNNs typically operate on a regular grid of pixels. The input image is a regular grid of pixels, the hidden representations

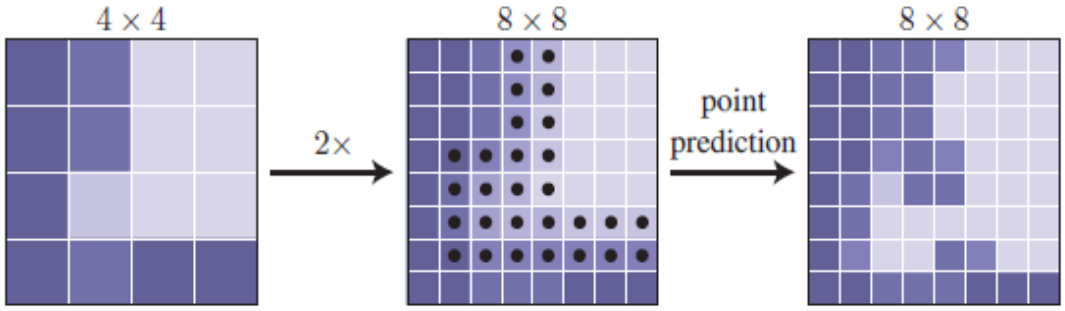

Figure 2.3: Example of one adaptive subsampling step in the PointRend module. Image from [17]. Predictions are only calculated for the marked sub-pixels and kept for all others.

are feature vectors on a regular grid and the output is a prediction map on a regular grid. This is not necessarily computationally efficient. The reason for this is that the output from such a network should, in general, be *smooth*, i.e., neighbouring pixels tend to have the same value. By the nature of performing operations on a regular grid, these networks will *oversample* the smooth areas and *undersample* object boundaries. Simply put, too much computational power will be put into regions of the image where it is not needed, since every pixel not on a boundary will have the same label as its neighbours.

The central idea to solve this problem in PointRend is to approach image segmentation similarly to how the problem of *rendering* has been approached in computer graphics for many years. Rendering is the problem of mapping a continuous model, e.g. a 3D mesh, to a rasterized image, e.g., a regular grid of pixels. A common strategy for solving this task is to compute pixel values at an irregular subset of adaptively selected points. An example of this is the *subdivision* technique described by Whitted [27].

With that in mind, Kirillov et. al. introduce PointRend as a neural network module that uses the subdivision strategy to calculate pixel values at a non-uniform set of points as seen in figure 2.3. A pointrend module consists of three main components:

- A point selection strategy. Avoiding excessive computation by selecting a small number of points for which to make predictions on.
- Extracting a feature representation at each selected point to utilize sub-pixel information by bilinear interpolation.
- A point head, a small neural network computing a label for the generated feature representation of the selected point.

In itself, it is a general module that can take intermediate feature maps of a CNN as input and output predictions per-pixel. In practice this results in an output of a much higher resolution along the object boundaries compared to what would otherwise be achieved by the same order of computational complexity.

2.6.1 PointRend architecture

In the PointRend module developed by FAIR, they build on their existing framework Detectron [28]. It builds on a ResNet-50 backbone, but this acts as a backbone in a Mask R-CNN [14] implementation. Mask R-CNN is one of the most successful network architectures for solving semantic *instance* segmentation. It builds on the architecture described in Faster R-CNN [23]. The main extension to this architecture that is presented by He et al. in Mask R-CNN is to extended the output with an object mask. Faster R-CNN outputs an object label and a bounding box offset, aiming to solve the problem of object detection. It consists of two separate stages: the first stage is called a Region Proposal network (RPN) proposing candidate object bounding boxes. The second step extracts features from each candidate box and performs classification and bounding box regression. The procedure of the second step is practically identical to the process described in Fast R-CNN [12].

Mask R-CNN extends this second stage by, in parallel, calculating an object mask for each candidate box. The candidate boxes and the region they contain is often referred to as region of interest (ROI). Doing this in parallel is the main contribution of Mask R-CNN compared to previous approaches.

This works by defining a combined loss function

$$L = L_{cls} + L_{box} + L_{mask}. \tag{2.10}$$

The classification loss L_{cls} and the bounding-box loss L_{box} are used as defined in [10].
That means that with a ground truth class u and a ground truth bounding box offset target v, the loss is described as

$$L_{cls} = -\log p_u, \tag{2.11}$$

and

$$L_{\text{box}}\left(t^u, v\right) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}\left(t_i^u - v_i\right), \tag{2.12}$$

where we have that

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise .} \end{cases} \tag{2.13}$$

In the equation for the class loss (2.11) we simply have that L_{cls} is equal to the log loss for the true class u.

Equation 2.12 for the bounding box loss is calculated over true bounding box regression targets $u, v = (v_x, v_y, v_w, v_h)$ and the predicted tuple $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$. Once again, with the use of class agnostic segmentation we set the number of classes to 1. The L_{mask} is what is new compared to Faster R-CNN. L_{mask} is of Km^2 dimensionality, which encodes K binary masks of size $m * m$, one for each of the K classes. For an RoI associated with class k only the corresponding binary mask will be considered

when calculating the loss. This has the result that masks can be generated for each class without competition among classes, a *decoupling* of the mask and class prediction. They show in Mask R-CNN that this generates better segmentation results.

The PointRend architecture also contains a feature pyramid network (FPN). Its purpose is similar to what the pyramid pooling module is used for in the DEXTR networks, that is to take the global context into consideration. In short, it does so by making use of feature maps from different levels of the ResNet backbone, also in a similar way to the pyramid pooling module. The important thing to note is that the last module in the PointRend architecture is not the same as in the other architectures presented.

2.7 Evaluation metrics

In this section, a description of the evaluation metrics that are used in this thesis is provided. In order to evaluate the results, Intersection over Union, IoU and *Dice* coefficient are used. We start by explaining the IoU, sometimes called Jaccard index, which measures the overlap between our ground truth and our predicted segmentation divided by the union of them both and ranges between 0 and 1. It is defined as

$$J(A, B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid}, \tag{2.14}$$

where A and B represents the ground truth and the prediction respectively.

The Dice coefficient is two times the overlap between the ground truth and the segmentation, divided by the total number of pixels in them both. It is also used to measure the similarity between two samples, where our samples again are the ground truth and the prediction. Formally, it is defined as

$$DSC = \frac{2 \mid A \cap B \mid}{\mid A \mid + \mid B \mid}, \tag{2.15}$$

where the numerator is the amount of true positives and the denominator contains the amount of true positives, false positives and false negatives.

Both the IOU and the Dice coefficient are set to 1 if the samples completely match, which means that the prediction is one hundred percent correct and 0 if the prediction is one hundred percent wrong. It is also noteworthy that if one model performs better for one of the metrics, it also performs better under the other metric, although they are not equivalent. In general, the IoU penalizes bad classifications more than the Dice coefficient does, even if they both consider the classification to be, in some sense, wrong. This means that the Dice coefficient will resemble the average performance of that model, and the IoU measure will resemble the worst case performance.

3
Methods

To solve the problem of semantic instance segmentation where object detection is performed by the annotator we implement a network architecture very similar to the one presented in DEXTR [20]. We also investigate how to improve on the interactive segmentation by using two types of extensions to the data, contour clicks and positive/negative clicks and how to use information about previous segmentation when choosing the click positions. Furthermore, we analyze the performance of such data combined with a PointRend module.

This chapter first gives insight on the data used for training the networks designed in this thesis. After the data is presented, the network architecture used for the baseline, extreme click model and extended models is presented. With the architecture in mind, we then present our solutions on how to extend the input data, from being simply extreme clicks to now contain even more information. This information is what represents the interactive improvements to the segmentation. After that, we will describe our work to generate more realistic training data for the solutions on extending the input. Lastly we describe the use of the PointRend module to generate segmentations, a separate architecture than the one previously described, even though there are many similarities.

3.1 Dataset

The dataset was provided by Annotell and an example image from the dataset can be seen in figure 3.1. Among the images in the dataset we also make sure to use only the ones that actually contain objects, since we do not segment the surroundings of the objects and thus have no need for images of only background. The dataset consists of road images where the image objects' segmentations are manually annotated. This is done by clicking on the object boundary and sequentially drawing a detailed polygon around the object, where the annotator clicks become the polygon vertices. The result is a segmentation of the object in the image and its vertices are saved as a Multi-Polygon. A Multi-Polygon is basically a list of the polygon vertex coordinates, which could also contain inner lists if some occluding object divides the visible object in several parts. These are saved in GeoJSON format [5]. From the pixel coordinates it is simple to find the extreme points of the object and thereafter simulate user extreme clicks. We simply find the extreme point coordinates corresponding by finding the minimum and the maximum of the x and y coordinates of

the object.

The extreme points are not only used as input to the network, but also as a means of augmenting the data. The full augmentation pipeline overview can be seen in figure 3.2. As the figure displays, the extreme point coordinates will be used to crop the image, which for figure 3.1 results in something like figure 3.3.

Figure 3.1: Example of an image with a truck before it has been cropped. An image can have multiple objects in them but one will be in focus at a time. Each object in an image has a GeoJSON file connected to it, which contains the polygon vertex coordinates of the annotated segmentation of the object.

Ideally, the training data would include extreme clicks made in real time by human annotators in order to capture human behaviour in the input. The data used here does not have that, although it is an annotator that has marked the ground truth coordinates that are used as extreme clicks. It is not feasible to collect the extreme clicks data for this thesis, but for our purposes the already annotated coordinates will suffice.

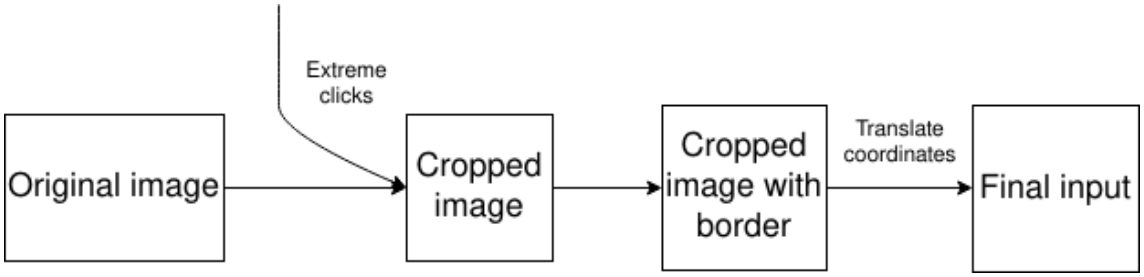

Figure 3.2: The data augmentation pipeline used to transform the original images into the input used for our network training. The border is the pixels we extend the crop with to adjust the ratio between foreground and background.

There can be multiple objects in one image but we will only be interested in training towards predicting one of them at a time. We disregard the possible other objects by not letting the pixels that belong to them have any effects on the loss-value. This can easily be done, since the loss function is calculated for each pixel and then averaged over all pixels. By setting the loss value to 0 for each pixel belonging to another object; the prediction for these pixels will be considered correct which will

Figure 3.3: Example of an image from the dataset, cropped with the help from the extreme coordinates. The image was then extended by some pixels surrounding the object in order to have more background pixels.

minimize their impact on the averaged loss. By that, we do not allow the network to learn anything from pixels of other objects.

Cropping the image further mitigates the problem of other objects in the image, along with the solution of not allowing learning for their pixels. The reason for this is that cropping the images brings focus on specific regions of the image that are currently of interest rather than all of it. When the crop border of the object is found, additional background pixels are added to it. This was inspired partly by the definition of the loss function and partly by the data augmentation of DEXTR [20]. By expanding the image border and including more context, the ratio between the object pixels and the background pixels gets more balanced. The images are also resized to be 256 x 256 pixels before we use them as input in the network, this is important since there is a large difference in the size of the different object depending on how far away they are in the image. Finally, when the image gets cropped, the object coordinates need to be translated correspondingly in order to fit the new scale. After being translated, the coordinates of the Multi-Polygon is used to produce a ground truth mask. Before finally being used as input to the network, the pictures are flipped along the horizontal axis with a probability p_1 and a color jitter transform that randomly changes the brightness, contrast and saturation of an image is applied with a probability p_2 in order to augment the data. During our experiments, $p_1 = p_2 = 0.5$.

3.2 Network architecture

The first network that we created was similar to the network that is presented by Maninis et. al. in DEXTR [20]. It was based on the ResNet backbone as described in chapter 2. The only noticeable difference is that our model was trained with the ResNet-34 backbone instead of the larger ResNet-101 [15], because of constraints on

computational power and time.

This network was used for most of the models that we implemented. We will discuss the details of each model further on. We also used this architecture for the baseline model, which simply takes the images without any of the extensions, crops them around the object and uses the cropped RGB image as input to the network. The baseline model then outputs the same sort of semantic segmentation as the other models.

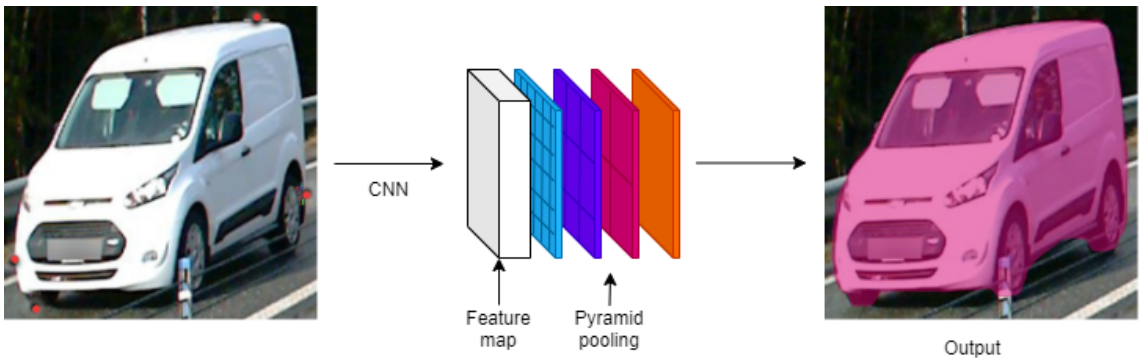

Figure 3.4: Overview of the model idea with this network architecture. The RGB image with concatenated extra clicks channels is input to the networks. The output is a segmentation of the object. Note that the output image is an example segmentation and is not taken from our results.

In figure 3.4 is an overview of the model idea. The input in the figure is an RGB image with a concatenated extra channel containing the simulated extra clicks, although the type and amount of extra clicks varies between the models. The input is then passed to the convolutional neural network (CNN), which for these models is the ResNet-34 network. The CNN outputs a feature map which becomes input to the pyramid pooling module from [32]. As explained in the theory section, the pyramid pooling module uses convolutional filters of different scales to gather features from smaller regions in the image. It then upsamples the features and concatenates them into the final feature representation. This is used by the classifier to predict which pixels belong to the object and which belong to the background. We note that the pyramid pooling module might have more impact if the images would have had even more background around the chosen object and when classifying more categories, but looking at DEXTR [20], it could still be profitable for this model architecture.

3.2.1 ResNet with extreme clicks

In this subsection we present one of our models that is used for comparison. This model is in its core idea similar to DEXTR [20], but differs when it comes to the dataset and the network depth. It uses the network architecture described above and the simulated user input for this model is the extreme points of the object. The reason for implementing a model so similar to existing work is to make a fair

comparison to the other types of user input that we will test (which are described in detail in upcoming sections). Again, this is due to the fact that our other models are trained with a different network depth and a different dataset, so in order to see the impact of the extra clicks, we needed to make sure that the models we compared them to are built in the same way. Then, the only thing that varies is the input type.

Figure 3.5: The extreme click model input. The red points represent the extreme clicks and they are placed on the topmost, leftmost, rightmost and bottom-most pixel coordinates of the object.

As mentioned before, it is a trivial task to find the extreme points from the given ground truth Multi-Polygon that describes the object in focus. When the extreme points are found, the image is cropped based on the values of the extreme points. Each extreme point is then represented as a Gaussian distribution with the point coordinate at its center and added to a fourth channel of the image. The gaussian is set up with the click location as the mean and the standard deviation is set to $\sigma = 10$ as described in [20]. The model input can be seen in figure 3.5 and is an image with four channels, three regular (RGB) channels and one channel with the Gaussian distributions around the extreme clicks.

3.3 Random click policy

In this section we present two models with the extra input clicks where the click positions were sampled randomly, still according to the type. All the models still have the extreme clicks as input. For the models described below, the training data was created in the input transforms by first extracting the coordinates of the object and then sampling points required for the training of the respective models.

The first model that we present is a model that has contour clicks as additional input. When used with this policy, the contour clicks are sampled randomly from the object boundary. As mentioned before, the extreme clicks are still present. The

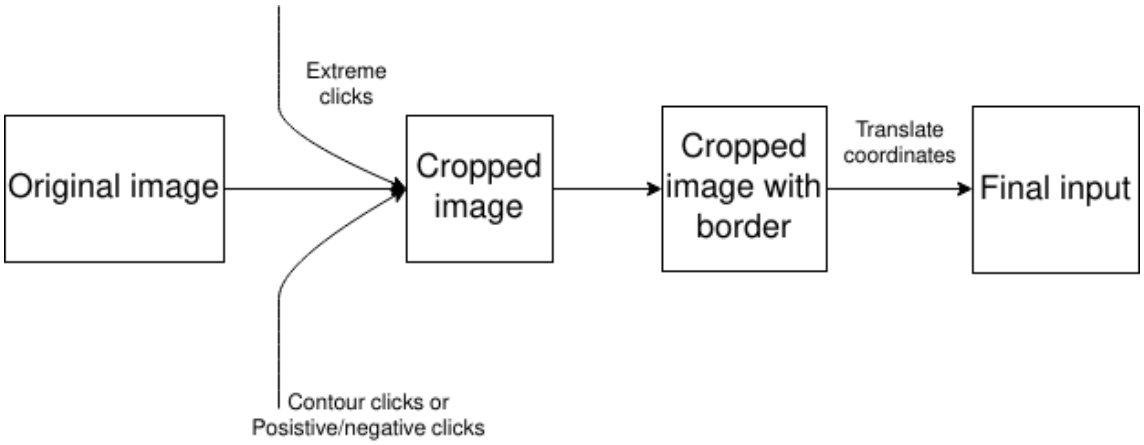

Figure 3.6: The data augmentation pipeline used to transform the original images into the input used for the contour click model and the positive/negative click model. Notice that all training data is constructed before being input into the network. The extended data is constructed by randomly generating contour clicks or positive/negative clicks respectively.

second model that we present in this section is one that has positive and negative clicks. Positive clicks are clicks sampled from the set of object pixels, and negative clicks are sampled from the set of background pixels. When used with our random policy as is described in this section, the clicks are sampled randomly from their respective sets.

3.3.1 Random policy with contour clicks

This model had the same architecture as was mentioned in section 3.2. Like the previous model, it used the simulated extreme clicks as input. In addition to the extreme clicks, we randomly sampled a click from the contour of the object and added it to the extra click channel.

Randomly sampling a click on the contour was the most naive strategy to begin with. Although we believe that an annotator might place the clicks in a more considerate manner on the boundary than randomly, this model adds robustness to our investigation. The simulated contour click was represented in the same way as the extreme clicks, as a Gaussian distribution around its point center with a standard deviation σ set to 10. The contour click and the extreme clicks were then placed in the same channel, so the input to the model was still four channels; the RGB-channels from the image and the extra click channel. The data augmentation pipeline for creating the data for this model was similar to how it was done for the positive/negative clicks model, and the process overview can be seen in figure 3.6.

3.3.2 Random policy with positive/negative clicks

The third model that we implemented had the same backbone as the others with the backbone and the pyramid pooling module, as described in section 3.2. It still

Figure 3.7: The contour click model input. The red clicks represent the extreme clicks and the yellow point is the additional click, placed on the boundary of the object. The contour point is placed randomly on the boundary in this image.

used the simulated extreme clicks in a fourth channel that was concatenated to the image. The differences between this model and the Contour model was in this model the type of the simulated extra user clicks and that the extra clicks were placed in two additional channels. The model therefore has a six channel input, where the RGB-channels and the extreme click channel form the first four. The fifth channel contains positive clicks and the sixth channel contains the negative clicks. As mentioned, this data was created according to figure 3.6.

Figure 3.8: The positive and negative clicks model input. The red clicks represent the extreme clicks. The yellow points represent the negative clicks which are placed outside of the object. The blue points represent the positive clicks, which are placed on pixels that belong to the object. Two positive and two negative points are placed in this image in order to give an example of where the different clicks could be placed.

This model was inspired by [30], where they used Euclidean distance maps to repre-

sent the clicks while we kept with the Gaussian distribution as for the other models in order to evaluate the models in a fair manner. Our version of the model still had two types of clicks for this model which were placed either on the object we want to annotate or on the background pixels, as seen in figure 3.8. The background pixels in this sense include other objects that were still visible after the image was cropped, but those pixels were handled differently, as previously described. The simulated clicks that were placed on object pixels are called *positive* clicks, and the ones places on the background pixels are called *negative* clicks.

The clicks are sampled randomly from the set of object or background pixels, as compared to [30] where they had different techniques for how to sample the negative clicks. The reason for having different sampling techniques for the negative clicks was that a random sampling strategy would be too hard to learn for the network. We decided to keep with the random sampling strategy also for the negative clicks since we crop the image around the object and the space of possible pixel positions therefore is smaller, which means that the meaning of the negative clicks should be easier to learn.

3.4 Smarter click policy

The smarter click policy differs from the random policy in that the extra clicks are placed where the segmentation has previously failed for that sample instead of being placed randomly. The smart policy was implemented in order to investigate if the click positions made a difference for the network. We believe that an annotator may choose to place the clicks carefully, and therefore it is interesting to see if the click position matters. If it does, that would show some promise towards creating training data that makes the networks better suited for improving on its created segmentations.

We implemented two models that uses this click policy that use the same network architecture as described in section 3.2. The first model is the contour click model with this smart policy. These contour clicks are sampled on pixels of the object boundary where the segmentation was previously wrong. The second model is the positive and negative click model. Trained with this policy, the model places positive clicks on object pixels that was previously wrong, and negative clicks on background pixels that previously was incorrectly predicted.

3.4.1 Building the training data

The previously implemented models with ResNet as backbone sampled clicks randomly, but given that an annotator will place a click with more thought we implemented a training policy for the scenario of an annotator clicking where the segmentation was incorrect. The idea behind this was to compare the learning capacity between placing the clicks randomly and placing them where the model had previously been wrong and therefore guide the network more in areas that might be

difficult to predict correctly.

The training was divided into two phases, where the first phase only utilized the extreme clicks as extra input, as seen in figure 3.9. Any other extra channels were empty at this point of the training. The reason for beginning training with only the extreme clicks as input for the first epochs was that the predictions from the first few epochs performed quite poorly and we wanted to sample locations where the erroneous areas in the image were less randomized in order to make a difference to the random click policy. Even though the initial training performance was bad, we still sampled sets of pixels that were wrongly classified for each sample in order to have clicks ready for those samples once we entered phase two.

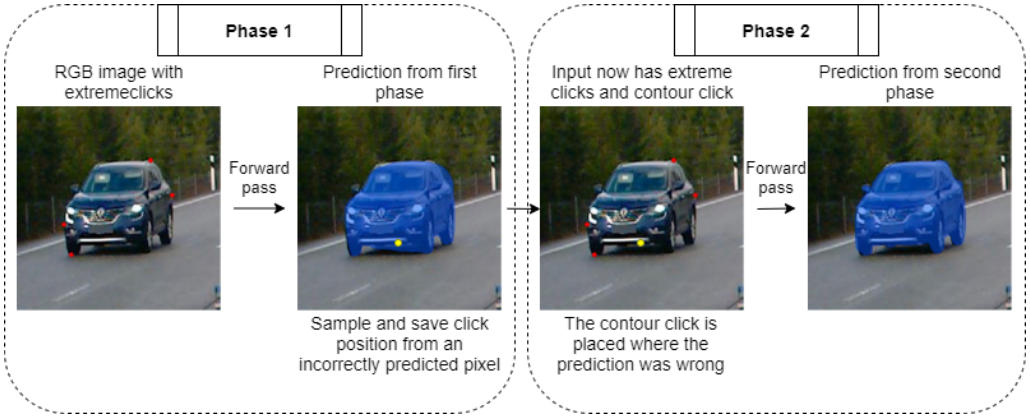

Figure 3.9: Training policy overview for the interactive improvements. In phase one, the input to the network is the RGB image along with extreme clicks in an extra channel as before. The additional clicks, such as contour clicks or positive and negative clicks are then chosen from pixel coordinates where the prediction was wrong. In the image is an example of how a contour click could be chosen. Then, in phase two, the previously saved click(s) are concatenated to the input.

Phase two was initiated in the training when the validation loss was less than 0.6. For most of our experiments, this happened already after the first epoch. In this second phase, the samples had saved click locations where their segmentations had been wrong in an earlier epoch. These click positions are from the erroneous pixels from the previous time the network saw that sample. For samples that for some reason had not yet been seen by the network, a click was randomly sampled, and then the smarter click locations were saved for the next epoch by comparing that sample's generated output segmentation to the ground truth. The data is created as described in figure 3.10.

Throughout the training, the predictions are different for each time the network sees the sample, since the network learns and the segmentation improves. This means that the click locations that are saved between the epoch are different. We will discuss the implications of this in section 5. The sampled clicks were stored in a dictionary, keeping each object as a key and its sampled click positions as the value.

The clicks were then added to an extra channel and concatenated to the RGB image as for the previously described models.

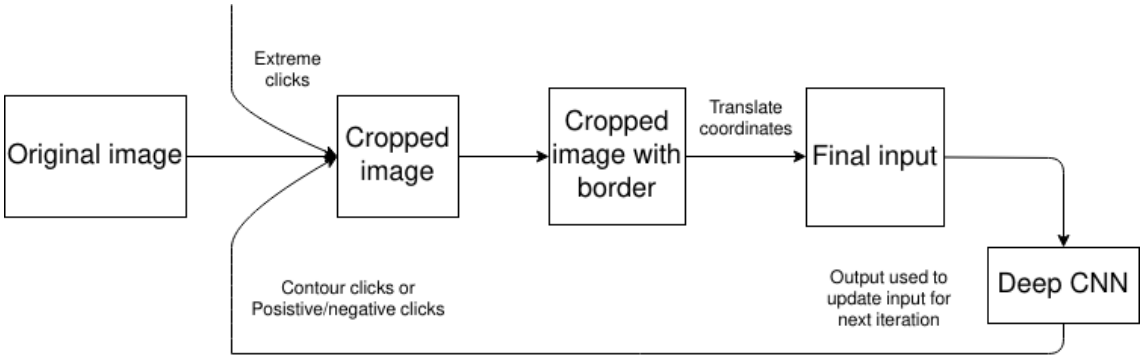

Figure 3.10: The data augmentation pipeline for the smart click policy. Here, we transform the original images into the input used for the contour click model and the positive/negative click model. Notice that the extended input is dependent on the previous output of network and is thus changing for each iteration. The new data is sampled from previously misclassified pixels.

For the validation data samples, the clicks are still sampled randomly before concatenation. This is because when making predictions on the validation set we do not backpropagate, meaning that there are no previous iterations over the same sample from which we can construct the data according to these strategies.

3.4.2 Smart policy with contour clicks

The smarter click policy version of the contour click model stores, as previously mentioned, pixel coordinates from the object contour that has previously been predicted incorrectly. The clicks are stored in a dictionary with one entry (key) for each data sample. The clicks are selected by comparing the values of the ground truth bitmap and the prediction bitmap for the coordinates on the boundary and sampling from the ones that are incorrect. This is done every time the sample is processed, which means that the clicks may have different positions for every epoch, every time the image is seen by the network. The idea behind this is, as mentioned before, to guide the model to the segmenting more difficult areas of the contour. For the validation, the contour click is sampled randomly from the object boundary.

3.4.3 Smart policy with positive/negative clicks

This version of the positive and negative click model stores the sampled clicks in one dictionary for each click type, containing one entry for each object. Finding the positive clicks is done by comparing the ground truth bitmap to the prediction bitmap and thereby find the coordinates for the false negatives in the prediction. The semantics of the positive clicks are that they indicate an area where there should be a segmentation. Once found, they are added to a channel and concatenated to the input. The negative clicks are found by comparing the ground truth and the

prediction as before, but instead collecting the false positives and sampling pixel coordinates from them. These are locations where there should be no segmentation, but where a previous prediction did segment the pixels. The negative clicks coordinates still include both background pixels and other objects' pixels. When some negative clicks are found, they are also added to a channel and concatenated to the input, which means that this model still has six input channels. Thus, the only difference is that the clicks are sampled from points where the segmentation has been wrong, instead of being sampled randomly from the object or background respectively.

3.5 Pointrend

The PointRend module is released as a python package by FAIR (Facebook AI Research) [28]. It is not released as a standalone component but rather as a part of their object detection architecture called Detectron [28]. As such, there are two options on how to examine the impact of a PointRend module on the data extended by a human annotator.

- The first option is to implement the PointRend module as a separate module and to use it for the classification part, i.e. the last layer, of the architectures described above. As such it would then, theoretically, be used instead of the pyramid scene parsing module.
- The other option is to make use of the existing packages in detectron and create a training algorithm and a set of transforms to be able to use our data with that implementation of the PointRend module.[28]

Since the network backbones are similar, the second option seemed to most clearly indicate whether or not the PointRend module would be useful for data extended by a human annotator. Some tweaking to the training setup, such as registering our dataset in the dataset catalog built into the detectron, was needed to allow for images with four channels to be used as input to the detectron with a PointRend module as its classifier and to also make use of the instance detection performed by the annotator.

3.5.1 Variations

For the PointRend tests, we developed four variations of input data in order to test the effect of each element. The variations are an image that has not been cropped, a cropped image, a cropped image with the extreme clicks and a cropped image with both the extreme clicks and a contour click. The reason for the models without simulated user input was that this has not been tested for the architecture that the Pointrend module was tested for. Therefore, we believed that such a test was necessary in order to have a clear evaluation of how the simulated user clicks affected the result.

The variations of the PointRend input in the list below gives an insight on how much the extended data helps the PointRend module in its predictions. Visualizations of the different input variations can be seen in figure 3.11.

Figure 3.11: The variations on the PointRend input. Here, a) is the first input which is the image without any edits. Second was the cropped image, b). The third variation had a cropped image and concatenated extra channel contatining the extreme clicks as seen in c). In the fourth variation, d), a contour click was also added to the extra channel.

- Training the network on the full-size image. The extreme clicks are only used to designate the chosen object but nothing more. In other words, the extreme clicks are used to detect the object in the image.
- Training the network on an image that is cropped with help of the extreme clicks. In this case the extreme clicks are used to crop the image *and* detect the object in the cropped image, giving the algorithm a much smaller part of the image to focus on.
- The "final" version, the one mentioned above, is to use the extreme clicks both to crop the image but also as a fourth input channel in the image. This is identical to the extended input used for the DEXTR model as previously described. Thus the detectron network with the pointrend module is trained on cropped images that are extended with a fourth dimension containing the clicks.
- Lastly we created a model that takes on extra click on the contour, similar to the input described in section 3.3.2. This was to examine if extending the input in such a way would also be beneficial for the PointRend module.

The purpose of this experimental setup was to determine whether or not the extension of the input data that we had previously developed would work well together with a PointRend module. If results showed that the PointRend module performed better with a fourth channel in the input images that would imply great potential for making use of it as a standalone model for other solutions to the machine assisted annotation problem.

4
Results

This chapter presents the results from the experiments conducted in this thesis. The different evaluated models are the extreme click model, the contour click model, the positive/negative clicks model and the PointRend model. The different extended models (contour clicks and positive/negative clicks) will be presented with different amounts of clicks. This is to show whether or not more clicks are beneficial for the segmentation. For the PointRend module, focus lies on whether or not it can improve with the extended training data. In order to understand our experiments and the results that we report, we start by describing the experimental setup.

In this thesis, focus is not on optimizing the results for state-of-the-art performance, but rather to investigate tendencies and solutions for the interactive part of the annotation. Achieving top performance would, by looking at current state-of-the-art solutions, demand a more substantial network architecture, more training data and longer training time. The results below are therefore to be seen as relative to each other rather than for their absolute performance. In order to show the results and analyze the components, we have conducted a study of our models with different settings.

4.1 Experimental setup

Each of the models, except for those with a PointRend module, were trained until they reached a validation holdout that stopped the training when the validation loss did not decrease for five epochs, after a minimum of 40 epochs. This setup of the validation holdout is to prevent overfitting and to handle the varying training times for the models. The learning rate was lowered during training for the random click policy, for every 40th epoch it was multiplied by 0.10. For the smarter click policy, decreasing the learning rate was deemed unnecessary since the training was usually finished after about 40 epochs. Most importantly, no models are trained for a substantially longer time, enabling comparisons between the extreme click model, contour click models and positive/negative click models.

For the PointRend networks, 25 epochs was enough to train on this amount of data since they got high accuracy much faster than the other models. One reason for this is probably the pretraining, which will be discussed in the next chapter. This might complicate comparisons between the networks with PointRend modules and the contour click and positive/negative click models due to different training setup.

There are however substantial differences in the architectures that already complicate such comparisons. As previously mentioned, the purpose is to investigate the tendencies of the different models and for the PointRend module, the goal is to show whether or not it benefits from the extended input. Thus it is not a problem that comparisons between them are not straightforward, as long as we keep it in mind when analyzing the results.

We report results for the different models in terms of mean Intersection over Union (IoU), and mean Dice coefficient from the predictions on a test dataset, calculated as previously described in the theory section. These metrics are used since the main interest is to have an algorithm that improves on the annotation speed while maintaining the accuracy of the segmentation. Since measuring the annotation times is outside of the scope of this thesis it is a natural choice to evaluate the algorithms based on the segmentation accuracy. The IoU and Dice scores reported in this chapter generally comes from two separate training runs, three for some models.

4.1.1 Input data

All models were trained on 4000 images sampled from sequences, with images without cars being removed which brings it down to 3573 images. Out of these, 20% of the images were used in a validation dataset. The test dataset is taken from the end of a sequence in order to make sure that the samples are different and it is almost as large as the training dataset. The images are cropped around the object for most models, and then downsampled to size 256 x 256 before being input to the network. Apart from the image input, all models but the baseline model takes the extreme clicks as input in order to crop the image around the object unless otherwise specified.

4.1.2 Baseline model

As it was previously described in section 3, we implemented a Baseline model in order to be able to analyze the results compared to it. The difference between the baseline and the other models was that this model only took the RGB-images as input, and not any simulated user input clicks. Input images were still cropped from the extreme point positions in order to give a fair examination of the extreme point impact, but they were not put in a channel and concatenated to the input for this model.

4.1.3 Extreme clicks model

Before investigating the results for the different click policies, the model with only extreme clicks is evaluated. As previously described, the input to this model is the image as well as four extreme clicks that are concatenated as a fourth channel of the image. This is exactly the same data extensions that is examined in the DEXTR paper, but in order for us to evaluate properly, we implemented en extreme click model in a similar way as the other models. As reported in DEXTR, the results

Table 4.1: Table showing overview test results. The comparison is between the different models for the random click policy and smart click policy. Furthermore, the comparison is done between the Baseline model, the extreme click model, the contour click model and the positive and negative click model (PosNeg). We see that the models with additional input perform better than the baseline model and the extreme click model for both policies.

Overviewing results	Random policy		Smart policy	
Model	IoU (%)	Dice (%)	IoU (%)	Dice (%)
Baseline	70.43 ± 0.55	74.18 ± 0.60	70.43 ± 0.55	74.18 ± 0.60
Extreme clicks	73.45 ± 0.62	76.08 ± 0.51	73.45 ± 0.62	76.08 ± 0.51
Contour (1 click)	74.46 ± 0.17	76.69 ± 0.10	74.45 ± 0.33	76.77 ± 0.27
PosNeg (1 of each click)	74.63 ± 0.13	76.94 ± 0.20	74.97 ± 0.34	77.20 ± 0.36

Table 4.2: Table showing test results for the Contour click model. The models listed in the table were trained with the same amount of clicks that they were tested for. The results are not conclusively showing an increase in performance as more clicks are used.

Contour clicks	Random policy		Smart policy	
Model	IoU (%)	Dice (%)	IoU (%)	Dice (%)
1 click	74.46 ± 0.17	76.69 ± 0.10	74.45 ± 0.33	76.77 ± 0.27
2 clicks	74.34 ± 0.19	76.63 ± 0.22	74.44 ± 0.56	76.70 ± 0.55
3 clicks	74.59 ± 0.36	76.86 ± 0.33	74.30 ± 0.16	76.66 ± 0.22
4 clicks	74.43 ± 0.25	76.71 ± 0.19	74.42 ± 0.27	76.72 ± 0.21
5 clicks	74.46 ± 0.06	76.68 ± 0.04	74.56 ± 0.17	76.79 ± 0.15

of the experiments show that the segmentations are better when we use extreme clicks compared to the baseline model. Results can be seen in table 4.1 with the extreme click results reported together with results from the baseline and other models.

Overall, the models with simulated user input perform better than the baseline model. Adding the extreme points gives better segmentations and adding the extra contour or positive and negative clicks gives an additional increase to the performance compared to using only the extreme clicks.

4.2 Results for random click policy

In this section we show the results for the models trained with the random click policy, where the click positions were randomly chosen. These models used ResNet-34 as backbone with the pyramid pooling module on top and simulated user input for the training data was added in the data transforms for all of these models. The images were cropped around the object for all models evaluated with the random click policy.

Table 4.3: Table showing the test results for the positive and negative click model. The comparison is between the random and smart policies and between one and five extra clicks. The results show an increase for the random policy but a decrease for the smart policy.

Positive/negative clicks	Random policy		Smart policy	
Model	IoU (%)	Dice (%)	IoU (%)	Dice (%)
1 click (of each)	74.63 ± 0.13	76.94 ± 0.20	74.97 ± 0.34	77.20 ± 0.36
5 clicks (of each)	74.71 ± 0.10	76.99 ± 0.11	74.65 ± 0.09	76.90 ± 0.15

In table 4.2 is an overview of the results for the random click policy for the contour click model. The same data for the positive/negative model can be seen in figure 4.3. It is clear for both models that more clicks in general should result in a better segmentation, even though some of the intermediate steps do not report an increase. Our results are not conclusively showing this increase from adding more clicks. This will be discussed more in depth in the next chapter.

4.2.1 Contour clicks

In order to evaluate how much one single click affects the model and if the model could benefit from receiving multiple clicks, we designed a test comparing contour models with varying amount of clicks.

We chose to perform the tests of one up to five extra contour clicks since there should be a tendency of receiving better results with more clicks. Hypothetically, if an annotator would place an infinite amount of clicks on the object contour, it would be the same as drawing the detailed contour by hand. Therein lies the reasoning behind this test, that more clicks should guide the network better. Since adding only one click shows a difference, the effect of more clicks should be visible already when increasing the number of contour clicks from one to five. Furthermore, a solution where an annotator has to click more than four extreme clicks and an additional five contour clicks is getting closer to the workload of manual annotation and we want to minimize the needed user interaction. Therefore, we settled with five extra clicks as the maximum in our tests.

In table 4.2 we have the overall results from the different contour click models trained with the random click policy. Each model listed in the table was trained according to the experimental setup mentioned previously. The models are tested with the same amount of extra clicks that they were trained with in order to grant fairness in the evaluation.

Table 4.2 shows that there is no distinct tendency of increased accuracy when adding more contour clicks, which is not what was expected. Since one contour click increased the accuracy compared to the extreme click, additional clicks were expected to keep increasing the segmentation accuracy. The differences are not very large and

Figure 4.1: Semantic segmentation and extended image input for the network trained on four contour clicks from the random policy. The clicks are in regions of the image that would be difficult for the network. Many clicks are also located around the rearview mirrors, which could be beneficial since it is a tricky part to segment. The resulting segmentation is rather good.

it is not a linear relation between the amount of clicks and the increase in accuracy. We note that three and four contour clicks gave best performance in our tests, although the deviations of the other models makes the results difficult to conclude.

The random click policy sometimes generates clicks at very beneficial locations on the contour. Hypothetically, such locations are far away from the already existing extreme clicks as the contour clicks will then provide more information on the actual contour. An example of this can be seen in figure 4.1 where we note that the clicks are very well placed to generate a good segmentation for the rearview mirrors, which is a part of a car that networks often struggle with. A visualisation of this struggle can be seen in figure 4.2 where the initial extreme click is the only click placed on the right rearview mirror and the resulting segmentation is not very good for that part of the contour. Compare this to the segmentation in figure 4.3 where the contour is much better and all of the clicks seem to be a part of the segmentation.

In figure 4.4 is an example of two interesting things. First off, the contour clicks are spread out quite evenly along the border of the object, which is as previously mentioned most likely good for the network since it gets more information about the boundary. The other thing to note is the antenna, a tricky shape for the network due to it being so thin. One extreme click is placed on the top of the antenna as it is the top part of the car, and the segmentation reflects that by going above the roof of the car. It is however too wide, and does not at all resemble the actual shape of the antenna.

Another example of the random click policy resulting in very little additional information can be seen in figure 4.5a. The contour click on the right wheel is difficult to notice since it is so close to one of the extreme clicks. Most other positions for the contour click would probably give the network more information and so this is a clear example of the random policy not being ideal. The resulting semantic segmen-

Figure 4.2: Semantic segmentation and extended image input for the network trained on four contour clicks from the random policy. The right rearview mirror is of particular importance, with only one click in the input and a bad contour of the segmentation. The segmentation as a whole is not particularly good.

Figure 4.3: The extended input and predicted segmentation produced by the network trained on two clicks with the random policy. Notice that the segmentation is of high quality and that all of the contour clicks seem to be a part of the segmentation such that the network has learnt to classify them correctly.

Figure 4.4: Output segmentation from the random click policy with three contour clicks. The input (left) shows a relatively clear image of a car that has an antenna at the top, at which an extreme click is placed. The resulting segmentation (right) is quite accurate in comparison to the ground truth (middle), but has a hard time with the antenna. We note that the extreme click guides the segmentation to stretch upwards, but the model can not predict its exact shape.

tation can be seen in figure 4.5b where the segmentation is not perfect. It is hard to tell from the image if the contour click has provided any helpful information for the network. We note that the rightmost extreme click is annotated on what supposedly is the right rearview mirror. It is however hard even for the human eye to deduce that there actually is a rearview mirror there. This is an example of the problems of segmenting objects with low resolution due to the distance to the camera.

4.2.2 Positive/Negative clicks

In order to evaluate how the amount of positive and negative clicks affect the model, we conducted a test with one positive and one negative click as input and another test with five positive and five negative clicks as input. By definition, one of these clicks contains less information than one on the contour. This is due to the points on the contour not having the same labels as all their neighbours, which is more likely the case for the locations chosen for the positive/negative clicks. As such, our main interest was to investigate if more positive/negative clicks could guide the network better than less clicks. This is why there are no results for the intermediate number of positive/negative clicks. The small intermediate differences for the contour click also merits not studying each and every case, but to instead get a grasp of the bigger picture.

In table 4.3 are the results of the two models using the random click policy as well as the smarter policy, whose results we will discuss further down in the report. We see that the segmentation improves slightly by using more positive/negative clicks, but since the deviations are as big as the differences between the models, it is difficult to decide whether five clicks improve the segmentation. Since we received better results when adding positive and negative clicks compared to only having the extreme clicks, this was an unexpected result.

Worth noting is that we report the amount of each type of click, thus there are twice

(a) The input image with extra clicks.

(b) Semantic segmentation output.

Figure 4.5: The input, clicks and image, to the network trained on one contour click with the random policy. This is an example of the downside of the random click policy since the contour click is placed almost exactly on one of the extreme clicks. The semantic segmentation output in 4.5b resulted from the input in 4.5a is overall not quite good enough and it is hard to tell if the contour click has had any impact and if it has guided the network at all.

as many inputs as for the corresponding rows in the contour click table. This means that there is twice as much input for an annotator to produce in a hypothetical application. It could perhaps explain why the one click results are better for positive/negative clicks than for contour clicks, since that model has trained on twice as many clicks as the contour model ($2 * 1 = 2$).

In figure 4.6 is an example of the click locations being far away from the object boundary. The positive clicks are all pretty well inside the object while the negative clicks are much closer to it, but still not close to the contour. We note one gap in the segmentation of the top of the truck, but it is not placed where the negative click that is very close to the contour is. These areas of the object is something that this network architecture typically has had no problem segmenting when only using extreme clicks as input, and thus it seems that the positive and negative clicks do not provide much extra information to the network in that case. This exemplifies the problems of the random policy placing the clicks in irrelevant areas. This is especially a problem for large objects, since the random policy has more pixels to choose from for the positive clicks i. e. the contour is a smaller part of the whole object. This does, on the other hand, increase the probability of the negative clicks being placed closer to the border. The resulting segmentation is still rather good, but without any noticeable impact from the positive and negative clicks.

An example of the random policy where positive clicks are close to the object border can be seen in figure 4.7. The negative clicks, on the other hand, are scattered but

Figure 4.6: Locations of the negative clicks, the positive clicks and the resulting segmentation for the network trained on the random policy. The positive and negative clicks are all far way from the contour and the resulting contour is not perfect in the details.

Figure 4.7: Locations of the negative clicks, the positive clicks and the resulting segmentation for the network trained on the random policy. The positive clicks are close to the object boundary while the negative are far away. This is often the case for small objects. The segmentation is better on the bottom of the car, where there are more positive clicks.

nowhere near the contour. The segmentation is once again quite good, but it seems very unlikely that especially the negative clicks have had any impact on the segmentation. There are however some substantial problems classifying the top parts of the car, where the segmentation stretches outside of the object. We note that there are no positive clicks at that part of the car. We discuss this further in the section 5.

4.3 Results for the smarter click policy

In this section we report the results and show some segmentations from the smarter click policy. The section is further divided into contour clicks and positive and negative clicks where their respective results are displayed. The smarter policy was our strategy of constructing input data that mimics the imagined behaviour of a human annotator. This behaviour was to click in an area where the segmentation was wrong, with the intention to correct that area. Noteworthy is however that our implementation of this policy still segments the entire object with this new input data, and not just the area where the click was placed. The implications of this solution will be discussed in chapter 5.

We start off by analyzing the results of the models using the smarter click policy that collects one click position where segmentation was previously incorrect. The only models that were affected by this policy are the contour clicks and the positive and negative clicks, which is why the baseline and extreme clicks results are the same as in the random policy. The tests were done in order to compare the random click positions to instead placing the clicks where the model have had difficulties predicting correctly. The extreme clicks are still present in every data sample in both the contour click model and the positive and negative click model.

In table 4.1, in the rightmost columns, we see that adding extra clicks according to the smart policy provides a better segmentation accuracy than having only the extreme clicks as input. It also appears that the smart policy gives a slight *increase* in performance when compared to the random policy for the PosNeg model with one extra click, but a slight *decrease* for the contour click model. With regards to that, one last thing to notice in the table is that the PosNeg model performs better than the contour click model for this test.

4.3.1 Contour clicks

In this section we compare the amount of contour clicks for the smarter policy in order to see if a difference in the amount of clicks makes a difference for this policy. The motivation behind this test is the same as for the contour model test with the random policy, and to make comparisons between the policies.

We see the results of the contour clicks test according to the smart policy in table 4.2, in the rightmost columns. Noticeable is that although the accuracy for the smart policy does not seem to increase clearly when adding even more clicks, we

(a) The input image with extra clicks.

(b) Semantic segmentation output.

Figure 4.8: The click input 4.8a to the network trained with the smarter click policy on one contour click. Note that the contour click is still very close to one of the extreme clicks. This input made the network produce the output in figure 4.8b. We see that the object boundary is not good and that the shadow under the car specifically is partly labelled as car.

see that the deviation between runs is decreasing for the smart policy model with 5 clicks. There is also an increase in the segmentation quality from three to five clicks. Combined with the large deviation for the models for one and two clicks it is not unreasonable to think that more clicks does indeed improve the segmentation but that some of our training runs for one and two clicks randomly performed better than expected.

In figure 4.8 is an example of when the smarter click policy with one contour click does not perform a segmentation that is good enough. The input that generated this segmentation can be seen in figure 4.8a. We note that the contour click is placed very close to one of the extreme clicks which is a sign that the previous segmentation of this image was not correct at that location. We also see that both the upper left extreme click and the contour click is outside of the segmentation. This tells us that the network has not learnt that the clicks should always be part of the segmentation.

In figure 4.9 is a sample with one contour click that the network managed to predict a much better semantic segmentation for, as seen in 4.9b. The clicks are evenly spread out, which as mentioned before could be a reason for the better segmentation. We see that for this image, there seems to be no problems with the shadow underneath the car. We note in figure 4.9a that the contour click is placed on the bottom of the object border, inside the wheel. This could have helped the network to separate the object from its shadow.

One example of the segmentation becoming somewhat more accurate can be seen in

(a) The input image with extra clicks.

(b) Semantic segmentation output.

Figure 4.9: An example of a good segmentation from the contour click model with one click. The click and image input 4.9a has evenly spread out clicks, of which there is a contour click underneath the car. We see that the segmentation in 4.9b manages to correctly predict the shadows underneath the car.

(a) The input image with extra clicks.

(b) Semantic segmentation output.

Figure 4.10: Click and image input with which the network produced the semantic segmentation in 4.10b. Semantic segmentation produced by the network trained with the smart click policy and three contour clicks. The overall contour of the object is a quite good fit.

Figure 4.11: The ground truth semantic mask, where other objects are marked in grey, compared to the predicted segmentation from the network trained on the smart click policy with 3 contour clicks. Note that occlusion seems to be a big problem since large parts of the occluding vehicle is also segmented. As can be seen in this ground truth image, the annotation is not overlapping, so the pixels do only belong to the current object.

Figure 4.12: The extended input, ground truth segmentation and predicted segmentation respectively. This was produced by the network trained on four contour clicks and the smarter click policy. It is clear that occlusion is a problem, especially when a large part of the object in focus is covered. It is also noticeable that the lower parts of the occluding object is not at all segmented.

Figure 4.13: The extended input and predicted segmentation for the network trained on the smarter policy with five contour clicks. It is noticeable how the amount of extra clicks somewhat tells the shape of the object and highly interesting is that the only part where the segmentation is a bit off is where there are no extreme clicks, the left wheel in the picture.

figure 4.10. This network was trained with three additional contour clicks, and as can be seen in the input in figure 4.10a, the click positions were evenly spread out. The resulting segmentation can be seen in 4.10b, and the most interesting thing to note here is that all of the extreme clicks seem to be correctly segmented and the overall quality of the segmentation is quite high. The image resolution is quite good and the sample is one of the easier types of objects which also impacts the segmentation.

The same network that produced the good segmentation in figure 4.10 has issues when it comes to occlusion, as can be seen in figure 4.11. The white part in the left figure is the object that the network is trying to segment. The resulting segmentation to the right in the figure covers the roof of the occluding car in the foreground. We note that the left side of the object is not segmented correctly either, even though an extreme click should have marked the left boundary of the car.

Another example of the problems with occluding objects is seen in figure 4.12 where the model is trained on four contour clicks. The occluding trailer is covering a very large part of the car that tows it, and the four contour clicks does not make up for this. Instead almost all pixels belonging to the occluding object are segmented. It is however noticeable that the extreme point coordinates has a lot of impact on the network since the lower part of the occluding object is not segmented at all, due to its placement below the lowest point of the object in focus. This tells us that the network makes decisions based on the extreme click locations. We also note that the contour clicks are positioned on the outside of the object, when an annotator might have placed them on the border between the objects to mark the object shape more precisely.

We conclude the contour clicks results by looking at figure 4.13 where five contour clicks have been used with the smarter policy. These five clicks combined with the

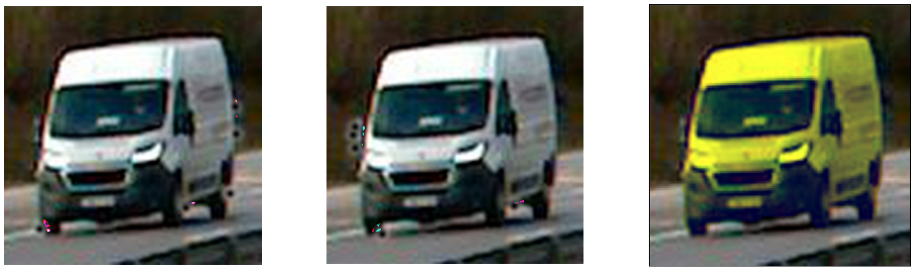

Figure 4.14: Negative clicks on the left, positive clicks in the middle and segmentation on the right. Input created with the smarter policy. There is almost no possibility of seeing the difference between the negative and positive clicks. The resulting segmentation is rather good.

four extreme clicks give the human eye an intuition of the shape of the car. It is especially interesting to note that the segmentation of the car is a bit off on the outside of the left wheel in the picture, which is the only part of the car where there are no contour clicks. The same can be said about the pixels in the front of the car, where the segmentation stretches outside of the object boundary and there is no click positioned at that area.

4.3.2 Positive/Negative Clicks

The setup for the positive/negative click model is similar for the smarter click policy and for the random policy. The difference is that the positive clicks will be sampled from previously misclassified pixels of the object and the negative clicks are sampled in the same way, but from the background pixels. This means that as the segmentation gets better from the training, there will be less points to sample from and the positive and negative clicks will be closer to the object contour. This is because when the network has trained for a while, and is rather good at segmenting, the most uncertain points will generally be in that area of the image.

In table 4.3 we notice that the positive/negative clicks show improvements when using the smart policy compared to the random policy for the model with only one of each click. Quite surprisingly, the smart policy with five positive and five negative clicks instead had *lower* segmentation accuracy when compared to the random policy model with five clicks of each.

In figure 4.14 we see that the clicks are chosen exclusively on the contour of the object. This is at the end of the training where the previous segmentation was already rather good. Compare this to the clicks seen in figure 4.15, which is taken from the start of the training. There is a big difference in the click locations for these two images, which shows that the patterns of the click locations changes as the network learns. This is expected since the clicks will be drawn towards the more difficult parts of the object, which are exclusively located on the object boundary.

We see in image 4.15 that the bus is also stretching all the way to the image bound-

Figure 4.15: Negative clicks on the left, positive clicks in the middle and segmentation on the right. Input created with the smarter policy, in the first iteration. The negative clicks are a bit outside of the contour, and the positive are not at all on it. The segmentation completely misses all of the positive clicks.

ary. In most of the data samples, we have some background surrounding the image, so the network seems to have difficulties segmenting that part. This makes the location of the clicks more important, but it still appears that they do not guide the network well enough.

In figure 4.14, the segmentation is much better, but as said the network has had more training to that point. We clearly see changes on the click locations and that is primarily the reason for the smarter click policy performing better than the random policy. However, it seems that even though four of the positive clicks are on the rearview mirror to the left, the segmentation is not completely correct there. This means that the the segmentation was wrong for that part also in the previous iteration, which is why the clicks ended up there. It also shows that the network has not yet learned to segment exactly according to the positive clicks. Perhaps future iterations with the same click locations would be better and could perfectly segment the mirror.

Another interesting thing to note in figure 4.15 is that the segmentation completely misses all of the positive clicks. This is however only in the first epoch of training and thus it is not unexpected that the segmentation is still rather bad at this point. This also shows that that part of the image was not segmented in the previous epoch either, since the positive clicks are sampled from that region of the image. Thus we can, in this image, also see how little the segmentation improves between two epochs giving further merit to training the networks for a long period of time.

One final example of the smart click policy for positive/negative clicks can be seen in figure 4.16. The segmentation is quite good, but once again it is hard to see the positive and negative clicks having any immediate impact on the segmentation. We note that the click locations are very close to the contour of the object which is expected from the smarter policy at this point in the training. The actual improvements of using the smart policy is however hard to deduce since the results are worse for five clicks than for one. The results for one click is better than with the random policy, while the opposite holds for five clicks.

Figure 4.16: Negative clicks on the left, positive clicks in the middle and segmentation on the right. Input created with the smarter policy, with five clicks of each type. Both the positive and negative clicks are once again very close to the contour as we near the end of the training.

4.4 PointRend

Since the PointRend model is different in its architecture as well as training setup compared to the other models, we compare results between different variations of the PointRend model by itself in order to make a just evaluation. The results from the tests of the PointRend module is seen in table 4.4. Once again it is important to note that comparing these numbers directly with the ones presented for previous models is a bit misguiding due to the differences in the training setup and the network architecture. It is however still interesting to see that this network benefits a lot from cropping the image and from the extra input. The implications of the network differences will be discussed more in the next chapter.

Table 4.4: Table showing test results of the different variations of the PointRend model. The performed tests are of different types of input.

PointRend model	IoU (%)	Dice (%)
RGB not cropped	69.57 ± 0.25	80.12 ± 0.05
RGB cropped	89.95 ± 0.09	94.28 ± 0.09
with extreme clicks	90.66 ± 0.26	94.74 ± 0.18
with extreme clicks and 1 contour click	91.01 ± 0.05	94.95 ± 0.01

The main interest here is to make comparisons between the results for the different inputs to the PointRend module. Four different tests were conducted in order to test the separate properties of each model. The first model had the RGB image as input without cropping it around the object. The second model had a cropped RGB image as input, butt only three channels still. The third model had cropped images and the extreme clicks channel concatenated to the image. The fourth model had cropped images and then an extra channel containing the extreme clicks and one contour click. The contour click was placed randomly on the object boundary.

Noticeable in table 4.4 is that the model where we do not crop the images, the only model where that is not done, performs much worse than the others. To our

Figure 4.17: PointRend segmentation example with the extreme clicks as input. The PointRend module is not at all successful in its predictions for this image. It is a quite hard image to classify correctly, but we can clearly see that it is nowhere near correct.

knowledge, cropping the images in this way has not been tried for the PointRend module previously and our results show that the detectron setup of PointRend benefits greatly from it.

The next improvement is between extending the image with the extreme clicks as a fourth channel instead of only using them to crop the image. Our results show that the PointRend module benefits from extending the input data with another channel, something that has not been tried before. Finally, extending the input with a randomly placed contour click also increases performance a bit. This shows some promise towards using the extensions to the input data developed in this thesis in combination with the PointRend module. Due to time constraints, we did not test the smart policy for this model, and not the positive and negative clicks either. We will discuss this further in the discussion chapter.

When looking at figure 4.17, we can see an example of the PointRend architecture that has the extreme clicks as its only extra input. Even though this is a tricky image to segment, the PointRend module is supposed to be good at focusing on interesting parts of the image such as the object boundary. We expected it to be able to somewhat handle the poles in the vehicle since the PointRend module utilizes a higher resolution along the contour in order to make a better segmentation. Predicting difficult contours is where the PointRend module excels in the paper where it is introduced and as such this bad segmentation was a bit unexpected. It seems that the network gives a lot of importance to the ground truth bounding box and the extreme points and has a hard time when so much of the area inside the bounding box is classified as background. As said, this is a very tricky image, but still one where the PointRend module was expected to be better than what is seen in this figure.

Figure 4.18: PointRend segmentation example with the extreme clicks as input. The PointRend module is rather successful in its predictions for this object. Especially noteworthy is the precision along the underside of the car even though its shadow is of almost the exact same color.

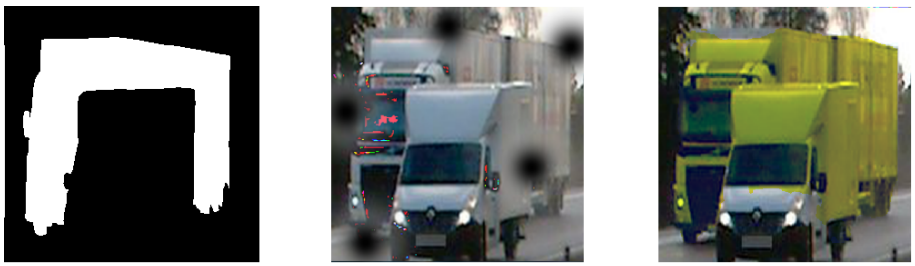

Figure 4.19: A bad semantic segmentation by the PointRend network with contour clicks. This object is heavily occluded and even though the contour click is on the boundary between the object in focus and the occluding object it is not enough to succesfully segment the object in focus. The network has not learned that the contour click is a delimitation of the object. The clicks being in the alpha channel of the image is what leaves some strange colors.

Opposite of that, we note the segmentation in figure 4.18. We see that the segmentation is very good, and especially that the network is able to distinguish between the shadow beneath the car and the dark parts on its bottom. This problem is however somewhat easier when the extreme clicks is provided as the object fills out a lot more of the area of the ground truth bounding box.

For the PointRend model with extreme clicks and contour clicks as input, we note that the IoU score of the segmentation is a bit higher. However, for the example in figure 4.19 it is clear that the contour click does not always steer the segmentation to predict a boundary where the click is placed. This network appears to have problems with occluding objects, especially when a large part of the area in the ground truth bounding box is not actually part of the object. Since most of the data sampled are not occluded, the network may have difficulties separating the objects.

Figure 4.20: A bad semantic segmentation by the PointRend network with contour clicks. We can see that parts of the occluding object is also segmented, with the ground truth bitmap to the left and the predicted segmentation to the right. We also see from the click placement that the contour click is very close to one of the extreme clicks and as such does not provide a lot of new information. The clicks being in the alpha channel of the image is what leaves some strange colors.

Figure 4.21: A comparison of some of the best segmentations produced by the PointRend module and the contour click model.

Another example of the problem with occlusion and also the image resolution can be seen in figure 4.20, where it is clear from the ground truth image to the left that something is occluding the truck. The clicks are unfortunately placed in other locations in the image and does therefore not guide the segmentation in the bottom left area of the truck. As a result, the segmentation of the truck covers the occluded area too. It is worth noting that the image resolution makes the features belonging to each object quite difficult to distinguish even for a human.

As a final concluding element of the PointRend results, we provide an image comparing one of the better segmentation results of PointRend to one of the better results from the contour click model with four contour clicks in figure 4.21. The PointRend model produced the segmentation to the left in the image, and the right image was produced by the contour clicks model. Although the images are different, we choose to compare them since some characteristics of these two models still are displayed in the images. Both the models segment the shape of the car well and manage to distinguish the object from its shadows underneath. One thing to notice

is that the PointRend solution manages to segment the shape of the rearview mirror more precisely than the contour model. Although capturing the body of the car, the contour clicks model is not always accurate along to boundary of the object, as can be seen underneath the car.

5

Discussion

This chapter starts with a discussion of the results presented in the previous chapter and their implications. We discuss our choice of methods and their impact on the results of the thesis. Followed by that is a discussion of our overall results and the conclusions drawn from them. After that we will present individual discussions for each of the models we have setup before we go into depth of the implications of some of the design choices made for this thesis. The discussion of the models are divided into Contour clicks, Positive and Negative clicks and PointRend and in these sections we discuss the solutions respectively. Lastly we present the potential ethical implications of the work in this thesis.

5.1 Methodology

Our methods were chosen due to their possibilities of showing the impact of many sorts of data extensions for the task of machine assisted annotation. A lot of thought went into the representation of the data extensions, with the main question being how to make use of the extra input. We settled on the solution with contour click and positive and negative click represented as Gaussian distributions partly because that is what was used in the DEXTR paper and partly because it seemed to be the most intuitive solution. Other forms of creating this input could probably be used for a network to only improve on the segmentation instead of creating a new one with each forward pass.

Another very important part of the chosen method is how to present the constructed segmentation to the annotator. We chose to simply output the segmentation as created and to train for the interactive improvements by using more clicks on that segmentation. Another, very different idea, is to make use of the segmentation to produce a polygon around the object. If that is done, the annotator can make the corrections by only dragging the vertices of this polygon. We considered constructing such a solution, inspired by the algorithm by Acuna et. al. [1]. Their patent on the network architecture used for that solution was one of the things that made us focus on a different sort of representation for the interactive improvements instead of producing a polygon around the segmentation for the annotator to correct.

5.2 Overall results

Our results show that extending the input data is better than not doing so, in all tests conducted. The model trained with the extreme clicks as an input channel outperforms the baseline model, which substantiates the potential of machine assisted annotation since such simple forms of user input can improve the segmentation. This is further emphasized by the contour click models and positive/negative click models producing even better segmentation than with only the extreme clicks.

It is hard to tell which solution is best for a couple of different reasons. Our results show that additional contour clicks or positive/negative clicks improves the segmentation results. This leads to a straightforward conclusion that it is recommended to use one of the click types. From the tests in this thesis it would seem that the positive and negative clicks are somewhat better than the contour clicks. One possible reason for this is that there are simply more clicks since one of each of the positive and negative clicks is two clicks, while one contour clicks is just that one click. As all clicks are modelled in the same way in the data, more clicks means more information.

With that said, the results show very little difference in the segmentation precision between the two click types, so it is hard to tell if one of them is better than the other. Especially since our results do not show any clear increase in the segmentation quality for either model when using more clicks, at least not so that we can confidently say that more clicks increase the quality. Our results are simply not certain enough for us to say that contour clicks are better than positive/negative clicks or vice versa even though the scores obtained by the positive and negative clicks are a bit better.

One factor to also consider is the amount of work that each click type would demand from an annotator, where the contour clicks demand one interaction and the positive and negative clicks demand two. Perhaps one of the input forms is also less intuitive for the annotator to provide. Although a small difference, it could be significant in a larger annotation project and would therefore need further research in order to conclude which click type is more efficient.

It is worth noting that our setup of the training and creation of the training data does not take the temporal difference in the input forms into consideration. What we mean by this is that a network as designed in this thesis will always use either only extreme clicks or extreme clicks plus the extended data. In an imagined application for machine assisted annotation, the *interactive improvements* will not be created by the annotator at the same time as the extreme clicks. The annotator would first input the extreme clicks to extend the image. Then if the annotator is not satisfied with the segmentation, more input would be provided. It is possible that by taking the temporal difference into consideration, the algorithm could be designed in another way. One idea that we had but did not have time to explore is presented in section 5.8.

An algorithm where the annotator provides input and receives segmentation in iterations could hypothetically generate other results than showed here. Those results could possibly make it easier to decide which one of the click policies is best for such an application. This reasoning builds on the human notion that correcting something is not the same thing as redoing it from scratch. It is not obvious that a neural network could learn to correct segmentations based on this notion. The networks designed in this thesis would not operate in such a way in an application, but instead they would redo the segmentation for every time that new input is provided. Correcting only the immediate surroundings of a click requires well defined rules, like how big the corrected area should be. It is therefore not obvious how such a solution should be designed. We however believe that it would be interesting to do some further research of a solution designed to correct existing segmentations instead of simply constructing new ones.

One final thing that we believe to be of high importance when designing an application for this machine assisted annotation flow, is the interaction between the human and the algorithm. Perhaps one of these input forms for interactive improvements is more intuitive for the annotator than the other. If so, it would most likely lead to the best results on the actual annotation time which, as mentioned, is a core metric for such an application. We would recommend further studies into both click types to be able to fully decide which one of these data extensions are best.

In conclusion; the results of this thesis confirm that the extreme clicks provides an improvement in segmentation accuracy and that the extra clicks improve it further. Our thesis does *not* show any significant difference between the different extra click input formats (the contour clicks and the positive and negative clicks) and as such we believe them to be equally suited for being used by an annotator based on the segmentation quality achieved. However, as discussed here, there are many other things than segmentation quality to take into consideration before implementing either solution in an application.

5.3 Contour clicks

In chapter 4 we noticed that there was a small increase in performance between five contour clicks and one contour click for the smart policy and none for the random policy. For the difference between using one contour click and only extreme clicks we found that there was an improvement that was a bit larger than between different amounts of contour clicks. This is in line with the findings of DEXTR[20], where they saw an impact from the additional contour click. The improvement observed by them is however more substantial than what our tests show, although the contour click was added only to the most difficult objects in their study. A probable reason for the larger impact is their use of a deeper architecture as well as more data and longer training times.

Even though our results do not show conclusively that more contour clicks generate a better segmentation it is still our hypothesis that this is the case. Since we see

improvements compared to using only the extreme clicks, we know that the network is able to learn *something* from having one contour click. That makes us believe, even though it does not show in our results, that the network should be able to learn even more from more clicks.

In order to strengthen this belief, consider placing the maximum possible amount of clicks on the contour of an object. That would be the same thing as drawing the object contour just like what is done when manually annotating. If the whole contour is in the input data, it has complete information about the object that we want to segment. Since more clicks imply that we are closer to the maximum amount of clicks, the intuition is that the segmentation should be better. This leads us to believe that the training setup is what makes the results inconclusive, rather than the network not benefiting from more contour clicks.

To test this hypothesis further, we tried training a contour model with 100 contour clicks, to get a notion of how much the network can actually learn from the clicks. While most contours are longer than 100 pixels this example clearly contains a lof of information about the object. The IoU score was 75.88% which is an improvement on the other contour click models but not very significant. This gives an insight into what improvements that we can reasonably expect. If such an amount of input only results in that little improvement, perhaps it is a better idea to explore other architectures to make more use of the interactive improvements.

When comparing the results for the different contour click policies, we did not see any clear tendencies. Similar to what we just described regarding the amount of clicks. A possible reason for the rather small differences between the policies could be that if the contour is largely misclassified, the selection strategy of the contour point is pretty much the same as the random policy. This means that at the start of the training, the strategy does not differ a lot from choosing at random. Further on in the training, when more of the contour is correctly classified, the selection strategy will be less similar to choosing at random. This similarity of click positions between the two policies can also be an explanation for the segmentation precision being rather alike for both of them.

Another reason for the small impact of constructing the data according to the smart policy could be the fact that doing another, completely different, forward pass of the network neglects a lot of the previous information. When training the network in this way, we do not make use of the previous segmentation for anything else than choosing positions for the simulated clicks for the training. This means that a lot of information is lost and that could be a reason for the small differences on the segmentation IoU for the policies. As mentioned before, a very interesting thing to examine would be to construct a machine learning algorithm that only improves on existing segmentation instead of creating new ones.

It could also be the case that constructing the training data according to the smart policy has no particular impact on the decision making of any sort of networks, since

it does not seem to impact our models more than the random policy. The smart policy was an interesting thing to investigate since we believe that this is a likely behaviour for an annotator. If the segmentations in our results do not cover the click centers, their precise location might not matter. In that case, a random policy could be enough even in a final application and a human annotator might not need to put that much thought in where to place the click.

All in all, the test conducted in this thesis do not show any clear difference in using the random click policy and the smarter click policy for the contour clicks. The differences in the table are a bit too small and irregular for us to be able to say anything definitely.

5.4 Positive/negative clicks

Adding extra positive and negative clicks to the input in addition to the initial extreme click input gave better performance for the random policy and the smarter policy, as seen in table 4.1. We do however only observe an increase in the segmentation quality as we increase the amount of clicks for the random policy and not for the smart policy. The result for the random policy was expected since previous work received similar results and shows that it seems useful to guide the network with this type of extra click. With that in mind, not observing the same thing for the smart policy is surprising. As the different policies contradict each other we can't conclusively say anything about the impact of using more positive and negative clicks.

In the test that compared the effect of a varying amount of positive and negative clicks as input we saw in table 4.3 that for the random policy, more clicks gave a slight increase in accuracy. The opposite happened for the smarter policy, which is quite hard to explain. Since the differences in the results are very small between the policies, a theory is that more clicks are actually better for the smart policy also, but that the training runs made for this thesis randomly ended up with worse results. Since there is some variance in the exact training performance, this is possible. To solve this problem, the simple solution is to do more training runs. Of course, the possibility that the network is not able to learn anything at all from the increased amount of contour clicks can not be fully discarded. But since there is a clear improvement when using one positive/negative click compared to only the extreme click it does seem that the network can learn something from at least that one click. And since that is the case, our conclusion is that more clicks should lead to a better segmentation which is why we believe that the fault lies in the training setup.

The results that we expected was that the smarter policy would outperform the random policy quite substantially. This hypothesis build on the fact that it is less likely that the randomly sampled positive/negative clicks are sampled from an interesting part of the image, while the smart policy will choose locations where the segmentation was wrong. As the network gets better, these locations will eventually

be only difficult areas.

A possible reason for the outcome is that as the segmentation becomes better, the variance in click positions decreases for the smart policy. In the images, the contour is the hardest area to classify. This means that the sampled positive and negative clicks will be closer to the actual contour when the predictions become better. Since the segmentation learns the majority of the object pixels rather quickly, both the positive and negative clicks may end up close to the contour already in the beginning of the training. With this in mind, the network may have difficulties learning the meaning of the respective clicks and therefore the positive and negative clicks makes no difference to the network, but they are all considered the same way as contour clicks.

Observations show that the rate at which the model reaches such good performance that the clicks are placed along the boundary of the object is faster for more clicks. For the model with only one extra click, it runs longer before the clicks are placed tightly to the boundary. This observation gives merit to the argument that more clicks gives the network less time to learn the implications of each click type which could explain more clicks being worse for the smart policy.

As the positive and negative clicks for the smart policy will imitate the contour clicks as the networks gets better, it would imply that contour clicks are superior to positive/negative clicks. Our results show that this is however not the case, and one possible explanation for this is that it could be beneficial to have different types of information in different input channels. For the results that we have, the values are changing a bit too irregularly for us to be able to say that one of the policies is definitely better than the other. It is however clear that the smarter policy leads to the positive and negative clicks being much closer to the object boundary.

5.5 PointRend

The PointRend model showed promising results when it comes to adding additional input to the network, see table 4.4. Not surprisingly, cropping the image around the object improved the result. A reason for this could be that distinguishing the object from the background already before inputting the image to the network simplifies the problem. Also, when not cropping the image, the background pixels outweigh the object pixels by a large magnitude which complicates the problem for the network.

What makes the PointRend results interesting is that adding the initial simulated user extreme clicks to the image improves the performance more than just using the extreme coordinates to crop the image. This suggests that the PointRend model also benefits from the user guidance in the form of extreme points. The performance appears to be further increasing when an extra contour click is added, although that click is randomly sampled on the contour and an annotator probably would want to

click where the segmentation was incorrect.

These results indicate that there is a lot of promise in using a PointRend module for the classification step of machine assisted annotation networks. The PointRend module seems to be able to perform classification based on the extended data in a similar way to what was done with the DEXTR inspired network. In that architecture a ResNet backbone with a pyramid pooling module for classification was used to learn from this data. Our results give merit to the hypothesis that using a PointRend module as the classifier for such a ResNet backbone could generate even better semantic segmentations. Even though our results can not necessarily be compared straight off due to differences in the architecture, we show that the PointRend can make use of the information contained in the extended data and that it performs really well. A future study comparing the exact gains of a similar network with a PointRend module and a pyramid pooling module could be highly interesting.

When looking at the results from the different models of this thesis, it is clear that the PointRend model reaches a better absolute performance than the other models. This might be due to the fact that the PointRend model was pretrained for problems that likely are more difficult to learn than ours, since our data is cropped and contains only a few types of objects. The PointRend backbone also contains a deeper network than the one used in the other models of this thesis, which could affect the result even further.

One interesting thing about our use of the PointRend module is that it achieves very low training loss and good IoU scores on the training set even after the first epoch. This is interesting since the other architecture we used predicted bad segmentations to begin with. This tells us that the setup of the pretraining for the PointRend module in Detectron is perhaps better than the pretraining based on the DEXTR architecture. The possible reasons for this will be discussed more in the next section.

5.6 Pretraining

When using the ResNet architecture that is pretrained with weights loaded from the model developed in DEXTR we note that even though we make use of transfer learning, the initial accuracy is bad. We believe that this is because one of three things:

- One possibility is that the differences in the training data used in DEXTR compared to the data used by us are so large that the weights from the DEXTR model do not help at all in making predictions on our data. This seems a bit unlikely since the DEXTR model is trained on data with objects of many different classes. As such, the feature maps it learns should theoretically be able to do some generalization on our data, which it does not. Especially since the objects present in our data are also present in the training data for DEXTR, such as cars and trucks. However, transferring weights from a network trained on other classes is not necessarily improving the initial classification at all, which is what is seen in our training as well.

- The second possibility is that the transfer learning has a very low impact because we are not using the same exact ResNet architecture. Since we are using a more shallow model, we can only load the weights of the layers that are present both in the DEXTR model and in our model. It might be that these weights are only of use in that exact architecture and that they do not really help with the initial classifications of our model after transferring them. This seems more likely to be the case, that the weights transferred simply needs to be in a deeper context for them to make an improvement on the classification task at hand. It is also not really known that transfer learning from different network architecture improves the results at all, and most likely it was a bad hypothesis that it would work at all.
- The third possibility worth considering is that we have altered the input data by for example adding input channels. DEXTR also trained with the extreme clicks in an extra channel, so if the pretraining worked bad for us because of the changed input, it should have shown a difference for the extreme click model. No such difference was seen, which makes it less likely to be the sole cause for the pretrainng not working correctly, although we can not exclude the possibility that the input has affected it.

5.7 Class agnostic segmentation

In Annotell's tool for annotating images, the annotator draws a shape around an object and denotes which class it belongs to. This gives us the possibility of disregarding the class of the object and train our networks for class agnostic segmentation. We believe this to generally be a simpler problem to solve, since there are fewer classes to make the predictions on. There might, however be some complications to the training due to this. The main complication with using class agnostic segmentation is that things that do not look particularly similar are supposed to be of the same class, i. e. *object*. This means that the feature maps will need to "classify" very different features as all belong to this class. Perhaps this makes the prediction task somewhat more difficult.

An interesting approach on how to solve this problem in an application would be to have a separate network for each class. The annotator could then, before supplying the extreme clicks, denote which network to input the data to by marking the class of the object to be segmented. The problem with this approach is the amount of training data needed for each class. Ideally, each network should only be trained on objects of the class it is to do segmentation for. It also increases the storage used, since a whole network would be needed for each class.

One idea of solving the problems discussed in this thesis is to simply let the network also do the classification and let the user guide the network to perform semantic segmentation and classification. We do however believe that such a solution does not fully utilize the human annotator. Object classification for these sorts of images is quite simple for humans and it is hard to see algorithms outperform humans on data similar to what has been used for this thesis. The aim of this thesis has been

to investigate solutions where humans can be aided by algorithms and not replaced by them. Of course, there is a possibility that semantic instance segmentation algorithms will one days be so good that human annotators are not needed anymore. In conclusion, we believe that machine assisted annotation works best if approached as a class agnostic problem, simply because it is easy for a human annotator by detect what class an object belongs to.

5.8 Two networks for interactive improvement

As for the smarter click policy that was divided into phases, we note that the effect of taking advantage of the previously incorrect segmentation could perhaps be magnified by letting the network see the prediction that generated the click position. Regarding the interactive parts of correcting the segmentation, one could think that this type of solution could benefit from seeing what previously went wrong.

An idea that arose but was not implemented due to time constraints, is to divide the model into two networks where one larger network handles the RGB image with initial clicks as input and then outputs a segmentation. In an end-to-end manner, a second network could then take that segmentation, the RGB image and initial clicks, along with an additional click placed where the segmentation was wrong. This type of solution opens up for the possibility of perhaps learning to correct only the subregion where the additional click was placed. Regarding the user experience of a solution like this, since it is still humans that will interact with it, it is preferable if a forward pass is fast and you as an annotator get to see the resulting segmentation from your extra click as fast as possible. If the interactive corrections are then only iterated by the second, smaller network, that forward pass might be faster.

5.9 Evaluation

In order to fairly evaluate our techniques for creating training data, their performance would need to be measured compared to the performance on data produced by human annotators. As mentioned, no training data with annotated extreme clicks is available for this thesis and it is outside of its scope to construct a system for generating such data as well as have it annotated. Because of this, no conclusions can be drawn regarding the similarity of performance to a human annotator. It is still possible to compare the training data of the smart policy, that is created to mimic an annotator, against training data produced randomly. We hope that further studies can more thoroughly evaluate the gains of constructing training data in this way and whether or not it improves the performance of a network that is to take input data from a human annotator.

5.10 Ethical aspects

The ethical implications of the techniques developed in this thesis are in themselves rather small. The work is performed in a field where ethical implication can theoretically be extremely significant, that is when we consider both computer vision and autonomous vehicles. Our work is not to develop autonomous vehicles but to help with the creation of training data for them. Even though we certainly acknowledge the ethical complications in autonomous vehicle development we do not believe that this specific work has any implications on them. For interesting discussion on these topics we recommend "The social dilemma of autonomous vehicles" by Bonnefon, Shariff and Rahwan [4] as well as "The moral machine experiment" by Awad et. al. [3] amongst many other articles available online.

One possible problem is of course if the annotations become incorrect due to insufficient quality checks of the data. If an autonomous vehicle is then trained with wrongly annotated data, this could lead to serious safety implications. In order to avoid this situation, we believe that the very high quality requirements are important, no matter how the annotated data is constructed.

Another, possible, ethical implication of this work is the problem of human workers being replaced by algorithms. In this case, that problem would consider annotators being replaced by algorithms similar to the ones discussed here. We however emphasize that this thesis in particular discuss techniques that can not work in the way described here without human annotators. As such, these techniques are only meant to increase the speed of the work of the annotators. There are, of course, also possibilities of developing algorithm for annotation completely without humans but that is out of scope for this thesis.

6
Conclusion

The purpose of this thesis was to investigate different ways of incorporating user input in a neural network, with the intention of improving the process of annotating image training data for autonomous vehicles. The idea was to let an annotator provide initial input to the network and in return receive a segmentation. Given this suggestion, the annotator could then choose to provide additional input to the network in order to improve the segmentation. This could let the network segment simpler parts of the data and let the annotator focus on the more difficult problems, and hopefully decrease the annotation time. In order to answer the research questions of this thesis and investigate the possible impact of machine assisted annotation, a few different models were implemented and evaluated with simulated user input.

Previous work in the field encouraged the usage of extreme clicks instead of bounding boxes to annotate the object in an image. Therefore we utilized the extreme coordinates of an object and concatenated them to the image as an extra channel containing simulated user clicks as the initial input form to guide the network. This also helped the network to detect the object and perform semantic segmentation instead of semantic instance segmentation, since the annotator takes care of the object detection. Adding this simulated extreme clicks input to the network showed an increase in performance for the models, compared to only cropping the image around the object.

In order to investigate ways of improving the segmentation even more, two different types of simulated user interactions were implemented onto the previously mentioned extreme click model. The first of these models carried additional clicks on the boundary of the object in order to further guide the network. The second technique instead carried additional positive and negative clicks, simulating a user wanting to extend or decrease the segmentation respectively. This extra input was concatenated to the initial input to the network. Both of these input types gave an increase in the segmentation accuracy.

Since we imagine that an annotator would be likely to place a click where the segmentation had previously failed, we decided to examine if a smarter positioning of the extra clicks could affect the model more than simply placing them randomly in accordance with the technique, i.e. a contour click randomly on the boundary. This showed no particular effect for the contour clicks, which lead to the conclusion that the placement of the contour click either does not matter or that its effect does

not show in the experiments that we conducted. The smarter click policy showed a slight performance increase for the positive and negative clicks containing only one click, but a decrease when trained and tested with five clicks. A reason for the results could be that performance improvements for the smart policy model lead to changes in the position patterns for the click types and therefore could make it harder for the network to learn.

In an effort to boost the performance and try to confirm the results of the user guided segmentation, we also tried the extreme click and contour techniques on a module called PointRend. The extreme click input gave an increase to the performance when compared to not inputting any clicks, and the randomly placed extra contour click gave even more of a performance increase. We showed that the PointRend module can also make use of the extensions to the input data. This indicates that the extra input could have a lot of use in other network architectures used for interactive annotation.

Concluding the findings of this thesis, we see and confirm that providing user input in the form of extreme clicks gives the network useful guidance and simplifies the problem since the object detection part is handled. The results also indicate that additional input in the form of clicks either on the boundary or in the form of positive and negative clicks increase the segmentation accuracy of the models. These improvements were also showed for a model never tested with user inputs before, which lead to the conclusion that there might be a lot to gain from machine assisted annotation.

6.1 Future work

There seems to be quite some time until stand-alone semantic instance segmentation algorithms can match the quality accomplished by human annotators. Training data for autonomous vehicles is reliant on vast amounts of such data and the expensive process of creating it would benefit from any possible speed-up. We show that there are possible ways of interactively improving the quality of the segmentation, although there are many things left to examine in regard to this.

Our results do not show any specific improvements when using the smarter policy. It might however be that creating this training data in another way is the best way to go. Possibly, contour clicks and positive/negative clicks can be very impactful but our idea on how to construct the training maybe was not good enough. Another possible policy would be to sample the contour clicks to have them as spread out as possible and thus provide as much information about the object as possible. A similar idea for the positive and negative clicks is to choose the pixel that is misclassified and also has the most misclassififed neighbours. As such the training data would focus even more on the interesting areas of the image.

An interesting extension to this work would be to manipulate the network structure to make better use of input added in two separate iterations, so that there is an

initial input producing the first segmentation and then an extra input to correct it. An interesting hypothesis is that there is a network structure, or a combination of structures, that can make better use of the input used for correcting the initial segmentation, as discussed in section 5.8.

We also believe that further improvements to networks performing semantic (instance) segmentation will easily be applied to this field as well. As new architectures become even better at doing semantic segmentation without a human-in-the-loop, this same architectures can most likely use data extended in the ways examined in this thesis to get even better results. An argument for this being the case is that the very new PointRend module's segmentation improved with these data extensions. These findings can be evaluated more in depth regarding the possibilities of integrating PointRend modules in networks used for machine assisted semantic instance segmentation.

Many other works in machine assisted segmentation output a polygon around the final segmentation to allow for easier corrections by the annotator. This is examined by Acuna et. al. [1] as well as Ling et. al. [19]. Comparisons between having the annotator correct the segmentation with the help of polygons in this way and correcting by extending the input with clicks, as discussed in this thesis, would be interesting.

We have provided no clear consensus as to which of these methods are best regarding time spent per image, a core metric in these tasks for many business purposes. We believe that interactive annotation solutions need to be studied in detail with regards to the annotation time. This is important both to understand the business implication and speed-up achieved by using the algorithm but also to understand the interaction between the annotator and the algorithm.

Bibliography

[1] David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 859–868, 2018.

[2] Anurag Arnab and Philip HS Torr. Bottom-up instance segmentation using deep higher-order crfs. *arXiv preprint arXiv:1609.02583*, 2016.

[3] Edmond Awad, Sohan Dsouza, Richard Kim, Jonathan Schulz, Joseph Henrich, Azim Shariff, Jean-François Bonnefon, and Iyad Rahwan. The moral machine experiment. *Nature*, 563(7729):59–64, 2018.

[4] Jean-François Bonnefon, Azim Shariff, and Iyad Rahwan. The social dilemma of autonomous vehicles. *Science*, 352(6293):1573–1576, 2016.

[5] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub, and Christopher Schmidt. The geojson format specification. *Rapport technique*, 67, 2008.

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[7] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[10] Ross Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[12] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

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

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[17] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. *arXiv preprint arXiv:1912.08193*, 2019.

[18] Justin Liang, Namdar Homayounfar, Wei-Chiu Ma, Yuwen Xiong, Rui Hu, and Raquel Urtasun. Polytransform: Deep polygon transformer for instance segmentation. *arXiv preprint arXiv:1912.02801*, 2019.

[19] Huan Ling, Jun Gao, Amlan Kar, Wenzheng Chen, and Sanja Fidler. Fast interactive object annotation with curve-gcn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5257–5266, 2019.

[20] Kevis-Kokitsi Maninis, Sergi Caelles, Jordi Pont-Tuset, and Luc Van Gool. Deep extreme cut: From extreme points to object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 616–625, 2018.

[21] Dim P Papadopoulos, Jasper RR Uijlings, Frank Keller, and Vittorio Ferrari. Extreme clicking for efficient object annotation. In *Proceedings of the IEEE international conference on computer vision*, pages 4930–4939, 2017.

[22] Lorien Y Pratt, Jack Mostow, Candace A Kamm, and Ace A Kamm. Direct transfer of learned information among neural networks. In *AAAI*, volume 91, pages 584–589, 1991.

[23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[24] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[25] Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron Courville, and Yoshua Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*, 2015.

[26] Zian Wang, David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Object instance annotation with deep extreme level set evolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7500–7508, 2019.

[27] Turner Whitted. An improved illumination model for shaded display. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, page 14, 1979.

[28] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[29] Ning Xu, Brian Price, Scott Cohen, Jimei Yang, and Thomas S Huang. Deep interactive object selection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 373–381, 2016.

[30] Ning Xu, Brian Price, Scott Cohen, Jimei Yang, and Thomas S Huang. Deep interactive object selection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 373–381, 2016.

[31] Yuhui Yuan, Xilin Chen, and Jingdong Wang. Object-contextual representations for semantic segmentation. *arXiv preprint arXiv:1909.11065*, 2019.

[32] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.