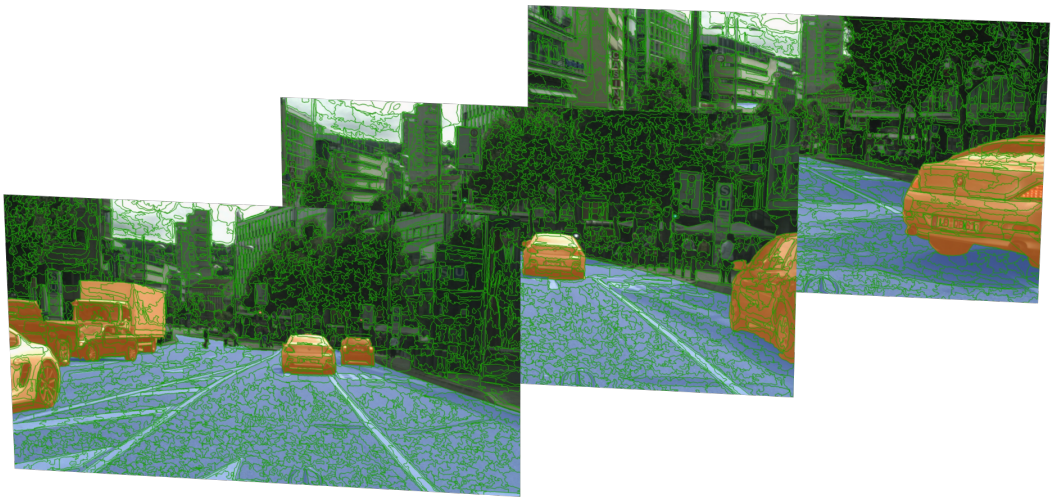

# Annotation of Image Sequences Using Superpixels

Generating and linking superpixels between images for annotation purposes

Master's thesis in Systems, Control and Mechatronics

PHILIP ANDERBERG

FANNY LIESÉN GULLMANDER

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

Master's thesis 2020

# Annotation of Image Sequences Using Superpixels

Generating and linking superpixels between images for annotation purposes

PHILIP ANDERBERG
FANNY LIESÉN GULLMANDER


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
*Division of Signal Processing and Biomedical Engineering*
Chalmers University of Technology
Gothenburg, Sweden 2020

Annotation of Image Sequences Using Superpixels

Generating and linking superpixels between images for annotation purposes

PHILIP ANDERBERG
FANNY LIESÉN GULLMANDER



Supervisor/Examiner: Torsten Sattler, Department of Electrical Engineering

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

Cover: An image sequence from the Cityscapes dataset [1] annotated using the proposed annotation tool.

Typeset in LaTeX
Gothenburg, Sweden 2020

Annotation of Image Sequences Using Superpixels
Generating and linking superpixels between images for annotation purposes

PHILIP ANDERBERG
FANNY LIESÉN GULLMANDER

Department of Electrical Engineering
Chalmers University of Technology

# Abstract

A main challenge within the development of autonomous vehicles is to ensure operability in different conditions. This is commonly solved by using more labeled image data, to obtain a more general model. Availability of annotated data is however limited and expensive across the entire field of computer vision. As such it would be beneficial to streamline the annotation process to achieve a higher throughput and lower price per annotation. This thesis presents a new approach for annotating images using superpixels in combination with a method for propagating annotations forward through sequences, giving the user suggested annotations to be corrected.

A variety of structures for a Superpixel Sampling Network (SSN) were investigated and evaluated, concluding that U-Net was the best network architecture. The U-Net SSN was then connected with a proprietary propagation technique utilizing optical flow and morphological operations. Lastly, the usefulness of the method was evaluated by implementing the full pipeline with an annotation tool and running a time study on four users. The study showed that the propagation did not contribute to a better solution, it was however shown that superpixels can reduce the annotation time with a trade off in annotation accuracy. With a correction tool enabled the method also allowed for annotations with the same quality as the baseline, although without any reductions in time. As such we believe that the proposed method could be beneficial given further development of the annotation tool.

Keywords: Superpixel segmentation, Covolutional neural networks, Video mask propagation, Optical flow, Color matching, Annotation.

# Acknowledgements

We would like to thank our adviser and examiner Associate Professor Torsten Sattler of the Electrical Engineering institution at Chalmers University of Technology for his guidance and helpful feedback throughout the project. Our supervisors Oskar Karnblad and Saudin Botonjic for their support in all aspects of the project as well as their continued encouragement. Associate Professor Peter Almström of the Department of Materials and Manufacturing Technology at Chalmers University of Technology, without whom the time study would have been challenging. The annotation team at Volvo Cars for their participation in evaluation of the methods developed in this project. Lastly we would like to thank Volvo Car Corporation for providing resources and giving us the opportunity to carry out this project.

Philip Anderberg, Fanny Liesén Gullmander, Gothenburg, May 2020

# Contents

# List of Figures

# List of Tables

# 1
# Introduction

In recent years the pace of work on autonomous vehicles has been picking up. One main challenge for future autonomous vehicles is to ensure operability in different driving environments such as urban or rural areas as well as varying weather conditions. This is something that Volvo Cars want to solve and the main way of tackling this problem when using machine learning for visual scene understanding is by using more data containing varied scenes and conditions. This allows for a more general solution, providing more stability in different environments. However, data needs to be annotated manually which is expensive and time consuming. Additionally, the high cost of annotations also holds back the field of image analysis as it limits the amount of available data.

Semantic segmented annotations are particularly time consuming as the annotations are pixel-wise. In the works [1], [2] it is stated that precise segmented annotations for one image takes on average 90 minutes for an experienced annotator. Therefore, it would be highly beneficial to streamline the annotation process such that it requires less human interaction and offers a higher throughput.

An approach to reduce complexity and save time in operations with high amounts of image data is to use superpixel segmentations [3]. This is a technique for segmenting images into groups of pixels with semantic significance, see Figure 1.1. One approach for streamlining the annotation process might therefore be to pre-segment images using superpixel segmentation.

**Figure 1.1:** Superpixels generated by the untrained superpixel sampling networks, SSN-pix [4], on an image from the Cityscapes dataset [1].

Further, an important property of many data streams is that the images to be annotated are collected in a sequential stream. Thus, it would be desirable to use the sequencing of images to propagate annotations and segmentations between images.

This thesis strive to achieve a solution for annotating data faster by using superpixel segmentations to speed up the annotation process of an initial image. The intention is to also use propagation techniques for transferring or linking superpixels and annotations to future frames, providing the annotator with a suggested annotation that only needs to be corrected. To the best of our knowledge, previous works focuses exclusively on the superpixel problem [4]–[8] or video mask propagation [9]–[15], as opposed to this work which will focus on merging superpixel segmentation and video mask propagation.

## 1.1 Objective

The thesis aims to provide a pipeline for generating and transferring segmentations between images in a sequence. To do this, an existing superpixel segmentation algorithm and a self-developed propagation method based on modern techniques needs to be implemented. The superpixel segmentation is required to provide superpixels with state-of-the-art performance. Further, the approach for propagation should be able to transfer masks with good precision such that the accuracy of the segmentations is constant. To solve this problem in a satisfactory way the thesis intends to answer the following questions:

- How can superpixel segmentations be implemented to support and mesh with the annotation process of images?
- How can modern superpixel techniques be used in collaboration with video mask propagation to pass on semantic information in image sequences?
- Which implementation of video mask propagation would prove to be most useful in the annotation process of sequential images?

## 1.2 Contributions

The thesis presents a pipeline for generating and propagating superpixels between images in a sequence. In the part of superpixel segmentation, different base network structures for superpixel sampling networks have been implemented and evaluated. For transferring superpixels, optical flow and color matching have been incorporated in a proprietary technique also utilizing noise reduction with neighbour superpixel matching and morphological operations. Additionally, an annotation tool that incorporates superpixels and propagation has been developed.

## 1.3 Related Work

This section present works related to the tasks of generating and propagating superpixel in annotation purposes. What was found during literature studies is that

previous work mainly focuses exclusively on the superpixel problem and video mask propagation. Thus, these subjects are reviewed separately. Furthermore, a literature review on annotation of image data is also presented since this is a fundamental area for the thesis.

### 1.3.1 Superpixel Segmentation

Superpixels was first introduced as an oversegmentation pre-processing stage, by Ren *et al.* [3]. Superpixels are a group of connected pixels which Ren *et al.* state should be local, coherent, and capture structures that are required for the level of interest. The reasons for grouping pixels were that pixels by themselves does not hold much information and that the number of pixels in an image is high which makes the optimization problem hard to solve [3]. Since then, superpixels have been a building block in various fields including image segmentation [4], [6]–[8], object detection [16], [17] and image parsing [18]. During the recent years, properties for good superpixel algorithms have been specified further and as such commonly desired properties are: they should generate superpixels that adhere to natural boundaries, they should be efficient and the generated superpixels should be similar in size and shape [19].

One of the most prominent algorithms for generating superpixels is Simple Linear Iterative Clustering (SLIC) [5]. Proposed by Achanta *et al.*, SLIC is an algorithm in which the superpixel generation is seen as a $k$-means clustering problem. As a distance measure SLIC uses the CEILab colorspace [20] in combination with $x$ and $y$ pixel coordinates. Further, to speed up the search process when assigning pixels to cluster centers, the algorithm uses a fixed search region. However, due to this it is possible for pixels to be assigned to clusters of which they are not connected. SLIC solves this by running an algorithm which enforces connectivity of superpixels after all pixels have been assigned to superpixels.

Based on SLIC, Achanta *et al.* [6] improves the algorithm and propose Simple Non-Iterative Clustering (SNIC). SNIC is non-iterative and does not, compared to SLIC, use k-means clustering. Instead it uses a priority queue and a distance measure to decide which pixel to add to each cluster. Achanta *et al.* also presents a version of SNIC which transforms the superpixel shapes into polygons. This is realized by merging vertices that are too close in the original superpixels and then drawing lines between the remaining vertices. Algorithms generating polygon segmentation have shown to perform well on images with geometric and fabricated structures [6].

A problem brought up by Tu *et al.* [7] is that, since superpixels are over-segmentations of an image, the superpixel boundaries within an object can be arbitrary, making them hard to train. Tu *et al.* strongly focuses on the poor boundary adherence of superpixel methods when the background color strongly resembles the object color and proposes the Pixel Affinity Net (PAN) [7] as a solution. As loss, Tu *et al.* proposes Segmentaion-Aware Loss (SEAL) which computes a loss at every pixel. PAN is combined with entropy rate superpixel (ERS) segmentation [8] to generate superpixels.

To archive end-to-end trainable superpixels Jampani *et al.* [4] proposes superpixel sampling networks (SSNs). Jampani *et al.* create a structure, in which a deep network is set in sequence with a differentiable version of SLIC. To make SLIC differentiable, Jampani *et al.* replace the hard pixel-superpixel association in the nearest-neighbour operation with soft associations. To balance this, the superpixel center calculation is also updated to accommodate the soft-association. Jampani *et al.* advocate that a strength of SSNs is the ability to vary the loss function to create different superpixels adapted for various tasks. It is also noted that the deep network structure chosen by Jampani *et al.*, is a convolutional neural network, motivated by its simplicity. This leaves room for exploration as [4] states that other network structures could be conceivable.

### 1.3.2 Video Mask Propagation

A method for transferring semantic information between frames is to link superpixels from frame to frame using Optical flow [21]. Optical flow is the motion estimation of pixels between two consecutive frames [21]. This movement can be caused by the relative movement of the observer and the frame or movements of objects within the frame [22]. Optical flow has been used to generate segmentations, however with varied results since algorithms made for this struggles with clear object borders [9]. A better approach might therefore be to use optical flow to analyse the movement patterns between frames and create predictions for the movement of superpixels and then linking them together.

There exist many methods for solving the problem of estimating optical flow, two prominent algorithms used for dense flow is Horn and Schunck [22] and Farnebäck [23] which uses additional constraints and image pyramids respectively. For sparse optical flow, one of the most prominent algorithms is the Lucas Kanade method [24] that utilize the spatial intensity gradient to estimate the flow. A more recent algorithm is the PCA-flow [25] that estimates flow by adapting a sparse-to-dense methodology. There have also been improvements within the areas as deep convolutional networks designed for the task of optical flow has emerged. One of the earlier networks made for estimating optical flow is FlowNet [26] where Dosovitskiy *et al.* introduce optical flow as a supervised learning problem. Further, SelFlow [27] increases the accuracy of the flow by presenting a method consisting of two networks with the same architecture, one trained for estimation of flow for non-occluded pixels and one for flow on all pixels.

Another approach for linking the superpixels is to propagate the previous superpixels to the current frame and connecting these with the current superpixels. Being able to propagate superpixels can be seen as a video mask propagation problem. Within this area, video propagation networks (VPNs) [10] rely on temporal coherence by using a bilateral network structure to take in previous masks and frames and propagate associated information to the current frame to generate a new mask. Caelles *et al.* [11] points out problems with relying on temporal coherence and instead pro-

poses one-shot video object segmentation (OSVOS). OSVOS [11] uses a pre-trained fully convolutional network (FCN) to learn propagation of an object from the first frame. The remaining frames are segmented using only that information, thus no temporal coherence is necessary.

Video object segmentation (VOS) is the problem of annotating one or several objects across video frames. This problem is often solved by starting with one manual annotation of one or several frames and attempting to propagate this annotation through the image sequence, [9]–[11]. These solutions can be considered as semi supervised since they require manual input but to a certain extent create automatic annotations from that input. Unsupervised methods also exists which focus on automatically generating and tracking masks across the video stream instead of relying on an original manual annotation [12] [13]. For semi-supervised VOS there is a subarea called interactive video segmentation [14], intended for when high precision is needed on all frames. The technique build on propagation of an original mask just like semi-supervised solutions but relies on user corrections of the mask for each frame to ensure a high quality across all frames.

Recently, Voigtlaender *et al.* presented a new approach for single as well as multi-object VOS called Fast End-to-End Embedding Learning (FEELVOS) [15]. This solution extracts pixel-wise embeddings from the current frame in order to both globally match with the first frame embeddings but to also match locally with the embeddings from the previous frames. By performing the local and global matching, both local and global distance maps can be extracted. These distance maps along with backbone features extracted using deeplabv3+ [28] are then distributed as inputs to the dynamic segmentation head which computes the final masks.

Another technique that is used for estimation of an unknown environment is Simultaneous Localization and Mapping (SLAM). SLAM is a computational problem where an agent is supposed to estimate its location in an environment while simultaneously building a model of that environment [29]. SLAM is most often used for generating a 3D map of its environment for the purpose of navigation [30]–[33], however, this does not provide semantic information about the scene. Kundu *et al.* [12] proposes a pipeline for generating 3D semantic segmentation in combination with spatial information from monocular video. The work [12] uses voxels (3D subvolumes) to label objects in 3D space. The 3D space allows for object tracking through sequences, information that can be inferred back to 2D images.

These solutions are semi-supervised, meaning that only one frame is manually annotated. This can result in less accurate segmentations than what is required in an annotation workflow. This problem is brought up by Price *et al.* [14] who instead suggests LIVEcut, an interactive segmentation method where the user, if necessary, in each frame corrects the annotation and the algorithm learns from the user input. LIVEcut lowers the user interaction time in video editing [14] and could conceivably be used to annotate image sequences. While LIVEcut closely relates to annotating video frames, the underlying techniques are not modern and could be improved

upon. However, the notion of interactively correcting annotations is highly relevant to this thesis and would have to be investigated in combination with more modern techniques.

### 1.3.3 Annotation

Annotation of image data is the task of marking and labeling objects or events within an image. Depending on which is the wanted task this is done differently. When using machine learning for visual scene understanding, two important tasks are: Object detection and Semantic segmentation [34], [35]. In object detection, one way of annotating is using bounding boxes which is placed around the object [36]. For the task of semantic segmentation each pixel in an image gets labeled as the object class of which the pixel belongs [37]. Annotation are usually done manually, however, there have been some approaches for streamlining the annotation process [13], [18].

Using superpixels for annotation purposes have been done before [18]. In the paper, Yamaguchi *et al.* presents an approach for parsing clothing in fashion photographs using superpixels as a tool for segmenting the image before performing pose estimation and labeling. Though [18] report good results, they also forwards the problem of superpixels that not correctly adhere to the boundaries of the specific object. Yamaguchi *et al.* describe this as a hard problem to solve as it occur for images with nearly invisible boundaries.

A different approach for automatic annotations that was recently introduced by Porzi *et al.* [13] generates training data for the task of multi-object tracking and segmentation (MOTS) [38] from generic street-level videos. This is realized by combining instance segmentation for every frame with an optical flow network to solve the linear assignment problem and extract tracklets. The optical flow network is based on the $HD^3F$ network [39] which was made for extracting pixel-wise correspondences between frames.

## 1.4 Thesis Outline

In chapter 2, the sufficient theory needed for the thesis is presented. The chapter starts by describing the background on machine learning followed by artificial neural networks and convolutional neural networks. Image segmentation and more importantly superpixel segmentation are then described. Lastly, optical flow and other techniques related to video mask propagation and image processing are presented.

Chapter 3 presents what has been developed and what new contributions have been made in this thesis. The chapter starts with describing the overall approach before moving into details about the different parts.

Following, chapter 4 describes the experiments that were performed in this thesis. The experiments are placed in a chronological order of which they were conducted.

Experiments related to the superpixel algorithm are first presented, followed by experiments related to the propagation method. Lastly, a time study is presented where the different parts of the pipeline are evaluated in an annotation environment.

Ultimately, in chapter 5 the thesis is concluded and future work is discussed.

# 2
# Theory

The thesis aims to tackle the problem of expensive annotations by investigating two possible improvements in the annotation process. These possible improvements are: generating superpixels that adhere to semantic boundaries and propagating annotations masks through image sequences.

To realize this, a superpixel algorithm that fits the task of semantic segmentation and a method for propagating annotations must be chosen and integrated. The selected algorithm for generating superpixels is the superpixel sampling networks [4] which was chosen as it is the state-of-the-art as well as being trainable to be task-specific. The method chosen for propagation is a proprietary technique that utilizes basic color matching in conjunction with optical flow from PCA-flow [25] and morphological operations.

This chapter presents the underlying theory about the areas that are relevant in this thesis. First knowledge about machine learning is presented, followed by image segmentation and more importantly superpixel segmentation. Further, background on optical flow and other techniques related to the propagation part are presented.

## 2.1 Machine Learning

In the modern era, computers are used for solving a broad variety of problems. To solve these problems an algorithm can be formulated and implemented on a computer. There are, however, problems which are not possible to formulate as a plain algorithm, problems of a non standard nature. When classifying emails, every email can be unique, making them hard to clearly cluster based on a hard criteria. Even if a basic filter can be achieved with a standard algorithm, this might break if a new unforeseen example occurs [40].

Despite these problems being difficult to formulate as a set of hard constraints, they are not random. It is possible for a human to decide if an email is spam or not. Problems such as these, where the input is known as well as the output are not impossible to solve [41]. A set of emails can be gathered with corresponding labels for if they are spam or not.

This is where machine learning enters, by possessing both the inputs and outputs it allows for the computer to 'learn' patterns in between [41]. It is assumed that

a good approximation can be achieved, even if the full pattern is not understood. Being able to approximate these patterns allows predictions to be made, assuming that future data will not differ much from the that of when the data was collected.

### 2.1.1 Supervised Learning

Problems where the input, together with the true output are known, are referred to as supervised learning. The task is for the computer to learn the relation from the input to the output [40]. The data is often collected by observing humans, such as in the case of predicting customers purchasing patterns. Data can also be collected or generated in a more conscious way. If the task is to recognize objects in images, first images containing the objects can be collected and then be annotated.

### 2.1.2 Unsupervised Learning

In unsupervised learning there is no expected output to learn from, there is only input data. As such the objective of unsupervised learning is instead to find patterns within the input distribution and find regularities to see what happens more generally. This problem is closely linked to statistics where it is called *density estimation*. One use for this is *clustering*, a model which groups together different entities based on similar attributes such as $x$ and $y$ coordinates in a 2D space, see Figure 2.1 [40].

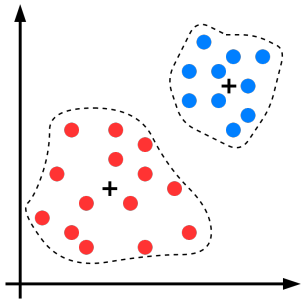

**Figure 2.1:** K-means clustering for $K = 2$.

### 2.1.3 Semi-supervised Learning

If both unlabeled and labeled data are used in combination, it is called semi-supervised learning. With semi-supervised learning the idea is that higher accuracy can be achieved by combining small amounts of labeled data with large amounts of unlabeled data instead of using only one or the other, see Figure 2.2. This is beneficial due to the requirement of humans to annotate data, making labeled data expensive [42]. For semi-supervised methods to work one or several assumptions are made, such as: The data needs to form *clusters* as points in the same cluster have a higher likely hood to belong to the same cluster. Points close to each other are more likely to share a label, thus *continuity* is often assumed [43].

## 2.2 Artificial Neural Networks

One of the most prominent ways of implementing supervised learning are Artificial Neural Networks (ANNs), a method inspired by the human neuron [44]. ANNs are constructed from artificial neurons that takes several inputs $[x_1, ..., x_n]$, multiplies them by corresponding weights $[w_1, ..., w_n]$ and summarizes them together with a bias term $w_b$, see Figure 2.3. This is then passed through an optional activation function $f$ to limit the output $\hat{y}$ to a certain range of values [45].

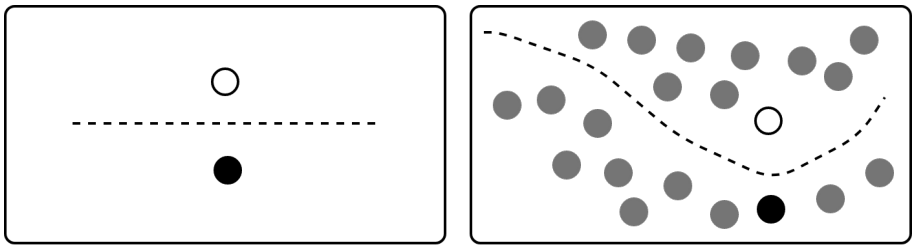

**Figure 2.2:** The influence of unlabeled data in semi-supervised learning. The left image shows the border with only labeled data, the right shows the border when unlabeled data is introduced.

To form an ANN multiple neurons are organized into layers, here each neuron is connected to neurons in the previous and later layer, see Figure 2.4. Neurons in the same layer are not interconnected. Layers that receive external data are called input layers, layers that produces the final result are called output layers and the layers in between are called hidden layers [40].

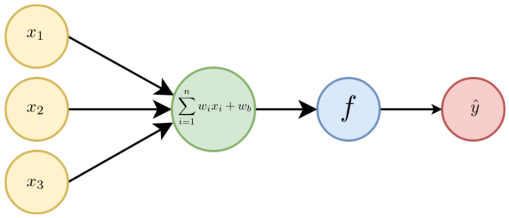


**Figure 2.3:** A typical ANN neuron.

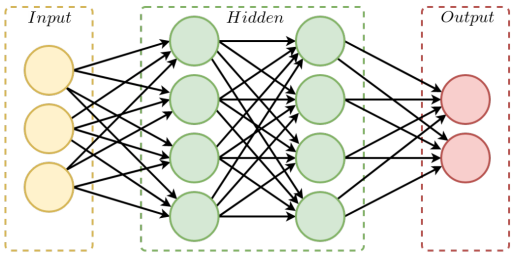


**Figure 2.4:** A simple ANN with three input neurons, two hidden layers consisting of four neurons each and an output layer with two neurons.

The goal of a neural network is to learn the mapping from the input $\boldsymbol{x} = [x_1, ..., x_n]^T$ to the expected output $\boldsymbol{y} = [y_1, ..., y_h]^T$, where $n$ is the number of inputs and $h$ is the number of outputs. This mapping is learnt by finding weights $\boldsymbol{w} = [w_1, ..., w_t]^T$ that minimizes the error between the given output $\hat{\boldsymbol{y}}$ and the expected output $\boldsymbol{y}$. For example, in the trivial case of just one neuron [46], as in Figure 2.3, no bias ($w_b = 0$) and with $p$ example pairs $[(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_p, \boldsymbol{y}_p)]$, the equation to solve is:

$$\min \frac{1}{p} \sum_{i=1}^{p} (\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2 = \min \frac{1}{p} \sum_{i=1}^{p} \left(\boldsymbol{y}_i - \boldsymbol{x}_i^T \boldsymbol{w}\right)^2. \tag{2.1}$$

### 2.2.1 Loss Function

Equation 2.1 is an example of an objective function which tells the network what is desired to achieve. In general the goal is to select the weights $\boldsymbol{w}$, such that the probability of mapping the input $\boldsymbol{x}$ to the output $\boldsymbol{y}$ is maximized. This is done in the training process through minimization of the *loss function*, in Equation 2.1 this is the part that is being summarized $\left(\frac{1}{p}(\boldsymbol{y}_i - \boldsymbol{x}_i^T \boldsymbol{w})^2\right)$. The loss function is a function that

needs to cover all of the properties of the task with a single value which determines how well the solution performed. This is also a value which allows comparison between different solutions. There exist multiple loss functions and depending on what tasks to solve, as such it is important to select the proper loss function for the task. Two common loss functions are cross-entropy loss for classification and mean square error for regression [47].

### 2.2.2 Training

The optimization of the model weights is referred to as training. There are multiple ways of achieving this, the dominant method however, is gradient descent [40]. Gradient descent is a method of finding the local minima for a multi variable function. This is an iterative process, where an initial guess for the parameters first is made, the parameters are then updated with the negative gradient of the function, see Equation 2.2. Here $\alpha$ is the size of the step to take, also called learning rate in machine learning, $F$ is the function to optimize and $\boldsymbol{a}$ is the parameters to optimize.

$$\boldsymbol{a}_{n+1} = \boldsymbol{a}_n - \alpha \nabla F(\boldsymbol{a}_n),\ n \geq 0 \tag{2.2}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \cdot \nabla_{\boldsymbol{w}} \sum_{i=1}^{m} L_i(\boldsymbol{w}) \tag{2.3}$$

More specifically for deep learning, the weights are updated according to Equation 2.3. Here, $L_i$ is the loss of example $i$ and $m$ is the number of examples (also known as *batch size*). $\boldsymbol{w}$ is the weights. The batch size determines the type of gradient descent is chosen, if $m = p$ (all examples) it is known as regular *gradient descent*. This type of gradient descent will always converge to a local minimum and if the loss function is convex, it will converge to the global minimum. At the other extreme ($m = 1$) *stochastic gradient descent* can be found, this approach updates for every example instead of for every batch, this makes it much faster, however with a noisier descent. The option in between ($1 < m < p$) is called *mini-batch gradient descent*, thus allowing tweaking of the batch size.

## 2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a category of neural networks made for the processing of data that has a grid-like structure. Images are an example of this type of data as they are a two-dimensional grid consiting of pixels [40]. The main difference between CNNs and ANNs is that CNNs consist of convolutional layers instead of the traditional neurons.

### 2.3.1 Key Components

The CNN architecture consist of a few key components and functions. This section describes the most important and relevant components to this thesis.

#### 2.3.1.1 Convolutional Layer

The target of the convolutional layer is to extract features from an input image using the convolution operator. The convolution is realized through sliding a window (kernel) of finite size across the input image and computing the dot product. The input image is often a multi-dimensional array of data whereas the kernel is a multidimensional array of weights which are updated during the learning process [40]. The output is referred to as the feature map and depending on the weights and the hyperparameters of the kernel, the convolution will generate different feature maps for the same input. The convolution is described by

$$f(i,j) = I(i,j) * w(i,j) = \sum_{k,l} I(k,l)w(k-i, l-j), \tag{2.4}$$

where $*$ is the convolution operator, $I$ is the input image, $w$ is the kernel and $f$ is the output feature map [21]. Two of the most important hyperparameters are the number of channels and the stride, which decides the kernel dimensions and the number of pixels which the kernel is shifted respectively.

Every channel of the input image will be convolved with a distinct kernel meaning that Equation 2.4 is performed for every channel. Thus, the number of channels in the kernel needs to be the same as in the input image. The output feature map is then obtained by element-wise sum the feature maps. A convolutional layer can contain multiple convolutions for which each produce one of the output feature maps, meaning the number of output channels is equal to the number of convolutions in a layer.

A simple example of a convolution can be seen in Figure 2.5.

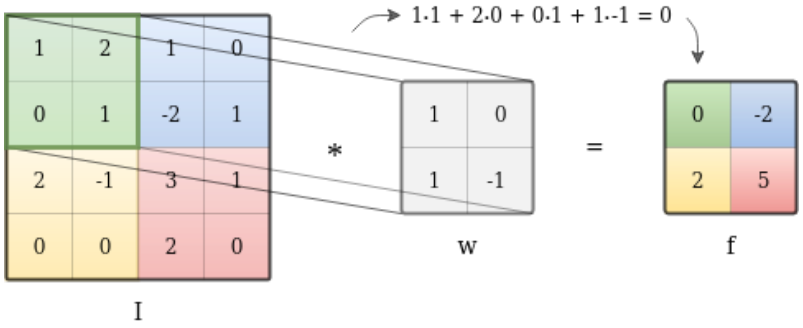


**Figure 2.5:** A toy example of convolution with a 2-dimensional input image of size $(4 \times 4)$, a kernel of size $(2 \times 2)$ and a stride of 2.

#### 2.3.1.2 Activation Layer

The activation layer in a neural network is the function that creates non-linearity in a model. Without the activation function, the network would only be able to provide linear mapping between the input and output. The operation is element-wise meaning that the mathematical function is performed on every value individually.

Examples of activation functions are *Rectified Linear Unit* (ReLU), *Sigmoid* and *Softmax* [40].

ReLU [48] activation function, which in the last few years has become very popular [49], uses the `max`-function to threshold at zero, it is described as

$$g(z) = \max\{0, z\}. \tag{2.5}$$

The Softmax function computes the probability distribution and thus the output is in the range between 0 and 1. The softmax function is rarely used inside a model. It is usually used at the end to classify the output into different classes [40]. The function is given by

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}. \tag{2.6}$$

#### 2.3.1.3 Pooling

The pooling layer reduces the size (downsamples) of the input features in a similar manner as the convolution but with a different function than the dot product. Depending on which pooling, and function, is used, different features are kept. Two types of pooling are *max pooling* and *average pooling* [40]. In max pooling the `max` operator is used to obtain the maximum value of the current patch when doing sliding window. This value is then passed on to the output. Average pooling the average value of the current patch is taken. Similar to the convolution layer, the pooling layer acquire the hyperparameter stride which decide the step size when performing the sliding window operation.

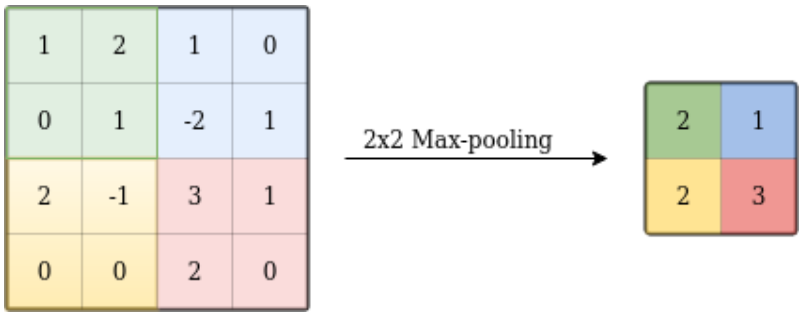


**Figure 2.6:** A toy example of max-pooling with a $2 \times 2$ kernel and a stride of 2.

#### 2.3.1.4 Batch Normalization

Batch normalization (BN) [50] is a method that is often applied after the convolutional and fully-connected layer to normalize the initialization and make the training faster [51]. BN is based on the assumption that the problems that follow the covariate shift for initialization also applies to layers inside the network. Meaning problems caused by change in the distribution of network activations. This makes

the networks more complicated to train as it is more sensitive to high learning rates and wrongly made parameter initializations. Thus, the BN layer is developed to reduce the internal covariate shift [50]. Ioffe *et al.* found that by only adding BN to a state-of-the art image classification model reduced the training time substantially.

BN is realized through first computing the sample mean $\mu_{BN}$ and sample variance $\sigma^2_{BN}$ for the entire mini-batch. For one neuron this is expressed by:

$$\begin{aligned} \mu_{bn} &= \frac{1}{m}\sum_{i}^{m} z^i \\ \sigma^2_{bn} &= \frac{1}{m}\sum_{i}^{m}(z^i - \mu_{bn})^2, \end{aligned} \tag{2.7}$$

where $m$ is the mini-batch size and $z$ is the linear combination of the inputs to the neuron and their respectively weights. Further, the $z^i$ is normalized $z^i_{norm}$ and ultimately scaled and shifted $\tilde{z}^i$. This is defined by:

$$\begin{aligned} z^i_{norm} &= \frac{z^i - \mu_{BN}}{\sqrt{\sigma^2 + \epsilon}} \\ \tilde{z}^i &= \gamma z^i_{norm} + \beta, \end{aligned} \tag{2.8}$$

where $\epsilon$ is a small positive number and $\gamma$ and $\beta$ are learnable parameters [50].

#### 2.3.1.5 Upsampling

Upsampling is the technique that increases the resolution of an input image. There exists multiple methods for this such as: *Bilinear upsampling* and *Transposed convolution.*

In bilinear upsampling the values of the pixels are calculated using bilinear interpolation. Bilinear interpolation is the 2D version of the 1D linear interpolation where interpolation is realized by computing the linear interpolation in two directions by using its four nearest neighbours [52]. Linear interpolation is a method for estimating the value of a new constructed data point that is placed in between two measured data points. The value of the new data point lies on the line that goes through its two closest neighbours. If $x$ is the coordinate of the new point and $x_i < x < x_{i+1}$, then value $y(x)$ is estimated as:

$$y(x) = y(x_i) + (y(x_{i+1}) - y(x_i))\frac{x - x_i}{x_{i+1} - x_i}. \tag{2.9}$$

Thus, the upsampling is realized through computing the bilinear interpolation for the new positions with unknown values.

## 2.4 Image Segmentation

Image Segmentation is the task of clustering pixels that has some kind of spatial relation to each other. The clustering aims to identify meaningful segments that can be used to describe the content of the image. There exists multiple segmentation task in which what is segmented and the amount of things segmented is different.

### 2.4.1 Superpixel Segmentation

Superpixel segmentation is a method, in which pixels in an image are grouped together to create meaningful regions. The superpixels can then be used instead of the original pixels to reduce complexity and speed up performance in many computer vision applications [6]. Superpixels have been used in areas such as object localization [53] and stereo vision tasks [54]. For superpixels to be useful, Achanta *et al.* [5] and Stutz *et al.* [55] points out important properties of the superpixels and the algorithms used to create them as:

- Each superpixel must define a distinct partition of the image.
- Every pixel must be assigned to a superpixel.
- The pixels in the superpixels must be connected.
- The image boundaries should be preserved.
- Superpixels should be compact in shape, regularly placed over the grid and have smooth boundaries when there is an absence of image boundaries.
- Superpixels should be efficient to generate.
- The number of superpixels should be controllable.

#### 2.4.1.1 Superpixel Sampling Networks

The state-of-the-art algorithm for generating superpixels is superpixel sampling networks (SSNs) [4]. The algorithm combines a neural network for extracting image features with a differentiable version of the superpixel algorithm SLIC. A schematic view of the algorithm can be seen in Figure 2.7.

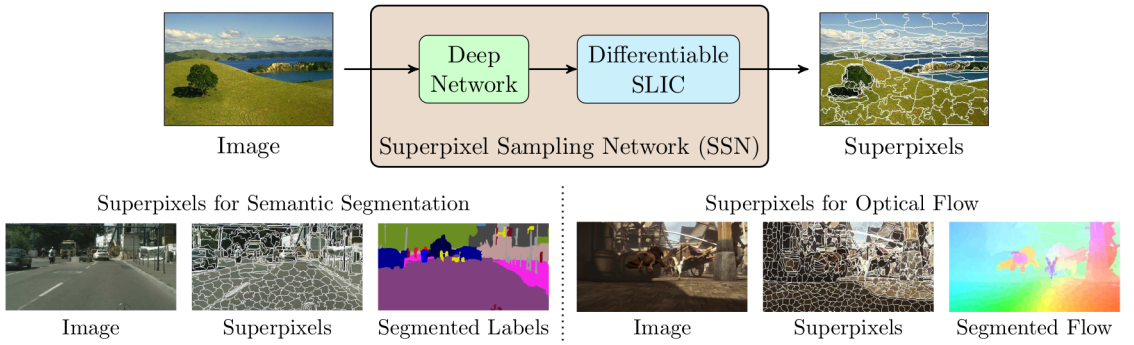


**Figure 2.7:** Image taken from [4] which shows an overview of the SSN [4]. The top shows an image being the input to the deep network which further sends the feature maps to the differentiable SLIC which generates the final superpixels. On the bottom, examples of when SSN is trained for semantic segmentation (left) and optical flow (right) are shown.

The approach is summarized in Algorithm 1. The input of the algorithm is an five-dimensional image with spatial dimension $H \times W$ (height $\times$ width). The 5D space has the dimensions $[labxy]$ where $[lab]$ is the pixel color vector from the CIELab color space [20]. The $[xy]$ dimensions is the pixel xy-coordinates.

In the initialization part of the algorithm, the interleaved CNN extracts deep features $F$ of size $(n \times k)$ from the input image and the superpixel centers are initialized (line 1 and 2). Where $n$ is the number of pixels in the image and $k$ is the number of channels decided by the network. The $m$ superpixel centers $S^0 \in \mathbb{R}^{m \times 5}$ are then sampled uniformly across the pixel grid. The centers are then moved to the position within a $3 \times 3$ neighborhood with the lowest gradient to prevent placing superpixel centers on object boundaries [5].

Next, on line 3 the differentiable SLIC is looped $v$ times and updates the soft pixel-superpixel associations $Q_{pi}^t \in \mathbb{R}^{n \times k}$ and superpixel centers $S^t$ are updated. The pixel-superpixel associations $H \in \{0, 1, \ldots, m-1\}^{n \times 1}$ computed in the original SLIC are hard associations, meaning that every pixel is said to belong to a specific superpixel in the SLIC loop. However, in the differentiable SLIC these are changed to soft associations enabling the algorithm to be end-to-end trainable. These soft associations are given by

$$Q_{pi}^t = e^{-||F_p - S_i^{t-1}||^2}, \tag{2.10}$$

where $t$ refers to the current SLIC iteration. Further, $p$ and $i$ refers to the pixel and superpixel respectively. The distance computations are restricted to only the surrounding 9 superpixels due to the memory and computational power it takes to compute the distance for the complete set of superpixels.

The superpixel centers are derived using

$$S_i^t = \frac{1}{Z_i^t} \sum_{p=1}^{n} Q_{pi}^t F_p \tag{2.11}$$

where $Z_i^t = \sum_{p=1}^{n} Q_{pi}^t$ is denoted as the normalization constant.

After the SLIC iterations, the hard association $H_p = \arg\min ||F_p - S_i^{t-1}||^2$ are computed to assign the pixels to the final superpixel clusters (line 8). Enforced connectivity (line 9) is then applied to the output by merging superpixels that are smaller than a threshold with its surrounding superpixels. This reassigns small superpixels as well as pixels that are not spatially connected to their assigned superpixel.

**Algorithm 1** Superpixel sampling network [4].

**Input:** Image $\underset{n\times 5}{I}$.

**Output:** Pixel-Superpixel association $\underset{n\times m}{Q}$.

1: Pixel features using a CNN, $\underset{n\times k}{F} = \mathcal{F}(I)$.

2: Initial superpixel centers with average features in regular grid cells, $\underset{m\times k}{S^0} = \mathcal{J}(F)$.

3: **for** each iteration $t$ in 1 to $v$ **do**

4: Compute association between each pixel $p$ and the surrounding superpixel $i$, $Q^t_{pi} = e^{-||F_p - S_i^{t-1}||^2}$.

5: Compute new superpixel centers, $S_i^t = \frac{1}{Z_i^t}\sum_{p=1}^{n} Q^t_{pi}F_p$; $Z_i^t = \sum_p Q^t_{pi}$.

6: **end for**

7: (*Optional*) Compute hard-associations $\underset{n\times 1}{H^v}$; $H_p^v = \underset{x\in\{0,1,\ldots,m-1\}}{\arg\max}\ Q^v_{pi}$.

8: (*Optional*) Enforce spatial connectivity.

### 2.4.2 Optical Flow

Optical flow is the estimation of motion computed at each pixel [21]. The input of an optical flow algorithm is a pair of images from a sequence. The algorithm produces a two-dimensional image where each pixel value represent the approximated movement vector in $x$ and $y$-direction.

Optical flow is based on the assumption that the intensity or color of a pixel moving is constant between frames. This assumptions is called the brightness constancy constraint [56]. With the intensity of a pixel $(x, y)$ at time $t$ expressed as $I(x, y, t)$ and the brightness constancy constraint applied, the intensity when moving the pixels by $(\Delta x, \Delta y)$ will be constant. The relationship between the old and new frame can be expressed as

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{2.12}$$

By applying the first order Taylor expansion, the optical flow constraint can be derived

$$I_x u + I_y v + I_t = 0, \tag{2.13}$$

where $I_x$, $I_y$ and $I_t$ denotes the partial derivatives respectively $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ [56]. These can be obtained from the two frames. The movements over time are denoted $(u, v) = (\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t})$ and are unknown in this equation making it an underdetermined system. There exist a few methods to solve this problem and common techniques are: gradient-based algorithms [24], polynomial expansion based algorithms [23] and sparse-to-dense approaches [25].

#### 2.4.2.1 PCA-flow

PCA-flow is a method that uses a sparse-to-dense approach when estimating the optical flow. The algorithm uses pre-computed basis vectors $\boldsymbol{b}$ to map the sparse

features to dense features [25].

To estimate the sparse features, the PCA-flow first normalize the frames using *Contrast limited adaptive histogram equalization* (CLAHE) [57] in order to enhance the information that the feature detectors can capture [25]. This method remaps the image intensity in order to flatten the intensity histogram. Resulting in an enhanced information content as the entropy in the image increases [58].

The feature points are then extracted and matched by utilizing the method presented in [59]. This method extracts features by finding blob max, blob min, corner max and corner min features. This is done by first filtering the frames with blob- and corner detectors and then using non-maximum- and non-minimum-suppression on this. Further, feature points are matched based on Sobel filter responses. These sparse feature points are defined as: $\{(\boldsymbol{p}_k, \boldsymbol{q}_k)\}$ for $k = 1, \ldots, K$ where $\boldsymbol{p}_k$ is the 2-dimensional location of a feature point in the first frame and $\boldsymbol{q}_k$ is the corresponding locations in the second frame. The displacement of these feature points are defined as:

$$\boldsymbol{v}_k = \boldsymbol{q}_k - \boldsymbol{p}_k. \tag{2.14}$$

To convert the sparse features to dense optical flow, finding the weights $\hat{\boldsymbol{w}}$ that provides the best description of the feature matches is seen as a regression task and can thus be expressed as a least square problem

$$\hat{\boldsymbol{w}} = \text{argmin}_w ||\boldsymbol{A}\boldsymbol{w} - \boldsymbol{y}||_2^2, \tag{2.15}$$

where $\boldsymbol{y} = [v_{1,x}, \ \ldots, v_{K,x}, v_{1,y}, \ \ldots, v_{K,y},]^T$ and $\boldsymbol{A}$ is a matrix containing the learned basis vectors $\boldsymbol{b}$. These basis vectors were computed offline beforehand and this was done by first extracting optical flow using GPUFlow [60] from movies. The basis vectors $\boldsymbol{b}$ were then selected using the line fitting method *principal component analysis* (PCA) as presented in [61].

Lastly, the optical flow is approximated as the weighted sum the basis vectors

$$\boldsymbol{u} = (\boldsymbol{u}_x^T, \boldsymbol{u}_y^T) \approx \sum_{n=1}^{N} w_n \boldsymbol{b}_n. \tag{2.16}$$

### 2.4.3 Morphological Operations

Morphological operations are tools used in image processing for content enhancement and removal of noise. These are most often used on binary but also grayscaled images. Where on binary images, one class is considered the foreground and the other background. Two of the most basic operations are *erosion* and *dilation*. Erosion remove pixels from boundaries of foreground objects as well as increase size of holes. In oppose to erosion, the dilation operation adds pixels to boundaries and fills holes. Thus, dilation is the opposite of erosion and therefore erosion on the foreground is equivalent to dilation on the background. These operations takes

two inputs, the image to be processed and a structuring element. The structuring element (also known as kernel) decides the effect and magnitude of the operation. There exist multiple structuring elements and a few common are: squared, cross-shaped and disk-shaped [62].

Based on erosion and dilation are the two operations: Opening and Closing. The opening operations starts by performing erosion and thereafter dilation whereas the closing operation works in a reverse manner as it first performs dilation followed by erosion [63]. The effect of opening is that it smoothes the edges of objects and removes thin protrusions. However, closing also smoothes borders but instead of removing border pixels it adds border pixels and fills small holes.

# 3
# Method

To solve the problem of expensive annotations, an approach that utilizes superpixels and propagation in the annotation process was developed. The approach consists of two main parts: superpixel segmentation and video propagation. A schematic overview of the approach can be seen in Figure 3.1.

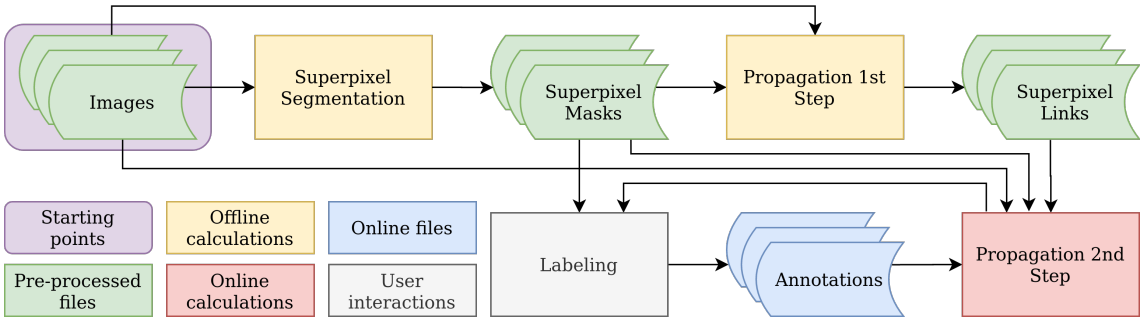


**Figure 3.1:** Schematic overview of the approach.

In the superpixel part, superpixels are generated from images using the method superpixel sampling networks. Further, superpixels are used together with the images to link superpixels between images. This is realised by a self developed method that utilizes color matching, supported by optical flow for better movement accuracy and morphological operations to clean up the propagation at run time. Lastly, to fully investigate the effects of superpixels in combination with the propagation an annotation tool utilizing these has been developed.

## 3.1 Superpixel Sampling Networks

For the generation of superpixels, the algorithm superpixel sampling networks (SSNs) [4] was selected. This method combines a deep neural network for feature extraction with a differentiable version of the original SLIC algorithm [5]. The approach is the state-of-the-art and has the ability to learn task-specific superpixels by varying the loss function and training data [4]. Based on the evaluation of different network structures, see 4.1.6, the U-Net [64] was selected as a base architecture for the final network. After superpixels have been generated, connectivity is enforced by dividing non-connected superpixels into new unique superpixels. The overview of the approach can be seen in Figure 3.2.

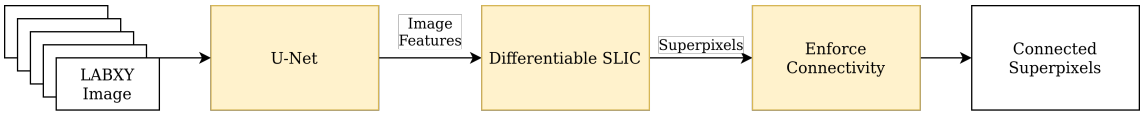


**Figure 3.2:** Schematic overview of the superpixel generation process.

### 3.1.1 Network Architecture

The selected network architecture is very similar to the original U-Net [64] architecture but instead of using transposed convolutions for upsampling, bilinear upsampling is used. Bilinear upsampling is used as the U-Net implementation our implementation was based on used bilinear upsampling instead of transposed convolutions. An overview of the network architecture can be seen in Figure 3.3.

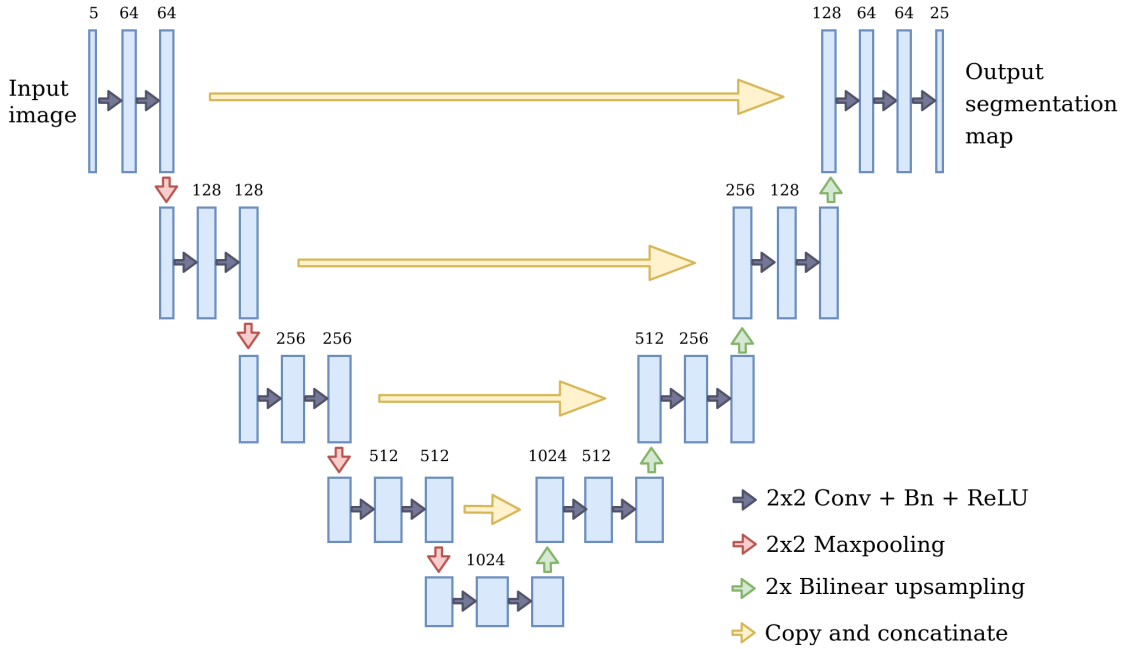


**Figure 3.3:** The U-Net network architecture used in this thesis.

U-Net is based on the fully convolutional network [65] and was originally made for biomedical image segmentation. The architecture consist of two distinct parts: the *contracting* and *expansive*. In the contracting part, the input is downsampled and the number of channels are increased. This part consists of a series of $3 \times 3$ convolution layers interleaved with BN [50] and ReLU activations [48] and $2 \times 2$ max-pooling layers with stride 2 for downsampling. The number of filters for each convolution layer is increased by two, thus resulting in twice the feature maps after each convolutional layer.

The expansive part is a symmetric version of the contracting part and thus consist of four convolution and upsampling steps. The convolutions are performed with a kernel of size $3 \times 3$ and are interleaved with BN and ReLU activations. The upsampling is realized through $2 \times 2$ bilinear upsampling which increases the spatial size.

After each upsampling step, the feature maps are concatenated with the respective feature maps from the contracting part to regain additional information regarding localization of features. Due to the convolutional layers not using padding, the feature maps from the contracting part needs to be cropped. Ultimately, the feature map is run through a $1 \times 1$ convolution layer to obtain an output with 20 output channels.

### 3.1.2 Loss Function

Two different loss functions are used: Reconstruction loss and Compactness Loss. The overall loss function is defined as the summed losses

$$L = L_{reconstruction} + L_{compact}.$$

#### 3.1.2.1 Reconstruction Loss

The reconstruction loss is a task-specific loss designed to help in the process of generating superpixels that adhere to semantic boundaries. This is done by using the semantic labels as target and a pixel representation based on the pixel-superpixel association matrix $Q$, see 2.4.1.1, generated from SSN and the target as the input. The target semantic labels is a pixel representation denoted as $R \in \mathbb{R}^{n \times l}$ where $n$ is the number of pixels and $l$ is the number of classes in the target.

To obtain the input, the target superpixel representation $\bar{R}$ is mapped from the target $R$ with the help of the column-normalization $Q$ matrix, denoted $\hat{Q}^T$,

$$\bar{R} = \hat{Q}^T R,$$

where $\bar{R} \in \mathbb{R}^{m \times l}$ and $m$ is the number of superpixels. Further, the superpixel representation $\bar{R}$ is converted back into a pixel representation using the row-normalized $Q$ matrix, denoted $\tilde{Q}$,

$$R^* = \tilde{Q}\bar{R},$$

where $R^* \in \mathbb{R}^{n \times l}$. This pixel representation $R^*$ is then comparable to the target semantic labels $R$ and is used as the input to the reconstruction loss

$$L_{reconstruction} = \mathcal{L}(R, R^*)$$

where the loss $\mathcal{L}$ is defined as the negative log likelihood loss. Ultimately, the final loss is computed as the mean value of the logarithmic values.

#### 3.1.2.2 Compactness Loss

To encourage superpixels to have low spatial variance, a compact loss is introduced. The target of the loss is the original positional pixel features $I^{xy} \in \mathbb{R}^{n \times 2}$ which is defined as the $[xy]$ dimensions of the input $[labxy]$ image. Furthermore, input of the loss is represented by the positional pixel features $\bar{I}^{xy} \in \mathbb{R}^{n \times 2}$ from the generated superpixels which is defined by Equation 3.1.

To compute the input $\bar{I}^{xy}$, the superpixel representation $S^{xy}$ is first mapped from the original position features $I^{xy}$ using the column-normalization association matrix $\hat{Q}^T$

$$S^{xy} = \hat{Q}^T I^{xy}$$

where $S^{xy} \in \mathbb{R}^{m \times 2}$. Further to obtain the new positional pixel representation $\bar{I}^{xy}$, pixels belonging to a specific superpixel are given the positional features of that superpixel. This is done by utilizing the hard superpixel-pixel association $H$, which is the matrix that directly maps every pixel to one specific superpixel, together with the positional features that are mapped into the superpixel representation $S^{xy}$. This is defined by

$$\texttt{if } H_p = i: \ \bar{I}_p^{xy} = S_i^{xy}, \tag{3.1}$$

meaning that if pixel $p$ belongs to superpixel $i$ then the positional feature of superpixel $i$ is assigned to $\bar{I}_p^{xy}$. The compact loss is then defined by the mean square error

$$L_{compact} = \frac{1}{n} \sum_{i=1}^{n} (I^{xy} - \bar{I}^{xy})^2.$$

## 3.2 Propagation

To propagate annotations in between images a variety of techniques was used. A focus was put on doing most of the processing offline, as time consuming processing during the annotation phase would not support the end goal of lower interaction time per image. The propagation begins by linking every superpixel in mask $i + 1$ to a superpixel in mask $i$ as well as calculating the internal links between superpixels in mask $i + 1$ (neighbours), see Figure 3.4. The calculate links step is further explained in 3.2.1

At runtime the user then labels superpixels in mask *i+1* to create an annotation. This annotation is then propagated using the preprocessed linked list, see Figure 3.4. In the propagated labels mask *i+1* that is had after the propagation, outliers might occur, as such this mask is refined in the clean suggestion step to achieve the suggested labeled mask *i+1*, see 3.2.2.

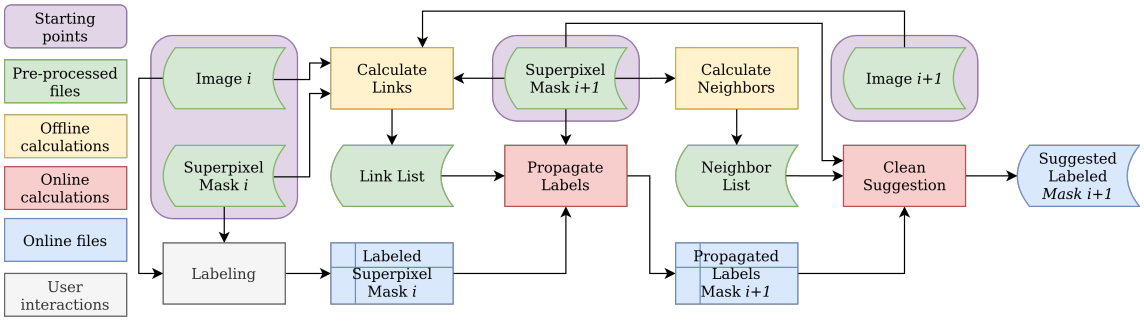


**Figure 3.4:** A flowchart showing the overview for the propagation of annotations.

### 3.2.1 Link Calculation

The links between superpixels and images were calculated from a combination of color matching and optical flow, see Figure 3.5.

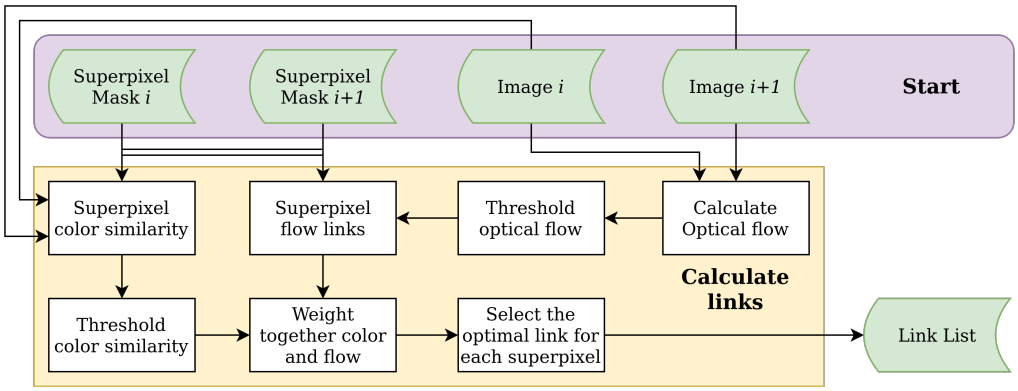


**Figure 3.5:** The procedure for linking superpixels between images.

For each superpixel in both images the average color is calculated. Taking the distance in the CIELab color space, a measure for color similarity between each superpixel in mask $i$ and mask $i+1$ can be derived. The measure is calculated by treating each color channel as a position, claculating the measure as the euclidean distance between the average color of 2 superpixels:

$$d = \sqrt{(a_{i,j} - a_{i+1,k})^2 + (b_{i,j} - b_{i+1,k})^2 + (L_{i,j} - L_{i+1,k})^2}.$$

Here d is the distance measure calculated between superpixel $j$ in mask $i$ and superpixel $k$ in mask $i+1$, calculated in the CEILab color space, a, b and L are the CEILAB channels. This measure can then be thresholded, meaning, for each superpixel in mask $i+1$ only superpixels in mask $i$ with a similarity below the threshold are taken into consideration as a potential match.

To get a better temporal coherence in the linking of superpixels, optical flow from image $i$ to image $i+1$ is used in combination with the color matching. The algorithm selected for computing optical flow is PCA-flow [25]. This algorithm was chosen as it performs well for the task and is simple to implement as it is part of the opencv library [66].

After the optical flow is computed it is thresholded in two steps: First to reduce noise from pixels that are not moving and second to remove outliers. To reduce noise from pixels that are not moving, the absolute value of flow is lower bounded. Meaning, pixels which movement in any direction is less than $\tau$ is assumed to be not moving in that direction. The threshold is expressed

$$\texttt{if} : |l_i| < \tau \;\Rightarrow\; l_i = 0,$$

where $l$ is the flow vector and $\tau = 2$. The value is chosen through testing different thresholds.

To remove outliers, the second threshold was set at 2.5 standard deviations of the size of the optical flow vectors, as such a lower and an upper bound for the optical flow was set dynamically for every image pair. The threshold is defined as

$$\mu_l - 2.5\sigma_l \geq l_i \geq \mu_l + 2.5\sigma_l,$$

where $\mu_l$ is the mean and $\sigma_l$ is the variance of the optical flow vectors.

Using the optical flow, the average expected position of each superpixel from image $i$ in image *i+1* was calculated. Taking the average position of each superpixel in image *i+1*, the distance between between each superpixel in *i+1* and the expected position of each superpixel from $i$ could be derived.

Ultimately, by weighting together the optical flow and the color matching, a correspondence score between each superpixel in *i+1* and $i$ was calculated. The link with the highest score was deemed a match and was saved in a list pertaining all the correspondences, this is refereed to as link list in Figure 3.5. The weighted correspondence score was calculated according to

$$c_{score} = d_e^3 + d_c^4,$$

here $d_e$ is the distance between the expected position of a superpixel from $i$ to $i+1$, $d_c$ is the color distance. The exponents was chosen based on trail and error, as this combination gave the best result when evaluating the method numerically for mIoU.

### 3.2.2 Cleaning Propagation

When linking superpixels, the output is not perfect. There might for instance be outliers such as superpixels labeled as sky among superpixels labeled as road. As this is highly unlikely to be correct it is desirable to get rid of these outliers, therefore a method for cleaning the propagations at run time was proposed, see Figure 3.6.

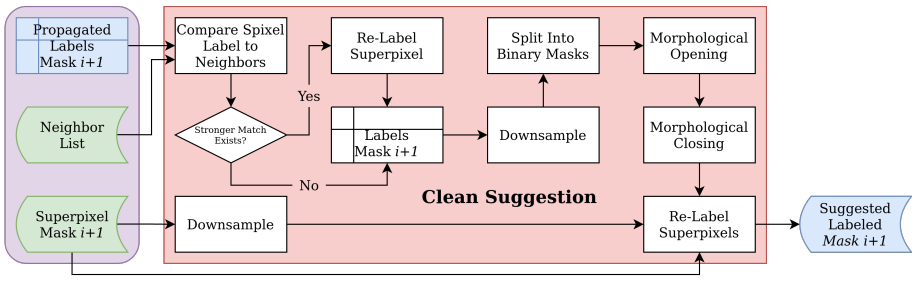


**Figure 3.6:** Flowchart that portrays how the cleaning of annotations after propagation occurs.

The first step, calculating neighbours, occurs offline. Here every superpixel is processed and all of its neighbouring superpixels indices are saved so they can be referenced later.

The second step, which occurs at run time, is to process each superpixel and compare its label to those of its neighbours, see Figure 3.6. If another class is a stronger match, with a certain margin to the originally labeled class, the superpixel is relabeled as the strongest match. The margin is set as a threshold:

$$\tau = \frac{0.4}{n_{classes}}.$$

Here $n_{classes}$ is the number of classes contained in the neighbouring superpixels (no label is treated as a class). Based of $\tau$ the criteria for relabeling a superpixel is set as:

$$\frac{n_{other\ class}}{n_{neighbors}} > \frac{n_{same\ class}}{n_{neighbors}} + \tau.$$

$n_{neighbours}$ is the number of neighboring superpixels, $n_{same\ class}$ is the number of neighboring superpixels with the same class as the central superpixel and $n_{other\ class}$ is the number of neighbouring superpixels belonging to one class that is not the same as the central superpixel. If this criteria is met, the superpixel is relabeled as the class that fulfills the criteria. As such for a relabeling to occur, the new label has to be $\frac{40\%}{number\ of\ classes}$ better match than the original label.

The neighbour matching works well for removing obvious outliers, however the mask still has a lot of noise. One approach to remove noise in binary masks is morphological operations. However, the labeled mask in this case is not binary. To solve this, the labeled mask is split into multiple binary masks, one for each class with no label being treated as a separate class. Morphological opening is then performed on each binary mask followed by Morphological closing to remove noise. This is done on a mask that is downsampled to half the resolution to reduce processing time, see Figure 3.6.

When up-sampling the mask again, it is important that the classes adhere to the superpixel borders. To achieve this the original unlabeled mask is downsampled to the same resolution as the binary masks. Every superpixels in the full resolution mask gets labeled as the class which has the highest number of overlapping pixels with the superpixel in the low resolution mask.

## 3.3 Annotation Tool

To be able to evaluate the full annotation pipeline using superpixels and propagation, an annotation tool using these methods was developed. To enable annotations using the superpixels, the annotator was given a few basic tools. The first of which was the ability to overlay a superpixel mask over the image, this mask could be toggled as it can obstruct the ability to see true object borders within the image. Other basic functionalities are:

- Left click to select a superpixel.
- Right click to deselect a superpixel.

- Left click and drag to select multiple superpixels.
- Right click and drag to deselect multiple superpixels.
- Left click and circle to select all encircled superpixels.
- Right click and circle to deselect all encircled superpixels.
- Scroll to zoom.
- Middle click and drag to pan.
- Zoom inversion to facilitate user preferences.
- Undo and Redo functions to quickly correct mistakes.
- Clear function to start the annotation from the beginning.

As superpixels alone do not match the borders that are desired perfectly the ability to correct these was built in to the tool. This was done through a process called *add corrections*, a tool that allowed the user to add new superpixels by drawing polygons. The polygons simply overwrite the underlying superpixels for an intuitive correction with a low processing requirement. The layout of the tool as well as the demonstration of a few basic functions can be seen in Appendix A.

# 4
# Experiments

This chapter describes the experiments performed in the thesis as well as information regarding the metrics, datasets and the implementation. Further, the chapter also presents the results and a discussion regarding the experiments. The experiment chapter is divided into three parts: evaluation of the superpixel segmentation, evaluation of the propagation method and the time study. The evaluation of the superpixel segmentation showed that the SSN U-Net model presented in this thesis have a performance close to the state-of-the-art SSN. The propagation method demonstrated that propagating annotations are only beneficial for classes that are big in size and and consistent in appearance. Finally, the time study indicate that the approach could be used for annotation purposes, however the annotation tool needs to be developed further.

The experiments are presented in a chronological order, starting with presenting the evaluation of the superpixel segmentation followed by the experiments done on the propagation part. Lastly, a time study is conducted where different parts of the pipeline are measured in an annotation environment.

## 4.1 Superpixel Evaluation

To find a suitable network to include in the final algorithm, a few different network structures are evaluated against each other. Later, through evaluating the final superpixel segmentation algorithm it is possible to find out how the suggested approach stand compares to the current state-of-the-art.

### 4.1.1 Dataset

The dataset used for training and evaluation of the superpixel segmentation is the Cityscapes dataset [1]. Cityscapes is a dataset developed mainly for semantic urban scene understanding which aligns well with the thesis. Other datasets were also considered, such as BDD100K [67], Mapillary Vistas [2] or ApolloScape [68]. These datasets also aligns well with the thesis and would be suitable to be used for evaluation. Ultimately Cityscapes was chosen as it has been used for evaluating state-of-the-art superpixel algorithms [4], [7], which allowed easy comparison.

Cityscapes consists of $25,000$ urban street scenes from 50 different cities in Germany and in neighboring countries. Out of these images, $5,000$ are high quality pixel-

level annotations. Boundary adherence is a highly desirable property when using superpixels for annotations, as it is the main factor for the quality of the annotations. Intuitively, to get superpixels with high boundary adherence, it is necessary to use ground truth annotations with high boundary adherence, as such only the high quality Cityscapes annotations were used.

### 4.1.2 Evaluation Metrics

To evaluate the performance and the quality of superpixels, three different metrics were used, namely Achievable Segmentation Accuracy (ASA), Boundary Precision (BP) and Boundary Recall (BR). ASA is a measure of how much of the image is correctly labeled by the superpixels and is a commonly used metric for evaluating superpixels [4]–[7], [55]. This is calculated on a pixel level to give a percentage of the amount of correctly labeled pixels, as such an ASA-score of 100% implies that all pixels are labeled correctly.

ASA can be a good metric but it does not give the whole truth, as it does not give any information about the boundary adherence of the superpixels. To measure the boundary adherence, BR and BP are used as they are common when evaluating superpixel boundary performance [6], [5], [4]. BR is a measurement of how many of the ground truth border pixels are captured by the superpixel boundaries. This can however be misleading, as simply labeling every pixel as a boundary pixel would give a recall of 100%. Due to that, BP is also measured as the percentage of correctly labeled boundaries with respect to the total number of labeled boundaries. Thus by labeling every pixel as a boundary pixel BP would be 0%.

#### 4.1.2.1 Achievable Segmentation Accuracy

To compute the ASA score each superpixel is labeled as the ground truth segment which has the largest overlap with the superpixel. From there the number of correctly labeled pixels can be calculated and summed up for all superpixels. This is then divided by the total number of pixels in the image to give segmentation accuracy. Thus, for a given superpixel segmentation and the respective ground truth segmentation, the ASA score is given by

$$ASA(S, G) = \frac{1}{n} \sum_{S_j \in S} \max_{G_i} |S_j \cap G_i|. \tag{4.1}$$

Here $S = (S_1, ..., S_{n_S})$ and $G = (G_1, ..., G_{n_G})$ denotes the superpixel mask and the ground truth sematic segmentation mask, with $S_i$ and being the $i_{th}$ superpixel and $G_i$ and being the $i_{th}$ ground truth segmentation. $S_k \cap G_i$ refers to the intersection of pixels between the superpixel $S_k$ and the ground truth segmentation $G_i$.

#### 4.1.2.2 Boundary Precision and Recall

BR is calculated by taking the ratio between the amount true positives (TP) boundary pixels and the amount of actual borders pixels, in other words, the TP plus the

false negatives (FN). Whereas BP describes how many of the labeled superpixel borders are actually correct. BP is therefore described as a ratio between the amount of TP boundary pixels and the amount of predicted boundary pixels, in other words, the TP plus the false positives (FP). BR and BP is thus defined as:

$$BR(B_S, B_G) = \frac{TP(B_S, B_G)}{TP(B_S, B_G) + FN(B_S, B_G)}, \tag{4.2}$$

$$BP(B_S, B_G) = \frac{TP(B_S, B_G)}{TP(B_S, B_G) + FP(B_S, B_G)}[6]. \tag{4.3}$$

A superpixel boundary pixel is determined to be true or false depending on if the pixel can be matched to a ground truth boundary pixel. This matching is done arbitrarily within a local neighborhood of size $(2\varepsilon + 1) \times (2\varepsilon + 1)$, meaning that, a superpixel boundary pixel is labeled as a true positive if there exists a ground truth boundary pixel within that area. $\varepsilon$ [55] is the radius of the area and is calculated based on the image width $w$ and height $h$ as:

$$\varepsilon = 0.0025 \cdot \sqrt{w^2 + h^2}. \tag{4.4}$$

The $TP(B_S, B_G)$ are the number of ground truth boundary pixels for which there also exist a boundary pixel in $S$ within the threshold range. The $TP(S, G)$ are obtained from

$$TP(B_S, B_G) = \sum_{i=1}^{N_G} \mathbf{1}_{j \in \mathcal{N}(i,\varepsilon)}(B_{Gi} \times B_{Sj}). \tag{4.5}$$

Where $\mathcal{N}$ represent the local neighborhood of the size $(2\varepsilon + 1) \times (2\varepsilon + 1)$ around boundary pixel $i$. The function $\mathbf{1}$ is a binary operator that returns 1 if there exist is a match between $B_{Gi}$ and $B_{Sj}$ in the neighborhood and 0 if not.

Since superpixels are over segmentations of the image there exists a lot of borders within the ground truth semantic segmentations (a car might consist of 10 superpixels). Treating all of the inner borders as false positives would amount to a non descriptive precision metric that would decrease with the amount of superpixels. For this reason superpixel labels $SL$ are introduced, $SL$ are the result of merging adjacent superpixels that have the same label. The original superpixels are labeled as the ground truth segment of which they have the largest overlap with. This removes all the inner borders between superpixels segmenting the same object. With this $FP(B_{SL}, B_G)$ can be calculated as:

$$FP(B_{SL}, B_G) = \sum_{i=1}^{N_{SL}} \left(1 - \mathbf{1}_{j \in \mathcal{N}(i,\varepsilon)}(B_{SLi} \times B_{Gj})\right). \tag{4.6}$$

$FN$ is the number of ground truth boundary pixels for which there does not exist a boundary pixel in $B_S$ within the threshold range. The sum of the $TP$ and $FN$ is equal to the number of boundary pixels in the ground truth and is obtained by

$$TP(B_S, B_G) + FN(B_{SL}, B_G) = \sum_{i=1}^{N} G_i. \tag{4.7}$$

### 4.1.3 Implementation Details

During the evaluation of the different network architectures, the networks were trained on the Cityscapes training dataset at half resolution. This was done as there at lower resolution exists more semantic boundaries for the same patch size (at full resolution a 200 by 200 pixels patch might only contain one car, if the image is down sampled, the same patch could contain 4 cars). Superpixel sampling networks learns from semantic boundaries, as such this speeds up training. From these images patches of 200 by 200 were taken at random, for each patch 100 superpixels was generated. The chosen optimizer was the Adam optimizer [69] and the final hyperparameters was a learning rate of $1^{-4}$ and a batch size of 4.

### 4.1.4 Evaluated Networks

To find the base network architecture that performs the best, a few different networks were compared. These were: SSN (the default deep network structure from [4]), U-Net [64], FCN [65], DeepLabv3 [70], and PAN [7]. These were all trained for 90 epochs before evaluating. Beyond these, a few other structures were also evaluated for comparison: an untrained SSN (random weights), a pretrained SSN trained for 45000 epochs on the BSDS500 dataset [71] and a U-Net model trained for 200 epochs. This was done to see the effects of varying amounts of training and the influence of the data on the final performance. All models were evaluated for differing amounts of superpixels generated on the Cityscapes validation dataset.

### 4.1.5 Results and Discussion

The different evaluation metrics for each of the base network structures can be seen in Figure 4.1.

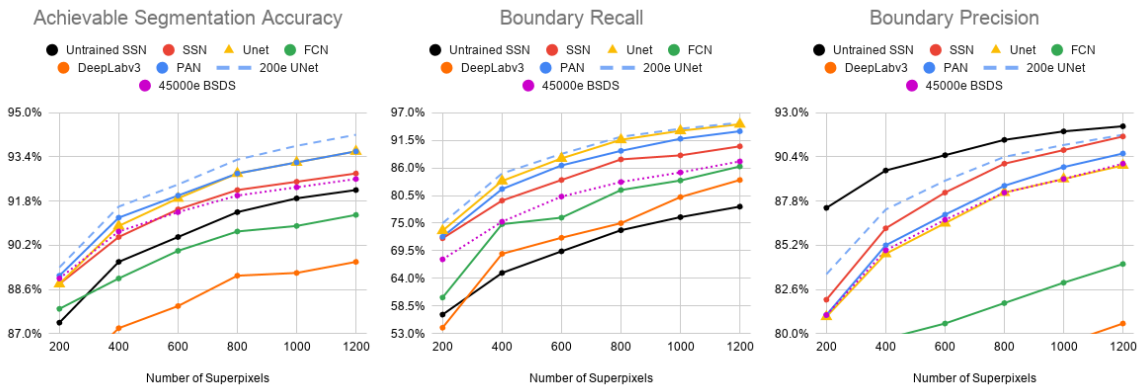


**Figure 4.1:** Performance of the different base network structures in superpixel sampling networks.

From the graphs it can be seen that FCN and DeepLabv3 perform the worst, only achieving higher results the untrained SSN in BR. The BSDS500 trained network performs worse than SSN in all metrics, despite having the same architecture and

having been trained much longer. This shows that it is more important to train on data similar to that of the evaluation than training for many epochs. Between U-Net, PAN and SSN, it is harder to decide a best architecture, with the performance varying between the three among the different measures.

When continuing training beyond 90 epochs, slight improvements are gained for the ASA-score and marginal improvements for BR, however for the BP the improvements are more substantial. As such 200e U-Net was chosen as a basis for the rest of the project as this gives the best results for both ASA and and BR while still performing competitively for BP.

Interestingly the untrained SSN performs the best for BP, this might be a result of the more contoured edges of the other methods leading to a higher number of superpixel boundary pixels. This can then yield a higher number of false positives and true positives, decreasing the boundary recall and increasing boundary precision. As such there appears to exist a trade off between precision and recall. Figure 4.2, visualizes a few results of superpixels generated from Cityscapes.

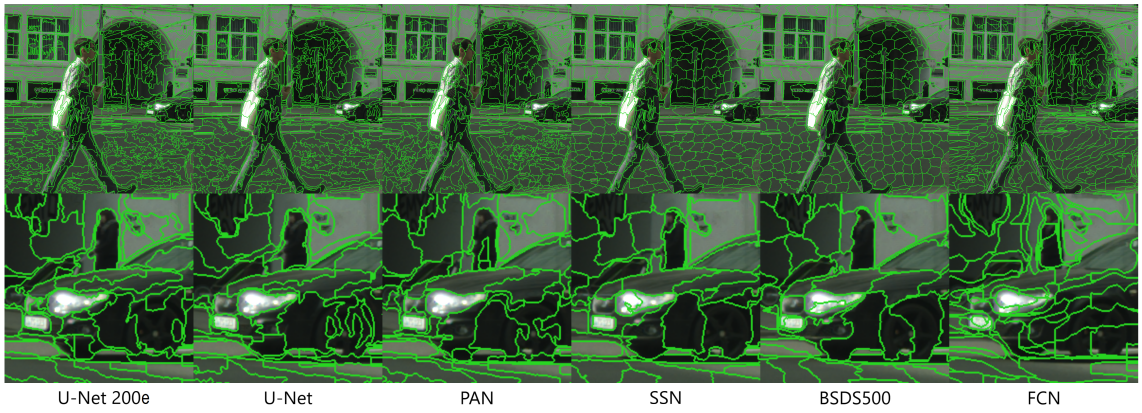


**Figure 4.2:** Visual results on Cityscapes data with 1000 superpixels.

### 4.1.6 Comparison With the State-of-the-art

The results are compared to SEAL-ERS [7] and SSN [4], as they, from what was shown in the literature study represents the state-of-the-art within the superpixel area and also evaluated on the Cityscapes dataset [1]. Only ASA-scores are compared as [7] does not report BP and uses $\varepsilon = 3$ instead of utilizing Equation 4.4. Jampani *et al.* [4] utilizes the evaluation methods from [55] which calculates $\varepsilon$ according to Equation 4.4. However, they do not report BR and BP as a function of the number of superpixels, but instead presents BP as a function of BR. In this comparison they do not merge superpixels belonging to the same ground truth object, meaning BP gets penalised for an increased number of superpixels. In this experiment superpixels were merged when calculating BP, as we are interested in seeing how superpixels can perform in an annotation task and are therefore not concerned about borders that lies inside of objects.

The ASA-scores achieved by the U-Net model trained for 200 epochs (150K iterations) does not reach those performed by the original SSN [4], however it does reach a higher ASA-score compared to SEAL-ERS [7] with a slight margin, see Table 4.1. The large discrepancy to SSN can most likely be explained by the number of training iterations, as [4] presents SEAL-ERS results 3 percentage points higher than the original paper [7]. That increase might be seen as a direct result of training as [7] trains the network for 100K iterations, while [4] trains for 500K iterations. Thus it is not unfeasible that U-Net SSN could match the performance of SSN [4] or outperform it, given the same amount of training.

**Table 4.1:** Approximate ASA-scores compared to the state-of-the-art [4], [7].

| Model | ASA-score |
|---|---|
| U-Net SSN (Ours) | 93.8% |
| SEAL-ERS [7] | 93.5% |
| SSN [4] | 96.8% |
| SEAL-ERS (as presented in [4]) | 96.5% |

## 4.2 Propagation Evaluation

To get an understanding of how well the propagation method performed it was evaluated for different jumps of frames, meaning it the propagation was done for in between frames of different spacing. This was done by calculating the Intersection over Union (IoU) between propagated superpixel annotations and ground truth superpixel annotations.

### 4.2.1 Dataset

The dataset used for evaluating the propagation is Playing for Benchmarks [72]. The dataset consist of high-resolution sequential image frames obtained from the video game Grand Theft Auto V. For all frames there also exists ground truths for a variety of items such as optical flow, environmental conditions and semantic class labels. The dataset was chosen as it has semantic class labels for every frame, thus allowing for experimentation with different frame jumps.

### 4.2.2 Evaluation Metrics

To evaluate the propagation of superpixels, the metric IoU was used. As the propagation technique is built upon linking superpixels, the input must adhere to superpixel boundaries and the output will always adhere to superpixel boundaries, as such the input to the propagation is a superpixel ground truth. The superpixel ground truth is created by labeling every superpixel as the class it has the highest intersect with, see figure Figure 4.3. As the output always adheres to superpixel boundaries, simply calculating the mIoU of the propagation with respect to the ground truth semantic segmentation would be affected by the quality of the superpixels. It is desirable to not have the propagation be impacted by the superpixels, as the propagation

performance would then hold for any superpixel generation method used. Thus the semantic ground truth for the propagation was also converted to a superpixel ground truth segmentation, see Figure 4.4.

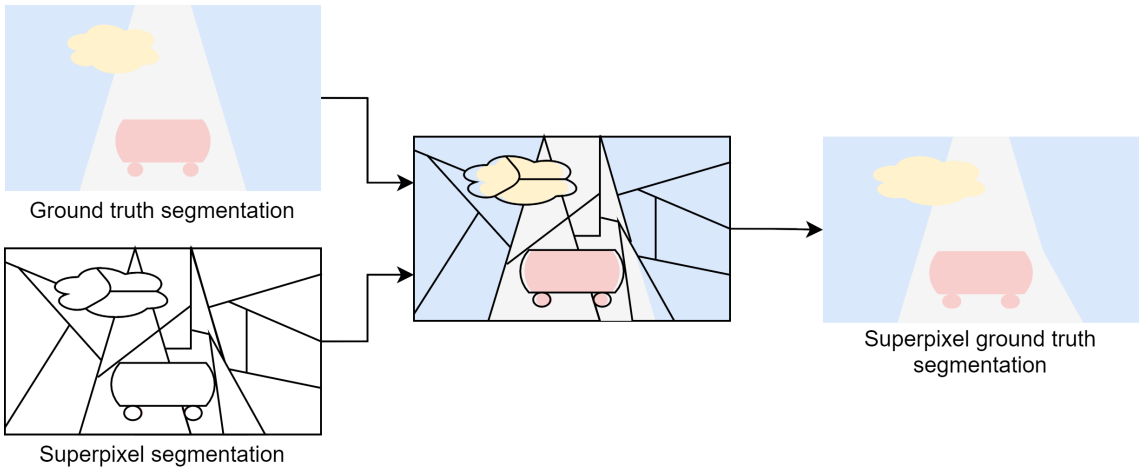


**Figure 4.3:** The creation of a superpixel ground truth.

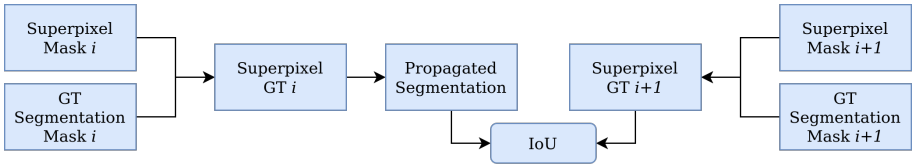


**Figure 4.4:** Evaluation of the propagation.

The mean intersection over union (mIoU), also known as the Jaccard Index, is a metric that measures the percentage of overlap between the predicted segmentation and the ground truth (Here the superpixel ground truth is used instead). For each class, the IoU is defined as the ratio of the true positives to the union between actual positives and false positives. The mIoU for multi-class segmentation is calculated by computing the average over each class-wise IoU. The equation can be written as:

$$mIoU = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c}{TP_c + FP_c + FN_c}, \tag{4.8}$$

where $C$ denotes the number of classes.

### 4.2.3 Implementation Details

Before evaluating the propagation, 1000 superpixels were generated on the sequenced data by the 200 epoch U-Net SSN model. In order to evaluate the impact of the gap in between frames on the performance of the propagation, the experiment was conducted for frame jumps 10, 40 and 100 frames.

### 4.2.4 Results and Discussion

The propagation was evaluated for three different frame jumps on the validation dataset and the resulting performance can be seen in Table 4.2. Only classes with a IoU of above 15% for 100 skipped frames are presented, as other classes are deemed to perform too poorly to be considered actual propagation and not noise. The skipped classes mostly consists of small objects such as humans, mailboxes or poles.

As can be seen in Table 4.2, there exists a relationship between the different frame jumps and the accuracy of the propagation, with a shorter frame jump leading to a higher IoU. The main decrease in performance when adjusted for the size of the jump, appears to be among the lower frame jumps, with performance then flattening out. However, there are too few data points to confirm this conclusion.

**Table 4.2:** The IoU for different classes compared to the superpixel ground truth after propagating superpixel ground truth on the Playing for benchmarks dataset [72].

| | | IoU [%] | | |
|---|---|---|---|---|
| | Frame Jump | 10 | 40 | 100 |
| Class | Buildings | 47.5 | 40.6 | 33.1 |
| | Sky | 84.3 | 73.9 | 65.7 |
| | Greenery | 32.9 | 22.3 | 16.8 |
| | Ground | 41.5 | 27.3 | 19.2 |
| | Cars | 85.1 | 65.6 | 57.6 |
| | Curbs & Sidewalks | 38.6 | 31.0 | 24.0 |
| | Road | 79.9 | 61.9 | 53.8 |
| | Trucks | 44.6 | 24.6 | 15.6 |

Intuitively, classes that appear often and usually are located in the same area of the frame, such as cars, roads and sky, performs the best. This is most likely due to as previously mentioned, they do not disappear and reappear or move much within the frame.

## 4.3 Time Study

In order to evaluate different parts of the approach in an annotation environment, a time study was conducted. A time study is an experiment where the operating time for a specific task is measured. Time studies are often used to establish operating times, evaluating different methods and finding improvements in methodology [73]. The persons participating in the study are qualified operators meaning that they possess good skills in the specific task to be studied.

### 4.3.1 Dataset and Classes

The dataset used for the study consisted of 10 images, with a jump of 100 frames from Cityscapes demo videos. The spacing of 100 frames was chosen since small jumps would be unrealistic in a real annotation workflow, as it often is desirable to have more varied scenes, allowing a network trained on the data to generalise better. From these images three classes were annotated: road, vehicles and humans. Road was defined as all spaces in which cars are supposed to drive. The vehicle class includes all vehicles that have four wheels or more: cars, vans, trucks and busses. Motorcycles and bicycles were not included. The classes were selected due to the simplicity of comprehending them as well as the variety in shape and size between them. Road covers a big part of the frame and always occurs in the approximate same place in the frame with the same color, making it a simple class. Humans are more complex in shape, color, size and occurrence. By incorporating classes with different appearance, biases in regards to shape, size and colour were minimized.

### 4.3.2 Evaluation Metrics

The evaluation metrics used in this experiment is time per frame and accuracy. The accuracy is measured using the metric mIoU with the annotations from a baseline polygonal method as ground truths. The polygon method, described in 4.3.4, is a commonly used method for creating semantic segmentation ground truths [1], [74], [75]. This method also sets a baseline for the annotation time.

### 4.3.3 Experiment Details

The study had four participants. The operators works as annotators and therefore hold knowledge about the annotation processes, annotation tools and classes. Since it was not certain that all participants had the same experience, knowledge and precision there might be varying results between the operators. This effect could be offset by having multiple operators. Before starting the study, the operators got to work with the new tool for one working day (8 hours), to learn and get familiar with the new annotation process.

The proposed method was divided into three sub-methods: superpixel annotations, superpixel annotations with propagation and superpixel annotations with manual correction. The operators annotated images using all of these methods as well as a freehand annotation method to set a baseline for quality and time.

To be able to compare the different methods, the same data was annotated in all experiments. However, this might cause the image to be easier to annotate as the annotator learns the traits of the image. To reduce this bias, the operators annotated using the methods in different order, as can be seen in Figure 4.5.

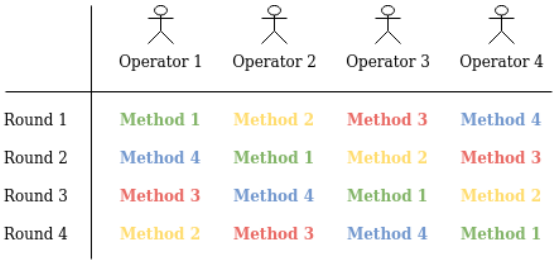


**Figure 4.5:** Annotation setup using different annotators.

### 4.3.4 Annotation Methods

To get an understanding of how the different parts of the proposed method performs, the method was divided into three sub-methods independent of each other. A baseline method for freehand annotations was also studied so annotation quality could be evaluated and annotations times could get put into context. The evaluated methods were:

**Freehand annotations (Polygons):** To get a baseline that the developed methods could be compared to, a freehand manual method for annotation was measured. The freehand method used was polygonal annotations, where each annotated object is a polygon that the user creates by clicking along the edges of the object.

**Superpixel annotations (Corrections disabled):** The experiment to annotate using only superpixels was performed to investigate how the superpixels alone had an effect on the time and accuracy of the annotation. The annotation is realized through marking superpixels as the class they should belong to. Since superpixels do not always adhere to boundaries correctly, the operators were told to annotate what they believed to be the best annotation.

**Superpixel annotations with manual correction (Corrections enabled):** As there are cases where the generated superpixels do not adhere to semantic boundaries it is desirable to correct these errors. As such the full tool allows for corrections of the superpixels. This method was evaluated by allowing the creation of new superpixels where superpixel borders do not align with object borders.

**Superpixel annotations with propagation (Propagation):** This experiment was performed to evaluate the impact of propagating superpixels in the annotation process. The annotators were first prompted with a suggested superpixel annotation, which they then corrected. The adding of superpixels was not enabled in this step, therefore it could be directly compared to superpixel annotations.

### 4.3.5 Results and Discussion

The results of the time study can be seen in Table 4.3 and Table 4.4.

**Table 4.3:** The mean time spent annotating one image.

| Method | Time [m] |
|---|---|
| Corrections Enabled | 10.9 |
| Corrections Disabled | 5.4 |
| Propagation | 5.5 |
| Polygons | 10.4 |

**Table 4.4:** The mIoU with respect to the Polygon method.

| Method | mIoU [%] |
|---|---|
| Corrections Enabled | 98.6 |
| Corrections Disabled | 97.3 |
| Propagation | 96 |

Comparing propagation and corrections disabled, it can be seen that there were no improvements from automatically generating annotation suggestions. It can even be seen that propagation performed the worst in mIoU, see Table 4.4. This can most likely be attributed to small border errors not getting found or being considered "good enough" by the annotator. However, from the experiments on propagation it could be seen that the performance varies greatly for different classes, see Table 4.2. As such propagation might be useful, if only used for the higher performing classes.

With corrections disabled, an almost 50% reduction in annotation time over the polygon method was achieved, while still reaching a mIoU of 97.3%. Even though this is not good enough to call high quality annotations, it could still be a useful tool for creating coarse annotations. Coarse annotations have proven to be a valid technique for lowering the cost of annotated data by mixing in coarse annotations when training neural networks as an alternative to data augmentation. This method has been used by the top ten contenders on the Cityscapes benchmark [76]. Yuan *et. al* [77] reports an improvement in mIoU of 1.4 percentage points when utilizing the Cityscapes coarse annotations in addition to the fine annotations.

Intuitively, the polygon and corrections enabled methods took the longest, likely due to the fact that they require more user interaction and results in higher quality annotations. The difference in average time between the two methods was not substantial enough to determine a fastest method, see Table 4.3. Looking at the distributions of the annotation times, see Figure 4.6, it can be seen that there is a wider spread for the polygon method, despite the annotators being more experienced with this method. As the annotators are less experienced with the new method there could still be room for improvement with further training.

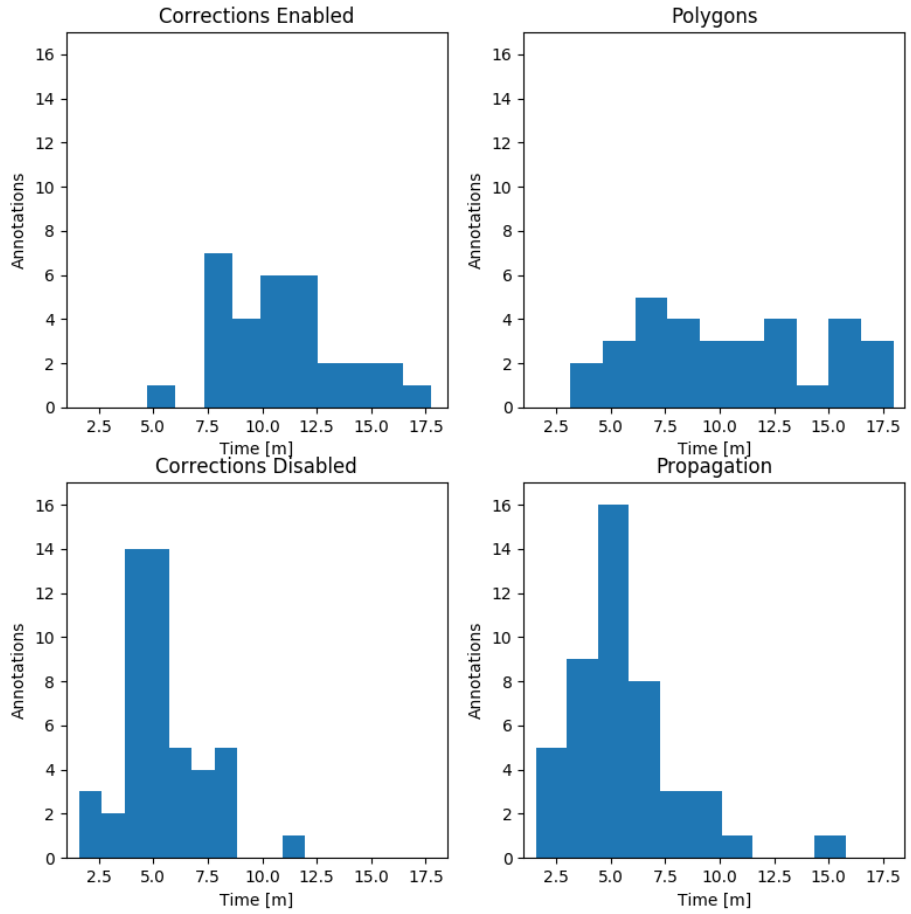


**Figure 4.6:** Histograms of the time per image for the different methods.

Looking at mIoU for corrections enabled, see Table 4.4, it is not 100%, meaning that the annotations differed from the polygon method. However, visually inspecting the annotations showed that the quality of the the annotations did not not differ, instead the mIoU could be explained by slight inconsistencies in the different annotations for the same image.

### 4.3.6 Error Sources and User Experience

After the study was completed, a discussion with the annotators was held to find out their experience of the using the new tool. A main point that was brought up, was the way superpixels is selected. The method of holding down left-click and circling to select multiple superpixels was seen as challenging. This was due to the fact panning the image was hindered during this operation, resulting in the operator having to select bigger areas in sections. The users mentioned that they would have preferred a method more similar to the polygon method when selecting superpixels.

During the discussion, it also came forward that all methods had not been used as intended. Specifically in the *Corrections enabled method*, the annotators periodically used the correction tool for more than what was intended. The tool was

sometimes used to mark entire objects as new superpixels instead of only correcting the incorrect superpixels. This was said to be because it sometimes felt easier than doing the correct selection due to the aforementioned panning issue.

The experiment was carried out remotely as a result of the global pandemic. Working remotely caused a few issues and complications, the main ones being: communication with- and oversight of- the participants. The communication issues resulted in challenges with helping the participants when they struggled with a task or came across issues with the program. This was due to the fact that it was hard to grasp what the actual problem was when it could not be observed in person. When the problem or challenge was understood, further difficulties were communicating the solution to the operator as it could not quickly or easily be demonstrated.

The oversight issue showed itself in the discussions with the users after the study. The problems brought up were problems that could have been caught during the practicing stage of the study and fixed by giving more specific information on how the correction tool should be used. The method for selecting multiple superpixels could also have been updated to select superpixels more similar to how polygons work.

# 5
# Conclusions and Future Work

The main goal of the thesis was to investigate how superpixels and video mask propagation could mesh with the annotation workflow to lower the user interaction time per image. A second goal was to investigate which type of video mask propagation could be most useful when used in the annotation process.

We developed a new tool that utilizes superpixels and propagation, allowing the user to select multiple superpixels as annotations and also add corrections so that the same level of quality as with polygon annotations can be achieved. The new tool offers a trade-off between quality and annotation time by switching the usage of the correction function. This allows for a higher throughput of images at the cost of annotation quality. A proprietary propagation technique was developed, linking superpixels in between images offline, enabling annotations to quickly be transferred at runtime.

The results of the time study should be interpreted with caution, as problems in the study have caused uncertainty in the results. Especially for the add corrections method, were the tool was not used as intended. Future work should as such focus more on the tool itself, developing it more iteratively with supervised trial runs, and better usage guidelines. Thus, faults could be found and better interaction methods with the superpixels developed. Based on that, a new time study should be conducted.

Further, the value of the lower quality annotations, obtained by only annotating with superpixels should be investigated. Hence, a better understanding can be achieved of how much value they add for training networks used for semantic segmentation in relation to the time it takes to create them. This would also need to be compared to the performance gained if instead the same time was spent annotating at full quality.

# Bibliography


[1] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[2] G. Neuhold, T. Ollmann, S. Rota Bulo, and P. Kontschieder, "The mapillary vistas dataset for semantic understanding of street scenes", in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4990–4999.

[3] X. Ren and J. Malik, "Learning a classification model for segmentation", in *null*, IEEE, 2003, p. 10.

[4] V. Jampani, D. Sun, M.-Y. Liu, M.-H. Yang, and J. Kautz, "Superpixel sampling networks", in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 352–368.

[5] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods", *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012.

[6] R. Achanta and S. Susstrunk, "Superpixels and polygons using simple non-iterative clustering", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4651–4660.

[7] W.-C. Tu, M.-Y. Liu, V. Jampani, D. Sun, S.-Y. Chien, M.-H. Yang, and J. Kautz, "Learning superpixels with segmentation-aware affinity loss", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 568–576.

[8] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa, "Entropy rate superpixel segmentation", in *CVPR 2011*, IEEE, 2011, pp. 2097–2104.

[9] Y.-H. Tsai, M.-H. Yang, and M. J. Black, "Video segmentation via object flow", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3899–3908.

[10] V. Jampani, R. Gadde, and P. V. Gehler, "Video propagation networks", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 451–461.

[11] S. Caelles, K.-K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool, "One-shot video object segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 221–230.

[12] A. Kundu, Y. Li, F. Dellaert, F. Li, and J. M. Rehg, "Joint semantic segmentation and 3d reconstruction from monocular video", in *European Conference on Computer Vision*, Springer, 2014, pp. 703–718.

[13] L. Porzi, M. Hofinger, I. Ruiz, J. Serrat, S. R. Bulò, and P. Kontschieder, "Learning multi-object tracking and segmentation from automatic annotations", *arXiv preprint arXiv:1912.02096*, 2019.

[14] B. L. Price, B. S. Morse, and S. Cohen, "Livecut: Learning-based interactive video segmentation by evaluation of multiple propagated cues", in *2009 IEEE 12th International Conference on Computer Vision*, IEEE, 2009, pp. 779–786.

[15] P. Voigtlaender, Y. Chai, F. Schroff, H. Adam, B. Leibe, and L.-C. Chen, "Feelvos: Fast end-to-end embedding learning for video object segmentation", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9481–9490.

[16] H. Lu, X. Zhang, J. Qi, N. Tong, X. Ruan, and M.-H. Yang, "Co-bootstrapping saliency", *IEEE Transactions on Image Processing*, vol. 26, no. 1, pp. 414–425, 2016.

[17] W. Zhu, S. Liang, Y. Wei, and J. Sun, "Saliency optimization from robust background detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 2814–2821.

[18] K. Yamaguchi, M. H. Kiapour, L. E. Ortiz, and T. L. Berg, "Parsing clothing in fashion photographs", in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2012, pp. 3570–3577.

[19] Y.-J. Gong and Y. Zhou, "Differential evolutionary superpixel segmentation", *IEEE Transactions on Image Processing*, vol. 27, no. 3, pp. 1390–1404, 2017.

[20] Nilsjohan, *The principle of the cielab colour space*, `https://commons.wikimedia.org/w/index.php?curid=35029428`, CC BY-SA 4.0.

[21] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[22] B. K. Horn and B. G. Schunck, "Determining optical flow", in *Techniques and Applications of Image Understanding*, International Society for Optics and Photonics, vol. 281, 1981, pp. 319–331.

[23] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion", in *Scandinavian conference on Image analysis*, Springer, 2003, pp. 363–370.

[24] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision", 1981.

[25] J. Wulff and M. J. Black, "Efficient sparse-to-dense optical flow estimation using a learned basis and layers", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 120–130.

[26] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2758–2766.

[27] P. Liu, M. Lyu, I. King, and J. Xu, "Selflow: Self-supervised learning of optical flow", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4571–4580.

[28] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation",

in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[29] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i", *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[30] M. Milford and G. Wyeth, "Persistent navigation and mapping using a biologically inspired slam system", *The International Journal of Robotics Research*, vol. 29, no. 9, pp. 1131–1153, 2010.

[31] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem", *IEEE Transactions on robotics and automation*, vol. 17, no. 3, pp. 229–241, 2001.

[32] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam", *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[33] J. Engel, J. Sturm, and D. Cremers, "Scale-aware navigation of a low-cost quadrocopter with a monocular camera", *Robotics and Autonomous Systems*, vol. 62, no. 11, pp. 1646–1656, 2014.

[34] J. Yao, S. Fidler, and R. Urtasun, "Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation", in *2012 IEEE conference on computer vision and pattern recognition*, IEEE, 2012, pp. 702–709.

[35] M. Naseer, S. Khan, and F. Porikli, "Indoor scene understanding in 2.5/3d for autonomous agents: A survey", *IEEE Access*, vol. 7, pp. 1859–1887, 2018.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[37] X. Liu, Z. Deng, and Y. Yang, "Recent progress in semantic image segmentation", *Artificial Intelligence Review*, vol. 52, no. 2, pp. 1089–1106, 2019.

[38] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, "Mots: Multi-object tracking and segmentation", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7942–7951.

[39] Z. Yin, T. Darrell, and F. Yu, "Hierarchical discrete distribution decomposition for match density estimation", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6044–6053.

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

[41] K. Hao. (2018). What is machine learning? machine-learning algorithms find and apply patterns in data. and they pretty much run the world., [Online]. Available: `https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/d` (visited on 05/20/2020).

[42] X. J. Zhu, "Semi-supervised learning literature survey", University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2005.

[43] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]", *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[44] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[45] G. Hanrahan, *Artificial neural networks in biological and environmental analysis*. CRC Press, 2011.

[46] R. Paleja. (2019). Single Neuron Training how does general optimization fit into neural networks?, [Online]. Available: `https://towardsdatascience.com/single-neuron-training-3fc7f84d67d` (visited on 03/25/2020).

[47] *Common loss functions in machine learning*, `https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23`, Accessed: 2020-02-28.

[48] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[49] *Stanford cs class: Convolutional neural networks for visual recognition. lecture 2*, `http://cs231n.github.io/neural-networks-1/`, Accessed: 2020-02-26.

[50] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *arXiv preprint arXiv:1502.03167*, 2015.

[51] *Stanford cs class: Convolutional neural networks for visual recognition. lecture 2*, `http://cs231n.github.io/neural-networks-2/`, Accessed: 2020-02-26.

[52] P. Thévenaz, T. Blu, and M. Unser, "Image interpolation and resampling", *Handbook of medical imaging, processing and analysis*, vol. 1, no. 1, pp. 393–420, 2000.

[53] B. Fulkerson, A. Vedaldi, and S. Soatto, "Class segmentation and object localization with superpixel neighborhoods", in *2009 IEEE 12th international conference on computer vision*, IEEE, 2009, pp. 670–677.

[54] C. L. Zitnick and S. B. Kang, "Stereo for image-based rendering using image over-segmentation", *International Journal of Computer Vision*, vol. 75, no. 1, pp. 49–65, 2007.

[55] D. Stutz, A. Hermans, and B. Leibe, "Superpixels: An evaluation of the state-of-the-art", *Computer Vision and Image Understanding*, vol. 166, pp. 1–27, 2018.

[56] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow", *International journal of computer vision*, vol. 92, no. 1, pp. 1–31, 2011.

[57] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations", *Computer vision, graphics, and image processing*, vol. 39, no. 3, pp. 355–368, 1987.

[58] J. M. Gauch, "Investigations of image contrast space defined by variations on histogram equalization", *CVGIP: Graphical Models and Image Processing*, vol. 54, no. 4, pp. 269–280, 1992.

[59] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time", in *2011 IEEE intelligent vehicles symposium (IV)*, Ieee, 2011, pp. 963–968.

[60] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof, "Anisotropic huber-l1 optical flow.", in *BMVC*, vol. 1, 2009, p. 3.

[61] S. Hauberg, A. Feragen, and M. J. Black, "Grassmann averages for scalable robust pca", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3810–3817.

[62] N. Jamil, T. M. T. Sembok, and Z. A. Bakar, "Noise removal and enhancement of binary images using morphological operations", in *2008 International Symposium on Information Technology*, IEEE, vol. 4, 2008, pp. 1–6.

[63] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Hypermedia image processing reference", *Department of Artificial Intelligence, University of Edinburg, available on http://www. cee. hw. uk*, 1996.

[64] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

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

[66] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[67] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving video database with scalable annotation tooling", *arXiv preprint arXiv:1805.04687*, 2018.

[68] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The apolloscape open dataset for autonomous driving and its application", *arXiv preprint arXiv:1803.06184*, 2018.

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

[70] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation", *arXiv preprint arXiv:1706.05587*, 2017.

[71] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics", in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, IEEE, vol. 2, 2001, pp. 416–423.

[72] S. R. Richter, Z. Hayder, and V. Koltun, "Playing for benchmarks", in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2213–2222.

[73] H. B. Maynard and K. B. Zandin, *Maynard's industrial engineering handbook*, Sirsi) i9780070411029. 2001.

[74] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: A database and web-based tool for image annotation", *International journal of computer vision*, vol. 77, no. 1-3, pp. 157–173, 2008.

[75] K. Wada, *labelme: Image Polygonal Annotation with Python*, `https://github.com/wkentaro/labelme`, 2016.

[76] Y. Zhu, K. Sapra, F. A. Reda, K. J. Shih, S. Newsam, A. Tao, and B. Catanzaro, "Improving semantic segmentation via video propagation and label relaxation", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8856–8865.

[77] Y. Yuan, X. Chen, and J. Wang, "Object-contextual representations for semantic segmentation", *arXiv preprint arXiv:1909.11065*, 2019.

# A

# Annotation tool

A few images describing the functionality of the annotation tool can be seen below, all of the images shown in the tool are taken from the Cityscapes dataset [1].


**Figure A.1:** Overview of the proposed annotation tool.

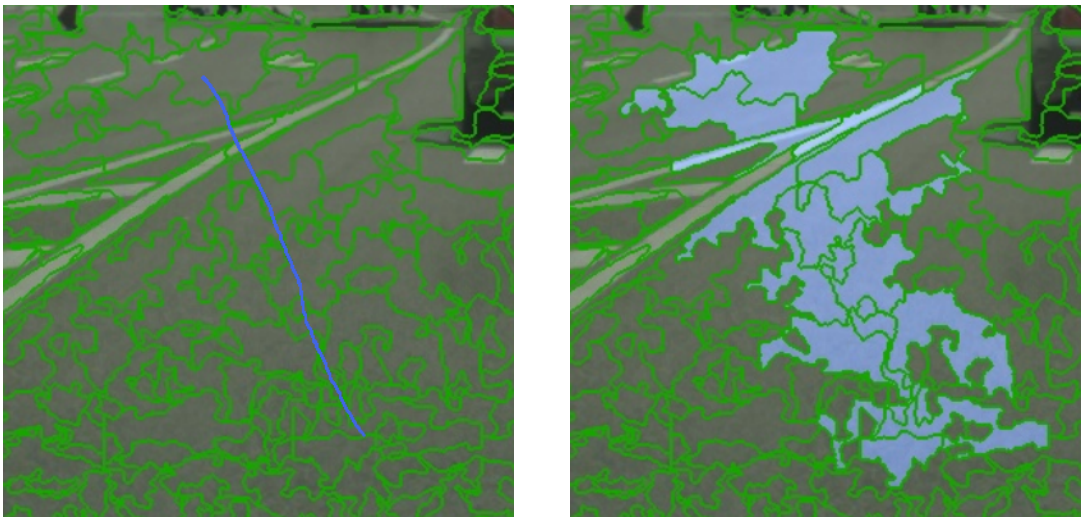

**Figure A.2:** The process of selecting multiple superpixels as an annotation. Here the user is left-clicking and dragging the mouse to select superpixels under the drawn path.

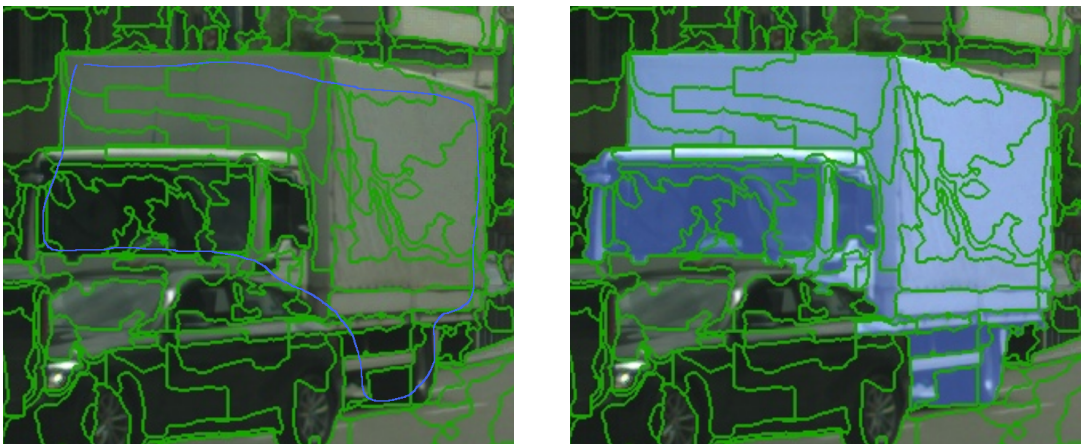

**Figure A.3:** The process of selecting superpixels within an area as an annotation. Here the user is left-clicking and dragging the mouse to encircle the superpixels to be selected.

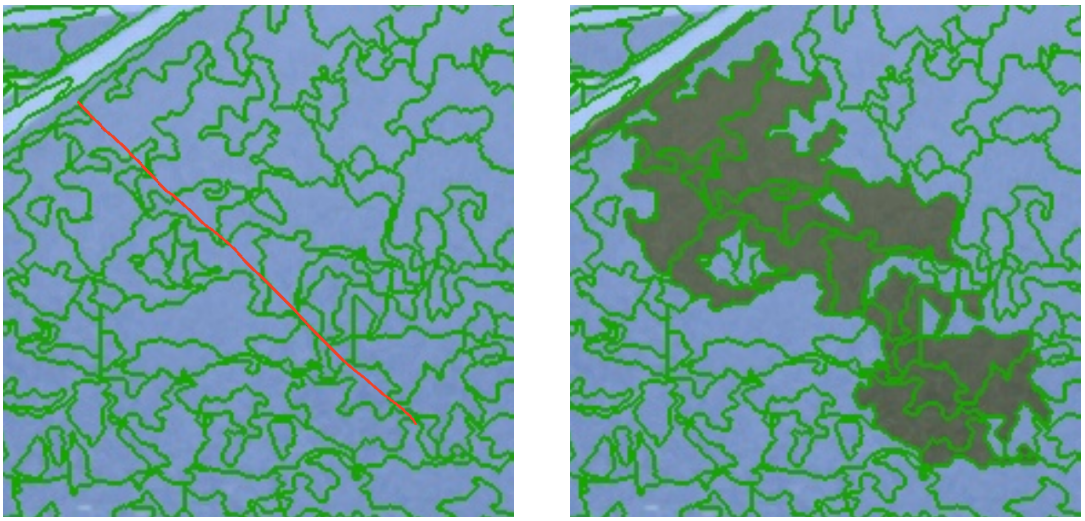

**Figure A.4:** The process of deselecting multiple superpixels as an annotation. Here the user is right-clicking and dragging the mouse to deselect superpixels under the drawn path.

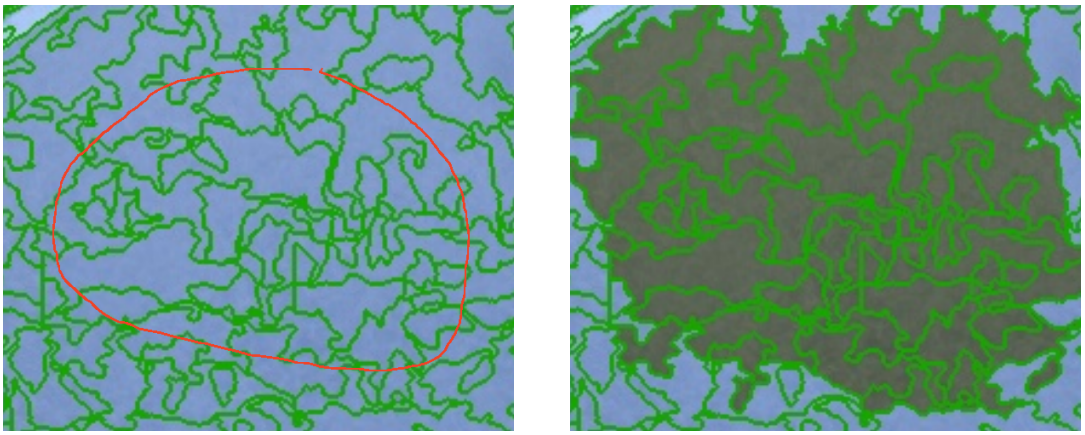

**Figure A.5:** The process of deselecting superpixels within an area. Here the user is right-clicking and dragging the mouse to encircle the superpixels to be deselected.

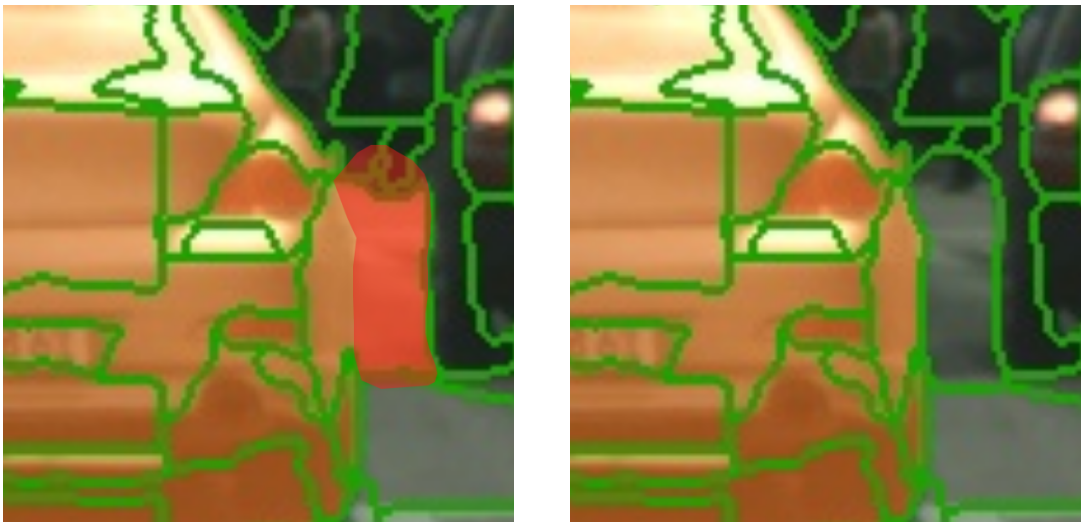

**Figure A.6:** The process of adding new superpixels to correct areas where the superpixel performance is not satisfactory. Here the user is left-clicking several times to create each corner of a polygon, the polygon is then finalised by a right-click which adds the polygon as a new superpixel.