Augmenting Labelling For Semantic Image Segmentation Networks

Presented by:
Josh Stein

Prepared for:
Dr. P. Amayo Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfillment of the academic requirements for a Bachelor of Science degree in Electrical and Computer Engineering

October 14, 2019

Plagiarism Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:.....
Josh Stein

Date:.....

Acknowledgments

I would like to thank my supervisor, Dr. Paul Amayo, for the generous time and support he has given me throughout this project. His expertise and assistance have been invaluable. I appreciate him coming all the way from Oxford just to supervise my project [1].

I would also like to thank my family for supporting me and providing me the opportunity to study.

[1] That was a joke. I am sincerely grateful to have had Dr. Amayo as my supervisor.

Abstract

For many computer vision tasks deep learning provides state-of-the-art results. However, learning (especially supervised learning) requires vast amounts of labelled data. Generating these data are non-trivial, and usually require laborious human effort to manually annotate images.

This project aims to automatically generate image segmentations (i.e. data labels) by using LiDAR sensors, with applications for autonomous vehicles. The main approach taken uses hierarchical clustering to determine important segmentations. An interactive GUI was created to refine and improve segmentations. An additional alternative approach made use of a pre-trained neural network to determine object bounding boxes, followed by hierarchical clustering refinements. All approaches were developed and tested on the KITTI dataset [1].

Best results achieved utilized the interactive GUI, achieving a mean IoU of 72.97%, with an approximate processing time of 22.08 seconds per image (163 images per hour). Current state-of-the-art results for KITTI semantic segmentation utilize deep neural networks, achieving a mean IoU of 72.87%, while requiring significantly less time [2]. Faster results can be achieved by sampling LiDAR data and/or restricting use of the interactive GUI. This can improve timing performance significantly, theoretically achieving a throughput of 20 000 images per hour with a mean IoU $\approx 41\%$.

These data indicate that interactive approaches to image segmentation yield near comparable results to start-of-the-art neural network approaches, albeit with increased processing time. Purely heuristic approaches, without interactivity, lose precision but may nonetheless prove useful as a reference for human annotators. Several different directions for future work exist - particularly important areas could focus on combining multiple segmentation techniques with sampled data, and/or expanding the sophistication of the GUI.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background to the study

Over the past few decades, neural networks have been applied to solve a variety of 'difficult' computational problems. 'Difficult' in this sense refers to there being no clear algorithmic strategy. Classic problems include classification (handwriting recognition, facial recognition, spam vs not-spam, etc.), clustering (finding similar images/videos/sounds, detecting anomalies, etc.) and pattern recognition tasks.

The power of neural networks and artificial intelligence is transforming our world. However, a neural network requires significant data to train - without which it will fail to learn an objective function, or learn incorrectly.

Generating these data are a resource intense task - data needs to be collected, annotated and classified. In particular, annotation is frequently laborious, requiring hours of human effort. Strategies such as CAPTCHAs, Amazon's Mechanical Turk [4] and private companies dedicated to annotation (e.g. Scale AI [5]) exist in an effort to streamline the annotation process. However, these strategies only provide partial solutions, and still require significant human input. From an economic perspective, this is one of the most expensive parts of the neural network training pipeline.

1.2 Objectives of this study

This study aims to investigate the power of augmenting data annotations for use in training neural networks, specifically with regard to image semantics for self-driving vehicles. Ideally, a near ground-truth pixel-wise segmentation will be achieved, without interactive input. More realistically, this study aims to segment objects of interest that could be used as a reference for annotation work.

1.2.1 Problems to be investigated

The primary problem is to determine whether image segmentations can be determined computationally, without human input. Further, it is important to determine whether the problem can be solved heuristically - that is, without making use of neural networks.

1.2.2 Purpose of the study

Huge effort is being put into developing self-driving cars. One fundamental problem is how to semantically segment an image (i.e. how to ensure that self-driving cars can use visual or other inputs to make sense of the world around them). Neural networks offer the solution - feeding data of people, cars, or other inputs allows for learning of what these things are, thereby allowing networks to segmentate future instances of similar data.

However, learning neural networks segmentation requires vast pre-annotated data, with good ground-truth segmentations. On average, one human can label one image in approximately one hour [6]. In addition, each annotator needs to be trained for the task of labeling per-pixel masks.

Current annotations are performed by hand, either doing a pixel-wise segmentation, or using interactive techniques (e.g. graph cuts) to generate segmentations.

This study is aimed at supplementing the annotation process by automating segmentation labelling. Specifically, this study aims to use LiDAR data to heuristically generate segmentations within camera images. LiDAR (Light Detection and Ranging) is a surveying method that makes use of light pulses, transmitted and received, to measure distance to a target. It is comparable to radar, although it uses shorter wavelengths, thereby achieving better resolution. Due to its high precision and impressive range (most modern sensors offer detection further than 100m), it is being used by autonomous driving leaders, including Waymo, Uber and Toyota [7].

1.3 Scope and Limitations

The scope of the generated segmentations is limited to self-driving vehicles. A core part of the strategy implemented depends on LiDAR data - thus, we are limited to datasets that contain LiDAR information (here, the KITTI dataset [1] is used). This is somewhat contentious - companies such as Tesla have been outspoken against the use of LiDAR, arguing that it is unnecessary for autonomous driving[1] [9]. Other key leaders in the self-driving car industry are strong proponents for LiDAR (Waymo, Uber, Ford, GM Cruise)

[1]More recently, Tesla have been seen experimenting with a Velodyne LiDAR sensor [8]

[7].

A number of additional limitations are relevant. First, frame integration (discussed further in methodology) requires odometry data. Although this could be determined (indeed, determining odometry is a core challenge in autonomous vehicles), here ground-truth odometry data is used, as provided in the KITTI dataset.

Second, this study is limited to segmentation. That is, there is no classification in the pipeline - data is simply segmented, without an 'understanding' of what that data is.

Finally, the computer used throughout testing was an 8 core Intel i7-8565U at 1.80 Ghz. This limits the speed at which the algorithm can run, and influences all timing data.

1.4 Plan of development

This report begins with a literature review, focused on various traditional and contemporary segmentation techniques. It then outlines a method of segmentation that harnesses LiDAR data. Following this, results are examined based on comparison with ground-truth data from the KITTI dataset. Finally, conclusions are drawn and recommendations are made.

Chapter 2

Literature Review

Semantic image segmentation is a rich field with significant challenges. Part of the difficulty lies in determining what a correct segmentation should produce. That is, there are several segmentations (within a single image) that could be correct. Deciding *how* to segment an image is therefore a difficult challenge, with no one criterion (colour, texture, brightness, etc.) being a clear choice.

Another difficulty is that images are inherently hierarchical. That is, some objects can also be a part of another object. This has lead to some segmentation approaches forming a hierarchy of tree-like structures, corresponding to hierarchical partitions, rather than a 'flat' partition.

Due to the inherent difficulty of segmentation, there have been many different methods and approaches, with varying degrees of success. Traditional segmentation (prior to 2000) was approached using direct techniques. From 2000 to 2010, research shifted towards graph-based segmentation, classification using SVMs, and clustering techniques. In 2012, AlexNet [10] performed remarkably well in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [11] using a Convolutional Neural Network (CNN), leading to a surge of research into CNNs. Over the past few years, Deep Neural Networks (DNNs or DCNNs) have become popular for a variety of computer vision tasks, including semantic segmentation. These DNNs have performed excellently on image classification tasks, achieving super-human performance [12]. However, deeper models use more pooling layers, which cause a trade-off between classification and localization accuracy (that is, as one moves deeper into a network more spatial information is lost), causing difficulty in segmentation tasks. Recently, several solutions have evolved to achieve spatial invariance [13] [14] [15]. In this literature review I focus on some of the traditional methods as well as more contemporary neural network based approaches.

2.1 Traditional approaches

2.1.1 Thresholding [3]

Thresholding is the simplest method for segmentation. An image is segmented based on its pixel intensities (values).

Global thresholding applies the same threshold everywhere:

$$p(x,y) = \begin{cases} b_0 & p(x,y) \leq t \\ b_1 & p(x,y) > t \end{cases}$$

where $p(x,y)$ is a pixel at some x, y location, b_o and b_1 are new desired intensities and t is some threshold. Multiple thresholding uses more than two threshold levels to separate the image into more than two segments.

Local thresholding applies different thresholds to different parts of an image. Adaptive thresholding varies the value of t according to the local neighbourhood of pixels around (x,y). Automatic thresholding allows for the t value to be determined heuristically. For example, Otsu's method [16] finds the threshold by minimizing intra-class intensity variance, an example of which can be seen in figure 2.1.

(a) Original image. (b) Gray-scale image. (c) Thresholding using Otsu's method.

Figure 2.1: Using Otsu's method to segment an image.

Thresholding proves useful in images that have a natural high contrast, such as grey-scale images. Thus, thresholding has proved useful in analyzing CT scans and MRIs.

2.1.2 Edge detection

Edge detection uses edges for segmentation. That is, edges are assumed to be object boundaries, which can then be used for segmentation. Naive implementations compute

the gradients vectors of pixels. A gradient vector simply computes the changes along the x and y directions of adjacent pixels respectively (the derivatives in horizontal and vertical directions). This can other be done explicitly (i.e. for pixel p calculate $(p+1)-(p-1)$ to determine the x-direction gradient). Alternatively, an equivalent result can be achieved by convolving a gradient kernel with the image. Various gradient kernels exist - commonly used kernels include the Prewitt [17], Sobel [18] and Laplacian operators. Figure 2.2 shows edge detection using different kernels.

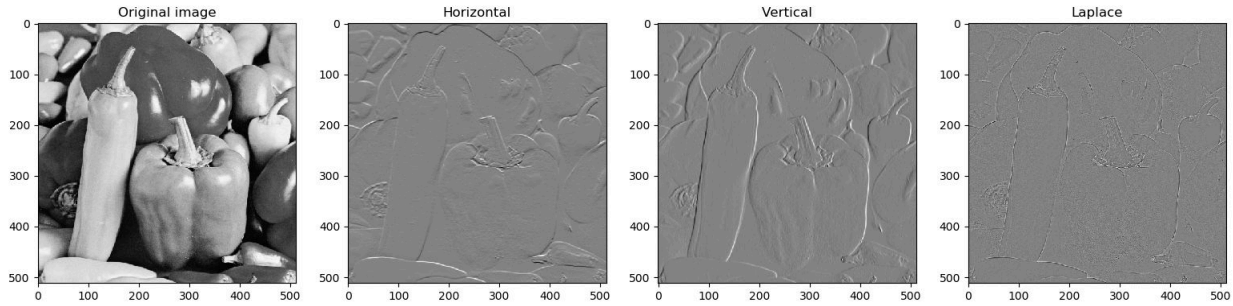


Figure 2.2: Edge detection using horizontal, vertical and Laplacian kernels.

2.1.3 Clustering

Clustering techniques attempt to group similar data into unique clusters. This can prove useful for segmentation, especially if the image undergoing segmentation has distinctive features (e.g. clear colour separation).

K-means clustering [19] attempts to cluster n points into k clusters by minimizing the sum of squares within each cluster. The algorithm is well known and is not repeated here.

K-means clustering is extremely efficient. However, it has some drawbacks: namely, one needs to supply the parameter k before clustering and convergence may not always produce correct or expected results.

Fuzzy clustering (or soft k-means) allows each data point to belong to more than one cluster. The most popular algorithm is Fuzzy C-means (FCM) [20]. By adding a parameter of 'fuzziness', the threshold between clusters becomes less distinct. This allows segmentation to be more robust towards noise, shadowing and variations in camera, but it otherwise has the same drawbacks of hard k-means. Chuang *et al.* [21] introduce a FCM algorithm that takes into account spatial information (i.e. the idea that neighbouring pixels are highly correlated, and thereby should belong to the same cluster). This spatial FCM further reduces the effects of noise and biases segmentation towards homogenous clustering.

The mean shift [22] [23] is a clustering algorithm that does not depend on specifying

the number of clusters. It is based on Kernel Density Estimation (KDE), and works by finding the modes (maxima) of a density function. The mean shift is advantageous as one only supplies the *bandwidth*, a measure of spread of the kernel. However, it is significantly slower than k-means.

Several authors have combined deep learning with clustering [24] [25]. However, this combination has led to degenerate solutions (where a single cluster dominates or some clusters disappear). Recently, Ji *et al.* [26] introduced invariant information clustering (IIC) - a novel method for clustering that learns a neural network to maximize mutual information between pairs of images; the details of which are beyond the scope of this review. IIC is robust to degeneracy, and avoids the problem of distractor classes by training with an auxiliary output layer that learns to over-cluster. IIC is currently state-of-the-art for unsupervised semantic segmentation.

2.1.4 Graph-based approaches

Graph based approaches to segmentation treat an image as a mathematical undirected graph. That is, each pixel is treated as a node/vertex in the graph, and the edges are formed between every pair of nodes. The weight of each edge correspond to some function of similarity between nodes. This similarity could be based on brightness, colour, texture, motion, etc. A *cut* is defined as:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

where A and B are disjoint sets of the graph, and $w(u, v)$ is the weight between nodes u and v. A cut can then be made in the graph according to some criterion. Cutting effectively segments the graph (i.e. image).

Wu and Leahy [27] use *minimum cuts* for segmentation. That is, if a cut is defined as the sum of all weights in the graph, then a minimum cut simply finds the cut that minimizes all the weights, as shown in figure 2.3. However, this method is biased towards cutting small sets of isolated nodes. Intuitively, small isolated nodes contribute a large weight, which can be reduced by cutting them out of the graph, as illustrated in figure 2.4.

To address this bias, Shi and Malik [28] introduce the idea of *normalized cuts*:

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}$$

where $assoc(A, V) = \sum_{u \in A, t \in V} w(u, t)$ is the total connection from nodes in A to all nodes in the graph.

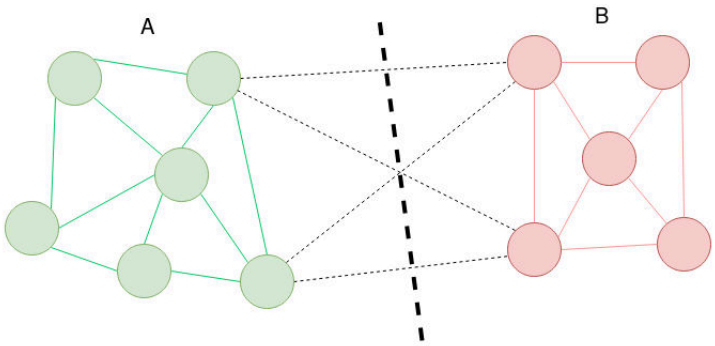


Figure 2.3: An example using minimum cuts to segment a graph.

This is a new measure of association between nodes. By computing the normalized cut as a fraction of all node connections it gives the node a frame of context within the graph, and thereby uses global impressions, rather than just local features. This allows for better segmentation over minimum cuts, as seen in figure 2.4. Unfortunately, using normalized cuts for segmentation is an NP hard problem, as outlined by the authors. They give approximations that allow for computation with some error. However, this method of segmentation is very slow ($O(mn)$, where n is the number of pixels and m is the number of steps required by an Lanczos eigensolver - computing the optimal partition requires solving an eigendecomposition problem) and does not scale well to larger images.

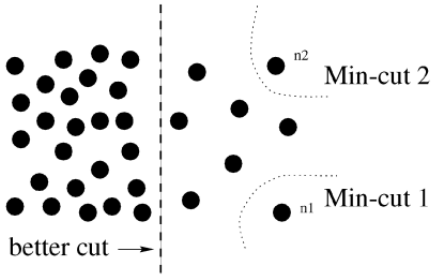


Figure 2.4: Normalized cut, yielding a less biased cut than a min-cut [28].

In 2004, Felzenswalb and Huttenlocher [29] extended the idea of using a graph-based approach, introducing a greedy algorithm that runs in $O(n\log n)$ time, where n is the number of image pixels. This is much faster than previous approaches, and allows for segmentation to be practically run on video. In addition, this technique respects gradients within images, as well as regions of wildly differing intensities.

The greedy algorithm adaptively adjusts segmentation based on variability in intensity between regions of an image. As before, a graph is constructed from the image. However, this graph is then segmented into components, where a component is a region of the graph within which all pixels have a high level of similarity. Pixels in different components have high levels of dissimilarity. The weights of adjacent pixels are used as a measure of similarity. Using a bottom-up approach, the algorithm uses a predicate to determine if

there is a boundary between two components. If there is not (high similarity between components) they will be merged into one larger component. The predicate compares the internal differences *within* a component to differences *between* components. That is, components are compared internally and to one another to determine their similarity, and subsequently whether or not they should be merged. The predicate is defined as having a threshold that is determined by the size of the components being compared. This allows the algorithm to be adaptive.

2.1.5 Variational methods

Variational methods approach image segmentation as a mathematically rigorous problem - that is, they define an image as a continuous function, from which segmentations can be found via optimization. This function is known as the *energy* function. The advantage of this is a clearly defined cost function, which allows for optimization to find the best segmentations.

Active contours (snakes)

Active contours [30] was the first variational method applied to image segmentation [31]. It is an edge-based method (it 'locks onto' nearby edges, in such a way creating segmentations), which aims to minimize the following energy function [31]:

$$E(C) = E_{ext}(C) + E_{int}(C)$$

with an external energy (a data term):

$$E_{ext}(C) = -\int_0^1 |\nabla I(C(s))|^2$$

and an internal energy (a regularization term):

$$E_{int}(C) = \int_0^1 \{\frac{\alpha}{2}|(C_s(S))|^2 + \frac{\beta}{2}|C_{ss}(S)|^2\}ds$$

Here, I is the input image and C is some parametric curve. C_s and C_{ss} denote the first and second derivatives of the parametric curve with regard to its parameter, s. α and β are user-supplied parameters which control the amount of regularization.

The external energy term attempts to maximize the gradient (in order to minimize the energy). At each position on the curve the gradient is calculated, squared and then negated. A larger gradient will therefore yield a smaller energy. This term is, in a sense,

Figure 2.5: Using the snakes algorithm to find the closest contour [32].

a measure of 'edge strength' along a curve.

The internal energy term (regularization term) penalizes for lengths of segmentations. The first term penalizes the *elastic length* (or stretch) of the curve - it is smaller for shorter curves. The second term penalizes the *stiffness* (or curvature) of the curve - it is smaller for straighter curves, and larger for more 'wiggly' curves.

Minimizing the total energy (usually done with gradient descent, graph cuts or convex relaxation) results in snakes that are short and stiff. An example of using active contours can be seen in figure 2.5.

Mumford-Shah

The Mumford-Shah functional [33] computes a piece-wise smooth approximation, u, to a given input image, I. That is, an input image is decomposed into a set of regions:

$$\Omega = \Omega_1 \cup \Omega_2 \cup ... \cup \Omega_n$$

where Ω is the domain of the image, such that the image is smooth within each Ω_i and varies discontinuously (or rapidly) across most of the boundary K between different Ω_i.

The energy function to be minimized is:

$$E(u, K) = \mu^2 \int_{\Omega} (u - I)^2 dx + \int_{\Omega/K} |\nabla u|^2 dx + \nu |K|$$

where u is an approximation to the image and K is a discontinuity set (a boundary that allows discontinuities).

The first term is a data term, which computes the error of approximation between u and the image, I, using least-squares. This asks that u approximates I.

The second integral is simply the gradient of the image squared. This term tends to over-smooth edges - that is, by minimizing energy (i.e. minimizing this gradient term),

smaller (smoother) gradients are preferred. This causes u, and ultimately I, to not vary much within each Ω_i.

The third term penalizes the lengths of the boundaries of discontinuity. This term is required because K allows for discontinuities. At these discontinuous positions, the gradient is not penalized (i.e. K is excluded from Ω). This can cause the problem of the algorithm labelling all pixels as boundaries (thereby excluding all points, ultimately minimizing energy). The third term addresses this problem by regularizing the second term, allowing for *some* discontinuities. As the number of discontinues increases, the length of the boundary also increases, which will subsequently be penalized. This has the effect of limiting the number of discontinuities, thereby causing the boundaries to be as small as possible.

The result of minimizing the Mumford-Shah functional leads to the formation of a piecewise smooth approximation to the input image.

Combining colour with space

In some senses, the graph-based methods discussed above use *location* information as a basis for segmentation. Other techniques (e.g. Mumford-Shah, thresholding) use *colour* as a basis, attempting to minimize colour variance within segments. Nieuewenhuis and Cremers [34] introduce a method that combines spatial locations and colour similarity of user scribbles (inputs), bridging the gap between purely colour-based and purely location-based approaches.

First, space-variant colour distributions are estimated from user inputs. This is done by maximizing the conditional probability:

$$\underset{u}{\operatorname{argmax}}\, \mathrm{P}(u|I) = \underset{u}{\operatorname{argmax}}\, \mathrm{P}(I|u)\mathrm{P}(u)$$

where I is the input image and u is a labelling indicating which n regions each pixel belongs to (i.e. a labelled approximation to the image). A kernel density estimator is then used to estimate the conditional probability $\hat{\mathrm{P}}(I(x), x|u(x) = i)$, which denotes the joint probability for observing a colour value I at location x given that x is part of region Ω_i. The details of this are beyond the scope of this literature review. Once these distributions have been constructed the following variational energy function is minimized:

$$E(\Omega_1, ..., \Omega_n) = \frac{1}{2}\sum_{i=1}^{n} Per_g(\Omega_i) + \lambda \sum_{i=1}^{n} \int_{\Omega_i} f_i(x)dx$$

with

$$f_i(x) = -log\hat{\mathrm{P}}(I(x), x|u(x) = i)$$

Figure 2.6: Segmentation using spatially varying colour distributions [34]. From left to right: the original image with foreground and background pixels marked by the user, a segmentation only using colour information, a segmentation only using location of input information, combining both approaches.

Here, $Per_g(\Omega_i)$ is a regularization term, and denotes the perimeter of each region Ω_i. This causes the minimization to prefer shorter boundaries. The function $f_i(x)$ is the dataterm, which is simply the negative logarithm of the estimated probability distribution. λ is a weighting parameter which controls the influence of the dataterm.

Minimization is performed using convex relaxation, the details of which are beyond this literature review. The overall process allows for drastic improvements in segmentation, especially for difficult segmentations (e.g. images with textures), as show in figure 2.6.

2.1.6 Selective search

Selective search [35] is a bottom-up approach for segmentation that uses a regions based approach. It aims to combine the structure of an image and the power of exhaustive search into a more effective algorithm that 'generates a small set of high-quality object locations' [35]. By using image regions as a guide for segmentation, an exhaustive search is avoided, with the result that an image can be grouped into regions more quickly than previous methods.

Selective search works similarly to the graph-based approach of Felzenswalb and Huttenlocher [29]. Indeed, the authors use the work of [29] to create initial regions for their algorithm. However, selective search takes a more regions-based (rather than pixel-based) approach to groupings. Multiple aspects of similarity (colour, texture, size and fill) between regions are combined as a single measure. The algorithm then iteratively groups the two most similar regions together, following which new similarities are calculated. This continues until the whole image becomes a single region.

2.2 Segmentation via region proposals

In 2013, Girshick [36] introduced a region-proposal based CNN (R-CNN). Although the key results spoke to object recognition, R-CNN also achieved good results on semantic

segmentation. The broad idea is to use a bottom-up segmentation method for initial region proposals, followed by a CNN for classification.

R-CNN works in 3 stages. First, selective search is used to produce a set of candidate regions. Second, these regions are fed to a CNN (usually a pre-trained classification CNN, e.g. AlexNet, VGG, ResNet) for feature extraction. Finally, the extracted feature vector is scored using a SVM trained for that class.

By using selective search (as opposed to SIFT or HOG), R-CNN produces far fewer features per class, which results in significantly faster performance and less memory use than other existing methods at time of publication. Although the performance improvements for object detection were significant, only slight gains were achieved for semantic segmentation tasks. An alternative region proposal method uses superpixels, an approach used by Mostajabi *et al* [37].

R-CNN was improved first with the introduction of fast R-CNN [38], second with faster R-CNN [39] and most recently with mask R-CNN [40]. Mask R-CNN (figure 2.7) performs excellently on instance segmentation, resulting in state-of-the-art instance segmentation results in 2016.

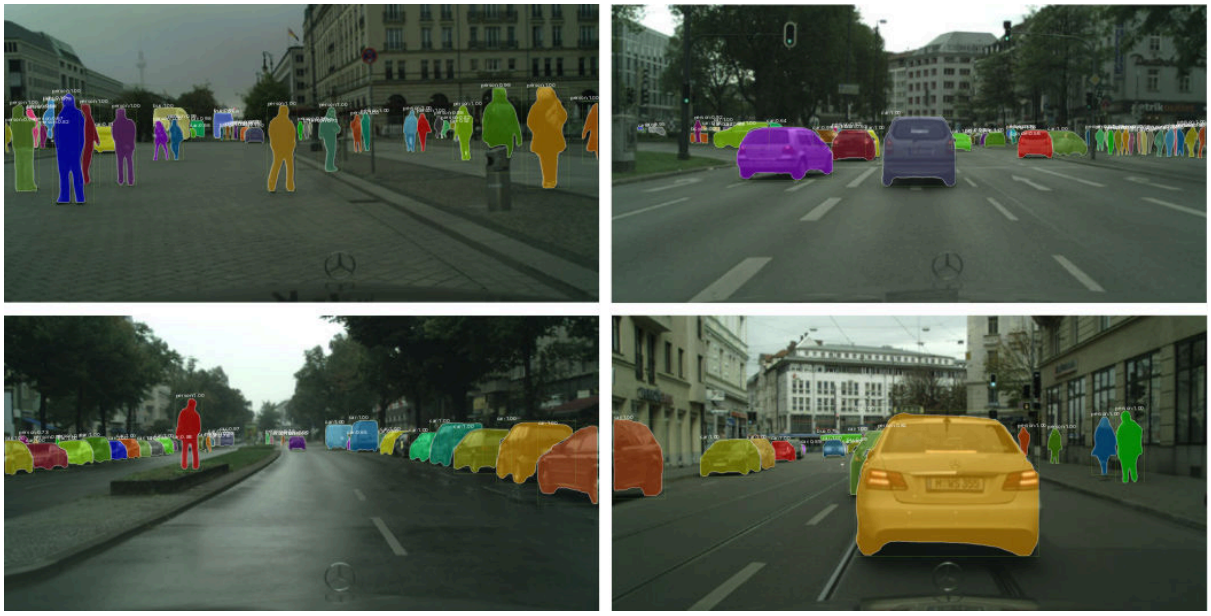


Figure 2.7: Mask-RCNN segmentations[40].

2.3 Fully convolutional networks (FCNs)

Long *et al.*[13] introduced the idea of using FCNs for semantic segmentation. FCNs differ from previous networks, in that there are no region proposals extracted from the input image. Rather, a direct end-to-end pixel-to-pixel mapping is learnt by the network. At

the time of introduction, FCN implementations achieved state-of-the-art segmentation.

As their name implies, FCNs use only convolutional, pooling and activation functions (i.e. no fully connected layers). This results in the network computing a nonlinear *filter*, as opposed to a traditional DNN which computes a nonlinear *function*. One advantage of using FCNs is their speed of training - both feedforward computation and backpropagation are much more efficient. Another advantage is they can take any arbitrary-sized input image. After a series of convolutional layers, which downsample the input, upsampling (or backwards convolution, sometimes called deconvolution) is required to recover the activation positions, giving a meaningful output related to the input.

However, the problem with this approach is a loss of resolution due to the downsampling by the convolutional and max-pooling layers (figure 2.8). The authors use 'skips' to address this problem, whereby activation from previous, higher resolution layers is interpolated with later, lower resolution layers. A different solution to this uses a deconvolution or expansion network, which has a series of unpooling and/or deconvolutional layers. Ronneberger *et al.* [14] use a symmetric expansion branch in their implementation of U-Net, which is currently state-of-the-art for segmenting medical images.

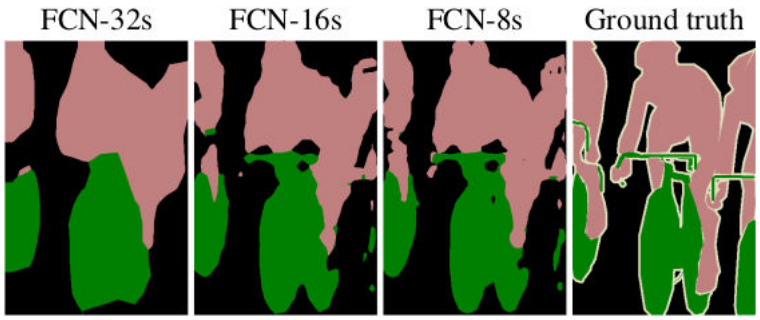


Figure 2.8: Segmentation using a FCN. Note the loss in resolution [13].

2.3.1 DeepLab

Deeplab is a state-of-the-art deep network for semantic image segmentation. All versions of Deeplab use a DCNN (such as VGG or ResNet) trained for image classification, which is then re-purposed for semantic segmentation (by transforming the networks into FCNs and using atrous convolutional layers).

DeepLabv1 [15] overcomes the problem of poorly localized features of FCNs by using atrous convolution (the 'hole' algorithm). The authors replace the final deep max-pooling layers with atrous convolutional layers, which effectively upsample, efficiently producing denser feature maps. In addition, to maintain spatial invariance, Conditional Random Fields (CRFs) are used on top of the DCNN. These CRFs are able to capture fine detail. Further, the authors use simple decimation to spatially sub-sample the first fully

connected layer in their network - this has the effect of reducing the receptive field size of the network, which reduces computation time by a factor of 2-3. The result of using this pipeline can be seen in figure 2.9.

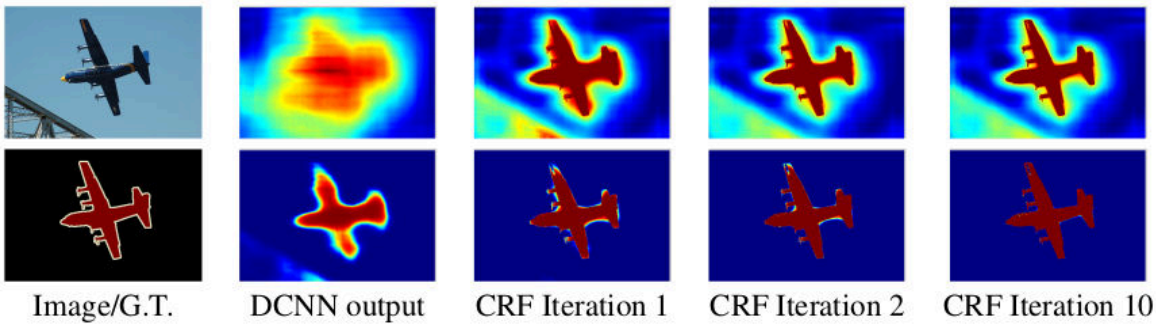


Figure 2.9: Segmentation using DeepLab [15].

DeepLabv2 [41] introduces atrous spatial pyramid pooling (ASPP). This deals with the problem of existence of objects at multiple scales. One way to deal with multiple scales is to provide rescaled versions of training data to the DCNN. However, this approach requires feature responses at all CNN layers for the multiple scales of the input image. Rather, the authors use multiple parallel atrous convolutional layers with different sampling rates - this effectively resamples a given feature layer at multiple rates prior to convolution. That is, multiple parallel atrous convolutional layers extract and process features in separate branches, which are then fused together to generate a final feature result.

Deeplabv3 [42] augments the ASPP layers of DeepLabv2 by incorporate image-level features, thereby encoding global context. This is done applying global average pooling on the last feature map of the model. In addition, batch normalization is included to facilitate faster training.

Deeplabv3+ [43] is the most recent iteration of Deeplab. It includes an encoder-decoder module to refine segmentation, esepecially along object boundaries. Deeplabv3+ is currently state-of-the-art for semantic segmentation. Results of segmentation can be seen in figure 2.10.

Figure 2.10: Segmentation results using Deeplabv3+ [43].

2.4 Weak supervision

The previously discussed DNNs rely on large amounts of annotated data to learn from. Acquiring this data is expensive and time-consuming. Therefore, there have been recent efforts to learn from weakly annotated data.

Dai *et al.* [44] use bounding boxes annotations (more economic than ground-truth masks) for training. That is, they treat bounding box as very coarse masks, which, through training, become more refined. 'BoxSup' produces competitive results compared to similar networks trained with access to true ground-truth masks.

Papandreou *et al.* [45] learn a DCNN from either weakly annotated training data (bounding boxes or image-level labels) or a combination of a few strongly labeled and many weakly labeled images. They introduce an expectation-maximization (EM) algorithm used to train Deeplabv1.

Khoreva *et al.* [46] recursively train a CNN from bounding boxes. Their approach reached 95% of the quality of the equivalent fully supervised model for semantic labelling and instance segmentation.

2.5 Interactive segmentation

As described in the introduction, segmentation is an inherently difficult task - it is not always clear what needs to be segmented.

To help solve this, a variety of interactive segmentation methods have been developed. That is, a user will give some sort of indication(s) as to important starting pixels that are in some way useful for segmentation. The way this indication is done varies - some methods require a bounding box to be drawn around the object, while others require more precise boundaries via drawing a polygon.

Adobe's Magic Wand tool [47] starts with user specified pixels as seeds. These seeds provide colour information, and a region of connected pixels is then computed such that the connected pixels fall within some colour tolerance related to the seed colour statistics. This works well if the segmentation has only a few, distinct colours.

Intelligent scissors (a.k.a Live Wire or Magnetic Lasso) [48] allows for a polygon to be traced around the object to be segmented. This polygon represents a minimum cost contour. As more points are created the minimum cost path from the current point to the previous point is computed. This minimum cost path follows colour weights, as can be seen in figure 2.11. Unfortunately, intelligent scissors performs poorly for highly textured images - there are many paths that can be computed.

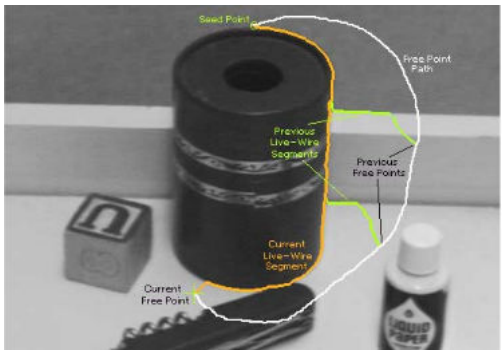


Figure 2.11: Intelligent scissors finding the minimum cost path [48].

Grabcut [49] is a two-step process that begins with a graph cut [50] to determine a 'hard segmentation', which can then be refined by iteratively selecting foreground and background pixels. Following this, border matting is computed by using alpha values (i.e. transparency) around the hard segmentation boundaries. This allows for a smoother segmentation outline.

Simple Interactive Object Extraction (SIOX) [51] requires a user to indicate a region of interest, known background, and optional known foreground pixels. Clustering of the colour signature of the known backgrounds and foregrounds is then performed - these clusters are stored in k-d trees. Given a particular pixel, the trees are traversed to determine if the pixel is part of the foreground or background clusters. If not, the Euclidean distance between the pixel and each cluster's centroid is computed to determine if the pixel is foreground or background. SIOX is fast and noise-robust (see figure 2.12). However, as with other methods, SIOX is dependent on colours for segmentation. In addition, the user must select a region of interest containing the entire foreground object. A further weakness is that multiple objects cannot be segmented at once. SIOX is currently the open-source standard for foreground extraction (used in GIMP, Inkscape, ImageJ, Blender and Krita).

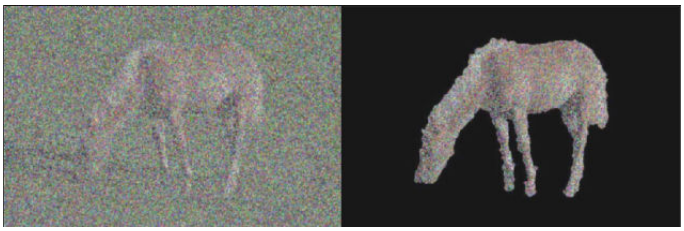


Figure 2.12: SIOX is robust to noise [51].

The combined colour and space approach of Nieuewenhuis and Cremers [34] is dependent on user input - the authors comment that unsupervised approaches to combining spatial and colour information in an unsupervised fashion degrades segmentation. In contrast, allowing interactive input provides excellent results.

Chapter 3

Methodology

Two methods were used for segmentation. The first method is purely heuristic. The second relies on a neural network for bounding box region proposals, followed by heuristic refinement.

3.1 Heuristic approach

A 6 stage pipeline (figure 3.1) was used to generate foreground and background segmentations: projection of 3D LiDAR points to the 2D frame of interest (with optional frame integration), coarse identification of background data based on height thresholding, clustering of background data, clustering of foreground data, merging of similar foreground and background clusters, and generation of convex hulls.

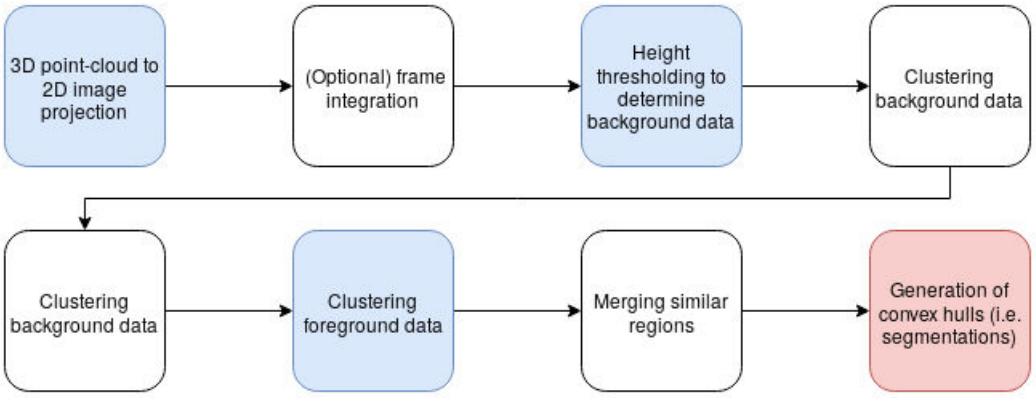


Figure 3.1: Heuristic pipeline.

3.1.1 3D point-cloud projection onto 2D image

The first stage of the pipeline projects 3D LiDAR data onto the 2D image. Here, Velodyne points from the KITTI dataset are projected onto a corresponding frame of interest. The number of points varies between environments and is dependent on the number of objects in frame.

A 3D point $\boldsymbol{x}$ in the LiDAR space gets projected to a 2D point $\boldsymbol{y}$ in the i'th camera image through the following transformation:

$$\boldsymbol{y} = \boldsymbol{P}^{i}_{rect}\boldsymbol{R}^{0}_{rect}\boldsymbol{T}^{cam}_{velo}\boldsymbol{x}$$

where $\boldsymbol{P}^{i}_{rect}$ is the i'th projection matrix after rectification (i.e. the camera intrinsic and extrinsic calibration, which performs the function of mapping 3D rectified camera co-ordinates to 2D image view), $\boldsymbol{R}^{0}_{rect}$ is a rectifying rotation matrix that represents the base (0'th) camera position (the transform from cam 0 co-ordinates to rectified cam 0 co-ordinates) and $\boldsymbol{T}^{cam}_{velo}$ is the concatenation of a 3x3 rotation ($\boldsymbol{R}^{cam}_{velo}$) and 1x3 translation matrix ($\boldsymbol{t}^{cam}_{velo}$) that transforms from the LiDAR co-ordinate space to the base camera co-ordinates (i.e. cam 0 co-ordinates):

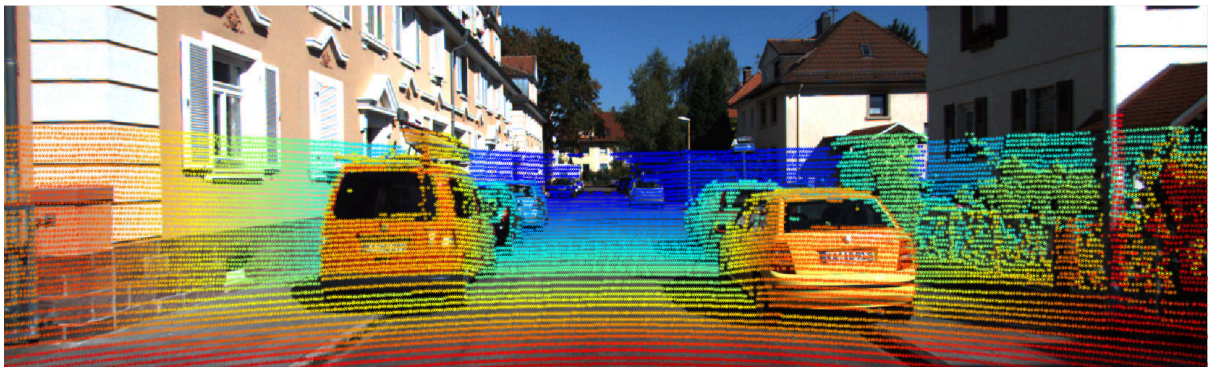
$$T^{cam}_{velo} = \begin{pmatrix} R^{cam}_{velo} & \boldsymbol{t}^{cam}_{velo} \\ 0 & 1 \end{pmatrix}$$

These matrices which contains rotational and translational information combined as a single transformations, are denoted as the special Euclidean group $\boldsymbol{SE}(3)$, and are also referred to as poses. They maintain Euclidean distance between points and represent transformations between different frames of reference.

Throughout this project, i was set as 2, which represents the left colour camera; the reader is referred to the original KITTI paper [1] for more details.

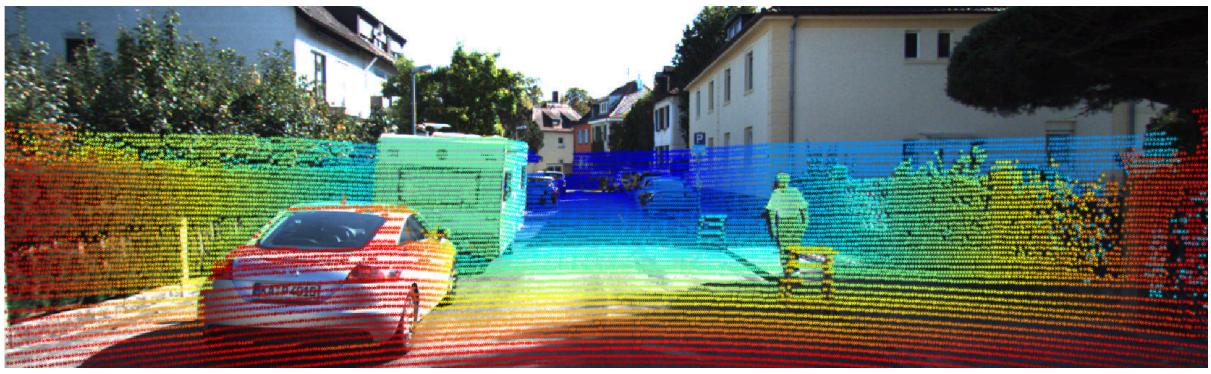
The results of projecting 3D LiDAR points onto a 2D camera image can be seen in figure 3.2.

(a) Before LiDAR projection.



(b) After LiDAR projection.

(c) Before LiDAR projection.



(d) After LiDAR projection.

Figure 3.2: Projecting 3D LiDAR points onto a 2D image. Here, the LiDAR points have been coloured according to their distance from the camera.

3.1.2 Frame integration

It is useful to be able to project LiDAR data from previous or future frames (i.e. perform frame integration). This was accomplished using available odometry data. Transforming odometry information to LiDAR points allowed for those points to be back- or forward-projected, thereby integrating multiple frames into a single frame of interest.

Within the KITTI odometry development kit, ground-truth odometry poses of the left camera are given, relative to the 0th frame. These poses are given as a 3x4 transform matrix (i.e. a 3x3 rotational matrix and 1x3 translational matrix concatenated) and must therefore be supplemented similarly to $\boldsymbol{T}_{velo}^{cam}$ - that is, append a $(0, 0, 0, 1)$ row in order to form a $\boldsymbol{SE}(3)$ transform:

$$\boldsymbol{T}_{source,dest} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix}$$

To transform LiDAR data into a frame of interest from previous or later frames the transform between those frames, relative to the 0th frame, must be calculated.

For example, consider the transform from frame 2 to frame 3 ($\boldsymbol{T}_{23}$) - that is, forward projecting the points from 2nd to the 3rd frame. From the ground-truth data, we know the transforms from the 0'th frame to the 2nd and 3rd frames (i.e. $\boldsymbol{T}_{02}$ and $\boldsymbol{T}_{03}$). We can obtain $\boldsymbol{T}_{23}$ by using the relationships between these transforms:

$$\boldsymbol{T}_{02}\boldsymbol{T}_{23} = \boldsymbol{T}_{03}$$

$$\therefore \boldsymbol{T}_{02}^{-1}\boldsymbol{T}_{02}\boldsymbol{T}_{23} = \boldsymbol{T}_{02}^{-1}\boldsymbol{T}_{03}$$

$$\therefore \boldsymbol{T}_{23} = \boldsymbol{T}_{02}^{-1}\boldsymbol{T}_{03}$$

due to the nature of $\boldsymbol{T}_{03}$ being the equivalent of $\boldsymbol{T}_{02}\boldsymbol{T}_{23}$ as illustrated in figure 3.3.

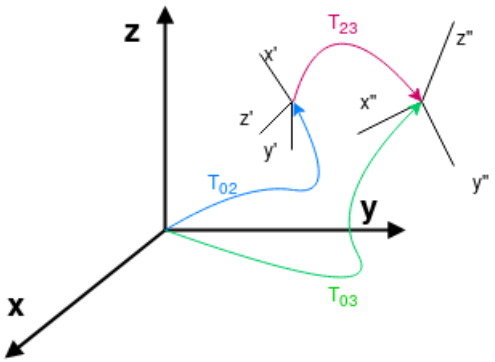
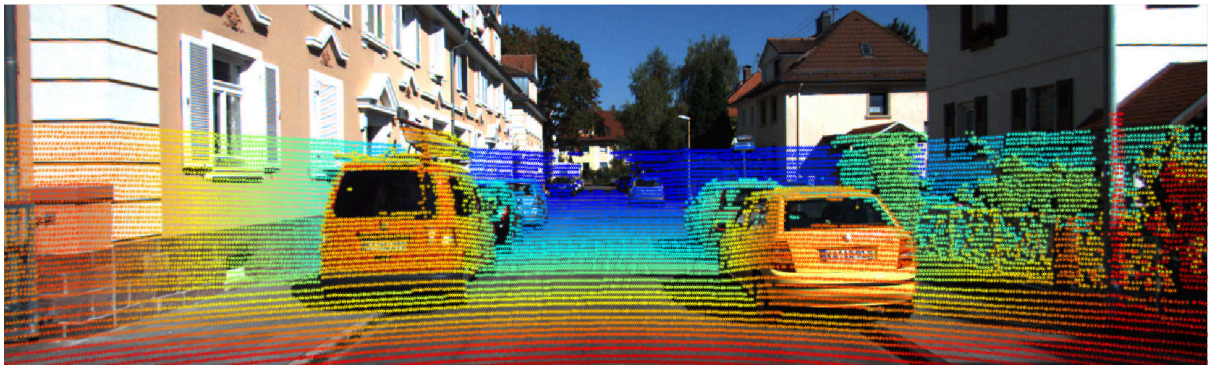


Figure 3.3: Relationships between pose transforms.

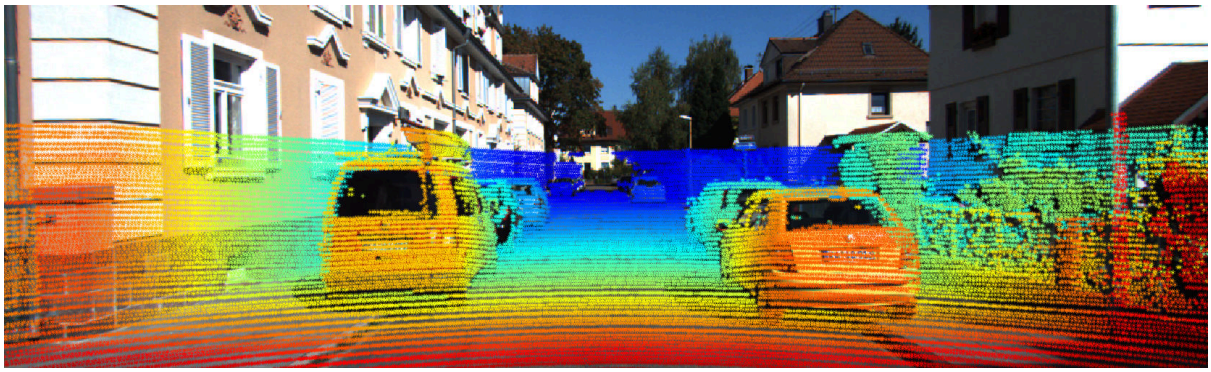
The resulting transform matrix is left-multiplied with $\boldsymbol{T}_{velo}^{cam}$, resulting in the full projection:

$$\boldsymbol{y} = \boldsymbol{P}_{rect}^{i}\boldsymbol{R}_{rect}^{0}\boldsymbol{T}_{s,d}\boldsymbol{T}_{velo}^{cam}\boldsymbol{x}$$

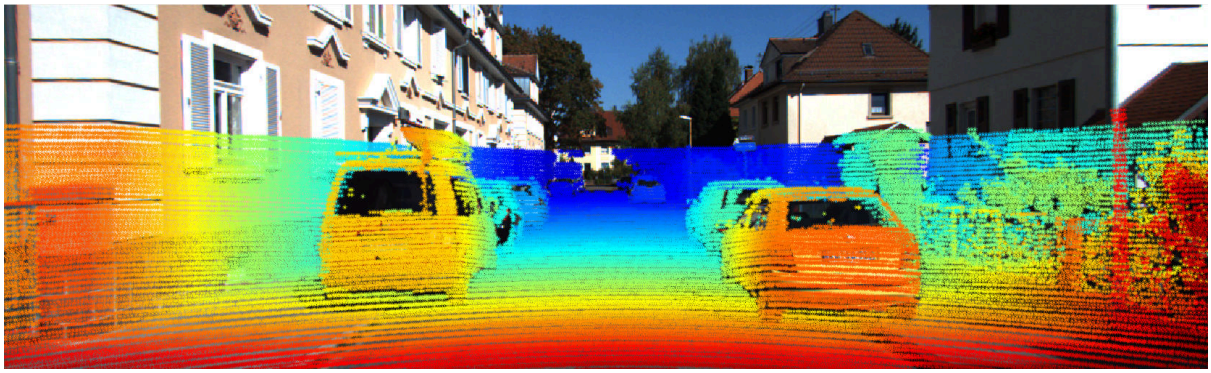
A similar derivation is used for back-projecting points. Results of frame integration can be seen in figure 3.4.



(a) Base LiDAR.



(b) 1 frame forward projected.

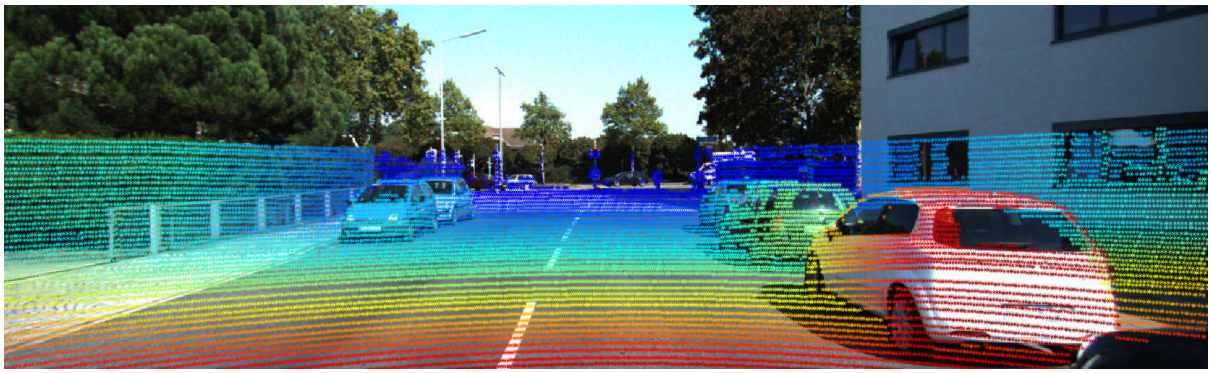


(c) 1 frame forward projected, 1 frame back-projected.

Figure 3.4: Frame integration - the LiDAR space gets more dense as more points are integrated from other frames into the frame of interest.

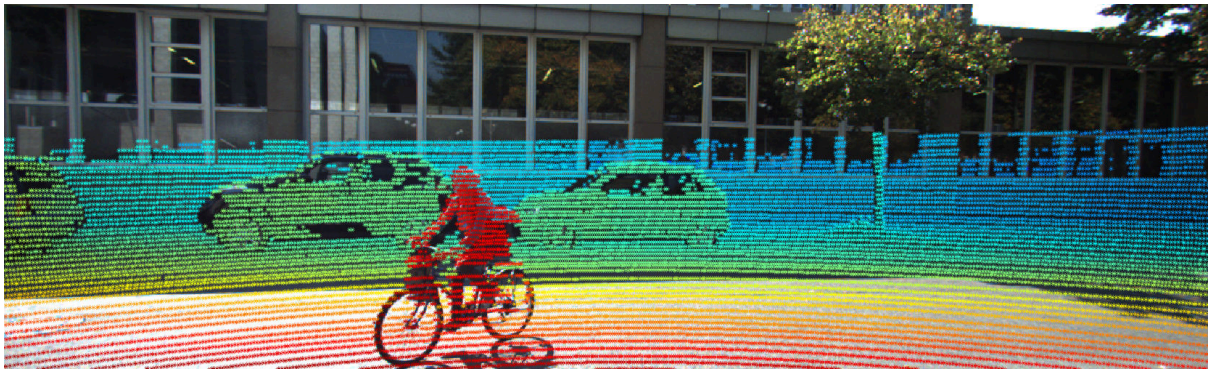
3.1.3 Background clustering

The next step was a coarse removal of assumed background elements. Here, it is assumed that all points beneath approximately 0.2m are part of the background (this eliminates most road and pavement) and are removed (figure 3.5). Further, it is assumed that points above a certain height threshold (2.5m) are also part of the background. These points, however, are not removed.

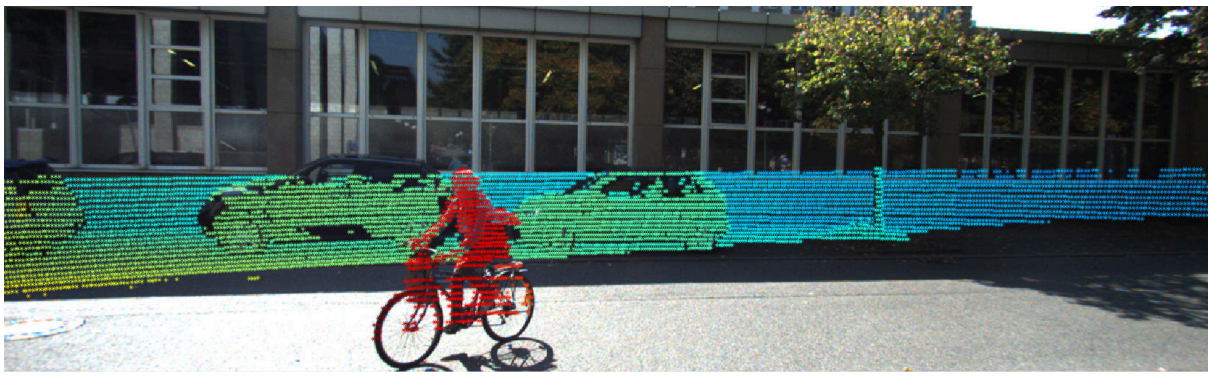


(a) Base LiDAR.

(b) After height thresholding.



(c) Base LiDAR.



(d) After height thresholding.

Figure 3.5: Applying height threshold to LiDAR points.

A further coarse removal of distant points is then performed. Here, points further than 28m from the camera are identified and treated separately. Because these distant points are naturally more sparse than closer data, density is used to segment distant clusters (i.e. sparse point areas are removed). Remaining points are then clustered based on x, y position, yielding several dense clusters of distant points.

Significantly high points (i.e. above the height threshold) are then clustered, based on x, y, approximate depth (estimated from LiDAR data) and colour. A greater weight is given to position and colour when calculating cluster linkages - this allows background clusters to extend across multiple objects not necessarily at the same depth (e.g, multiple bushes (of the same colour) that span across several depth intervals - see figure 3.6). These background clusters are optionally used to remove persisting foreground segmentations (referred to as foreground/background similarity merging).

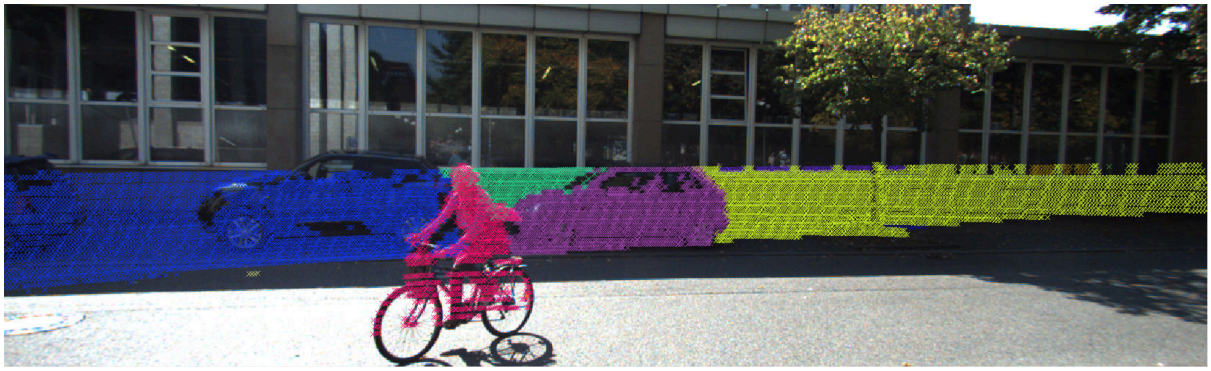
Figure 3.6: Background clusters.

3.1.4 Foreground clustering

A hierarchical clustering of all remaining LiDAR points is then performed. Here, a weighted Euclidean distance is used to link all pixels together in a hierarchy, based on x, y and depth (once again, estimated from LiDAR data). A greater weight was given to depth, thus causing cluster linking to prefer points that have a similar depth[1]. Due to the high number of LiDAR points (especially with frame integration), this is the most expensive part of the pipeline. The cluster dendrogram was cutoff at a maximum of 10 clusters; the results of the clustering can be seen in figure 3.7.

[1]Colour was not used for this clustering. This is due to foreground objects (e.g. cars) frequently having a range of significantly different colours (i.e. white car body, black windows, red tail-lights).

(a)



(b)

Figure 3.7: Foreground clusters - each cluster shown in a different colour.

Once foreground points have been clustered, colour and positional signatures are determined for all foreground and background segmentations. Similar clusters are merged together, shown in figure 3.8. Here, similarity means that the cluster has a similar colour signature and position to the known background cluster.

3.1.5 Colour and positional signatures

Colour signatures for each cluster were calculated using colour histograms. Separate histograms for red, green and blue channels were determined, each with 16 buckets. After normalization (i.e. dividing each histogram by the number of pixels contributing to the histogram) a sum of differences was computed between the background and foreground histograms. The resulting sum represents a measure of correlation between the background and foreground colours. A threshold of 0.3 was empirically determined to correctly indicate significant difference.

This colour signature was chosen for its simplicity and efficiency - other signature methods, such as building a KD-tree (similarly to SIOX), or using a joint distribution (similarly to Nieuwenhuis and Cremers) were deemed unnecessarily complex for this application. This histogram approach was inspired by HOGs.

(a) Here, foreground points (in blue) are determined to be similar to known background points (in red).

(b) Before similarity merges.

(c) After similarity merges. Note that persisting incorrect foreground elements have been removed.

Figure 3.8: Merging similar foreground and background regions.

Similarity in position was determined by comparing the square distance between average background and foreground x, y positions. Here, a positional threshold of 7000 was empirically found to correctly identify significant positional difference[2].

3.1.6 Convex hulls

At this stage, core foreground clusters exists. Convex hulls are drawn around these clusters (using Delaunay triangulation). Hulls that are directly above one another are compared (again using colour and position signatures) and merged if similar. Here, a smaller positional signature threshold of 3 000 is used - this allows for similarly coloured and very closely positioned elements directly above one another to be merged into a single object of interest (for example, a tall pole might have background and foreground elements due to height thresholding - this step allows the individual clusters from both background and foreground to be merged).

Finally, hulls are then examined to determine any overlapping regions. Shallower (i.e. closer to the camera) clusters are given preference, and any intersections between hulls are removed based on the depth levels *within* overlapping regions. This can be seen in figure 3.9.

[2]This large value is due to the distance not being square-rooted for computational efficiency.

(a) All convex hulls. Note the overlap between hulls in the right side of frame.

(b) Convex hulls with intersecting regions removed, giving preference to hulls closer to the camera.

Figure 3.9: Convex hulls drawn around persisting foreground clusters.

3.2 Region proposal approach

A 4 stage pipeline was used for the neural network region proposal approach (figure 3.10).

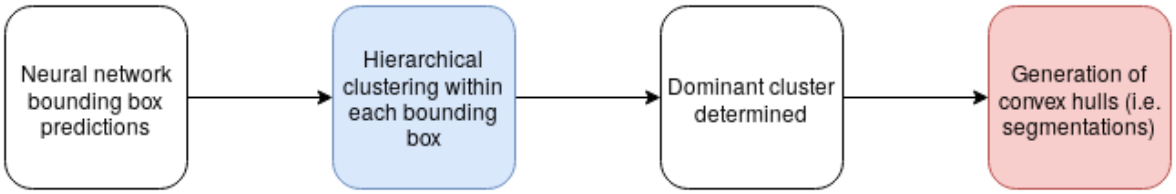


Figure 3.10: Neural network region proposal pipeline.

A pre-trained TensorFlow network with a ResNet backbone [52] was used to determine instances of cars and pedestrians. The output from the network is a bounding box (see figure 3.11). Thus, in a sense, this network acts as a region proposal network.

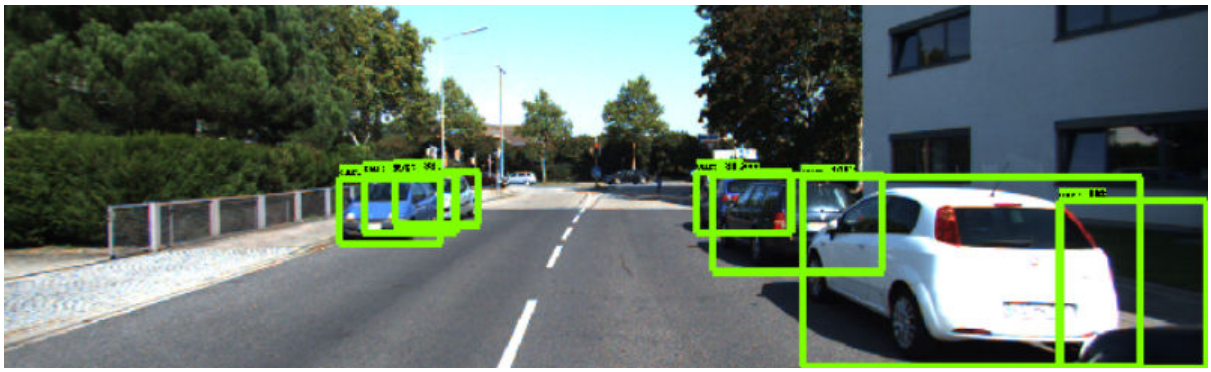


Figure 3.11: TensorFlow network bounding boxes.

The output boxes are then refined using the same strategy of the heuristic method. However, rather than clustering the entire LiDAR space, each proposed region is clustered independently. Further, only hierarchical clustering occurs - there is no merging of foreground/background, or attempting to threshold for background pixels. This is due to the high accuracy of proposed regions. Once again, a hierarchical clustering based on position and depth was used.

A cluster threshold of 5 maximum clusters was used, following which the biggest cluster is segmented using convex hulls. Naively, it may be expected to use a cluster threshold of 2 clusters (one for foreground elements and one for background). However, for bigger vehicles that have multiple colours and span multiple depth intervals, 2 clusters frequently failed to correctly group pixels of interest. Using more clusters allows more groups to be found. However, over-clustering leads to overly specific regions of a vehicle being grouped - 5 clusters was experimentally determined to be a good balance between over- and under-clustering.

3.3 Interactivity

An interactive GUI was built to allow a user to quickly remove/add segmentations (i.e. convex hulls). Although this increases the overall time of the pipeline, and no longer makes the method fully automatic, it is thought the gains of deleting/adding segmentations outweigh these drawbacks.

A user can left click on existing polygons to delete them. Alternatively, a right click will delete all polygons except the one clicked on. The user can press ‘a’ to enter an append mode wherein they can add new polygons.

3.4 Testing

Each frame was segmented 5 times to allow for cache warming. Segmentations were run with and without frame integration, as well as with and without similar foregrounds and backgrounds being merged, in order to determine what was most effective. Testing with sampling of LiDAR points was also investigated.

Timing results of using the interactive GUI were obtained separately.

3.5 Evaluation

Some ground-truth segmentations are provided in the KITTI dataset. However, these are (purposefully) not linked with their corresponding sequences - thus, obtaining relevant LiDAR points is difficult. Several independent authors have created ground-truth annotations [53], linked to particular sequences, which are used for evaluation.

The primary evaluation metric measured was intersection over union (IoU), defined as:

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

where TP, FP and FN are the number of true positive, false positive and false negative pixels. IoU was measured only for car and pedestrian classes.

In addition, the pre-trained TensorFlow network was used to count the instances of cars and pedestrians. This is useful for evaluating sequences that do not have ground-truth segmentations, and for further understanding the nature of segmentations (ground-truth segmentations do not separate instances of the same class, unlike the TensorFlow detector).

Chapter 4

Results

4.1 Timing and evaluation

Timing and mean IoU for two residential drives (for which there is corresponding ground-truth data[1]) are shown in tables 4.1 and 4.2.

Table 4.1: Results for residential sequence 2011_09_30_drive_0027.

Type	**Average number LiDAR points**	**Average time (s)**	**Average polygons**	**Mean IoU (%)**
No background/foreground similarity merge (no frame integration).	7 552	1.85	5.65	44.56
Similar background/foreground merged (no frame integration).	7 552	1.92	4.33	36.26
Integration with 1 future, 1 past frame.	23 128	15.51	7.30	46.26
Integration with 2 past frames.	19 606	11.76	5.80	46.17
Interactive GUI removals (no merge, no integration).	7 552	3.41	2.8	62.19
Interactive GUI removals and additions (no merge, no integration).	7 552	22.35	5.46	72.97
TensorFlow detections.	NA	10.06	4.50	NA
TensorFlow refinements.	7 552	0.12	4.50	53.13

[1]Timing results from other sequences (without ground-truth segmentations) can be found in the appendix. These are omitted from results as mean IoU cannot be calculated without ground-truth.

Table 4.2: Results for residential sequence 2011_10_03_drive_0027.

Type	**Average number LiDAR points**	**Average time (s)**	**Average polygons**	**Mean IoU (%)**
No background/foreground similarity merge (no frame integration).	6 700	1.65	5.75	35.68
Similar background/foreground merged (no frame integration).	6 700	1.63	3.85	28.64
Integration with 1 future, 1 past frame.	20 341	12.18	7.6	36.21
Integration with 2 past frames.	17 573	8.92	5.98	35.31
Interactive GUI removals (no merge, no integration).	6 700	4.09	3.09	53.35
Interactive GUI removals and additions (no merge, no integration).	6 700	21.81	5	68.44
TensorFlow detections.	NA	9.83	5	NA
TensorFlow refinements.	6 700	0.10	5	58.96

Table 4.3: Comparison of TensorFlow mean IoU across different classes.

Class	**TensorFlow mean IoU (%) over all sequences**
Cars	55.90
Pedestrians	1.57
Overall	56.05

4.2 Sampling

Here, a sampling of x indicates that only every $x'ith$ point was used. That is, a sampling of 2 indicates that only every second LiDAR point was used, a sampling of 3 every third LiDAR point, etc.

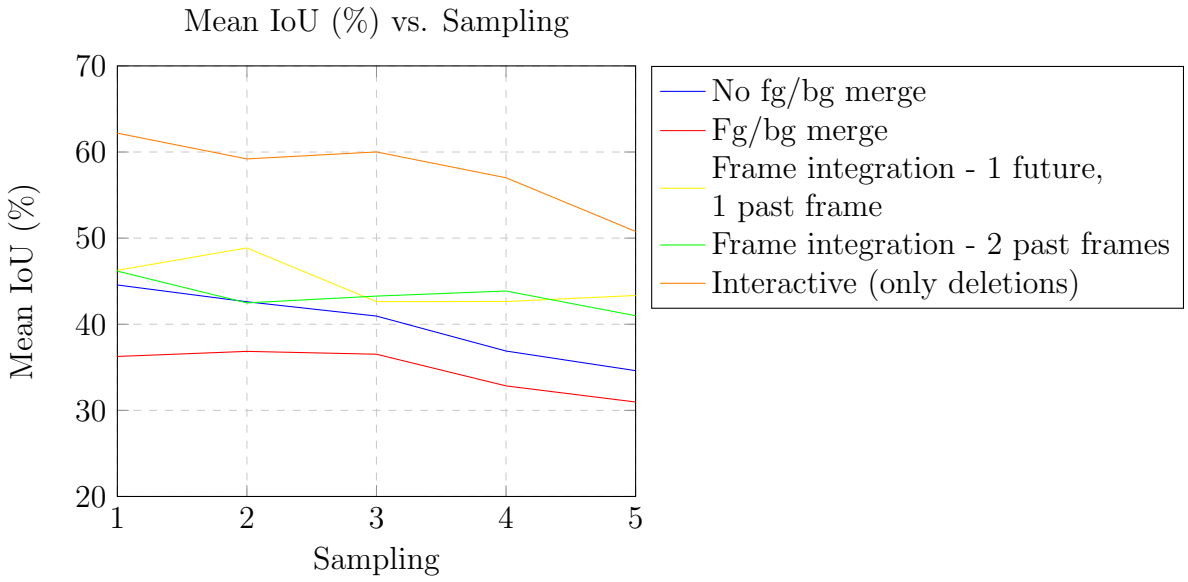


Figure 4.1: Mean IoU vs. sampling of LiDAR data

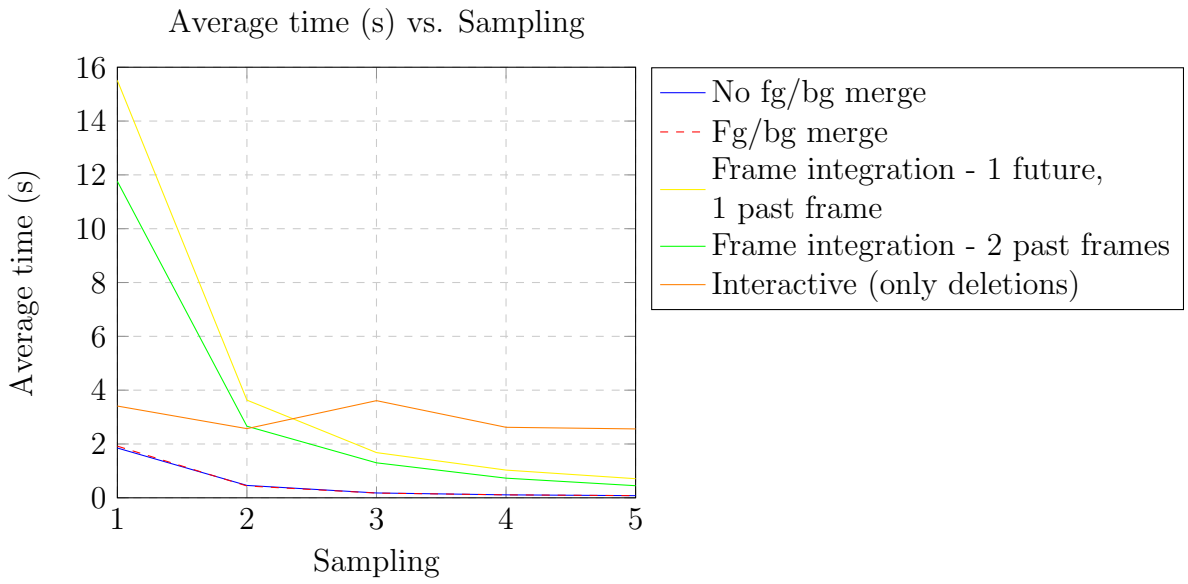


Figure 4.2: Average time vs. sampling of LiDAR data

4.3 Interactivity

The interactive GUI can be seen in figure 4.3 below. As MATLAB was used as the programming language of choice, this GUI Was built using MATLAB's GUIDE tool. The objective was not to create a complex tool that would allow fine-grained instance segmentation. Rather, it was desired to build a simple tool that would quickly allow for deletions and/or addition of segmentations. Average time to delete over-segmentations was 3.75 seconds. Average time to delete and add new segmentations was 22.08 seconds.

(a) Before removing segmentations.

(b) After removing segmentations.

(c) After removing and adding segmentations. Note the inclusion of the cars closest to the camera which were previously not segmented.

Figure 4.3: Examples of interactive segmentations using the GUI app.

4.4 Heuristic segmentations

Different outcomes of the heuristic segmentation can be seen in figure 4.4. Resulting segmentations can be seen in figure 4.5.

(a) Base image.

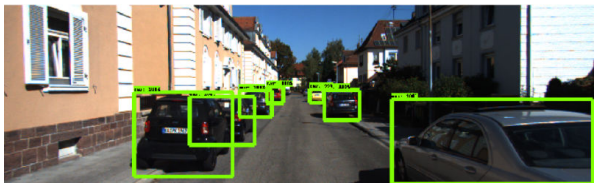
(b) No fg/bg merge.

(c) Fg/bg merge.

(d) Frame integration - 1 future, 1 past frame.

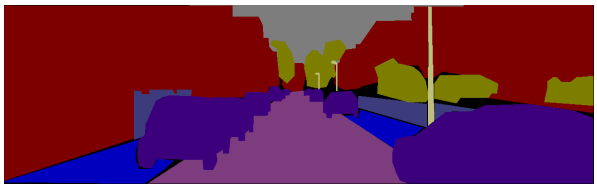
(e) Frame integration - 2 past frames.

(f) Interactive.



(g) TensorFlow detections.

(h) TensorFlow refinements.



(i) Ground truth.

(j) Ground truth, filtered for cars and pedestrians.

Figure 4.4: Segmentation approaches.

Figure 4.5: Resulting segmentations. Left column is original image. Right column is original image with segmentations over-layed.

4.5 Neural network refinements

Example results of using the heuristic clustering to refine bounding box proposals can be seen in figure 4.6 below.

Figure 4.6: Resulting refinement of neural network bounding box proposals. Left column is neural network proposals. Right column is heuristic refinements.

Chapter 5

Discussion

5.1 Timing

The time taken to process each frame expectedly increases with the number of LiDAR points. Thus, forward- or back-projecting points becomes increasing computationally expensive. On average, 1 frame of data contains 7126 points (this will vary between environments and depending on what is in frame), taking approximately 1.75 seconds (without fg/bg merging) to process.

Processing of neural network bounding boxes was much quicker (average of 0.11 seconds). This is due to clustering only being performed *within* each bounding box - this is far less computationally expensive than performing clustering across the entire LiDAR space. However, generating bounding boxes was significantly slow, requiring on average 9.95 seconds. Further, the time taken to train the network is unknown, and is likely to further add to processing time.

5.2 Evaluation

Mean IoU was lower for non-interactive techniques (average of 40.12%, without fg/bg merging). This is mainly due to extraneous segmentations causing an increased union, and therefore decreased IoU. Deleting these extraneous segmentations using the interactive GUI therefore significantly improved precision to 57.75%. Adding additional segmentations (i.e. hand-traced) further improved mean IoU to 70.71%.

Based on the average number of polygons segmented, removed and/or added, it seems that non-interactive approaches segment the correct number of objects (similar results for non-interactive number of polygons and interactive with deletions and additions), but frequently over-extend segmentations (an average of 2.76 incorrect segmentations

removed interactively).

The average number of polygons using the TensorFlow model is lower than that of the fully interactive approaches (an average of 4.75 vs 5.23 polygons respectively). This is due to the TensorFlow network not detecting all objects of importance (for example, the right-most car in figure 4.4 (f) is undetected).

Refinement of generated neural network bounding boxes shows reasonably good precision (an average of 56.05%). This is higher than pure hierarchical approaches, but comparable to hierarchical approaches in combination with interactivity (limited to deletions) and worse than hierarchical approaches with deletions and additions.

5.3 Sampling

Precision of segmentations decreased for all approaches with sampling. Intuitively, as the amount of sampling increases the LiDAR data is more sparse, and therefore clustering becomes less robust. In addition, sampling is likely to cause inaccurate cluster boundaries - true boundaries may be removed in the sampling process.

The least amount of precision lost was for frame integration - integration with 1 frame forward and 1 frame forward had a maximum loss of 3.63%, while integration with 2 past frames had a maximum loss of 5.18%. These low losses are attributed to frame integration 'supplementing' the sampled data, thus keeping a dense LiDAR map.

Mean IoU of interactive segmentation (i.e. removing extraneous segmentations) decreases significantly (a maximum loss of 11.41%) with sampling. This is due to the sampled segmentations, while often overlapping ground-truth, frequently being inaccurate. These inaccurate segmentations were therefore removed, leading to a decrease in IoU.

Time to process each frame decreases with increased sampling. This decrease was non-linear, as can be seen in figure 4.2. Thus, increasing sampling has significant time performance improvements.

5.4 Clustering

The most expensive part of the algorithm is foreground clustering. As previously mentioned, a hierarchical clustering is used, with a cutoff at a particular number of clusters - in this case, 10.

Ideally, the clustering hierarchy should cutoff at some inconsistency coefficient, where the coefficient is a measure of relative height variation within the clustering dendrogram. Ex-

periments were performed with inconsistency cutoffs, but performed poorly as compared to a fixed cluster cutoff.

A better solution may use 'soft' clustering - that is, each pixel is clustered with a particular probability. Pixels with low probabilities could then be removed from clusters, or assigned to different (more probable) clusters.

5.5 Occlusions

One area the algorithm performs particularly poorly is with occluded cars (see figure 5.1). Here, multiple cars are placed into the same cluster, resulting in a single segmentation with multiple objects.

Figure 5.1: An example of incorrect clustering. Here, the 2 cars in left side of the frame are put into the same cluster.

Various strategies were attempted in an effort to solve this issue. These included further intra-cluster clusterings based on depth, position, colour (in multiple colour spaces), attempting to draw active contours (using 'snakes'), thresholding (in multiple colour spaces) and finding hard edges (using various edge-detecting kernels). Ultimately, no one solution could be found to consistently correctly determine intra-cluster clusters (i.e. detect occlusions).

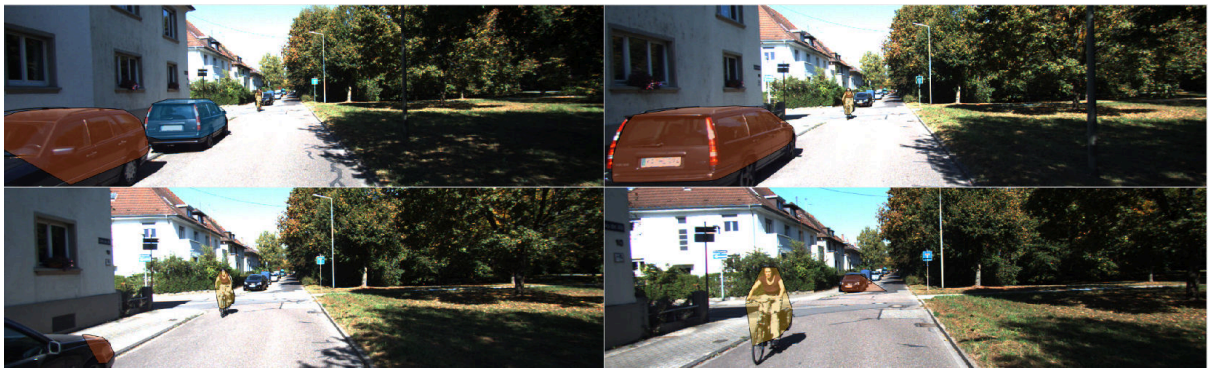
Investigating soft hierarchical clustering, with intra-clustering based on colour features may offer a solution.

5.6 Dynamic vs. static objects

Clustering of static objects improves with frame integration. This is expected - as more data is forward- or back-projected the frame of interest becomes 'richer', allowing for better segmentations to be drawn.

However, dynamic object detection usually performs better without frame integration. This is because the transform of the points from previous/later frames to the current frame is incorrect, due to the movement of the dynamic object. In contrast, simply using one frame leads to better results, due to only the points *at that time* being used for segmentation. Figure 5.2 shows a car and a cyclist dynamically being tracked across frames without integration. The same car, with frame integration, can be seen in figure 5.3.

(a) Segmenting a moving car.



(b) Segmenting a moving cyclists.

Figure 5.2: Segmenting dynamic objects.

(a) Without frame integration.

(b) With frame integration

Figure 5.3: Frame integration leading to inaccuracy in segmenting dynamic objects.

5.7 Environment

The algorithm produces better results in dense urban environments, as compared to more open environments (such as highways). This is due to LiDAR points being more spread out in open environments; the clustering is therefore more difficult.

However, when there is a dense cluster of points, this is almost always detected. Thus, when a car comes into frame (on a highway), this car is successfully segmented, despite the dynamics at play ((a) of figure 5.2). In contrast, there is often more intra-cluster failure in city and residential environments, due to the natural densities of many objects in a single frame.

Another environmental consideration is hills. Due to the naive height threshold applied to eliminate the road, the algorithm performs poorly on inclined environments - the part of the road on the incline is not removed, as show in figure 5.4.

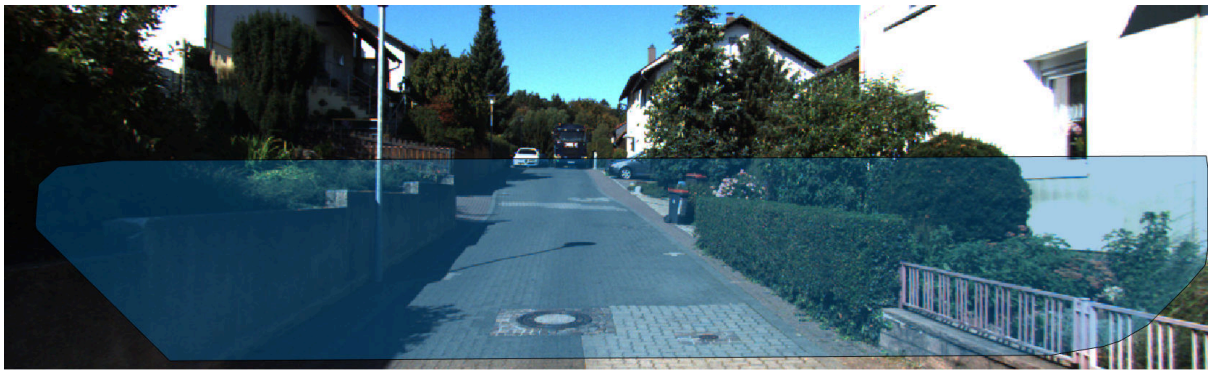


Figure 5.4: Height threshold failure on an incline. Here, the road is incorrectly added to persistent foreground clusters.

5.8 Convex hulls

Computing convex hulls proves useful in generating masks for segmentation. However, these are obviously limited to convex shapes. Thus, applying convex hulls to clusters containing humans, bicycles or other non-convex objects leads to segmentations that fail to capture the true shape/borders. This can be seen in (b) of figure 5.2 - the convexity fails to draw accurate borders around the cyclist.

5.9 LiDAR

This report is not the forum to discuss the advantages/disadvantages of incorporating a LiDAR system for self-driving cars. However, one aspect of LiDAR worth noting is that it is not in any way colour dependent. This allows clustering (and ultimately segmentation) to find objects that are traditionally very difficult to segment (and would be near impossible using a colour or edge-based method).

For example, consider frame 85 from 2011_09_30_drive_0027 ((a) of figure 5.5). Here, the car in the right hand side of the frame, near the border, is almost impossible to see with the naked eye. Traditional and neural network approaches to segmenting this car would therefore fail, as these are usually dependent on finding colours/edges and/or using region proposals. However, using LiDAR allows for 'seeing' of this car, despite it blending so heavily into the background ((b) of figure 5.5). Therefore, a segmentation is easily found (even with heavily sampled LiDAR data - this picture taken from the 5x sub-sampled run).

(a) Before segmentation.

(b) After segmentation.

Figure 5.5: Frame 85 of 2011_09_30_drive_0027. On top, the car hiding in the right near the border is almost impossible to see. On bottom, this car has been segmented via LiDAR clustering.

5.10 Interactivity

The interactive GUI significantly increases mean IoU. This comes at the cost of increased overall pipeline time. Simple deletions of extraneous segmentations took an average time of 3.75 seconds with an increase in mean IoU of 17.63%. Deletions in combination with additions (i.e. drawing new segmentations) took longer (an average of 22.08 seconds), while further increasing mean IoU to 72.97% (a further increase of 12.96%). The interactivity may be an affordable sacrifice to make, depending on application. Regardless, these increased times are negligent as compared to current annotation techniques.

5.11 TensorFlow bounding boxes

The neural network approach of generating bounding boxes followed by hierarchical clustering for refinement performed better than pure heuristic segmentation (mean IoU of 56.05% vs 40.12%), and comparably to heuristic segmentation with interactive deletions

(mean IoU of 56.05% vs 57.75%).

However, the danger of this approach is that initial bounding boxes are only as good as the data the network has been trained on. That is, heuristic clustering is more likely to find all objects of interest, even if subsequent clustering leads to inaccurate borders/overlap between objects. In contrast, the neural network is likely to miss objects of importance when it has not been trained to recognize them. For example, despite being trained to recognize pedestrians, the network frequently failed to find cyclists. This is evident when assessing mean IoU per class, as measured in 4.3.

Other region proposal methods (e.g. Selective Search) could be investigated as a replacement for the neural network.

Chapter 6

Conclusions

Although mean IoU is low (for non-interactive approaches), the aim of this project was not to generate perfect segmentations. Rather, the objective was to generate baseline segmentations from which better annotations may be drawn. In this regard, the project is successful - baseline segmentations provide good borders around objects of interest.

Time required to perform segmentations improves with sampling of LiDAR data. Without sampling, and without any interactivity, approximately 2060 images can be processed per hour. Including interactivity by deleting extraneous segmentations reduces this to 655 images. Combined deletions and additions of segmentations further reduces this to 141 images per hour.

This increased time comes at the benefit of increasing mean IoU (i.e. precision). The interactive GUI tool significantly improved results, achieving a maximum of 72.97% mean IoU.

Sampling of LiDAR data significantly reduces processing time per frame. At the same time, all approaches suffer from some loss in precision. The degree of loss varies between approaches - frame integration suffered the least loss, while interactive approaches suffered the most.

Using every third point (i.e. a sampling factor of 3) reduced mean IoU by $\approx 3.6\%$ for the baseline approach (no frame integration, no foreground/background similarity merging). However, the time required to process each image drops significantly, by $\approx$ 1.67 seconds, allowing a theoretical processing of 20 000 images per hour with a mean IoU of $\approx 41\%$. Further sampling follows similar trends of decreasing processing time at the cost of decreasing IoU.

Using a pre-trained neural network to find bounding boxes around objects of interest showed promising results, with an average mean IoU of 56.06%. However, the danger of this approach is that the network is only as good as the data it is trained on. Fusing the

two methods could yield better results than either method alone.

Several recommendations are made in the next chapter to extend this project. However, based on the promising results of the GUI tool, preliminary work could focus on extending this aspect of the project, perhaps in combination with sampling.

Chapter 7

Recommendations

Segmentation is a difficult computer vision task. Given the limitations of this project, there are several areas where further work could be done:

First, the method of clustering could be improved. Currently, a hard clustering is performed, which is only dependent on position and pseudo-depth. A soft clustering could be investigated, which is also dependent on colour. This could generate more accurate segmentations.

Second, the clustering could be merged with other algorithms and/or methods. For example, a Selective Search could be performed, after which region proposals from both algorithms are compared. Alternatively, a depth map could be generated from available stereo cameras, which could be used to supplement LiDAR data. Other strategies to investigate include region growing, contour detection and graph-cut algorithms.

Third, better integration across multiple frames could be introduced. Thus, if an object is detected in one frame, there should be a level of consistency with segmentations in the next frame. Even with frame integration, clustering is performed separately. In other words, there is no tracking of segmentations across multiple frames.

Fourth, the road segmentation could be improved. Rather than using a naive height threshold (as explained in methodology) a dedicated FCN could be used to segment the road (e.g., [54] or [55]), allowing for more accurate borders along the bottom of objects to be determined.

Fifth, the interactive GUI could be improved, allowing users to expand or finely adjust segmentations. In addition, class allocators could be incorporated to allow a user to mark a segmentation as belonging to a particular class. Lastly, investigations into using the GUI to indicate bounding boxes could be performed (i.e. replace the role of the current neural network with a GUI tool).

Sixth, in order to improve dynamic object detection, optical flow algorithms could be

introduced to track pixels across multiple frames. This would allow the benefits of frame integration for static object detection to be combined with better dynamic object tracking.

Seventh, the outputs of this pipeline could be fed into a SVM or other classifier to classify segmentations. This could remove the need for an interactive GUI - if a proposed segmentation has a low probability of recognition (i.e. identification as a pre-trained object), that segmentation should be removed.

Eighth, odometry data could be generated (rather than depending on provided odometry data). This would allow for frame integration along any sequence.

Ninth, this algorithm could be optimized, either by porting the code to C/C++ or by implementing a GPU solution. The most expensive part of the pipeline - hierarchical clustering - can be parallelized [56]; work could therefore be done in parallelizing the algorithm.

Tenth, due to the shortage of good ground-truth segmentation data for KITTI, an existing neural network (e.g. [2]) could be used to generate pseudo ground-truth segmentations, which would allow IoU to be calculated for any sequence.

Lastly, this method should be tested on other datasets. Several open-source datasets contain LiDAR data, including the recently released Waymo dataset [57]. The developed algorithm should be tested for consistency across these datasets.

Appendix A

7.1 Timing

There is little to no ground-truth data available for these sequences which the algorithm was run on. Therefore, mean IoU cannot be determined, or what can be calculated is likely a rough estimate. Nonetheless, this data is included for its timing results.

Ground-truth odometry is lacking for some sequences, thus there are no frame integration results.

Table 7.1: Timing results for city sequence 2011_09_26_drive_0009.

Type	**Average number LiDAR points**	**Average time (s)**	**Average polygons**
No background/foreground similarity merge (no frame integration).	3750	0.64	5.66
Similar background/foreground merged (no frame integration).	3750	0.63	3.85
Interactive GUI removals (no merge, no integration).	3750	3.91	2.98
TensorFlow detections.	NA	9.93	5.81

Table 7.2: Timing results for campus sequence 2011_09_28_drive_0038.

Type	**Average number LiDAR points**	**Average time (s)**	**Average polygons**
No background/foreground similarity merge (no frame integration).	5354	0.99	4.55
Similar background/foreground merged (no frame integration).	5354	0.99	3.5
Interactive GUI removals (no merge, no integration).	5354	3.86	2.36
TensorFlow detections.	NA	9.76	6.14

Table 7.3: Timing results for residential sequence 2011_10_03_drive_0042.

Type	**Average number LiDAR points**	**Average time (s)**	**Average polygons**
No background/foreground similarity merge (no frame integration).	2965	0.31	3.87
Similar background/foreground merged (no frame integration).	2965	0.30	2.96
Integration with 1 future, 1 past frame.	8721	2.76	6.29
Integration with 2 past frames.	6913	1.64	4.01
Interactive GUI removals (no merge, no integration).	2965	3.01	3.28
TensorFlow detections.	NA	9.95	0.76

7.2 Code

Code for this project can be found at:

`https://github.com/joshestein/EEE4022S_final_project`

Bibliography

[1] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[2] 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[3] P. K. Sahoo, S. Soltani, and A. K. Wong, "A survey of thresholding techniques," *Computer vision, graphics, and image processing*, vol. 41, no. 2, pp. 233–260, 1988.

[4] Amazon, "Amazon Mechanical Turk," 2019. [Online; accessed 15-Sept-2019].

[5] Scale AI, "The Data Platform for AI," 2019. [Online; accessed 15-Sept-2019].

[6] P. Amayo, "Interview." Personal communication.

[7] A. Neal, "LiDAR vs RADAR," 2018. [Online; accessed 5-Oct-2019].

[8] L. Elliott, "LiDAR put into a Tesla – and the world doesn't come to an end," 2019. [Online; accessed 12-Oct-2019].

[9] T. Lee, "Elon Musk: 'Anyone relying on LiDAR is doomed.'," 2019. [Online; accessed 15-Sept-2019].

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

[14] 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*, pp. 234–241, Springer, 2015.

[15] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014.

[16] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[17] J. M. Prewitt, "Object enhancement and extraction," *Picture processing and Psychopictorics*, vol. 10, no. 1, pp. 15–19, 1970.

[18] I. Sobel and G. Feldman, "An isotropic 3x3 image gradient operator for image processing," *Mach. Vis. Three–Dimens. Scenes*, no. June, pp. 376–379, 1968.

[19] A. Sydow, "Tou, j. t./gonzalez, r. c., pattern recognition principles, london-amsterdam-dom mills, ontario-sydney-tokyo. addison-wesley publishing company. 1974. 378 s., $ 19,50 .," *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 57, no. 6, pp. 353–354, 1977.

[20] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[21] K.-S. Chuang, H.-L. Tzeng, S. Chen, J. Wu, and T.-J. Chen, "Fuzzy c-means clustering with spatial information for image segmentation," *computerized medical imaging and graphics*, vol. 30, no. 1, pp. 9–15, 2006.

[22] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on information theory*, vol. 21, no. 1, pp. 32–40, 1975.

[23] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 5, pp. 603–619, 2002.

[24] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," *CoRR*, vol. abs/1807.05520, 2018.

[25] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," *CoRR*, vol. abs/1511.06335, 2015.

[26] X. Ji, J. F. Henriques, and A. Vedaldi, "Invariant information distillation for unsupervised image segmentation and clustering," *CoRR*, vol. abs/1807.06653, 2018.

[27] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 11, pp. 1101–1113, 1993.

[28] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Departmental Papers (CIS)*, p. 107, 2000.

[29] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[30] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[31] D. Cremers, "Variational Methods for Computer Vision - Lecture 9." [Online; accessed 10-Oct-2019].

[32] Dake, "Snake-contour-example," 2005.

[33] D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and associated variational problems," *Communications on pure and applied mathematics*, vol. 42, no. 5, pp. 577–685, 1989.

[34] C. Nieuwenhuis and D. Cremers, "Spatially varying color distributions for interactive multilabel segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 5, pp. 1234–1247, 2012.

[35] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[36] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[37] M. Mostajabi, P. Yadollahpour, and G. Shakhnarovich, "Feedforward semantic segmentation with zoom-out features," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3376–3385, 2015.

[38] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[39] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[40] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[41] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

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

[43] 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)*, pp. 801–818, 2018.

[44] J. Dai, K. He, and J. Sun, "Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation," *CoRR*, vol. abs/1503.01640, 2015.

[45] G. Papandreou, L. Chen, K. Murphy, and A. L. Yuille, "Weakly- and semi-supervised learning of a DCNN for semantic image segmentation," *CoRR*, vol. abs/1502.02734, 2015.

[46] A. Khoreva, R. Benenson, J. H. Hosang, M. Hein, and B. Schiele, "Weakly supervised semantic labelling and instance segmentation," *CoRR*, vol. abs/1603.07485, 2016.

[47] I. Adobe Systems, "Adobe photoshop user guide," 2002.

[48] E. N. Mortensen and W. A. Barrett, "Intelligent scissors for image composition," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 191–198, ACM, 1995.

[49] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," in *ACM transactions on graphics (TOG)*, vol. 23(3), pp. 309–314, ACM, 2004.

[50] Y. Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary & region segmentation of objects in nd images," in *Proceedings eighth IEEE international conference on computer vision. ICCV 2001*, vol. 1, pp. 105–112, IEEE, 2001.

[51] G. Friedland, K. Jantz, and R. Rojas, "Siox: Simple interactive object extraction in still images," in *Seventh IEEE International Symposium on Multimedia (ISM'05)*, pp. 7–pp, IEEE, 2005.

[52] TensorFlow, "Tensorflow detection model zoo," 2019. [Online; accessed 1-Oct-2019].

[53] G. Ros, S. Ramos, M. Granados, A. Bakhtiary, D. Vazquez, and A. Lopez, "Vision-based offline-online perception paradigm for autonomous driving," in *WACV*, 2015.

[54] Y. Lyu, L. Bai, and X. Huang, "Road segmentation using cnn and distributed lstm," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2019.

[55] G. L. Oliveira, W. Burgard, and T. Brox, "Efficient deep models for monocular road segmentation.," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[56] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel computing*, vol. 21, no. 8, pp. 1313–1325, 1995.

[57] "Waymo open dataset: An autonomous driving dataset," 2019.