Undergraduate Thesis

# Semantic Segmentation of Curbs and Curb Cuts from Street Imagery

Yvan Putra Satyawan

Examiner: Prof. Dr. Wolfram Burgard
Advisors: Jannik Zürn

Albert-Ludwigs-University Freiburg

Faculty of Engineering

Department of Computer Science

Chair for Autonomous Intelligent Systems

September 2nd, 2019

**Writing Period**

20. 4. 2019 – 2. 9. 2019

**Examiner**

Prof. Dr. Wolfram Burgard

**Advisors**

Jannik Zürn

# Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I also hereby declare, that my thesis has not been prepared for another examination or assignment, either in its entirety or excerpts thereof.

| | |
|---|---|
| Place, Date | Signature |

# Abstract

In this work we propose CurbNet, a deep learning model based on DeepLab v3+ [1] with a modified version of cross entropy loss, for the purpose of curb, curb cut, road, and sidewalk segmentation. Using the encoder-decoder architecture from DeepLab v3+, CurbNet is capable of segmenting curbs and curb cuts in street-level imagery and can be used to aid the pathfinding pipeline used by the Obelix robot [2]. In this work, we show that CurbNet is capable of performing this task, with its hyperparameters optimized using bayesian optimization and hyperband tuning [3] and trained on the Mapillary dataset [4].

# Acknowledgments

First and foremost, I would like to thank my parents Go Utama Handy Satyawan and Tuti Budiman for supporting me in my pursuit for a degree in Germany. Thank you for your support in giving me the opportunity to earn a Bachelor of Science in Computer Science. I would also like to thank my advisor Jannik Zürn for the support and feedback he has given me during both the experimental and the writing phase of my thesis. Thank you for helping me and being my guide in this journey through the sometimes opaque world of machine learning.

Huge thanks also to my friends Lukas König, Maximilian Roth, and Nina Pant for contributing ideas and their support throughout my time working on this thesis.

And finally, thank you to Prof. Dr. Wolfram Burgard for taking the time to look through my thesis and for any feedback you will provide.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Semantic segmentation is a popular and promising research topic in the field of computer vision, and particularly for autonomous vehicles. By semantically segmenting the scene, the autonomous vehicle is able to identify various objects in its field of view, which is important for autonomous vehicle and robots to safely navigate an environment. Generally, most implementations aim to segment road surfaces but in this thesis, we propose additionally the segmentation of curbs and curb cuts to allow safer sidewalk navigation. Sidewalks, roads, curbs, and curbs cuts will be jointly referred to as traversability classes for the remainder of the work.

The Europa project has resulted in the Obelix robotic platform, which has already been demonstrated to successfully perform pedestrian navigation in urban environments [2][5]. We propose to add to this platform the ability to detect traversability classes using semantic segmentation. The Obelix platform is the result of a joint project to build a robotic platform capable of autonomous robotic navigation [2]. In its current state, Obelix is already capable of localization and navigation using a map generated using LIDAR, but it is unable to determine what surface type it is driving on, e.g. asphalt, concrete, carpet, etc. [6]. Research is currently underway to extend its path finding ability, using the onboard cameras to classify terrains that Obelix will encounter [6]. This work specifically aims to add to said research by additionally identifying traversability, providing addtional constraints for the local trajectory planner. This will allow the local trajectory planner to identify curbs, which it cannot traverse, and curb cuts, which it can.


**Figure 1:** The Obelix Robot platform developed by the Europa project. We plan to extend its path finding capabilities using the work in this thesis.

The aim of this work is thus to implement a deep learning model capable of the semantic segmentation of the aforementioned traversability classes using a single RGB camera image. To this end, we propose the implemention of a convolutional neural network (CNN) based on the paper "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation" by Chen Liang Chieh et al [1]. We additionally use a modified loss function based on cross entropy loss. We call our proposed implementation CurbNet.

This thesis begins by discussing related works in Chapter 2. Background knowledge that will be important in understanding the approach taken and the work in general will be discussed in Chapter 3. The approach is then discussed in detail in Chapter 4, with a discussion regarding the architecture the model uses and the loss function used. The experiments performed are discussed in Chapter 5 and their results analyzed in section 5.7. Finally, a conclusion and a short discussion of potential future research is presented in Chapter 6.

# 2 Related Work

CurbNet uses semantic segmentation of RGB camera images to identify traversability classes. As such, related works can be divided into two categories: semantic segmentation and curb detection. The following is a discussion of relevant related works.

## 2.1 Image classification

The field of semantic segmentation using trainable neural network models started in 1989 with the pioneering work of Eckhorn et al. and their paper describing how the visual cortex of a cat functions and its implications for network models [7]. This early method uses a pulse coupled neural network, which produces synchronous bursts of pulses, effectively grouping the neurons by phase and pulse frequency, which can then be analyzed for feature extraction.

In the same year, Y. LeCun et al. developed the first algorithm to use backpropagation and convolutional neural networks to identify and classify images [8]. Their paper titled "Backpropagation Applied to Handwritten Zip Code Recognition" proposes using convolutional filters directly on an image input and training using backpropagation. This technique is able to reliably classify images into predetermined classes. This is the first network architecture that took a normalized image as input and returned the image class directly as an output. LeCun et al. also shows that using backpropagation

to learn the convolutional filter coefficients performed significantly better than using hand selected coefficients. This pioneering work set the stage for modern day image classification and segmentation algorithms using CNNs and automated learning.

"Very Deep Convolutional Networks for Large-Scale Image Recognition," published in 2014 by Karen Simonyan and Andrew Zisserman, is one of the earlier works to show that a deeper network setup could result in very high accuracy image classification [9]. Their setup improves upon the use of convolutional neural networks by proposing instead to use small ($3 \times 3$) convolutional filters and changing the depth to 16-19 layers. This network is now commonly known as VGG16 - VGG19, with the number corresponding to layer depth. This simple change in architectural design results in their architecture securing first and second place in the ImageNet Challenge 2014 localization and classification tracks respectively [9]. The authors do note that these very deep networks tended to overfit on smaller datasets and were difficult to train.

"Deep Residual Learning for Image Recognition," published in 2016 by Kaiming He et al., proposes a solution to this problem [10]. They reformulate the layers as learning residual functions and use the layer inputs as references. This architecture is now commonly known as ResNet. By doing so, they are able to empirically show that their residual networks are easier to optimize and have lower complexity despite being up to eight times deeper than VGG networks. ResNet was able to obtain a 28% relative improvement on the COCO object detection dataset compared to previous methods [11].

## 2.2 Semantic Segmentation

"Fully Convolutional Networks for Semantic Segmentation," published in 2015 by Jonathan Long et al., takes these classification networks and added fully convolutional layers to take the encoded features and use it for semantic segmentation [12]. This paper lays out a novel insight to the problem of image segmentation as nothing more

than a dense image classification problem. The proposed network architecture is now commonly referred to as FCN. In essence, they propose that image segmentation is nothing more than image classification on a per-pixel basis. As such, previously developed CNNs for image classification could be used as a feature encoder for semantic segmentation networks. They show that their network, using VGG16 as its feature encoder, was able to outperform competing state-of-the-art approaches on the PASCAL Visual Object Challenge dataset by a relative margin of 20% [13].

DeepLab v3+, proposed in "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," published in 2018 by Liang-Chieh Chen et al., improves on FCN by refining the segmentation, especially along object boundaries [1]. DeepLab v3+ became the basis of the network we used for CurbNet. Using pyramid pooling and an improved encoder-decoder architecture, DeepLab v3+ achieves 89.0% mean intersection over union (IoU) accuracy on the Cityscapes dataset [14].

## 2.3 Curb Detection

"Curb Detection for a Pedestrian Robot in Urban Environments" by Jérôme Maye, Ralf Kaestner, and Roland Siegwart, published in 2012, uses LIDAR sensors mounted on Obelix to map the world around it [15]. Using this point cloud data, a virtual representation of the environment can be computed and horizontal planes from the scene extracted. By detecting sudden changes in the vertical position of horizontal planes, a curb can be implicitly identified. This relies on the assumption that curbs take the form of vertical planes connecting two horizontal planes. This work does not take into account curb cuts, which may not necessarily form a significant vertical height difference and are usually sloped.

"WalkNet: A Deep Learning Approach to Improving Sidewalk Quality and Accessibility," published in 2018 by Andrew Abbott et al., specifically addresses the identification of curb cuts [16]. This paper proposes the use of a deep neural network

to classify images in which curb cuts exists. Their goal is the use of Google Street View data to map which intersections in a city have curb cuts and which do not. This data is then to be supplied to city governments to provide relevant information regarding sidewalk accessibility and quality. This work is focussed on the classification images which contained curb cuts and not the segmentation of said curb cuts. The neural network architecture they used was not described in depth.

# 3 Background

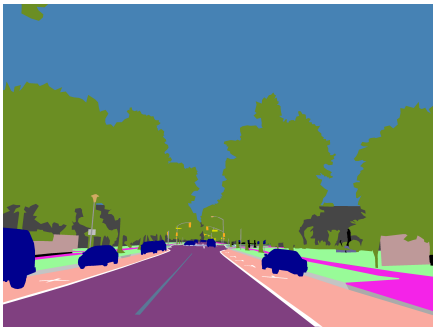
Some background knowledge is necessary for the reader to have an understanding of the principles behind semantic segmentation and convolutional neural networks. This chapter on the background of this work will discuss semantic segmentation, machine learning with artificial neural networks and convolutional neural networks, curb segmentation, loss functions, network optimizers, and binary dilation.

## 3.1 Semantic Segmentation

Unlike image classification, which classifies the contents of an image as a whole, semantic segmentation is the use of some algorithm to process an image and assign class labels to each individual pixel [17]. This adds the ability to locate and classify multiple objects in a given scene. For example, the ground truth segmentation of the street-level image in Figure 2a can be seen in Figure 2b. Each pixel of the image has been assigned a class, which is represented by the different colors in the segmented image. This allows a computer or program to understand what objects are in the image it is shown.

By segmenting an image in this way, the program can interpret the scene semantically. For example, by receiving the segmented image, a program can identify that there are line markings on the road and that there is a vehicle in front of it. The segmentation

**(a)** An example of a street-level image that is taken using a regular color camera.



**(b)** The segmented version of the image, each color representing a different class.

**Figure 2:** An example of semantic segmentation on a color image. This example is taken from the Mapillary dataset [4]. In this example, the RGB image in Figure 2a has been segmented by a human in order to produce the ground truth segmentation seen in Figure 2b. These image pairs can be used to train and validate the performance of a deep learning based semantic segmentation model.

of images in this way is essential in many robotic applications as it allows further higher level processing of the scene.

Towards the goal of this work, the segmentation of traversability classes in a scene allows Obelix to more accurately find a path allowing for the safe traversal from sidewalk to street level via curb cuts.

## 3.2 Artificial Neural Networks

Artificial neural networks are a computing system inspired by biological neurons. Neural networks are comprised of neurons which are capable of taking any number of numerical inputs and outputs a numerical value.

Mathematically, a single neuron is a function. The function of a neuron $j$ receiving input $x_j$ and producing output $y_j$ is composed of the activation $a_j$, an activation function $f_a$ which returns the activation, and an output function $f_o$. The activation

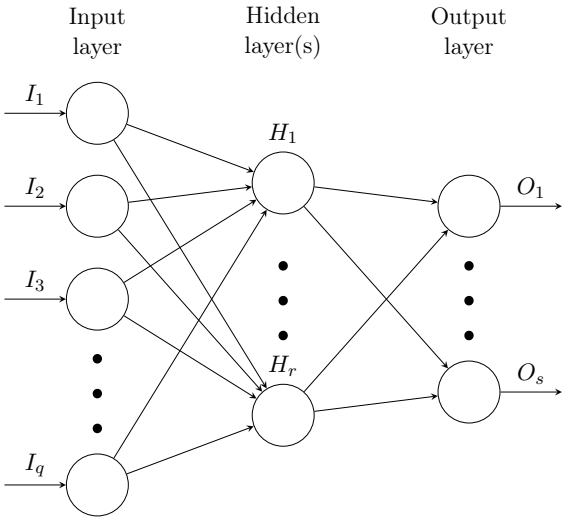



**Figure 3:** A Visualization of how neurons interact within an artificial neural network. In this example, the network has 3 layers: An input layer, a hidden layer, and an output layer. The input layer consists of $q$ neurons, each having one input value. Each of these inputs neurons are connected to each of the $r$ neuron in the hidden layer. Each neuron in the hidden layer is also connected to each neuron in the output layer. Finally, the output layer contains $s$ neurons and returns $s$ output values.

$a_j$ can also be considered the neuron's state. The activation function $f_a$ calculates $a_j$ given the network input $x_j$ and can be defined as:

$$a_j = f_a\left(x_j\right) \tag{1}$$

The output function $f_o$ computes $y_j$ based on $a_j$ and is defined as:

$$y_j = f_o\left(a_j\right) \tag{2}$$

Many activation functions exists and are used including the identity function and the rectified linear unit (ReLU), defined as:

$$f_a(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \tag{3}$$

Between each neuron in the network are connections which transfer the output of neuron $i$ to neuron $j$. Each of these connections are assigned a weight $w_{ij}$, which is computed by the learning algorithm. Each neuron also has a bias $w_{0j}$. These parameters are used to provide a neuron its input $x_j$. This is defined as:

$$x_j = \sum_i y_i w_{ij} + w_{0j} \tag{4}$$

Figure 3 shows a visualization of the interactions between multiple neurons.

Learning occurs by using an algorithm known as backpropagation to modify the parameters of the neural network. Further discussion on backpropagation can be found in Section 3.6.

Deep neural networks are so called deep due to having multiple "hidden" layers between the input and output which are not accessed directly.

### 3.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specific class of deep neural networks and have been found to perform exceptionally well for image analysis, such as image classification and segmentation. The inspiration for CNNs come from biological processes to simulate the organization of the visual cortex in animals [18]. Convolutional layers are the core building blocks of CNNs. These convolutional layers consist of a set of learnable filters which are convolved across the entire input and compute the dot products between the different entries of the filter.

By using multiple convolutional layers, higher level features can be extracted, generating a feature map. The module responsible for extracting the feature map is commonly known as the encoder.

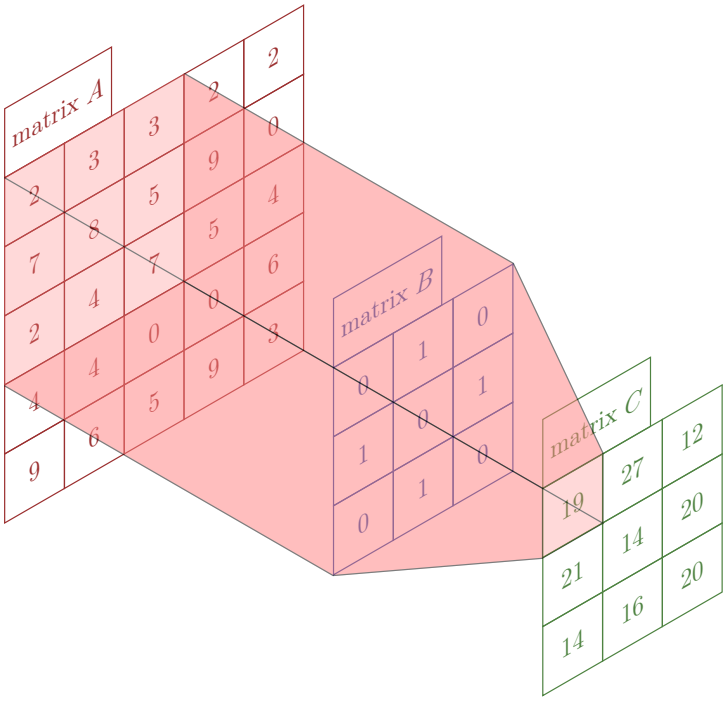



**Figure 4:** An example of a how a convolution on matrix $A$ would result in matrix $C$. In this example, a convolution operation is being performed on $A$ with a kernel $B$ of size $3 \times 3$.

The convolutional layers that make up the CNN are called the feature encoder. The output of these convolutional layers are then fed into a module with one or more fully convolutional layers to produce a classification of each pixel in a scene. These convolutional layers work by taking the deep feature maps produced by the encoder and using deconvolutional or upsampling layers to produce a log-likelihood vector for each pixel. This module is known as the decoder. The resulting vector produced is a probability of each pixel being a certain class. Applying an argmax operation on this vector produces a segmentation of the input image.

A bottleneck structure may also be used to increase performance. A bottleneck structure in a convolutional neural network reduces the number of features by performing a $1 \times 1$ convolution on a tensor before performing a convolution operation. A final $1 \times 1$ convolution is then performed to expand the tensor to contain a meaningful com-

bination features. As the input features are correlated, redundancy can be removed by performing the first $1 \times 1$ convolution.

For example, performing a $3 \times 3$ convolution on a $256 \times 256$ input tensor requires more than 589,000 operations. A bottleneck structure can be used to reduce the dimensionality of the input tensor to only 64 dimensions using a $1 \times 1$ convolution. The $3 \times 3$ convolution is then performed on this tensor. Finally, the resulting tensor is re-expanded back to the original 256 dimensions using another $1 \times 1$ convolution. This would result in slightly over 69,000 operations, significantly less than without the bottleneck structure.

This reduces both the number of operations required as well as the number of parameters in the network.

## 3.3 Curbs and Curb Cuts

Curbs are the concrete or stone edging of a road. Curbs usually separate the road from some other area with a different elevation, usually a pedestrian sidewalk. An example of a sidewalk with a curb can be seen in Figure 5a. Curb cuts are "cuts" in the curb that allow a pedestrian sidewalk to have a gentle slope down to street level. An example of a sidewalk with a curb cut can be seen in Figure 5b. Originally, these cuts were made to allow accessibility access, especially for those requiring wheelchairs. To a wheelchair user, curbs represent a significant barrier in terms of traversability, as was discussed in the article "Curb Cuts" by Cynthia Gorney and Delaney Hall [19]. Curb cuts first started appearing fifty years ago from the efforts of activist Ed Roberts and his push to make city streets more accessible.

Wheeled robots are also unable to traverse curbs. They therefore require curb cuts to traverse urban environments where curbs are present.

**(a)** An example of a sidewalk with a curb, the curb highlighted in blue.

**(b)** An example of a sidewalk with a curb cut, the curb cut highlighted in red.

**Figure 5:** An example of a curb and a curb cut. Being able to segment these structures is one of the goals of this thesis.

## 3.4 Loss Functions

The loss function $\ell$ maps the output of a neural network $\hat{y}$ and the target $y$ onto a real number and represents the "cost" of an output. The goal of learning is to minimize this cost value. Effectively, the loss function measures the difference between the predicted value and the target. In our case, the target value is the ground truth labeling of an image and the output is the softmax class predictions from the network. There are many different loss functions that are commonly used. For the purposes of image segmentation, the most commonly used function is cross entropy loss.

### 3.4.1 Cross Entropy Loss

The cross entropy loss is formally defined as:

$$\ell(y, \hat{y}) = \sum_{m} -y_m \log(\hat{y}_m) \tag{5}$$

where $m$ is the class, $y_m$ the ground truth value for a class $m$, and $\hat{y}_m$ the softmax prediction of a class $m$ by the model. As such, cross entropy loss calculates the error of the model to classify a certain value correctly. The loss of two predictions could be

the same, as long as the prediction for the true class, i.e. the class that has value 1 in $y$, remains constant. The loss is thus independent of how the probability is split between the remaining classes.

For example, let the ground truth classification be class 0, i.e. $y = \{1, 0, 0\}$ with one-hot encoding. Let the current model prediction be $\hat{y} = \{0.5, 0.2, 0.3\}$, also with one-hot encoding. The loss would then be:

$$\begin{aligned} \ell(y, \hat{y}) &= \sum_m -y_m \log(\hat{y}_m) \\ &= -(1)\log(0.5) + (-0)\log(0.2) + (-0)\log(0.3) \\ &= -\log(0.5) \\ &= 0.301... \end{aligned} \tag{6}$$

Alternatively, if model prediction is exactly the same as the ground truth, then we would have:

$$\begin{aligned} y = \hat{y} &= \{1, 0, 0\} \\ \ell(y, \hat{y}) &= \sum_m -y_m \log(\hat{y}_m) \\ &= -(1)\log(1) + (-0)\log(0) + (-0)\log(0) \\ &= -\log(1) \\ &= 0 \end{aligned} \tag{7}$$

Applying this on the full image, this becomes:

$$\ell(y, \hat{y}) = \sum_u \sum_v \sum_m -y_{m,u,v} \log(\hat{y}_{m,u,v}) \tag{8}$$

where $u$ and $v$ are the pixel coordinates.

The weighted variant, known simply as weighted cross entropy loss, is defined as:

$$\ell(y, \hat{y}) = \sum_{m} -y_m \log(\hat{y}_m) \cdot d_m \tag{9}$$

where $d_m$ is the weight for a given class $m$.

Using the weighted version allows changes to how much each class contributes to the loss. This is done to counteract class imbalance that could be caused by having an imbalance in the distribution or frequency of each class in the dataset.

## 3.5 Optimizers

During training, network parameters must be updated to minimize the loss function at each iteration. This is done by the optimizer. The optimizer updates the network parameters in such a way as to minimize the loss function [20].

Gradient Descent is one of the earliest optimizers. It works by calculating what a small change in each network parameter would do to the loss function, i.e. determines the gradient of the loss for the current iteration. It then adjusts the network parameters according to this gradient. This process is repeated at each iteration to further minimize the loss function. This process can be time-consuming, especially with larger datasets. Stochastic Gradient Descent (SGD) is a more commonly used variant of gradient descent and works similarly, but working only on randomly selected batches of the training data at each iteration [20]. Due to the large size of datasets, training is usually done in batches. By using SGD, the optimization step can occur after processing every batch, rather than after the entire dataset is processed. This speeds up with respect to wall-clock time network training.

Adaptive Moment Estimation (Adam) is another optimizer that is commonly used and first proposed in the paper "Adam: A method for stochastic optimization" by

Diederik Kingma and Jimmy Ba [21]. It combines concepts from Adaptive Gradient Algorithm (AdaGrad), which has per-parameter learning rates, and Root Mean Square Propagation (RMSProp), whose learning rates are adapted based on the average of the magnitude of recent gradients [21]. Adam utilizes the concept of momentum, which incorporates previous gradients into the current one. It does this by calculating an exponential moving average and the square of previous gradients and using these to determine the next gradient.

## 3.6 Backpropagation

Backpropagation is a learning procedure used in artificial neural networks to adjust network weights and produce internal representations of "important features in the task domain," first described in 1986 by David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams in their paper "Learning representations by back-propagating errors" [22]. Backpropagation efficiently and repeatedly adjusts the network weights to minimize the network error, which is calculated by the loss function.

Calculating the network weights is done layer by layer, starting with the final output layer. The partial derivative of $\ell$ with respect to $y_j$, intuitively the effect the output unit has on the loss, is defined as:

$$\frac{\partial \ell}{\partial y_j} \tag{10}$$

where $y_j$ is the output of a single unit in the final layer. The contribution of the input of the unit $j$ on the error can then be calculated using the value previously calculated in Equation (10) using the chain rule for differentiation as

$$\frac{\partial \ell}{\partial x_j} = \frac{\partial \ell}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} \tag{11}$$

We now have a description of how changing the input of unit $j$ will affect the loss. As such, we can then also represent how changing the weight $w_{ji}$ of a connection between the unit $j$ and a unit in the previous layer $i$ will effect the error as

$$\frac{\partial \ell}{\partial w_{ji}} = \frac{\partial \ell}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ji}} \tag{12}$$

In this case, the previous layer means the layer which during forwards-propagation would be calculated immediately before the layer in which unit $j$ is. Again, due to the use of the chain rule, the previous value calculated in Equation (11) can be used here. This gives us the contribution of the weight $w_{ji}$ on the error and, by multiplying by the learning rate, the amount by which the gradient must be changed for the next training iteration.

The error contribution of the output of unit $i$ via a single successor unit $j$ can also similarly be calculated using the value from Equation (11).

$$\frac{\partial \ell}{\partial y_i} = \frac{\partial \ell}{\partial x_j} \cdot w_{ji} \tag{13}$$

Since unit $i$ is connected to multiple successor units, its value is a summation of its output contribution via all units $j \in J$, where $J$ is the set of all units in successive layers that $i$ is connected to.

$$\frac{\partial \ell}{\partial y_i} = \sum_{j \in J} \frac{\partial \ell}{\partial x_j} \cdot w_{ji} \tag{14}$$

This calculation can be done in parallel, thanks to parallel computing architecture, completing the gradient calculations for the penultimate layer. This procedure is then repeated for all layers, each time using the previously computed values to allow for efficient computation.

## 3.7 Hyperparameter Tuning

Hyperparameter tuning is the process of optimizing a model to find the hyperparameters that will perform optimally given a specified metric. This metric is usually either the validation loss or the validation accuracy. Hyperparameters are generally chosen by hand or by some automated process, but are not learned as a part of the model itself as opposed to the model weights.

Methods to find the optimal hyperparameters include manual tuning, grid search, random search, genetic algorithms, bayesian optimization, and hyperband, among others. Random and grid searches are very time-consuming and computationally expensive, as they potentially run less than optimal parameters and do not use knowledge of previous runs to select the next set of parameters. Bayesian optimization attempts to curtail this problem by taking previous runs into account, greatly speeding up the learning process and not experimenting with less than optimal hyperparameters [23]. Hyperband is a bandit-based approach to hyperparameter optimizations that can abandon or reduce the budget of an experimental run if the network performance is below with a chosen parameter set is below a certain threshold [24].

### 3.7.1 Bayesian Optimization

The idea behind Bayesian optimization is that the hyperparameters of a network follow a probabilistic distribution $\mathrm{P}(y|x)$ where $y$ is the score determined by some objective function and $x$ is a set of hyperparameters. Based on this probabilistic model, an optimal set of hyperparameters can then be calculated. Applying these hyperparameters to the model, the objective function can then be used to calculate the scores. The probabilistic distribution is then updated and the process is repeated until the maximum number of iterations is reached.

Bayesian optimization relies on the following [25]:

1. A clearly defined hyperparameter domain of possible configurations $\chi$ over which to search,
2. An objective function which calculates a score that we wish to minimize using a set of hyperparameters,
3. A surrogate model that represents the probabilistic distribution of the hyperparameters,
4. A selection function to evalute which hyperparameters should be evaluated next, and
5. A history of score-hyperparameter pairs used to update the surrogate model.

The domain must be chosen by the researcher and is usually defined with respect to previous knowledge or based on related works. The objective function is evaluated by running the model and calculating its error using a predetermined loss function.

The surrogate model used is essentially a mapping of hyperparameters to scores from the objective function. An example for the surrogate model is the Tree-structured Parzen Estimator (TPE) [26]. TPE takes advantage of Bayes' rule and represents the probability function $\mathrm{P}(y \mid x)$ as

$$\begin{aligned} \mathrm{P}(y \mid x) &= \frac{\mathrm{P}(x \mid y) \cdot \mathrm{P}(x)}{\mathrm{P}(y)} \\ \text{with } \mathrm{P}(x \mid y) &= \begin{cases} l(x) & \text{when } y < y^* \\ g(x) & \text{when } y \geq y^* \end{cases} \end{aligned} \tag{15}$$

where $y^*$ is a threshold value. $l(x)$ is thus a probability density function formed from the set of observations using hyperparameters $x^i \in \chi$ which have been performed such that the loss of the network is less than $y^*$. Conversely, $g(x)$ is the density function formed from the remaining observations. $y^*$ is chosen by the algorithm such that $\mathrm{P}(y < y^*) = \gamma$ with $\gamma$ being a value chosen by the researcher.

The selection function chooses which hyperparameters $x \in \chi$ should be chosen for each successive experiment and is commonly based on the expected improvement [25]:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y \mid x)dy \tag{16}$$

Here, $y^*$ is again a threshold value, $x$ is the proposed set of hyperparameters, $y$ is the actual value of the objective function using hyperparameters $x$ and $p(y \mid x)$ is the surrogate probability model expressing the probability of $y$ given $x$. Substituting the values of $p(y \mid x)$ from the surrogate function we get

$$\begin{aligned} EI_{y^*}(x) &= \frac{\gamma y^* l(x) - l(x) \int_{-\infty}^{y^*} p(y)dy}{\gamma l(x) + (1-\gamma)g(x)} \\ &\propto \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1} \end{aligned} \tag{17}$$

We can see from this function that the expected improvement is proportional to $\frac{l(x)}{g(x)}$ and thus, to maximize the expected improvement, samples should be drawn from the maximum value in $l(x)$. The selected hyperparameters are then evaluated with regards to the objective function and the results used to update the surrogate model.

### 3.7.2 Hyperband

Hyperband is a learning strategy that "relies on a principled early-stopping strategy to allocate resources" [24]. It is a variation of successive halving, and in fact uses successive halving in its inner loop [27].

Successive halving randomly samples $n$ hyperparameter sets in the search domain $\chi$. It then evaluates all of these sets for $B$ iterations to calculate the validation loss. The lowest performing half is then discarded and the remaining evaluated for a further $B$ iterations. This is repeated until only one hyperparameter set remains.

Successive halving suffers from the $n$ vs $\frac{B}{n}$ problem, which is whether to train more configurations $n$ or to explore fewer, but with more resources $B$. If a larger $n$ is chosen, configurations which are slower to converge might be killed off too quickly. If a larger $\frac{B}{n}$ is chosen, then lower performing configurations may be given more resources, thus wasting resources that could otherwise be spent on higher performing configurations [24].

Hyperband attempts to solve this issue by considering several possible values of $n$ for a fixed $B$. The pseudocode in Algorithm 1 shows how Hyperband uses an outer loop, performing essentially a grid search with multiple values of $n$ and an inner loop utilizing a method similar to successive halving but with the top $\lfloor n_i/\eta \rfloor$ instead of the top half.

**Algorithm 1:** HYPERBAND algorithm for hyperparameter optimization

**Input** : $R$, $\eta$ (default $\eta = 3$)
**Output** : Configuration with the smallest intermediate loss seen so far.
**Initialize** $s_{max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{max} + 1)R$
**for** $s \in s_{max}, s_{max} - 1, ..., 0$ **do**
  $n = \left\lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \right\rceil$
  $r = R\eta^{-s}$
  $T =$ get_hyperparameter_configuration$(n)$
  **for** $i \in \{0, ..., s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L =$ {run_then_return_validation_loss$(t, r_i) : t \in T$}
    $T =$ top_k$(T, L, \lfloor n_i/\eta \rfloor)$
  **end**
**end**

### 3.7.3 Bayesian Optimization with Hyperband

"BOHB: Robust and Efficient Hyperparameter Optimization at Scale" by Stefan Falkner, Aaron Klein, and Frank Hutter proposes a hyperparameter optimization approach which seeks to combine the benefits of both Bayesian optimization and

Hyperband [3]. This approach seeks to have strong anytime performance, strong final performance, effectively use parallel resources, be easily scalable, and be robust and flexible.

BOHB works by using Hyperband to determine the number of configurations to evaluate with a given budget, but replaces the random selection of configurations with a Bayesian model based search. The process of using Bayesian optimization for configuration sampling can be seen in Algorithm 2. In this algorithm, $l'(x)$ is the $l(x)$ component of the newly updated probability density function.

Even though Hyperband also provides strong anytime performance compared to random search and Bayesian optimization, Fakner et al. observed an improvement of over 55x against a random search at larger budgets, converging to the global optimum much faster than either Hyperband or Bayesian optimization alone [3].

**Algorithm 2:** BOHB algorithm for hyperparameter optimization

**Input :** observations $D$, fraction of random runs $\rho$, percentile $q$, number of samples $N_s$, minimum number of points $N_min$ to build a model, and bandwidth factor $b_w$
**Output :** next configuration to evaluate
**if** *rand()* $< \rho$ **then**
  **return** *random configuration*
**end**
$b = \arg\max \{D_b : |D_b| \geq N_min + 2\}$
**if** $b = \emptyset$ **then**
  **return** *random configuration*
**end**
fit $\mathrm{P}(x \mid y)$ according to Equation (15)
draw $N_s$ samples according to $l'(x)$
**return** *sample with highest ratio* $l(x)/g(x)$

## 3.8 Binary Dilation

Binary dilation is a basic morphological operation and is usually represented by the operator $\oplus$. With regards to this work, binary dilation is used in our loss function.

For a given binary image viewed as an integer grid $\mathbb{Z}^d$ for some dimension $d$, let $E$ be an integer grid, $A \in E$ a binary image, and $B \in \{0,1\}^d$ a structuring element. The binary dilation of $A$ by $B$ is then defined as:

$$A \oplus B = \bigcup_{b \in B} A_b \tag{18}$$

where $A_b$ is the translation of $A$ by $b$. This can be seen as extending the area of the binary image $A$ by locus of the points covered by $B$ given that $B$ has a center on the origin [28].

# 4 Approach

To semantically segment urban scenes and extract predictions for traversability classes, we proposed to use a deep neural network based on the DeepLab v3+ architecture with a loss function inspired by "U-Net: Convolutional Networks for Biomedical Image Segmentation" by Olaf Ronneberger et al. [29]. In this chapter, the architecture used and an explanation of our loss function will be discussed.

## 4.1 Architecture Selection

We came to use the DeepLab v3+ architecture after running these experiments as it outperformed other networks that were tested. A further discussion of the results can be found in section and the experiments done to select the network can be found in Section 5.3.

### 4.1.1 Network Architecture

We propose to apply the DeepLab v3+ architecture to our stated problem of curb and curb cut segmentation. The architecture itself is given in Figure 6 with further descriptions in Tables 1-4[1].

[1]The full implementation written using the PyTorch framework can be found at `github.com/yvan674/CurbNet`

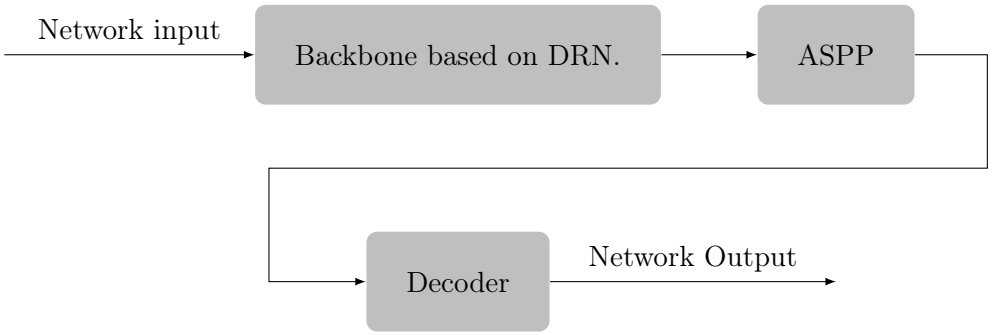



**Figure 6:** A simplified visualization of the network architecture. The contents of each module is listed in Tables 1-4.

| Layer | Module | Parameters |
|---|---|---|
| 0 | Convolution + ReLU | in = 5, out = 16, kernel = 7, stride = 1 |
| 1 | Convolution + ReLU | in = 16, out = 16, kernel = 3, stride = 1 |
| 2 | Convolution + ReLU | in = 16, out = 32, kernel = 2, stride = 1 |
| 3 | Bottleneck | in = 32, low = 64, out = 256 |
| | Bottleneck | in = 256, low = 64, out = 256 |
| | Bottleneck | in = 256, low = 64, out = 256 |
| 4 | Bottleneck | in = 256, low = 128, out = 512 |
| | Bottleneck | in = 512, low = 128, out = 512 |
| | Bottleneck | in = 512, low = 128, out = 512 |
| | Bottleneck | in = 512, low = 128, out = 512 |
| 5 | Bottleneck | in = 512, low = 256, out = 1024 |
| | Bottleneck | in = 1024, low = 256, out = 1024 |
| | Bottleneck | in = 1024, low = 256, out = 1024 |
| | Bottleneck | in = 1024, low = 256, out = 1024 |
| | Bottleneck | in = 1024, low = 256, out = 1024 |
| | Bottleneck | in = 1024, low = 256, out = 1024 |
| 6 | Bottleneck | in = 1024, low = 512, out = 2048 |
| | Bottleneck | in = 2048, low = 512, out = 2048 |
| | Bottleneck | in = 2048, low = 512, out = 2048 |
| 7 | Convolution + ReLU | in = 2048, out = 512, kernel = 3, stride = 1 |
| 8 | Convolution + ReLU | in = 512, out = 512, kernel = 3, stride = 1 |

**Table 1:** Dilated Residual Network (DRN) Module Details

| Layer | Module | Parameters |
|---|---|---|
| 0 | Convolution | in = in, out = low, kernel = 1, stride = 1 |
| 1 | Convolution | in = low, out = low, kernel = 3, stride = 1 |
| 2 | Convolution + ReLU | in = low, out = out, kernel = 1, stride = 1 |

**Table 2:** Bottleneck Module Details

| Layer | Module | Parameters |
|---|---|---|
| 0 | Convolution + ReLU | in = 512, out = 256, kernel = 1, stride = 1 |
| 1 | Convolution + ReLU | in = 512, out = 256, kernel = 1, stride = 1 |
| 2 | Convolution + ReLU | in = 512, out = 256, kernel = 1, stride = 1 |
| 3 | Convolution + ReLU | in = 512, out = 256, kernel = 1, stride = 1 |
| 4 | Global Average Pool<br>Dropout | in = 1280, out = 256<br>$p = 0.5$ |

**Table 3:** Atrous Spatial Pyramid Pooling (ASPP) Module Details

| Layer | Module | Parameters |
|---|---|---|
| 0 | Convolution + ReLU | in = 256, out = 48, kernel = 1, stride = 1 |
| 1 | Convolution + ReLU<br>Dropout | in = 304, out = 256, kernel = 3, stride = 1<br>$p = 0.5$ |
| 2 | Convolution + ReLU<br>Dropout | in = 256, out = 256, kernel = 3, stride = 1<br>$p = 0.5$ |
| 3 | Convolution | in = 256, out = 3, kernel = 1, stride = 1 |

**Table 4:** Decoder Module Details

## 4.2 Loss Function

We chose to use a modified weighted cross entropy loss due to the severe class imbalance. Using the assumption that all curbs and curb cuts must be located along the perimeter of roads, we used a loss function inspired by the paper "U-Net: Convolutional Networks for Biomedical Image Segmentation," which we call Masked Cross Entropy (MCE) and is defined in Equation (21) [29]. The weighted cross entropy loss function was modified to penalize according to the given weights when labeling within a certain border around road classes, which we call the mask $M$. We define road classes $\text{class}_{\text{road}}$ as all classes which can reasonably be expected to be found on roads, including road, road markings, potholes, etc. This mask was calculated using a binary dilation on a full $b \times b$ matrix $B$ where $b$ is $0.05 \times \text{image}_{width}$ on the road class, then subtracted by the road class itself. Thus, this can be formalized as follows:

$$B = \underbrace{\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}}_{b \text{ columns and } b \text{ rows}} \tag{19}$$

$$M = (\text{class}_{\text{road}} \oplus B) - \text{class}_{\text{road}} \tag{20}$$

A visualization of this mask applied to a color street-level image from the Mapillary dataset results in Figure 7.

The value 0.05 was chosen empirically after looking at samples in the dataset and measuring what area around roads are typically curbs. The road class was simply taken from the ground truth data. Any labeling outside of $M$ by the network is then given an increased penalty, incentivizing the network to focus labeling around road edges. We chose to multiply the penalty for areas outside $M$ by a factor of 3. This

**Figure 7:** An example of the resulting mask that is used by masked cross entropy loss. The areas in the image that are marked with a blue overlay are the areas contained within the mask. The street-level color image was taken from the Mapillary dataset [4]

can be seen in the following formalization of the loss function we used:

$$\ell_{MCE}(y, \hat{y}) = \sum_{m} -y_m \log(\hat{y}_m) \cdot d_m \tag{21}$$

$$\text{with } d_m = \begin{cases} d'_m & \text{if } x \in M \\ d'_m \cdot 3 & \text{if } x \notin M \end{cases} \tag{22}$$

where $d'_m$ are the user defined weights. This loss function operates on the assumption that curbs and curb cuts must be located adjacent to roads. A visualization of the resulting mask $M$ can be seen in Figure 7.

# 5 Experiments

A series of experiments were carried out to determine the performance of the network in its stated goal of classifying traversability. To this end, the network was tested on the Mapillary Vistas dataset [4] and its hyperparameters optimized using the Bayesian optimization with hyperband (BOHB) procedure[3]

## 5.1 Evaluation Metrics

The mean Intersection over Union (mIoU) is the main evaluation metric chosen. IoU measures the intersection of the prediction and the ground truth data, divided by the union of the prediction and the ground truth data. This is formally defined as

$$J(A_m, B_m) = \frac{|A_m \cap B_m|}{|A_m \cup B_m|} \tag{23}$$

where $A$ is the prediction, $B$ the ground truth, and $m$ the class. The mIoU is thus defined as

$$J(A, B) = \frac{1}{|m|} \sum_{m} \frac{|A_m \cap B_m|}{|A_m \cup B_m|} \tag{24}$$

and can be intuitively understood as the average of the class-wise IoU accuracy values.

## 5.2 Dataset

We evaluate our approach on the Mapillary Vistas dataset as it was, to our knowledge, the only street-level dataset that included ground truth segmentations of both curbs and curb cuts at the time of writing. This dataset contains images from all around the world and includes images captured from different imaging devices including mobile phones, action cameras, and professional imaging solutions. These images are captured during various weather, seasonal, and daylight conditions. As such, the dataset provides a challenging level of diversity.

Some image samples from the dataset can be seen in Figure 8. These images show some examples of how diverse the images in the dataset can be. The dataset also contains images taken from sidewalks, such as in the case of Figure 8c. This is especially useful for our end goal of training a segmentation network for an autonomous pedestrian robot. The diversity of the dataset improves the generalizability of our model.

The image dimensions in the dataset are also not standardized, with 156 different image dimensions ranging from $(640 \times 480)$ to $(6528 \times 5248)$. With such a diverse range of dimensions, the images must first be preprocessed. We process all the images to conform to a 4:3 size ratio, eliminate images without images of curbs or curb cuts, and extract only road, curb, and curb cut classes.

To optimize training speed with respect to wall clock time, and given the computational constraints, images were resized for training. The chosen image resolution was $360 \times 320$ pixels. We use an NVIDIA Titan X GPU with 12 GB of VRAM. This allows a batch size of 16 per GPU. To further optimize training, the images are resized prior to training and the resized images stored. This allows all training and validation images to be loaded into memory once at the beginning of training, reducing the number of file accesses required and reducing training wall clock time by a factor of two.

**(a)** This image shows an axample of an evening scene.

**(b)** This image shows an example of a very bright scene in a rural area.

**(c)** This image shows an example of an image from a sidewalk.

**(d)** This image shows an example of an image taken during rainy conditions.

**Figure 8:** These sample images from the Mapillary dataset shows how varied and diverse the dataset is. This diversity makes the dataset an excellent choice in training a network that can generalize to unseen conditions.

After running an analysis of the dataset with respect to their curb and curb cut content and image dimensions, we found that there were 15,160 usable images in the training set and 1610 usable images in the validation set. Usable images in this case refers to images that at least contain both curbs and curb cuts. A detailed results of the analysis of the dataset can be found in Appendix section 7.2.

We also applied image augmentation to our dataset. The augmentations we have applied are Gaussian blur, dropout, brightness adjustment, and contrast adjustment.

## 5.3 Network Evaluation

Experiments were done to evaluate the performance of different networks on our dataset. Specifically, we identified the small size of curbs relative to the rest of the image causes a severe class imbalance and increases the difficulty in recognizing the smaller features.

Relative to the entire image, curb and curb cut classes make up a relatively small proportion of the image, making up an average of 0.986% and 0.196% of images respectively. As such, three networks with good known performance for urban scene segmentation were chosen and evaluated for their performance classifying traversability classes. We experimented with GoogLeNet, described in the paper "Going Deeper with Convolutions" [30]; FCN16s, described in "Fully Convolutional Networks for Semantic Segmentation" [12]; and DeepLab v3+, described in the paper "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation" [1].

To evaluate which network would have the most potential, we trained each network on a small subset of the whole dataset to find which networks would yield the highest overall accuracy given a certain budget. We use a subset of 64 images, sampled randomly from the Mapillary Vistas training subset, for training and a subset of

64 images, also sampled randomly from the Mapillary Vistas validation subset, for validation. For each of these runs, the default recommended hyperparameters from the DeepLab v3+ paper were used. Due to time constraints, conduct hyperparameter optimization was not done on each of the tested networks. The results can be seen in Figure 9 and in Table 5.

We observe that DeepLab v3+ starts with the lowest mIoU performance but quickly achieves 28.61% mIoU accuracy after only 80 iterations, already outperforming the final result of GoogLeNet, after which training begins to plateau. After 1000 iterations, it was able to achieve 30.83% mIoU accuracy. GoogLeNet is able to also train relatively quickly, achieving 22.05% mIoU accuracy after 80 iterations, after which performance also began to plateau achieving a final 25.08% mIoU accuracy. FCN16s was able to achieve 19.97% mIoU overall after 1000 iterations. We believe this may be due to FCN16s not being designed to identify smaller structures and features, such as in the case of curbs and curb cuts.

Given these results, we chose to use DeepLab v3+ as our network.

## 5.4 Loss Function Evaluation

Our custom loss function masked cross entropy (MCE) loss was evaluated against weighted cross entropy (WCE) loss. We evaluated the two loss functions using the same network hyperparameters and a pretrained backbone. The backbone is

| Network | mIoU |
|---|---|
| DeepLab v3+ | 30.83% |
| FCN16s | 19.97% |
| GoogLeNet | 25.08% |

**Table 5:** Results of running the DeepLab v3+, FCN16s, and GoogLeNet on a subset of the main dataset after one thousand iterations. It can be seen that DeepLab v3+ was able to achieve the highest mIoU performance overall.

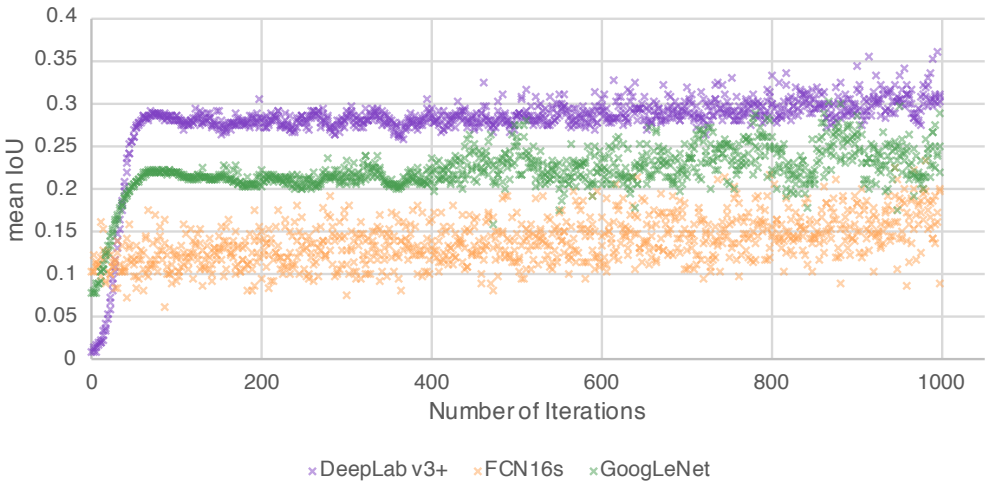
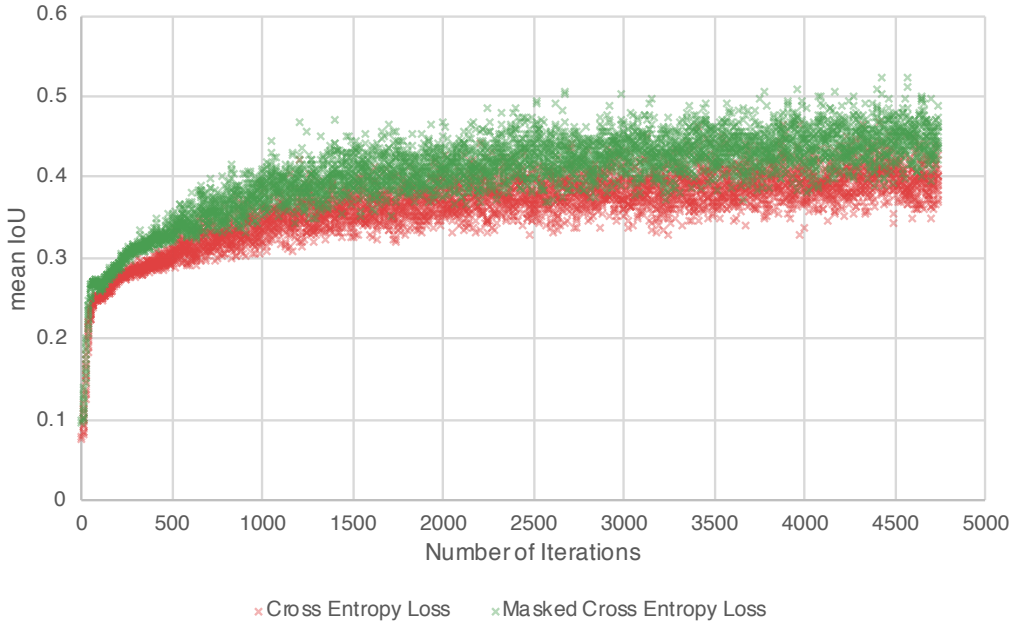



**Figure 9:** Training mIoU Accuracy over 1000 Iterations using DeepLab v3+, FCN16s, and GoogLeNet. It can be seen that DeepLab v3+ achieves the best mIoU performance overall.

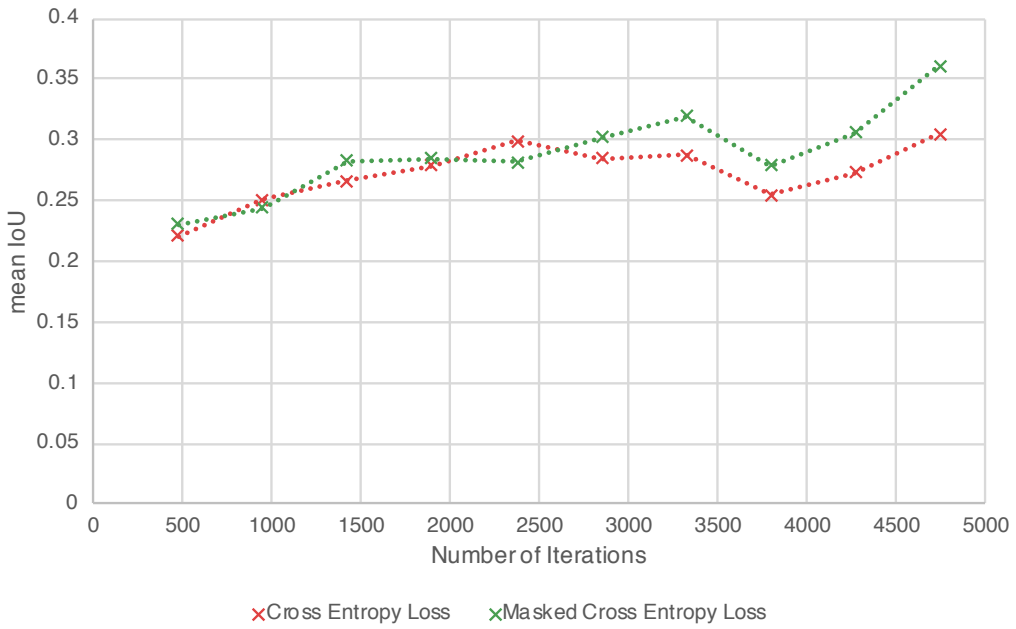
pretrained on the ImageNet database, published in 2009 by J. Deng et al [31]. Training results, shown in Figure 10a, show that both lost functions show similar results until the $40^{\text{th}}$ iteration, after which the network using MCE loss is able to produce better results. Validation accuracy is plotted in Figure 10b. We observe the validation accuracy to be similar for the first 2370 iterations. After this point, MCE loss shows a clear advantage in validation accuracy compared to WCE loss. After 4740 iterations, a clear performance advantage can be seen, with a difference of 5.56 mIoU percentage points. As such, we have chosen to implement our network using MCE loss as our loss function.

## 5.5 Hyperparameters

Hyperparameter tuning was done by using Bayesian Optimization with HyperBand. The hyperparameters that were tuned were the learning rate; optimizer; momentum, if the optimizer chosen was stochastic gradient descent; epsilon, if the optimizer chosen was Adam; and the loss criterion.




**(a)** Training mIoU Accuracy over 4740 Iterations comparing weighted cross entropy loss and masked cross entropy loss. The results show that although performing similarly to begin with, masked cross entropy loss was able to outperform weighted cross entropy loss.




**(b)** Validation mean IoU Accuracy over 4740 Iterations, measured every 474 iterations, comparing cross entropy loss and masked cross entropy loss. Validation accuracy was evaluated against a validation set of 1680 images. The lines shown in the chart are to emphasize the data.

**Figure 10:** Comparisons of masked cross entropy loss and weighted cross entropy loss. Both loss functions were evaluated using the same hyperparameters against a dataset consisting of 15160 training images and 1680 validation images.

| Parameter Name | Parameter Type | Range | Comments |
|---|---|---|---|
| Learning rate | Float | $\left[1 \times 10^{-5}, 1 \times 10^{-2}\right]$ | On a logarithmic scale |
| Optimizer | Categorical | Adam or SGD | |
| SGD momentum | Float | $[0, 0.99]$ | Only active when the optimizer is SGD |
| Adam epsilon | Float | $\left[1 \times 10^{-2}, 1\right]$ | Only active when the optimizer is Adam |
| Loss criterion | Categorical | Weighted cross entropy or masked cross entropy | |

**Table 6:** The hyperparameters and their possible value ranges which we optimized for using Bayesian Optimization and HyperBand. These hyperparameters have been chosen as we believe that they will have the greatest impact on network performance.

The learning rate is optimized between the interval $\left[1 \times 10^{-5}, 1 \times 10^{-2}\right]$, varied on a logarithmic scale. The optimizer is a categorical choice between stochastic gradient descent or Adam. The momentum is optimized between the interval $[0, 0.99]$. The epsilon value is optimized between the interval $\left[1 \times 10^{-2}, 1\right]$. The weight ratio for the loss weight determines the ratio of the weights between the curb and curb cut. The loss criterion is a categorical choice between weighted cross entropy loss or masked cross entropy loss. Further details of the hyperparameters optimized can be found in Table 6.

The hyperparameters are optimized according to the above parameters using a cluster of 4 GPUs over a budget of 4 iterations. The resulting optimized values for the hyperparameters are in Table 7. More detailed results of each run of the hyperparameter optimization can be found in Appendix section 7.1.

| Parameter Name | Value |
|---|---|
| Learning rate | $8 \times 10^{-5}$ |
| Optimizer | Adam |
| SGD momentum | - |
| Adam epsilon | 0.4 |
| Loss criterion | Masked cross entropy loss |

**Table 7:** Each hyperparameters and their respective values after tuning with Bayesian Optimization and HyperBand.

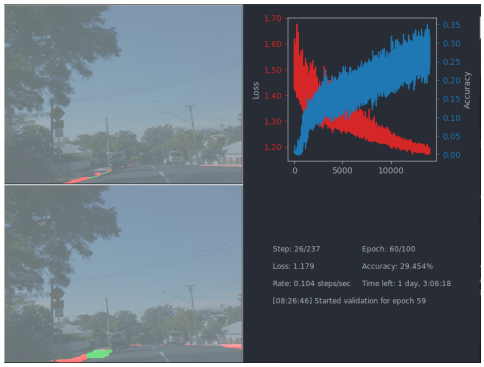
## 5.6 Implementation

The training pipeline is a main script which calls the training loop with a set of parameters defined in either a JSON file or through command-line arguments. Using a JSON file makes it simpler to change parameters as nothing is hard-coded and command-line arguments do not have to be memorized or meticulously typed out each time.

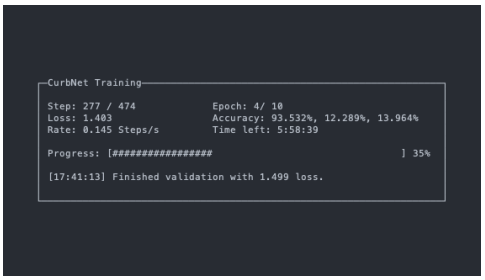
A Graphical User Interface (GUI) is also available, made using the tkinter framework, which is capable of visualizing the ground truth segmentation, the network output, a live display of the current loss and mIoU accuracy, and other statistics. This GUI can be seen in Figure 11a.

A Command Line Interface (CLI) is also available, made using the ncurses framework, which is capable of displaying various statistics during training. The CLI also allows training remotely using secure shell (SSH), as the GUI would fail to start if invoked remotely through the command line. A screen capture of the CLI can be seen in Figure 11b

The different components necessary to train the network are written in such a way as to be modular, with nearly everything being easily configurable. For example, the network, optimizer, and loss criterion can easily be swapped by changing command-line




**(a)** A screen capture of the GUI that was used to visualize training results. In clockwise from the top left, the GUI visualizes the ground truth data, a live plot of the loss and accuracy, various statistics, and the network output.




**(b)** A screen capture of the CLI that was used to train the network remotely.

**Figure 11:** The two available user interfaces available to use with CurbNet. The GUI was mean to more simply and quickly deliver the most important information first to the researcher. The CLI was designed to allow training status along with the most pertinent information to be shown remotely with little overhead.

arguments or parameters in the JSON file. This makes running different experiments straightforward.

## 5.7 Results

Running our network with the tuned hyperparameters chosen by the hyperparameter search in Section 5.5 for 30,336 iterations, we were able to achieve a mIoU accuracy of 50.188% on the validation dataset.

The network is able to produce the segmentations seen in Figure 12. We also notice that the network is able to achieve very high IoU accuracy for the road class, as seen in Table 8. This is due to the original purpose of DeepLab v3+ as a road segmentation network [1]. The IoU accuracy of curbs, curb cuts, and sidewalks are lower than of the road class. One possible reason for this is that IoU is especially hard to fulfill

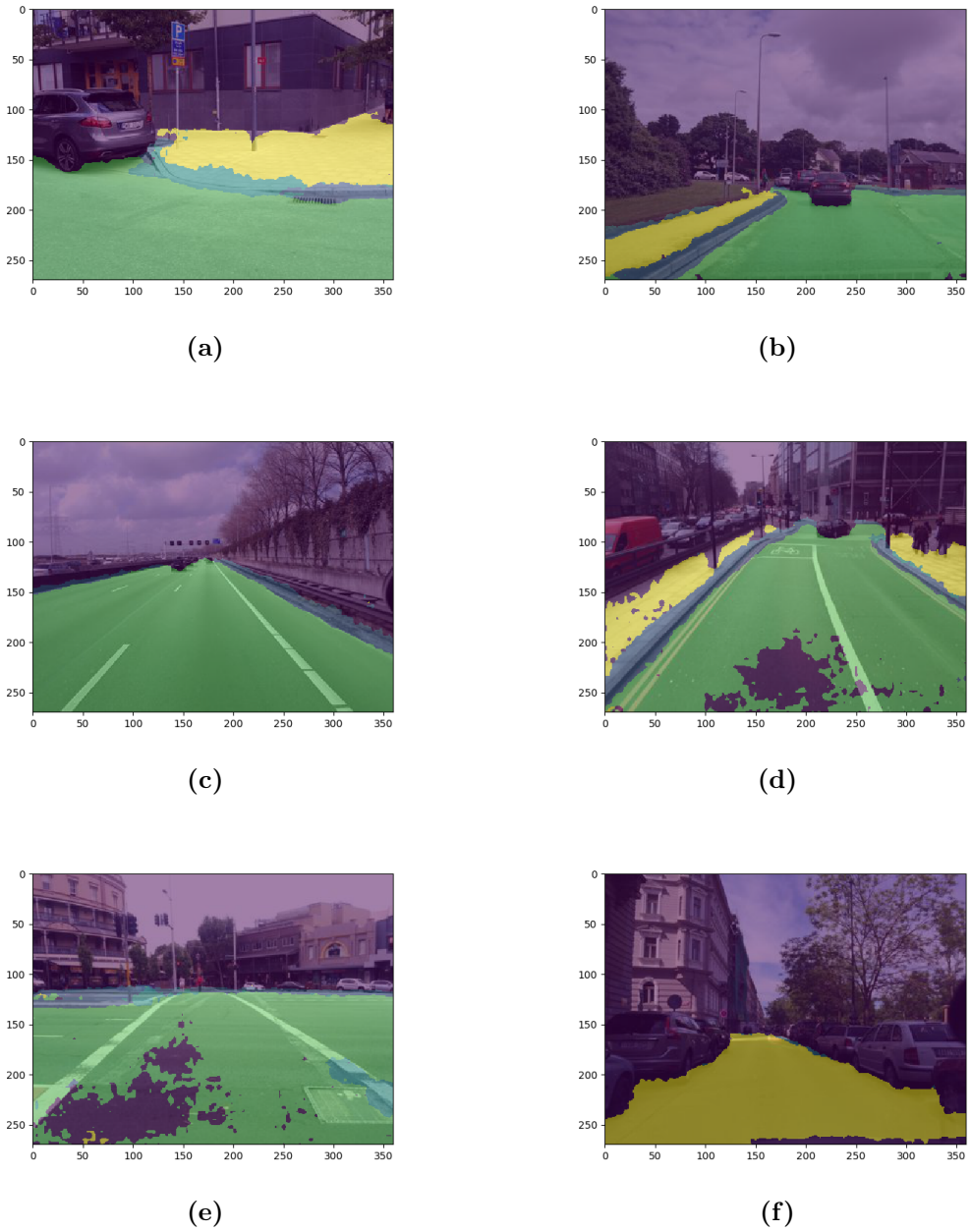
| | IoU | | | | | |
|---|---|---|---|---|---|---|
| Dataset | Curb | Curb Cut | Road | Sidewalk | Ignore | mIoU |
| Training | 24.219% | 28.064% | 83.372% | 31.523% | 95.673% | 52.570% |
| Validation | 22.987% | 25.292% | 79.942% | 28.587% | 94.131% | 50.188% |

**Table 8:** Results after training the network for 30,366 iterations on the Mapillary Vistas dataset [4] and evaluating it against both the training and validation datasets using IoU. The network achieves higher accuracy on the training dataset than on the validation dataset, but did not overfit on the training set.

when there are only a few pixels belonging to the class. Cases where the class has only a few pixels and the borders do not match exactly would return a low IoU.

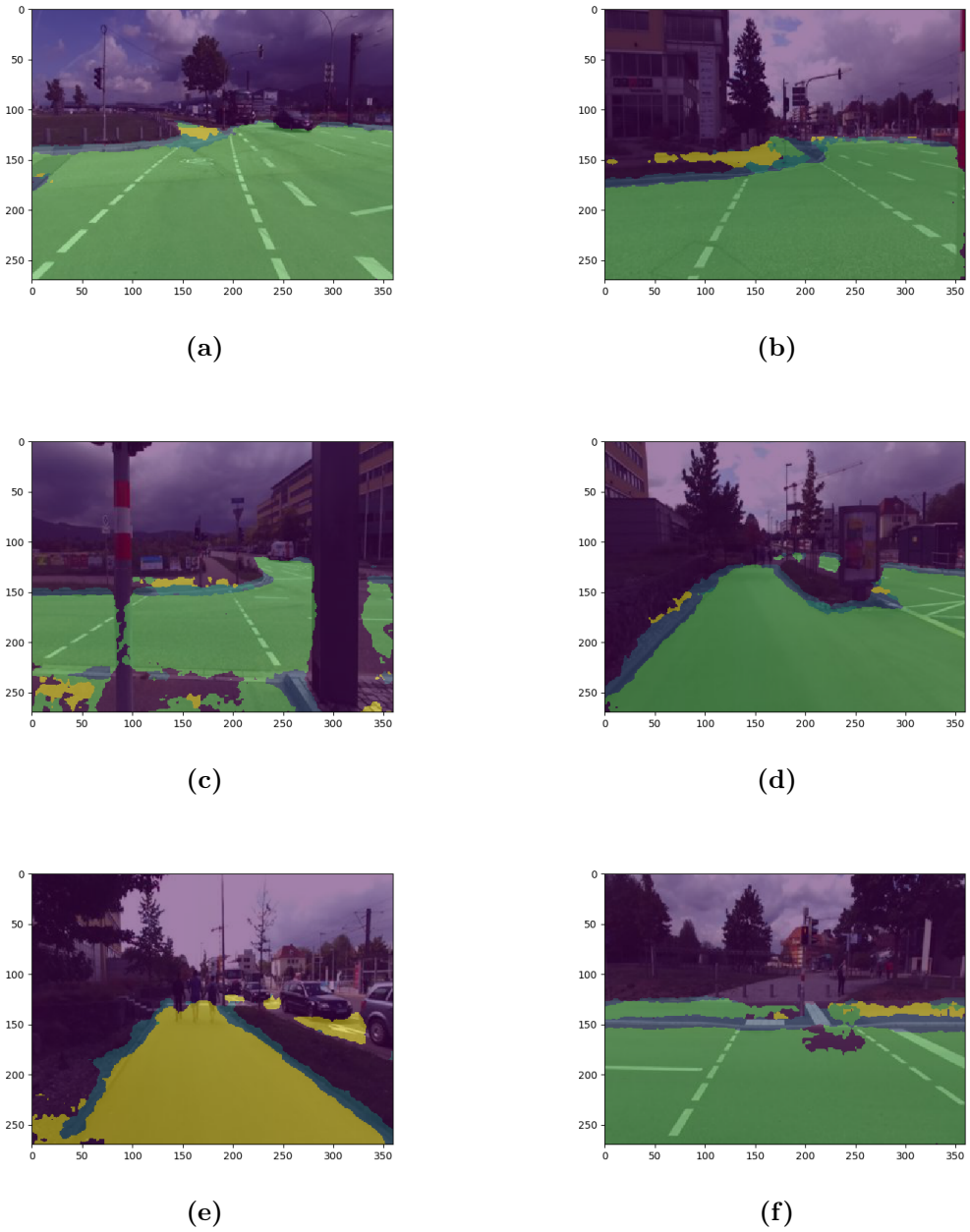
In these results we can observe that the network is quite capable of identifying each of the traversability classes. Figure 12a shows an example of how the network is able to identify each of the traversibility classes. The network can also identify traversability classes, specifically curbs and curb cuts, which are in the distance and thus have a relatively small size, as can be observed in Figures 12b - 12e. In these cases, the segmentation is significantly larger than the size of the curb itself, i.e. even though the intersection is significant, there is a significant area around the curb that is falsely identified as also being a part of the curb. Figure 12e shows a failure case where the road marking in the bottom right corner is mislabeled as a curb. Figures 12d and 12e also show failure cases where the road is labeled as the ignore class. Figure 12f shows another failure case where the entire road is mislabeled as sidewalk.

We also ran inference against a dataset gathered by the onboard cameras on the Obelix robot using the network. The results of this experiment can be seen in Figure 13. Figures 13a and 13b show that the network is able to generalize to the Obelix dataset and is able to identify the different traversability classes in the images, despite never having been trained on them. Figure 13c shows although curb cuts are correctly identified, as in the case of the curb cut across the street, they are also sometimes mislabeled, as in the case of the curb cut directly ahead of the robot. We can observe



**(a)** **(b)** **(c)** **(d)** **(e)** **(f)**

**Figure 12:** A sample of some segmentation results from the trained CurbNet model. The images are from the Mapillary Vistas dataset [4]. Green is road, yellow is sidewalk, blue is curb, turquoise is curb cuts, and purple is the ignore class. Figure 12a shows how the model is able to identify all of the different traversibility classes. Figures 12b and 12c shows that segmentation of curbs even in the distance can be performed well. Figures 12d and 12e shows that curb cuts in the distance are also identified and segmented by the network, although sometimes roads are mislabeled as ignore. Figure 12e also shows a failure case where some road markings in the bottom right corner have been mislabeled as curb. Figure 12f shows a case where the entire road is labeled as sidewalk.



**(a)** **(b)**

**(c)** **(d)**

**(e)** **(f)**

**Figure 13:** A sample of some segmentation results from the trained CurbNet model. The images are from the Obelix dataset. Green is road, yellow is sidewalk, blue is curb, turquoise is curb cuts, and purple is the ignore class. Figures 13a and 13b shows how the model is able to identify all of the different traversibility classes in the Obelix dataset even though it was only trained on the Mapillary dataset. Figure 13c shows curb cuts are sometimes also identified properly, as seen in the identification of the curb cut across the street, but not always, as seen in the curb cut directly in front of Obelix. Figures 13d and 13e shows a case where the network seems to be unable to decide if the sidewalk is a sidewalk or a road. In both cases, the sidewalk edge is marked as a curb, when it should not. Figure 13f shows a case where the left half of the sidewalk is labeled as road while the right half is labeled as sidewalk.

in Figures 13d and 13e that the network seems unable to decide whether the asphalt sidewalk is a road or a sidewalk. The same failure case can be seen in Figure 13f where the left side of the sidewalk is mislabeled as road while the right side is labeled correctly as sidewalk.

We observe that the main failure cases of the network are falsely identifying certain road markings as curbs and identifying certain roads as sidewalks or vice versa. Certain road markings which are located near road edges are sometimes segmented as curbs, as can be seen in the bottom right corner of Figure 12e. We believe that this is due to the texture of the marking being colored similarly to some curbs as well as its shape which does resemble a curb. In some cases, the sidewalk is identified as a road, as in Figure 13e or the entire road is identified as a sidewalk, as in Figure 12f. We believe that this may occur when sidewalks have an asphalt-like texture, as in the case in Figure 13e.

# 6 Conclusion and Future Work

In this work we proposed CurbNet, a deep learning model that is capable of segmenting road, sidewalk, curbs, and curb cuts in street-level imagery. CurbNet uses DeepLab v3+ to perform semantic segmentation on images and trains using a modified loss function known as masked cross entropy loss.

Training and validation was performed on the Mapillary dataset [4]. Hyperparameter optimization was performed using the bayesian optimization and hyperband framework.

To evaluate our model, we used mean Intersection over Union and used a validation subset of the Mapillary dataset. Our network was able of achieving 50.188% mIoU on this validation set. Our model is also capable of segmenting curbs and curb cuts when they are relatively small compared to the rest of the image, such as in the case of when they are located further away from the camera. In these cases, though, the network is prone to identifying areas around the curb as also being part of the curb itself. Failure cases include the network identifying certain road markings as curbs, identifying sidewalks that are made of an asphalt-like material as roads, and identifying roads as sidewalks.

We also show that our trained network can be used on the Obelix dataset and produce reasonable results, demonstrating that our model is generalizable to unseen data.

Our hyperparameter tuning was performed with a very limited budget. It is therefore reasonable to assume that given increased compute resources and time, we would be able to find better optimized hyperparameters for our network. Furthermore, our network has also not shown any signs of overfitting to the training set after training for 30,366 iterations. We therefore believe that given increased compute resources and time, we could further train the network and achieve better results with our network.

We propose that future work should be done with additional depth information and using LIDAR joint training. Including this information would reduce the number of ambiguous cases where a road marking is visually similar to a curb, thus increasing accuracy. The addition of depth information could also potentially increase the accuracy of curb cut detection, as visually, curbs and curb cuts tend to be quite similar.

Finally, we hope to fully integrate CurbNet into Obelix's local trajectory planner pipeline.

# 7 Appendix

## 7.1 Detailed Hyperparameter Optimization Results

| | Parameter | | | | |
|---|---|---|---|---|---|
| Run Number | Learning rate | Optimizer | Momentum | Epsilon | Loss criterion |
| 1 | $2.017 \times 10^{-5}$ | Adam | - | 0.8 | MCE |
| 2 | $3.567 \times 10^{-5}$ | Adam | - | 0.95 | MCE |
| 3 | $8.073 \times 10^{-5}$ | Adam | - | 0.4 | MCE |
| 4 | $2.116 \times 10^{-5}$ | SGD | 0.5 | - | MCE |

**Table 9:** Hyperparameter configurations used for each run performed. These were the values selected by the bayesian optimization and hyperband algorithm used to optimized the hyperparameters for the CurbNet model.

| | IoU | | | |
|---|---|---|---|---|
| Run Number | Curb | Curb Cut | Ignore | mIoU |
| 1 | 0.918% | 0.125% | 50.919% | 17.320% |
| 2 | 0.860% | 0.136% | 51.222% | 17.406% |
| 3 | 0.080% | 22.450% | 83.008% | 35.179% |
| 4 | 0.414% | 0.289% | 68.653% | 23.11% |

**Table 10:** Hyperparameter tuning results based on the configurations in Table 9 after performing bayesian optimization and hyperband tuning with a budget of 5-10 epochs for each configuration.

## 7.2 Dataset Statistics

| Statistic | Value |
|---|---|
| Total images | 18,000 |
| Maximum image dimensions | $(6528 \times 5248)$ |
| Minimum image dimensions | $(640 \times 480)$ |
| Number of different dimensions | 156 |
| Percent of images with curbs | 84.21667% |
| Percent of images with curb cuts | 44.79444% |
| Average proportion of image that are curbs | 0.81557% |
| Average proportion of images that contain curbs that are curbs | 0.96842% |
| Average proportion of image that are curb cuts | 0.08869% |
| Average proportion of images that contain curb cuts that are curb cuts | 0.19799% |
| Images with curb cuts but without curbs | 30 |
| Usable Images (contains both curbs and curb cuts) | 15160 |

**Table 11:** General dataset statistics

# Bibliography

[1] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[2] "The european pedestrian assistant," http://europa.informatik.uni-freiburg.de 2009.

[3] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pp. 1436–1445, July 2018.

[4] G. Neuhold, T. Ollmann, S. Rota Bulò, and P. Kontschieder, "The mapillary vistas dataset for semantic understanding of street scenes," in *International Conference on Computer Vision (ICCV)*, 2017.

[5] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard, "A navigation system for robots operating in crowded environments," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, May 2013.

[6] "Interview with Jannik Zürn." Interview, July 2019.

[7] R. Eckhorn, H. J. Reitboeck, M. Arndt, and P. Dicke, "Feature linking via stimulus - evoked oscillations: Experimental results from cat visual cortex and

functional implications from a network model," in *International 1989 Joint Conference on Neural Networks*, IEEE, 1989.

[8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, pp. 541–551, Dec. 1989.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[11] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[12] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[13] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan. 2015.

[14] 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 *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] J. Maye, R. Kaestner, and R. Siegwart, "Curb detection for a pedestrian robot in urban environments," in *Proceedings - IEEE International Conference on Robotics and Automation*, 05 2012.

[16] A. Abbott, A. Deshowitz, D. Murray, and E. C. Larson, "Walknet: A deep learning approach to improving sidewalk quality and accessibility," *SMU Data Science Review*, vol. 1, no. 1, 2018.

[17] G. Seif, "Semantic segmentation with deep learning," *Towards Data Science*, September 2018.

[18] K. Fukushima, "Neocognitron," *Scholarpedia*, vol. 2, no. 1, p. 1717, 2007. revision #91558.

[19] C. Gorney and D. Hall, "Curb cuts," *99 Percent Invisible*, May 2018.

[20] D. Sabinasz, "Deep learning from scratch IV: Gradient descent and backpropagation," *deep ideas*, 2017.

[21] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR 2015*, p. 13, International Conference on Learning Representations (ICLR), 2015.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, October 1986.

[23] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, P. Prabhat, and R. P. Adams, "Scalable bayesian optimization using deep neural networks," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 2171–2180, JMLR.org, 2015.

[24] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, pp. 1–52, 04 2018.

[25] W. Koehrsen, "A conceptual explanation of bayesian hyperparameter optimization for machine learning," *Towards Data Science*, June 2018.

[26] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, (USA), pp. 2546–2554, Curran Associates Inc., 2011.

[27] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (A. Gretton and C. C. Robert, eds.), vol. 51 of *Proceedings of Machine Learning Research*, (Cadiz, Spain), pp. 240–248, PMLR, 09–11 May 2016.

[28] J. Serra, *Image Analysis and Mathematical Morphology*. Orlando, FL, USA: Academic Press, Inc., 1983.

[29] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds.), (Cham), pp. 234–241, Springer International Publishing, 2015.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

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