Undergraduate Thesis

CurbNet: 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

July 22nd, 2019

Writing Period

20. 4. 2019 – 22. 7. 2019

Examiner

Prof. Dr. Wolfram Burgard

Second Examiner

?

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

(EXTEND: Write this.)

Acknowledgments

First and foremost, I would like to thank...

- advisors
- examiner
- person2 for the great suggestion
- proofreaders

Contents

List of Figures

List of Tables

1 Introduction

Semantic scene segmentation is a popular research topic in the field of computer vision, and especially important for autonomous vehicles. The ability to semantically understand a scene is especially important for autonomous vehicles and robots to safely navigate an environment. Generally, most implementations attempt to segment road surfaces but in this thesis, we propose the segmentation of curbs and curb cuts to allow safer sidewalk navigation.

The Europa project has resulted in the Obelix robotic platform, which has already been demoed to successfully perform pedestrian navigation [1][2]. We propose to add to this platform the ability to detect curbs and curb cuts using semantic segmentation. The Obelix platform is the result of a joint project to build a robotic platform capable of robotic navigation [1]. **(TODO: Add picture of Obelix)**. 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 [3]. To extend its pathfinding ability, a project is currently underway to use the stereo cameras to classify surfaces that Obelix will encounter [3]. This work specifically aims to add to said project by additionally identifying curbs and curb cuts, providing constraints for the route planner. This will allow the route planner to identify curbs, which it cannot traverse, and curb cuts, which it can.

Our goal is thus to implement a computer vision algorithm capable of the semantic segmentation of curbs and curb cuts using a single camera image. To do so, we

implement 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 [4]. We additionally include prior knowledge to the training, as it can be assumed that the camera setup and attitude for Obelix will remain relatively constant throughout its lifespan.

We will begin by discussing the motivation behind this thesis, followed by a discussion of related works and the background. The approach will then be discussed in detail along with the experiments and results. Finally, a discussion of potential future research will be presented followed by the conclusion.

2 Related Work

CurbNet uses semantic scene segmentation to identify curbs and curb cuts. 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 Semantic Scene Segmentation

There are many works in the field of semantic scene segmentation in recent years, both discussing object scene segmentation and road segmentation. 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 [5]. This early method used a pulse coupled neural network, which produced 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 [6]. Their paper titled "Backpropagation Applied to Handwritten Zip Code Recognition" proposed that using convolutional filters directly on an image input and training using back-propagation could reliably classify images into predetermined classes. This was the first network architecture that took a normalized image as input and returned the

image class directly as an output. LeCun et al. also showed that using backpropagation to learn the convolutional filter coefficients performed significantly better than 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" by Karen Simonyan and Andrew Zisserman was one of the earlier works to show that a deeper network setup could result in very high accuracy image classification [7]. Their setup improved upon the use of convolutional neural networks by proposing instead to use small (3×3) convolutional filters and changing the depth to 16-19 layers. This network is now commonly known as VGG16 - VGG19, the number representing layer depth. This simple change in architectural designed resulted in their architecture securing first and second place in the ImageNet Challenge 2014 localization and classification tracks respectively. Unfortunately, these very deep networks tended to overfit on smaller datasets and were difficult to train.

"Deep Residual Learning for Image Recognition" by Kaiming He et al. proposed a solution to this problem [8]. 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 were 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 [9].

"Fully Convolutional Networks for Semantic Segmentation" by Jonathan Long et al. took these classification networks and added fully connected layers to take the encoded features and use it for semantic segmentation [10]. This paper laid 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 proposed 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 and indeed, they implemented their model based on VGG16. Their network based on VGG16 was able to outperform competing state-of-the-art approaches on the PASCAL Visual Object Challenge dataset by a relative margin of 20%.

DeepLab v3+ was proposed in "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation" by Liang-Chieh Chen et al. and improves on FCN by refining the segmentation, especially along object boundaries [4]. DeepLab v3+ became the basis of the network we used for CurbNet. Using pyramid pooling and an improved encoder-decoder architecture, DeepLab v3+ achieve 89.0% accuracy on the Cityscapes dataset [11].

2.2 Curb Detection

Research into curb detection is quite plentiful, with even one work from ETH Zürich being made specifically for the Obelix robotic platform [12]. Curb Detection for a Pedestrian Robot in Urban Environments" by Jérôme Maye, Ralf Kaestner, and Roland Siegwart used the LIDAR sensors that Obelix has to map the world around it. 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. Unfortunately, this work does not take into account curb cuts, which may not necessarily form a significant vertical height difference and are usually sloped.

The only work we were able to find that specifically address curb cuts was "WalkNet: A Deep Learning Approach to Improving Sidewalk Quality and Accessibility" by Andrew Abbott et al. [13]. This paper discusses the use of a deep neural network to classify images in which curb cuts existed. Their goal was the use of Google Street

View data to map which intersections in a city already had curb cuts and which didn't. This data would then be supplied to city governments to provide relevant information regarding sidewalk accessibility and quality. Unfortunately, this work only classified images which contained curb cuts and the neural network architecture they used was not described in depth. As such, we were unable to use any part of their research, despite being one of the few papers published regarding curb cuts.

Thus far, there seems to be no works related directly to the goal of this thesis; the semantic segmentation of curbs and curb cuts using computer vision.

3 Background

The basis of CurbNet is semantic scene segmentation using convolutional neural networks. As such, it is imperative that the reader has an understanding of the principles behind semantic scene segmentation and convolutional neural networks. This chapter on the background of this work will discuss semantic scene segmentation, machine learning with artificial neural networks and convolutional neural networks, curb segmentation, loss functions, and network optimizers.

3.1 Semantic Scene Segmentation

Unlike image classification, which predicts what an entire scene is, semantic scene segmentation is the processing of an image and assigning class labels to each individual pixel [14]. This allows for finer details and adds the ability to locate and classify multiple objects in a scene. For example, the image in figure **(TODO: add image)** has been segmented and the result is figure **(TODO: add segmented)**. Each pixel of the image has been assigned an associated class, in this case, **(TODO: add classes segmented)** 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 or its environment semantically similarly to the way a human might. 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 of images in this way is essential in many robotic implementations as it allows further higher level processing of the scene.

In our case, being able to segment and identify curbs in a scene would allow Obelix to map the location of the surrounding curbs and curb cuts, allowing for the safe traversal from sidewalk to street level via curb cuts.

Over the past few years, state-of-the-art methods in image segmentation have relied entirely on deep learning using CNNs to achieve better results.

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 time dependent function. The function of a neuron j receiving input $p_j(t)$ and producing output $o_j(t)$ is composed of the activation $a_j(t)$, potentially a threshold Θ_j, an activation function f that returns the activation at time $t+1$, and an output function f_{out}. The activation $a_j(t)$ can also be considered the neuron's state and is dependent on the time parameter t. The threshold Θ_j, if it exists, is fixed unless a learning function changes it. The activation function f calculates $a_j(t+1)$ given $a_j(t)$, Θ_j, and the network input $p_j(t)$ and can be defined as:

$$a_j(t+1) = f\left(a_j(t), \Theta_j, p_j(t)\right) \tag{1}$$

The output function f_out computes $o_j(t)$ based on $a_j(t)$ and is defined as:

$$o_j(t) = f_out\left(a_j(t)\right) \tag{2}$$

The output function is usually simply the identify function $f(x) = x$. Many activation functions exists and are used including the identity function and the rectified linear unit (ReLU), defined as:

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

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 and potentially a bias term and is computed by the propagation function to provide a neuron its input $p_j(t)$. This typically is defined as:

$$p_j(t) = \sum_i o_i(t) w_{ij} + w_{0j} \ , \text{ where } w_{0j} \text{ is the bias, if it exists.} \tag{4}$$

Learning occurs by using an algorithm to modify the parameters of the neural network.

Deep neural networks are so called due to having multiple "hidden" layers between the input and output. These hidden layers are not visible to the end user and their values are typically never 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 for image analysis, such as image classification and segmentation. The inspiration for CNNs come from biological processes to simulate the orgainization of the visual cortex in animals [15]. Convolutional layers are the core build blocks of CNNs. These convolutional layers consist of a set of learnable filters which are convolved across the entire input and computing

the dot products between the different entries of the filter. This allows the layer to activate when it detects a specific type of feature at some point in the input. **(TODO: convolution image)** By using multiple convolutional layers, higher level features can be extracted and a feature map generated.

For semantic scene segmentation applications, the convolutional layers that make up the CNN are called the feature encoder. The output of these convolutional layers are then fed into one or more fully connected linear layers to produce a classification of each pixel in a secene. These fully connected layers are called the decoder. The resulting classification produced is a probability of each pixel being a certain class.

3.3 Curbs and Curb Cuts

Curbs are the edges of roads, or the separator between the road and some other area, usually a pedestrian sidewalk, and are usually stone or concrete. Curb cuts are "cuts" in the curb that allow a pedestrian sidewalk to have a gentle slope down to street level. Originally, these cuts were made to allow accessibility access, especially for those requiring wheelchairs, as to a wheelchair user, a curb is as good as a wall in terms of traversibility, as was discussed in the article "Curb Cuts" by Cynthia Gorney and Delaney Hall. These curb cuts first started appearing fifty years ago from the efforts of activist Ed Roberts and his push to make city streets more accessible. **(TODO: Include picture of a curb cut)**

3.4 Curb Segmentation

A discussion of the state-of-the-art of curb and curb cut segmentation would be moot, as there is no work we could find specifically discussing this topic.

There are quite a few papers regarding the identification and localization of curbs using LIDAR sensors, but none that specifically tackle the problem of curb cuts and none that use computer vision approaches. The challenge for curb segmentation is the relatively small size of curbs in relation to everything else in the scene. As such, there is a heavy class imbalance. With such an unbalanced dataset, it is possible for a neural network to simply optimize towards classifying no curbs at all, since this would produce a reasonably low loss.

Furthermore, the segmentation and differentiation between curbs and curb cuts using a single image with no depth information is quite difficult, as visually, curbs and curb cuts can seem very similar.

3.5 Loss Functions

The loss function l maps the output of the 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, known as the network loss. 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 predicted labeling 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.

The weighted variant, known simply as weighted cross entropy loss, is the basis of the custom loss function we use for CurbNet and is defined as:

$$\text{loss}(x, class) = weight[class] \left(-\log \left(\frac{\exp(x[class])}{\sum_j \exp(x[j])} \right) \right) \tag{5}$$

where x is the predicted labeling, $class$ is the ground truth labeling, $weight$ is the weight given to each class, and j is the individual pixels in x.

The use of cross entropy loss is to ensure that the cost increases exponentially as the predicted probability approaches zero. This due to its use of a negative log function, whose value exponentially increases as it approaches zero. Using the weighted version allows us to change how much each class contributes to the loss. We do this to counteract the class imbalance caused by how small spatially the curbs are relative to the rest of the scene.

3.6 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 [16].

Gradient Descent is one of the oldest loss functions used. It works by calculating what a small change in each network parameter would do to the loss function, i.e. determines the optimal gradient 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 [16].

Adaptive Moment Estimation (Adam) is another loss function that is commonly used and the one implemented in CurbNet. 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 [17]. 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. The original paper that proposed

Adam, titled "Adam: A method for stochastic optimization" by Diederik Kingma and Jimmy Ba, also shows that Adam allows a network to converge faster than competing optimizers, including SGD [17].

3.7 Back-propagation

Back-propagation is a learning procedure used in artificial neural networks to adjust network weights and produce internal representations "important features in the task domain", first described the David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams in their paper "Learning representations by back-propagating errors" [18]. Back-propagation 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 effect the output unit has on the loss is calculated first as

$$\frac{\partial \text{loss}}{\partial y_j} \tag{6}$$

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 (6)

$$\frac{\partial \text{loss}}{\partial x_j} = \frac{\partial \text{loss}}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} \tag{7}$$

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

$$\frac{\partial \text{loss}}{\partial w_{ji}} = \frac{\partial \text{loss}}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ji}} \tag{8}$$

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 (7) 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 (7).

$$\frac{\partial \text{loss}}{\partial y_i} = \frac{\partial \text{loss}}{\partial x_j} \cdot w_{ji} \tag{9}$$

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.

$$\frac{\partial \text{loss}}{\partial y_i} = \sum_{j \in J} \frac{\partial \text{loss}}{\partial x_j} \cdot w_{ji} \tag{10}$$

This calculation can be done in parallel, thanks to parallel computing architecture, completing to 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.

The main drawback of back-propagation is that it may approach local minima which are not the global minimum, at which point the network can potentially get stuck and become unable to improve. The addition of more units, thus increasing the weight dimensionality, can increase the number of paths out of local minima. Futhermore, the authors of the previously mentioned paper have also concluded that this is not a plausible model for how learning works in biological brains and, thus, that looking into other biologically based procedures may be useful.

3.8 Hyperparameter Tuning

Hyperparameter tuning is the process of optimizing a model to find the hyperparameters that will perform the best according to a specified metric. This metric is usually either the validation loss or the validation accuracy. Hyperparameters are generally chosen by the researcher or using some other automated process, but is 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 on all but the smallest of networks, as they potentially run less than optimal parameters and disregard previous runs. 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 [19]. Hyperband is a bandit-based approach to hyperparameter optimizations that can abandon or reduce the budget of an experimental run if it deems the results to be significantly poor [20].

3.8.1 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 actual model the objective function can then be used to calculate the actual scores. The probabilistic distribution is the fine tuned and the process is repeated until the maximum number of iterations is reached.

Bayesian optimization relies on the following [21]:

1. A clearly defined hyperparameter domain of possible configurations χ over which to search,
2. An objective function which calculates a score using a set of hyperparameters that we wish to minimize,
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 regards to previous knowledge or based on related works. The objective function is calculated by running the actual 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) [22]. TPE takes advantage of Bayes' rule and represents the probability function $\mathrm{P}(y \mid x)$ as

$$\mathrm{P}(y \mid x) = \frac{\mathrm{P}(x \mid y) \cdot \mathrm{P}(x)}{\mathrm{P}(y)} \tag{11}$$

$$\text{with } \mathrm{P}(x \mid y) = \begin{cases} l(x) & \text{when } y < y^* \\ g(x) & \text{when } y \geq y^* \end{cases} \tag{12}$$

where y^* is a threshold value. $l(x)$ is thus a probability density function formed from the set of observations $x^i \in \chi$ which have been performed such that the loss of the network running with hyperparameters x^i 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 γ 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 expected improvement [21]

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y \mid x)dy \tag{13}$$

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

$$EI_{y^*}(x) = \frac{\gamma y^* l(x) - l(x) \int_{-\infty}^{y^*} p(y)dy}{\gamma l(x) + (1-\gamma)g(x)} \tag{14}$$

$$\propto \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1} \tag{15}$$

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.

3.8.2 Hyperband

Hyperband is a learning strategy that "relies on a principled early-stopping strategy to allocate resources" [20]. It is a variation of Successive Halving, and in fact uses successive halving in its inner loop.

Successive halving randomly samples n hyperparameter sets in the search domain χ. 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 1 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 [20].

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, η (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

3.8.3 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 [23]. 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 [23].

Algorithm 2: BOHB algorithm for hyperparameter optimization

Input : observations D, fraction of random runs ρ, 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 (11)
draw N_s samples according to $l'(x)$
return *sample with highest ratio* $l(x)/g(x)$

3.9 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:

$$A \oplus B = \bigcup_{b \in B} A_b \tag{16}$$

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 [24].

4 Approach

To semantically segment urban scenes and extract predictions for curbs and curb cuts, 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. [25]. In this chapter, the architecture used and an explanation of our loss function will be discussed.

4.1 Architecture Selection

Relative to the entire image, curb and curb cut classes make up a relatively small proportion of the image, making up 0.986% and 0.196% of images on average respectively. As such, four networks with good known performance for urban scene segmentation were chosen and evaluated for their performance classifying curbs and curb cuts. We experimented with ENet, described in the paper "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation" [26]; GoogLeNet ,described in the paper "Going Deeper with Convolutions" [27]; FCN16s, described in "Fully Convolutional Networks for Semantic Segmentation" [10]; and DeepLab v3+, described in the paper "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation" [4]. 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 converge the fastest given a certain budget. This was done to compare how each network would perform classifying curbs and curb cuts.

We came to use the DeepLab v3+ architecture after running these experiments due to significantly outperforming the other networks, as seen in **(TODO: make table)**. A further discussion of the results can be found in section 5.4.

4.1.1 Modifications to DeepLab v3+

Given that we are working with the assumption of a camera with a constant attitude, we can also make the assumption that curbs will be more likely towards the bottom and sides of an image. Therefore, we modified DeepLab v3+ by adding a preprocessing step which turns the input image into a five-dimensional tensor, with the fourth and fith dimensions being the x and y coordinates of the pixel, respectively. The initial convolutional layer was also modified to accept five channels as input. **(EXTEND: add image of example tensor input showing what each channel is)**

4.1.2 Network Architecture

We propose to apply the DeepLab v3+ architecture to our stated problem of curb and curb cut classification with the modifications described in subsection 4.1.1. The architecture itself is given in Figures **(TODO: ref figures)** with further descriptions in Tables 1-4[1].

4.2 Loss Function

Initially, the network was trained on standard cross entropy loss, but due to the severe class imbalance, this did not produce meaningful results with the network simply labeling everything as the 0th class, i.e. not curb or curb cut. We then used a weighted cross entropy loss function. The weights were chosen by taking the

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

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 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 Dropout	in = 1280, out = 256 $p = 0.5$

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 Dropout	in = 304, out = 256, kernel = 3, stride = 1 $p = 0.5$
2	Convolution + ReLU Dropout	in = 256, out = 256, kernel = 3, stride = 1 $p = 0.5$
3	Convolution	in = 256, out = 3, kernel = 1, stride = 1

Table 4: Decoder Module Details

normalized inverse of the average proportion other, curb, and curb cut classes in the dataset. This provided better results, but was still not producing the results that we wanted **(TODO: make this sound more formal and include image)**.

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 can be seen in Equation (19) [25]. 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 $0.03 \times \text{image}_{width}$ on the road class, then subtracted by the road class itself. Thus, this can be formalized as follows:

$$B = \underbrace{\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}}_{b \text{ columns and } b \text{ rows}} \tag{17}$$

$$M = (\text{class}_{\text{road}} \oplus B) - \text{class}_{\text{road}} \tag{18}$$

The value 0.03 was chosen by the researcher 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 are 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 can be seen in the following formalization of the loss function we used:

$$\text{loss}_{MCE}(x, class) = weight[class] \left(-\log \left(\frac{\exp(x[class])}{\sum_j \exp(x[j])} \right) \right) \tag{19}$$

$$\text{with } weight[class] = \begin{cases} weight'[class] & \text{if} x \in M \\ weight'[class] \cdot 3 & \text{if} x \notin M \end{cases} \tag{20}$$

where $weight'[class]$ 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 **(TODO: image of mask)**.

5 Experiments

A series of experiments were carried out to determine the performance of the network in its stated goal of classifying curbs and curb cuts. To this end, the network was tested on the Mapillary Vistas dataset [28] and its hyperparameters optimized using the Bayesian optimization with hyperband (BOHB) procedure from the AutoML group at University of Freiburg [23]

5.1 Dataset

The Mapillary Vistas dataset was chosen for this task as it was one of the few street-level dataset that included ground truth segmentations of both curbs and curb cuts. 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.

With such a diverse dataset, 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.

5.2 Hyperparameters

Hyperparameter tuning was done by using BOHB. **(EXTEND: include graph and discussion)**

5.3 Training Pipeline

5.4 Evaluation Method

5.5 Results

6 Future Research

7 Conclusion

ToDo Counters

To Dos: 10; 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Parts to extend: 3; 1, 2, 3

Draft parts: 0;

Bibliography

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

[2] 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.

[3] Y. P. Satyawan, "Interview with jannik zürn." Interview., July 2019.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

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

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

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

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

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

[25] 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.

[26] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," June 2016.

[27] 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.

[28] 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.

(TODO: replace all arXiv links with journal links, if possible)