

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

September 2nd, 2019

Writing Period

20. 4. 2019 – 2. 9. 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

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 purposes of curb and curb cut 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. By adding this capability to Obelix, we can further its pathfinding capabilities [2]. In this work, we show that CurbNet is a capable model, 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...

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

Contents

Acknowledgments	v
1 Introduction	1
2 Related Work	3
2.1 Image classification	3
2.2 Semantic Segmentation	4
2.3 Curb Detection	5
3 Background	7
3.1 Semantic Segmentation	7
3.2 Artificial Neural Networks	8
3.2.1 Convolutional Neural Networks	10
3.3 Curbs and Curb Cuts	11
3.4 Loss Functions	12
3.4.1 Cross Entropy Loss	12
3.5 Optimizers	14
3.6 Backpropagation	15
3.7 Hyperparameter Tuning	17
3.7.1 Bayesian Optimization	17
3.7.2 Hyperband	19
3.7.3 Bayesian Optimization with Hyperband	20
3.8 Binary Dilation	21

Contents

4 Approach	23
4.1 Architecture Selection	23
4.1.1 Network Architecture	23
4.2 Loss Function	26
5 Experiments	29
5.1 Dataset	29
5.2 Network Evaluation	30
5.3 Modifications to DeepLab v3+	31
5.4 Hyperparameters	32
5.5 Training Pipeline	34
5.6 Evaluation Metrics	35
5.7 Results	35
6 Conclusion and Future Work	37
7 Appendix	39
7.1 Detailed Hyperparameter Optimization Results	39
7.2 Dataset Statistics	39
Bibliography	46

List of Figures

1	The Obelix Robot	2
2	An example of semantic segmentation.	8
3	Artificial Neural Networks	9
4	Convolution operation	11
5	Curbs and curb cuts	12
6	Simplified Network Architecture	24
7	Masked Cross Entropy Mask	27
8	Network Comparison Chart	31
9	Training UI	35

List of Tables

1	DRN Module Details	24
2	Bottleneck Module Details	25
3	Atrous Spatial Pyramid Pooling (ASPP) Module Details	25
4	Decoder Module Details	25
5	Network Comparison Results	31
6	Hyperparameter Details	33
7	Hyperparameter Optimization Results	34

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 field of view, which is important for autonomous vehicle 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 in urban environments [2][5]. 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 [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 pathfinding 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 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.

The aim of this work is thus to implement a deep learning model capable of the

1 Introduction



Figure 1: The Obelix Robot.

semantic segmentation of curbs and curb cuts using a single camera image. To this end, we propose the implementation 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 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 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

2 Related Work

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 and published in 2014 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 (3×3) 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" by Kaiming He et al. and published in 2016 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" by Jonathan Long et al. and published in 2015 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

2.3 Curb Detection

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" 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 the LIDAR sensors that Obelix has 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" by Andrew Abbott et al. and published in 2018 specifically addresses the identification of curb cuts [16]. This paper proposes the use of a deep neural network

2 Related Work

to classify images in which curb cuts existed. 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, 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 of images in this way is essential in many robotic applications as it allows further higher level processing of the scene.

3 Background



(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].

Towards the goal of this work, the segmentation of curbs and curb cuts 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 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:

$$a_j = f_a(x_j) \quad (1)$$

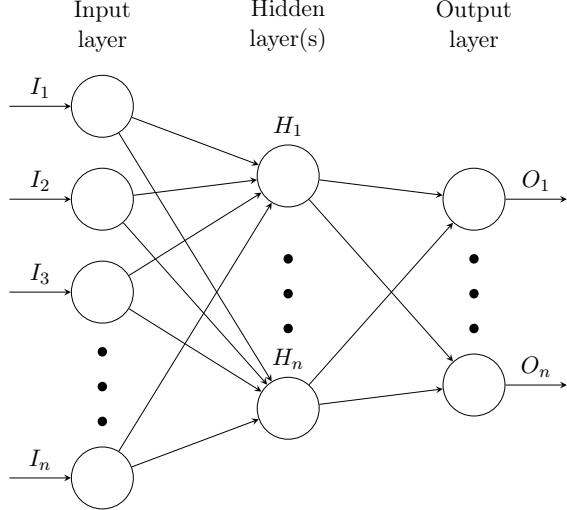


Figure 3: A Visualization of how neurons interact within an artificial neural network.

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

$$x_j = f_o(a_j) \quad (2)$$

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

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

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:

$$x_j = \sum_i y_i w_{ij} + w_{0j} \quad (4)$$

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

3 Background

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 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 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.

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

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.

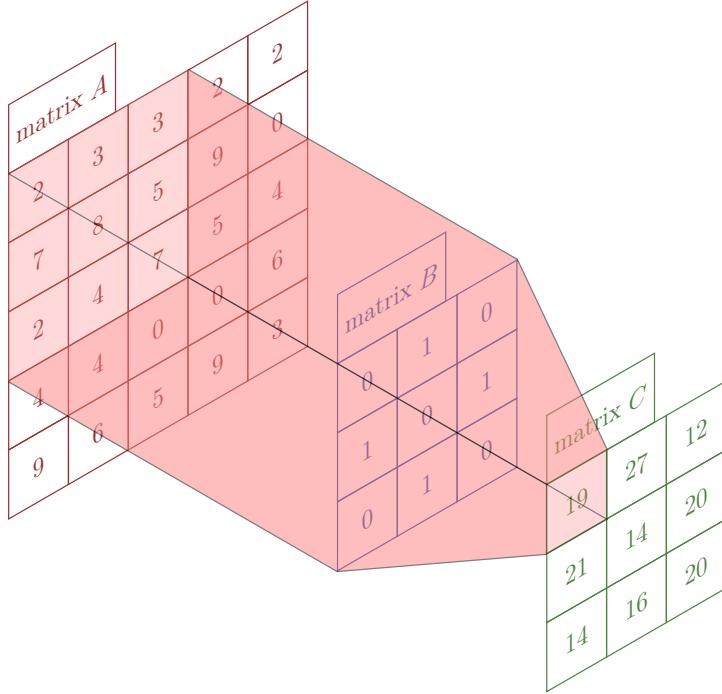


Figure 4: An example of how a convolution on matrix A and matrix B would result in matrix C .

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, as 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]. 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.

3 Background



Figure 5: An example of a curb and a curb cut.

3.4 Loss Functions

The loss function ℓ 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:

$$\ell(y, \hat{y}) = \sum_m -y_m \log(\hat{y}_m) \quad (5)$$

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

3.4 Loss Functions

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:

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

3 Background

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 "important features in the task domain", first described 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 ℓ with respect to y_j , intuitively the effect the output unit has on the loss, is defined as:

$$\frac{\partial \ell}{\partial y_j} \quad (10)$$

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

$$\frac{\partial \ell}{\partial x_j} = \frac{\partial \ell}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} \quad (11)$$

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

3 Background

between the unit j and a unit in the previous layer i will effect the error as

$$\frac{\partial \ell}{\partial w_{ji}} = \frac{\partial \ell}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ji}} \quad (12)$$

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).

$$\frac{\partial \ell}{\partial y_i} = \frac{\partial \ell}{\partial x_j} \cdot w_{ji} \quad (13)$$

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.

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

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.

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 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, 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 $P(y|x)$ 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.

3 Background

Bayesian optimization relies on the following [25]:

1. A clearly defined hyperparameter domain of possible configurations χ 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 evaluate 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 $P(y | x)$ as

$$P(y | x) = \frac{P(x | y) \cdot P(x)}{P(y)}$$

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

(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 $P(y < y^*) = \gamma$ 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 the expected improvement [25]:

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

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 $p(y | x)$ is the surrogate probability model expressing the probability of y given x . Substituting the values of $p(y | x)$ from the surrogate function we get

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

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 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 one hyperparameter set remains.

3 Background

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  $s \in s_{max}, s_{max} - 1, \dots, 0$  do
     $n = \left\lceil \frac{B}{R(s+1)} \right\rceil$ 
     $r = R\eta^{-s}$ 
     $T = \text{get\_hyperparameter\_configuration}(n)$ 
    for  $i \in \{0, \dots, s\}$  do
         $n_i = \lfloor n\eta^{-i} \rfloor$ 
         $r_i = r\eta^i$ 
         $L = \{\text{run\_then\_return\_validation\_loss}(t, r_i) : t \in T\}$ 
         $T = \text{top\_k}(T, L, \lfloor n_i/\eta \rfloor)$ 
    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 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  $P(x | y)$  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.

3 Background

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:

$$A \oplus B = \bigcup_{b \in B} A_b \quad (18)$$

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 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. [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 due to significantly outperforming the other networks. A further discussion of the results can be found in section 5.2.

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 1-4¹.

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

4 Approach

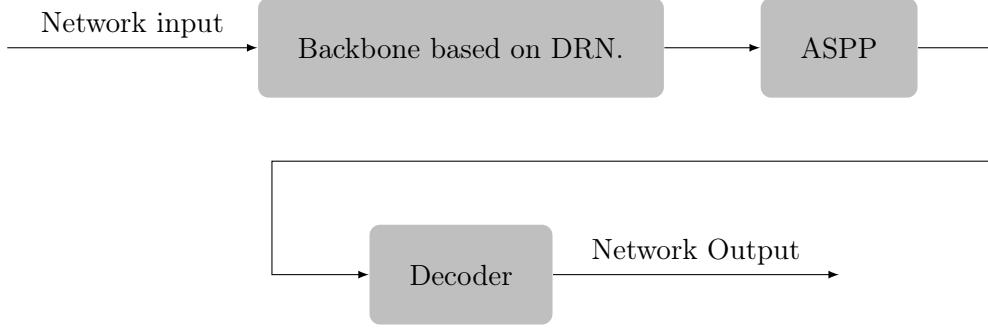


Figure 6: A simplified visualization of the network architecture. The contents of each module is listed in Tables 1-4.

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
	Bottleneck	in = 32, low = 64, out = 256
3	Bottleneck	in = 256, low = 64, out = 256
	Bottleneck	in = 256, low = 64, out = 256
	Bottleneck	in = 256, low = 128, out = 512
4	Bottleneck	in = 512, 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
	Bottleneck	in = 512, low = 256, out = 1024
5	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
	Bottleneck	in = 1024, low = 256, out = 1024
	Bottleneck	in = 1024, low = 512, out = 2048
6	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 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

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 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 $0.05 \times \text{image}_{\text{width}}$ on the road class, then subtracted by the road class itself. Thus, this can be formalized as follows:

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

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

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 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.



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]

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

$$\ell_{MCE}(y, \hat{y}) = \sum_m -y_m \log(\hat{y}_m) \cdot d_m \quad (21)$$

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

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 curbs and curb cuts. 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 from the AutoML group at University of Freiburg [3]

5.1 Dataset

The Mapillary Vistas dataset was chosen for this task as it was, to our knowledge, the only 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.

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

5 Experiments

360×320 pixels. This allows a batch size of 16 per GPU, given that the GPU has 12 GB of VRAM. 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, reducing the number of file accesses required and reducing training wall clock time by a factor of two.

An statistical analysis of the dataset with respect to their curb and curb cut content can be found in appendix section 7.2.

5.2 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 0.986% and 0.196% of images on average respectively. As such, three networks with good known performance for urban scene segmentation were chosen and evaluated for their performance classifying curbs and curb cuts. 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 converge the fastest given a certain budget. For each of these runs, the default recommended hyperparameters were used due to time constraints to conduct hyper-parameter

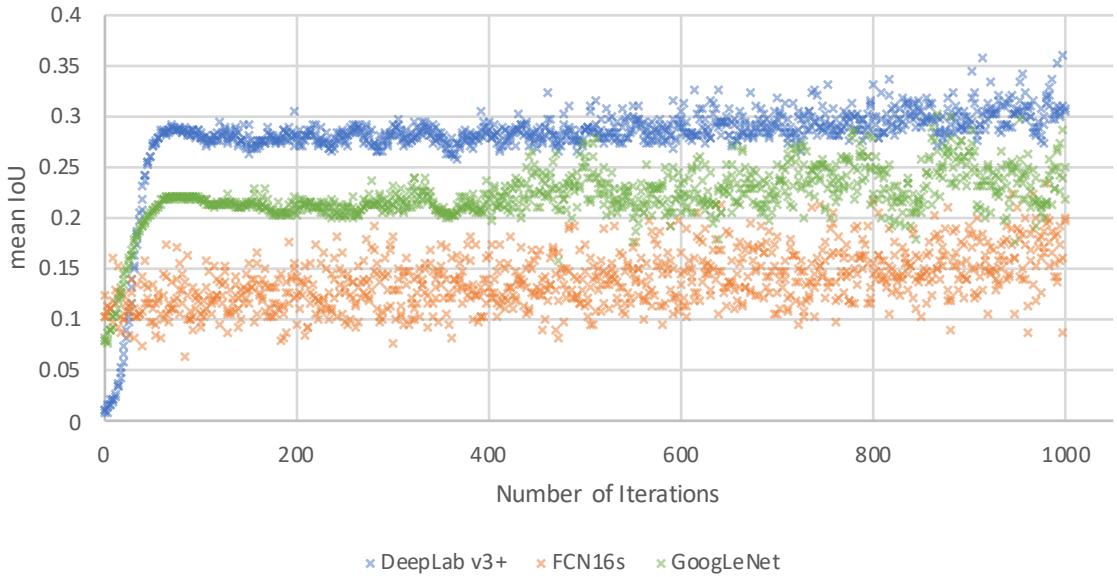


Figure 8: Training mean IoU Accuracy over 1000 Iterations using three different networks

optimization with every network. The results can be seen in Figure 8 and in Table 5. Given these results, we chose to use DeepLab v3+ as our network.

5.3 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, modifications were made to DeepLab v3+ by

Network	mean IoU
DeepLab v3+	30.8%
FCN16s	19.9%
GoogLeNet	25.1%

Table 5: Results of running the different networks on a subset of the main dataset after one thousand iterations.

5 Experiments

adding a preprocessing step which turns the input image into a five-dimensional tensor, with the fourth and fifth dimensions being the x and y coordinates of the pixel, respectively. The initial convolutional layer was also modified to accept five channels as input. Both versions of the network were run with identical hyperparameters on a limited budget and dataset. While there was an increase in mean IoU accuracy when using the five-dimensional input tensor, it was negligible and we decided to use the three-dimensional input tensor, given the increased training time with a five-dimensional input tensor.

(TODO: include 5D vs 3D results)

5.4 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; the weight ratio for the loss weights; and the loss criterion.

The learning rate is optimized between the interval $[1 \times 10^{-5}, 1 \times 10^{-2}]$, 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 $[1 \times 10^{-2}, 1]$. The weight ratio for the loss weight determines the ratio of the weights between the curb and curb cut. The value of the weight for the "other" class is one-tenth of the sum of curb and curb cut weights, formally defined as:

$$\begin{aligned} \frac{x}{y} &= r, \quad r \in [2, 6] \\ z &= \frac{x+y}{10} \end{aligned} \tag{23}$$

Parameter Name	Parameter Type	Range	Comments
Learning rate	Float	$[1 \times 10^{-5}, 1 \times 10^{-2}]$	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	$[1 \times 10^{-2}, 1]$	Only active when the optimizer is Adam
Loss weight ratio	Float	$[2, 6]$	
Loss criterion	Categorical	Weighted cross entropy or masked cross entropy	

Table 6: Table of hyperparameters and their values that were optimized for using Bayesian Optimization and HyperBand.

where x is the weight for the curb class, y the weight for the curb cut class, z the weight for the "other" class, and r the ratio value that is optimized. 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.

(**EXTEND: include graph**) The hyperparameters are optimized according to the above parameters using a cluster of (**TODO: number of**) GPUs over a budget of 12 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.

Given increased compute resources and time, we believe that we could further optimize the hyperparameters and achieve better results with our network.

5 Experiments

Parameter Name	Value
Learning rate	0.0002
Optimizer	Adam
SGD momentum	-
Adam epsilon	0.1
Loss weight ratio	4
Loss criterion	Masked cross entropy loss

Table 7: Table of hyperparameters and their values that were optimized for using Bayesian Optimization and HyperBand. (**TODO: complete table values**)

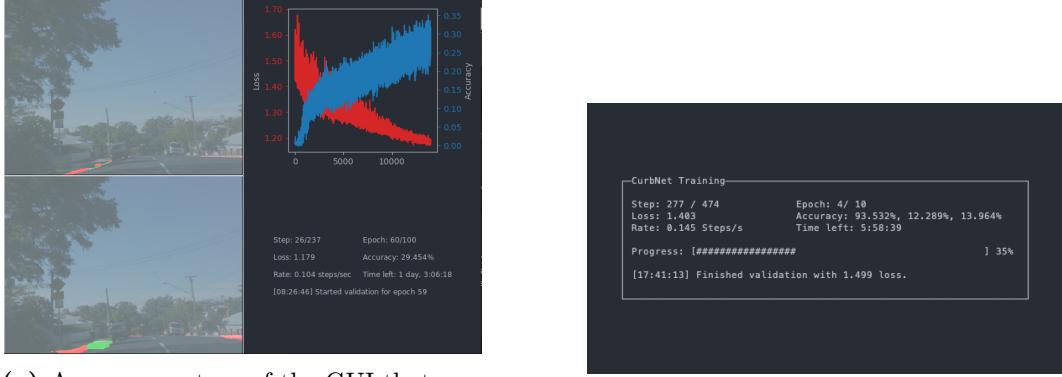
5.5 Training Pipeline

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 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 mean IoU accuracy, and other statistics. This GUI can be seen in Figure 9a.

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 9b

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



(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 9: The two available user interfaces available to use with CurbNet.

network, optimizer, and loss criterion can be easily swapped by changing command-line arguments or parameters in the JSON file. This makes running different experiments straightforward.

5.6 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 done on a per class basis and then averaged to get the mIoU.

5.7 Results

Running our network with the optimal hyperparameters chosen by the hyperparameter search in 5.4 for (**TODO: how many iterations did we train**) iterations, we were

5 Experiments

able to achieve an accuracy of **(TODO: value)**. We were able to train the network for **(TODO: how many before overfit)** before the network started overfitting to the training set.

The network is able to produce the segmentations seen in **(TODO: include figure)**.

Analyzing these results, we can see that the network excels at **(TODO: what does it excel at)**. Surprisingly, the network can even identify curbs and curb cuts which are in the distance which have a relatively small size. 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 it falsely identifies as curb. An example of this can be seen in **(TODO: figure)**

The main failure modes of the network are falsely identifying non-curb road edges as curbs and identifying certain road markings as curbs. Certain road edges which have a different texture, but are not raised, are sometimes also segmented as curbs. Certain road markings which are located near road edges are also sometimes segmented as curbs. Examples of both of these cases can be seen in figure **(TODO: figure)** We hypothesize that this may be a result of the network not having depth information.

Given depth information, it may be possible to reduce these false positives and further increase the performance of the network.

6 Conclusion and Future Work

In this work we propose CurbNet, a deep learning model that is capable of segmenting 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]. We attempted to improve performance by introducing pixel coordinates to the input pixels, but this resulted in negligible gains and was thus omitted from the final model. Hyperparameter optimization was performed using the bayesian optimization and hyperband framework developed by the AutoML group at the University of Freiburg.

To evaluate our model, we used mean intersection over union and used a validation subset of the Mapillary dataset. Our network was able to perform with **(TODO: value)%** mean IoU on this validation set. It is capable of segmenting curbs and curb cuts even when they are located a distance and are small relative to the rest of the image. Failure cases typically occur at road edges which have a different material, but are not elevated, and with road markings which are located very close to the road edge.

We propose that future work should be done with additional depth information. Including this information would reduce the number of ambiguous cases where a road edge is visually similar to a curb. Furthermore, we propose fully integrating our network with the pathfinder used by Obelix to fulfill the original motivation behind

6 Conclusion and Future Work

this work. By fully integrating CurbNet into the its pathfinder pipeline, we hope that we can increase Obelix’s capabilities.

7 Appendix

7.1 Detailed Hyperparameter Optimization Results

(TODO: Hyperparameter Optimization Results)

7.2 Dataset Statistics

ToDo Counters

To Dos: 13; 1, 2, ??, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

Parts to extend: 1; 1

Draft parts: 0;

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. Kortscheder, “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

Bibliography

- 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.

Bibliography

- [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.

Bibliography

- [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 hyperparameter 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.

