Image Detection Techniques on Daimler Pedestrian Monocular Data

Christopher Ling
x24ling@stanford.edu

Abstract

The project aims to look at images from a camera attached to a car that drives around for a while and develops a model that finds pedestrians on the image by drawing a bounding box around the pedestrian. The paper employs a R-CNN to detect the pedestrians and the process can be broken down into two sections. First, critical regions are proposed on the image itself. Secondly, these critical region proposals are run through a CNN that classifies whether those regions are pedestrians or not.

1. Introduction

With the increased development of self-driving vehicles and other similar technologies, it needs to be able to sense where the obstacles are so that the vehicle will be able to safely navigate. This is incredibly critical when the obstacles are human lives, so the image detection model needs to be able to provide the vehicle information where the pedestrian is, and the first step is to detect the location of the pedestrian on the 2d image. Given the importance of safety, the pedestrian detection model must be very accurate.

The project uses the Daimler Monocular Pedestrian Detection Benchmark data, which is comprised of a several-minute car camera recording of the street as the vehicle drives. The frames have instances where pedestrians exist or not. The benchmark data also notes whether a pedestrian, vehicle, bicyclist, and motorcyclist exists in the image and where in the image the target is, defined by a bounding box. For simplicity, we will only look at pedestrians.

To classify the existence of a pedestrian will requires the use a simple CNN network with only two classes, contains pedestrians and no pedestrians. It network will only take a small region of the image as input. Since the input space doesn't need to be that large, the CNN itself doesn't need to be very complicated.

However, to determine where the pedestrian exists will be slightly trickier. The idea is that we need to test the classifier on several bounding boxes various sizes and locations. However, there is the challenge of how to determine the properties of that bounding box. The project experiments with different kinds of region extraction methods, primarily exhaustive (try out all of the regions), selective search, and edge boxes.

Figure 1. Example of a full 640x480 image from the Daimler dataset

2. Theory

A discussion on how to go about solving this problem needs to include the theoretical approach towards image classification and detection as well as a quick understanding of literature on what has been attempted and done.

The overall architecture that I will use it to first select regions in the image that will most likely have crucial sections that should be detected, then for each region perform a classification if a pedestrian truly exists in that region. In the end, we have the coordinates for the pedestrian location so the final loss function will account for the euclidean distances between the coordinates. Figure 2 shows the overall design of the detector.

2.1. Pedestrian Classification

The first problem is to create a classification of what a pedestrian is. The first thing is that we need to acquire images of pedestrians and non-pedestrians, which we use from our data from Daimler. Because this classifier should rather quick and simple, many similar papers use a linear SVM or a CNN to classify images [5, 6].

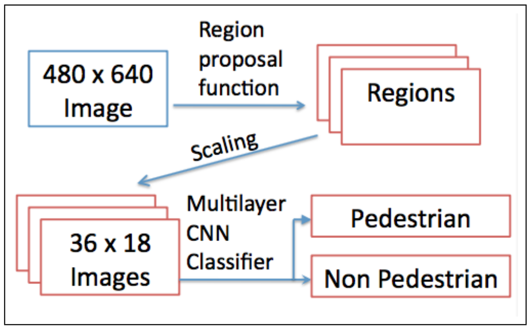

Figure 2. Overall architecture of the pedestrian detection model

There are also several methods of feature extraction that other papers have found useful, which include HOG, 1D and 2D Haar Transforms [2, 4]. These feature extractions methods have been useful for image detection, but I don't employ these methods in this project.

Furthermore, Daimler also provides image segments all of the same pixel size of just pedestrians and frames that don't contain any pedestrians. The model can use the model to train on the pedestrians as well as random segments of images with no pedestrians. When given a segment, it will output a score for the segment being a pedestrians, and a score that it is not.

Moreover, these segments need to be properly scaled so that they can be properly compared to the segments that the model was trained with. This will involve the dropping pixels if the bounding box is too large or the duplication of pixels if the bounding box is too small.

The layer architecture I use to classify the regions is a [Conv-Relu]x2 - [Conv-Relu-Pool] - [Affine]X2 - SVM. Convolution layers are used in the classifier layers. There aren't that many pooling layers because the input dimension space is small. The scoring used is SVM, which defined as

$$L_i = \sum_{j \neq y_i} max(0, s_i - s_{y_i} + \Delta) \tag{1}$$

where s_i refers to the final scoring of the ith index in the CNN before the SVM layer.

2.2. Region Proposal

The first thing we need to do is to determine on which section of the image do we perform the classification on. The easiest method would be to try out multiple detection windows of various sizes and locations at different set increments [5]. Furthermore, the detection window that classifies a pedestrian with a large enough will be the bounding box. If the pedestrian classifier is fast enough, this method might be sufficient, but classifying so many regions for simply one image can take a very long time.

If the classification ends up being very time-consuming, there must be a way to intelligently determine regions in the image that are most interesting to the model and then classify on those regions. There are several methods researched and outlined in literature [2].

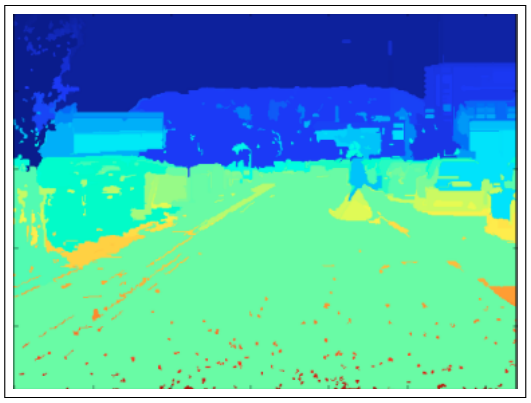

Figure 3. How Selective Search assigns regions according to how they've clustered together pixels

The first attempt will do the brute force method to get many detection windows exhaustively, but to improve the prediction time, other region selection methods such as SelectiveSearch [3] and EdgeBoxes [7] would be useful in cutting down classification time. This project attempts to propose regions exhaustively and then experiments with two other more intelligent regional proposal methods.

2.2.1 Selective Search

This region proposal method is outlined by Felzenszwalb and Huttenlocher [1] and was implemented further by a python library that was developed by AlpacaDB.

The basic premise of the this proposal system is that it groups together pixels of similar colors and draw bounding boxes around those regions. The similarity metric between pixels/regions of certain properties can be adjusted such that the regions can be grouped together or certain combinations. Even if regions have vastly different colors, they can still be a part of the same object that we wish to detect.

Figure 3 shows an example of how the selective search region proposal divides the regions of similar pixels, but the final resulting bounding boxes also take into grouping together the proposed regions. The one that is most interesting is the pedestrian that got captured by the proposal method.

2.2.2 Edgeboxes

This region proposal method is outlined by Zitnick and Dollar [7] and was implemented by them as well. They have a Matlab library that allowed them to output bounding boxes using the EdgeBox method, but I had to rewrite the code so that it was usable for python.

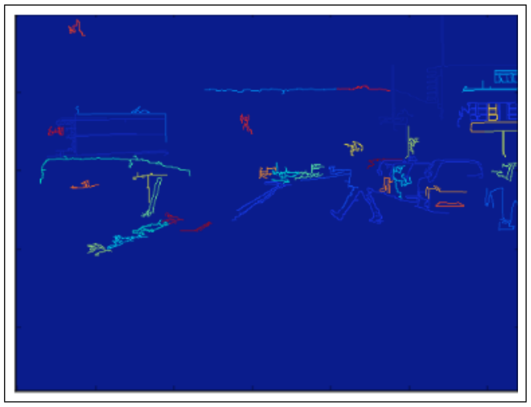

Figure 4. Edgeboxes method of grouping together edges together

Instead of simply looking at pixels, the EdgeBox method first determines the edge locations of the images, groups together edges of similar orientation, and then forms bounding boxes around the edge groups. Figure 4 shows how the EdgeBox method groups together the edges where each edge region gets grouped together into the same color. The regions then get assigned a bounding box that encompasses the edge group as well as nearby EdgeBox groups that are similar to each other. Notice how the edges that gets grouped include the pedestrian in the middle.

When training the pedestrian classifier and the detection location, we need to have a loss function that we can minimize. The classification loss function is pretty straightforward, but we need to set up a loss function for the pedestrian location. The Daimler dataset contains the coordinates of the upper-left corner and the lower-right corner. Once the image detector outputs the two coordinates for where it thinks the pedestrian is, the loss function will simply be the euclidean distances between the actual and predicted coordinates.

3. Daimler Pedestrian Dataset

The publicly available Daimler Pedestrian Detection Benchmark Dataset is used in this project contains 21790 images of resolution 640x480 from a 27 minute drive through the city. For each image, a ground truth bounding box provides where the pedestrian actually is in the image. The model will be using these images as part of the testing dataset where the predictor extracts regions and classifies each regions.

In order to train our region classifier, we need to provide region proposals of both positive and negative examples of pedestrians. The Daimler dataset further includes 15560 images of positive examples of pedestrians with resolution 36x18. The input space of the pedestrian classifier will be 36x18 as well.

As for the negative examples, Daimler provides 6744 images of 640x480 resolution that do not have any pedestrians in them. In order for the classifier to be trained, we need to feed 36x18 pixel images of non-pedestrians as well. What I did was that I used a region proposal method on the full-sized non-pedestrian images to find regions the method finds interesting, scale those images to the appropriate size, and feed them as negative examples into the classifier to train. This allows the classifier to use actual region proposals each method thinks are critical to train on.

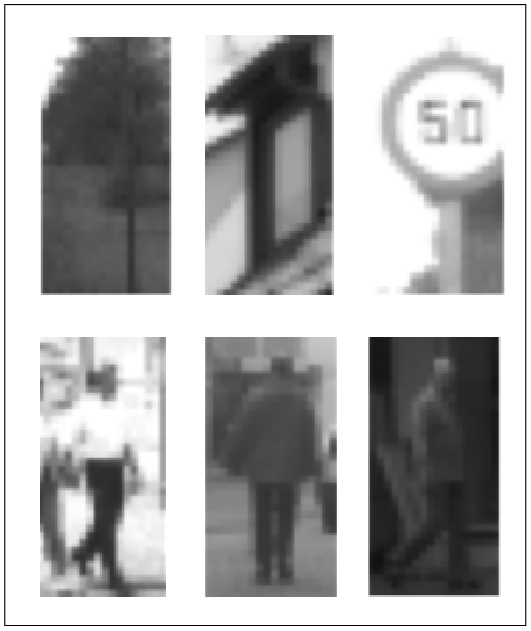

Figure 5. Negative and positive examples fed into the CNN classifier. Negative examples in the top row were extracted from the full-sized examples. Positive examples in the bottom row came directly from Daimler.

4. Implementation

4.1. Training the CNN

The first step was to develop a fast binary-output CNN that will classify the regions, so by feeding in multiple 36x18 images into the classifier and training it, a predictor was developed. Experimentally by testing out various parameters, the final architecture used is outlined in table 4.1. The total amount of memory needed to store the data for each image when predicting is around 69K x 4Bytes = 274KB per image. Furthermore, the total amount of parameters excluding the bias terms is around 2.64M x 4Bytes=105MB, most of which comes from the first affine layer.

The training parameters are also selected to as seen in Table 4.1.

Training the model for 5 epochs had very positive results. As shown in Figure 4.1, both the training accuracy and val-

Layer	Number of Filters	Size of Filters	Memory	Number of Parameters (excluding biases)
INPUT	-	-	[36x18]=648	-
CONV-RELU	32	[3x3]	[36x18x32]=20,736	[32x3x3]=288
CONV-RELU	32	[3x3]	[36x18x32]=20,736	[32x3x3]=288
CONV-RELU	32	[3x3]	[36x18x32]=20,736	[32x3x3]=288
POOL 2	-	-	[18x9x32]=5,184	-
AFFINE	-	-	[500]	[5184x500]=2,592,000
AFFINE	-	-	[2]	[500x2]=1,000
SVM	-	-	[2]	-

Table 1. Layers of the Region Convolutional Neural Network Classifier

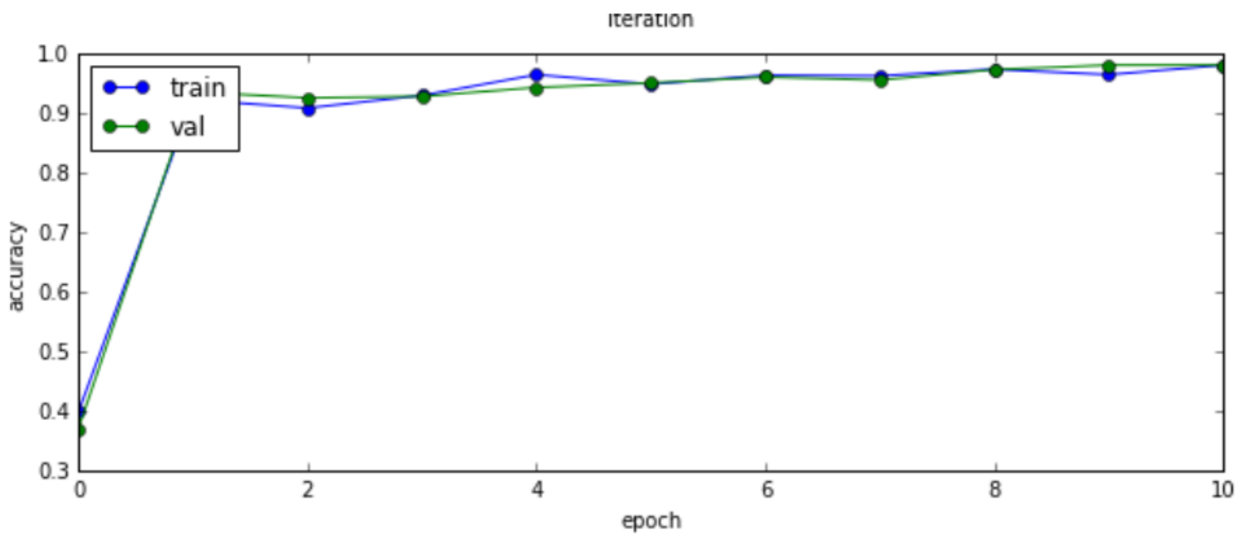

Figure 6. Training and Validation Accuracy of CNN on 36x18 images

Parameter	Value
Learning Rate	8 x 10^-4
Weight Scale	1 x 10^-3
Number of Batches	50
Epochs Trained	10
Training Time	Approx 7 hours

Table 2. Training Parameters Convolutional Neural Network Classifier

idation accuracy were very close to each other, showing no signs of over-fitting on the training data. Furthermore, the accuracies of both datasets are very high, both above 90 percent accurate. The runtime of the classifier itself is quite fast as well, which makes this model perfect for quickly classifying regions.

I further tested the three different region proposal methods: Exhaustive, Selective Search, and EdgeBox. These methods are open-loop, meaning they don't really take into account the error of detection of the image. They do have some parameters that can be adjusted, such as the stride/sweep of the bounding box as well as the size of the bounding box. Not a lot of time was spent on setting those parameters, mostly the default recommended values that were given in the paper were used.

5. Results

Overall, what was most interesting was that both Selective Search and EdgeBoxes were able to capture the pedestrian on the image with some relatively success, but what ended up happening was that there were a lot of false positives in the output of the model.

As a case study, let's look at the same image that I've been using as the example. Figure 7 depicts the actual location of the pedestrian, and as you can see, there is only one pedestrian.

Figure 8, 9, and 10 shows all the bounding boxes that the model interprets as a pedestrian, and while all methods end up producing a box that does indeed surround a pedestrian, it doesn't do a good job with rejecting the boxes that are not pedestrians.

When measuring the error of the bounding box, we take

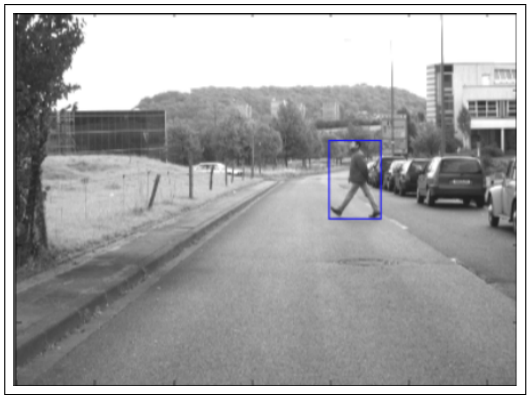

Figure 7. Actual Bounding Box of Pedestrian given by Daimler.

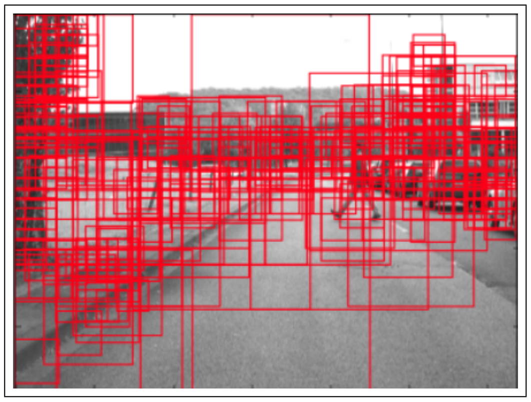

Figure 8. Detected Pedestrians on Image using the Exhaustive Region Proposal Method.

the euclidean distance between the the top-left corners and the bottom-right corners of the actual bounding box shown in figure 7 and the output bounding box from the predictor. In the case of multiple bounding boxes returned by the predictor, Table 5 depicts the smallest euclidean error, in order words the bounding box that's closest to the actual bounding box.

In Figure 8, the bounding boxes seem to detect the regions with interesting features, most notably the buildings, cars, tree, and fence posts. This method seems really bulky because the predictor have to sort through so many regions which increased the runtime of the predictor and decreased the accuracy of the region classifier since there were so many regions to process through.

When using either Selective Search or EdgeBoxes, the runtime of predictor goes down dramatically as shown in Table 5 because the number of region proposals decreases radically. It is possible to reduce the runtime of exhaustively proposing regions by reducing the step size, but this would sacrifice bounding box frequency, which in this instance, Exhaustive search has the best specification on.

In Figure 9, there are fewer false positives and the

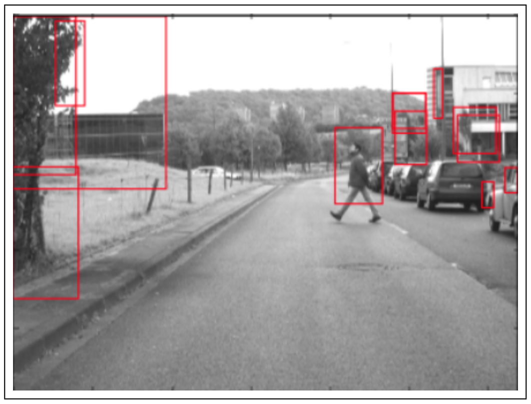

Figure 9. Detected Pedestrians on Image using the Selective Search Region Proposal Method

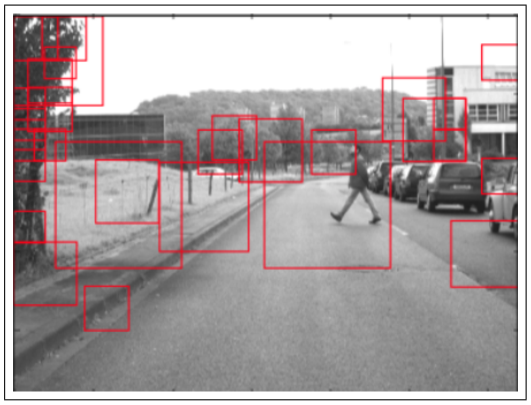

Figure 10. Detected Pedestrians on Image using the EdgeBox Region Proposal Method

method was able to find the pedestrian. The bounding box that was proposed is slightly off because it does not include the feet of the pedestrian. This was a common issue with the Selective Search method: it oftens cuts off portions of the pedestrian whether it is the legs or the upper body. This is probably due to the colors of the upper and lower body of pedestrians tend to be different and get separated into two regions. This causes the resulting bounding boxes to only capture the halves of the pedestrian. There would need to be some adjustment of the parameters to allow the combination of nearby regions so that both halves of the person gets combined.

In Figure 10, the runtime is even faster, but the error of the bounding box is larger. This time, the bounding box is slightly larger than the pedestrian. What probably happened was that the region proposed saw the edge from the street as a critical edge group that should be combined with the edge from the pedestrian and the classifier ended up detecting that region as the pedestrian.

What seems to be a common mistake between all methods is that the classifier seemed to predict the trees and

Region Proposal Method	Time	Smallest Euclidean Error	Proposed Regions	False Positives
Exhaustive	602.6s	20.85	96114	8134
Selective Search	29.46s	33.67	46	13
EdgeBoxes	22.59s	99.5	100	20

Table 3. Results of R-CNN model on image set

buildings as pedestrians.

6. Conclusions and Future Development

We managed to improve the runtime and reduce the portion of false positives in the image by using more intelligent region proposal methods to detect pedestrians in the image; however, the accuracy of the bounding box falls slight. In the instance of selective search, the box encompasses too small of a region while EdgeBox encompasses too large of a region.

Overall, the process detected pedestrians too easily. The challenge with the classifier is that more than half of the training example fed to the classifier are pedestrians but the number of times the classifier detects a pedestrian should ideally be very very low. For example, for a given image, there should only really be a handful of pedestrians but the number of region proposals is going to be significantly larger than the number of actual pedestrians.

One thing that can be done is to only report bounding boxes that have a much larger score of being a pedestrian than a score of not being a pedestrian. Furthermore, I would like to try out using pedestrian images of a higher resolution so that we have better images to train on.

References

[1] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision, 59(2), 167181*, 200.

[2] P. D. B. S. J. Hosang, R. Benenson. What makes for effective detection proposals? *PAMI*, 2015.

[3] T. G. J. R. R. Uijlings, K. E. A. van de Sande and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision, 104(2):154171*, 2013.

[4] A. D. K. Piniarski, P. Pawowski. Video processing algorithms for detection of pedestrians. *CMST*, 21(3):141–150, 2015.

[5] D. M. G. M. Enzweiler. Monocular pedestrian detection:survey and experiments. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 31(12):2179–2195, 2009.

[6] M. N. V. E. Neagoe, C. T. Tudoran. A neural network approach to pedestrian detection. *Proc. of ICCOMP09*, pages 374–379, 2009.

[7] C. L. Zitnick and P. Dollar. Edge boxes: Locating object proposals from edges. *ECCV*, 2004.

CNN code was adopted from Python code developed from Stanford University's CS231N modules. Python libraries selectivesearch and opencv2 were heavily used. edgebox libraries from Dollar [7] repurposed for use in Python.

Much acknowledgement to the entire CS231N teaching staff for a phenomenal quarter.