Patch-Based Real Time Road Object Detection Using YOLOv3

Megan Rowe
Mechanical Engineering
Stanford University
mrowe2@stanford.edu

Naveen Krishnamurthi
Computer Science
Stanford University
naveenk1@stanford.edu

Puyang Ma
Data Science
Stanford University
pym86@stanford.edu

Abstract

One of the key challenges in autonomous vehicles is building accurate real time object detection algorithms. Traditionally, there tends to be a tradeoff between speed and accuracy, and a standard approach is to decrease the resolution of input images in order to increase the speed of detection. The drawback of this approach is that it can make it difficult to detect small or more distant objects. In this paper, we examine how running parallel object detection on multiple patches of a high definition (HD) image can allow us to detect smaller objects in the overall image without decreasing its resolution. We found that precision for smaller object classes tended to improve, precision for larger object classes decreased slightly, and overall mean average precision was comparable.

1 Introduction

We investigate how real time object detection can be applied to high resolution driving images. The input to our algorithm is an HD image captured by a driving system, and the output is a series of annotations for the common road objects (ie. traffic signs, traffic lights, cars, and people) contained within the given image. Each annotation consists of an x_1 and y_1 value, which represents the top left corner of the object, a w value which represents the objects width, an h value which represents the object's height, a class value which represents the object's class, and confidence value, which represents the models confidence with regards to the object's classification. This problem is an important area of research that helps lay the foundation for self-driving vehicles, as the location and context of nearby objects can help inform autonomous decision making.

2 Related work

The advent of neural networks has enabled substantial improvements when it comes to object detection, and the state of the art for object detection currently relies on a number of different CNN-based architectures, including Faster R-CNN [1], YOLOv3 [2], and SSD [3]. Of these, YOLOv3 is particularly noteworthy because it runs in real time, which makes it an ideal match for the self-driving vehicles use-case. Applying YOLOv3 to HD images is challenging because it generally involves downsampling the images to a lower resolution, which often makes it difficult for a given model to detect small or distant objects [4]. In the past, researchers have demonstrated success in avoiding many of the problems associated with downsampling HD images by employing patch-based approaches for semantic segmentation tasks [5]. In our project we employ a similar patch-based approach for object detection that involves splitting the HD input image into smaller image patches, applying YOLOv3 to each of these patches, and merging the resulting detections to generate an output detection for the original HD image.

3 Dataset and Features

Our dataset consists of 100,000 images and labels from the BDD-100K dataset [6]. The images for the BDD-100K dataset were collected from over 1100 hours of driving in a variety of different locations, driving scenarios, and weather conditions. They contain RGB channels and are of size $1280x720$. The labels for the images are JSON formatted and encode 2D bounding boxes and Classifications for a number of common road objects including buses, traffic lights, traffic signs, people, bikes, trucks, motorcycles, motorcycle riders, cars, and trains. The BDD-100K dataset consists of a training set with 70,000 image-label pairs, a test set with 20,000 image-label pairs, and a validation set with 10,000 image-label pairs. A graph of the distributions of the different objects in the dataset is provided below [7].

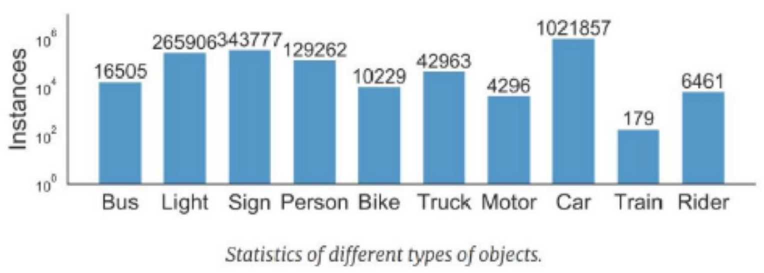

Figure 1: Prevalence of Different Road Objects in the BDD-100K Dataset.

In order to implement the patch-based real time road object detection, we wrote a script to crop square patches by sliding a square window over the original 1280 x 720 images. We originally set the window size to be 416 x 416 because that is the input size of YOLOv3's detector. We confirmed that the window size is reasonable by computing and found the median bounding box size of all objects in the validation set to be about 29 x 29. We updated labels of objects in a cropped patch based on how much an object's bounding box is contained in a patch. If the original bounding box is 100% contained in a patch, it is kept unchanged. If the bounding box is at least 30% contained in a patch, it gets updated by cutting off the part outside. If It is less than 30%, it is eliminated. Since patches are now the inputs to YOLOv3's detection, bounding box coordinates were also updated to be relative to a patch. We tested several choices of thresholds from 30% to 70%, and did not observe much difference in our model performance. Since we designed the stride size of the sliding window as a parameter, we also had to pad crops to ensure our patches are all 416 x 416. We tested and compared our model performance for stride size 100 and stride size 416, and found the latter to be better. This is probably due to few new information learned from many overlaps of patches when the stride size is small. Finally, we merged the bounding box detections from the individual crops by converting the patch relative coordinates back to the full size image relative coordinates.

4 Methods

For both our baseline and patch-based models, we fine tuned a PyTorch implementation of YOLOv3 that was pre-trained on ImageNet [8]. We modified the repository to account for our object classes (traffic light, traffic sign, person, and car) and training data format. The baseline model was trained on 70,000 images that were resized and padded to 416x416 and the second model was trained on 12,000 416x416 crops generated from 2,000 randomly selected full sized images of the same distribution as the baseline model.

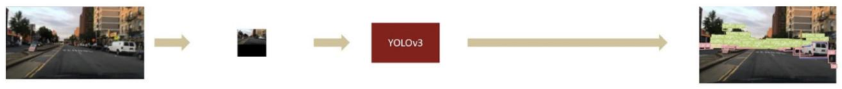

Figure 2: Baseline Model process

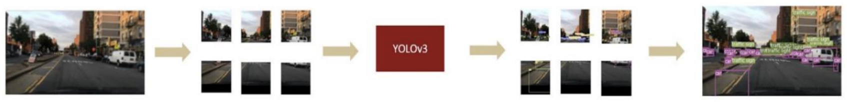

Figure 3: Patch-Based Model process

For our baseline model, we used the default training hyperparameters given in the repository since they were successfully used to train the COCO dataset, which includes similar object classes to our own. Due to the large size of our training set and memory constraints, we were only able to train the baseline model for 10 epochs. When training our patch based model, we ran into issues with an overly large training set since each training image was cropped into 6 smaller images. This would equal a training set size of 420,000 images for our patch based model which would take too long for us to train. We decided to use a 2,000 image subset of the training set that was representative of the class proportions we used in our baseline model to get results more quickly. We trained the patch-based model until we saw the loss start to converge at 144 epochs with the same hyperparameters as the baseline model, but adjusting the learning rate every time the total training loss would start to oscillate. We used a validation set size of 500 (out of the full 10,000 images in the BDD dataset) because of the significant amount of extra computational power it took to run our testing script on the patch-based model. Given more compute, we would have liked to run the full validation set on our images. Our

Training Hyperparameters	Baseline Model	Patch Based Model
Batch size	16	16
Learning rate	.001	.001 (Epochs 0-31) .0001 (Epochs 32-125) .00001 (Epochs 126-144)
decay	.005	.005
Input image size	1280x720 (resized and padded to 416x416)	416x416
epochs	10	144
Confidence threshold	0.8	0.8
NMS threshold	0.4	0.4
momentum	0.9	0.9
Training examples	70,000	12,000 (generated from 2,000 full size images)
Validation set size	500	500

Table 1: Training parameters for baseline model and patch-based model

training loss for the baseline model settled around 1.0 and the patch-based model settled around 0.2. The average precision in our last epoch of training was 0.352 for the baseline model and 0.684 for the patch-based model.

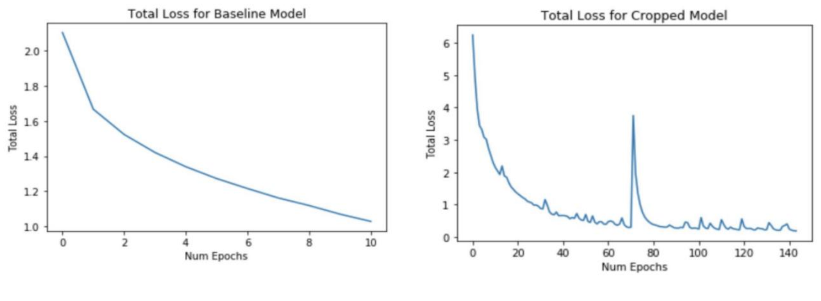

Figure 4: Training Losses

5 Experiments/Results/Discussion

Overall, the mAP for our baseline model and patch-based model is very similar. When we examined the AP of each class, we found that our baseline model performed better on larger objects like cars and pedestrians, while our patch-based model performed better on smaller objects like traffic lights and traffic signs. Our baseline model was unable to learn these smaller objects because our input

Class	**Baseline AP**	**Patch-Based AP**
Traffic Light	0.23298267714457677	0.26508976596455036
Traffic Sign	0.1293044049833006	0.2611623498825088
Person	0.22894372776494953	0.13484839086805916
Car	0.44440030362719785	0.3046103773117473
Total	0.2589077783800061875	0.2414277210067164

Table 2: Validation Set Results

image size of 1280x720px was shrunk down and padded to be the 416x416 size that YOLOv3 takes as its input. When the image was shrunk down, many of the already small objects were sized down to very few pixels and were too small for our network to learn, leading the model to have an extremely high false positive rate on smaller objects as shown in the images below.

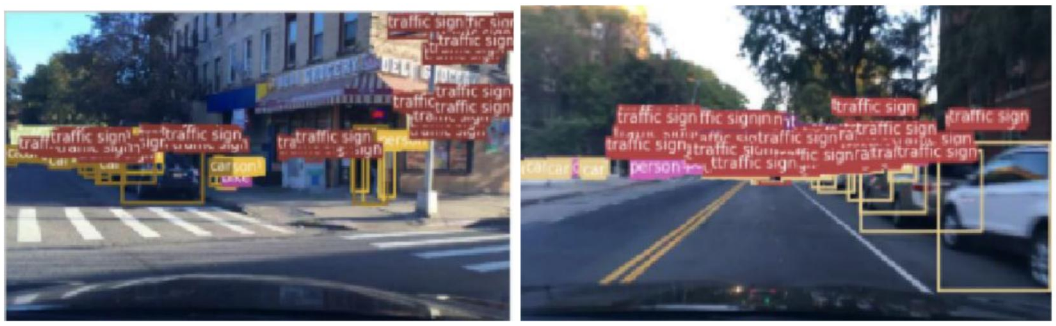

Figure 5: Baseline model has high false positive rate on smaller objects

We predicted that our patch based model would eliminate this issue, since it didn't involve decreasing the resolution of the input image. One disadvantage of the patch-based model is that it cannot detect objects that are larger than the 416x416 patch size. However, the majority of the objects in our images are smaller than 416x416, with the median bounding box size in our validation set being 29x29. Still, we found that the mAP scores for larger objects like cars and pedestrians were lower than the baseline model. Below, we see the false positive rate significantly improve in the patch-based model when compared to the baseline.

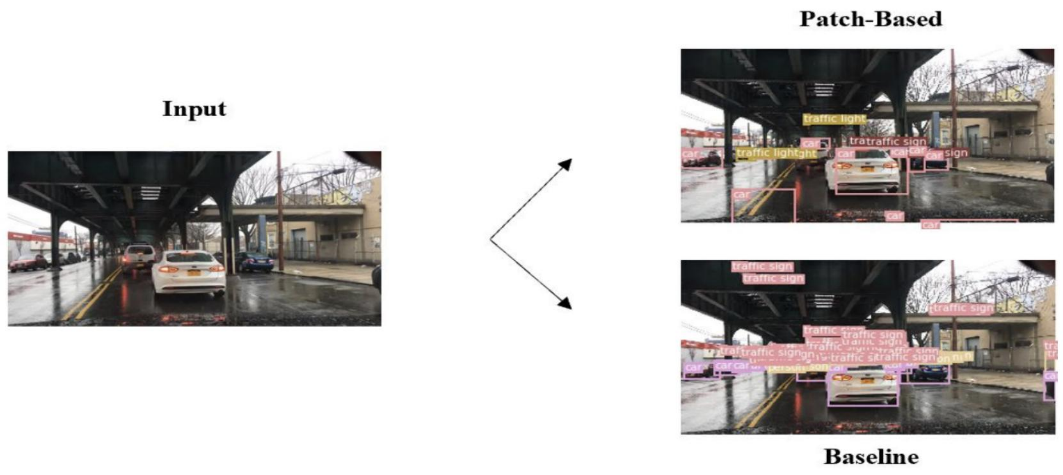

Figure 6: Less false positives in the patch-based model

6 Conclusion/Future Work

The patch-based model appears to have promising results. It achieved a mAP score that was comparable to our baseline model, with significantly fewer false positive predictions of traffic lights and traffic signs. However, it is difficult to compare the results directly, because the two models were trained on different training set images with a different number of epochs due to computational constraints. With more computational resources, we could gain further insights by training the baseline model for longer and training the patch-based model on our entire training dataset.

In addition, we would like to experiment with other more advanced strategies for generating and merging patches. If we had used a smaller stride on our patches and combined patches based on IoU, we might have been able to improve our ability to detect objects that were larger than our sliding window size. Lastly, we would like to spend more time productionizing some of the real time elements of the patch-based model.

7 Source Codes

Source codes are available at https://github.com/naveen401/cs230

8 Contributions

All authors contributed equally to the project. M.R. was primarily responsible for training and preliminary data preprocessing. N.K. was responsible for testing the models, configuring the development environment, and assisting M.R with training. P.M. was responsible for the data preprocessing for the patch-based model and detector modification to help with merging by non-maximum suppression. All three authors were responsible for creating the poster and writing the final paper.

References

[1] Ren, Shaoqing., et al. (2015) Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*.

[2] Redmon, Joseph. & Beeman, D. (2018) Yolov3: An incremental improvement. *arXiv:1804.02767*.

[3] Liu, Wei., et al. (2016) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *European conference on computer vision*. Springer.

[4] Xu, Jianfeng., Lertniphonphan, Kanokphan. &Kazuyuki, Tasaka. (2018) Fast and Accurate Object Detection Using Image Cropping/Resizing in Multi-View 4K Sports Videos. *Proceedings of the 1st International Workshop on Multimedia Content Analysis in Sports*. ACM.

[5] Camilo, Joseph., et al. (2018) Application of a semantic segmentation convolutional neural network for accurate automatic detection and mapping of solar photovoltaic arrays in aerial imagery. *arXiv:1801.04018*.

[6] Fisher, Yu., Wenqi Xian. &Yingying, Chen., et al. BDD 100K: A Diverse Driving Video Database with Scalable Annotation Tooling.

[7] Available at https://bair.berkeley.edu/blog/2018/05/30/bdd/.

[8] Erik, Lindernoren. PyTorch-YOLOv3. https://github.com/eriklindernoren/PyTorch-YOLOv3