Object Detection for Autonomous Vehicles

Gene Lewis
Stanford University
Stanford, CA
glewis17@cs.stanford.edu

Abstract

In this work, we have examine an approach to deep object detection that makes bounding box predictions for an image without the need for expensive preprocessing or expensive deep evaluations; the resulting DIY network, SimpleNet, gives reasonable prediction accuracy and runs in near-real-time. SimpleNet is capable of making predictions at a rate of approximately 11 frames per second, and achieves a mAP accuracy of around 12.83% *on our validation set.*

1. Introduction

In recent years, there has been a significant increase in research interest supporting the development of the autonomous vehicle, which is an automobile platform capable of sensing and reacting to it's immediate environment in an attempt to navigate roadways without human intervention. The task of environment sensing is known as perception, and often consists of a number of subtasks such as object classification, detection, 3D position estimation, and simultaneous localization and mapping (SLAM)[6].

In many autonomous driving systems, the object detection subtask is itself one of the most important prerequisites to autonomous navigation, as this task is what allows the car controller to account for obstacles when considering possible future trajectories; it therefore follows that we desire object detection algorithms that are as accurate as possible. Many high-quality object detectors have seen astounding improvements in recent years [5]; however, the object detection problem ported to the autonomous driving setting presents unique challenges not addressed by many leading techniques. These challenges include:

- Car controllers often must solve optimization problems at least once a second to achieve feasible control; this means that the car controller must receive detection results in a similar timeframe.
- The requirement that the entire detection pipeline be fast generally precludes the use of image preprocessing that often boosts detection performance, such as extracting SIFT descriptors, region proposals, or sliding windows. Thus, a detector for an autonomous car must remain reasonably accurate while operating on strictly the input image.
- Car computing systems are often heavily memory constrained, and so it is often infeasible to store and run detectors with large amounts of parameters, especially with large input image volumes. In particular, this constrains the depth of neural network approaches.

2. Related Work

2.1. OverFeat

OverFeat [10] is a Convolutional Neural Network model released in 2013 that jointly performs object recognition, detection, and localization. OverFeat is one of the most successful detection models to date, winning the localization task in the ImageNet Large Scale Visual Recognition Challenge 2013. OverFeat is eight layers deep, and depends heavily on an overlapping scheme that produces detection boxes at multiple scales and iteratively aggregates them together into high-confidence predictions.

2.2. VGG16

VGG16 [11] was an attempt to usurp OverFeat's dominance in object classification and detection by exploring the effects of extreme layer depth. A 16-layer and 19-layer model was produced, setting new benchmarks on localization and classification in the ImageNet ILSVRC 2014 Challenge.

2.3. Fast R-CNN

Fast R-CNN [3] is a model that attempts to capture the accuracy of deeper models while improving their speed. Fast R-CNN predicts on region proposals, and utilizes shared computation per region proposal and truncated SVD factorizations to speed up the training and prediction time of the model.

2.4. YOLO

YOLO [9] is a recent model that operates directly on images while treating object detection as regression instead of classification. YOLO has the poorest performance out of all the above models, but more than makes up for it in speed; YOLO is able to predict at an astounding 45 frames per second. YOLO is, however, quite large at 12 layers.

2.5. SimpleNet

In this work, we investigate the possibility of using simple techniques to construct an object detector that excels in the metrics provided by our autonomous driving environment; that is, we endeavor to train a detector that is capable of fitting in restricted memory, predicts in real-time, and has acceptable accuracy for use in an autonomous driving platform. We name our resulting detector SimpleNet; our approach and experiments are detailed below.

3. Approach

3.1. Architecture

We utilize Convolutional Neural Networks (CNN) as our basic structure for SimpleNet, as CNNs have recently been shown to achieve very significant results in computer vision tasks [5]. A CNN model can be thought of as a recursive composition of models, where the input at the lowest level is some input image and the output is a matrix of bounding box predictions.

The choice of functions to recursively compose varies widely, but often involves a convolutional operation followed by a non-linear activation followed by a maximum pooling operation; we refer to the composition of these three functions together as a "layer".

In our SimpleNet architecture, we choose to have n layers, followed by a final convolution/non-linearity and a dense fully-connected layer that outputs a prediction matrix. We trained a 3-layer and 5-layer model to examine the effects of network depth on predictions; more details are given in the Experiments section.

3.2. Hidden Units

For each Convolution Layer, we utilized a filter size of $3x3$ in order to train each neuron to respond to a more local area and get the maximum number of activations. Each Convolution Layer also had 3 neurons; this choice was made to keep the volume size similar to the input volume, and keep the number of parameters fairly low.

3.3. Non-linear Activation Function

For our non-linear activation we used Rectified Linear Unit (ReLU) [7] activations, which take the form:

$$ReLU(x) = \max\{0, x\}$$

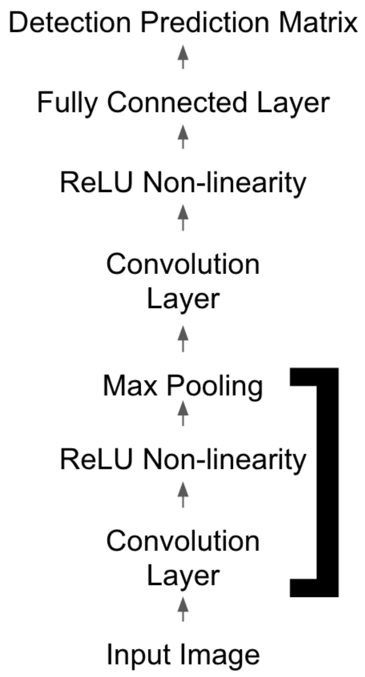

Figure 1: SimpleNet Architecture. Dark bracket indicates portion with repeated layers

ReLU's are standard activations used in the computer vision literature that allow for large gradients while suppressing negative activations.

3.4. Detection Prediction

We represent a bounding box as a vector $b \in \mathbb{R}^4$, where the first two elements encode the x and y coordinates of the top-left corner of the bounding box detection and the last two elements encode the x and y coordinates of the bottom-right corner of the bounding box detection. Our detection matrix has a fixed number of rows, and so can predict a fixed number of bounding boxes; in our implementation, we cap our number of predictions at 21, corresponding to the maximum number of detections in the training and testing sets.

3.5. Horizon Suppression

After predicting bounding boxes, we utilize a heuristic to help suppress false positives which we call horizon suppression. Because the cameras on the autonomous car are fixed, most road surfaces appear at the same angle and so have the same horizon line. As a heuristic, we suppress any detection box whose bottom-right corner appears above the estimated horizon line $y = 200px$; the intuition behind this procedure is that if the detected object is above the horizon line, either it lies very far ahead in the distance or lies in the sky. If it lies in the sky, then the detected object cannot be a car or pedestrian and so is a false positive; otherwise, we will have a high chance of detecting the object again as we approach closer and the predicted bounding box falls below the horizon line.

This procedure significantly improves false positives, as many insignificant predictions are made above the horizon line as an artifact of the training procedure.

3.6. Loss Function

In this work we experimented with two different loss functions on the element-wise difference between the predicted detection matrix and ground truth matrix: vector L1 loss and vector L2 loss. Let $vec(X)$ denote the operation reshaping the matrix $X \in \mathbb{R}^{n \times m}$ into a vector $x \in \mathbb{R}^{nm}$, let A be our prediction matrix, and let B be our ground truth detection matrix; our two losses are then:

$$L1(A-B) = \sum_i |vec(A-B)_i|$$

$$L2(A-B) = \sum_i vec(A-B)_i^2$$

Note that the L2 loss penalizes large differences between the predicted and ground truth matrices more harshly than the L1 loss; this turns out to be a very desirable property that leads to much better training in experimentation.

In our loss, we also incorporated L2 regularization on our weight matrices in order to combat overfitting; over 10 trial runs, we found that a regularization value of $1e-5$ works best.

3.7. Training Algorithm

For our optimization procedure, we utilized Adaptive momentum (Adam) [4], which updates each parameter of the network with a separate momentum parameter that serves as a kind of average of previous gradient updates and helps accelerate training and convergence.

For our learning rate, we used $1e-3$. We neglected to use learning rate annealing, as we couldn't find a set of decay rate and decay steps over 10 trials that resulted in better performance than no decay at all.

For our batch size, we chose to train on one image at a time in order to maximize the utility of each training example. We trained for a total of 30 epochs.

4. Experiments

4.1. Dataset

The dataset we utilize is the KITTI Object Detection Benchmark 2012 [2]. This dataset consists of 7481 training images and 7518 testing images; these images are large and high-resolution, and so in the interests of space and time we only train and test on a subset of this consisting of 1000 training images and 100 testing images.

Each image is accompanied by a list of detections, where each detection gives the classification of the detection, the image bounding box coordinates, the width, length, and height of the detection in meters, and the 3D position and orientation of the detection in world coordinates. In this work, we only leverage the bounding box ground truth predictions, but believe that our approach should be easily extensible to predicting the other real-valued ground truths by simply extending the number of columns in the detection prediction matrix.

For data normalization and preprocessing, we omit various preprocessing techniques such as lighting normalization, orientation adjustment, and random cropping; many of these techniques are infeasible to perform in real-time on an autonomous system, and don't necessarily reflect real-world detection scenarios. As the Convolutional Network can only ingest a fixed size image, we pad each image that is less than a threshold height and width with rows of zeros along the bottom border and columns of zeros along the right-most border. The thresholds for padding that we used were 376px for the height and 1242px for the width; the total number of images padded was 14.

4.2. Computing Environment

For our software framework, we leveraged the recently-released open-source machine-learning package Tensorflow [1] as it is fast (the computation graph is written and calculated in C++), easy to use (it has a python interface and easy install instructions for Ubuntu), is able to take advantage of GPU acceleration (a computational choice shown to dramatically speed up network training [8]), and circumvents some of the issues found in other graph computation frameworks (such as long compile times for especially deep networks).

For our computing environment, we utilized an Amazon AWS g2.2xlarge machine running Ubuntu 14.04; this environment was practical since it is cheap, customizable, and provides access to GPU hardware with pre-configured NVIDIA drivers. The specs of the machine include an NVIDIA Grid K520 GPU, 60GB of hard disk space, 15GB of RAM, and a 8 core CPU.

4.3. Training

In our training, we ran ten trials apiece of four experiments: two different architectures (3-layer and 5-layer) with two different losses (L1 and L2); the plots of training and validation losses for each experiment can be seen in Figures 2 & 3. For both losses, the 5-layer model severely overfits the data, with validation loss being significantly higher than training loss. With L1 loss, we note that both models overfit the data; we reason that this is because the L1 loss is not a severe enough penalty to encourage weights to update rapidly enough for bounding box predictions to align with

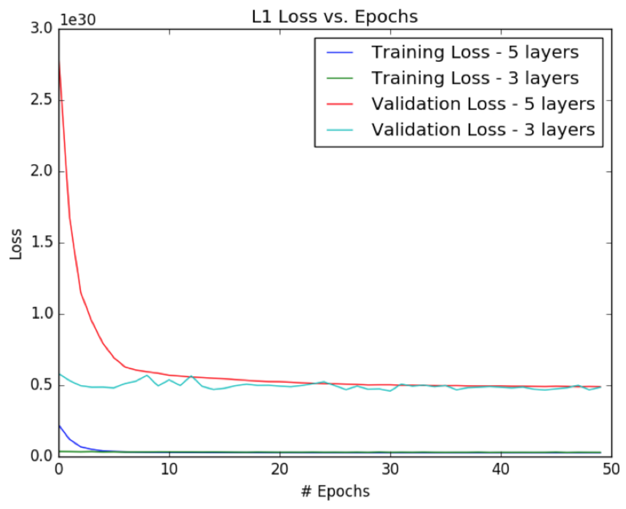

Figure 2: Training and Validation values with L1 loss function

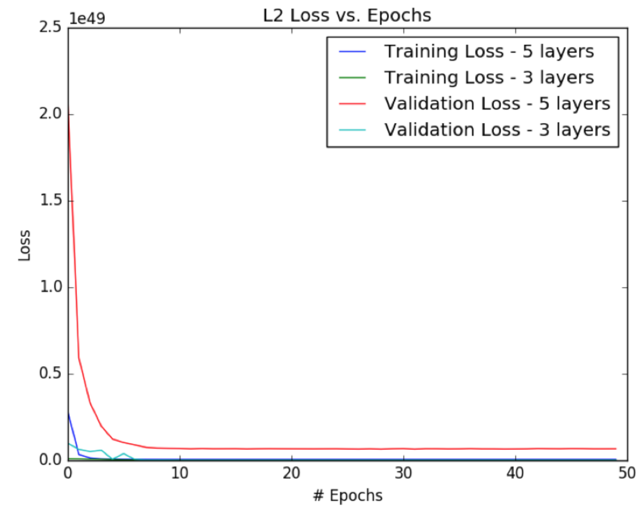

Figure 3: Training and Validation values with L2 loss function

their ground truths.

We then tried the L2 loss; we see that the overfit gap is significantly smaller than in the L1 case. The 5-layer model still overfits by a small amount, but the validation loss is lower than the training loss for the 3-layer model, indicating a good fit. We therefore used the 3-layer model trained with L2 loss as our testing model. For all curves shown, we fit the best of ten trials to allow for variation from randomized parameter initialization.

5. Results

5.1. Quantitative Results

For our quantitative results, we examined the mean Average Precision score, which is defined as the area under the Precision-Recall curve. We determine the true positive and false positive rate of our prediction via the Intersection over Union (IoU) metric, which takes the area of overlap between our predicted box and the ground truth box divided by the total area covered by our predicted box and the ground truth box; if this overlap is is greater than some threshold, then we mark our detection as a true positive. Here, we used a threshold of 0.5. On our validation set, our 3-layer L2-loss model achieves a mAP value of around 12.83%. Though the related work mentioned in this paper hasn't been evaluated on the KITTI Object Detection Benchmark, it's clear that there is significant work to be done in improving the accuracy of this approach.

We also compare our model with recent work in the literature in terms of prediction speed. To be considered "real-time", a detection algorithm needs to predict at a rate of at least 30 frames per second, or 0.033 seconds per image [9]. In 10 trials, our model predicts at a rate of, on average, 0.092 seconds per image, 3x slower than the minimum rate considered for real-time. Comparison with other methods can be found in Figure 4. We see that, though SimpleNet achieves respectable prediction time, it is still significantly outpaced by the YOLO architecture. Note that, in the below table, region proposal time can take up to 2s [3].

Model	**Time to Predict Single Image**
VGG16	47s + region proposal time
Fast R-CNN	.3s + region proposal time
SimpleNet	**.09s**
YOLO	.0222s

Figure 4: Prediction Speeds for Deep Object Detectors

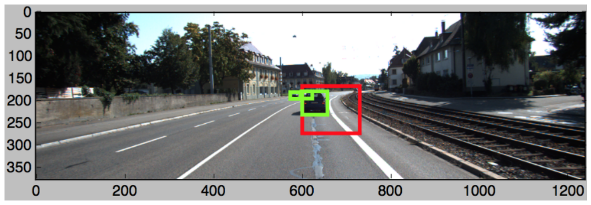

Figure 5: Positive prediction results on a training image

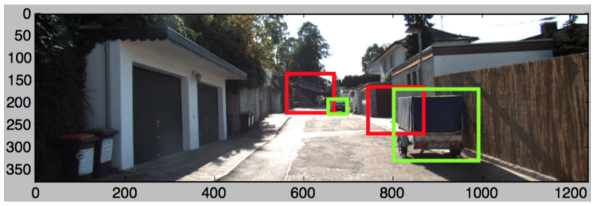

Figure 6: Positive prediction results on a validation image

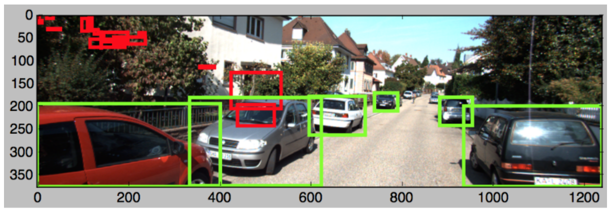

Figure 7: Negative prediction results on a validation image

5.2. Qualitative Results

For qualitative results, we examine successful and unsuccessful examples of detections to better understand our model's performance.

In Figure 5, we have an example of a detection where most of the area subsumes the ground truth predictions; though there are multiple ground truth predictions and our detector only emits one, we still consider this a success since the box is localized relatively correctly and the information passed to a car controller would be enough for the car to take a decisive action at the next timestep.

In Figure 6, we have a situation where there are two ground truth predictions and our detector has output two detections; in this scenario, the detector boxes aren't well-aligned with the ground truths. Though this situation could be improved (and indeed, counts as two false positives according to our mAP threshold), this isn't exceedingly poor behavior given that the closer of the two objects is detected relatively well; hopefully, detection at the next timestep would offer a refined estimation of the bounding box location.

In Figure 7, we have an example of multiple severe misclassifications, and have also turned off horizon suppression. We can see that there is a cluster of small misclassifications in the top-left corner of the image; we hypothesize that these misclassifications are an artifact of our training scheme, where rows that often see no detections are biased towards zero (and thus appear in the top-left corner). This also helps explain why detection is so poor when there are multiple detections spread out throughout the image: the prediction scheme seems to learn likely locations of bounding boxes along with image patterns that prompt a detection, leading to a dependence of bounding box distribution in the training data. We hypothesize that this can be remedied by using a different loss function, such as one that directly takes horizon suppression and the Intersection over Union metric into account. We leave such pursuits to further work.

6. Conclusions

In this work, we have examined an approach to deep object detection that makes bounding box predictions for an image without the need for expensive preprocessing or expensive deep evaluations; the resulting DIY network, SimpleNet, gives reasonable prediction accuracy and runs in near-real-time. Though by no means state-of-the-art, SimpleNet is an interesting look into the power of loss functions; while most networks derive power from depth of layers, we look to derive power from suitable loss functions with a limited number of parameters and functional representation. Interesting future work includes trying to extend the training loss function to more accurately capture the desired metrics at hand, increasing the prediction speed by sharing computation, and increasing the size of data trained and tested on.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[3] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

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

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In

Advances in neural information processing systems, pages 1097–1105, 2012.

[6] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. 2002.

[7] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[8] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 873–880, New York, NY, USA, 2009. ACM.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*, 2015.

[10] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.