Real-Time Object Detection

Xiaolan Cai, Warren Ferrell, Chu-Sheng Ku, Soham Shah, and Ryan Skinner

February 2018

1 Introduction

As autonomous vehicles become increasingly integrated into our everyday life, it is critical that these vehicles are able to operate effectively and safely in any environment. One of the big challenges in this area is making sure all the computation can be done quickly and in a cost effective manner.

Of the $250,000 estimated cost total of a self-driving car, $5,000 is devoted to computational resources [8]. This high price tag doesn't even include the costs of developing the software, electrical, and mechanical systems necessary for all of this to work that will likely be amortized over the sales of many cars as production scales up. While this certainly sounds like a hefty investment in electronics, it actually leaves a lot to be desired in terms of processing power. High-quality sensors by themselves are expensive and cars often need a variety of sensors facing all directions around the car. Furthermore, there needs to be redundancy within the car's computer systems to meet regulatory and safety requirements.

A key aspect of self-driving car proliferation will be taking robotics research and making it usable in real-world production environments, and especially on low-cost platforms. In the case of small autonomous vehicles, the question of sufficient computational resources becomes an even greater concern. Many hobbyists have adopted the Raspberry Pi as their microcontroller of choice. It offers a good amount of computational power and a large amount of General Input Output Pins (GPIO) in order to interface with various sensors at an exceptional price.

Our goal in this project is to take traditional machine learning techniques used in autonomous navigation—specifically image classification—and optimize them for a performance-constrained environment like the Raspberry Pi.

2 Theory

One of the key challenges in building fully autonomous self-driving cars is being able to identify and track objects near the vehicle. Before the modern renaissance of deep learning, the car would have to extract shapes from images and then look up those shapes in a large database of objects in order to identify them. This approach was slow and thus carried significant risk due to the quick reaction time needed to prevent collisions [13].

The convolutional neural network (CNN) is now the standard deep learning approach to object detection. Pioneered by LeCun et al. in 1995 with the LeNet architecture, CNNs convolve adjacent regions in space to pick out local features and then pool or sub-sample to reduce the effective image size and prevent over-fitting [7]. This approach, applied repeatedly over successive layers, significantly reduces the size of the network while allowing its depth and performance to grow [1]. In 2012, AlexNet refined this paradigm by adding multiple convolution layers between pooling layers, allowing the CNN to learn richer spatial features at every scale [6]. AlexNet won the 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC) by a large margin, leading to the widespread application of CNNs to computer vision tasks we see today.

Since then, innovations on this basic conv-pool architecture have substantially reduced the number of neurons, cost to train, and cost to predict, while maintaining or increasing accuracy. The first major development was the Inception architecture used by GoogLeNet, which adds so-called Inception modules that explicitly look at cross-channel and spatial correlations separately, rather than combining them as in traditional convolution layers [16]. This simplification allowed the number of parameters to drop from AlexNet's 60 million to only 4 million. Inception was followed by Xception, which despite having the same number of parameters, significantly outperforms InceptionV3 on a data set of 350 million images and 17,000 classes, which is much larger and more general than the ImageNet data set [3]. Finally, the recent ResNet uses a residual learning framework to efficiently train deeper networks than were previously feasible, which allowed it to win the 2015 ILSVRC with accuracy levels on par with a human [5].

However, in production contexts, we wish to do more than simply classify images. Rather, we wish to identify objects within (potentially multiple) regions of the image. Luckily, the aforementioned developments have increased efficiency and accuracy of CNNs to the point where applying them to real-time object detection is feasible. One approach is called regions with CNN features (R-CNN) [4, 10]. R-CNN identifies promising regions to perform image recognition, and returns the bounding box and label for regions where the CNN is sufficiently confident in its classification.

In contrast to R-CNN, a competing state-of-the-art object detection network is the you-only-look-once (YOLO) architecture, which on a Titan X GPU "processes images at 40-90 FPS" [9, 14]. Rather than segmenting the image and "looking" at each individual region with a CNN, it makes multiple predictions by running the entire image through a *single* network evaluation. Because it looks at the entire image once instead of performing segmentation, YOLO makes less than half of background errors than Fast R-CNN [14]. YOLO has accrued a significant following and was even featured in a TED talk [13]. It was originally built upon the Darknet open source neural network framework—also maintained by the authors of YOLO. The original paper provides ample supporting resources for getting YOLO working, including multiple pre-trained models, code examples, and training images [12]. We propose building and training our own YOLO network to recognize images relevant to an autonomous vehicle.

3 Data

We will use data from one or several available repositories to train our resource-constrained YOLO network. Udacity has a self-driving car repository with several hours of driving data including over 80,000 frames of annotated images [17]. A separate repository is even devoted to scripts for easily reading the Udacity data [18]. The YOLO project also has several images and the Darknet site hosts data from the Pascal Visual Object Classes Challenges that were used to train YOLO [11].

4 Implementation

We plan to test our algorithm on the Raspberry Pi 3, a small, cheap, and powerful microcontroller. It has a 1.2 GHz quad-core 64-bit ARM CPU, a 400Mhz VideoCore IV GPU, 1GB of SDRAM, on-board 802.11n WiFi and Bluetooth 4.0 [15]. It runs Raspbian which is a special flavor of Debian specifically for the Raspberry Pi. It also has a very high throughput camera bridge which can be used for streaming video from the Pi camera [2]. It supports 1080p video at 30fps or 720p video at 60fps via Video 4 Linux which is an API for real-time video capture on Linux devices.

These components make the Raspberry Pi perfect for implementing a real-time object detection system. A high throughput camera system ensures that we can test in real time without being bottle-necked by hardware issues. Our primary concern will instead be simplifying the algorithm enough that we can get good performance considering the CPU and GPU limitations. Since we have access to a couple Raspberry Pi's, we will be able to test our algorithm quite easily on the platform and figure out what kind of optimization's we can do to make it run faster. In addition, we can mount it on a real vehicle to verify the performance on the road. If real-time object detection performs well on this cost-effective device, others can use our knowledge to scale YOLO to other resource-limited application areas.

5 Conclusion

Overall, we are excited by the opportunities that this project offers. We will get to learn more about the state of the art in image classification and how to optimize some of these algorithms to make them accessible and able to run on low-cost computing platforms.

References

[1] Stanford CS 231 Course Webpage. URL http://cs231n.github.io/convolutional-networks/.

[2] Camera module v2. URL https://www.raspberrypi.org/products/camera-module-v2/.

[3] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. URL http://arxiv.org/abs/1610.02357.

[4] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[7] Y. Lecun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. *Comparison of learning algorithms for handwritten digit recognition*, pages 53–60. EC2 Cie, 1995.

[8] S. LeVine. What it really costs to turn a car into a self-driving vehicle. 2017. URL https://qz.com/924212/what-it-really-costs-to-turn-a-car-into-a-self-driving-vehicle/.

[9] Nvidia. Titan x. URL https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/.

[10] D. Parthasarathy. A brief history of cnns in image segmentation: From r-cnn to mask r-cnn. URL https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4.

[11] Pascal. Visual object classification challenge dataset. URL https://pjreddie.com/projects/pascal-voc-dataset-mirror/.

[12] J. Redmon. Darknet. URL https://github.com/pjreddie/darknet.

[13] J. Redmon. How computers learn to recognize objects instantly, 2017. URL https://www.ted.com/talks/joseph_redmon_how_a_computer_learns_to_recognize_objects_instantly#t-445892.

[14] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. URL http://arxiv.org/abs/1612.08242.

[15] Seralo. Raspberrypi models comparison. 2018. URL http://socialcompare.com/en/comparison/raspberrypi-models-comparison.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

[17] Udacity. Self driving car repository. URL https://github.com/udacity/self-driving-car/tree/master/datasets.

[18] R. Wightman. Udacity driving reader. URL https://github.com/rwightman/udacity-driving-reader.