YoloFlow

Konstantine Buhler
buhler@stanford.edu

John Lambert
johnwl@stanford.edu

Matthew Vilim
mvilim@stanford.edu

ABSTRACT

Todo

1. INTRODUCTION

Todo

2. OVERVIEW

2.1 Basics of YOLO

2.2 Motivation

3. TENSORFLOW IMPLEMENTATION

YOLO is implemented as a 32 layer deep convolutional neural network (DNN). The open source implementation released along with the paper is built upon a custom DNN framework called *darknet* [1]. This application provides the baseline by which we compare our implementation of YOLO. Redmon et al. have released several variants of YOLO. For our purposes, we chose the variant outlined in [1]. In places in which the paper lacks details, we refer to the baseline *darknet* implementation to resolve ambiguities.

The TensorFlow implementation was performed in two stages. First, we constructed the dataflow graph, importing pre-trained weights from *darknet*. With the pre-trained dataset, inference may be performed easily without need to create the loss function used during training. Our dataflow graph implementation is shown in Figure [TODO]. This graphical representation of the model was generated by TensorBoard, an included profiling tool that can be used to understand and visualize TensorFlow's execution.

3.1 Weight conversion

To simplify the port of YOLO from *darknet* to TensorFlow, we begin by using a pre-trained model. This greatly simplifies debugging as we can obtain immediate results from our execution without the need to implement a loss function and back propogation to adjust the weights. Furthermore, training a new model is naturally slow. For example, our pre-trained model was trained by [1] for a week on the latest high-performance GPUs.

However, choosing a pre-trained model is disadvantageous in that we must interface with a data format chosen by the model's author. In this case, we select one of YOLO's pre-trained models, *yolo_small.weights* provided on the author's website [2]. These weights are represented as a large binary blob of 32 bit floating points for each layer's weight/bias values. We modify the source of *darknet* to export the weights in a portable machine-independent format convenient for use in TensorFlow, comma-separated values (CSV). Parsing these large weight files (approximately 1GB) takes several minutes, so after they have been initialized in TensorFlow, we save a compressed TensorFlow *.ckpt* file for quick reload (available for download our project's Github page).

3.2 Model validation

As the original YOLO paper serves only a summary of YOLO, it omits many details. Details of the 2D convolution parameters and connection between convolutional and fully connected layers were taken from the *darknet* implementation. Validation and debug is drastically simplified in comparison to developing a new neural network model. Each input and output of our model can be compared with that of the *darknet* implementation to ensure correctness or discover bugs. For example, during development the weights were being read in an improper sequence at a particular convolutional layer. Comparing the output of the two network's allowed the problem to quickly narrowed down and corrected.

For this checkpoint, we have validated the outputs of both networks match on the pre-trained model for the test image shown in Figure 1. However, we have yet to parse the prediction output in order to calculate and draw bounding boxes. As in *darknet*, the inputs are processed and resized with OpenCV to be $(448, 448)$ with three normalized color channels.

4. REFERENCES

[1] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[1]github.com/pjreddie/darknet

[2]pjreddie.com/darknet/yolo/

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

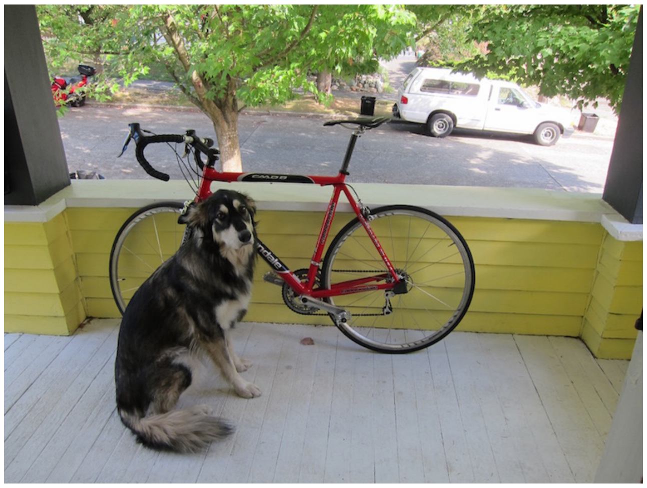

Figure 1: Test image