Robot Pose Estimation Using YOLO

Author:
Yunjie KE
Tsinghua University
kyj14@mails.tsinghua.edu.cn

Supervisor:
Prof. David TOURETZKY
Carnegie Mellon University
dst@cs.cmu.edu

September 16, 2017

This research was supported in part by an education grant from Google Cloud Platform to Carnegie Mellon University.

Contents

Abstract

We build an end-to-end robot pose estimation system by using an object detection network. We also introduce our data auto-collection method. The system has been tested in real environment and shows high accuracy and robustness.

1 Introduction

Multi-robot coordination and collaboration requires the study of robot vision to understand the pose and position of other robots. Considering a colony of robots that interact with each other in various ways, such as cooperating to accomplish a common goal, or competing against each other (perhaps in teams), or just coexisting in a shared environment without getting in each other's way.

These cooperating robots have many uses, e.g., it may take multiple robots to move a large object. Or one robot could provide guidance for others if they are trying to push an object around a corner or through a complex passage. Or robots could cooperate by dividing up a large task into smaller subtasks, e.g., each robot picks up and moves one object from a large pile that needs to be transported somewhere.

In order to get robots to cooperate this way, they need to be aware of each other's positions in the world. The Cozmo robot has a vision system that can recognize only a few things: the light cubes that come with it, human faces, and the faces of dogs and cats. A Cozmo robot cannot detect the presence of another Cozmo robot, much less derive its pose from a camera image. It's a hard problem because there are no markers on the Cozmo robot like there are on the light cubes. We have to look for more subtle features.

Here we propose a method for estimating a robot's position and orientation through the use of a convolution neural network.

The rest of the paper is structured as follows. Our approach and system architecture are introduced in Section 2. Section 3 presents our experiments and performance. Finally, Section 4 concludes our work.

2 Method

Our system localizes the robot using an object detection network called YOLO(Redmon et al., 2016). With the information of the bounding box given by localization, we can crop the image contained in the bounding box. Then we utilize another convolution neural network to estimate the orientation of the robot. Finally, with the coordinates of the bounding box and orientation, we train a random forest regressor to estimate the distance and bearing angle of the robot. Figure 1 shows our system pipeline.

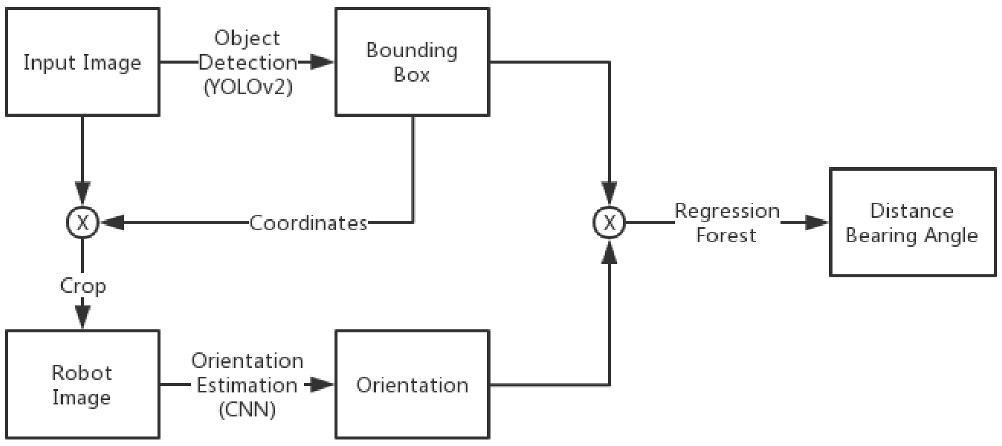

Figure 1: System Pipeline

2.1 YOLO (You-only-look-once)

YOLO is a state-of-the-art object detection approach. Compared to R-CNN methods, YOLO learns very general representations of objects and detects them much more quickly (real time speed). It sees the entire image during training stage without an RPN (region proposal network) or sliding window. Figure 2 shows the architecture of the YOLO convolution neural network part.

YOLO is really simple and clean. It tests the object detection problem as a pure regression task, giving bounding box coordinates (center coordinates, width, and height) and class probabilities directly from the input image.

However, while YOLO can find objects in an image quickly, sometimes it fails to localize really tiny objects and predict suitable, tight bounding boxes.

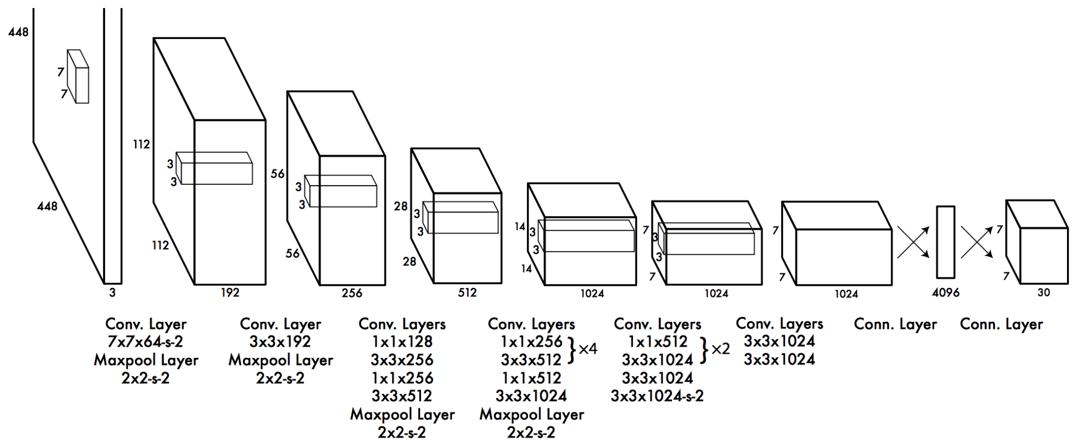

Figure 2: YOLO Architecture

The YOLO approach divides the image into an S × S grid and predicts B bounding boxes and corresponding confidence scores per grid cell. The loss function of YOLO during the training stage is as follows:

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(x_i-\hat{x_i})^2+(y_i-\hat{y_i})^2]$$

$$+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2]$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}(C_i-\hat{C}_i)^2$$

$$+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{noobj}(C_i-\hat{C}_i)^2$$

$$+\sum_{i=0}^{S^2}1_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2$$

where x_i, y_i, w_i, h_i, C_i denote the center coordinates, width, height and confidence score of the bounding box, hatted variables denotes YOLO's prediction.

YOLO decreases the loss from confidence score for identification and increase the loss from bounding box prediction. So it uses two parameters, λ_{noobj} and

λ_{coord} to discriminate this. In the original paper, the author set $\lambda_{noobj} = 0.5$ and $\lambda_{coord} = 5$, the same as our experiments.

2.2 Robot Orientation Encode (Circular Mean)

In mathematics, a mean of circular quantities is a mean which is sometimes better-suited for quantities like angles, daytimes, and fractional parts of real numbers. This is necessary since most of the usual means may not be appropriate on circular quantities. For example, the arithmetic mean of 0° and 360° is 180°, which is misleading because for most purposes 360° is the same thing as 0° (Bishop, 2006).

For our robot orientation convolution neural network, we utilize a method called circular mean to encode the original orientation. We convert all angles to corresponding points on the unit circle.

We plan to estimate the robot orientation $\theta \in [0, 2\pi)$. We find a set of angles $\phi_i = \frac{2\pi i}{T}, i \in \{0, 1, ..., T-1\}$. In our work, we set $T = 8$. Let

$$k_i = \cos(\phi_i - \theta)$$

So we encode the orientation as a vector which has continuous value like polar coordinate system and is preferred by regressor.

During prediction, we can decode the output of our regressor,

$$\hat{\theta} = atan2(\sum_{i=0}^{T-1} \sin \phi_i \hat{k_i}, \sum_{i=0}^{T-1} \cos \phi_i \hat{k_i})$$

2.3 Regression Forest

Random forest (Kam, 1995) is a common approach based on decision trees for solving classification problems in machine learning. Regression forest is similar to random forest but is based on a regression tree.

A general decision tree splits the data into subsets that contain similar instances with respect to one value. On the other hand, for a regression model, considering its value is a real value, we have to fit it to the target value using its features (or variables). Then for each feature, the data is partitioned at several points. We calculate the MSE (Mean Squared Error) between the actual value and the predicted value. Then we select the variable resulting in minimum MSE. The partition process will be recursively continued till the stopping requirements are met.

3 Experiments

3.1 Data Collection

Since there was no existing dataset for our robots, we recorded the pose and position of the robot by multi-robot controll and use the following techniques to expand the dataset and increase the robustness of regressors. Our dataset contains 1398 robot images and 56 background images.

3.1.1 Background Replacement

The background during the test phase could be complex and diverse, however, it is not easy to capture data in so many different backgrounds. So we try to replace the background.

We noticed that our robot only consists of red, white and black colors. Considering RGB channels, if we use pure green as the background in data collection, we are able to separate the robot and the background. Figure 3 shows an example.

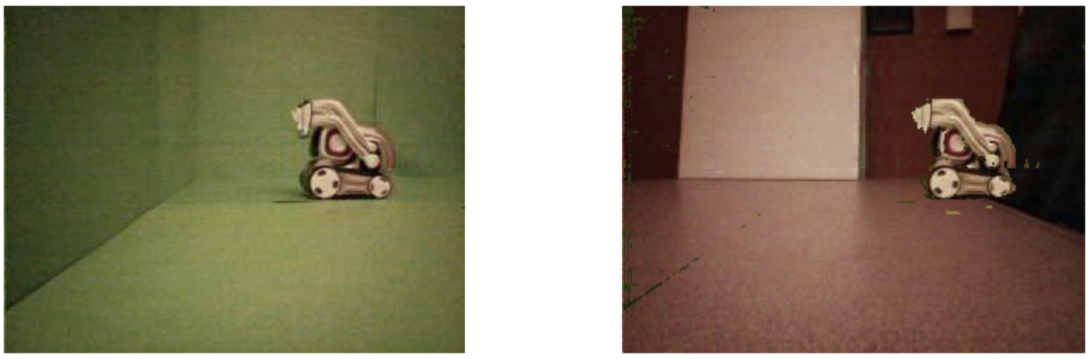

Figure 3: background replacement example

We captured many images of the robot against a green background, We then used the algorithm above to expand the dataset. It greatly increases the amount of data, and the variety of scenes should increase the robustness of the neural network.

3.1.2 Bounding Box Generation

We noticed that we need to label the bounding box of the robot in each image. But manual annotation is also tedious. We proposed an algorithm, using the background replacement algorithm, we can also calculate the mask matrix of the image to distinguish the foreground and background. Though the matrix still contains a lot of noise, our algorithm can overcome the problem. First, we calculate a point as a rough estimation of the bounding box. Then we use the greedy method to dynamically increase the bounding box. Figure 4 shows an example.

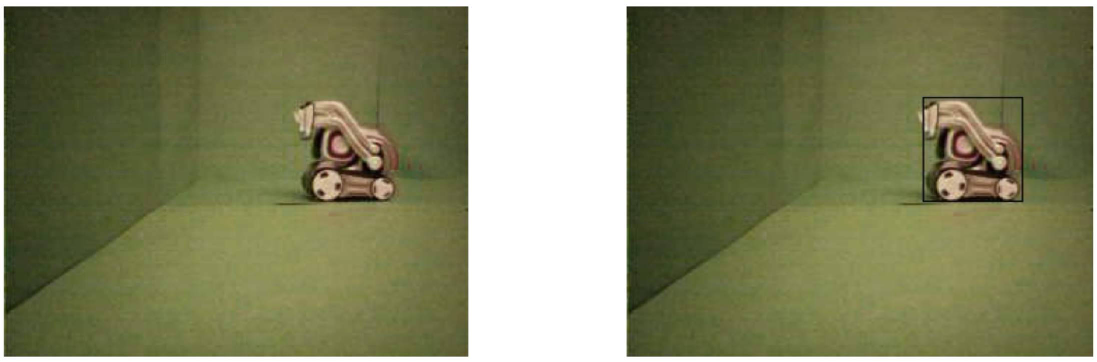

Figure 4: bounding box generation example

3.2 Training Methods and Parameters

For object detection network, we choose the YOLOv2 (Redmon and Farhadi, 2016) trained on VOC2007 (Everingham et al., 2007) and VOC2012 (Everingham et al., 2012) as our initial pre-trained model. We only modify the output units and fine-tune its top layer. We choose Adam (Kingma and Ba, 2014) as our optimizer. It converges after 150 epochs.

For orientation estimation CNN, we adopt the following pipeline in figure 5 and use optimizer Adam. It converges after 200 epochs.

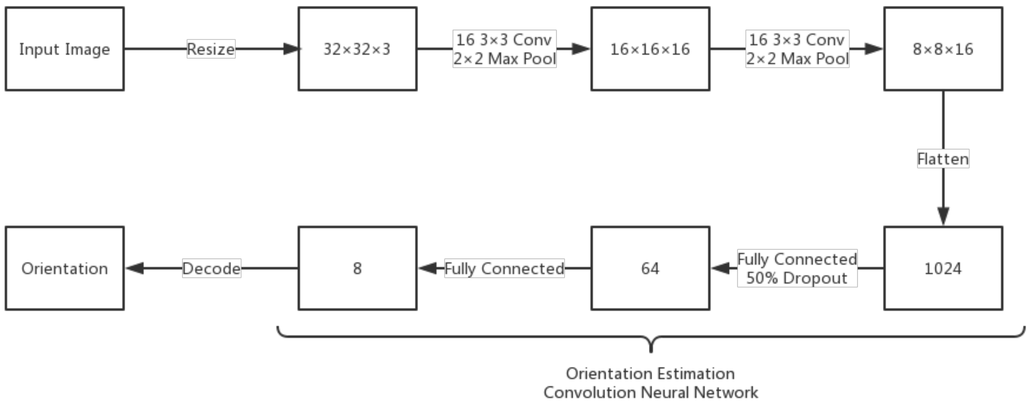

Figure 5: orientation estimation pipeline

For distance and angle estimation regressor, we use a regression forest with 100 base regression trees and set the minimum samples split as 3.

3.3 Results

We test our result using hold-out validation (90% for training and 10% for validating).

We trained our YOLO. Figure 6 shows some robot detection results on the validation dataset. From these images, we can infer that YOLO is able to give accurate and tight bounding box as our robots.

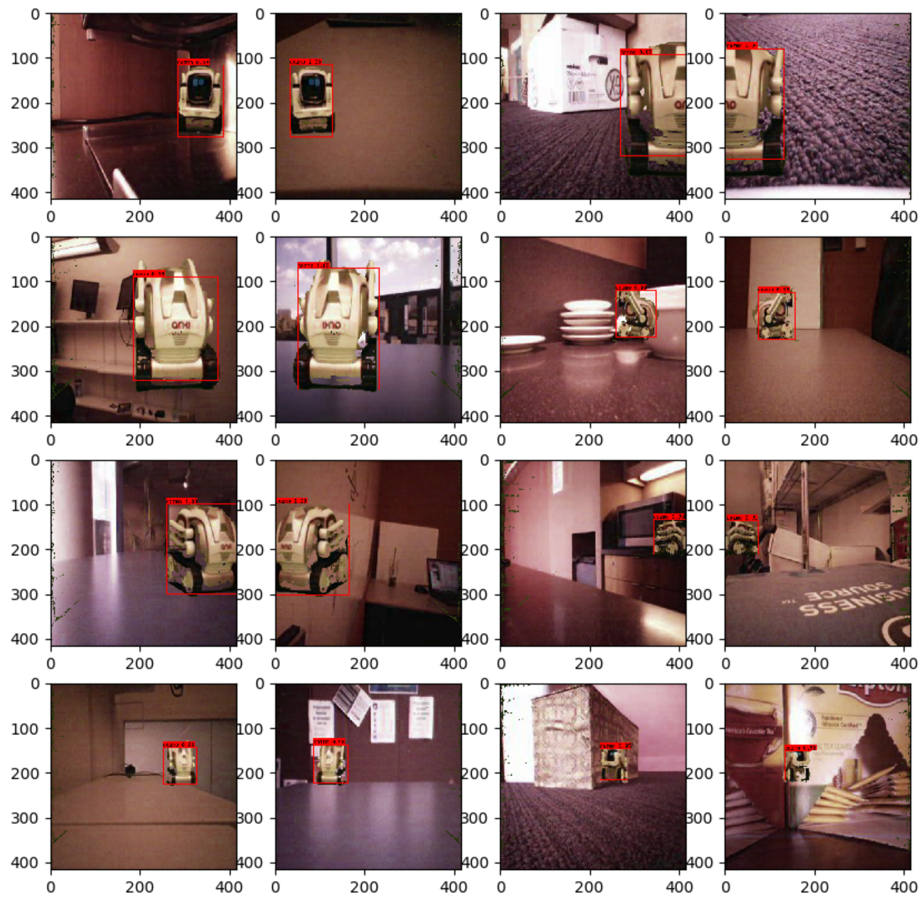

Figure 6: robot detection result

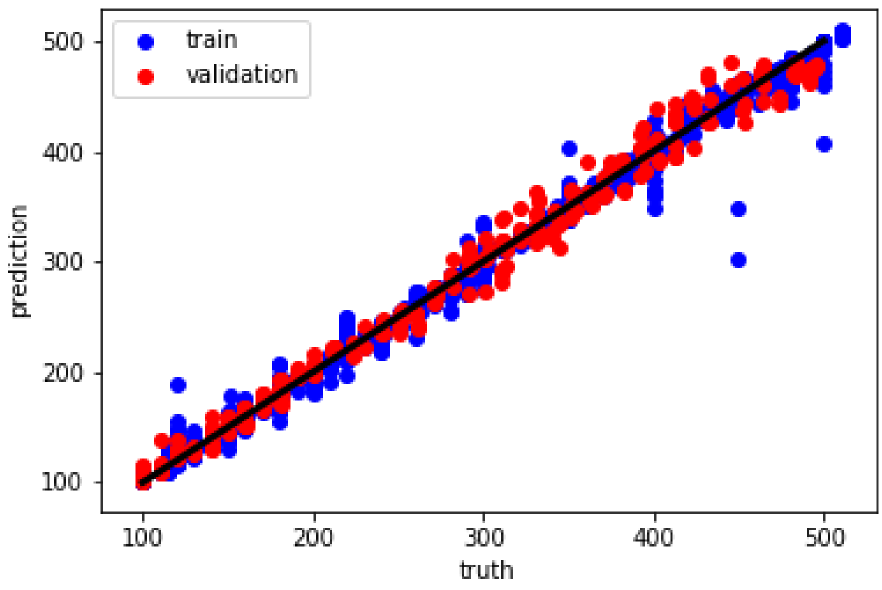

Figure 7: distance estimation result

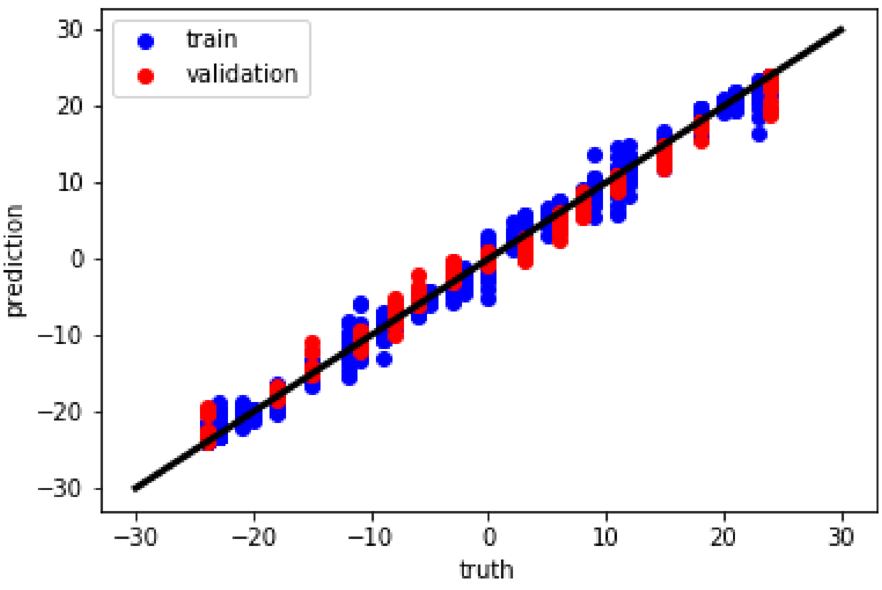

Figure 8: angle estimation result

Figure 7, 8 and 9 shows the results of our final pose estimation. The points' coordinates represent the ground truth and our estimation. When the points fit the black line more closely, it suggests a better results. We have to point out that in Figure 9, there are some points seemingly deviate from the fitting line seriously. However, $-180°$ is the same thing as $180°$. So actually those 'deviating' points fit the line pretty well.

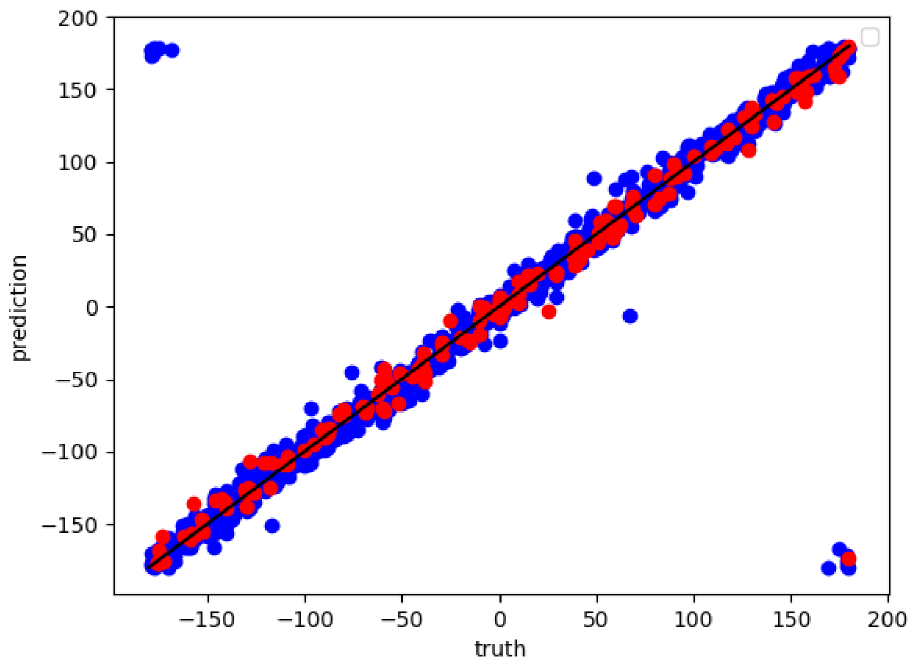

Figure 9: orientation estimation result

3.4 Performance

To test our pose estimation system in the real environment, we implemented a simple robot coordinate application. The robot uses its vision to locate another robot and tries to bring them face to face. We record the video as our supplementary material.

4 Conclusion

We built a robust and accurate robot pose and position estimation model using object detection neural network. We also proposed our methods including data collection, augment, and auto-annotation. Our model performs with high accuracy and robustness in a complex environment.

References

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Kam, H. T. (1995). Random decision forest. In *Proc. of the 3rd Int'l Conf. on Document Analysis and Recognition, Montreal, Canada, August*, pages 14–18.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788.

Redmon, J. and Farhadi, A. (2016). Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*.