

---

# Object Detection and Orientation Estimation for Autonomous Driving

---

**Jinyi Lu**

Information Networking Institute  
jinyil@andrew.cmu.edu

**Xiaoqing Tao**

Information Networking Institute  
xtao@andrew.cmu.edu

**Keywords:** [computer vision, object detection, object orientation estimation, autonomous driving]

## 1 Introduction

### 1.1 Problem Overview

Nowadays cameras are available onboard of almost every new car produced in the last few years. Computer vision provides a very cost effective solution not only to improve safety, but also to one of the holy grails of AI, fully autonomous self-driving cars. In this project we are planning to use deep neural networks to solve the object detection and object orientation estimation problems for autonomous driving.

There are lots of potential challenges that we need to solve for example, due to the weather, road conditions and car location, images from the car cameras will have a high-variety, which requires high robustness for our model. And besides the classical object detection task, we want to further estimate the 3D orientation from the 2D images. Last, but not least our system need to produce a good result within a limited runtime in order to be used in practice.

### 1.2 Dataset

The dataset that we are planning to use is the KITTI Vision Benchmark Suite [1]. It's developed for use in mobile robotics and autonomous driving research. So it contains several novel challenging benchmarks for the tasks of stereo, optical flow, visual odometry/SLAM and 3D object detection. In our project, we mainly focus on the object detection and orientation estimation task. The corresponding benchmark <sup>1</sup> consists of 7481 training images and 7518 test images, comprising a total of 80,256 labeled objects (up to 15 cars and 30 pedestrians are visible per image). All images are color and saved as png.

Object	Avg Number of Object
Car	3.8429
Pedestrians	0.5998
Cyclists	0.2175

Table 1: Average Number of Objects Per Image

### 1.3 Evaluation

For evaluation, the benchmark is split into three parts: First, we need to evaluate the classical 2D object detection by measuring performance using the well established average precision (AP) metric as described in [2]. Detections are iteratively assigned to ground truth labels starting with the largest

---

<sup>1</sup>[http://www.cvlibs.net/datasets/kitti/eval\\_object.php](http://www.cvlibs.net/datasets/kitti/eval_object.php)



Figure 1: Some Ground Truth Examples

overlap, measured by bounding box intersection over union. True positives are required to overlap by more than 50% and multiple detections of the same object are counted as false positives.

Second, we assess the performance of jointly detecting objects and estimating their 3D orientation using a novel measure which is called the average orientation similarity (AOS) [1] and is defined as:

$$AOC = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} \max_{\tilde{r}: \tilde{r} \geq r} s(\tilde{r}) \quad (1)$$

Here,  $r = \frac{TP}{TP+FN}$  is the PASCAL object detection recall, where detected 2D bounding boxes are correct if they overlap by at least 50% with a ground truth bounding box. The orientation similarity  $s \in [0, 1]$  at recall  $r$  is a normalized ( $[0..1]$ ) variant of the cosine similarity defined as

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + \cos \Delta_{\theta}^{(i)}}{2} \delta_i \quad (2)$$

where  $D(r)$  denotes the set of all object detections at recall rate  $r$  and  $\Delta_{\theta}^{(i)}$  is the difference in angle between estimated and ground truth orientation of detection  $i$ . To penalize multiple detections which explain a single object, we set  $\delta_i = 1$  if detection  $i$  has been assigned to a ground truth bounding box (overlaps by at least 50%) and  $\delta_i = 0$  if it has not been assigned.

Finally, we will also evaluate pure classification (16 bins for cars) and regression (continuous orientation) performance on the task of 3D object orientation estimation in terms of orientation similarity.

## 1.4 Related work

Traditional methods for object detection usually utilize image features, such as SIFT and HOG. We investigated three methods utilizing deep convolutional network in object detection.

### 1.4.1 R-CNN: Rich feature hierarchies for accurate object detection and semantic segmentation

The first challenge for object detection is how to implement localizing within an image. The new method proposed in this paper [3] combines CNN with region proposals. So it is called R-CNN.

The general detection process is to extract about 2000 region proposals for each input image. Then utilize CNN to compute a fixed length feature for each region proposal. Finally utilize linear SVM to classify each region.

The second challenge is how to train a large CNN using limited labeled data. This paper proposed to pre-train CNN on a large auxiliary data set, then continually train on a small data set. This method significantly improved the accuracy of objection detection compared with feature learning models. But training is expensive in space and time, and detection is also slow at test time.

### 1.4.2 Fast R-CNN

This paper[4] talked about how to train a detection network faster. R-CNN is very slow because it extracts feature for each region proposal and there are many duplicate computations. The process could be faster if we share computation.

This paper proposed to modify R-CNN's architecture by taking an image and multiple regions of interests as input. Region proposal method usually depends on Selective Search. Each region of interest is pooled into a fixed-size feature map and fully connected layers are used to extract features. There two output vectors: softmax probabilities and per-class bounding-box regression offsets. The second one is to reduce mislocalization.

## 2 Methods

### 2.1 You Only Look Once (yolo)

Yolo is a real-time object recognition algorithm [5]. Unlike sliding window and region proposal-based techniques, it trains on full images and reframes object detection as a single regression problem. The yolo system firstly resizes images to 448\*448, then runs a single convolutional network, and predicts multiple bounding boxes and class probabilities for those boxes. It only looks at images once, unlike R-CNN which require thousands for a single image. This makes yolo extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

Yolo system divides an image into  $S \times S$  grid. Each grid cell predicts  $B$  bounding boxes and  $C$  conditional class probabilities. Each bounding box contains 5 predictions:  $x$ ,  $y$ ,  $w$ ,  $h$ , and confidence score. So the final prediction is a  $S \times S \times (B * 5 + C)$  tensor.

Yolo network has 24 convolutional layers followed by 2 fully connected layers. The initial convolutional layers extract features from the image while the fully connected layers predict the output probabilities and coordinates.

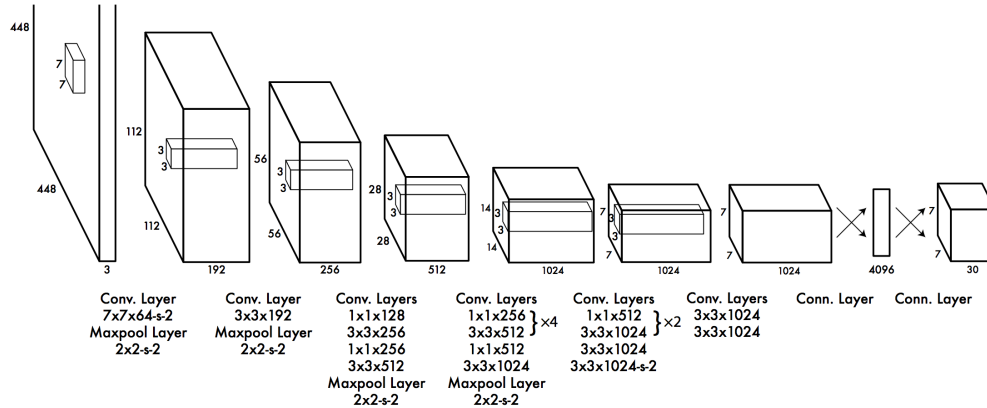


Figure 2: Yolo Network Architecture

## 2.2 Faster R-CNN

Faster R-CNN is an elegant and effective solution for object detection. It is an end-to-end object detection framework using a Region Proposal Network (RPN) and an Object Detection Network. Previous methods such as SPPnet and Fast R-CNN, rely on proposal algorithms to hypothesize object locations, which becomes a bottleneck in order to reducing the running time. In Faster R-CNN, we don't need to pre-generated several region proposals using some inexpensive features and economical inference schemes such as Selective Search. Instead, it has a RPN on top of the convolutional features, which can be used for generating detection proposals and be trained end-to-end.

A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score. Such process is modeled by a fully convolution network [6]. The region proposals are generated by the box-regression layer and box-classification layer using the features from a small network sliding over the convolutional feature map output.

After generating a set of object proposals using RPNs, Fast R-CNN can be using to make detection. And since RPN and Fast R-CNN all rely on some convolutional neural networks, Faster R-CNN make them share some convolutional layers to make it converge quickly.

## 3 Preliminary Results

- Sampling
- Converge speed
- Network size
- Boxes

In this section, we will discuss some of our preliminary results. We first randomly split all 7841 labeled images into two parts: training set (4953 images) and validation set (2538 images). For now, we have conducted three experiments on object detection, including using a pre-trained yolo model, using a fine-tuned tiny yolo model based on our training set and using a pre-trained Faster-RCNN model. All the experiments are conducted on AWS using a g2.2xlarge instance with GRID K520 GPU. We directly use the Yolo implementation from Darknet <sup>2</sup> and the Faster R-CNN python implementation <sup>3</sup>.

In terms of evaluating the object detection results, KITTI benchmark requires a minimum overlap of 70% for cars and 50% for pedestrians. It also has three kinds of difficulties: easy, moderate and hard. Difficulties are defined as follows:

Difficulty	Min. bounding box height	Max. occlusion level	Max. truncation
Easy	40 Px	Fully visible	15%
Moderate	25 Px	Partly occluded	30%
Hard	25 Px	Difficult to see	50%

Table 2: Different Difficulties Requirements

The pre-trained yolo model is trained on Pascal VOC 2012 and 2007 dataset <sup>4</sup>. It supports about 20 object categories including car and people (corresponding to pedestrian). However it don't support cyclist category. So in all the following experiments, we only evaluate the performance for cars and pedestrians.

For now, we only detect cars and pedestrians in images. Considering the limitation of time and compute power, we choose the tiny yolo model to fast tune. Tiny yolo model is very similar to yolo model, but only contains 8 convolutional layers. We use convolutional weights that are pre-trained on ImageNet classification task as initial weights. Fine tuning tiny yolo model makes it possible to benefit from the features the model has learnt previously and speed up training. It took about 16 hours to train tiny yolo after 10,000 iterations.

<sup>2</sup><https://github.com/pjreddie/darknet.git>

<sup>3</sup><https://github.com/rbgirshick/py-faster-rcnn>

<sup>4</sup><http://pjreddie.com/darknet/yolo/>

The pre-trained Faster-RCNN model is trained on Pascal VOC 2007 dataset. It is able to detect cars and pedestrians and also lacks of supports for cyclists.

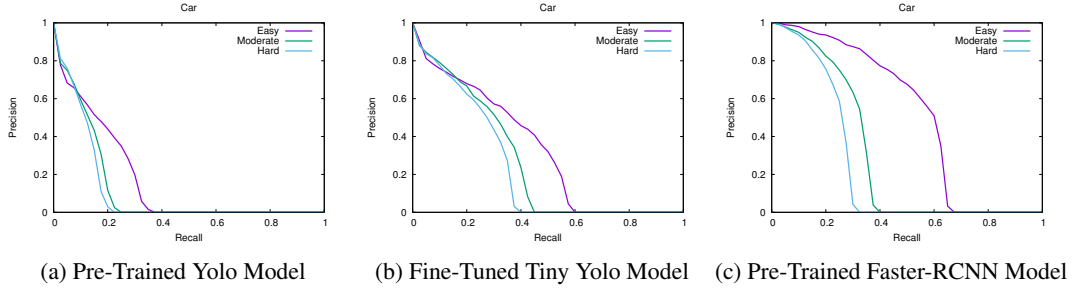


Figure 3: Car Detection

Method	Easy	Moderate	Hard
Pre-Trained Yolo	0.204438	0.155358	0.145101
Fine-Tuned Tiny Yolo	0.344293	0.291823	<b>0.257863</b>
Pre-Trained Faster-RCNN	<b>0.522754</b>	<b>0.309875</b>	0.248554

Table 3: Average Precision on Car Detection

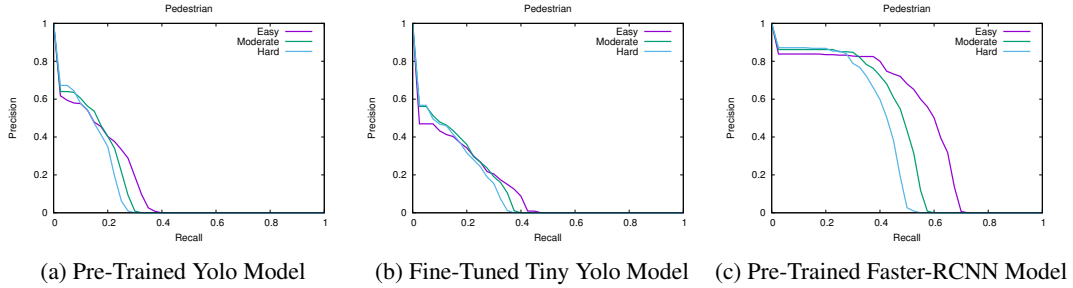


Figure 4: Pedestrian Detection

Method	Easy	Moderate	Hard
Pre-Trained Yolo	0.197457	0.183323	0.175022
Fine-Tuned Tiny Yolo	0.187898	0.184558	0.176187
Pre-Trained Faster-RCNN	<b>0.498992</b>	<b>0.429862</b>	<b>0.377569</b>

Table 4: Average Precision on Pedestrian Detection

As we can see from the precision-recall curve and the average precision results, for pedestrians, Faster-RCNN outperforms yolo models a lot in all three difficulty levels. And a fine-tuned tiny yolo model on our dataset shows no improvement compared with the pre-trained yolo model. For cars, Faster-RCNN outperforms yolo models in easy level. And our fine-tuned tiny yolo model shows great improvement in all three difficulty levels compared with the pre-trained yolo model. And in terms of predicting speed, the fine-tuned tiny yolo can achieve about 70 fps and the pre-trained Faster-RCNN can only achieve about 2 fps.

## 4 Discuss

## 5 Project Plan

In the next steps, we are planning to firstly conduct several experiments to figure out the reasons of Yolo poor performances on detecting pedestrians. Then we will build a fine-tuned Faster R-CNN

model for detecting cars, pedestrians and cyclists on a smaller dataset due to lack of computation power. We also will explore some possible methods for estimating 3D object orientation.

## References

- [1] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [2] M Everingham, L Van Gool, CKI Williams, J Winn, and A Zisserman. The pascal visual object classes challenge 2011 (voc 2011) results (2011). In URL <http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html>, 2010.
- [3] Trevor Darrell Jitendra Malik Ross Girshick, Jeff Donahue. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [4] Ross Girshick. Fast r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [5] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection. In *arXiv preprint arXiv:1506.02640*, 2015.
- [6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.