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 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 [1] 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

[1] http://www.cvlibs.net/datasets/kitti/eval_object.php

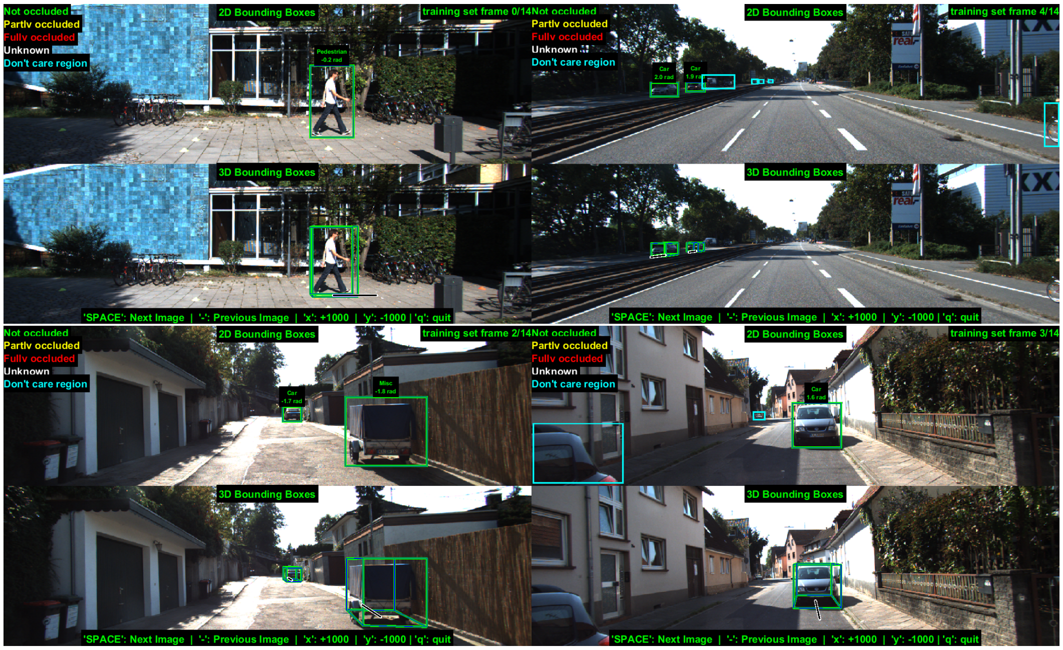


Figure 1: Some Ground Truth Examples

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:

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

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

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

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 × 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. (x, y) represents the center of the box relative to the bounds of the grid cell. (w, h) is the width and height predicted relative to the whole image. The confidence score for each score is defined as $Pr(Object) * IOU_{pred}^{truth}$. Conditional class probability is defined as $Pr(Class_i|Object)$. So the final prediction is a S × S × (B * 5 + C) tensor. At test time, the class-specific confidence score is

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \quad (3)$$

Yolo network architecture is inspired by the GoogLeNet model for image classification. It 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 both class probabilities and bounding box coordinates.

For training, the first 20 convolutional layers are pretrained on the ImageNet 1000-class dataset. Then 4 convolutional layers and 2 fully connected layers with randomly initialized weights are added.We assign only one bounding box predictor to be responsible for predicting object per grid cell. The predictor is chosen based on which prediction has the highest IOU between the current bounding box and ground truth. This leads to specialization between predictors. And the loss function is defined as the sum-squared error in the output of the model. The learning also fluctuates. It starts with a small value (0.001) to prevent the model from diverging due to unstable gradients. Then slowly raise the learning rate to 0.01 and continue training. After some time, decrease the learning rate again.

Yolo has some limitations. Firstly, each grid cell can only predict B (the default value is 2) bounding boxes. So the model have difficulties predicting small objects appear in group. Besides, the main of source of error is wrong localizations. The impact of a small error in a small bounding box should weigh more heavily than the impact of a small error in a large bounding. But they are same in the loss function of yolo.

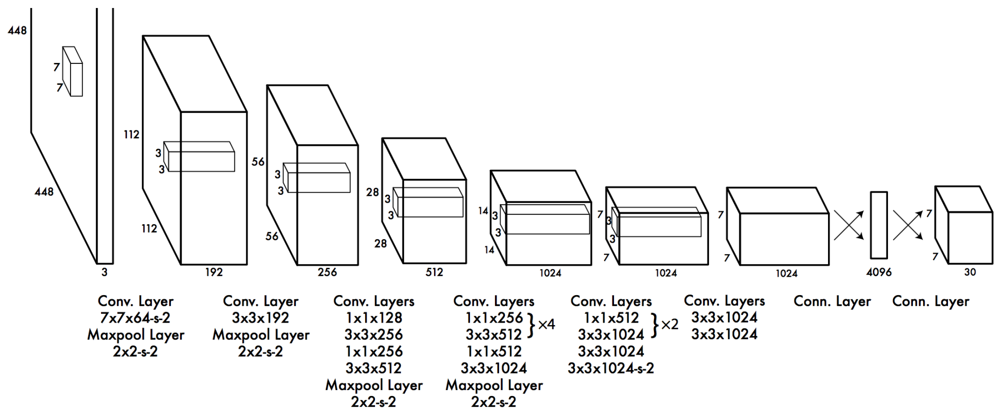


Figure 2: Yolo Network Architecture

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, reply 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.

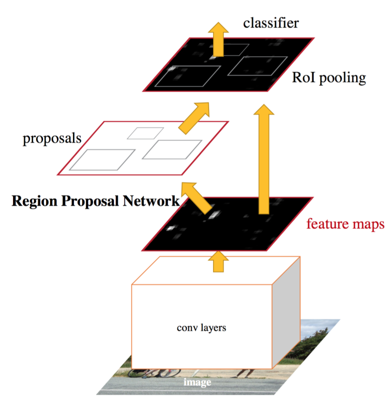


Figure 3: Faster R-CNN Network Architecture

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 for predicting the coordinates of the proposal and box-classification layer for classifying whether it's an object or non-object 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 using the generated proposals and convolutional feature map output. And since RPN and Fast R-CNN all rely on the same convolutional neural networks, Faster R-CNN make them share same convolutional layers to make it converge quickly. There are multiple choices of convolutional neural networks for generateing feature map, such as ZFnet, VGG, ResNets etc.

In terms of training, a carefully designed decresing policy for learning rate is used. For the first 60k mini-batches, a learning rate of 0.001 is used. And for the next 20k mini-batches 0.0001 is used.

3 Results

In this section, we will discuss some of our results. 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

Originally, we used all the data we have. However, since training on the whole set takes lots of time, we only work on a subset of all the data (about 30%).

3.1 Subset Sampling

We have 7841 labeled images in total with about 80,256 labeled objects. However, in order to speed up the development process, we choose to only use 30% of all the images we have (1637 images for training and 828 images for validation) and detect two kinds of object: car and pedestrians.

Object	Avg Number of Object (Whole Set)	Avg Number of Object (Subset)
Car	3.8429	3.8502
Pedestrians	0.5998	0.6630

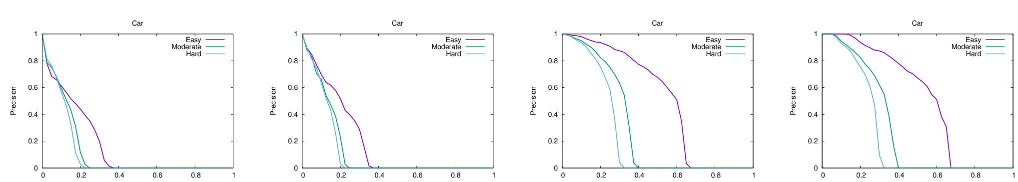
Table 3: Average Number of Objects Per Image

Previously, we have conducted two experiments on the whole dataset, including using a pre-trained yolo model, and using a pre-trained Faster-RCNN model. All the experiments are conducted on AWS using a p2.xlarge instance with Tesla K80 GPU. We directly use the Yolo implementation from Darknet [2] and the Faster R-CNN python implementation [3].

The pre-trained yolo model is trained on Pascal VOC 2012 and 2007 dataset [4]. 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.

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.

So next we will compare their performance on the subset with the previous results to see whether the subset sampling affects the model performance.



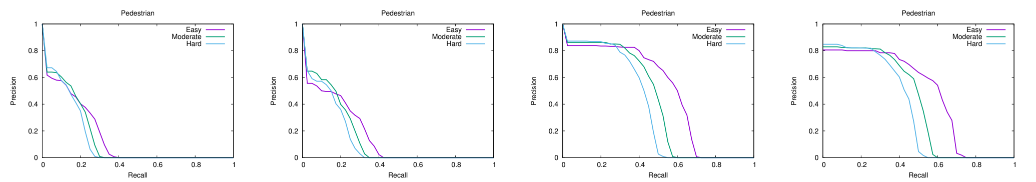
(a) Pre-Trained Yolo on Whole Set (b) Pre-Trained Yolo on Subset (c) Pre-Trained Faster-RCNN on Whole Set (d) Pre-Trained Faster-RCNN on Subset

Figure 4: Car Detection

[2] `https://github.com/pjreddie/darknet.git`
[3] `https://github.com/rbgirshick/py-faster-rcnn`
[4] `http://pjreddie.com/darknet/yolo/`

Method	Easy	Moderate	Hard
Pre-Trained Yolo Whole Set	0.204438	0.155358	0.145101
Pre-Trained Yolo Subet	0.227639	0.172312	0.151478
Pre-Trained Faster-RCNN Whole Set	0.522754	0.309875	0.248554
Pre-Trained Faster-RCNN Subet	0.524807	0.308296	0.252989

Table 4: Average Precision on Car Detection



(a) Pre-Trained Yolo on Whole Set (b) Pre-Trained Yolo on Subset (c) Pre-Trained Faster-RCNN on Whole Set (d) Pre-Trained Faster-RCNN on Subset

Figure 5: Pedestrian Detection

Method	Easy	Moderate	Hard
Pre-Trained Yolo Whole Set	0.197457	0.183323	0.175022
Pre-Trained Yolo Subet	0.207263	0.189966	0.178328
Pre-Trained Faster-RCNN Whole Set	0.498992	0.429862	0.377569
Pre-Trained Faster-RCNN Subet	0.467467	0.404372	0.356576

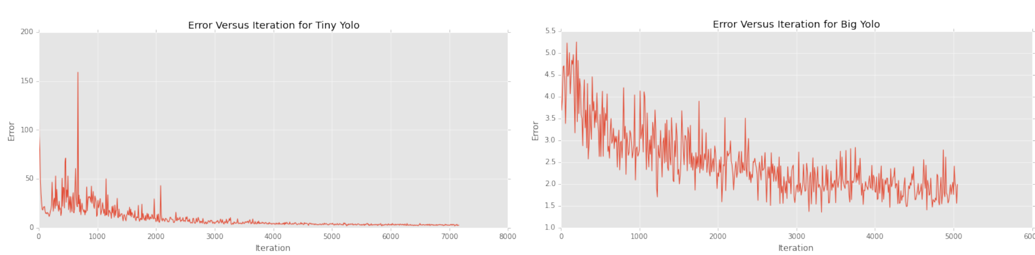
Table 5: Average Precision on Car Detection

As we can see, the performance of those two models doesnt' have significant difference on the subset compare with the whole set. So in all the following experiments, we only work on the subset.

3.2 Convergence Properties of Different Models

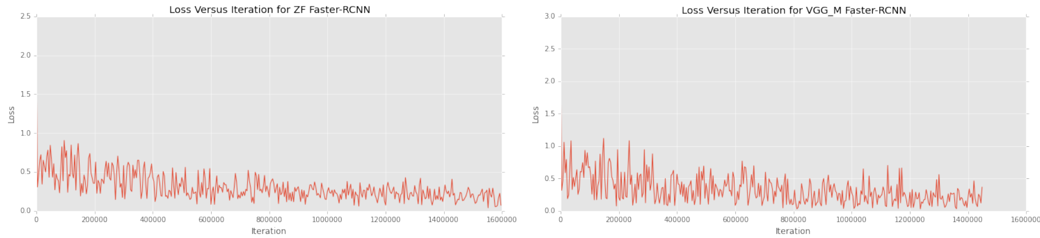
We build four fine-tuned models, which are tiny yolo, big yolo, Faster R-CNN using ZFnet and Faster-RCNN using VGG_CNN_M_1024 [7].

In this section we will show some results about their convergence properties by visualizing their loss/error changes with iterations.



(a) Tiny Yolo (b) Big Yolo

Figure 6: Convergence Properties of Yolo



(a) Faster R-CNN using ZFnet for Feature Map (b) Faster R-CNN using VGG_CNN_M_1024 for Feature Map

Figure 7: Convergence Properties of Faster R-CNN

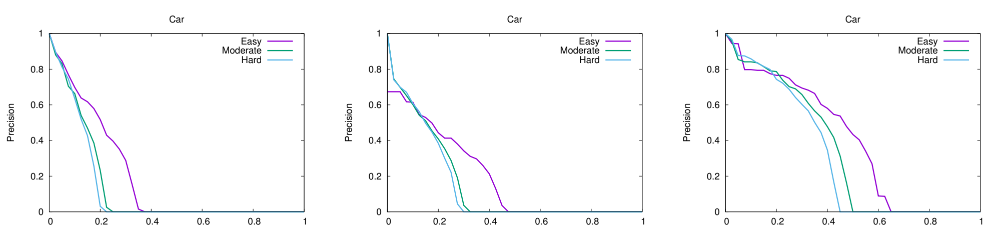
As we can see, all the models are very hard to converge even with their carefully designed adapting learning rates.

3.3 Fine-Tuned Models with Different Network Structure

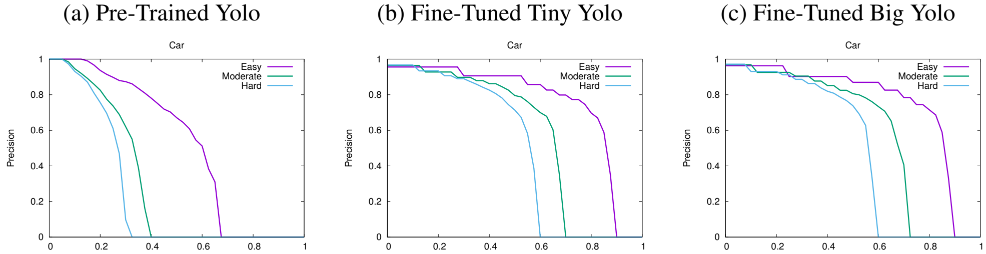
In this experiments, we test four different models which has different network structure. A tiny yolo model is a simplified version of the big yolo model by reducing 16 convolutional layers for generating features. And the difference between two Faster R-CNN models is using which convolutional networks for generating feature map, one is ZFnet [8] and the other one is VGG_CNN_M_1024 [7]. These two convolutional networks are quite similar except that VGG_CNN_M_1024 is wider than ZFnet.

We use convolutional weights that are pre-trained on ImageNet classification task as initial weights. Fine tuning models make it possible to benefit from the features the model has learnt previously and speed up training.

It took about 5 hours to train a tiny yolo after 5,000 iterations and 15 hours to train a big yolo after 5,000 iterations. For Faster R-CNN, the one using ZFnet took about 16 hours after 80,000 iterations and the one using VGG_CNN_M_1024 took about 12 hours after 80,000 iterations.



(a) Pre-Trained Yolo (b) Fine-Tuned Tiny Yolo (c) Fine-Tuned Big Yolo

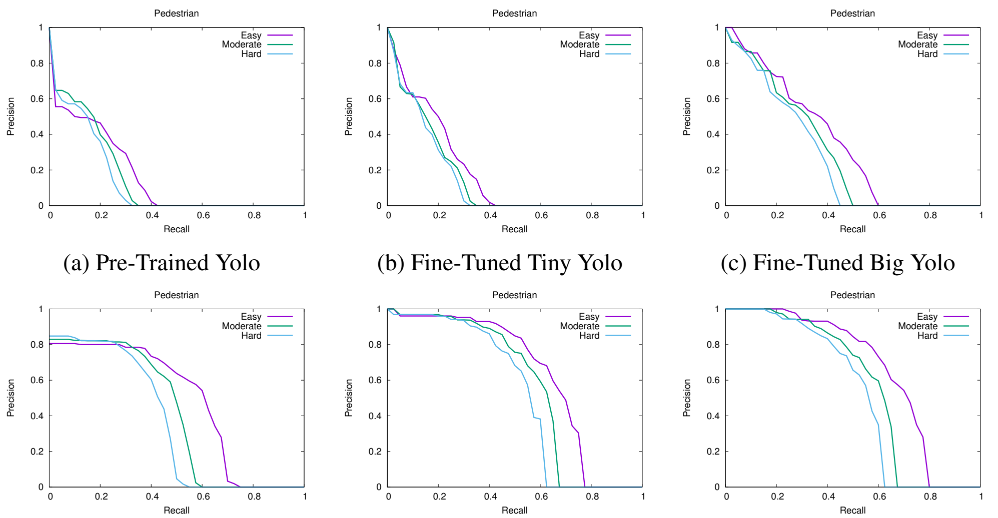


(d) Pre-Trained Faster R-CNN (e) Fine-Tuned Faster R-CNN using ZFnet (f) Fine-Tuned Faster R-CNN using VGG_CNN_M_1024

Figure 8: Car Detection

Method	Easy	Moderate	Hard
Pre-Trained Yolo	0.227639	0.172312	0.151478
Fine-Tuned Tiny Yolo	0.207844	0.186021	0.181117
Fine-Tuned Big Yolo	0.396407	0.342113	0.322760
Pre-Trained Faster R-CNN	0.524807	0.308296	0.252989
Fine-Tuned ZF Faster R-CNN	**0.721690**	0.555533	**0.481580**
Find-Tuned VGG_M Faster R-CNN	0.721039	**0.596753**	0.479783

Table 6: Average Precision on Car Detection



(a) Pre-Trained Yolo (b) Fine-Tuned Tiny Yolo (c) Fine-Tuned Big Yolo

(d) Pre-Trained Faster R-CNN (e) Fine-Tuned Faster R-CNN using ZFnet (f) Fine-Tuned Faster R-CNN using VGG_CNN_M_1024

Figure 9: Pedestrian Detection

Method	Easy	Moderate	Hard
Pre-Trained Yolo	0.207263	0.189966	0.178328
Fine-Tuned Tiny Yolo	0.214768	0.191912	0.179325
Fine-Tuned Big Yolo	0.351823	0.304227	0.284068
Pre-Trained Faster R-CNN	0.467467	0.404372	0.356576
Fine-Tuned ZF Faster R-CNN	0.621262	0.556017	**0.526138**
Find-Tuned VGG_M Faster R-CNN	**0.635249**	**0.556700**	0.520960

Table 7: Average Precision on Pedestrian Detection

3.4 Detected Object Bounding Boxes

In this section, we explore the detected object bounding boxes by different models by checking the average height, width and area of detected objects. For all the models, we only consider the detected object with confidence heigher than 0.2.

Method	Avg Height	Avg Width	Avg Area	cnt
Ground Truth	**66.6599**	**113.2698**	**11449.9983**	**3188**
Fine-Tuned Tiny Yolo	81.4113	130.9231	14518.8764	2601
Fine-Tuned Big Yolo	67.6324	117.1498	11676.8592	3029
Fine-Tuned ZF Faster R-CNN	63.7740	102.8394	10458.8683	3502
Find-Tuned VGG_M Faster R-CNN	62.7834	100.1970	9768.3991	3885

Table 8: Detected Car Bounding Boxes Comparison

Method	Avg Height	Avg Width	Avg Area	cnt
Ground Truth	**103.2774**	**43.3553**	**6081.0395**	**549**
Fine-Tuned Tiny Yolo	158.6333	69.7682	12327.1655	235
Fine-Tuned Big Yolo	132.4591	64.1588	9822.6585	225
Fine-Tuned ZF Faster R-CNN	105.9067	45.7684	6179.3364	585
Find-Tuned VGG_M Faster R-CNN	96.6618	44.1266	5529.7738	662

Table 9: Detected Precision Bounding Boxes Comparison

3.5 Prediction Speed

Method	FPS
Fine-Tuned Tiny Yolo	70
Fine-Tuned Big Yolo	14
Fine-Tuned ZF Faster R-CNN	4
Find-Tuned VGG_M Faster R-CNN	4

Table 10: Prediction Speed

4 Discuss

In terms of the model performance, we use the two pre-trained models as the baseline for comparsion. As we can see from the precision-recall curve and the average precision results, for cars, best result is achieved by using a Find-tuned Faster R-CNN models and there is no significant difference between which convolutional networks to be used for generating feature map probability because these two convolutional networks are very similar and have the same depth. In terms of yolo, a fine-tuned tiny yolo model don't have a significant improvement compare to the one using the pre-trained model. However, a fine-tuned big yolo mdoel do improve the performance a lot thanks to the deeper network. For pedestrian, the performance of models are quite similar, fine-tuned Faster R-CNN models achieve the best results. And a fine-tuned big yolo model improves the baseline while a fine-tuned tiny yolo model shows no significant improvement.

It seems hard to strike a balance between speed and accuracy of detection. Yolo is extremely fast on real-time detection because it only requires a single network evaluation. But the detection accuracy is not enough. Yolo only predicts 2 bounding boxes in each grid cell and 98 bounding boxes per image. This imposes strong spatial constraints to limit the number of nearby objects that the model can predict. Yolo performs poorly in detecting small objects, especially in groups. It also struggles to localize objects correctly. The impact of a small error in a small bounding box should weigh more heavily than the impact of a small error in a large bounding. But yolo treats them as the same.

Faster RCNN is another state-of-the-art object-detecting model. It is a high-accuracy detector, outperforming yolo. It benefits from explicit region proposals. But the VGG-CNN-M-1024 version of Faster R-CNN is about 4 times slower than YOLO.

In the future, we would like to increase the speed of real-time detection on KITTI dataset without sacrificing the detection accuracy. We may try other models such as SSD. It's fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature resampling stage. Other modifications include using a small convolutional filter to predict object categories and offsets in bounding box locations, using separate predictors (filters) for different aspect ratio detections, and applying these filters to multiple feature maps from the later stages of a network in order to perform detection at multiple scales. We hope SSD model can improve the detection on KITTI dataset.

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.

[7] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[8] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.