Performance Analysis of Modern Pedestrian and Vehicle Detection Approaches

Yichen Li and Suli Huang

***Abstract*— We are here to present a investigation of influences of different meta-architectures, feature extractors and network hyper parameters to the speed and precision of the padestrian and vehicle detection and localization. Specifically, we perform three approaches: Faster R-CNN, Yolo and SSD, combining with two feature extractors: Darknet19, VGG16, and in Faster R-CNN algorithm, we use different image sizes and different object proposals. After the analysis, the major findings are: the precision depends largely on how deep (big) the feature extractor is; decreasing size of input image would increase the speed of Faster R-CNN but with a decreasing precision; decreasing number of object proposals would increase the speed of Faster R-CNN, with little effect on the precision.**

I. INTRODUCTION

Every year around the world, hundreds of thousands of drivers, passengers and pedestrians die because of fatal traffic accidents. Based on 2016 data from the National Safety Council, more than 40,000 people died in motor vehicles crashes, which is a 6% rise from 2015 and a 14% rise from 2014. Aiming to avoid accidents and reduce human fatalities, research efforts have been put into the fields of environmental perception, vehicle and pedestrian detection, and driver assistance system. In particular, vehicle and pedestrian detection on road has become a hot topic. Thanks to the recent development of deep learning, vision based detection has become more and more important in the complex sensing system of intelligent vehicle, which consists of camera, radar, lidar, etc.

After the first decade of the 21st century, huge progress has been made in the field of computer vision due to the development of modern deep neural networks, especially the deep convolutional neural network. By mimicking the cognitive functions of human brain, these deep networks allow us to perform object classification and detection tasks in a precision that we could never imagine before. This is powered by better computer hardware (GPU in particular), larger dataset and improving algorithms.

Modern object detectors based on CNN, such as faster R-CNN, Yolo and SSD, are the most commonly used in the field of vehicle and pedestrian detection. However, their performances are largely depends on the feature extractors they use, and the hyper parameters they initialize. Depart from Googles paper Speed/accuracy trade-offs for modern convolutional object detectors[1], which evaluates the performance of these detectors in general object detection tasks, there are not much paper about the performance of these different detectors for the specific task of vehicle and pedestrian detection.

In this paper, we generally focus on exploring the key factors that influence the speed and precision of detecting pedestrians and vehicles. More specifically, we compare three different meta-architectures (Faster R-CNN, Yolo, SSD), two different feature extractors(Darknet19, VGG16), and other different hyper parameters for different testing approaches. For simplicity, we primarily focus the precision of 5 classes, which are person, bicycle, car, bus and motorbike, and the "speed" we declare here is refers to the test speed of the algorithm based on specific GPU we used.

Our contributions are as follows:

- We analyze different meta-architecture, and propose how different meta-architecture affects the performance of the pedestrian and vehicle detection.
- In general our final findings are Faster R-CNN architecture will have a slower testing time compare to Yolo and SSD, but Faster R-CNN has higher precision. To be more specific, the test precision depends on how deep the network is. And for Faster R-CNN, decreasing picture size reduces the test time but has significant side effects to the precision, and the decreasing number of proposals decreases the test time with little influence on the precision.

II. META-ARCHITECTURE

For modern object detection, the most popular deep learning architecture is convolutional neural network, this is because convolutional neural network tends to learn different appropriate features. For different meta-architecture of convolutional neural network, they also have different characteristics to influence the performance:

A. Faster R-CNN

Faster R-CNN(Faster Regions with Convolutional Neural Networks)[2] is a advanced version of R-CNN(Regions with Convolutional Neural Networks). R-CNN is a state-of visual object detection system that combines bottom-up region proposals with rich features computed by a convolutional neural network[3], it has excellent object detection precision by using a deep convolutional neural networks to classify object proposal, but the process of the object detection is too slow, because at test time, features are extracted from each object proposal in each test image[4]. And then Ross Girshick introduce the Fast R-CNN, which is a network first processes the whole image with several convolutional and max pooling layers to produce a convolutional feature map, then for each object proposal it uses region of interest (ROI) pooling layer extracts a fixed-length feature vector and then

each feature vector is fed into a sequence of fully connected layers, and the finally generate two sibling output: one is the softmax probability of the estimation over K objects classes, and the other layer that outputs four real-valued numbers for each of the K object classes, then we can use these two outputs the bounding-box positions for one of the K classes[4]. But the problem is, Fast R-CNN still relies on selective search, and selective Search is an order of magnitude slower, at 2 seconds per image in a CPU implementation[2]. In Faster R-CNN, authors introduce the the idea of Regional-Propose Network (RPN), which takes an image as input and outputs a set of rectangular object proposals, each with an objectness score[2], and with RPN, the Faster R-CNN can be composed of two modules: the first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions. To summarize, R-CNN use the convolutional neural network as classification, and Fast-RCNN use selective search generate regional proposal and output bounding box and label, and Faster R-CNN get rid of the selective search by regional propose network and put the regional proposal steps inside the convolutional steps.

B. Yolo

Yolo is a different but similar approach of Faster-RCNN. In R-CNN, Fast R-CNN and Faster R-CNN, the main idea is that use regional proposal and classification to provide bounding box and label, the problem is that for Faster R-CNN, the speed is still not fast enough. Therefore, Ross Girshick propose his idea of Yolo (You-Only-Look-Once) algorithm to illustrate the idea of transform the object detection problem into regression problem. Specifically, Yolo resizes resize the input image to 448 448, and then runs a single convolutional network on the image, then thresholds the resulting detections by the models confidence. By using the Yolo algorithm, we can generally reach the level of real-time objection detection[5].

C. SSD

SSD(Single Shot MultiBox Detector) is first deep network based object detector that does not re-sample pixels or features for bounding box hypotheses and and is as accurate as approaches that do[6]. There are three major features of SSD:

- Multi-scale feature maps for detection. The algorithm add convolutional feature layers to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales.
- Convolutional predictors for detection. Each added feature layer (or optionally an existing feature layer from the base network) can produce a fixed set of detection predictions using a set of convolutional filters.
- Default boxes and aspect ratios. The algorithm associate a set of default bounding boxes with each feature map cell, for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box relative to its corresponding cell is fixed[6].

III. Setup

Here are some details of the test of three different meta-architecture:

- Faster R-CNN: We used pretrained imagenet VGG16 model, and tested on voc2007 dataset.
- Yolo: We used pretrained coco, VOC 2007+2012, darknet19 model, and tested on voc2007 dataset.
- SSD: We used pretrained VOC07+12+COCO trainval, SSD-300 VGG-based model, and tested on voc2007 dataset.

The initial hyper parameter for each meta-architecture:

- Faster R-CNN
  - Batch size: 1
  - RPN batch size: 256
  - Number of regional proposal: 300
  - Image rescale: 600*1000
  - Test NMS threshold: 0.3
- Yolo
  - Batch size: 64
  - Threshold: 0.5
- SSD
  - Batch size: 1
  - Nms threshold: 0.45
  - Matching threshold: 0.5

Other hyper parameters refer to the code and original paper as default[2][5][6]

IV. Result

Here are the average precision of each meta-architecture of different features.

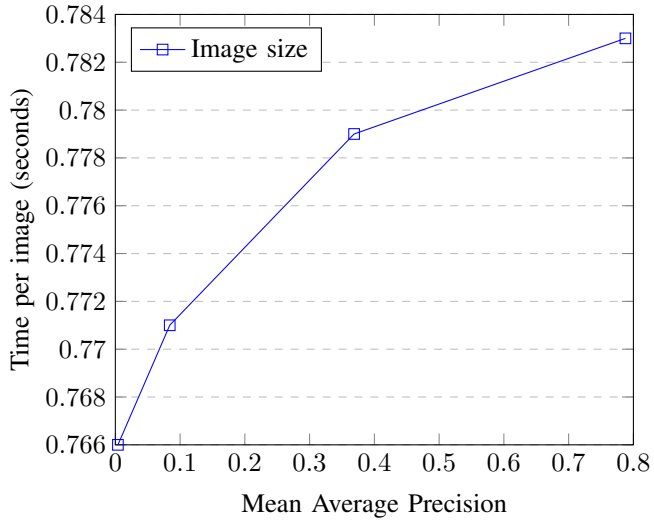
	Yolo	Faster R-CNN	SSD
Person	0.7729	0.7719	0.5190
Bicycle	0.8493	0.7835	0.7941
Bus	0.8439	0.7863	0.8050
Car	0.8590	0.8411	0.8370
Motorbike	0.8552	0.7558	0.8332
mAp	0.8360	0.7877	0.7577
Time	0.713	0.783	0.102

Through the graph, we can see that the mean average precision of Yolo is much higher than other two, this is because we use 19 layers feature extractors for Yolo, and 16 layers feature extractors for Faster R-CNN and SSD, the deeper the neural network is, the more accurate output we can expect. We can also spot that the running time of the SSD is significantly lower than others', this is caused by the difference of the running environment: we run Yolo and FaterR-CNN on linux AWS g2 instance with 4GB GPU k520, and we run SSD on linux with 8GB GPU gtx1080.

Therefore the GPU can also be a factor to influence the speed of the algorithm

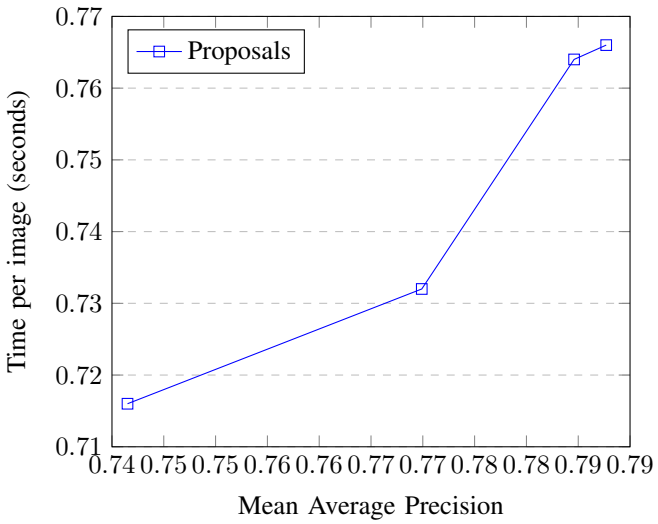
We also perform the Faster R-CNN with different hyper parameters: different image size, by applying different ratio to the original image, and different number proposals.
Here is the result of average precision of different features by using different ratio of the original input image:

	0.7	0.8	0.9
Person	0.0032	0.0478	0.2237
Bicycle	0.0009	0.0667	0.3527
Bus	0.0034	0.1174	0.5268
Car	0.0011	0.0829	0.3113
Motorbike	0.0114	0.1062	0.4288
mAp	0.0040	0.0842	0.3687
Time	0.766	0.771	0.779



The graph shows us that the the size of the image also can influence the precision significantly. Generally speaking, high resolution will tends to reveal more details of the small objects, and therefore low resolution images lead lower mAp than high resolution images. And we also realize that the image size have the direct ratio with the running time per image, this is because with high resolution will generate more bounding boxes in RPN.
Here is the result of average precision of different features by using different number of proposals:

	50	100	200
Person	0.7003	0.7687	0.2237
Bicycle	0.7663	0.7751	0.7855
Bus	0.7492	0.7486	0.7707
Car	0.7966	0.7976	0.8396
Motorbike	0.7593	0.1062	0.7562
mAp	0.7415	0.7699	0.7846
Time	0.716	0.732	0.764



From the information of the chart and graph, we can see that without decreasing much of the mAp, we can reduce the number of the proposals to save the running time.

V. Conclusion

Our major conclusions are:

- The accuracy depends largely on how deep (big) the feature extractor is:
  Using deeper feature extractor, which is darknet 19 in our test, will produce output with higher precision.
- Decreasing size of input image would increase the speed of faster rcnn, but with a decreasing accuracy:

- Decreasing number of object proposals would increase

Our future work will be focus on:

- Investigate loss function and matching, and control them as variables during comparison.
- Evaluate more feature extractors, such as AlexNet, ResNet, etc.
- More data on different input size and number of object proposals to determine mathematical relationship between speed and accuracy.
- Train our own model using more vehicle-specific dataset such as KITTI.

Code are in the github:
https://github.com/liyichen7887/yolo.git
https://github.com/liyichen7887/faster-rcnn.git
https://github.com/huangsuli/SSD-Tensorflow

References

[1] J.Huang, V.Rathod, C.Sun, M.Zhu, A.Korattikara, A.Fathi, I.Fischer, Z.Wojna, Y.Song, S.Guadarrama and K.Murphy, Speed/accuracy trade-offs for modern convolutional object detectors, arXiv preprint arXiv:1611.10012.

[2] S.Ren, K.He, R.Girshick and J.Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, arXiv preprint arXiv:1506.01497.

[3] R.Girshick, J.DonaHue, T.Darrell and J.Malik, Rich feature hierachies for accurate object detection and semantic segmentation,1506.014971311.2524.

[4] R.Girshick, Fast R-CNN, arXiv preprint arXiv:1504.08083.

[5] J.Redmon, S.Divvala, R.Girshick, A.Farhadi, You Only Look Once: Unified, Real-Time Object Detection arXiv preprint arXiv:1506.02640.
[6] W.Liu. D.Anguelov, D.Erhan, C.Szegedy, S.Reed, C.Fu, A.Berg, SSD: Single Shot MultiBox Detector, arXiv preprint arXiv:1512.02325.

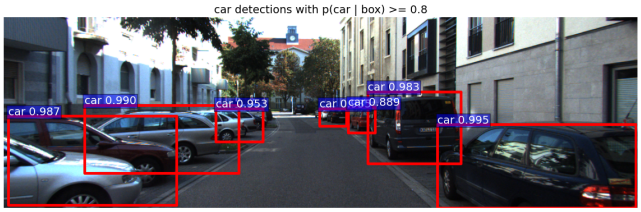


Fig. 1. Demo 1 of Faster RCNN.

Fig. 2. Demo 2 of Faster RCNN.

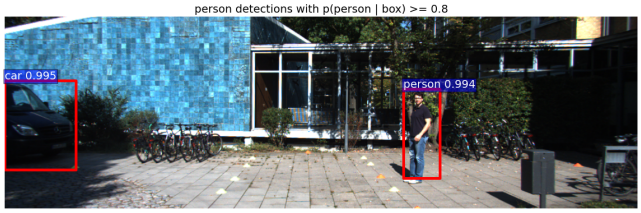


Fig. 3. Demo 3 of Faster RCNN.

Fig. 4. Demo 1 of YOLO.

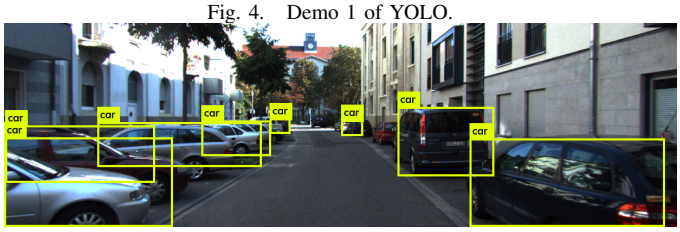


Fig. 5. Demo 2 of YOLO.

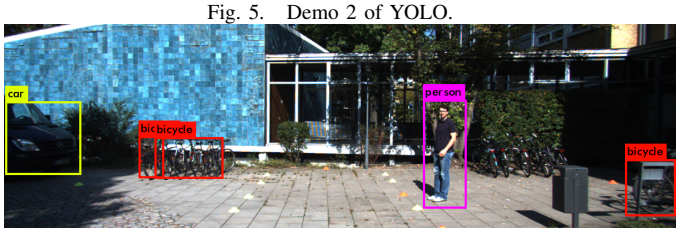


Fig. 6. Demo 3 of YOLO.

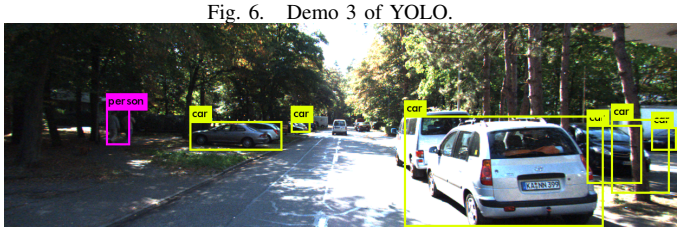


Fig. 7. Demo 1 of SSD.

Fig. 8. Demo 2 of SSd.

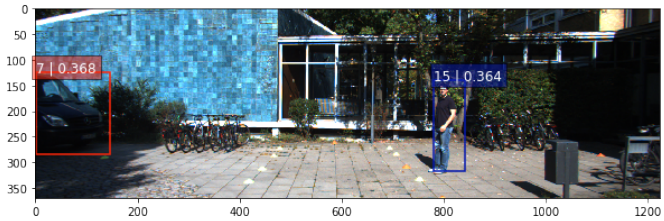


Fig. 9. Demo 3 of SSD.