Object Detection on Road Based on YOLO algorithms

Tianyu Ren
ID: tianyur
tianyur@andrew.cmu.edu

Mo Xu
ID: xmo
xmo@andrew.cmu.edu

Abstract

This project examined the performance for objects detection using YOLOv3 and YOLOv4. Different training strategies were used and compared. KITTI dataset with 7480 images in different environment was used, and most training process was finished on Google Colab platform. The optimal model gives satisfactory mAP up to 92.05% for all 8 classes in KITTI (except DontCare).

1 Introduction

Autonomous driving is quite a trending topic now in which the object detection algorithm regarding vehicles and pedestrians as the main target is one of the most important sub-algorithms. Recently, YOLO algorithm has become the most popular algorithm in object detection algorithm.

We decided to use different versions (V3, V4) of the yolo algorithm using different deep learning frameworks (tensorflow, pytorch) to discuss the difference in performance of the YOLO algorithms and how can we make the best of them.

2 Related Work

2.1 YOLOv3

The main improvements of YOLO v3 are: adjustment of the network structure; use of multi-scale features for object detection; object classification with Logistic instead of softmax.

- New network structure Darknet-53

In terms of basic image feature extraction, YOLO v3 uses a network structure called Darknet-53 (containing 53 convolutional layers). It draws on the practice of residual network and sets up shortcut links between some layers. (Shortcut connections).

- Object detection using multi-scale features

YOLO v2 used the passthrough structure to detect fine-grained features, and YOLO3 further used 3 feature maps of different scales for object detection.

The network performs three detections, which are performed at 32 times downsampling, 16 times downsampling, and 8 times downsampling. In this way, the detection on a multi-scale feature map is a bit like an SSD. The reason for using up-sample (up-sampling) in the network is: The deeper the network, the better the effect of feature expression. For example, if you are performing 16-fold down-sampling detection and directly use the fourth down-sampling feature to detect, then shallow features are used, this effect is generally not good. If you want to use the features after 32 times downsampling, the size of the deep features is too small. So yolo v3 uses up-sample with a step of 2

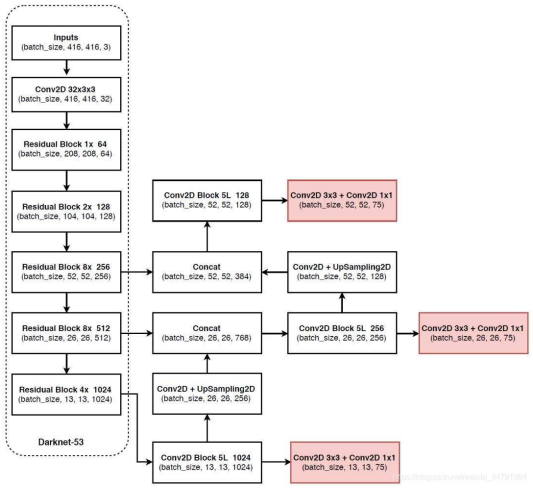


Figure 1: YOLOv3 Network Architecture

to double the size of the feature map obtained by downsampling by 32 times , and then it becomes the dimension after 16 times downsampling. Similarly, 8 times sampling is also to perform upsampling with a step size of 2 for 16 times downsampling features, so that deep features can be used for detection.

2.2 YOLOv4

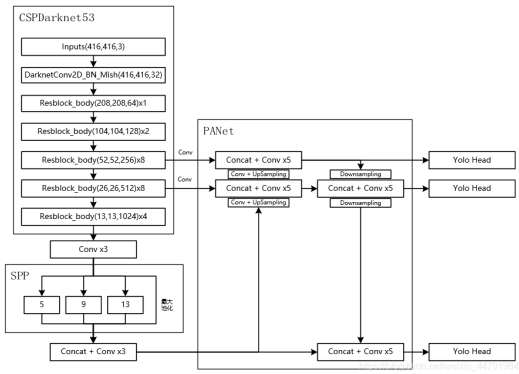


Figure 2: YOLOv4 Network Architecture

The structure of YOLOv4 can be shown in figure 2. Main advantages compared with YOLOv3 can be listed as below:
YOLOv4 uses CSPDarkNet53 instead of DarkNet53 as the backbone net. In CSPResNet, feature map is split into two parts: one does Conv2d and the other is concatenated with the result. This is a cross stage feature fusion strategy[1] and allows the net to directly store and use gradient information. Also, YOLOv4 applies SPPwhich is combined with 4 different size max-pooling in feature pyramid network. It helps to increase receptive field and extract the most significant features. In addition to traditional up-bottom path augmentation, YOLOv4 also applies PANet which is enhanced with accurate localization signals in lower layers by bottom-up path augmentation. In this way, the information path between lower layers and topmost feature is shortened.
Moreover, instead of using pure IOU(Intersection over Union) function in loss function, YOLOv4 uses CIOU, which incorporates all geometric factors: overlapping area, distance, and aspect ratio.

3 Methods

3.1 Data set

The KITTI data set was jointly founded by the Karlsruhe Institute of Technology in Germany and the Toyota Institute of Technology. This data set is used to evaluate the performance of computer vision technologies such as stereo, optical flow, visual odometry, 3D object detection and 3D tracking in a vehicle environment . KITTI contains real image data collected from scenes such as urban areas, villages and highways. In general, the original environment for data collection can be classified into 'Road', 'City', 'Residential', 'Campus' and 'Person'.

In this project, a KITTI dataset with 7480 2D-images and 9 classes was used which contains the targets in all the environments stated above for detection.

3.2 Training tricks

3.2.1 Kmeans for anchors

Both yolov3 and yolov4 use anchor boxes. In a nutshell, an anchor box is to help us set the width and height of common targets. When making predictions, we can use the already set width and height processing to help us make predictions. When training, we also need to use a anchor box to process "y-true" to find which grid point the ground truth corresponds to in the picture.

Sizes of anchor boxes are initially set randomly, which causes negative influence on the procession of bounding boxes decision at very beginning. After we systematically calculate the clustering width and height of anchor boxes using kmeans, we could accelerate the programming of prediction and improve the precision as well.

3.2.2 Label-smoothing

Label smoothing is a simple algorithm that helps to reduce overfitting. Label smoothing replaces one-hot encoded label vector y_hot with a mixture of y_hot and the uniform distribution. Equation can be found as follow: $Y\_ls = (1 - \alpha) \star y\_hot + \alpha/K$ where K represents the number of groups and α is a hyper-parameter. The larger the α is, the distribution will be more similar to a uniform distribution.

3.2.3 Mosaic data augmentation

Mosaic data augmentation combines 4 training images into one in certain ratios, as can be shown in figure 3. It allows for the model to learn how to identify objects at a smaller scale than normal. Also, it increases the diversity of background, and encourages the model to localize different types of images in different portions of the frame.

Figure 3: Example of Mosaic data augmentation

3.2.4 Cosine annealing learning rate decay

Cosine annealing means using cosine function as the learning rate annealing function. The main effect of it is to prevent the training from getting stuck in a local-minimal.

3.3 mAP for performance evaluation

Mean Average Precision, which can be denoted by mAP is used in this project to measure the performance. It can be simply given as the area under precision-recall curve. Precision is calculated by the percentage of correct predictions in all the predictions, and recall measures how many positive cases are detected for a certain group. They all use IOU to determine if the prediction matches with the ground truth. And IOU is given by the area of overlap between two bounding boxes over the area of union. In this project, threshold IOU is 0.5.

4 Experimental Results

The explanation of these results is included in the discussion part.

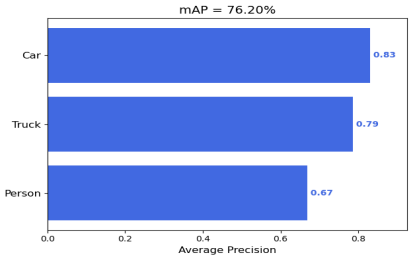
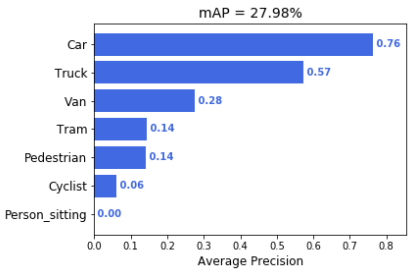
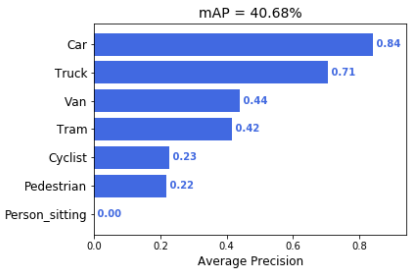


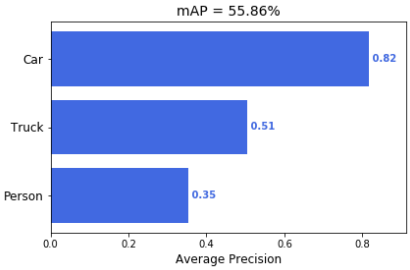
Figure 4: YOLOv3 best performance detecting 3 classes



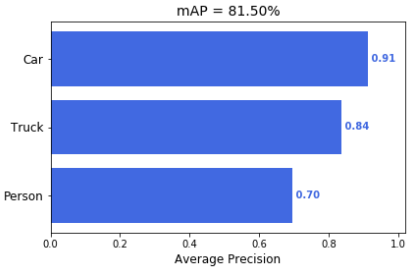
(a) Training_Loss17.96, Val_Loss43.09



(b) Training_Loss14.57, Val_Loss26.28



(c) Total_Loss13.74, Val_Loss25.82



(d) Total_Loss5.83, Val_Loss10.95

Figure 5: YOLOv4 mAP under different conditions

(a) Example result with random anchors (b) Example result with kmeans anchors

Figure 6: Example result with and without kmeans anchors

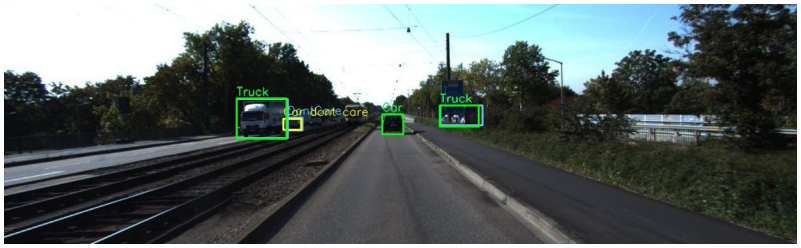
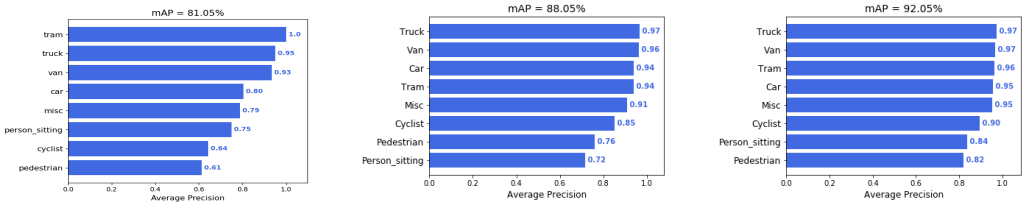


Figure 7: Example of correctly detected object but labeled as DontCare



(a) YOLOv3 416 × 416 resize (b) YOLOv4 416 × 416 resize (c) YOLOv4 608 × 608 resize

Figure 8: Comparison between YOLOv3 and YOLOv4 with different image size

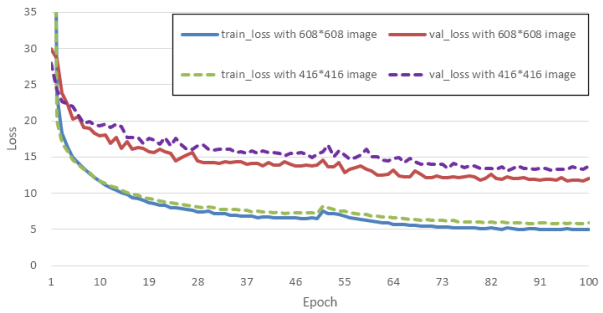


Figure 9: Loss comparison between different resize

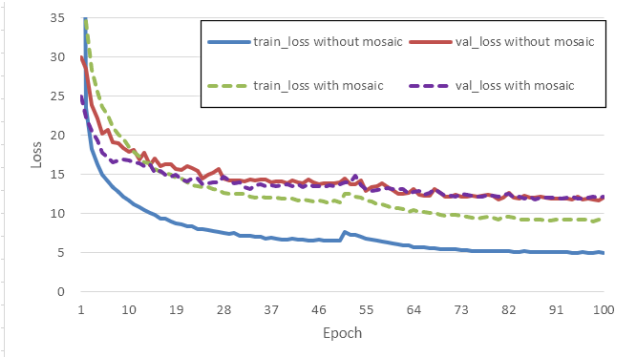


Figure 10: Loss comparison with or without mosaic

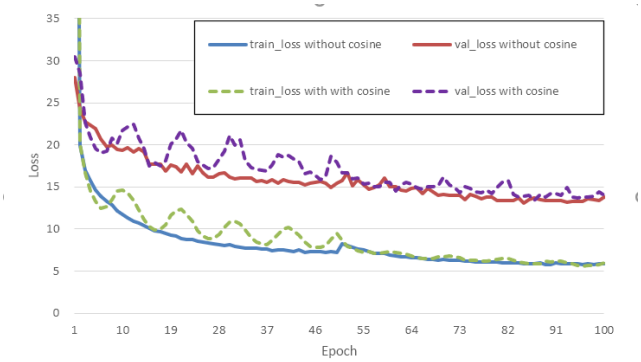
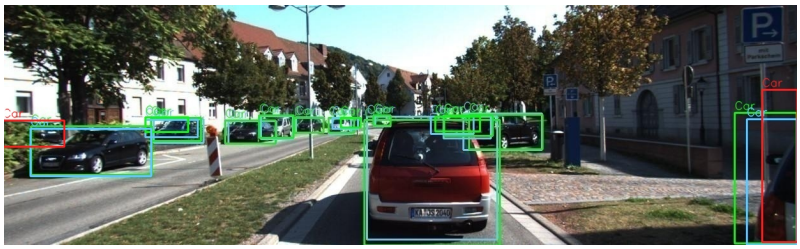
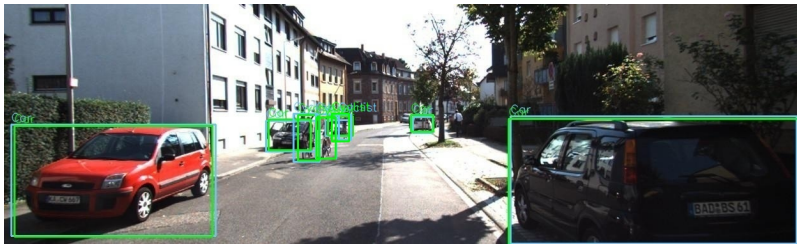


Figure 11: Loss comparison between different learning rate strategy (default: StepLR decay)



(a) Example result1



(b) Example result2

Figure 12: Example result with optimal model

	AP for differnt classes (%)	Car	Truck	Van	Tram	Cyclist	Person_sitting	Pedestrian	Misc	mAP
Traning with images resized into 608*608	without mosaic	95.49	**97.16**	**96.75**	**96.38**	**89.56**	**83.74**	**82.06**	95.27	**92.05**
	with mosaic	**95.56**	97.15	96.51	96.17	89.28	81.92	81.77	**95.32**	91.71
Traning with images resized into 416*416	without cosineLR	93.95	96.54	96.27	93.94	85.06	71.64	76.12	90.91	88.05
	with cosineLR	93.46	97	96.65	93.36	86.54	68.53	78.35	90.65	88.07

Table 1: Performance for each class under different training conditions

5 Conclusion & Discussion

5.1 Discussion

5.1.1 Initial tuning

At initial training stages, redundant training time may make parameter-tuning very inefficient. A whole KITTI dataset with 7480 images will require roughly a whole day to train. Hence a sub-set containing 1500 images was chosen for initial tuning stage. Also, all images are resized into 416×416 for both YOLOv3 and YOLOv4 training model.
From figure 5(a), the original YOLOv4 net has poor performance and so is YOLOv3. Training loss is much lower than validation loss (val-loss $\approx$ 2.4*Training-loss), which means over-fitting occurs.
One good way of reducing over-fitting is label-smoothing. The theory can be shown in chapter 3.2.2 As shown in figure 12(b), label-smoothing successfully reduce over-fitting (val-loss $\approx$ 1.8*Training-loss) and makes the training converges better. Now mAP increases for 40% to 40.68%.
Since only 1500 images are used, for some small groups such as Person_sitting, it may not have enough samples for training and test (only 14 out of 300 test images). So the groups are merged into 3 main groups, namely Person, Car and Truck. Also, Kmeans algorithm(chapter 3.2.1) is used to derive 9 anchor boxes before training. Results can be shown in figure 5(c). In this case, mAP increases for another 40%. In figure 6, it shows that kmeans is important for generating the correct bounding boxes. Finally, pre-trained backbone weights was used. Since the main function of backbone net is to extract meaningful features, a pre-trained weights for backbone net can be used even if it was pre-trained based on different dataset and different . In this project, the pre-trained weights was trained from COCO dataset with tens of thousands of images. The first 50 epochs were trained with a 'freeze' strategy: do not update the weights in backbone part and only update weights outside it. Then unfreeze gradient update in backbone after epoch 50 to further optimize the model. There are two benefits: firstly, the weights in backbone part will not be destroyed at initial training epochs; Secondly, fewer trainable parameters will make the training faster. Results can be found in figure 5(d). Now mAP is 81.5% which is good enough for next training stage.
Also, with the same strategy, YOLOv3 achieves an mAP of 76.20% as in figure 4, which is 7% lower than what YOLOv4 can achieve.

5.1.2 Final training and comparison

At the next stage, the whole KITTI dataset containing 7480 images was used for training. The batch size for the first 50 epochs is 8 and for the last 50 epochs is 4. All groups were reserved except DontCare. In KITTI document, it describes DontCare group as objects that are too far away or too vague and have not yet been classified by KITTI. However, in some rare cases, these objects can be correctly detected by YOLOv4. As is shown in figure 7, a car (marked with yellow box) is hidden in the shadow of a truck, which was labeled as DontCare by the KITTI, is correctly detected by YOLOv4 in this project. For these objects, the predictions matching them are ignored, namely they are not included in false positive nor in true positive.
Firstly the performance of YOLOv3 and YOLOv4 are compared, as is shown in figure 8. With the same 416×416 resizing, v4 also achieves 8% better mAP. Also, from paper[2], the author applies YOLOv4 with 608×608 resize and achieves good result. Hence this idea is also tested in this projected, and YOLOv4 with 608×608 achieves an impressive result with 92.05% mAP. Also, a loss comparison between different image resize is shown in figure 9. It can be found that larger image size gives lower loss especially for validation loss. This means 608×608 image can restore more feature details than 416×416. However, 608×608 image requires roughly 30% to 40% more time for training.
The performance of mosaic data augmentation is also tested. From figure 10, the validation loss is almost the same for training with and without mosaic. But mosaic gives higher training loss, in

which case overfitting is less severe. From table 1, it also shows that mosaic does not ensure better performance. It might because that the background for KITTI dataset mostly share similar features and is not that complex. Also, KITTI tends to ignore too small objects (small objects are labeled as DontCare), hence the advantage of mosaic to identify small objects is limited. Moreover, mosaic augmentation requires around 40% more training time than without it.
The effect of cosine annealing learning rate decay is also examined. Figure 11 shows that CosineAnnealingLR gives similar final result with StepLR. This means StepLR is enough in this project for the model to converge to a global minimum, or at least very close to a global minimum. For CosineAnnealingLR loss curve, some wave-like patterns with decreasing peak periodically appear, which means warm start is correctly taking effect.
Table 1 concludes the performance for each class under different training conditions. It shows it may be necessary to use 608×608 images, but mosaic and CosineAnnealingLR may not be needed for KITTI dataset. AP for Person_sitting, Cyclist and Pedestrian are generally lower than other classes, which shows that people may be more difficult to detect because of mutable motions, gestures and body figures. Figure 12 shows some example results for the optimal model.

5.2 Conclusion

To conclude, YOLOv4 achieves better result than YOLOv3. Several training strategy is needed to get satisfactory result for 2D traffic detection cases, including label smoothing, kmeans for generating anchor boxes, use 608×608 resize strategy instead of 416×416 if long training time is allowed. Also, if possible, pre-train the backbone network with large dataset and doing 'freeze' and 'unfreeze' for the gradients parameters in backbone during training. On the contrary, cosine annealing learning rate decay and mosaic data augmentation seem not to be necessary for traffic detection cases for YOLO network. Future work may focus on whether different traffic detection dataset will have effect on these training strategies. With more complex dataset, maybe other training tricks such as mosaic can take effect. Also, how to further modify and improve YOLO network may be another interesting topic.

References

[1] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.