Exploration of Pedestrian Detection based on Faster RCNN

Fangtong Zhou
fz756
fz756@nyu.edu

Yuqing Li
yl5644
yl5644@nyu.edu

Abstract

In this project, we explored the details of algorithm Faster RCNN which is used for solving object detection problem. To decrease the complexity, we make the problem simpler and only focused on pedestrian detection. We trained the model based on dataset Citypersons. Then we conducted several sub-experiments in order to learn the effect factor such as NMS threshold and scales and ratios of anchors.

1 Task Description

In this project, we plan to complete pedestrian detection based on Faster RCNN model. Pedestrian detection is a special case of object detection, which is significant for surveillance, driving assistance, mobile robotics, etc. Faster RCNN is one of the popular model to complete this task. In this project, we will explore Faster RCNN elaborately. First of all, we will implement Faster RCNN with Pytorch. Further, we will attempt to explore the issues and possible directions which can improve Faster RCNN. There are some challenges in Pedestrian detection: occlusion and lighting. Besides, there are three parts in Faster RCNN: CNN network, RPN network and Faster RCNN. Therefore, they are corresponding to three directions to improve Faster RCNN:

1 Proposing a better CNN network for extraction the feature map;

2 Proposing a more accurate RPN network to obtain more accurate region proposals;

3 A Better ROI classification method.

2 Dataset

In this project, we will complete our project based on database CityPersons, a set of person annotations on top of the Cityscapes dataset. To be specific, CityPersons contains a set of high quality bounding box annotations for pedestrian detection on the Cityscapes dataset (train, validation, and test sets). In fact, Cityscapes dataset provides instance level segmentation but directly using the segmentation result to annotate bounding box is not appropriate. Therefore, all the bounding boxes are adjusted. Using CityPersons has the following benefits:

1 Diversity in cities. The dataset is collected from 18 different cities. Comparatively, Caltech and KITTI only captured images in one city.

2 Diverisity in number of identical persons. CityPersons contains up to ~20000 different identities whereas Caltech and KITTI respectively contains ~1300 and ~6000 pedestrians.

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

Figure 1: One example in dataset CityPersons

2.1 Data Pre-processing

In this subsection, we will introduce how we process the dataset which chould provide for the Faster RCNN in detail. First of all, the dataset format for Faster RCNN should be in PASCAL VOC, which stores the result in XML. However, the dataset provided in Citypersons stores the label file in Json. Therefore, we first convert the label file into XML. Besides, the dataset organized for Faster RCNN should be similar to Fig. 2. To be specific, we need to put all the images including both train and validation images in one directory named "JPEGImages" and put the label files into a directory named "Annotations". In order to make the algorithm understand which images are used for training and which images are used for validation. We created three txt files containing the corresponding image names and put them into the directory "Imagesets/Main". In addition, we need to code images I/O python files and add out dataset into the factory file.

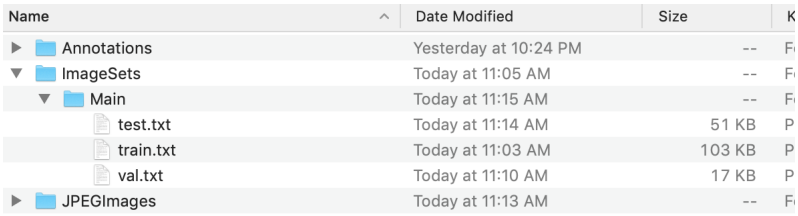

Figure 2: Organization of dataset in PASCAL VOC format

2.2 Issues in Label files

Even we complete all the operations above correctly, there still have issues when training. The reason is that the provided label files are wrong. Since the ground truth bounding boxes are labeled by human, it is unavoidable to have some wrong cases. For examples, some of the bound boxes are overlap with the edge in image. When accesssing its ROI, it will appears out of index error. Besides, since most of the mahcine learning algorithms will apply horizontal flip for the training images in order to do data augmentation. However, due to the wrong bounding boxes will also bring errors. Therefore, what we did is to shrink those bounding boxes whose edge is already out of the image range.

3 Methodology

In this section, we will elaborately introduce the architecture of Faster RCNN. Faster RCNN**?** is proposed by Kaiming He et al which is a improved version of RCNN and Fast RCNN. In this model, they proposed Region Proposal Networks (RPN) network that share convolutional layers with extraction component. In this way, the time consumption of proposing Region of Interest (ROI) will be extremely small. The previous approaches utilized Selective Search as a component to propose ROI which is a completely independent step. The architecture is shown in Figure 3

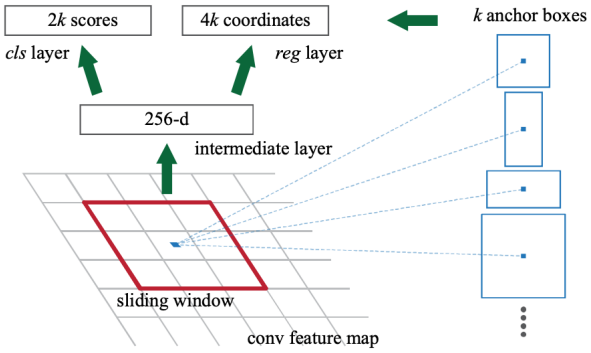

Figure 3: Region Proposal Network.

To be specific, for each anchor in feature map, it is corresponding to k windows with different aspects and scales such as when $k = 9$ it yields 3 kinds of aspects and 3 kinds of scales. For each window, a fully connected layer maps the input to 2 score outputs representing whether this window is foreground or background and 4 coordinates representing location x and y and width and height. The loss function for RPN network is

$$L(p_i, t_i) = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p_i*) + \lambda \frac{1}{N_{\text{reg}}} \sum_i p_i^* L_{\text{reg}}(t_i, t_i^*) \tag{1}$$

After proposing ROI based on RPN network, the proposed ROIs are fed into Fast RCNN **?**. The architecture is shown in Figure 4. Fast R-CNN network takes an image and a set of proposals. The network processes the image with several convolutional and max pooling layers to produce a conv feature map. Then, for each object proposal, a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map to make all of the ROI to be equal. Each feature vector is fed into fully connected network that finally branch into two sibling output layers: one that produces softmax probability over $K + 1$ object classes and another layer that outputs 4 values for each of the K object classes. Each set of 4 values represents the bounding-box positions.

Training Phase Faster RCNN is trained with alternative training method. First of all, RPN is trained independently. Secondly, with the generated ROIs, the conv layers in Fast RCNN are trained in this step. Finally, fine-tuning the FC layers of the Fast R-CNN.

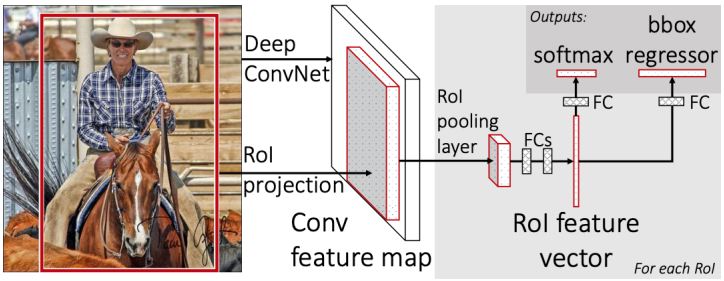

Figure 4: Architecture of Fast RCNN

4 Experiments

In this section, we first introduced the experiment environment and the implementation details. Then we will design several comparative experiments to explore the factors which could affect the performance of Faster RCNN.

4.1 Environment

We run our experiments on Google Cloud Platform as required. GPU is Tesla K80 and the its memory is 11441M. The version of Python is '2.7.13' and that of Tensorflow is '1.13.1'.

4.2 Implementation Details

We utilized VGG16 as the feature extractor and then we initialized the parameters with an already trained VGG16 model for a classification task. The training and validation data are pre-processed as mentioned in section 2. During the implementation, we confronted several issues. Most of them are caused by incorrect data annotation. For example, some bounding boxes are overpass the edge of image. Besides, the annotation are 1-base instead of 0-base, which means the left-top coordinate is (1,1) instead of (0,0).

4.3 Baseline Experiment

In this subsection, we will introduce a baseline experiments which is used for compared with the following experiments. There are several hyper parameters for Faster RCNN. In the baseline experiment, we set the hyper parameters as Tab. 1. And we trained the whole network without any freeze operation.

Table 1: Hyper Parameter Setting of Baseline Experiment

Hyper-Parameter	Value
Learning Rate	0.00001
NMS Threshold	0.8
Iterations	70000
Batchsize	28
Anchors Scale	[8, 16, 32]
Anchors Ratio	[0.5, 1, 2]
Positive Overlap	0.7
Negative Overlap	0.3

The performance of baseline experiment shows in Tab. 2. And Figure shows one test example selected from the test set by inputting into the baseline model.

Table 2: Performance of Baseline Experiment

Class Name	AP
Pedestrian	0.4853
Rider	0.1474
Mean-AP	0.3163

From Figure 5, we listed two test samples from a corner in Berlin. For the top image, there are many riders on the road and the algorithm detects three of them. For the bottom image, there are one rider and two pedestrians but the algorithm only recognize the rider. We can find that the environment in both of the images are too somber. Therefore, some riders in the first images is too hard to detect even for human. However, the pedestrians in the second image is obvious but there exist an occlusion. Therefore, the first operations we want to try is to adjust the NMS threshold when testing.

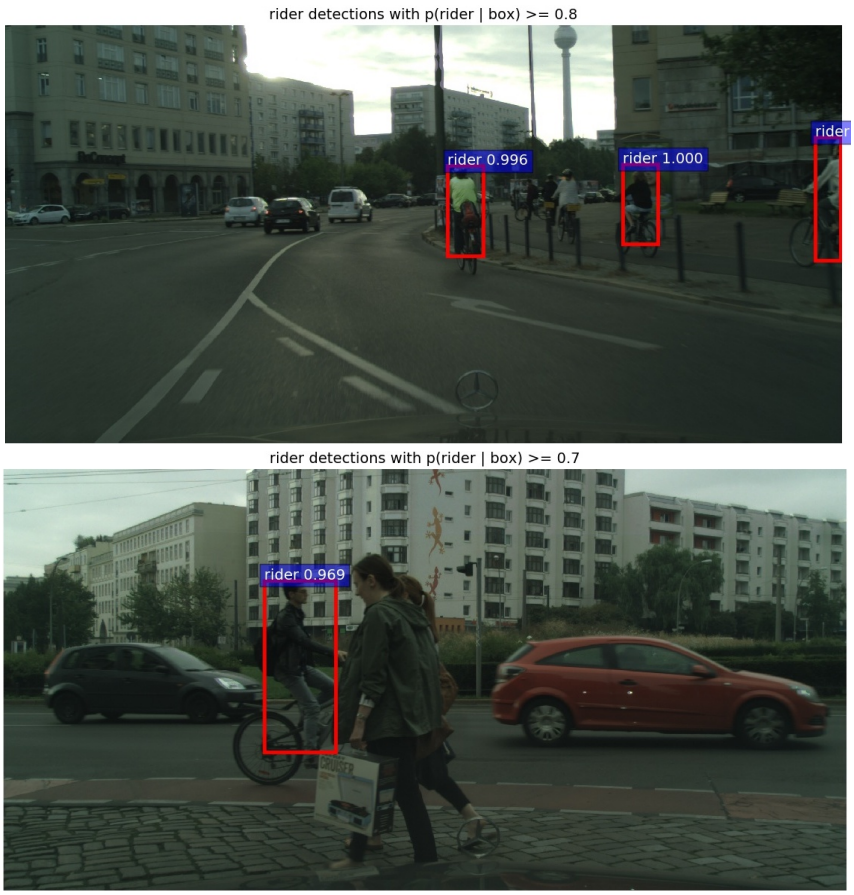

Figure 5: **Top:** One test image from road of Berlin.(MNS Threshold$\geq$ 0.8). There are three riders is detected. **Bottom:** Second test image from road of Berlin.(MNS Threshold$\geq$ 0.8). There's one rider been detected.

4.4 Affect of NMS Threshold

As we mentioned above, it might be the NMS criteria is too strict which leads to some negative false. Since all of the ROIs predicted by Fast-RCNN will transfer into NMS algorithm to do a filter operation. For those bounding boxes that Faster RCNN is unsure about its category, NMS algorithm will filter it out. Therefore, the threshold for NMS should be adjusted and it will directly influence the result. And in this subsection, we will conduct the experiments with different NMS Threshold values. The performance under different value of NMS threshold shows in Tab. 3. From the table, we can observe that the value should be adjusted into an appropriate value. Too large or too low will both bring negative impacts to the detector.

Table 3: Performance of Experiment across different NMS threshold

Experiment	NMS Threshold	Pedestrian	Rider	Mean-AP
Baseline	0.7	0.4853	0.1474	0.3163
NMS EXP1	0.6	0.4791	0.1437	0.3114
NMS EXP2	0.8	0.4873	0.1474	0.3174
NMS EXP3	0.9	0.4808	0.1459	0.3133

In addition, we apply the best value 0.8 among all the experiments and test with the same two images. We can find that in the top image, another rider has been detected but in the bottom image the two pedestrians still seem to be transparent to the detector.

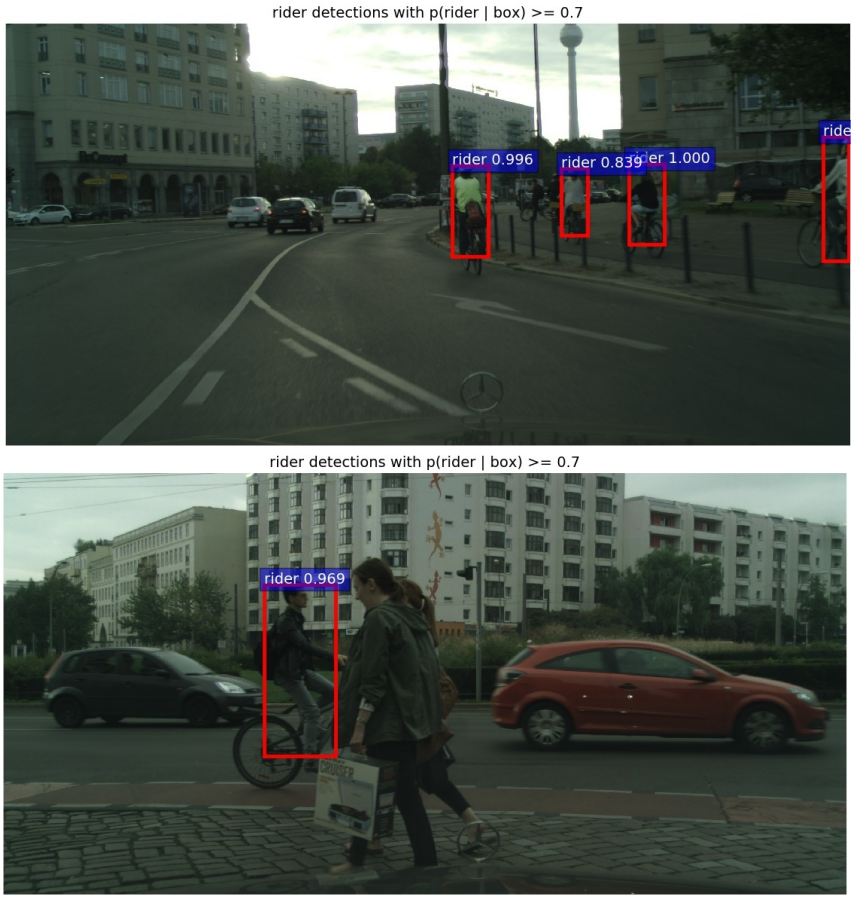

Figure 6: **Top:** One test image from road of Berlin.(MNS Threshold$\geq$ 0.8). There are three riders is detected. **Bottom:** Second test image from road of Berlin.(MNS Threshold$\geq$ 0.8). There's one rider been detected.

4.5 Affect of Different Anchor scale and ratio

From the bottom test image in Fig.6, we can find that the two pedestrian still can't be detected even though we change different nms threshold. Therefore, in this subsection, we are going to try different anchor scales and ratios combinations to improve the performance. Table 4 shows the result of Faster RCNN under different anchors scales and ratios.

Table 4: Performance of experiment across different anchor scales and ratios

Experiment	Anchors of scales and ratios	Pedestrian	Rider	Mean-AP
Baseline	[8,16,32]+[0.5,1,2]	0.4853	0.1474	0.3163
Anchor EXP1	[8,16,32,64]+[0.5,1,2]	0.4854	0.1472	0.3163
Anchor EXP2	[8,16,32]+[0.5,1,1.5,2]	0.4840	0.15074	0.3170
Anchor EXP3	[8,16,32,64]+[0.5,1,1.5,2]	0.4844	0.1508	0.3176

Figure 7 shows the above false negative test sample in the corner of Berlin. And we can find by adding more scales and ratios can improve the performance so that the detector can be fitful into different kinds of shape. However, since there exist occlusion between these two pedestrians so it is really hard to detect both of them.

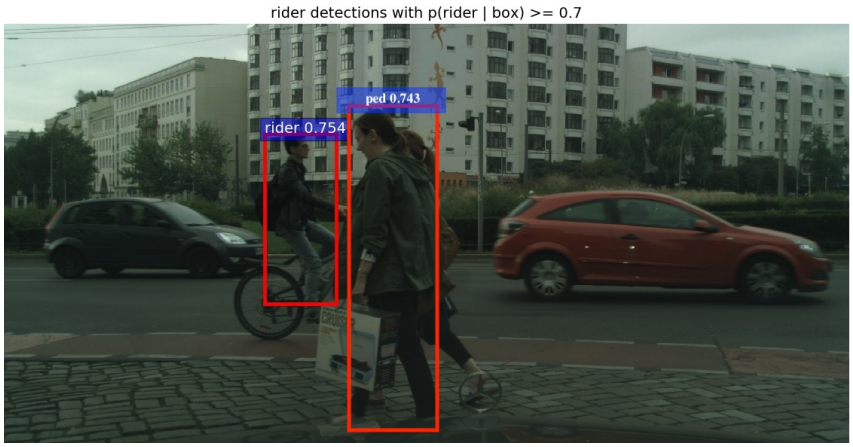

Figure 7: One test image from road of Berlin.(MNS Threshold$\geq$ 0.7). The anchor scale is [8,16,32,64] and ratio is [0.5,1,1.5,2].