# Proposal：Pedestrian Detection based on Faster RCNN

**Group member:**

Fangtong Zhou fz756 N10877027

Yuqing Li yl5644 N10560426

**Main Idea**

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.

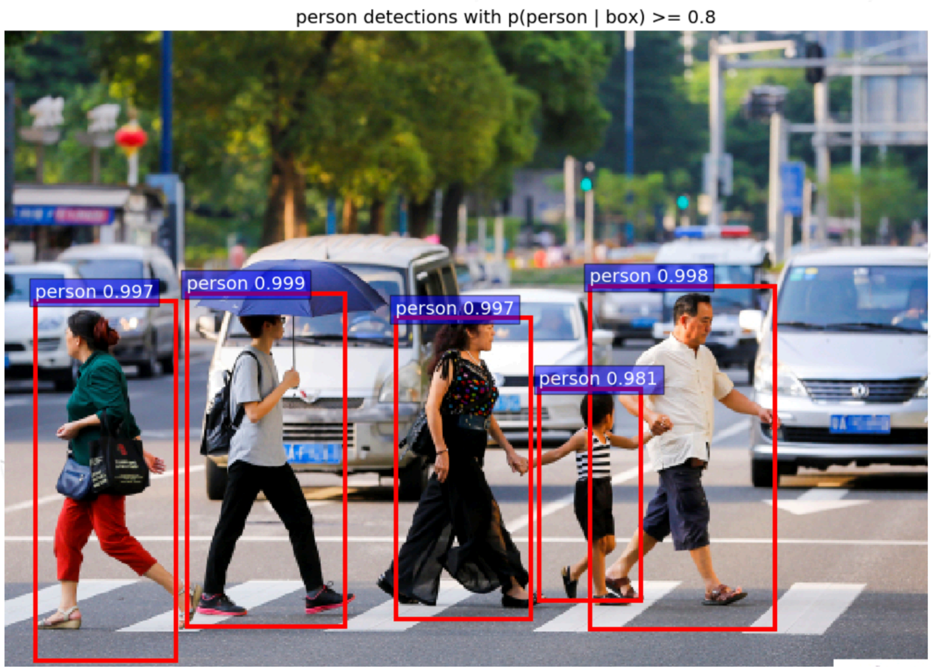

Figure1: Demo results

**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 Diversity in number of identical persons. CityPersons contains up to ˜20000 different identities whereas Caltech and KITTI respectively contains ˜1300 and ˜6000 pedestrians.

**Architecture**

In this section, we will elaborately introduce the architecture of Faster RCNN. Faster RCNN is proposed by Kaiming He et al which is an 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 2.

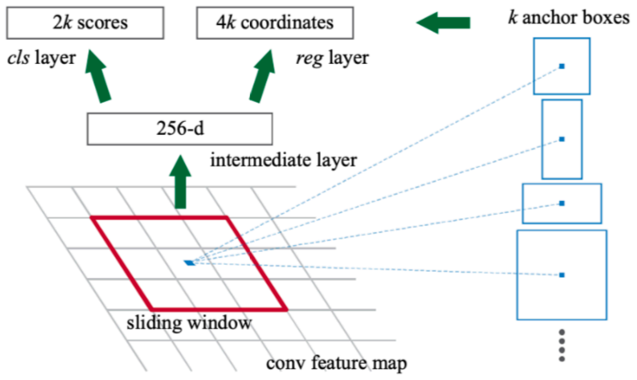

Figure2:Region Proposal Network

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

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

After proposing ROI based on RPN network, the proposed ROIs are fed into Fast RCNN. The architecture is shown in Figure 3. 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.

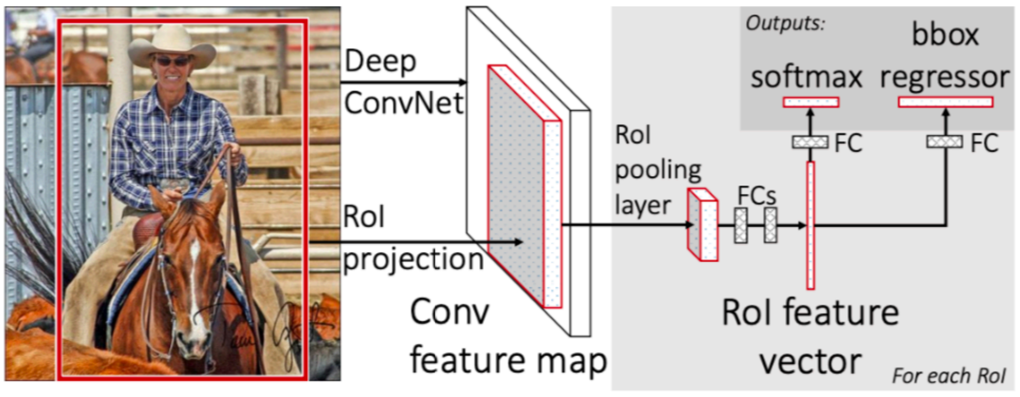

Figure3: Architecture of Fast RCNN

## References

We use other’s faster RCNN code as a staring point, whose github address is https://github.com/endernewton/tf-faster-rcnn, but this code can only run some demo and cannot train and test, so we adjust it and use our own dataset to train and test, and if possible, we will add some our new ideas to this algorithm.