# CA Report for Faster RCNN training and testing in keras

Team 2 – Ma Weizhong, Xu Kaixin, Yu Xiaoxi

## 1. Repository Link:

https://github.com/kartmannXu/keras-frcnn

## 2. Fast RCNN introduction

### 1) The overall framework

Faster RCNN was proposed to improve Fast r-cnn. Because the test time in the paper of Fast RCNN does not include the search selective time, and a large part of the test time is spent on the extraction of candidate regions. Faster RCNN was proposed to solve this problem.

The figure shows the network structure of Faster RCNN. It can be seen that Faster RCNN is composed of four parts

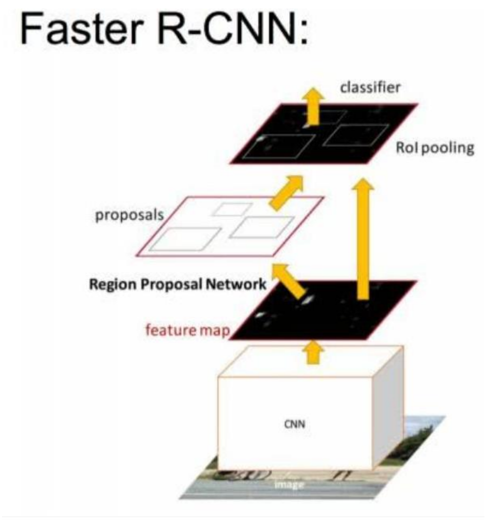


Fig. 1 Overall pipeline of Faster R-CNN

a) Conv layers:

As a kind of CNN network target detection method, Faster RCNN first uses a set of basic conv+relu+pooling layer to extract feature maps of input image, which will be used for subsequent RPN layer and full connection layer

b) RPN(Region Proposal Networks):

RPN network is mainly used to generate region proposals. First, it generates a bunch of Anchor boxes, and after filtering them by， it is decided by softmax that anchors belong to foreground or background, that is, object or not. At the same time, another branch bounding box regression modifiers anchor box to form a more accurate proposal

c) Roi Pooling:

This layer utilizes proposals generated by RPN and the feature map obtained by the last layer of VGG16 to get a proposal feature map of fixed size, which can be followed by full connection operation for target identification and positioning

d) Classifier:

The Roi Pooling layer will form a fixed size feature map for full connection operation, and use Softmax to classify specific categories. Meanwhile, L1 Loss will be used to complete bounding box regression operation to obtain the precise location of objects.

## 2) **Network Structure**

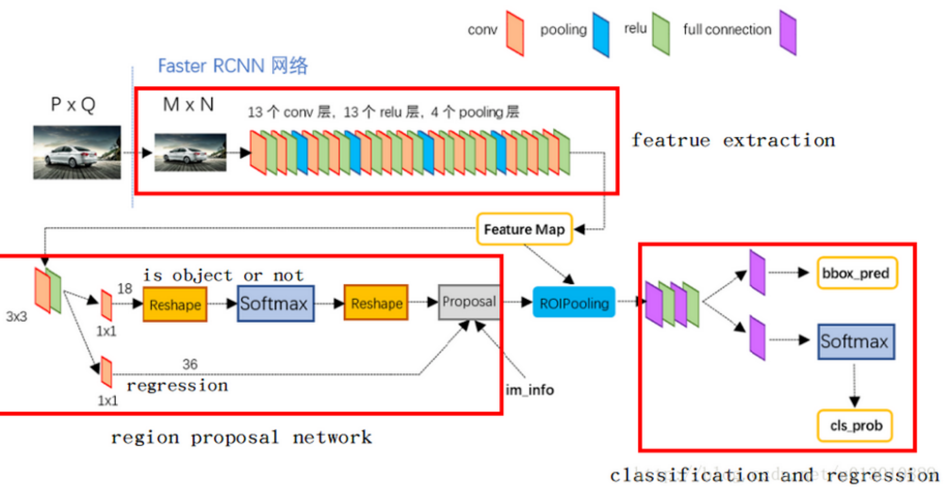


Fig. 2 Modules in Faster R-CNN (Image from Internet)

Now, start the layer-by-layer analysis through the figure above

1) Conv layers

Faster - RCNN first support the input of any size of images, such as in the figure above the input of P * Q, before entering the network set to the correct scale of image, such as image can be set no more than 600 * 1000, we can assume that M * N = 1000 * 600 (if the image is less than the size, edge can be 0, the image will have black edge)

- 13 conv layers: kernel_size=3,pad=1,stride=1; Therefore, the conv layer does not change the image size.
- 13 relu layers: activation function, does not change the image size
- 4 pooling layers: kernel_size=2,stride=2; The pooling layer will make the output picture be 1/2 of the input picture
- After Conv layers, the image size becomes (M/16)*(N/16).

2) RPN(Region Proposal Networks):

After the Feature Map enters RPN, it first goes through a convolution of 3*3, the purpose of which should be to further concentrate the Feature information, and then see two full convolution, kernel_size=1*1,p=0,stride=1.

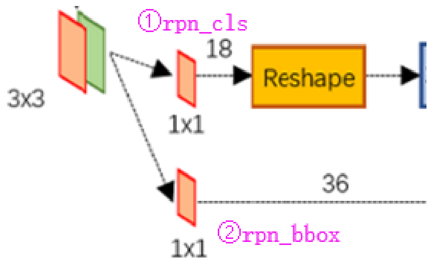


Fig. 3 Region Proposal Network (RPN)

As shown in the figure above:

(1) rpn_cls: classify its 9 Anchor boxes pixel by pixel

(2) rpn_bbox: get its 9 Anchor box　ａｎｄ four coordinate information pixel by pixel

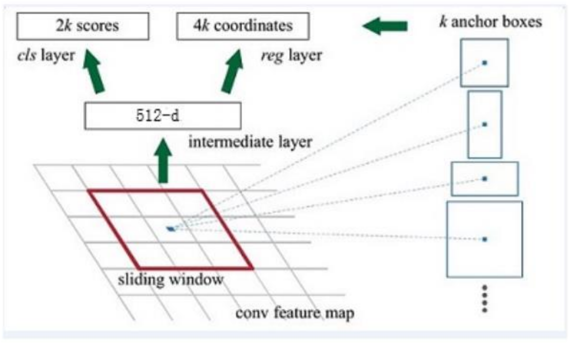


Fig. 4 Anchor Box in Faster R-　CNN

This layer mainly generates 9 Anchor boxes for each pixel in the feature map, and filters and marks the generated Anchor box. The rules for filtering and marking are as follows:

(1) remove the anchor box that exceeds the boundary of the original picture

(2) if the IoU value of anchor box and ground truth is the largest, mark it as positive sample and label=1

(3) if the IoU of anchor box and ground truth is larger than 0.7 is marked as positive sample, label=1

(4) if the IoU of anchor box and ground truth is less than 0.3, it is marked as negative sample and label=0

The remaining samples are neither positive nor negative samples and are not used for the final training. Label =-1

In addition to marking anchor box, another thing is to calculate the offset between anchor box and ground truth. Through the difference between ground truth box and predicted anchor box, the weight in RPN network can learn the ability of prediction box

3) ROI Pooling:

Region proposal generated by RPN layer and feature map generated by the last layer of VGG16 are input. Each region proposal is traversed and its coordinate value is reduced by 16 times. In this way, region proposal generated on the basis of the original image can be mapped to the feature map, and a region (defined as RB*) can be determined on the feature map.

The area RB* determined on the feature map is divided into 7*7, namely 49 small areas of the same size, according to the parameters pooled_w:7 and pooled_h:7. For each small area, Max pooling method is used to select the maximum pixel point as the output, thus forming a 7*7 feature map. After traversing the region proposal, many feature maps of size 7*7 will be generated as the input of full connection of the next layer

4) Full connected layer:

After the ROI pooling layer, fully connect the feature graph, refer to the following figure, and finally complete classification and positioning with Softmax Loss and L1 Loss

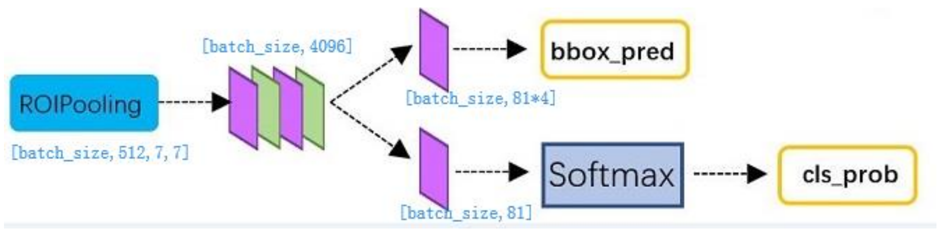


Fig.5 Classifier in Faster R-CNN

Calculate the specific category of each region proposal (such as person, horse, vehicle, etc.) through full connect layer and softmax, and output cls_prob probability vector. At the same time, bounding box regression is again used to obtain the position offset bbox_pred of each region proposal, which is used for regression to obtain a more accurate target detection box

That is, after the 7*7 proposal feature maps of are obtained from PoI Pooling, the main work is done through the full connection:

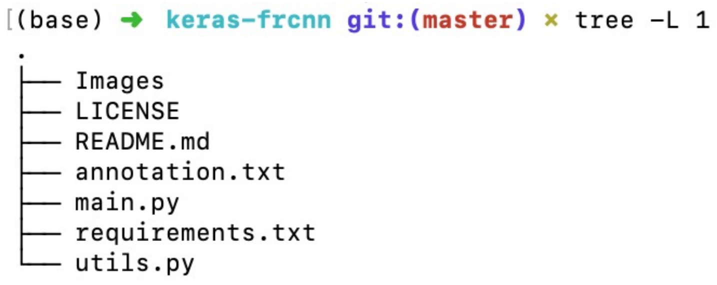
1.classify region proposals by specific category through full connection and softmax

2. bounding box regression is performed again for region proposals to get a rectangle box of higher accuracy

## 3. Implementation details

### 1) Code structure

The project code directory is as follows:



```
[(base) ➜  keras-frcnn git:(master) ✗ tree -L 1
.
├── Images
├── LICENSE
├── README.md
├── annotation.txt
├── main.py
├── requirements.txt
└── utils.py
```

**Explanations:**

- Images: directory containing our built dataset
- annotation.txt: Annotations
- main.py: Training, test code file
- utils.py: Library code

## 2) APIs design

The code running process is as follows

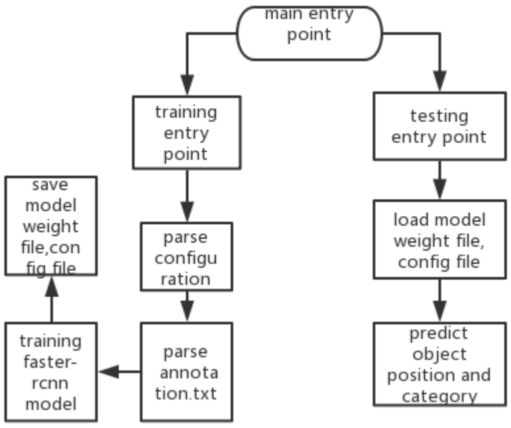

main entry point
training entry point
testing entry point
save model weight file,config file
parse configuration
load model weight file, config file
training faster-rcnn model
parse annotation.txt
predict object position and category

Fig. 7 Implementation Skeleton

## 3) API instructions and notes:

• Both theano and tensorflow backends are supported. However compile times are very high in theano, and tensorflow is highly recommended.

• main.py can be used to train and test a model. To train on Pascal VOC data, simply do:

```
python main.py --mode train -p /path/to/pascalvoc/.
```

• the Pascal VOC data set (images and annotations for bounding boxes around the classified objects) can be obtained from:
http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar

• simple_parser provides an alternative way to input data, using a text file. Simply provide a text file, with each line containing:

```
    filepath,x1,y1,x2,y2,class_name
```

For example:

`/data/imgs/img_001.jpg,837,346,981,456,cow`

`/data/imgs/img_002.jpg,215,312,279,391,cat`

The classes will be inferred from the file. To use the simple parser instead of the default pascal voc style parser, use the command line option -o simple. For example `python main.py --mode train -o simple -p annotation.txt`.

- Training will write weights to disk to an hdf5 file, as well as all the setting of the training run to a pickle file. These settings can then be loaded in testing part. Thus in testing, only these 4 arguments are required:
- `--mode, --config_filename, --path, --num_rois`
- Test mode in main.py can be used to perform inference, given pretrained weights and a config file. Specify a path to the folder containing images:

```
python main.py --mode test -p /path/to/test_data/ -c
/path/to/config_file.pickle
```

- Data augmentation can be applied by specifying --hf for horizontal flips, --vf for vertical flips and --rot for 90 degree rotations
- In training, the value of output_weight_path must contains a pair of empty brace, you could refer to the default value: `weights\\model_frcnn_{}`.
- In testing, the value of input_weight_path is NOT the file name, it should be the prefix of the weights files output by training process, without classifier/rpn.h5 suffix. for example, if the weight files are `weights\model_frcnn_resnet50classifier.h5` and `weights\model_frcnn_resnet50rpn.h5`, then the value of `input_weight_path` should be `weights\model_frcnn_resnet50`.
- The value of `--path` in training should be the path to annotation file, e.g. annotation.txt, while in testing the value should be a directory that includes the images to be tested.

SIMPLE EXAMPLE:

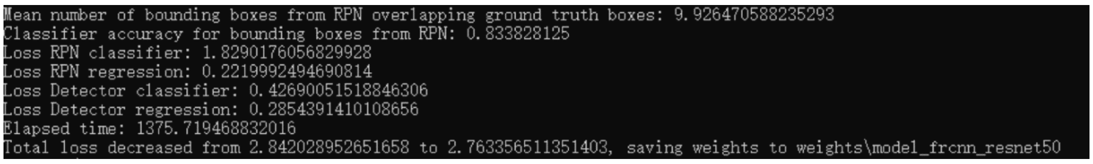
- Train: `python main.py --mode train -o simple -p annotation.txt -hf -rot -c config_own.pickle`
- Testing: `python main.py --mode train -o simple -p Image\\001`

NOTES:

- config.py contains all settings for the train or test run. The default settings match those in the original Faster-RCNN paper. The anchor box sizes are [128, 256, 512] and the ratios are [1:1, 1:2, 2:1].
- The theano backend by default uses a 7x7 pooling region, instead of 14x14 as in the frcnn paper. This cuts down compiling time slightly.
- The tensorflow backend performs a resize on the pooling region, instead of max pooling. This is much more efficient and has little impact on results.

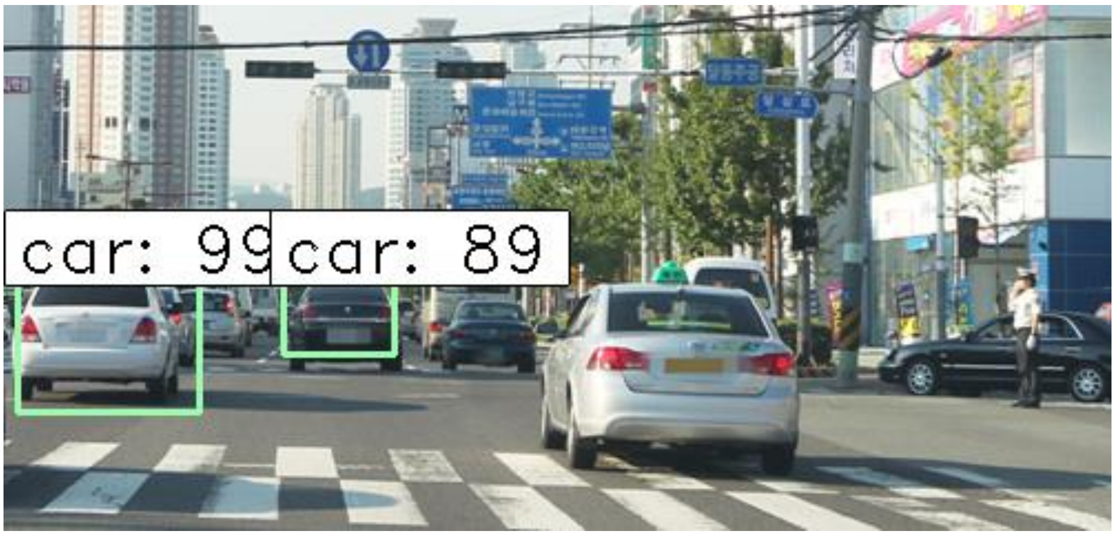
## 4) Training and Testing Results

The below screenshot shows the training stage loggings, containing the losses being optimized and iteration speed on our environment.



```
Mean number of bounding boxes from RPN overlapping ground truth boxes: 9.926470588235293
Classifier accuracy for bounding boxes from RPN: 0.833828125
Loss RPN classifier: 1.8290176056829928
Loss RPN regression: 0.2219992494690814
Loss Detector classifier: 0.42690051518846306
Loss Detector regression: 0.2854391410108656
Elapsed time: 1375.719468832016
Total loss decreased from 2.842028952651658 to 2.763356511351403, saving weights to weights\model_frcnn_resnet50
```

The test results are as follows,



## 4. Findings and Conclusion

Faster RCNN's main contribution is to design a network RPN to extract candidate regions, which replaces the time-consuming Selective Search and greatly improves the detection speed. Structurally, Faster RCNN has integrated feature extraction, proposal extraction, bounding box regression(rect refine) and classification into one network, which greatly improves the overall performance, especially in terms of detection speed.

With the continuous development of autonomous driving, the more important the visual target detection becomes. From the experimental results, we can further improve from the following aspects: 1. Increase the labeled data set 2. Make further adjustments to your own data set based on the online pre-trained model. 3. Since target detection is only a small part of the autopilot function, we need to size and speed the model. Also consider using YOLO V3, SSD or other lightweight model instead of Faster RCNN for real-time target detection since Faster R-CNN is still on the performance side in the trade-off.