IE534 Final Project: Faster R-CNN

Chaoyue Cui, Cunwei Fan, & Yuanyuan Zhai

December 2018

1 Introduction

Faster R-CNN is proposed to shorten the time spent on region proposal step during test time. Previous region proposal method such as Selective Search (SS), is an order slower than state-of-the-art object detection network (`Fast R-CNN`). Hence the authors of Faster R-CNN came with the idea that region proposal step can be combined with `Fast R-CNN`, so that it can take advantage of GPUs as well as sharing computations. Faster R-CNN is composed of two modules. The first one is Region Proposal Network (`RPN`), which share convolutional layers with the second module, `Fast R-CNN`.

The `RPN` first generates a set of bounding boxes with one or different size scales and ratios on each pixel. Since there will be many boxes with a relatively large overlap area, we will use Non-Maximum Suppression (`NMS`) technique to reduce the number of candidate boxes. Then, bounding boxes with Intersection-over-Union (`IoU`) score (with ground-truth box) higher than 0.7 are selected as foreground, and that lower than 0.3 as background. To optimize the `RPN` during the training, both classification loss and bounding box regression loss will be calculated. Classification Loss is the entropy loss on whether it's foreground or background. For Bounding Box Regression Loss, the difference between the regression of foreground box and that of ground truth box is plugged into L_1 function, and then sum all terms together[1].

For the second module (or detection network), `Fast R-CNN`, there is a "head" part which processes the original image using several convolutional layers and max pooling layers to produce a feature map. The "tail" part takes object proposals (from `RPN`) to the region of interest (`RoI`) pooling layer and extracts a fixed-length feature vector from the feature map. Then, the feature vector is put into a sequence of fully connected layers which will branch into two output layers. One will produce softmax probability to estimate over 20 object classes and a "background" class. Another will output four real-valued numbers for each of the 20 object classes. Note that a set of 4 values will be used to calculate refined bounding box positions for a certain class [2].

The author introduced three training methods. The first one is *4-Step Alternating Training*: initialize the convolutional layers in `RPN` network by a pre-trained model and train the `RPN` network; initialize the `Fast R-CNN` convolutional layers with the pre-trained model; use the proposals generated from the trained `RPN` to train the `Fast R-CNN`; use the trained `RPN` to train the layers above the shared convolutional layers of `Fast R-CNN`. The second one is *Approximate joint training*, which is the one we use to train out model, and it will be introduced later. The third one is *Non-approximate joint training*. It involves the derivative w.r.t. the proposal boxes' coordinates that is ignored in the second method.

2 Implementation

In this section, we will briefly introduce the main idea of the code our group developed for this project. We will explain the architecture of the code along with some important hyper-parameters as well as the training method we used for this network.

2.1 Architecture

The architecture of the network is explained in great detail in the original paper [1]. Thus, we mainly follow the code structure of Girshick, one of the paper author, from his github repository [1].

The structure of the network is that a convolutional neural network is on the base and its feature map from the last layer is used to feed a region proposal network (`RPN`). We used two kinds of base net in our

project: the VGG16 and Res101. On the other hand, we generate initial bounding boxes for each pixels through the image, (the initial bounding boxes are called anchors) and RPN learns to extract foreground anchors and adjust its shape to get bounding boxes. Then we apply non-maximum suppression (NMS) layer to the proposed bounding boxes to eliminate their numbers. The suppression is done to eliminate boxes that overlap greatly and only keeps those with the top foreground scores.

So far, the network has proposed bounding boxes for the region of interest, then we will use them to find classes for the object int he bounding boxes as well as a class dependent regression for the bounding boxes. We use crop pooling to extract feature maps to a given shape, so that all bounding boxes will give feature map with the same shape. As a result, we can pass the feature maps of standard shape from the bounding boxes to the last classifier layer to have class and bounding box regression coefficients for the bounding boxes. Both of the classifier and the bounding box regression layer are one layer linear fully connected network.

2.2 Training Method

In this architecture, we have introduced four losses. The RPN network will output a foreground/background classification cross entropy loss as well as the bounding boxes regression loss. The last classifier layer will introduce the object classification cross entropy layer and the class dependent bounding box regression loss. We sum all the four losses and use backpropagation to decrease the loss. This is the method called *Approximate joint training*. We choose this method because it is easier to implement and more convenient to train. Note this solution is approximate since it ignores the derivative with respect to the proposal boxes' coordinates which are also network responses. Although it is not exact, it works pretty good in practice. One reason might be that the RPN gets 'direct' supervision from optimizing the total loss.

With the main idea introduced, it is better to explain the bounding box regression loss we used since this is loss is not a classical loss. We chose to use smooth L1 loss as suggested by the original paper [1]. In implementing this loss for RPN, we choose anchors that overlap greatly with the ground truth box as foreground anchors. The RPN net will give regression coefficients for the foreground anchors $u_i($ predicted), with $i =$ width, height, x and y. We will use the ground box which overlap mostly with the anchor we are considering as our label and get its regression coefficient to the anchor u_i(target). Then our loss is

$$\begin{aligned} L &= \frac{1}{N} \sum_{\text{foreground}} \left[\sum_i S\left(u_i(\text{predicted}) \quad u_i(\text{target})\right) \right], \\ \text{where} \quad S(x) &= \begin{cases} \dfrac{\sigma^2 x^2}{2} & \text{when } \textit{kxk} < 1/\sigma^2 \\ \textit{kxk} \quad 0.5/\sigma^2 & \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

Here we choose $\sigma = 3$ and N is the total number of chosen foreground anchors.

Since RPN and the last classifier layer all shares the base convolution layers, the backpropagation will backpropagate to the base layers. To save computational resource, we fix the layer weights for the first few layers in the base network. In the VGG16 base case and we fixed the layer weights before the conv3 layer. On the other hand, in the Res101 case, we fixed the first convolutional layer and the first batch normalization layer.

Our code allows ADAM optimizer as well as SGD optimizer. However, we mainly focus on the SGD optimizer with decreasing learning rate.

2.3 Hyper-parameters

Above is the basic structure of our network, and we will introduce some important hyper-parameters such as the anchor initialization, the maximum number of bounding boxes kept and learning rate decay. We initialize our anchors on each pixel of the feature map got from the base net. This corresponds to put anchors on every 16 pixels on each dimension. For each pixel of the feature map, there will be 9 anchors initialized. The 9 anchors are generated by three different scale and three different height-width ratio. Our scales are 8, 16 and 32. Our ratios are $1:2$, $1:1$ and $2:1$.

Also, in training, we need to use the target bounding boxes as labels to train our RPN. We choose the anchors whose overlap with ground truth box is greater than RPN_POSITIVE_OVERLAP $= 0.7$ as foreground label and whose overlap with any ground truth box is lower than RPN_NEGATIVE_OVERLAP $= 0.3$ as negative

Net	Optimizer	step size	decay epoch	total epoch	batch size	# GPU
VGG16	SGD	4.0 10^{-3}	8	12	4	2
Res101	SGD	4.0 10^{-3}	8	10	4	2

Table 1: Training hyperparameter for `VGG16` and `Res101`

label or background label. Those who does not fulfill any of the two conditions are not considered. Thus, we have chosen the the appropriate number of anchors to train the `RPN` network. With the reduced number of anchors, we will have reduced number of `ROIs` which are bounding boxes generated from the chosen anchors by applying transformation calculated by `RPN`. However, we will choose `ROIs` whose overlap with the foreground higher than `FG_THRESH` $= 0.5$ as foreground `ROIs` and use those `ROIs` to do crop pooling and then get classifier's classification loss and bounding box regression loss.

Although we have many other hyperparameters, we do not list all of them here. Table 1 shows the training hyperparameters for two nets we trained, `VGG16` and `Res101`. Since we also did some other measurements and comparison with the original paper[1], we changed the above mentioned hyperparameters in those experiments and the particular choice will be mentioned when we introduce the experiment results.

3 Experiments

In this section, we will introduce all experiments we did with our developed Faster R-CNN code. We will show that the loss decreasing trend with trained epochs, the mean Average Precision (mAP) of two trained model (`Res101` and `VGG16`). We will also compare those results with the paper's results as well as the image detection time. We will also compare settings of anchors as did in the original paper [1].

3.1 Dataset

In the original paper, the authors did experiments on PASCAL VOC 2007, 2012, and COCO detection benchmarks. While in our project, we choose only to do experiments on the smaller dataset PASCAL VOC 2007, due to the time limit. In total this dataset consists of 9,963 images (containing 24,640 annotated objects) over 20 object categories. It has been split into 50% for training and 50% for testing. The distributions of images and objects by class are approximately equal across the training and test sets. [3]

3.2 Loss

As mentioned in section 2.2, we trained the network by decreasing the total loss. In Figure 1 in the next page, we plotted the total loss and the four components for a training session on net `VGG16` for 20 epochs. In this training session, we used `SGD` optimizer and the initial step size is 1.0 10^{-3}. The step size decrease by a factor of 10 for every 5 epochs. This is our first trial and thus the step size schedule may not be optimal since the decrease of rpn box loss and rcnn box loss is not dramatic.

3.3 mAP & Rate

Table 2 shows the two most important measures of our best trained models. The first measure is mAP, which is the most common metric for object detection. The second is the processing rate when testing the model. The test processing is performed on two Nvidia 1080 Ti GPUs in parallel. As can be seen, the net with `Res101` achieves higher accuracy than `VGG16` but at the expense of slower processing time.

Net	mAP(%)	rate
VGG16	68.7	18fps
Res101	74.1	14fps

Table 2: mAP and Time on PASCAL VOC 2007 test set.

The accuracy of detection of `VGG16` is comparable with the original paper result [1]. In that paper, the published number is 69.9. This is considerable since in the original paper, the net was trained for 240k

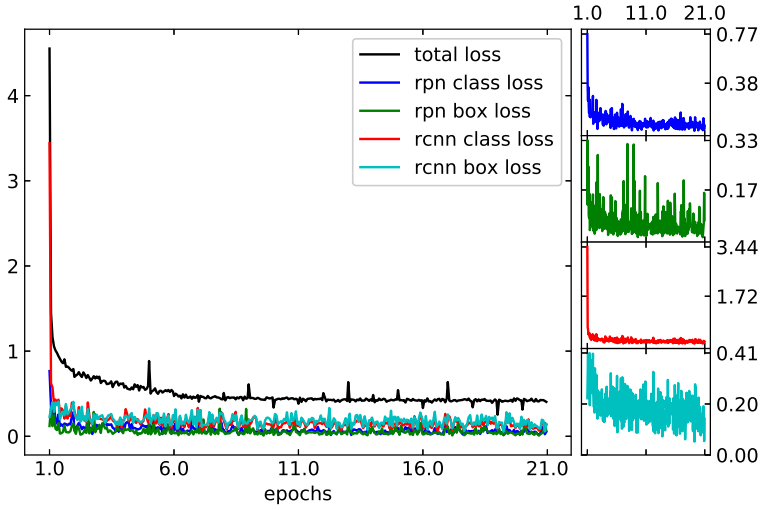

Figure 1: Total loss and its four components vs. trained epochs for `VGG16`

iterations which is approximately 50 epochs for VOC 2007 and our net was trained for 12 epochs. Besides, our detection time is also considerably good since in the original paper, the reported detection rate is 17 fps and we achieve similar number and even higher.

Table 3 shows mAP for our best trained model on different classes of objects. From the table, we can see that even though in general, `Res101` does better than `VGG16`, on detecting bus and cars, `VGG16` does better. Also, the two kinds of net share same preferences on classes. For example, both of them are not good at detecting chair potted plant and boat.

Net	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
VGG16	68.7	67.0	78.2	65.8	52.2	51.6	79.8	84.4	78.2	45.4	78.4	61.1	78.8	79.3	74.8	77.9	44.4	66.8	64.6	75.1	70.3
Res101	74.1	70.4	80.1	78.2	64.1	57.6	78.8	79.8	87.2	54.3	82.2	71.1	85.6	86.3	79.4	78.9	48.1	76.8	74.9	77.2	71.9

Table 3: Results on PASCAL VOC 2007 test set with Fast R-CNN detectors and VGG16 or Res101.

3.4 Comparison of different settings of anchors

This comparison is performed using `VGG16` pretrained net and Fast R-CNN detector. As suggested by the original paper, Faster R-CNN is sensitive to different setting of anchors. We did the experiment on the choice of the combination of one or three scales and one or three ratios. The detailed results are showed in the following table. (Note that the scale is relative to the original image instead of feature map) We got similar results as the paper: the settings of (3 scales, 1 ratio) and (3 scales, 3 ratios) have similar mAP, while the setting (1 scale, 1 ratio) has the worst mAP.

settings	anchors scales	anchor ratios	mAP (%)
1 scale, 1 ratio	128^2	1:1	64.7
1 scale, 3 ratios	128^2	2:1, 1:1, 1:2	66.9
3 scales, 1 ratio	128^2,256^2,512^2	1:1	68.4
3 scales, 3 ratios	128^2,256^2,512^2	2:1, 1:1, 1:2	68.7

Table 4: mAP and Time on PASCAL VOC 2007 test set.