**Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks**

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

* Source: <https://arxiv.org/pdf/1506.01497.pdf>

***Comparison of R-CNN, Fast R-CNN, and Faster R-CNN:*** *(This is not part of the paper)*

* *R-CNN uses Selection Search to generate regions of interest. These regions are then processed by CNN model one-by-one. The output of this CNN model (i.e. a conv. feature map) is then fed to class-specific SVMs for classification. The output of this CNN is also fed to a linear regressor that outputs bounding boxes for all the k-classes per RoI.*
* *R-CNN had three training stages: one for CNN, one for SVMs, and one for bounding-box regressor.*

*This was time and memory consuming. Moreover, each RoI was processed by CNN one at a time, which was very time consuming because each image had around 2000 RoIs.*

*So, Fast R-CNN fixed these issues.*

* *Fast R-CNN takes as input a whole image and RoI proposals. The CNN model processes* ***the whole image*** *in one go, rather than processing each RoI separately.* ***The region proposals are applied on the conv. feature map produced by the CNN.***

*The RoI layer takes each RoI* ***from the conv. feature map*** *and applies max-pooling. This is then passed through a couple of FC layers. Finally, the output of these FC layers is passed to two branches; one having a FC layer and a softmax layer for class prediction and the other having two FC layers for bounding box prediction.*

* *While Fast R-CNN is faster than R-CNN during training as well as prediction, the region proposal stage is still time consuming.*

*So, Faster R-CNN gets rid of the Selection Search algo*

* Since region proposal becomes a bottleneck in R-CNN and Fast R-CNN, Faster R-CNN introduces Region Proposal Network (RPN).
* RPN shares conv. layers with the detection network, and thus region proposal becomes nearly cost-free.
* First, we have few conv. layers that generate a conv. feature map. We construct RPN by adding few more **conv. layers** on top of this feature map. These conv. layers generate region bounds and objectness scores simultaneously for each location of the conv. feature map.

**So, RPN is basically a fully convolutional network.**

* To deal with objects of various scales, Faster R-CNN uses anchor boxes.
* Using RPN, the time for region proposal becomes 10 ms/image
* Since RPN learns to generate regions from data, it can benefit from using deeper and expressive network.
* Faster R-CNN has two modules: one is a fully conv. network that produces regions and the second is the **Fast R-CNN** detector that uses these proposed regions.
* RPN takes a whole image as input and outputs a set of rectangular object proposals, each with an objectnees score.

Objectness score specifies whether there is an object (of any type) or not in the proposal and how confident the model is about the prediction.

* To summarize:

We have few conv. layers that take a whole image as input and generate a conv. feature map.

Now, this feature map is passed to RPN for generating object proposals. The same conv. feature map is also passed to RoI which uses the above object proposals to classify and predict bounding boxes.

* To generate region proposals, we slide a small network over the conv. feature map generated by the last shared conv. layer. This network takes as input n\*n windows of the feature map (n=3 is used in the paper)

Each sliding window is mapped to a lower dimensional feature (256-d for ZF net and 512-d for VGG net)

This lower-dimensional feature map is then fed into two sibling FC layers – a box regression layer (which specifies the region proposal) and a box classification layer (which specifies the confidence score)

The above is implemented using a n\*n conv. layer followed by two sibling 1\*1 conv. layers.

*(So, in case of ZF net, the first RPN specific conv. layer must contain 256 filters of shape n\*n. It will produce an output of 1\*1\*256* ***per window****. So, the overall output shape of the first layer would be x\*y\*1\*1\*256 (may be, this is then converted to x\*y\*256; assume x\*y is the no. of windows formed of shape n\*n during sliding). This output will then be passed to a 1\*1 conv. layer having 2k filters as well as to 1\*1 conv. layer having 4k filters)*

* Anchors:

At each sliding window location, we propose k regions. So, the reg layer has 4k outputs encoding the coordinates of k bounding boxes and the cls layer has 2k outputs that give probability of object or not object for each proposal.

These k proposals are given relative to k reference boxes, which are called Anchors.

So, an anchor is basically a box of particular shape and aspect ratio that fits the objects in the task.

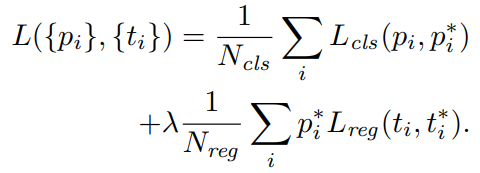
In this paper, 3 scales and 3 aspect ratios are used. So, there are 9 anchors. Thus, each sliding window location will output 9 bounding boxes.

(The anchors were manually decided)

* The no. of boxes that the model predicts per sliding window location is same as the no. of anchors that we use; i. e. there is one-to-one association between the no. of outputs per sliding window location and the anchor boxes.
* The region proposals are relative to anchors.
* For training, each anchor is assigned a binary label. We assign +ve to two kinds of anchors:

1. The anchor (s) with the highest IOU overlap with a ground-truth box
2. An anchor that has an IOU overlap > 0.7 with any ground-truth box.

* An anchor is labeled -ve if IOU < 0.3 for all ground-truth boxes.
* Anchors that are neither +ve nor -ve don’t contribute to the loss.
* Loss function:



is the index of an anchor/bounding box in a mini-batch and is the predicted probability that the predicted bounding box associated with this anchor is an object (or contains an object)

is 1 if the anchor is +ve and 0 if it is -ve

is a vector representing the predicted bounding box **associated with this anchor** and is a vector of the ground-truth box **associated with the anchor.** Note that the second loss term contains , which means this term will be calculated only for the anchors which are +ve.

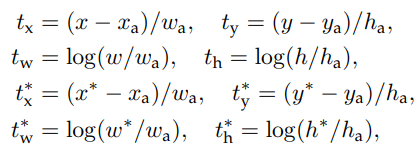
is the log loss

, where R is smooth L1 function used in Fast R-CNN paper.

The two terms are normalized using (i.e. batch size) and (i.e. the no. of anchor locations)

is used, but it is proved that you can use any lambda value in a particular range without making much difference

* For bounding box regression, we use below:



and are for predicted box, anchor, and ground-truth box respectively. Similarly for y, w , and h.

* The features used for regression are of the same spatial size (3\*3). The model is able to detect objects of varying shapes and aspect ratios due a set of k bounding-box regressors (or in other words, k anchors).
* Training RPN:

A mini-batch is created using a single image that contains many +ve and -ve anchors. Generally, there will be more -ve anchors in an image, so using all the anchors (+ve and -ve) for training will lead to bias. To deal with this, randomly select 256 anchors in the image, where +ve to -ve ratio is up to 1:1

The shared layers are created using a pre-trained ImageNet model (e.g. VGG, AlexNet, etc.)

For the newly added layers, weights are initialized using zero-mean Gaussian distribution with std 0.01.

Learning rate 0.001 for 60k batches and 0.0001 for the next 20k batches

Momentum: 0.9

Weight decay: 0.0005

* For the detection network, Fast R-CNN is used. So, the training and all other details are as specified in [Fast R-CNN](../2.%20Fast%20R-CNN/Summary.docx). The only change would be that the regions that Fast R-CNN uses for detection will now come from RPN rather than Selection Search.
* Paper gives three training techniques. Below is the one that is used.

In the first step, we train RPN. We start with a pre-trained ImageNet model and fine-tune it for region proposal task.

In the second step, we start with one more pre-trained ImageNet model and convert it into Fast R-CNN. We fine-tune this model for object detection task using the regions proposed by the above trained RPN model.

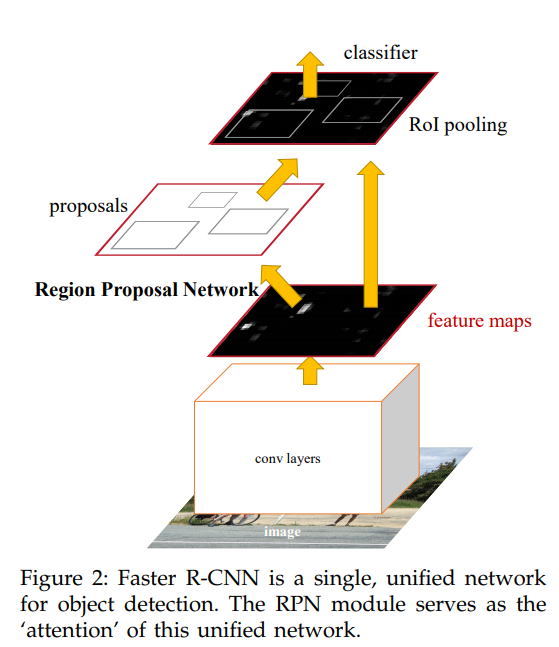
Now, we have two separate networks.

In the third step, we initialize RPN network using detector network. We fix the shared conv. layers and fine-tune only the RPN specific layers.

Now, the models share layers.

In the fourth step, we fine-tune the Fast R-CNN specific layers without affecting the shared conv. layers.

You may fine-tune this shared network further using alternate training explained in the third and fourth steps above, but generally it is not required as more training doesn’t change the accuracy much.



* Anchors used:

Box areas of 128, 256, and 512 pixels; and 3 aspect ratios of 1:1, 1:2, and 2:1

These hyperparameters are not chosen based on the data but are selected manually.

* Some RPN proposals highly overlap with each other. To reduce redundancy, use non-max suppression per class.

IOU threshold used for non-max suppression is 0.7

Refer <https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46> for more detailed, low-level explanation. Also go through comments.

TODO – <https://www.telesens.co/2018/03/11/object-detection-and-classification-using-r-cnns/>

Refer <https://towardsdatascience.com/understanding-fast-r-cnn-and-faster-r-cnn-for-object-detection-adbb55653d97>, if required.