**Fast R-CNN**

*Ross Girshick*

*RoI – Region of Interest*

* Source: <https://arxiv.org/pdf/1504.08083.pdf>
* This algorithm is an improvement over R-CNN (Region-based CNN).
* Compared to R-CNN, this is 9 times faster during training and 213 times faster during testing.
* It gives better mAP as well.
* R-CNN has below drawbacks:
  + *Training is a multi-stage pipeline:* R-CNN first fine tunes a CNN on object proposals using log loss. Then, it fits SVMs to CNN features. These SVMs act as object detectors, replacing the softmax classifier learnt by fine-tuning the CNN. In the third training stage, bounding-box regressors are learnt.
  + *Training is expensive in space and time:* For SVM and bounding-box regressor training, features are extracted from each object proposal in each image and written to disk. With very deep networks, this process requires high-end GPU, lot of time, and hundreds of gigabytes of storage.
  + *Object detection is slow:* At test-time, features are extracted from each object proposal in each test image, so detection is very slow.

i.e. R-CNN is slow because it performs a CNN forward pass for each object proposal, without sharing computation.

* Advantages of Fast R-CNN:
  + Higher detection quality (mAP) than R-CNN, SPPnet
  + Training is a single-stage process, using a multi-task loss
  + Training can update all network layers
  + No disk storage is required for feature caching
* Fast R-CNN takes as input a whole image and a set of object proposals. (Note that the object proposals or Regions of Interest (RoI) are not processed by the CNN; they are used by the RoI layer.)
* CNN processes the whole image and produces a conv. feature map.
* <https://towardsdatascience.com/understanding-fast-r-cnn-and-faster-r-cnn-for-object-detection-adbb55653d97>

Selection Search algorithm generates regions of interest (RoI) from an input image. In R-CNN, we feed these regions to the CNN. However, in Fast R-CNN, we first process the input image using a CNN and then apply RoIs to the conv. feature map.

In order to do that, we need to understand Sub-sampling ratio. It is the ratio of the conv. feature map size to the input image size.

e.g. If the input image shape is 224\*224 and the conv. feature map generated by CNN is of shape 8\*8, the sub-sampling ratio is 1/28 (=8/224).

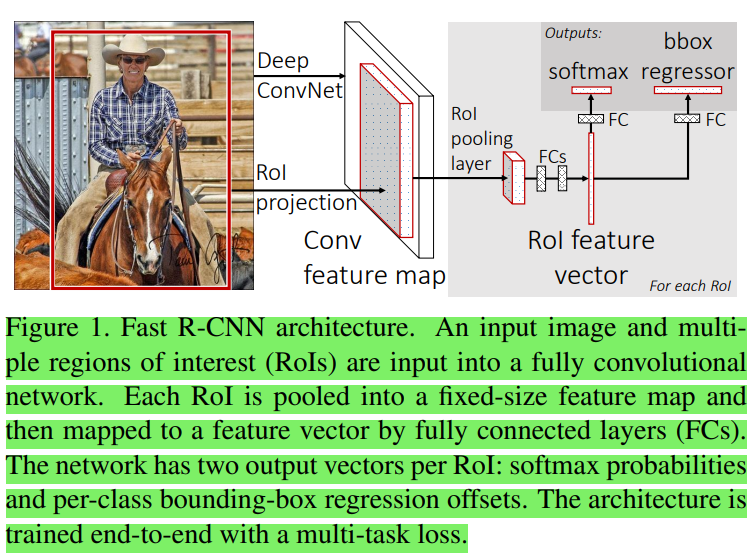
Now, we use this sub-sampling ratio on input RoIs to get RoIs for the conv. feature map; i.e. we project input RoIs on to the conv. feature map. These new RoIs, now, can be used by RoI pooling layer to extract various regions of the conv. feature map.

* For each object proposal, the RoI layer extracts a fixed-length feature vector from the conv. feature map.
* These feature vectors are fed to one/more FC layers, whose output is then given to two branches.
* The first branch contains one FC layer followed by a sofmax layer for the k+1 classes, where k is the no. of classes that you actually care about and the extra 1 is for background.
* The second branch contains a FC layer followed by another FC layer that outputs 4 real-valued numbers **for each of the K object classes** (so, total 4k output units). Each set of 4 real-valued numbers is the bounding box for one of the k object types.
* RoI Pooling Layer:
  + It is a max-pooling layer that converts input region to a fixed-size feature map of shape H\*W
  + An RoI is the region in the conv. feature map given by a tuple (r, c, h, w), where (r, c) specifies the top-left corner and (h, w) specifies the height and width of the proposed region.
  + RoI layer divides its input of shape (h, w) into a grid of shape H\*W. So, the approximate shape of each sub-window in the input will be h/H \* w/W

(Since regions can be of various lengths h/H and w/W may not be integers. In such scenario, we round the result to the nearest integer.

So, it is not required for all sub-windows to have same size.)

The layer then applies max-pool to each such sub-window to get the value of one cell in the output.



* CNN used for extracting features:
  + Experimented with various CNN having 5 max-pooling layers and between 5 to 13 conv layers.
  + Used pre-trained networks, by replacing the last max-pool layer by a RoI pooling layer.
  + Configured H\*W in such a way that the output of the RoI layer is compatible with the first FC layer in the CNN.

(Clarification: In a CNN model, we have a sequence of (one or more) conv - max-pool layers. After the last max-pool layer, we have FC layers. So, the authors replaced the last max-pool layer by a RoI layer, and the FC layer just after this replaced max-pool layer is used to decide the correct values of H and W.

e.g. For VGG16, H=W=7

* + Then, the last FC layer and the softmax layer from the pre-trained CNN model are removed and two branches are created, with the first containing a FC layer and softmax layer and the second containing two FC layers.
  + CNN is modified to take two inputs: a list of images and a list of RoIs in those images
* The proposed technique takes advantage of shared computations:

A mini-batch is made from RoIs from N images and the no. of RoIs from each image is R/N, where R is the total no. of RoIs.

So, during forward and backward passes, RoIs from the same image share computations and memory.

One concern with this approach is that RoIs from the same image will make convergence slower, but practically nothing like this was observed.

* Authors used N=2 and R=128
* Used SGD to train the **entire** model in one stage, rather than training CNN, SVMs, and bounding-box regressors in separate stages like in R-CNN
* Loss function:

Let be the probability distribution over k+1 classes for each RoI.

Let specify offsets of the bounding box for each of the K object classes in each RoI

(Check [R-CNN Summary](../1.%20R-CNN/Summary.docx#APPENDIX_C) for understanding how tk ­­­is computed.)

Each training RoI is associated with a ground-truth class and bounding box

**(Note: There can be at max one object per RoI)**

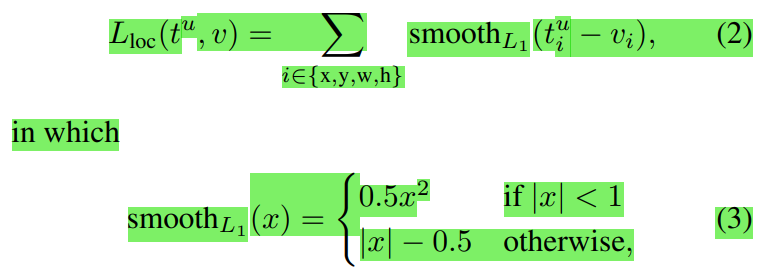
The total loss is given by:



where

 is the log loss for the ground-truth class

and

 is the loss for bounding box prediction.

evaluates to 1 whenever , otherwise 0

means the background class.

balances the two losses in the total loss.

Authors used

* During training, images are flipped horizontally with probability 0.5. No other augmentation was done
* Each mini-batch is constructed from N=2 images, chosen at random. Mini-batch size R=128, where 64 RoIs are from each image.
* SKIPPED: backpropagation in RoI pooling layer
* FC layers used in the class prediction branch and the bounding-box prediction branch are initialized using zero-mean Gaussian distribution with std 0.01 and 0.001 respectively. All biases are initialized to 0.

Momentum: 0.9 and weight decay=0.0005

* Detection:

The model takes as input an image and a list of R object proposals to score (R is typically 2000).

For each RoI , the forward pass outputs a probability distribution and a set of bounding-box offsets relative to (each object gets one bounding box)

Each bounding box will have a confidence score associated with it. This confidence score is nothing but the softmax probability for the class.

Then, non-max suppression is applied for each class independently (same as that used in [R-CNN](../1.%20R-CNN/Summary.docx))

* Skimmed through rest of the paper – not important