## Fast RCNN

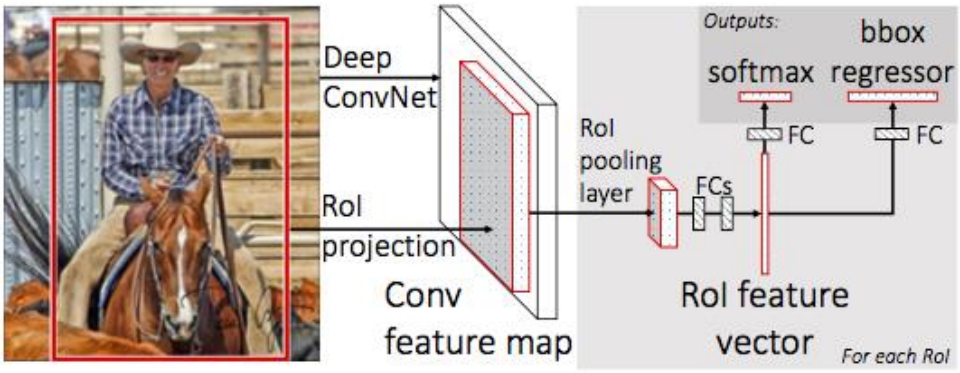
Fast RCNN is an updated variant of RCNN model architecture. Fast R-CNN was proposed as a new training algorithm that fixes the disadvantages of R-CNN and SPPnet. The key contributions of this network are: training in single stage pipeline, higher detection quality (mAP) compared to R-CNN and SPPNet, no disk space required for feature catching, and training can update all the layers. The following blogs have an overview of this network that can help to get basic concepts:

- https://medium.datadriveninvestor.com/review-on-fast-rcnn-202c9eadd23b
- https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022
- https://www.geeksforgeeks.org/fast-r-cnn-ml/
- https://pantelis.github.io/aiml-common/lectures/scene-understanding/object-detection/faster-rcnn-object-detection/index.html
- https://docs.kanaries.net/topics/Python/fast-rcnn

It is recommended (not mandatory) to gain basic understanding of spatial pyramid pooling and SPPNet before start reading/working about Fast R-CNN. Because, the Fast R-CNN architecture addressed some of the challenges and limitation that lies in R-CNN and SPPNet models. This network introduces RoI pooling layer to perform max pooling operation, which converts the features inside any valid region of interest into a small feature map with a fixed spatial extent of H ×W (e.g., 7×7). To understand the concepts of RoI pooling, go through this blog:

- RoI Pooling

**Architecture overview**




A Fast R-CNN network takes as input an entire image and a set of object proposals. The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map. Then, for each object proposal a region of interest (RoI) pooling layer ex tracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a catch-all “background” class and another layer that outputs four real-valued numbers for each of the K object classes. To develop a Fast R-CNN network, we can utilize pre-trained models like AlexNet, VGG-16, ResNet-50 etc. When a CNN model (scratch or pre-trained) initializes a Fast R-CNN network, it undergoes three major transformations:

- The last max-pooling layer is replaced by a RoI pooling layer, which will be compatible with the input of first FC layer. For example, if we use VGG-16 network, then the RoI pooling layer should be output 512x7x7 (512 is the number of feature maps) feature maps, which is the expected input of the first FC layer.
- Network's fully connected layer and softmax layer will be replaced with two sibling layers: a FC layer and softmax over K+1 categories and category specific bounding box regressor.
- The network is modified to take two inputs: a list of images and a list of RoIs of those images.

Fast R-CNN uses a streamlined training process with one fine-tuning stage that jointly optimizes a softmax classifier and bounding-box regressors, rather than training a softmax classifier, SVMs, and regressors in three separate stages. You can find the main paper here.

**Implementation**

- To train Fast R-CNN model, we need to create a custom dataset from PASCAL VOC or any other dataset, which will contain image and their corresponding proposed regions, class labels, ground truth bounding boxes and proposed regions bounding boxes. We can save this information in a pickle file, json or any other format.
- To get proposals, we will apply selective search algorithm to each input image that will generate certain numbers of proposed regions for that image.
- The input images were converted into 224x224 before passing through the backbone. The proposals are the coordinated of bounding boxes after the selective search, where each proposal contains 4 values (x1, y1, x2, y2).
- For developing model, we need a backbone for feature extraction, a RoI pooling layer after the convolution, a classification head for predicting class, a regression head for predicting the bounding boxes, a multi-task loss function to optimize model for both classification loss and regression loss.
- For backbone, we can use any pre-trained CNN models that will generate feature maps and pass to the RoI pooling layer.
- The RoI pooling layer takes feature maps and region of interest (proposals) as input. After that, it returns the pooled features to the classification and regressor heads.
- We need to develop a multi-task loss to optimize our model. We will use cross-entropy loss for classification and for smooth L1 loss for the regression. For bounding box regression, we will use only those boxes that contain object according to model prediction.
- We can use dropout for regularization and learning rate scheduler to reduce learning over the epochs. (In my experiment, I have used dropout probability of 0.25 in both classifier and regressor, and learning rate scheduler after 10 epochs with a factor of 0.1.
- Finally, we will train our model with an optimizer and save the model after completing the training process.

**Observation**

I have followed the original paper for the implementation and trained it for 20 epochs that shows a training accuracy of 98.29% for class prediction. However, there are some several behaviors has been observed during the training and faced issues during the inference.

- After few epochs, the training accuracy did not improve, it was stuck. May be the model reached to its local minima.

- Although the loss values are smaller but it did not reduce constantly with improvement, which is another crucial issue.
- Model prediction is biased, it always predicts the background and all of the predicted boxes are same. That means the model suffers from highly data imbalance problem and overfitting issue. It suffers from generalization, sort of did not learn that much to predict unknown samples.
- However, my model was trained once without any fine tuning. If we overview the model architecture again, address these problems and fine tune, the model should provide us its expected behavior for object detection.

**Codes and Papers**

- https://github.com/msuhail1997/Faster-RCNN-Pytorch_Object_Detection/blob/main/Fast_RCNN_FINAL.ipynb
- https://paperswithcode.com/method/fast-r-cnn

Compare RCNN families

- https://www.sefidian.com/2020/01/13/rcnn-fast-rcnn-and-faster-rcnn-for-object-detection-explained/

Built-in Implementation (Pytorch)

- https://github.com/ashishsalunkhe/Understanding-RCNN-Fast-RCNN-Faster-RCNN-and-Mask-RCNN/blob/master/FasterRCNN_pretrained.ipynb

Faster RCNN training with pre-trained models using PyTorch

- https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html