Rachit Srivastava

[rachitraj21@gmail.com](mailto:rachitraj21@gmail.com)

8317031801

**MY APPROCH**

# This custom object detector has been created with the help of Tensorflow Object Detection API. It has been trained to detect just 3 categories of objects

# Person

# Bird

# Animal

# Note:

# Under the ‘Animal’ category , I have taken only 3 animals into consideration i.e. dog, cat and horse (due to lack of GPU which led to increase in computational cost and quick exhaustion of memory resources).

BEST MODEL USED – FASTER RCNN(MODEL A)

SECOND BEST MODEL USED – RCNN(MODEL B)

**Question1:**

**Give a solid reason why you chose this model(say A). Any second best model’s (say B) name (in your choice). Why A is better than B (in your words)?**

**Answer1:**

A naive approach to solve Object Detection problem would be to take different regions of Interest from the image, and use a CNN to classify the presence of the object within that region. The problem with this approach is that the objects of interest might have different spatial locations within the image and different aspect ratios. Hence, you would have to select a huge number of regions and this could computationally blow up.

In RCNN,to bypass the problem of selecting a huge number of regions, [Ross Girshick et al](https://arxiv.org/pdf/1311.2524.pdf) proposed a method where we use selective search to extract just 2000 regions from the image and he called them region proposals. Therefore, now, instead of trying to classify a huge number of regions, you can just work with 2000 regions. These 2000 candidate region proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an [SVM](https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47) to classify the presence of the object within that candidate region proposal.

PROBLEMS WITH RCNN:

* It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
* It cannot be implemented real time as it takes around 47 seconds for each test image.
* The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

SO CAME IN RESCUE THE FASTER RCNN:

The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.

Difference between RCNN and Faster RCNN:

The reason “Fast R-CNN” is faster than R-CNN is because you don’t have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it. The Faster R-CNN not only brings down the region proposal time from 2s to 10ms per image but also allows the region proposal stage to share layers with the following detection stages, causing an overall improvement in feature representation.

**Question 2:**

**How you trained the weights?**

**Answer 2:**

I have created a train and test folder. Then using the transfer learning from the faster RCNN model , I started training my model .While training the model , there was a parameter named as loss that continuously displayed values that kept on decreasing up to a certain value which indirectly indicated that the weights are being continuously updating. I use SGD(Stochastic gradient descent ) optimization technique in this case. Since this custom object detector has been made from scratch and training was not done for longer duration (lack of GPU), so loss values were in the range between 0.3 to 1.5.

**Question 3:**

**Describe the model that you used in your own words.**

**Answer 3:**

I have used the Faster RCNN model for this custom object detector. In short the model does follow these steps:

* A region proposal algorithm to generate “bounding boxes” or locations of possible objects in the image
* A feature generation stage to obtain features of these objects, usually using a CNN
* A classification layer to predict which class this object belongs to
* A regression layer to make the coordinates of the object bounding box more precise.

Let’s expand the working of Faster RCNN model further:

* **Part 1 : Convolution layers**

In this layers we train filters to extract the appropriate features from the image, for example let’s say that we are going to train those filters to extract the appropriate features for a human face, then those filters are going to learn throughout training shapes and colors that only exist in the human face. Convolution networks are generally composed of Convolution layers, pooling layers and a last component which is the fully connected layer.

* **Part 2 : Region Proposal Network (RPN)**

RPN is small neural network sliding on the last feature map of the convolution layers and predict whether there is an object or not and also predict the bounding box of those objects. Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

* **Part 3 : Classes and Bounding Boxes prediction**

Now we use another Fully connected neural networks that takes as an input the regions proposed by the RPN and predict object class (classification) and Bounding boxes (Regression). To train this architecture, we use SGD to optimize convolution layers filters, RPN weights and the last fully connected layer weights