IE534 Final Project Report : Faster R-CNN Implementation

Group 16
Sanyukta Deshpande(spd4), Refik Mert Cam(rcam2)
Abhinav Garg(garg19)

May 14, 2021

1 Introduction

In this project, we have implemented the Faster R-CNN architecture [1]. Faster R-CNN is a state-of-the-art member of the R-CNN family; which is developed for detection of objects within an image. Its predecessors are Regions with CNN (R-CNN) and Fast R-CNN. The evolution between versions is motivated by the aim to increase computational efficiency and decrease the test time. The R-CNN family consists of a region proposal algorithm to extract the regions in an image where possible objects locate, a feature generation layer to extract the features of the objects, a classification layer to label the objects, and a fine-tuning layer to make the localization of the object preciser. The predecessors; R-CNN and Fast R-CNN, use CPU-based region proposal methods, which is the selective search algorithm that takes around 2 seconds per image. If the performance of the Fast-R-CNN during its test time is compared to the case where we don't use the region proposals, we see that there's a huge time gap, making region proposal the bottleneck of the algorithm.

Then, the Faster R-CNN replaces the selective search algorithm with a Region Proposal Network (RPN), which significantly increases the performance. Here, we reproduce the Faster R-CNN network schematic from [1], in Figure 1.

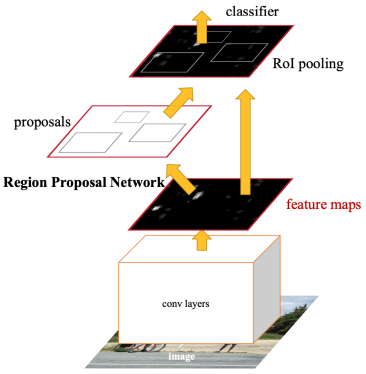


Figure 1: Faster R-CNN Network

Similar to Fast R-CNN, convolutional layers are used to extract the feature maps of the input image. Then, RPN is used to find the region proposals, which is a fully convolutional neural network that finds the object bounding boxes and objectness scores at all positions. The predicted region proposals are reshaped using the RoI pooling layer and then the classifer is used to detect the objects within the bounidng boxes. The RPN component provides "attention" and tells the unified network where to look. Furthermore, as the RPN shares its convolutional features with the object detection network, this innovation is almost cost-free. Compared to the previous algorithms used for object detection, Faster R-CNN performs significantly better, as depicted in Figure 2, from [2].

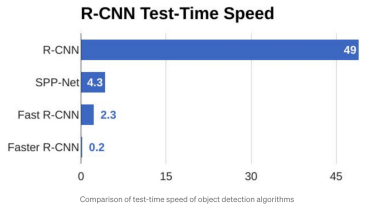


Figure 2: Comparison of various object detection algorithms

In Section 2, we discuss the detailed architecture of Faster R-CNN. In Section 3, we describe the specifications of our implementation and discuss the experiment results. We mainly analyze the difference between the usage of the following modules for the network : VGG16 and ResNET101. Moreover, we consider multiple anchor settings for training and describe the results. Finally, we conclude in Section 4.

2 Faster R-CNN : Functionality and Details

As shown in Figure 1, an image goes through the following in order:

- Region Proposal Network (RPN)
- Object Proposals
- RoI pooling
- Classifier and Regressor

Here are the details of the architecture :

1. The convolutional layers perform feature extraction from the image, where the VGG16/ ResNET101 module is used. Later, we also study the performance analysis of VGG16 and ResNET101.
2. Further, anchor boxes are created with different size scales and ratios on each pixel. The boxes are then assigned labels and locations depending on their IoU (intersection over union) values. We reproduce a figure from [3] here, in Figure 3, which shows the scales and aspect ratios of Anchor boxes.

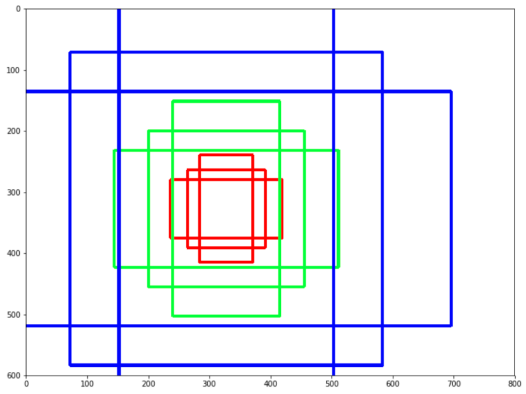


Figure 3: Scales and Aspect ratios of Anchor Boxes

3. Using anchor labels and targets as inputs, RPN network generates objectness score predictions, going through steps in Figure 4, from [4].

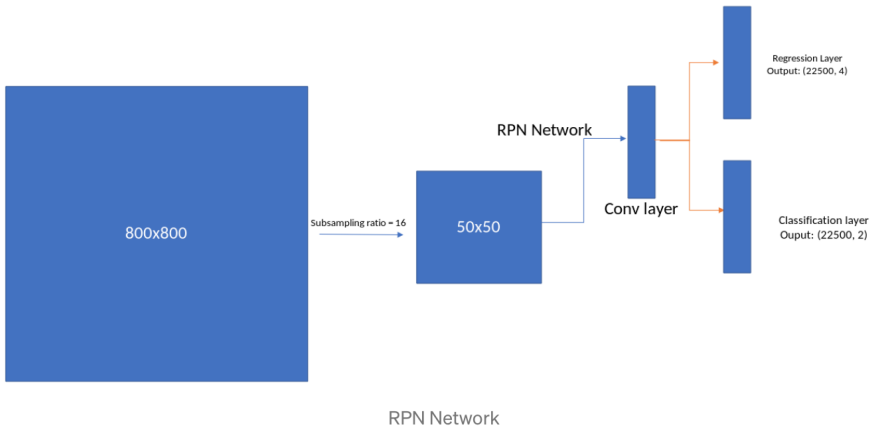


Figure 4: Regional Proposal Network

4. Top locations and their objectness scores are then considered to generate the region of interest proposals. Non Maximum Suppression (NMS) is used on the proposal regions to avoid the redundancy coming from overlapping region proposals and to keep the proposals with only the highest objectness scores.

5. In the next step, the final region proposals, ground truth boxes and labels are passed to the Fast R-CNN for generating proposal targets. This is again done by computing the IoUs. Anchor boxes with IoU scores greater than 0.7 are considered as foreground objects and those lesser 0.3 are considered as background.

6. Fast R-CNN then uses the Region of Interest (RoI) pooling to obtain fixed uniform sized feature maps. This increases the speed since it is then possible to use the same

feature input map. We can then pass the feature maps to the last classifier layer to get bounding box regression coefficients.

7. Classification and training losses for RPN and Fast R-CNN are computed using the following loss functions [4]:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1] * L_{loc}(t^u, v)$$
$$L(p_i, t_i) = \frac{1}{N_{cls}} * \sum_i L_{cls}(p_i, p_i^*) + \lambda * \frac{1}{N_{reg}} * \sum_i p_i^* L_{reg}(t_i, t_{i*})$$

Also, the cross entropy loss is computed for classification and final label is assigned based on softmax values:

$$H(y) = (-1) * \sum_i y_i * log(y_i)$$
$$softmax(l_i) = \frac{e^{l_i}}{\sum_k e^{l_k}}$$

For each iteration, both RPN and R-CNN losses are combined to find the total loss. Faster R-CNN can be trained by using ResNET101/VGG16 or any other classification network architecture.

8. The paper discusses three types of training procedures. The first one is a 4 step alternating training scheme, where RPN and R-CNN convolutional layers are alternatively trained, using input from the previous step. The second one is called approximate joint training, and it uses backpropogation to decrease the sum of all losses. Finally, the third procedure, non-approximate training, is similar to the second, but it computes the derivatives with respect to the proposal boxes' coordinates, which is ignored in the second.

3 Faster R-CNN Implementation

In this section, we describe the specifications for implementing the Faster R-CNN architecture and further discuss the results. We consider two different backbone networks for training the network and also various anchor scales and ratios for comparison. We mainly analyze our results using the following performance measures :

- Loss curves : Total loss, ROI classification loss, ROI regression loss, RPN classification loss, RPN regression loss.
- mAP (mean average precision) : both for different classes and over the general dataset
- Sample outcomes

3.1 Specifications

We have implemented the Faster R-CNN with the following specifications. [1]

1. Backbone network selection : As the backbone CNN architecture; which is shared by both object detection network and region proposal network, we have used VGG-16 and ResNET101 architectures as two different cases.

2. Dataset selection : The original paper uses PASCAL VOC 2007, 2012 and COCO datasets. We have used PASCAL VOC 2007 dataset; which contains 9963 images and 24640 annotated objects in total.

3. Training strategy : The original paper proposes three different training strategies, as mentioned earlier: (i) alternating training, (ii) approximate joint training, and (iii) non-approximate joint training. We have used strategy (ii) in our implementation.

4. Scale and aspect ratio of anchors : In the original paper, they use 3 scales and 3 aspect ratios. We have explored 4 different combinations (1,3 scale, 1,3 ratio) in our implementation.

3.2 Experiments

We now discuss all of our experiments on the PASCAL VOC 2007 dataset. Firstly, we look at the loss curves for VGG16 and ResNET101, trained over 20 epochs for various anchor scales and sizes. Then, we show the mean average precision (mAP) values for both the networks and also compare those with the values in the original paper. Finally, we consider a few images and perform R-CNN object detection on those using both VGG16 and ResNET101, and compare the results.

3.2.1 Loss curves

Here we have plotted the loss curves vs. number of epochs for backbone architecture selections of VGG-16 and ResNET101 and 4 different anchor scales and ratios. As mentioned earlier, we are using approximate joint strategy for training, where we aim to decrease the total loss through backpropogation. We have used online learning approach; thus, parameters are updated after each forward and backward pass. We selected stochastic gradient descent as the optimization method and used $1e-3$ as initial learning which decays to $1e-4$ after 8th epoch.

According to our experiments, the Faster R-CNN with ResNET101 backbone achieves lower total loss; however, when we evaluate the mAP over test dataset it performs worse. We suspect that a possible reason is overfitting.

1. Anchor scales=[128^2, 256^2, 512^2], Anchor ratios=[2:1, 1:1, 1:2]

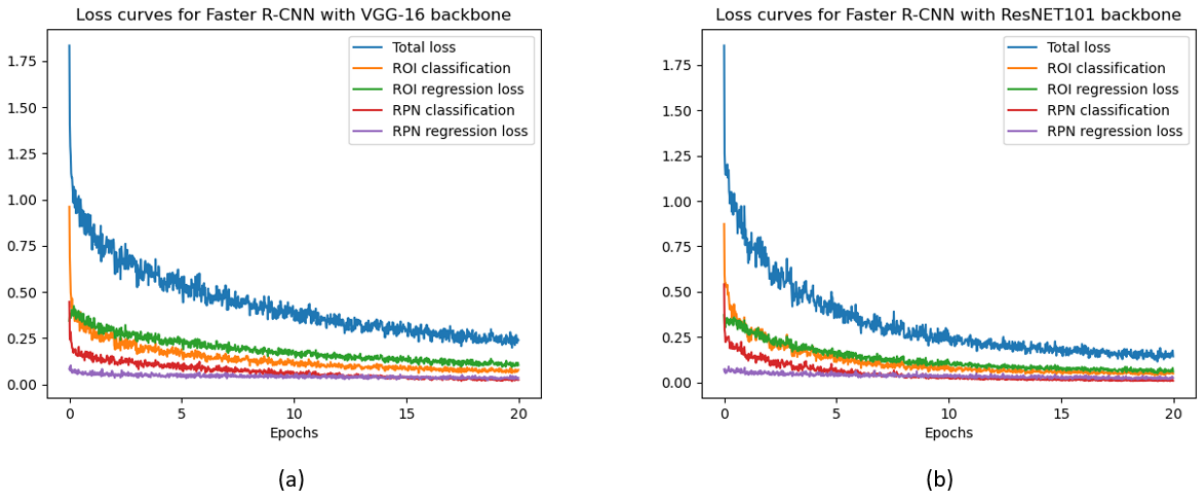


Figure 5: Loss vs. iterations for VGG-16 and ResNET101

2. Anchor scales=[128^2, 256^2, 512^2], Anchor ratios=[1:1]

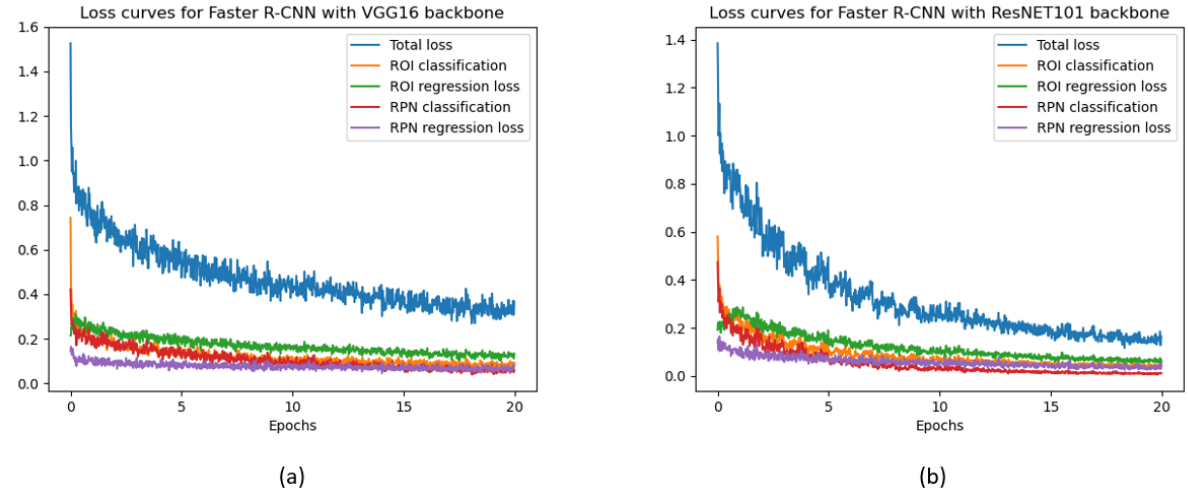


Figure 6: Loss vs. iterations for VGG-16 and ResNET101

3. Anchor scales=[128^2], Anchor ratios=[2:1, 1:1, 1:2]

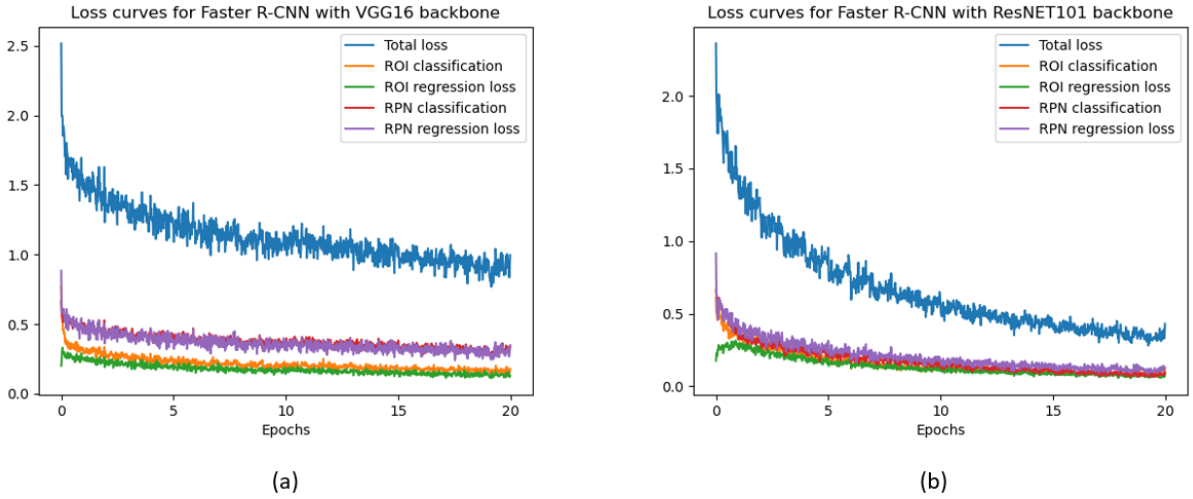


Figure 7: Loss vs. iterations for VGG-16 and ResNET101

4. Anchor scales=[128^2], Anchor ratios=[1:1]

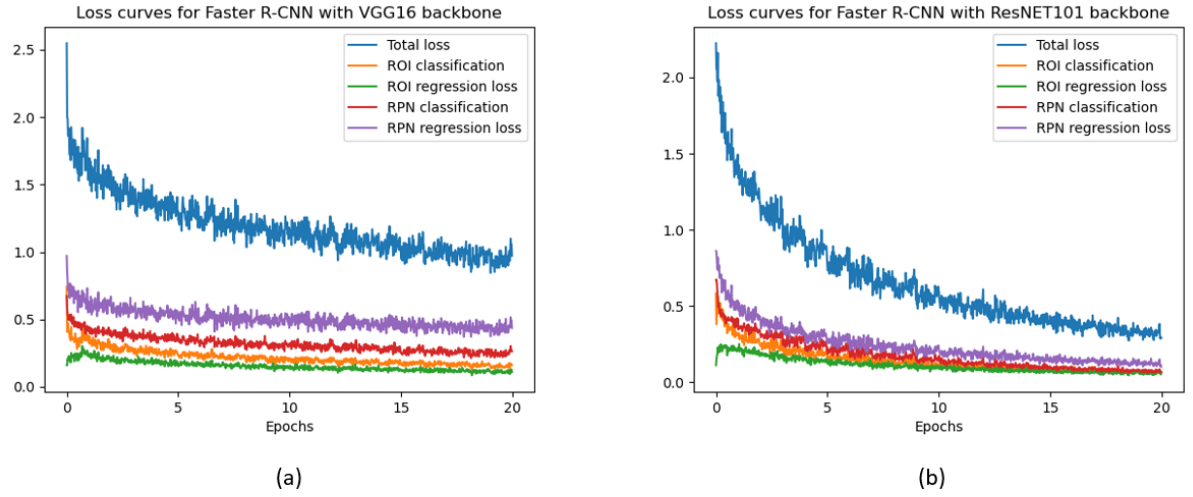


Figure 8: Loss vs. iterations for VGG-16 and ResNET101

3.2.2 Mean Average Precision (mAP)

Here, we show the Mean average precision (mAP) values for both backbone networks with 4 different anchor scales and ratios, which is the most commonly used metric for object detection. As we can see, the original paper records mean average precision of 69.9. Comparing the experiment results with the paper, we note that the results give slightly less values, which can be attributed to the selection of learning rate and decay choice and number of epochs used for training. The original paper trains the network for 240k iterations i.e. approximately 50 epochs; whereas these results correspond to training for 20 epochs. Our

experiments suggest a slightly worse mAP when Res101 is chosen as the backbone architecture. We also tabulate mAP values for various objects. As we can see, for most objects, VGG16 gives better accuracy, although the values are comparable for some objects. For 'dog' and 'sofa', we can see that Res101 values are higher. We also compare the mAP values for various selections of anchor scales and ratios. For all the four settings, VGG16 values are higher. We expect the mAP values to increase with the number of anchor scales and ratios. Our experiments with VGG16 are consistent with this expectation. However, we don't observe a significant change when we play with the anchor scales and ratios for the Res101 architecture.

1. Anchor scales=[128^2, 256^2, 512^2], Anchor ratios=[2:1, 1:1, 1:2]

Backbone network	mAP (%)	Rate (testing mode)
VGG16	67.6	13 fps
ResNET101	61.4	13 fps
Original paper	69.9	17 fps

	fly	bike	bird	boat	pin	bus	car	cat	chair	cow	table	dog	horse	moto	person	plant	sheep	sofa	train	tv
VGG16	69.7	78.8	65.4	51.8	53.3	77.1	78.5	79.5	46.2	71.4	66.2	75.7	78.9	74.0	75.2	36.0	65.3	61.5	73.2	73.8
ResNET101	69.0	69.8	55.6	48.5	34.7	68.0	70.4	78.1	40.1	65.6	55.1	75.8	78.1	66.6	68.8	33.1	59.3	60.0	74.1	54.2

2. Anchor scales=[128^2, 256^2, 512^2], Anchor ratios=[1:1]

Backbone network	mAP (%)	Rate (testing mode)
VGG16	65.0	13 fps
ResNET101	61.8	13 fps
Original paper	69.8	17 fps

	fly	bike	bird	boat	pin	bus	car	cat	chair	cow	table	dog	horse	moto	person	plant	sheep	sofa	train	tv
VGG16	62.7	73.9	64.0	48.5	51.1	75.9	78.7	78.7	43.9	66.8	64.1	70.1	77.5	74.3	74.5	39.4	60.4	54.9	72.5	68.8
ResNET101	72.1	74.6	58.2	45.4	35.2	71.4	75.5	77.4	40.4	64.2	54.0	77.8	77.4	67.0	69.5	31.7	56.0	64.8	67.9	56.2

3. Anchor scales=[128^2], Anchor ratios=[2:1, 1:1, 1:2]

Backbone network	mAP (%)	Rate (testing mode)
VGG16	64.3	13 fps
ResNET101	61.7	13 fps
Original paper	68.8	17 fps

	fly	bike	bird	boat	pin	bus	car	cat	chair	cow	table	dog	horse	moto	person	plant	sheep	sofa	train	tv
VGG16	66.9	76.9	62.8	50.1	40.3	72.2	77.6	78.0	36.0	69.0	63.4	76.5	79.3	75.2	73.7	30.5	63.2	63.8	71.3	59.9
ResNET101	66.8	73.8	57.3	46.9	35.0	70.5	77.4	77.6	38.8	61.0	57.5	73.8	76.5	68.7	69.1	30.1	57.6	63.4	74.3	56.6

4. Anchor scales=[128^2], Anchor ratios=[1:1]

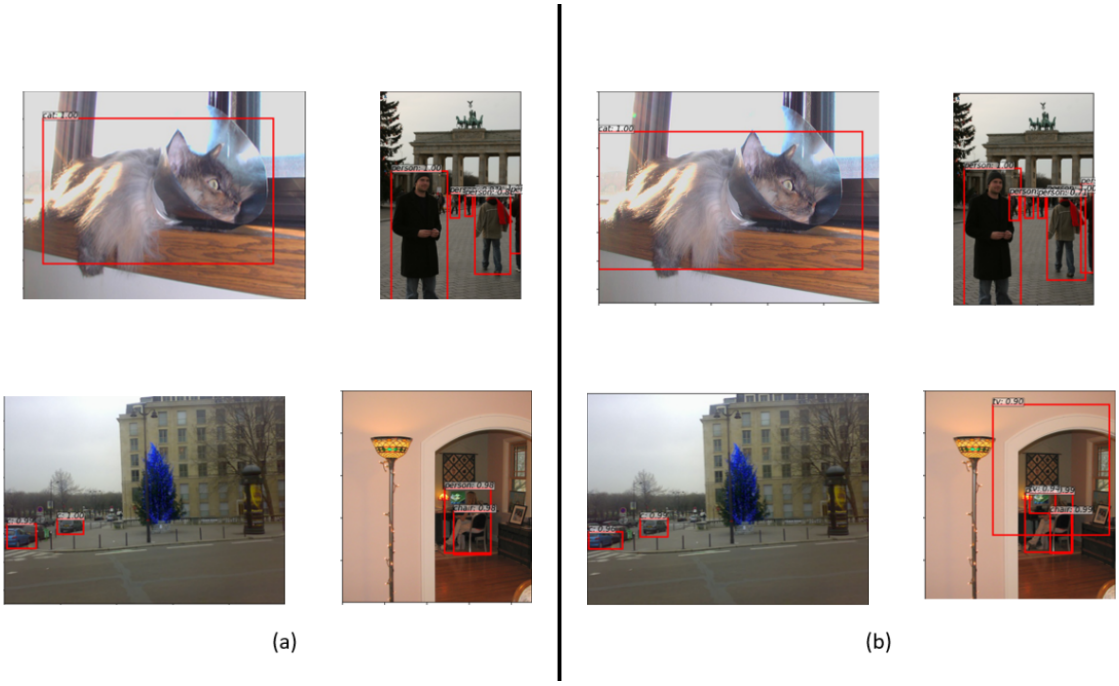
Backbone network	mAP (%)	Rate (testing mode)
VGG16	64.1	13 fps
ResNET101	61.5	13 fps
Original paper	65.8	17 fps

	fly	bike	bird	boat	pin	bus	car	cat	chair	cow	table	dog	horse	moto	person	plant	sheep	sofa	train	tv
VGG16	68.5	77.5	58.2	48.8	43.9	76.8	77.9	79.0	40.3	69.0	58.8	74.6	78.5	74.3	69.6	35.3	65.1	53.1	72.5	59.5
ResNET101	68.3	74.9	57.7	48.5	36.3	74.3	71.1	76.1	36.8	58.5	54.2	72.1	74.8	68.5	69.7	32.6	57.6	63.7	74.9	58.6

3.2.3 Sample Outcomes

As the last part of our results, we show some selected sample results for both VGG16 and ResNET101 with 3 anchor scales and ratios on the PASCAL VOC2007 dataset. From visual inspection, we see that in the upper right image, ResNET101 detects a TV falsely in the biggest box, but it also detects the computer screen as TV correctly. VGG16 seems to give better accuracy, overall. (a) samples are from VGG16 backbone model and (b) samples are from ResNET101 backbone in the following figures.

1. Anchor scales=[128^2, 256^2, 512^2], Anchor ratios=[2:1, 1:1, 1:2]

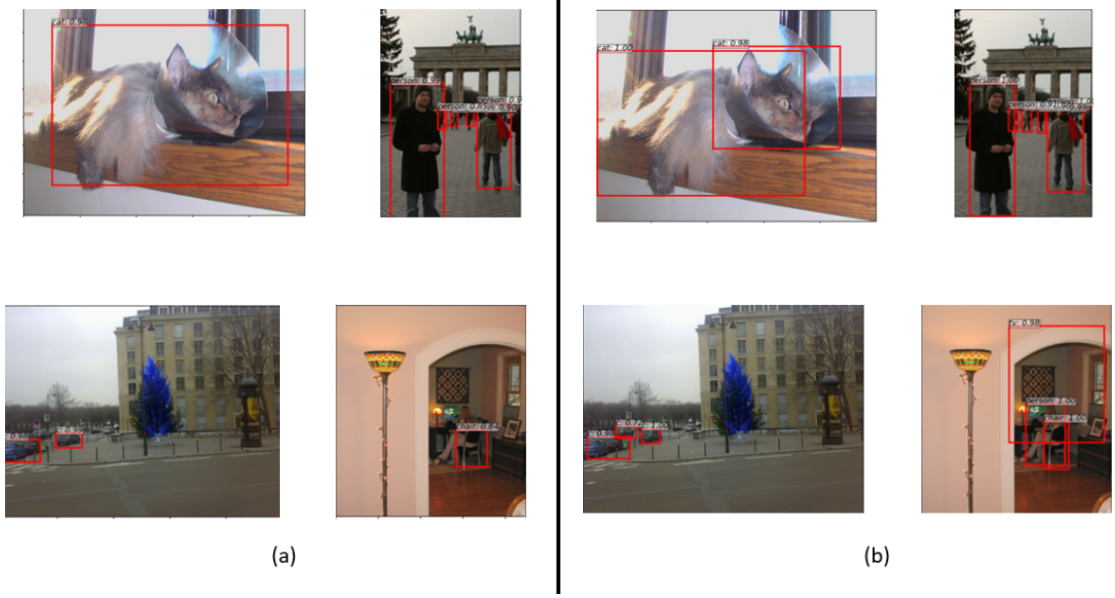


(a) (b)

2. Anchor scales=[128^2, 256^2, 512^2], Anchor ratios=[1:1]

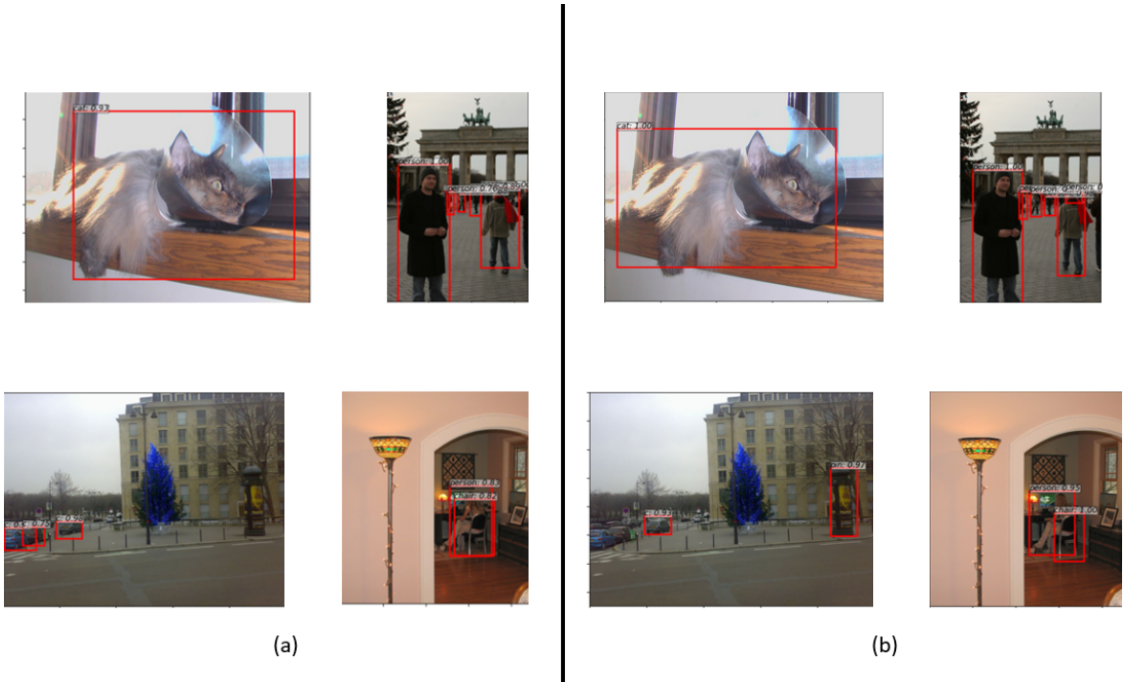
(a) (b)

3. Anchor scales=[128^2], Anchor ratios=[2:1, 1:1, 1:2]



(a) (b)

4. Anchor scales=[128^2], Anchor ratios=[1:1]



4 Conclusions

We began this project with understanding the paper [1] and its major differences with other R-CNN implementations. To summarize the work,

- We have successfully implemented the Faster R-CNN architecture by developing a code that has around 2000 lines, by following the guide in [4]. We cite our github repository in [5].
- We implemented the code for VGG16 and ResNET101 architecture and compared the results for various anchor attributes. We plotted loss curves, showed mean precision values and provided some sample outcomes. VGG16 with 9 anchors performed the best with a mAP value of 67.6%
- We used a total of 53.3 GPU hours, with training time of around 20 minutes per epoch. Our test time rate was 13 fps.
- Further improvement can be made by tuning the hyperparameters, trying 4-step alternate training approach and training for more epochs.

References

[1] Faster R-CNN Implement "Faster R-CNN: Towards real-time object detection with regional proposal networks", NIPS, 2015 by Ren et al.

[2] R. Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms," Medium, 09-Jul-2018. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Accessed: 20-Mar-2021].

[3] https://www.alegion.com/faster-r-cnn

[4] Machine-Vision Research Group, "Guide to build Faster R-CNN in PyTorch, " Available : https://medium.com/@fractaldle/guide-to-build-faster-R-CNN-in-pytorch-95b10c273439 [Accessed: 14-April-2021]

[5] https://github.com/garg-abhinav/Faster-R-CNN