Faster R-CNN: Towards real-time object detection with Region Proposal Networks

Authors: Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun

Project Members : Kavjit Durairaj, Nishant Velugula, Deborshi Goswami
Dhayabaran Ponsingh, Ved Prakash

December 12, 2018

Abstract

Object detection algorithms rely on various methods to generate region proposals from the image. Shaoqing Ren et al.[8] came up with the Faster RCNN algorithm to generate proposals by using a separate network called Region Proposal Network(RPN) which is fully convolutional. The image is fed through a head network to generate a feature map. The RPN is used to calculate object scores in a region as well as generate good bounding boxes for objects. The RPN and the detection network(Fast RCNN[5]) share features and therefore can be trained together. The predicted region proposals are reshaped using ROI pooling and then classification is done. Also, offsets for the bounding boxes are calculated here. The pre-trained VGG16 model is used in this project for classification. Pascal VOC 2012 dataset is used.

1 Overview of Faster R-CNN

In earlier approaches like RCNN and Fast RCNN, the algorithm deployed to generate region proposals was selective search. In selective search, basically we generate a set of 2000 regions. This is done by creating some initial proposals through sub-segmentation and then using a recursive algorithm to combine similar regions into larger ones. But, this proved to be time consuming and memory inefficient and this reduced the overall performance of the network.

In Faster RCNN, a separate network is used to predict these region proposals called Region Proposal Network(RPN). The input to the RPN is a convolutional feature map from the original image. The output is a set of region proposals and bounding boxes. The RPN is fully convolutional and this CNN is shared with the detection network. The CNN used in this project is VGGNEt. This makes Faster RCNN more faster and efficient.

2 Implementation of Faster RCNN

2.1 Image pre-processing

First, the image is normalized by a 3x1 mean vector and standard deviation vector. Each value corresponds to a color channel(RGB) and these values are common to all training and test images. The image is then re-scaled. The maximum dimension size is 1000 and the target size for the smaller dimension is 600.

2.2 Network architecture

The network is divided into three parts: head network, RPN and classification network. We use the VGG16 model pretrained on ImageNet. The VGG16 network has two parts - extractor and classifier. The extractor network is used in the head network which has 30 layers. The first 10 layers are frozen(no modifications). Training is done with the last 20 layers. This head network outputs convolutional feature maps.

These feature maps are then passed through the RPN which produces a set of promising Region of Interests(ROIs) that are most likely to contain a foreground object. The RPN has two output layers - one for class scores(9x2 array) and one for bounding box predictions(9x4 array). There are 9 anchors per pixel and we consider foreground and background for every anchor, hence it 9x2. In bounding box predictions, 9x4 represents 9 anchors and it's 4 corners' co-ordinates. Using the ROIs, we crop out the respective regions from the feature maps(from head). This is called as ROI pooling.

Now, these regions are passed through the classification network to classify the objects in the ROIs. This final network is the classification network in VGG16. In this project, we remove the 6th layer(final layer) and replace it with two parallel layers - one for classification(21 output classes) and one for regression for the bounding boxes(21x4 array). The RPN's final layer is also very similar to this. Also, we remove the dropout layers(2nd and 5th layers) in this classification network from the VGG16 model. The next step is to train this network. Our code follows the exact implementation described below.

2.3 Training: Implementation and RPN Hyperparameters

The main goal of training is to fine-tune the weights in the head VGG16 network and then tune the weights in the RPN and the classification network. To train the network, we need the ground truth boxes for objects. This is provided as 'annotations' in the PASCAL VOC dataset. The first step is to generate a set of anchors(bounding boxes) and this is same for all the images. We generate 9 anchors of different sizes for every pixel in the feature maps(output of head network). The anchor ratios are 0.5, 1 and 2 (where 1 represents a square). Now, we have to identify foreground and background anchors.

We pass the feature maps through RPN to get class probabilities(9x2) and bounding box regressors(9x4). We apply non-maximum suppression(NMS) based on the descending order of foreground scores from RPN and prune unwanted anchor box under consideration more than a certain thresholds. Then, we apply the regression coefficients to the remaining bounding boxes. The goal now is to train RPN to recognize foreground anchors and generate good regression coefficients. For that, we calculate the RPN loss.

$$RPN\ loss = Bounding\ box\ regression\ loss + Classification\ loss \tag{1}$$

The regression loss is the sum of losses for all the foreground anchors. The smooth L1 loss is used for this. For classification, we just use the cross entropy loss. We need the class labels(foreground or background) and scores for anchor boxes and also the target regression coefficients for the foreground anchor boxes to calculate the loss. Only those anchor boxes whose IoU with some ground truth box exceeds a set threshold(0.7 in our code) are considered foreground boxes.

$$Intersection\ over\ Union(IoU) = Area\ of\ overlap\ of\ boxes\ /\ Area\ of\ Union \tag{2}$$

If IoU less than a certain threshold value(0.3 in our code), they are background anchors. We also set two extra parameters - one for total number of boxes and one for the fraction of foreground boxes. This is set to 256 and 25% respectively in our code. Then, we calculate the regression coefficients between the foreground boxes and the corresponding ground truth box with maximum overlap. The regression co-efficients is calculated using the centre point coordinates of the bounding box, it's width and height.

Say b_x, b_y, b_h and b_w be the x coordinate, y coordinate, height and width of the bounding box. And, t_x, t_y, t_h and t_w are the corresponding target values. The regression coefficients are,

$$r_x = (t_x - b_x) \,/\, b_w \tag{3}$$

$$r_y = (t_y - b_y) \,/\, b_h \tag{4}$$

$$r_w = log(t_w/b_w) \tag{5}$$

$$r_h = log(t_h/b_h) \tag{6}$$

The loss for the classification layer is very similar to RPN loss. The only difference is that in RPN we considered only two classes - foreground and background. However, now we consider all the object classes(21 in PASCAL VOC 2012) in addition to background.

$$Classification\ network\ loss = Bounding\ box\ regression\ loss + Classification\ loss \tag{7}$$

In order to calculate this loss, we need class label predictions, regression coefficients which are the outputs from the classification layer and actual class labels and the target regression coefficients. Now, for selecting ROIS, we set the threshold to be 0.5 as opposed to 0.7 in RPN for foreground. For background, the threshold is less than 0.5. The target regression coefficients are calculated similar to the RPN. One important point to keep in mind is that for regression we consider only foreground ROIs, but for classification all the ROIs including the background are considered(since background is also a class).

Using these ROIs, we perform ROI pooling to extract the corresponding regions from the convolutional feature maps from the head network. This outputs square feature maps through affine transformations which is passed through the classifier of VGG16. It's then passed through two parallel fully-connected layers. This gives us class scores for each bounding box and also the corresponding regression coefficients. Using these regression coefficients we can get our final class-specific bounding boxes.

2.4 Computation Time

The team used over 300 hours of training over two Nvidia 1050 Ti, and cloud based 1080,1070,1060 GPUs. We were unable to use BlueWaters for training due to version control issues with the cupy library; cupy was extensively used in the ROI pooling layer. However an additional 30 hours were spent on BlueWaters training models without cupy; which was considerably slower and hence abandoned.

2.5 Dataset Description

The PASCAL VOC 2012 dataset is an object recognition and image segmentation dataset containing objects belonging to 20 different classes. The training and validation sets combined contain 11540 different images with a total of 27450 objects. In our implementation we use all of the training and validation images for training and the test set for evaluating test performance. The VOC 2007 test set was used for model testing.

2.6 Training Methods

2.6.1 Joint Approximate Training with shared Convolutions

The easiest way to implement Faster-RCNN training is using joint RPN and Detector networks with shared convolutional layers. However, since the proposal box coordinates are a response of the network,

gradient during backpropagation must also be computed with respect to proposal box coordinates. This is a non-trivial problem and hence the authors recommend an approximate training approach where the gradients are accumulated backwards while ignoring gradients w.r.t proposal layer. The head network generates feature maps for both the RPN as well as the classification network. In this method, the total loss (sum of losses over Classification and RPN network) is used to train the network. In the case of unshared convolutions, a different VGG16 head network generates features for the classification network. The head network for RPN and classification networks are therefore trained separately.

2.6.2 4 Step Training with shared Convolutions

In order to train Faster-RCNN in a non-approximate manner, the authors suggest a heuristic 4 step training method where the RPN and Detection network are alternately trained. In Step 1,

1. A VGG16 pretrained on the imageNet extracts features through its convolutional layers. The Features are used by the RPN network. The sum of RPN losses are considered to train the RPN network. In this case, we freeze none of the layers of the VGG16 extractor in order to generate accurate feature maps for RPN object detection.

2. The ROIs proposed by the trained RPN are scaled to the size of input image, and thus ROI's are extracted directly from the image itself. The ROI are passed through a newly initialized pretrained VGG16 which generate feature maps for the classification network. This time around, the total classification loss is used to train the network. The VGG16 convolution layers are therefore tuned to extract features specific to the 20 classes of PASCAL VOC dataset.

3. The VGG16 extractor trained in the previous iteration is used to initilaize the RPN network again. During training, the extractor is frozen and only the layers unique to RPN are fine-tuned. At this point the RPN and Classication networks share their convolutions.

4. In the final iteration the ROI's generated by the RPN are again used to fine tune the Classification layer.

2.7 Experiments

2.7.1 Training Methods Experiment

A number of training methods were implemented to better understand the model characteristics of Faster R-CNN. They are listed below:

1. Joint Approximate training with Shared Convolutions - Original anchor box sizes and scales
2. Joint Approximate training with un-shared Convolutions
3. 4 Step Non-Approximate training with shared convolutions - Failure
4. 4 Step Approximate training with shared convolutions

2.7.2 Anchor Box Experiment

The model used was Joint approximate training with shared convolutions. The following were the experiments performed:

1. 3 scales - 8, 16, 32 and 2 aspect-ratios - 0.5, 2 : This denotes only **rectangular anchor boxes**
2. 3 scales - 8, 16,32 and 1 aspect ratio - 1 : Only **square anchor boxes**
3. 1 Scale - 32, 3 aspect Ratios - 0.5, 1, 2 : Bounding boxes are **too large**

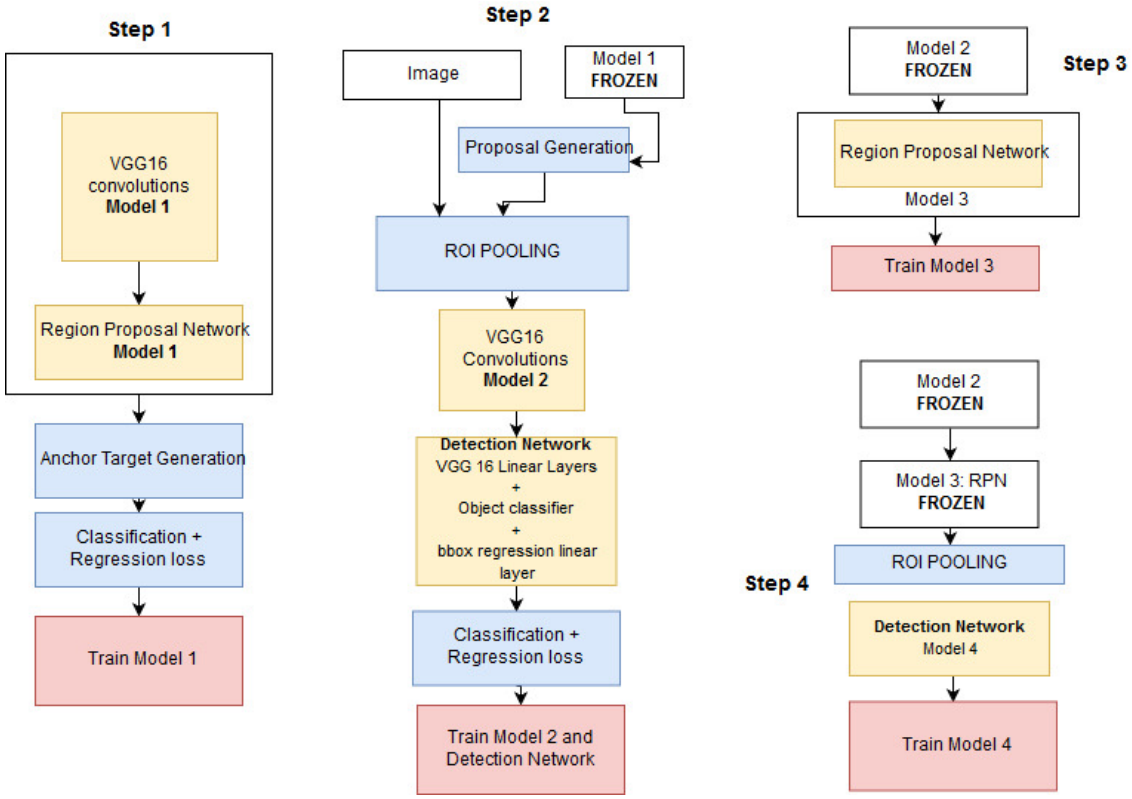


Figure 1: 4 Step Training Method

2.7.3 Hyperparameter Details

Epochs: All joint models were trained over 10 epochs, all 4 step models were trained over 11 epochs (Step1 : 3 epochs, Step 2: 4 epochs, Step: 2 epochs, Step 4: 2 epochs
Optimizer: We used the SGD optimizer all the way through as the choice of optimizer had no impact on the experiments.
Learning Rate: The base learning rate was 0.001 with a decay of 0.1 every 3 epochs.
Weight Decay:0.0005
Anchors: Base aspect ratio : 0.5,1,2, Base scale: 8,16,32

3 Results

3.1 Training losses

The training methods experiment was successful to varying extents. We encountered a number of problems with the 4 step training protocol due to its complexity. In the end we implemented a 4 step approximate solution where features for the classification network in the second step is generated by the RPN's VGG16 head network. The following are the total-loss plots for our successful implementations. Note that the plot includes the anchor box ablation experiments. The stepwise look of the losses is an indication of model overfitting due to lesser number of training images.

For the training methods experiment, we can see that the 4 step method and joint approximate method have similar losses at the end of training. The 4 step loss has a layered shape because at each step, either RPN loss or Classifier loss is computed and optimized, therefore contribution of the frozen loss is included in the total loss at each step. The joint training with unshared convolutions method(red) takes a longer time to reach similar performances as the other models. This concurs with our intuition

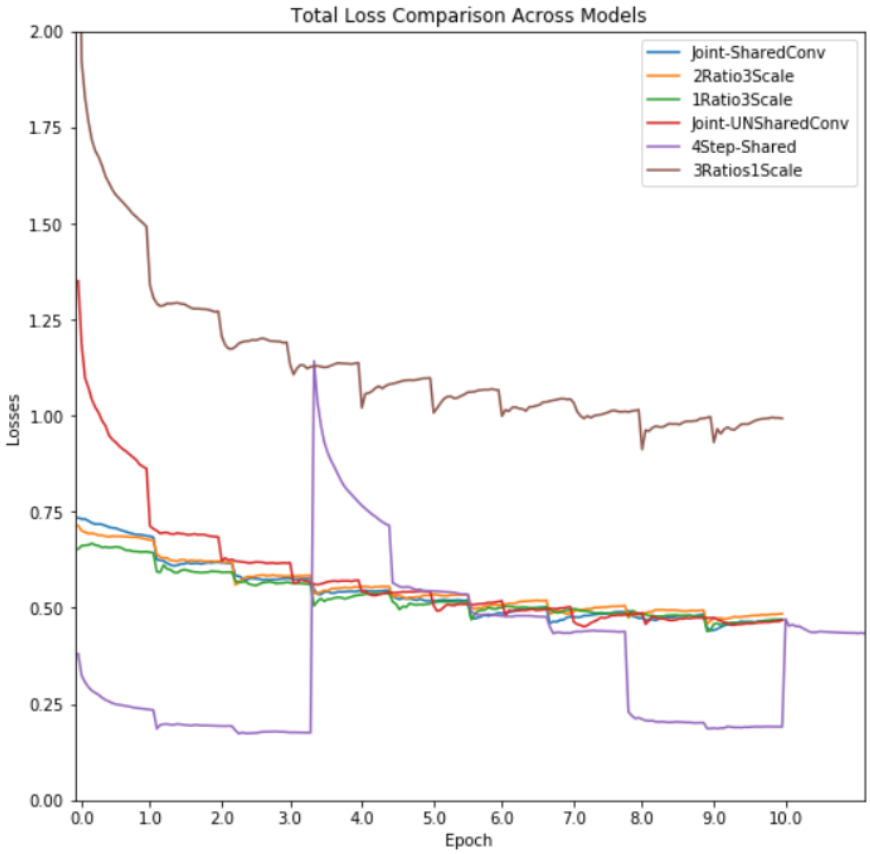


Figure 2: Total losses across successfully trained models

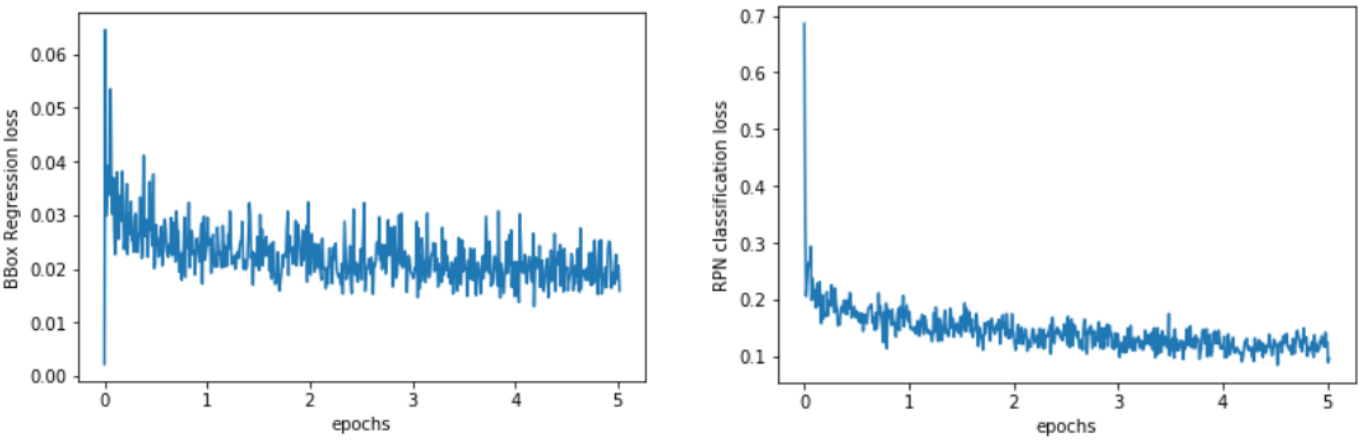
that the network should learn features quicker by sharing their convolutions with region proposal and object classification.

In the anchor box experiment, rectangular and square anchor boxes of various scales have similar looking loss plots. However, when the anchor box sizes are made too large the loss is by far the worst among all models. We explore this further in our MAP results.

In Figure 2 we have shown the 4 step approximate and not the 4 Step non-approximate training experiment which was a failure. However, we did train the RPN for non-approx. method, i.e. first step. Figures 3 and 4 show the comparison between the best training losses of the joint approximate model versus the 4 step non-approx. model in the 1st step. We can see that regression loss optimizes much quicker in the 4 step non-approx training over half the number of epochs. Even classification loss shows comparable performance in half the epochs of the joint training case. This underscores the potential of 4 step training to achieve better results down the pipeline.

3.2 Mean Average Precision on Test Set

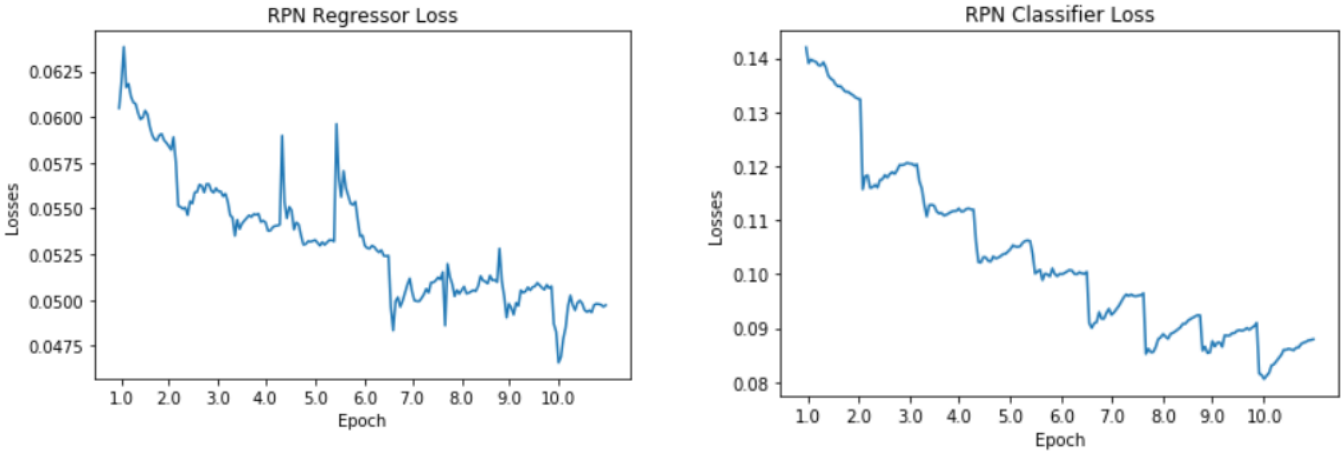
Figure 6 in the Appendix, shows detailed AP and Recall values over the 6 models experimented on. The classes that consistently perform badly in Average Precision are 'bottle', 'plant', 'chair', 'boat'. This is possibly because the bounding boxes are too small for the respective classes for the model to consistently train and detect the objects. In Figure 5(a), We can see a comparison of Mean Average Precision across the models. The best MAP is given by Join Approx. training which has the most amount of fine tuning time. It beats the MAP for Joint Approx. with unshared convolutions, which is intuitive as neither convolution sets have any knowledge about the specialized feature extraction of the other model. The lowest MAP is given when anchor box sizes are too large and medium or smaller



(a) Bounding box regression loss

(b) Foreground classification loss

Figure 3: Step 1 RPN losses of 4 Step Non-Approximate Model over 5 epochs



(a) Bounding box regression loss

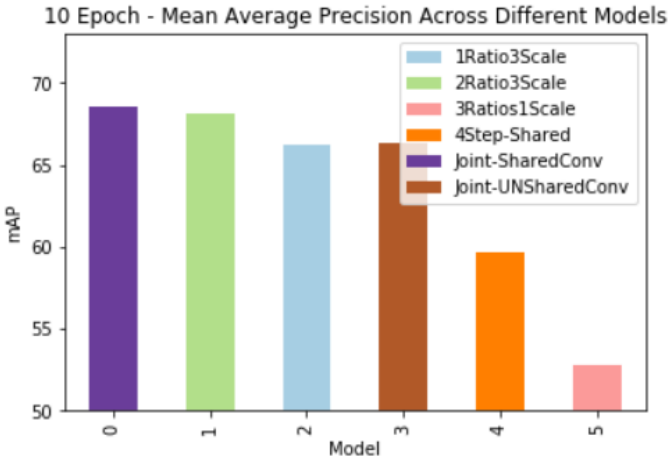
(b) Foreground classification loss

Figure 4: RPN losses of best Joint approximate model

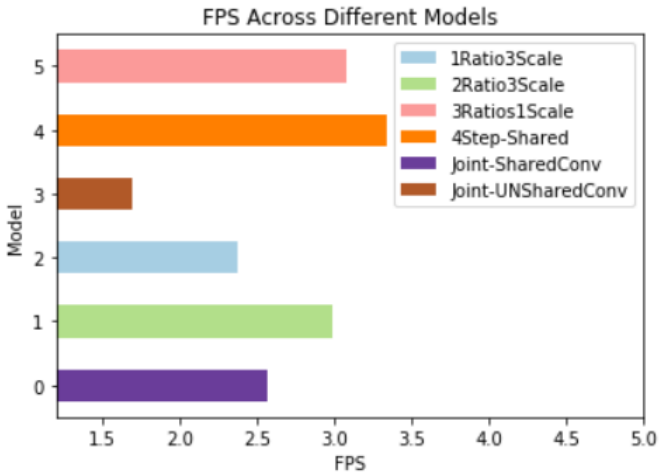
objects rarely get detected. For this experiment we included exceptions for a number of images which were panoramic(see appendix), and hence the anchor sizes exceeded the image dimensions resulting in errors. Most interestingly though, rectangular shaped anchor boxes perform better than square shaped anchor boxes. There can be three possible reasons for this; density of detections increase as there are twice the number of rectangular boxes, rectangles generalize better than squares(see appendix), and most object's ground truth bounding boxes are rectangular making them suitable to be trained from rectangular anchor boxes. The best results that we obtained is almost comparable with the results from the paper. The mAP as reported in the paper was 69.9 and ours was 68.5.

3.3 Object detection speed in Frames per Second

In Figure 5(b) we plot the model speed on webcam test video. The 4 step approximate method has the highest test speed possibly due to the lightweight nature of the final test model. The most computationally expensive model is the one with unshared convolutions, which is intuitive. Among the different anchor box sizes, we can see that 1 square anchor box leads to slower detections than 2 rectangular anchor boxes which is an interesting result. When the anchor box sizes are made too large the model



(a) Mean average Precision across models



(b) Object detection speed on test images in FPS

Figure 5: Object detection speed on test images in FPS

speed actually improves. This is probably because there are lesser object detections and therefore lesser anchor boxes to evaluate.

4 Conclusion

Our implementation of Faster-RCNN focussed on understanding the model characteristics through rigourous testing. The best performance in terms of Mean Average Precision is achieved by Joint Training, while the four step training method achieve fastest test performance due to its lightweight final model. We carried out a number of anchor box experiments which shed light on how different anchor boxes affect the behaviour of Faster RCNN. The appendix details test images when passed through Faster-RCNN

References

[1] Long Chen. https://github.com/longcw/faster$_r$cnn$_p$ytorch, Aug2018.

[2] Yun Chen. https://github.com/chenyuntc/simple-faster-rcnn-pytorch, Sep 2018.

[3] Hao Gao. Faster r-cnn explained, https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8, 2017.

[4] Ross Girshick. https://github.com/rbgirshick/py-faster-rcnn, Jan 2018.

[5] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[6] Potter Hsu. https://github.com/potterhsu/easy-faster-rcnn.pytorch, Dec 2018.

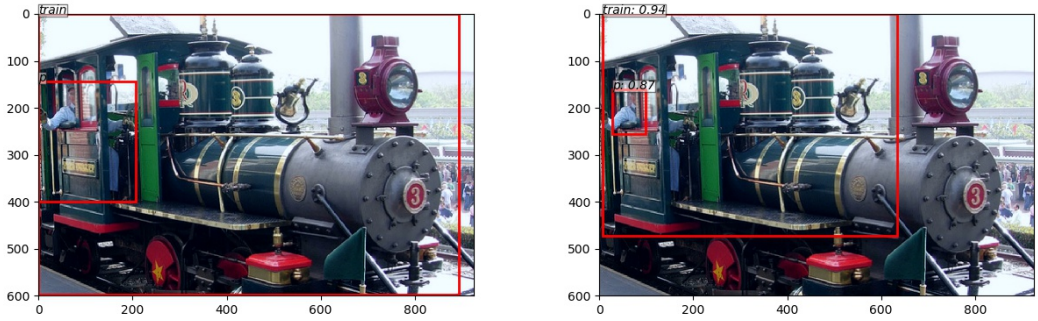
[7] Ankur Mohan. Object detection and classification using r-cnns, http://www.telesens.co/2018/03/11/object-detection-and-classification-using-r-cnns/, n.d.

[8] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

Appendices

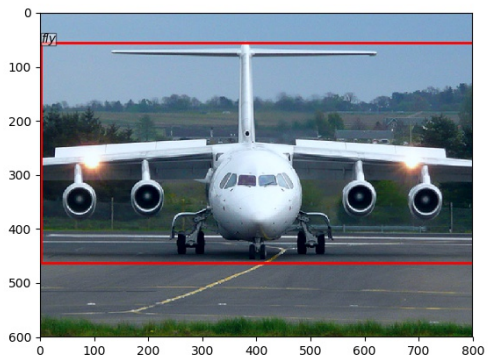
	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6		Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
			Average Precision							Recall			
Aeroplane	83.32	68.67	77.14	76.79	82.85	69.04		88.41	48.9	80.43	79.84	75.53	65.65
Bike	68.23	56.2	70.06	71.04	66	48.38		75.94	46.53	73.39	73.59	68.46	42.25
Bird	69.14	57.56	66.46	64.28	67.09	43.23		70.01	50.23	62.24	69.28	61.3	44.32
Boat	51.84	46.95	51.57	52.82	52.26	44.8		72.48	46.68	63.16	74.56	63.81	53.89
Bottle/Pin	55.46	35.52	50.81	53.35	52.73	14.85		65.52	35.29	54.57	63.14	50.93	15.68
Bus	75.27	70.49	81.34	79.75	74.51	68.89		80.6	62.94	84.9	85.64	81.8	64.72
Car	81.62	75.28	80.37	83.75	77.05	60.46		84.08	60.91	80.39	80.29	70.8	50.84
Cat	83.02	76.39	79.79	81.56	81.11	67.83		82.18	68.49	78.06	84.04	76.51	59.16
Chair	48.37	39.71	47.94	49.23	43.22	30.05		64.82	49.09	71.19	57.37	61.86	47.68
Cow	70.87	68.02	55.28	75.99	66.51	56.97		77.29	75.23	41.15	77.23	69.46	57.22
Table	65.26	63.24	62.52	63.68	62.92	61.71		75.04	70.54	81.09	73.57	75.02	79.04
Dog	63.51	74.28	67.02	71.61	70.91	61.18		66.11	56.69	67.68	71.39	68.34	62.48
Horse	78.67	74.07	77.16	81.06	73.03	76.06		78.3	60.06	76.19	76.16	76.5	66.86
Motorbike	77.86	66.04	76.27	73.61	72.89	64.27		83.14	52.68	79.69	73.38	75.42	66.73
Person	76.07	67.67	74.07	77.01	74.43	47.49		66.46	47.33	63.47	68.66	64.61	34.58
Plant	40.38	23.87	38.65	35.36	36.17	17.64		56.05	29.02	52.51	57.22	43.65	19.6
Sheep	72.37	60.02	62.87	68.34	67.28	49.14		66.6	57.02	62.56	66.27	70.34	44.43
Sofa	63.87	55.41	64.83	62.26	62.48	57.69		79.79	63.72	77.42	67.53	69.76	66.7
Train	72.35	63.16	68.42	68.81	66	59.86		81.66	57.72	80.34	80.08	76.07	63.41
TV	73.06	51.94	72.29	71.56	77.13	55.41		87.73	56.7	81.87	77.14	79.17	59.15

Figure 6: Precision and recall values | The 6 models are : **Model 1** - Joint approximate with shared convolutions; **Model 2** - 4 step approximate with shared convolutions; **Model 3** - 3 scales 1 ratio; **Model 4** - 3 scales 2 ratios; **Model 5** - Joint approximate without sharing convolutions; **Model 6** - 1 scale 3 ratios

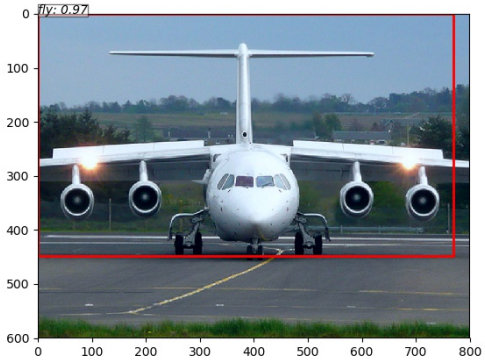


(a) actual (b) predicted

Figure 7: When only squares are used at anchor boxes, model fails to generalize to rectangles



(a) actual



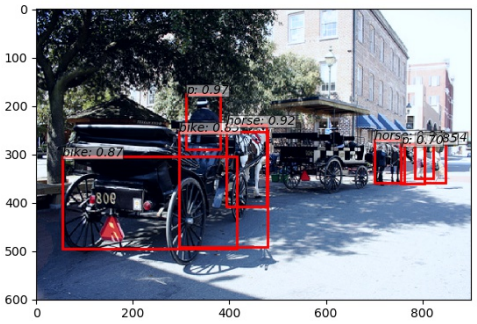
(b) predicted

(c) actual

(d) predicted

Figure 8: When anchor boxes are too large the model is capable of detecting only large objects, but also loses generalization capacity as seen in (c) and (d)

(a) actual



(b) predicted

Figure 9: When only rectangular anchors are used, bounding boxes are more generalizable but the model also generates more false predictions.

Figure 10: Some images were panoramic which was incompatible with anchor box sizes that were too large (See anchor box experiment 3)

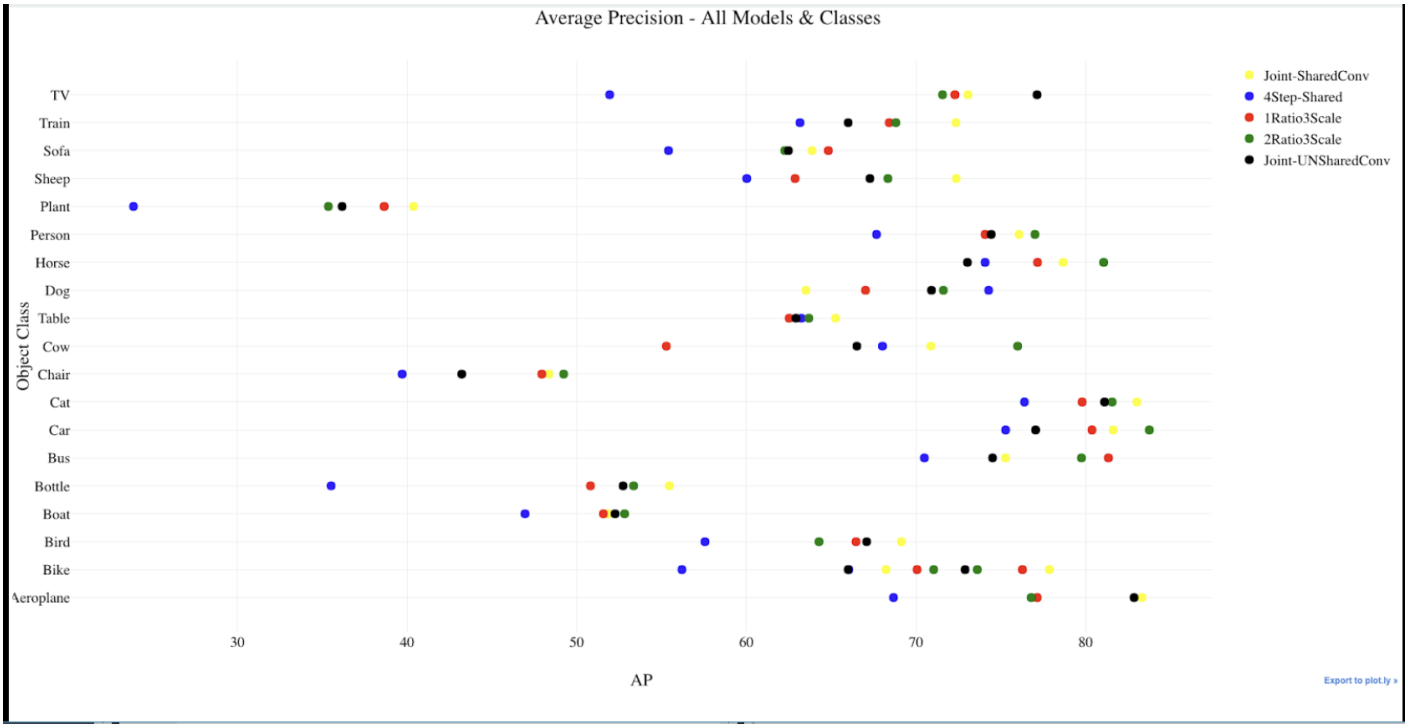


Figure 11: Detailed Average Precision plots for every model for every class