

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/313434032>

# An Implementation of Faster RCNN with Study for Region Sampling

Article · February 2017

CITATIONS

165

READS

1,415

2 authors:



Xinlei Chen

Carnegie Mellon University

24 PUBLICATIONS 5,939 CITATIONS

SEE PROFILE



Abhinav Gupta

BMS Institute of Technology

149 PUBLICATIONS 34,380 CITATIONS

SEE PROFILE

# An Implementation of Faster RCNN with Study for Region Sampling

Xinlei Chen  
Carnegie Mellon University  
xinleic@cs.cmu.edu

Abhinav Gupta  
Carnegie Mellon University  
abhinavg@cs.cmu.edu

## Abstract

We adapted the join-training scheme of Faster RCNN framework from Caffe to TensorFlow as a baseline implementation for object detection. Our code is made publicly available. This report documents the simplifications made to the original pipeline, with justifications from ablation analysis on both PASCAL VOC 2007 and COCO 2014. We further investigated the role of non-maximal suppression (NMS) in selecting regions-of-interest (RoIs) for region classification, and found that a biased sampling toward small regions helps performance and can achieve on-par mAP to NMS-based sampling when converged sufficiently.

## 1. Baseline Faster RCNN with Simplification

We adapted the join-training scheme of Faster RCNN detection framework<sup>1</sup> [6] from Caffe<sup>2</sup> to TensorFlow<sup>3</sup> as a baseline implementation. Our code is made publicly available<sup>4</sup>. During the implementation process, several simplifications are made to the original pipeline, with observations from ablation analysis that they are either not affecting or even potentially improving the performance. The ablation analysis has the following default setup:

**Base network.** Pre-trained VGG16 [8]. The feature map from conv5\_3 are used for region proposals and fed into region-of-interest (RoI) pooling.

**Datasets.** Both PASCAL VOC 2007 [2] and COCO 2014 [5]. For VOC we use the trainval split for training, and test for evaluation. For COCO we use train+valminusminival and minival, same as the published model.

**Training/Testing.** The default end-to-end, single-scale training/testing scheme is copied from the original implementation. Learning rate starts with .001 and is

reduced after 50k/350k iterations. Training finishes at 70k/490k iterations. Following COCO challenge requirements, for each testing image, the detection pipeline provides at most 100 detection results.

**Evaluation.** We use evaluation toolkits provided by the respective dataset. The metrics are based on detection average precision/recall.

The first notable change follows Huang *et al.* [4]. Instead of using the RoI pooling layer, we use the `crop_and_resize` operator, which crops and resizes feature maps to  $14 \times 14$ , and then max-pool them to  $7 \times 7$  to match the input size of `fc6`.

Second, we do not aggregate gradients from  $N = 2$  images and  $R = 128$  regions [3], instead we simply sample  $R = 256$  regions from  $N = 1$  images during a single forward-backward pass. Gradient accumulation across multiple batches is slow, and requires extra operators in TensorFlow. Note that  $R$  is the number of regions sampled for training the region classifier, for training region proposal network (RPN) we still use the default 256 regions.

Third, the original Faster RCNN removes small proposals (less than 16 pixels in height or width in the original scale). We find this step redundant, hurting the performance especially for small objects.

Other minor changes that does not seem to affect the performance include: 1) double the learning rate for bias; 2) stop weight decay on bias; 3) remove aspect-ratio grouping (introduced to save memory); 4) exclude ground-truth bounding boxes in the RoIs during training, since they are not accessible during testing and can bias the input distribution for region classification.

For ablation analysis results on VOC 2007, please check at Table 1. Performance-wise, our implementation is in general on par with the original Caffe implementation. The `crop_and_resize` pooling appears to have a slight advantage over RoI pooling.

We further test the pipeline on COCO, see Table 2. We fix  $N = 1$  and only use `crop_and_resize` pooling, which in general gives better average recall than RoI pooling. Keeping the small region proposals also gives consis-

<sup>1</sup><https://github.com/rbgirshick/py-faster-rcnn>

<sup>2</sup><https://github.com/BVLC/caffe>

<sup>3</sup><https://github.com/tensorflow>

<sup>4</sup><https://github.com/endernewton/tf-faster-rcnn>

Table 1: **VOC 2007 test** object detection average precision (%). All use Faster RCNN on VGG16. Legend: **C**: Caffe implementation, results are generated with models trained with open-sourced code; **T**: our TensorFlow implementation; **P**: use RoI pooling; **A**: use `crop_and_resize`; **N**: images per batch; **R**: total RoIs for training the region classifier; **S**: keep small RoIs.

N R S				mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	
C	P	2	128	X	70.0	68.7	79.2	67.6	54.1	52.3	75.8	79.8	84.3	50.1	78.3	65.1	82.2	84.8	72.9	76.0	44.9	70.9	63.3	76.1	72.6
C	P	1	128	X	70.1	69.4	78.3	67.4	55.9	53.9	81.5	84.6	85.2	49.3	74.3	62.4	80.6	83.2	74.6	76.9	43.0	72.1	62.1	75.6	71.3
T	P	1	128	X	69.8	71.9	76.4	67.8	54.8	51.9	80.1	85.4	83.8	46.9	73.6	65.7	79.8	80.2	75.4	76.2	41.4	70.5	65.0	75.7	73.5
T	A	1	128	X	69.9	69.2	77.5	67.1	56.9	52.9	75.8	85.4	82.5	47.4	77.8	61.9	81.2	83.5	76.0	76.9	43.2	69.9	65.9	74.4	72.1
T	A	1	256	X	70.7	69.4	78.9	66.8	57.4	55.7	75.2	85.1	83.0	50.6	80.0	67.6	81.4	80.4	74.5	77.1	42.8	71.1	66.3	77.1	73.5
T	A	1	256	✓	70.9	67.5	78.4	67.0	53.4	58.9	78.2	85.1	84.4	49.2	82.1	66.7	77.3	84.3	75.4	77.3	46.2	71.0	66.6	75.2	73.5

Table 2: **COCO 2014 minival** object detection average precision and recall (%) with provided evaluation tool. For Caffe we use the released model. Legend same as in Table 1.

N R S				AP	AP-.5	AP-.75	AP-S	AP-M	AP-L	AR-1	AR-10	AR-100	AR-S	AR-M	AR-L	
C	P	2	128	X	24.2	45.1	23.4	7.4	27.5	38.2	23.6	33.7	34.3	11.7	39.5	54.1
T	A	1	128	X	25.2	44.7	25.6	9.4	29.4	37.5	24.3	35.0	35.7	14.2	41.4	53.4
T	A	1	256	X	26.0	45.8	27.1	10.5	30.4	38.1	24.8	35.7	36.3	15.0	42.5	53.0
T	A	1	512	X	25.7	45.4	26.3	10.0	29.9	37.3	24.4	35.1	35.8	14.1	41.7	52.2
T	A	1	128	✓	25.4	45.6	25.4	11.0	29.2	37.0	24.6	35.8	36.5	16.8	41.5	52.8
T	A	1	256	✓	26.5	46.7	27.2	11.8	30.4	37.5	24.9	36.3	37.1	17.3	42.1	52.4
T	A	1	512	✓	26.1	46.4	26.3	11.6	29.9	36.6	24.7	35.8	36.5	16.4	41.8	51.9

tent boost on small objects. Overall our baseline implementation gives better AP (+4%) and AR (+5%) for small objects. As we vary  $R$ , we find 256 gives a good trade-off with the default training scheme, as further increasing  $R$  causes potential over-fitting.

### 1.1. Training/Testing Speed

Ideally, our training procedure can almost cut the total time in half since gradient is only accumulated over  $N = 1$  image. However, the increased batch size  $R = 256$  and the use of `crop_and_resize` pooling slow each iteration a bit. Adding the underlying TensorFlow overhead, the average speed for a COCO net on a Titan X (non Pascal) GPU for training is 400ms per iteration, whereas for testing it is 160ms per image in our experimental environment.

## 2. A Study of Region Sampling

We also investigated how the distribution of the region proposals fed into region classification can influence the training/testing process. In the original Faster RCNN, several steps are taken to select a set of regions:

- First, take the top  $K$  regions according to RPN score.
- Then, non-maximal suppression (NMS) with overlapping ratio of 0.7 is applied to perform de-duplication.
- Third, top  $k$  regions are selected as RoIs.

For training,  $K = 12000$  and  $k = 2000$  are used, and later  $R$  regions are sampled for training the region classifier with pre-defined positive/negative ratio (0.25/0.75); for testing

$K = 6000$  and  $k = 300$  are used. We refer to this default setting as **NMS**.

In Ren *et al.* [6], a comparable mean average precision (mAP) can be achieved when the top-ranked  $K = 6000$  proposals are directly selected without NMS during *testing*. This suggests that NMS can be removed at the cost of evaluating more RoIs. However, it is less clear whether NMS de-duplication is necessary during *training*. On a related note, NMS is believed to be crucial for selecting hard examples for Fast RCNN [7]. Therefore, we want to check if it is also true for Faster RCNN in the joint-training setting.

Our first alternative (**ALL**) works by simply feeding all top  $K$  regions for positive/negative sampling without NMS. While this alternative appears to optimize the same objective function as the one with NMS, there is a subtle difference: NMS implicitly biases the sampling procedure toward smaller regions. Intuitively, it is more likely for large regions to overlap than small regions, so large regions have a higher chance to be suppressed. A proper bias in sampling is known to help at least converge networks more quickly [1] and is actually also used in Faster RCNN: a fixed positive/negative ratio to avoid always learning on negative patches. To this end, we add two more alternatives for comparison. The first one (**PRE**) computes the final ratio of a pre-trained Faster RCNN model that uses NMS, and samples regions based on this final ratio. The second one (**POW**) simply fits the sampling ratio to the power law:  $r(s) = s^{-\gamma}$  where  $r$  is ratio,  $s$  is scale, and  $\gamma$  is a constant factor (set as 1.). While PRE still depends on a trained model with NMS, POW does not require NMS at all. To fit the target distribution, we keep all regions of the scale with

Table 3: **VOC 2007 test** object detection average precision (%). Analysis of different region sampling schemes for train/test combinations. Baseline (first row) uses NMS for both training and testing. Please refer to Section 2 for the detailed meaning of ALL, PRE, POW and TOP, none of which is based on NMS.

Train	Test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv
NMS	NMS	70.9	67.5	78.4	67.0	53.4	<b>58.9</b>	78.2	85.1	84.4	49.2	<b>82.1</b>	66.7	77.3	84.3	75.4	77.3	<b>46.2</b>	<b>71.0</b>	66.6	75.2	<b>73.5</b>
ALL	TOP	70.4	<b>73.9</b>	77.7	67.0	56.6	47.7	80.3	83.8	83.8	48.0	77.9	<b>68.6</b>	<b>80.8</b>	84.0	<b>76.5</b>	75.7	41.6	69.2	66.6	77.6	70.3
PRE	TOP	71.1	72.7	<b>79.0</b>	67.3	<b>58.8</b>	53.3	<b>80.9</b>	85.2	<b>84.8</b>	<b>50.6</b>	80.3	66.4	80.1	83.5	74.2	<b>77.6</b>	44.3	69.7	65.7	76.9	70.9
POW	TOP	71.0	<b>73.9</b>	78.5	67.1	57.7	53.1	80.1	<b>85.8</b>	83.6	50.0	80.0	65.6	80.6	80.5	75.4	76.8	44.4	70.6	66.0	<b>78.3</b>	72.6
NMS	TOP	<b>71.2</b>	67.6	78.9	<b>67.6</b>	55.2	56.9	78.8	85.2	83.9	49.8	81.9	65.5	80.1	<b>84.4</b>	75.7	<b>77.6</b>	45.3	70.8	<b>66.9</b>	78.2	72.9

Table 4: **COCO 2014 minival** object detection average precision and recall (%) with provided evaluation tool. Baseline (first row) uses NMS for both training and testing. Please refer to Section 2 for the detailed meaning of ALL, PRE, POW and TOP, none of which is based on NMS. stepsize is the number of train iterations before the learning rate is reduced; and itersize is the total number of iterations.

Train	Test	stepsize	itersize	AP	AP-.5	AP-.75	AP-S	AP-M	AP-L	AR-1	AR-10	AR-100	AR-S	AR-M	AR-L
NMS	NMS	350k	490k	26.5	46.7	27.2	11.8	30.4	37.5	24.9	36.3	37.1	17.3	42.1	52.4
ALL	TOP	350k	490k	23.2	41.2	23.7	7.1	24.1	36.9	23.0	32.9	33.5	12.1	36.5	52.8
PRE	TOP	350k	490k	25.1	44.1	25.7	9.0	27.4	38.8	24.4	35.1	35.7	14.1	39.5	55.0
POW	TOP	350k	490k	25.2	44.6	25.6	9.6	28.3	37.6	24.4	35.5	36.4	14.9	40.5	55.5
NMS	TOP	350k	490k	26.9	47.0	27.7	<b>12.0</b>	31.0	38.9	25.3	37.2	38.1	17.6	43.1	54.0
ALL	TOP	600k	790k	25.0	43.5	25.4	7.8	26.1	39.5	24.2	34.4	35.1	13.1	38.2	55.1
PRE	TOP	600k	790k	26.6	45.7	27.7	9.8	29.3	41.5	25.3	36.5	37.3	15.3	41.9	56.0
POW	TOP	600k	790k	26.9	46.4	28.2	10.8	29.8	40.8	25.4	36.8	37.6	16.2	42.1	56.6
NMS	NMS	600k	790k	27.9	48.2	29.0	11.8	31.8	40.3	26.0	37.5	38.3	17.6	43.4	55.4
NMS	TOP	600k	790k	<b>28.3</b>	<b>48.7</b>	<b>29.5</b>	11.8	<b>32.5</b>	<b>41.9</b>	<b>26.2</b>	<b>38.3</b>	<b>39.2</b>	<b>18.0</b>	<b>44.3</b>	<b>56.7</b>

the highest ratio in the distribution, and randomly select regions of other scales according to the relative ratio. *e.g.*, if the distribution is (0.4, 0.2, 0.2) for scales (8, 16, 32), then all the scale-8 regions are kept, and 50% of the other two scales are later sampled. Note that for both of them we set  $k = 6000$  ( $k$  is functioning as  $K$ ) during training, since roughly half the regions are already thrown away.

Following Ren *et al.* [6], we simply select top  $K$  proposals for evaluation directly. With little or no harm on precision but direct benefit on recall, mAP generally increases as  $K$  gets larger. We set  $K = 5000$  trading off speed and performance. This testing scheme is referred as **TOP**.

We begin by showing results on VOC 2007 in Table 3. As can be seen, apart from ALL, other schemes with biased sampling all achieve the same level of mAP (around 71%). We also include results (last row) that uses NMS during training but switches to TOP for testing. Somewhat to our surprise, it achieves better performance. In fact, we find this advantage of TOP over NMS consistently exists when  $K$  is sufficiently large.

A more thorough set of experiments were conducted on COCO, which are summarized in Table 4. Similar to VOC, we find biased sampling (NMS, PRE and POW) in general gives better results than uniform sampling (ALL). In particular, with 490k iterations of training, NMS is able to offer a performance similar to PRE/POW after 790k iterations. Out of curiosity, we also checked the model trained with 790k NMS iterations, which is able to converge to a better AP (28.3 on minival) with the TOP testing scheme. We

did notice that with more iterations, the gap between NMS and POW narrows down from 1.7 (490k) to 1.4 (790k), indicating the latter ones may catch up eventually. The difference to VOC suggests that 490k iterations are not sufficient to fully converge on COCO. Extra experiments with longer training iterations are needed for a more conclusive note.

## References

- [1] A. Bansal, X. Chen, B. Russell, A. Gupta, and D. Ramanan. Pixelnet: Towards a general pixel-level architecture. *arXiv preprint arXiv:1609.06694*, 2016.
- [2] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, 2010.
- [3] R. Girshick. Fast r-cnn. In *ICCV*, 2015.
- [4] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016.
- [5] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.
- [6] S. Ren, K. H. R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.
- [7] A. Shrivastava, A. Gupta, and R. Girshick. Training Region-based Object Detectors with Online Hard Example Mining. In *CVPR*, 2016.
- [8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.