Experiment: VGG_CNN_M_1024

FULL IMAGE

Classification Results of VGG_CNN_M_1024
Top 1 = 0.5962
Top 5 = 0.8482

GT REGIONS

Results of VGG_CNN_M_1024 with RoI pooling:

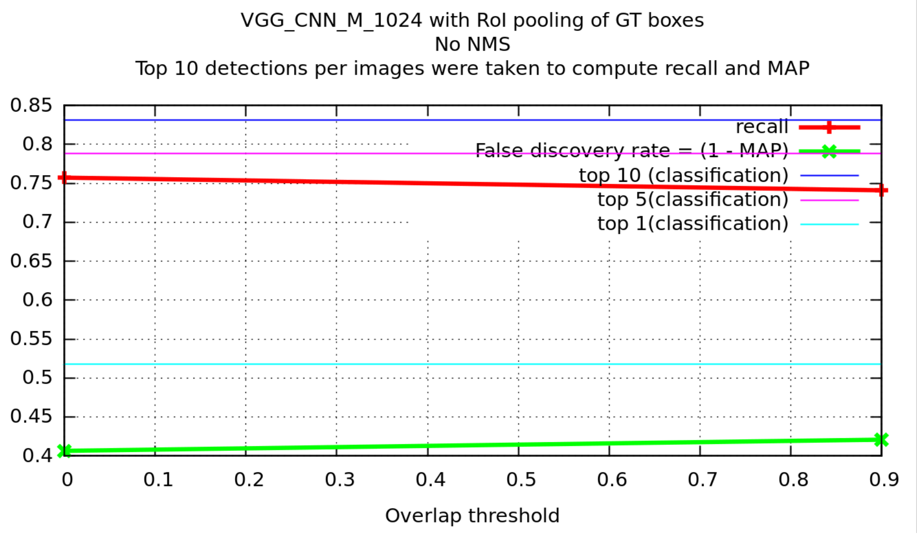
Training: RoI pooled only with GT bounding boxes. Fast-Rcnn network was used and the proposals from selective search algorithm were removed
Testing: GT bboxes of test images were used as object proposals. Proposals with scores less than 0.05 are screened out.

Classification Results = sum(atleast one tp per image)/num images
Now we check the ability of the network to identify atleast one of the GT regions in an image. Atleast one detection per image should be of the corresponding class in top-n confidence list (ie, if there are three GT regions, there will be 3*38 scores, where 38 is the number of classes. we sort for top n in this list).
Top 10 classification result is 83.1%. Top5 classification result drops to 79% (from 84.8% in full image) without background information. Top 1 drops from 59.6% to 51.9%

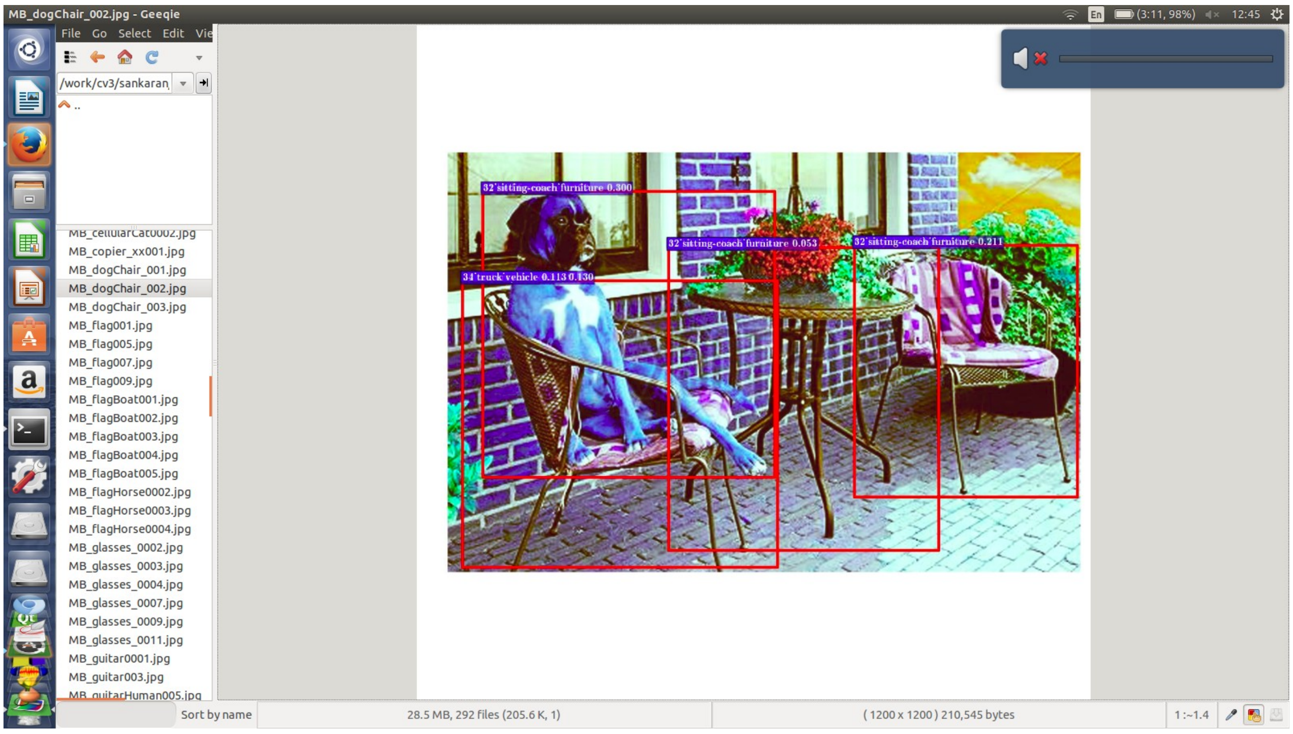
Sensitivity(Recall) = tp / all positive conditions (note: all positive conditions > num images)

First considering only top 10 detection scores. Sensitivity for each class is evaluated as follows: suppose there are 3 GT regions of the corresponding class in an image, not identifying even one of the three regions in top 10 list penalizes the sensitivity score.

Since we input only the GT bounding box, we expect the result with full overlap. all IoU scores were between 0.98 and 0.99 because of round off errors.
For complete overlap, sensitivity is 74%. The corresponding top 10 classification result was 84%. But this is because there can be multiple objects of same class in an image as mentioned earlier. Hence, the multiple positive conditions in an image should be satisfied for success(unlike the case of classification performance where just one positive condition per image is needed for success.)
The MAP is 57%

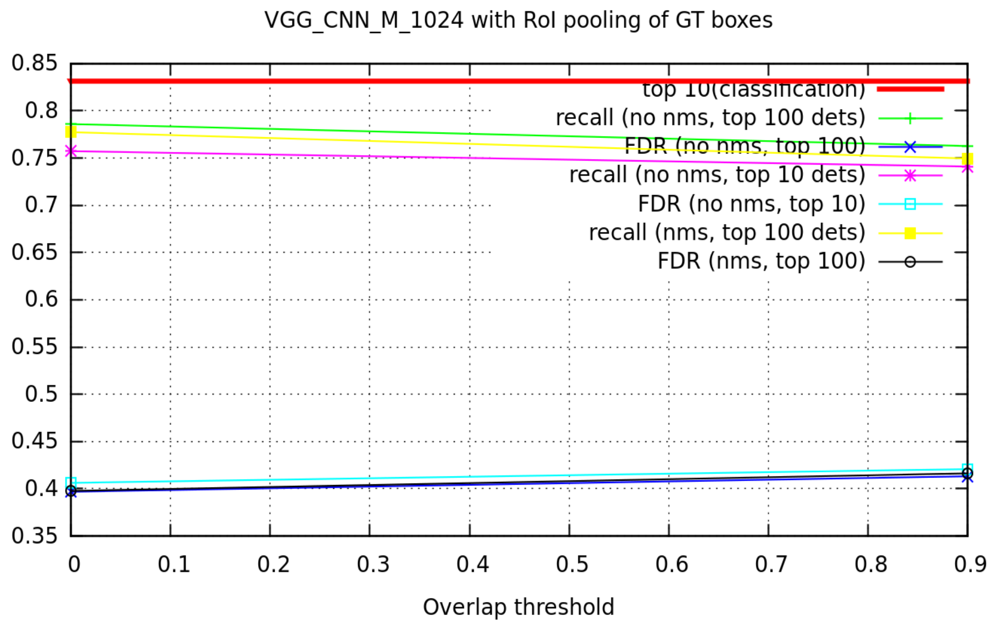


Although we expect the recall to be constant for all overlap thresholds, we see a slightly higher scores as threshold decreases. This is because of the following reason: consider the following image or the evaluation for class chair:



There are two GT regions of class chair and one GT region of class table that overlaps slightly with one of the chair regions. The chair on the left is never classified correctly, but the table gets better classified as a chair in top10. Since this region overlaps with the chair on the left, it gets counted as a true positive as we decrease the overlap threshold. This is reflected by decrease in false discovery rate (fp/(tp+fp)) as overlap threshold decreases.
Now, when taking top 100 instead of top 10, the sensitivity for full overlap increases from 74% to 76%
So far we have not applied NMS. Now with NMS (0.3) we do not see any changes in the classification performance. But Sensitivity decreases from 76% to 75% on applying NMS and sorting top 100. This result confirms the claim in the paper that "NMS does not harm the ultimate

detection accuracy".

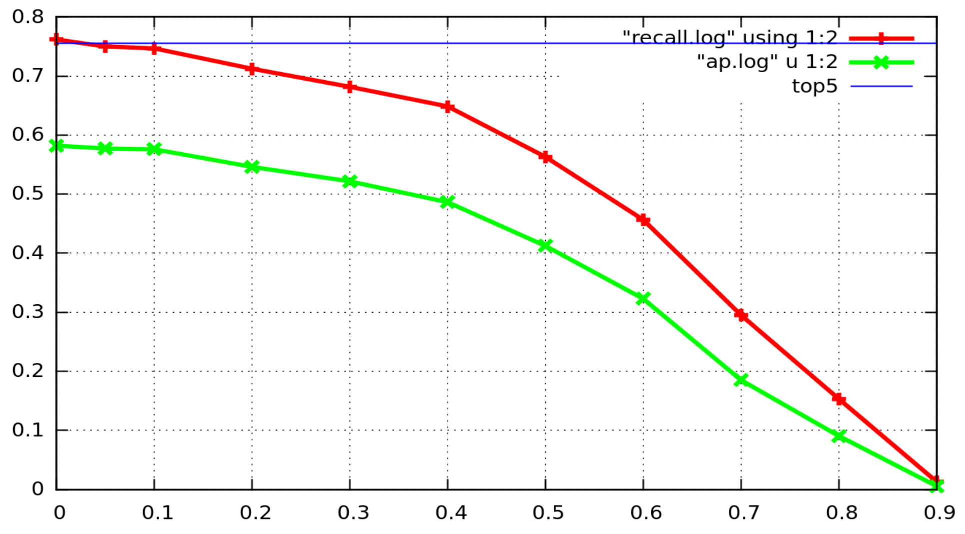


****WITH RPN****
Now instead of GT proposals, we use the proposals from RPN and train end to end with following config: 3 scales and 3 aspect ratios, feat stride: 16 (stride in the last convolutional layer. This also determines the number of generated proposals). Proposals with scores less than 0.05 are screened out.

Testing: Consider the top 100 proposal scores and with nms:0.3

Classification results:
Top 1: 54% (increases from 51.9% with only GT proposals. For full image, it was 59.6%)
Top 5: 76.7% (decreases from 79%. For full image, it was 84.8%)
Top 10: 84.7% (increases from 83.1%)
(these improvements might be because anchors larger than GT boxes might have classified better)

Sensitivity results: (with top 100 and nms 0.3) For 50% overlap, MAP: 41%, Recall: 57%



<u>**Experiment: VGG_16**</u>

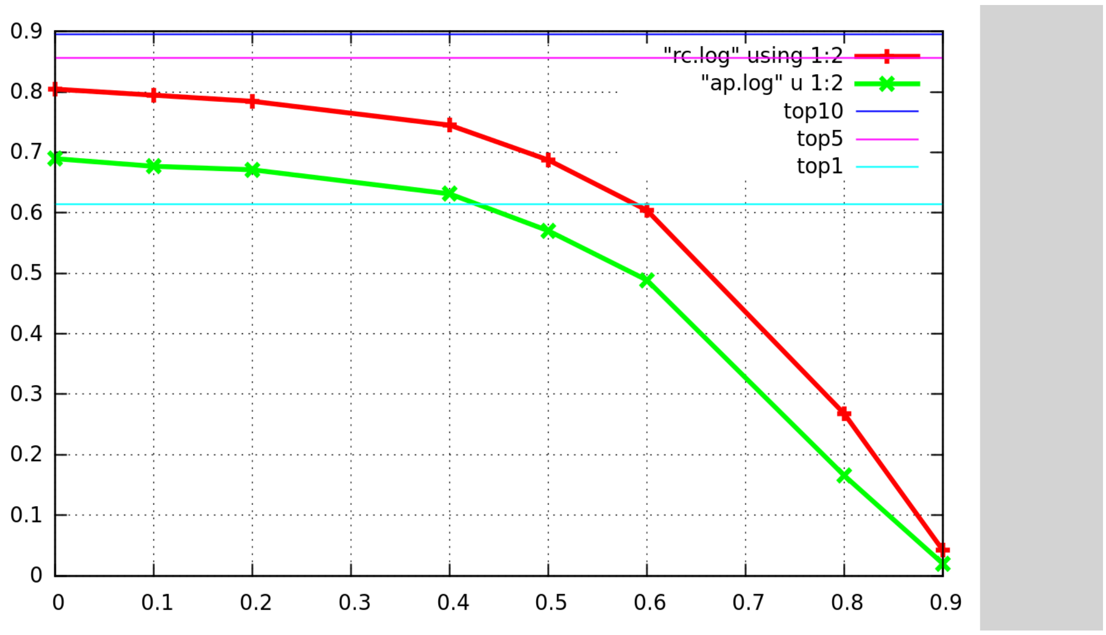
Full image Classification Results of VGG_CNN_M_1024
Top 1 = 0.6558
Top 5 = 0.8832

The experiment with RPN is repeated by replacing VGG_CNN_M_1024 with VGG16. All configurations remain same.

Classification results:
Top 1: 61.3% (increases from 54%)
Top 5: 85.5% (increases from 76.7%.)
Top 10: 89.7% (increases from 84.7%)

Sensitivity results: (with top 100 and nms 0.3) For 50% overlap,

MAP: 57%(Vgg 1024 – 41%) , Recall: 68.7% (vgg 1024 - 57%)



<u>Diagnostic study of Anchor scales, ratios and feat stride:</u>

anchor generation: (lib/rpn/generate_anchors.py)
<u>base anchor: [0,0,15,15]</u>

```
    anchor base center = ((0+ 0.5*(15-0 + 1)), (0+ 0.5*(15-0 + 1))
                        → (8,8)
    anchor base size; (16,16)  [.. (h,w)]
    apply ratios: [1:2, 1:1, 2:1] --> [0.5,1,2]
                 (16*16)/[0.5,1,2] = [512,256,128]
                  ws = [sqrt(512),sqrt(256),sqrt(128)]
                  hs = ws*[0.5,1,2] = [22*0.5, 16*1, 11*2]
    new anchor sizes = [(22,11),(16,16),(11,22)]

    apply scales: [8,16,32] (applied on new anchor sizes)
                   on (16,16) → [(16*8,16*8), (16*16, 16*16), (16*32, 16*32)]
                               →[(128,128), (256,256), (512,512)]
    scaled anchor sizes[   [(176,88),(352,176),(704,352)]
                           [(128,128),(256,256),(512,512)]
                           [(88,176),(176,352),(352,704)] ]

   Thus, using 3 scales and 3 ratios, we get 9 anchors for every center.
   Generated Anchors (data size) :(1,9,4)
```

Size of rpn_bbox_pred layer determines the number of centers.
For VGG1024, rpn_bbox_pred: (1,4*9,34,61) = (1,36,34,61), there will be 34*61 centers.

Feat Stride determines the distribution of the centers to the original image.
(proposal_layer.py)

```
  shifts_x = np.arange(0,34)*feat_stride
  shifts_y = np.arange(0,61)*feat_stride
  shifts_x, shifts_y = np.mehsgrid(shifts_x, shifts_y)
```

These shifts are applied to the each of the 9 generated anchors.
Anchors = generated anchors + shifts (datasize: (9*34*61,4))

So in total, there will be 9*34*61 = 18666 Anchors

output of rpn_bbox_pred (1,4*9,34,61) are the bbox deltas, which are applied to the anchors.
This output is rearranged as (9*34*61,4) which is now same as the anchor data size.

bbox_transform_inv: (bbox_transform.py)

```
    input: anchors(18666,4), deltas(18666,4)
    the dimension and centers of the anchors are calculated from their bbox coordinates
    anchor_widths, anchor_heigths, anchor_cts_x, anchor_cts_y = anchors(...)
    for i in anchors:
        pred_ctr_x[i] = deltas[i][0]*anchor_width[i] + anchor_cts_x[i]
        pred_ctr_y[i] = deltas[i][1]*anchor_height[i] + anchor_cts_y[i]
        pred_width[i] = exp(deltas[i][2])*anchor_width[i]
        pred_heigth[i] = exp(deltas[i][3])*anchor_heigth[i]
```

Screening in the proposal layer:

screen1: remove proposals(predictions) with heigth or width < thresholds
screen2: sort for top 6000 (pre_nms_TopN)
screen3: apply nms (RPN_NMS_THRESH)
screen4: sort for top 300

Thus, to be more precise, the input to the network is the image and a set of generated anchors. The output is a set a shifted anchors, that passes all the screening tests. The anchor scales, ratios and feat stride affect only the anchor generations. So it really doesn't affect the performance of the network too much. The performance depends more on how the network finds the right bounding box by changing the set of generated anchors. (ie, on how the deltas are learned for the set of generated anchors).

VGG1024 network diagnosis:

re scaled data: (1,3,562, 1000)
conv1: (1,96,276,497)
pool1:(1,96,139,248)
conv2(1,256,69,123)
pool2(1,256,34,61)
conv3(1,512,34,61)
conv4(1,512,34,61)
conv5(1,512,34,61)
    rpn_cls_prob(1,K*2,34,61)   rpn_bbox_pred(1,K*4,34,61)
    rois(300,5)
roiPool(300,512,6,6)
fc6(300,4096)
fc7(300,1024)
cls_score(300,38)
bbox_pred(300,152)

thus changing anchor scales and ratios only changes the number K in the network (and some reshape dimensions) feat stride actually does not affect the network architecture at all (since it is just a mapping)

So we get 300 detections for each image.

After detection screening:

screen5: apply nms
screen6: sort top100 detections

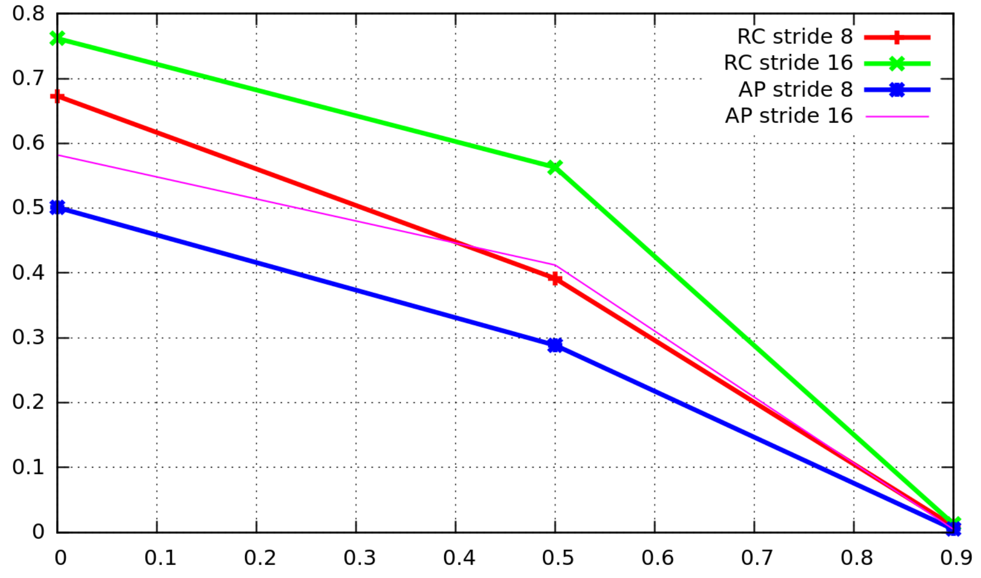
<u>Experiments:</u>

<u>Effect of Feat Stride on VGG_CNN_M_1024:</u>

<u>Classification perf for stride 8:</u>

Top1: 45.6% (it was 54% for stride 16)
Top5: 74.9% (it was 76.7% for stride 16)
Top10: 81.4% (it was 84.7% for stride 16)

<u>Recall and MAP:</u>



Remarks:
The feat stride dos not increase the number of anchors! But just qualifies more candidates for screening in proposal layer. Varying nms at roi screening was not studied.

<u>Experiments with single scale on VGG1024</u>

<u>Scale [8]</u>
<u>base anchor: [0,0,15,15]</u>

anchor base center = ((0+ 0.5*(15-0 + 1)), (0+ 0.5*(15-0 + 1))
→ (8,8)
anchor base size; (16,16) [.. (h,w)]
apply ratios: [1:2, 1:1, 2:1] --> [0.5,1,2]
(16*16)/[0.5,1,2] = [512,256,128]
ws = [sqrt(512),sqrt(256),sqrt(128)]
hs = ws*[0.5,1,2] = [22*0.5, 16*1, 11*2]
new anchor sizes = [(22,11),(16,16),(11,22)]

apply scales: [8] (applied on new anchor sizes)
scaled anchor sizes[[(176,88)]
[(128,128)]
[(88,176)]

Hence only three anchors for every center.
The experiment was repeated for scale 8 and scale 16

	Scale 8	Scale 16
Classification (top1)	51.9%	50.44%
Classification (top5)	76.41%	77.01%

Remarks:
In the paper, the mAP improved while different scales of Anchors were used. The improvement might also be because of just adding anchors with larger scales. → This doesnt seem to be true. We see top1 performance is better for scale 8. Moreover, the scales only affect the anchor generation.

Only the learned deltas affect performance.
Since we have to train the network again for ever scales or feat stride, it may not be possible to have a direct comparision of results.

Summary

			VGG1024	**VGG16**
Full image	Classification	top1	59.62	65.58
		top5	84.82	88.32
With GT regions	Classification	top1	51.9	
		top5	79	
		Top10	83.1	
	Recall(without test nms, full overlap)	Top10*	74	
		Top100*	76	
	Recall(with default test nms, full overlap)	Top10*		
		Top100*	75	
	MAP(without test nms, full overlap)	Top10*	57	
		Top100*	57.8	
	MAP(with default test nms, full overlap)	Top10*		
		Top100*	57.4	
With RPN	Classification	Top1(default)	54	61.3
		Top5(default)	76.7	85.5
		Top10(default)	84.7	89.7
		Top1(scale8)	51.9	
		Top5(scale8)	76.41	
		Top1(scale16)	50.44	
		Top5(scale16)	77.01	
		Top1(default scale, ft_stride halved)	45.6	
		Top5(default scale, ft_stride halved)	74.9	
		Top10(default scale, ft_stride halved)	84.1	
	Recall(default, 50% Iou)	Top100*	57	68.7

	MAP(default, 50% Iou)	Top100*	41	57

*for evaluating recall, we expect detections for all GT boxes in an image. So this cannot be directly compared with TopN classification results, where we expect only one detection of right class per image. And moreover, for classification results, the dectections are not compared with GT boxes