The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

R-FCN++: Towards Accurate Region-Based Fully Convolutional Networks for Object Detection

Zeming Li,[1] **Yilun Chen,**[2] **Gang Yu,**[2] **Yangdong Deng**[1]
[1]School of Software, Tsinghua University, {lizm15@mails.tsinghua.edu.cn, dengyd@tsinghua.edu.cn }
[2] Megvii Inc. (Face++), {chenyilun, yugang}@megvii.com

Abstract

Region based detectors like Faster R-CNN (Ren et al. 2015) and R-FCN (Li et al. 2016) have achieved leading performance on object detection benchmarks. However, in Faster R-CNN, RoI pooling is used to extract feature of each region, which might harm the classification as the RoI pooling loses spatial resolution. Also it gets slow when a large number of proposals are utilized. R-FCN is a fully convolutional structure that uses a position-sensitive pooling layer to extract prediction score of each region, which speeds up network by sharing computation of RoIs and prevents the feature map from losing information in RoI-pooling. But R-FCN can not benefit from fully connected layer (or global average pooling), which enables Faster R-CNN to utilize global context information.

In this paper, we propose R-FCN++ to address this issue in two-fold: first we involve Global Context Module to improve the classification score maps by adopting large, separable convolutional kernels. Second we introduce a new pooling method to better extract scores from the score maps, by using row-wise or column-wise max pooling. Our approach achieves state-of-the-art single-model results on both Pascal VOC and MS COCO object detection benchmarks, 87.3% on Pascal VOC 2012 test dataset and 42.3% on COCO 2015 test-dev dataset. Code will be made publicly available.

Introduction

We are witnessing significant progresses in generic object detection, partly due to the rapid development of deep convolutional neural networks. Among various CNN-based object detectors, one of the most notable work is Region-based Convolutional Neural Network (R-CNN) (Girshick et al. 2014). R-CNN exploits deep CNN to classify regions. Following region-based detection pipeline, Faster R-CNN (Ren et al. 2015) and R-FCN (Li et al. 2016) are proposed to further improve the accuracy and speed.

Compared with single-stage pipeline such as YOLO (Redmon et al. 2016) and SSD (Liu et al. 2016), Faster R-CNN achieves better performance by a second-stage classification and a bounding-box regression. However, Faster R-CNN is time-consuming when the number of object proposals in its first-stage is large. Also,

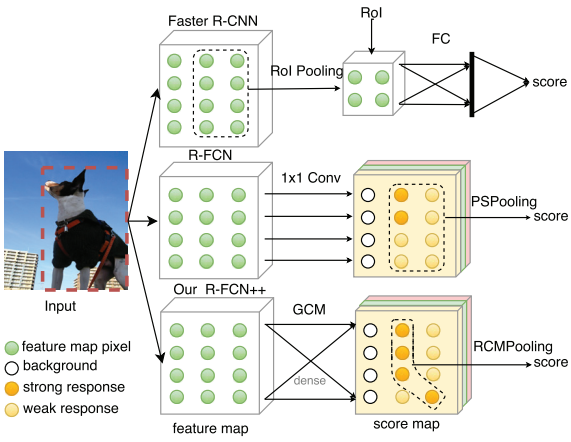

Figure 1: Comparison of Faster R-CNN, R-FCN, and our R-FCN++. Faster R-CNN uses RoI pooling layer to sample feature, then involving fully connected layer to do final prediction. R-FCN uses 1×1 convolution to generate score maps. Scores are obtained by average voting every part of the RoI, and each part is pooled by Position Sensitive RoI Pooling (PSPooling). Our R-FCN++ includes Global Context Model (GCM) to generate score map and Row-Column Max Pooling (RCMPooling) to pool the parts of high response for the final prediction.

Faster R-CNN needs to perform a RoI-pooling (may lose information due to regular sampling) on the cropped features, which might be harmful to the region classification. To avoid those shortcomings, R-FCN uses Fully Convolutional Network (FCN) to generate a classification score map and a location map. It efficiently gets the final prediction by simply averaging the scores of the object parts. However, the part in R-FCN might be too local to be discriminative enough because there is no fully connected layer or global average pooling layer to provide the global context.

In order to improve the classification capability of R-FCN, we need to let the network to see a larger context. However, it is inherently contradictory to employ a fully connected layer or a global average pooling into a FCN. Also, taking larger convolution kernel is expensive and

tends to seriously decrease the computation efficiency. To tackle such a dilemma, we employ a Global Context Module (GCM) to enlarge the kernel and also maintain its efficiency.

Given the score maps, it is worth noting that objects usually do not occupy the whole box. R-FCN pools every part in the box, which is not optimal for the final prediction. Different from Position Sensitive Pooling used in R-FCN, we introduce Row-Column Max Pooling (RCMPooling). Since objects should have at least one part in every row and column in box, RCMPooling only pools the max score in every row and column followed by averaging. Fig. 1 provides the general ideas of our algorithm and two baselines (Faster R-CNN and R-FCN).

In summary, this paper introduces an improved R-FCN detection method we called "R-FCN++" to tackle the following two issues: (i) Fully convolutional networks are lack of context information. (ii) Object's score can't correspond well to the box's score, because we pool from the box instead of the object. Our contributions are:

1. Considering the importance of the global context, we propose to use Global Context Module to improve score maps for classification. Furthermore, we present Light-GCM, which is a light version of GCM, to strike a good balance between accuracy and speed.
2. Given the score maps generated by GCM, we introduce a new pooling method called RCMPooling to extract proper scores for RoIs, which further improve the classification performance.
3. By integrating the proposed two improvements, our system (single model) significantly outperforms all recent state-of-art detectors on both Pascal VOC and MS COCO datasets..

Related work

With the rapid progress of convolutional neural networks (CNN), object detection has been largely improved. Some of the recent research papers for object detection are listed as follows.

Region-based Convolutional Neural Network: R-CNN has been seen as a milestone for CNN object detection. In R-CNN, hand-crafted methods such as Selective Search (Uijlings et al. 2013), Edge Boxes (Zitnick and Dollár 2014), and MCG (Arbeláez et al. 2014) are first involved to generate proposals. The whole network has to be forwarded for each proposal. To keep enough information of images and avoid repetitive computation, SPPnet (He et al. 2014) proposes Spatial Pyramid Pooling layer to pool the features for proposals to a fix shape before we do prediction, which is similar to the RoI pooling. But similar as R-CNN (Girshick et al. 2014), SPPnet (He et al. 2014) uses multi-stage training to do object classification and bounding box regression separately. Fast R-CNN (Girshick 2015) introduces a multi-task learning for joint training object classification and bounding box regression, which reduce training time and improve the detection accuracy simultaneously. However, generating proposals costs a lot of time and is independent of CNN. To further speed up proposal generation, Faster R-CNN (Ren et al. 2015) proposes a novel Region Proposal Network(RPN), which can be embedded into the Fast R-CNN framework seamlessly.

Proposal-free Convolutional Neural Network: Region based detectors rely on pre-computed object proposals, which involve additional computations. YOLO (Redmon et al. 2016) and SSD (Liu et al. 2016) directly predict object's position and category. Since there are no region proposals, YOLO-style models do not have to warp the features used in RoI Pooling. Though proposal-free models are usually faster than R-CNN style models, it seems that region-based detectors are more accurate.

Region-based Fully Convolution Network: R-FCN (Li et al. 2016) is another region based detection framework. Different from traditional R-CNN pipelines, instead of warping CNN features for RoI, R-FCN use Position Sensitive Polling to crop the score for region proposals. Thus R-FCN has a more unified structure. The shared computations among RoIs make it more efficient in predicting the labels and positions of RoIs. However, as mentioned above, R-FCN can not benefit from fully connected layer (or global average pooling) in classification which is widely used in typical R-CNN style detectors.

Our approach

In this section we will present the details of our method. Fig. 2 D shows our overall R-FCN++ framework. We use ResNet (He et al. 2016) as our basic feature extractor, which is shown as "Conv layers" in Fig. 2 D. We denote the feature maps from the last residual blocks of conv4 and conv5 as C_4 and C_5 respectively.

RoI-wise Subnetwork is a region-proposal detector that use features from C_5. R-FCN employs a 1×1 convolutional layer (the number of output channels is 1024) to generate score maps for classification and regression. Different from R-FCN, we apply Global Context Module (GCM) to enable score map to utilize global context information. The GCM is illustrated in Fig. 2 A. As GCM involves more computation in networks, we further introduce a light version of GCM (Light-GCM), which is illustrated in Fig. 2 B. Following GCM, we use two sibling 1×1 convolutions to adjust output channels for regression and classification respectively. All classes share one box except the background (we also predict regression for background class). We set the number of channels in regression to $8 \times p \times p$ (p is number of parts we divide each RoI into and $p = 7$ in our experiments). As for classification, we set the number of channels to $(classes + 1) \times p \times p$. After getting the score maps, we apply Row-Column Max Pooling (RCMPooling) to get the final predictions. See Fig. 2 C for more details.

In the following subsections, we will present the details of GCM (Light-GCM) and RCMPooling.

Global Context Module

Classification is a basic recognition task in deep learning and has made a great breakthrough. Recent frameworks like AlexNet (Krizhevsky, Sutskever, and Hinton 2012),VGGNet (Simonyan and Zisserman 2014),GoogleNet (Szegedy

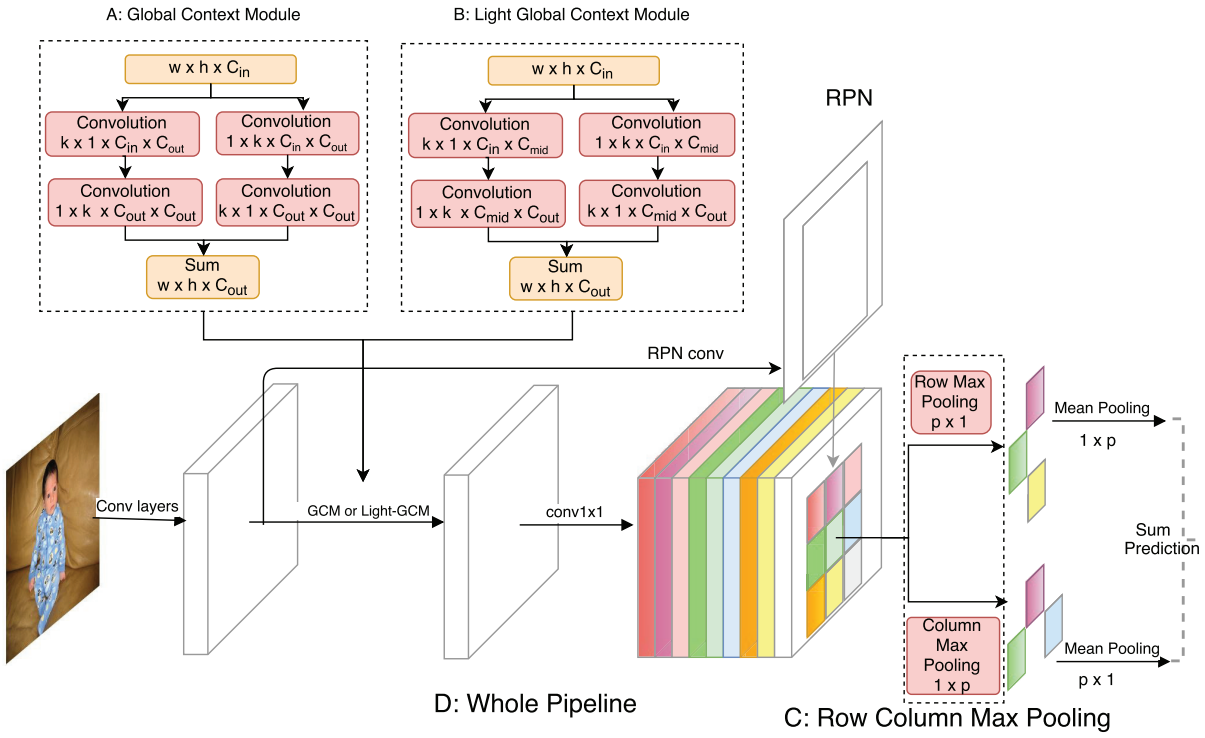

Figure 2: overview of our whole pipeline (D). Global Context Module (GCM), Light version of GCM (Light-GCM) and Row-Column Max Pooling are illustrated in (A), (B) and (C).

et al. 2015) and ResNet (He et al. 2016) first use stack of convolution layers to extract feature which is lack of global context information (Zhou et al. 2014; Peng et al. 2017), then use fully connected layer to globally handle large variations of transformations. In R-CNN (Girshick et al. 2014) style pipelines, fully connected layer is used to improve classification capability. Classifiers need to see feature map globally rather than locally to deal with spatial information and different types of input transformations.

In our design, RoI pooling is directly applied on score map instead of feature map to share computation between RoIs in RoI-wise subnetwork. De facto, predicting score map equals to classify label of each pixel in score map. As mentioned above, global context is important for classification. Thus we need to enable network to see larger. One possible way is exploiting the Fully-Connected layer (FC) to generate the score map. But in order to make the network totally fully convolutional, we can not simply employ the FC into the network.

To overcome those drawbacks, we present Global Context Module (GCM) to take advantages of fully connected layer into FCN detection framework. Design details are shown in Fig. 2 A. By involving separable kernel used by (Szegedy et al. 2016; Peng et al. 2017), GCM keeps the time efficiency. Meanwhile, we can involve more context in network by increasing parameter k (kernel size is controlled by k). When we enlarge the kernel size to the size of feature map, GCM acts just like a fully connected layer. Thus the network can deal with the spatial information better. As large separable convolution still runs slower than small convolution, we propose a light version of GCM, called Light-GCM as in Fig. 2 B. We can control the cost of computations by regulating the number of middle channel C_{mid} in Light-GCM. When C_{mid} is set to 64, the speed of Light-GCM is comparable to 1×1 convolution with 1024 output channels used in R-FCN.

Row-Column Max Pooling

Since we have introduced GCM to help the FCN network to get a better score map for RoI in last section, the next question is how to better pool from score maps and get the final prediction. In R-FCN, position-sensitive pooling will pool $p \times p$ score maps and each part of the map will be presented as a semantic part in object. But many boxes are not filled with objects. One example is shown in Fig. 3. The right-top part of the box has low score (The lighter the color on the graph represents, the lower the score), because actually it does not locate inside the object. This situation often occurs and is worse once we divide RoI into shape of 7×7 or finer. Intuitively the part is not well predicted such as the right-top can be harmful for the final prediction after average voting. If we simply use max voting instead, it will get very high classification score when only one part of RoI is predicted correctly.

Witnessing these disadvantages, we propose Row-

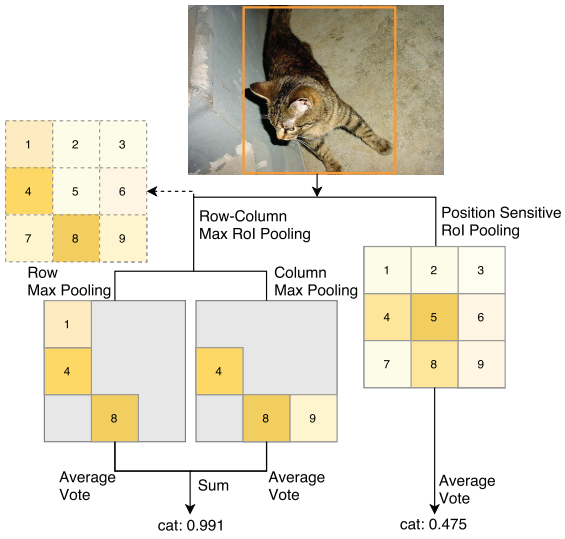

Figure 3: Difference between PSPooling (Position Sensitive Pooling) and RCMPooling (Row-Column Max Pooling). PSPooling layer pools the $p \times p$ parts (here $p = 3$). RCMPooling layer only pools the part with the maximum scores in every row and column, so RCMPooling layer pools $p \times 2$ parts.

Column Max Pooling (RCMPooling) as in Fig. 2 C. The channels of score maps are divided into $(classes + 1)$ groups to learn different classes independently. Following the method of position-sensitive RoI pooling, we divide the RoI of size $w \times h$ into $p \times p$ parts like a square grid. Each part has the size of $\frac{w}{p} \times \frac{h}{p}$. The scores between these parts are learned in separable channels independently, that is, $p \times p$ channels are used to learn different parts' score of RoIs. As is shown in Fig. 3, RCMPooling only pools the darkest part of each row and column while PSPooling pools all 3×3 parts.

Different from R-FCN pooling each part from position sensitive score maps, we only pool the parts of max scores in each row and column from our score maps. The idea of RCMPooling is based on the observation: when there exists an object in the box, in the voting score maps, every row and column should have at least one part of the object, while the object does not fill every row and column at most of the time.

Experiments

Our experiments use publicly available py-R-FCN code[1]. Following the setting of R-FCN, all experiments adopt *àtrous* (Mallat 1999; Long, Shelhamer, and Darrell 2015; Chen et al. 2014) algorithm and online hard example mining (OHEM) (Shrivastava, Gupta, and Girshick 2016). Our models are initialized with standard ResNet-101 (He et al. 2016) or ResNet-50 which is pre-trained on the ImageNet (Russakovsky et al. 2015) classification dataset. Only one Pascal TITAN GPU is used to train our model. We use a weight decay of 0.0005 and a momentum of 0.9. The detection results are measured by mean Average Precision(mAP). For single scale training, the shorter edge of image is resized to 600 pixels. During testing, we using single scale of 600 pixels unless explicitly stated.

In the next subsection, we first perform a series of ablation experiments to test the effectiveness of our methods. And the final results of ours are reported in public benchmarks: Pascal VOC 2007, Pascal VOC 2012 test dataset, and COCO (Lin et al. 2014) 2015 test-{dev, std} set.

Ablation Experiments

We validate our method on Pascal VOC (Everingham et al. 2010; 2015) and MS COCO (Lin et al. 2014) dataset. For the experiments on VOC, our algorithm is trained on Pascal VOC 2007+2012 trainval dataset, and tested on Pascal VOC 2007 test dataset. Learning rate is set to 0.001 for first 80k iterations and 0.0001 for later 40k iterations. The iter_size is set to 2. For the COCO experiments, our algorithm is trained on COCO train set and test on minival set. Learning rate is set to 0.0005 for first 1.2m iterations and 0.00005 for later 720k iterations. The iter_size for COCO is set to 1. Unless explicitly stated, all the experiments in this sub-section are evaluated on the VOC dataset.

k	1	3	5	7	9	11	15
ap	79.1	79.3	79.6	80.3	80.6	80.6	80.7

Table 1: Comparison of different kernel sizes used in GCM. As the kernel size increases, more context information is involved for prediction.

Global Context Module Our overall network structure is illustrated in Fig. 2 D. We adopt R-FCN as our baseline to discuss the effectiveness of GCM.

Global context information Design Choice. To discuss the influence of global context for R-FCN individually, we use PSPooling (pooling shape is 7×7) to pool our score instead of RCMPooling. We control the kernel size in GCM by enumerating different k. With expanding the kernel size, GCM acts more like a fully connected layer which involves more context information. We try the different kernel sizes ranging from 3 to 15. Since we use $àtrous$ algorithm in $conv5$ stage, kernel size 15 can almost cover the whole feature map. Thus we enable score maps to use global context features for their prediction. The results are shown in table 1. Enlarging kernel consistently improve the accuracy. When the kernel size we use is set to 15, our model surpasses the baseline by 1.6%. The results prove that global context information is important for R-FCN.

To clarify the improvement is not mainly due to the structure of separable convolution. We also test naive large kernel convolution shown in Fig. 4 B. As shown in table 2, we can see naive large kernel ($k \times k$ Conv) also consistently improves the accuracy.

As we increase the kernel size k, the number of parameters increases at the same time. Another hypothesis is that

[1]https://github.com/Orpine/py-R-FCN

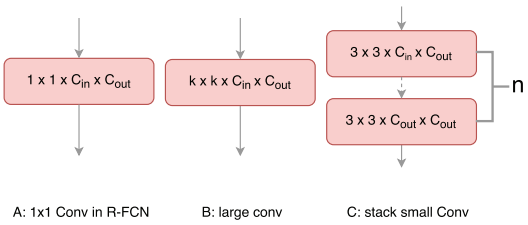

Figure 4: Different kernels we test to generate score maps. The 1×1 convolution in R-FCN is illustrated in (A), naive large convolution is illustrated in (B), stack of 3×3 convolution is illustrated in (C).

kernel size	3	5	7	9
GCM	79.3	79.6	80.3	80.6
$k \times k$ Conv	79.4	80.3	80.1	80.6
stack	79.4	79.6	79.7	79.7

Table 2: Comparison of different methods that enlarge the kernels. GCM uses large separable convolution whose computation is acceptable. $k \times k$ convolution is much slower for lots of parameters. Stack is the approximated way which uses a stack of small convolutions. i.e. 5×5 convolution can be approximated two 3×3 convolution.

the improvement is mainly due to more parameters instead of context information. In order to analyse the influence of number of parameters, we also test stack of 3×3 convolutions that also increase parameters as shown in Fig. 4 C. Actually the 5×5 convolution can be approximated as a stack of two 3×3 convolution. But there are some differences between stack version and GCM in valid receptive filed. Different from directly enlarging kernel, stack version tends to focus on local features (Zhou et al. 2014). Thus stack version has less context than large kernel. Table 2 shows the comparison results of GCM and stack convolution. From table we can see stack version slightly improves the accuracy. So more parameters are not the key-point to improve the performance.

We also test the influence of reducing the middle channels in GCM by using Light-GCM in Fig. 2 B. We compare different number of channels used in middle output layer. When the number of middle channels is decreased to 64, Light-GCM has only a few parameters and is comparable to 1×1 convolution used in R-FCN based on speed. Results are shown in table 3. According to the table, though we reduce the parameters in GCM by using Light-GCM, our model still significantly outperforms the baseline. Thus context information plays an important role in R-FCN.

C_{mid}	R-FCN	64	128	256	512	1024
score	79.1	80.3	80.4	80.5	80.6	80.7

Table 3: Comparison of different numbers of middle output channels (c_{mid}) used in Light-GCM. R-FCN is our baseline that uses 1x1 convolution with 1024 output channels.

How does Global Context improve the performance? We claim the good performance of GCM is mainly due to the improvement of the classification capability. Fig. 5 shows the score maps after PSPooling. When employ GCM, the prediction of the score map is more accurate compared with R-FCN. We also investigate the different IoU thresholds used in testing. When threshold is set to 0.1 and 0.3, most of the groundtruth box will be matched. So the comparison is mainly based on the classification ability. Table 5 shows the results. Clearly, combining with GCM improve the classification capability. In traditional classification task, feature map needs to be densely connected to handle large variations of transformations. GCM is much like the fully connection layer used in classification, which brings more context for score map's label prediction.

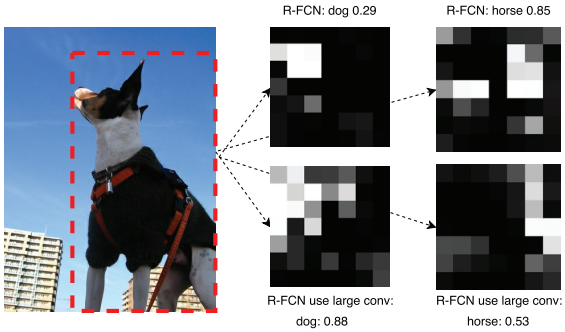

Figure 5: Score maps after PSPooling, above is 7×7 score map produced by R-FCN, below is produced by R-FCN using Global Context Model.

Row-Column Max Pooling Here we test RCMPooling on our framework shown in Fig. 2 D. R-FCN is adopted as our baseline. In subsection , we propose RCMPooling to try extracting score from score maps more properly. The key idea is to avoid pooling background part in RoI which probably harms the following voting. As we have introduced GCM to get better score maps, we expect that combining GCM with RCMPooling should further improve the final prediction accuracy. To verify these claims, first we remove the GCM model (replace GCM with 1×1 convolution used in R-FCN) and only test the influence of RCMPooling (We set p to 7 in RCMPooling). Then we add GCM model to check if they would conflict with each other. The results are shown in table 4. RCMPooling improves the accuracy and it works compatible with GCM.

method	VOC score	COCO score
baseline (R-FCN)	79.1	28.0
+GCM (k=15)	80.7	29.5
+RCMPooling	79.6	28.9
+RCMPooling + GCM (k=15)	81.2	30.4

Table 4: Comparison between models with RCMPooling and without RCMPooling. Baseline is R-FCN using PSPooling 7x7.

overlap thresh	0.1	0.3	0.5	0.7	0.9
baseline (R-FCN)	84.9	83.9	79.1	62.0	12.8
+RCMPooling	85.2	84.2	79.6	63.6	13.6
+GCM	85.7	84.7	80.7	64.3	13.0
+RCMPooling+GCM	86.0	85.0	81.2	65.1	14.3

Table 5: Comparison of different IoU thresholds used to test mAP.

RCMPooling provides a better prediction for voting (both classification and regression). We use a different IoU threshold to test mAP. When IoU threshold is set to 0.9, predicted box needs to be accurate in location. When IoU threshold is set to 0.1, predicted box needs to be accurate in classification. According to the table 5, we set IOU threshold ranging from 0.1 to 0.9, RCMPooling improves the accuracy consistently.

Pascal VOC

In this section, we evaluate our model on Pascal VOC2007 and VOC2012 (Everingham et al. 2010; 2015) dataset. Following (Li et al. 2016; Ren et al. 2015; Liu et al. 2016), we pre-train our model on MS COCO (Lin et al. 2014) dataset. Then fine-tune the model on VOC 2007 trainval and VOC 2012 trainval ("07+12"). For multi-scale training, following the method used in R-FCN, in each iteration, we sample the scale randomly from {400,500,600,700,800} pixels, and resize the short edge of image into the scale we sampled. For testing, we only using single scale 600 pixels.

Results are shown in table 6. Our single test model gets 84.9% on VOC2012 test, which significantly surpass R-FCN and Faster R-CNN. Meanwhile, we try the post-processing such as left-right flipped images, multi-scale testing and box voting, which also bring the benefits for final results. On Pascal VOC 2012 test set, we achieve 87.3%. Compared with all the recent state-of-the-arts shown in table 7, our single model reaches the new state-of-art.

method	training data	07test	12test
Faster R-CNN	07+12	76.4	73.8
Faster R-CNN +++	07+12+COCO	85.6	83.8
R-FCN	07+12	79.5	77.6
R-FCN multi-sc train	07+12	80.5	78.8
R-FCN multi-sc train	07+12+COCO	83.6	82.0
RFCN++	07+12	81.2	79.7
RFCN++ multi-sc train	07+12	82.1	80.6
RFCN++ multi-sc train	07+12+COCO	86.3	84.9

Table 6: Experimental results mAP(%) on VOC2007 test dataset and VOC2012 test dataset . R-FCN++ involves both GCM and RCMPooling.

Efficiency Study. Another thing we concern is the speed of inference. GCM has many parameters which results in additional overheads in network. With a slightly trade-off in accuracy, we introduce a fast version of our algorithm, which uses ResNet-50 as the basic feature extractor and Light-GCM to speed up network inference. We set the number of middle channels in Light-GCM to 64, noticing that

method	mAP(%)
YOLOv2 (Redmon and Farhadi 2016)	78.2
SSD512(VGG16) (Liu et al. 2016)	82.1
Faster R-CNN +++ (Ren et al. 2015)	83.8
R-FCN multi-sc train,test (Li et al. 2016)	85.0
GCM+RCMPooling multi-sc train,test	**87.3**

Table 7: Comparisons of single-model on VOC2012 test. All models are trained on VOC07+12+COCO dataset.

Light-GCM has less parameters and is faster in network inference. RCMPooling is also involved in our fast version. The Results are shown in table 8. Our fast version obtains competitive results of 79.8% based on VOC2007 test set, achieving speed of 0.072s per image at a test-time, which is faster than R-FCN. Compared with R-FCN which is based on ResNet-101, our fast version gets comparable result by using ResNet-50. It shows that our method is more economical than increasing the basic feature extractor complexity.

method	base model	mAP	test time
R-FCN	ResNet-101	79.5	0.093
GCM+RCMPooling	ResNet-101	**81.2**	0.130
Light-GCM+RCMPooling	ResNet-101	80.7	0.104
Light-GCM+RCMPooling	ResNet-50	79.8	**0.072**

Table 8: Comparisons of inference time. Our model uses GCM and RCMPooling. Compared with R-FCN, our model runs slightly slower but provides better results. Our fast version use Light-GCM and RCMPooling, the number of middle channel in Light-GCM is set to 64. Fast version yields competitive results and is also faster than R-FCN.

method	test-dev				
	$AP_{@.5}$	AP	AP_s	AP_m	AP_l
R-FCN	51.5	29.2	10.3	32.4	43.3
R-FCN ms train	51.9	29.9	10.8	32.8	45.0
R-FCN++	53.9	32.2	12.2	35.4	47.6
R-FCN++ ms train	56.6	**34.0**	14.1	36.9	49.1

Table 9: Comparison between our model and baseline R-FCN based on single-scale test. All methods use COCO2014 trainval dataset to train, and COCO2015 test-dev dataset to test.

MS COCO

We evaluate our model on MS COCO (Lin et al. 2014), following the training strategy provide by py-R-FCN. The single-scale test results are shown in table 9. Our single-scale trained R-FCN++ yields results of 53.9%/32.2% in COCO 2015 test-dev dataset, which outperforms R-FCN (51.5%/29.2%), and our multi-scale trained model yields 56.6%/34%, which surpasses R-FCN (51.9%/29.9%) by a large margin.

To compare our model with the other detectors listed in the COCO leaderboard. We reproduce the R-FCN baseline, following the training strategy provide by FPN (Lin et al. 2016). Table 10 shows our reproduce baselines. Since FPN

method	coco2014 minival				
	$AP_{@.5}$	AP	AP_s	AP_m	AP_l
R-FCN	56.9	34.4	17.8	38.7	47.3
R-FCN++	58.2	36.2	19.1	41.1	49.4
R-FCN++ ms train	59.8	37.5	21.0	42.7	50.8

Table 10: Reproduced baselines which used COCO2014 trainval dataset to train, and COCO2014 minival5K to test.

method	test-dev				
	$AP_{@.5}$	AP	AP_s	AP_m	AP_l
Faster R-CNN	55.7	34.9	15.6	38.7	50.9
R-FCN	53.2	31.5	14.3	35.5	44.2
FPN	59.1	36.2	18.2	39.0	48.2
R-FCN++	63.8	**42.3**	25.2	46.1	54.2

Table 11: Comparisons of single-model results on COCO 2015 detection benchmark

use 800 size for single-scale training, we use the scales {600, 700, 800, 900, 1000} for multi-scale training.

we involve left-right flipping, multi-scale testing and box voting. Besides we use Soft-NMS (Bodla et al. 2017) to filter the boxes. The results are shown in table 11. Our single model yields 42.3% in COCO 2015 test-dev dataset, surpassing all of competitors.

Conclusion

We introduce an improved R-FCN detection method called "R-FCN++". Our method takes advantage of both Faster R-CNN and R-FCN. First we present Global Context Module to improve score maps for classification, which is motivated by effectiveness of fully connected layer used to do classification in Faster R-CNN. Then we introduce a new pooling method RCMPooling to better extract scores in score maps. Our model shows significant improvements over state-of-art detectors on both Pascal VOC and COCO benchmark datasets.

References

Arbeláez, P.; Pont-Tuset, J.; Barron, J. T.; Marques, F.; and Malik, J. 2014. Multiscale combinatorial grouping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 328–335.

Bodla, N.; Singh, B.; Chellappa, R.; and Davis, L. S. 2017. Improving object detection with one line of code. *CoRR* abs/1704.04503.

Chen, L.-C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; and Yuille, A. L. 2014. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*.

Everingham, M.; Van Gool, L.; Williams, C. K.; Winn, J.; and Zisserman, A. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88(2):303–338.

Everingham, M.; Eslami, S. A.; Van Gool, L.; Williams, C. K.; Winn, J.; and Zisserman, A. 2015. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision* 111(1):98–136.

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 580–587.

Girshick, R. 2015. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, 1440–1448.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2014. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision*, 346–361. Springer.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Li, Y.; He, K.; Sun, J.; et al. 2016. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*, 379–387.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, 740–755. Springer.

Lin, T.-Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; and Belongie, S. 2016. Feature pyramid networks for object detection. *arXiv preprint arXiv:1612.03144*.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; and Berg, A. C. 2016. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, 21–37. Springer.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3431–3440.

Mallat, S. 1999. *A wavelet tour of signal processing*. Academic press.

Peng, C.; Zhang, X.; Yu, G.; Luo, G.; and Sun, J. 2017. Large kernel matters–improve semantic segmentation by global convolutional network. *arXiv preprint arXiv:1703.02719*.

Redmon, J., and Farhadi, A. 2016. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 779–788.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115(3):211–252.

Shrivastava, A.; Gupta, A.; and Girshick, R. 2016. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 761–769.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1–9.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2818–2826.

Uijlings, J. R.; Van De Sande, K. E.; Gevers, T.; and Smeulders, A. W. 2013. Selective search for object recognition. *International journal of computer vision* 104(2):154–171.

Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; and Torralba, A. 2014. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*.

Zitnick, C. L., and Dollár, P. 2014. Edge boxes: Locating object proposals from edges. In *European Conference on Computer Vision*, 391–405. Springer.