Improving Multi-stage Object Detection via Iterative Proposal Refinement

Jicheng Gong
jicheng.gong@westwell-lab.com
Zhao Zhao
zhao.zhao@westwell-lab.com
Nic Li
nic.li.edu@gmail.com

Westwell Lab
Research Group
Shanghai, China

Abstract

For object detection tasks, multi-stage detection frameworks have achieved excellent detection performance (e.g., Cascade R-CNN) compared to those one and two-stage frameworks (e.g., FPN). In this work, we introduce an LSTM-based proposal refinement module that iteratively refines proposed bounding boxes. This module can naturally be integrated with different frameworks. And the number of iterative steps is flexible and can differ between training and testing stages. In this work, we focus on improving the widely used two-stage frameworks by replacing the original bounding box regression head with our proposed module.

To verify the efficacy of our method, we perform extensive experiments on PASCAL VOC and MS COCO benchmarks with both ResNet-50 and ResNet-101 backbones. The results show that by having our LSTM based module it achieves significantly higher mAP than the vanilla R-FCN and FPN on both benchmarks. Meanwhile, it outperforms the existing state-of-the-art method Cascade R-CNN especially under high IoU thresholds.

1 Introduction

With the advancement of deep neural networks [14, 15], the performance of object detection [30, 33] has improved dramatically. Typically, an object detection task can be divided into two parts: *object localization* and *object classification*. The detectors first need to locate all the objects in a given image by assigning corresponding bounding boxes to them. Then, the classifier predicts the category to object in each bounding box. A detector fails having good object detection performance no matter it fails in either of the two tasks.

Most state-of-the-art object detection methods can be categorized into either (1) a two-stage framework [2, 6, 11, 12, 16, 18, 28, 36], or (2) a one-stage framework [8, 19, 25, 27, 37]. The two-stage methods follow the Faster-RCNN framework proposed by [28]. This approach generates a set of sparse candidate bounding box proposals and calculates an objectness score for the content of each box in the first step and then conduct further regression and classification in the second step. In contrast, one-stage methods, such as YOLO [26, 27], performs bounding box regression and classification directly by dense sampling from each

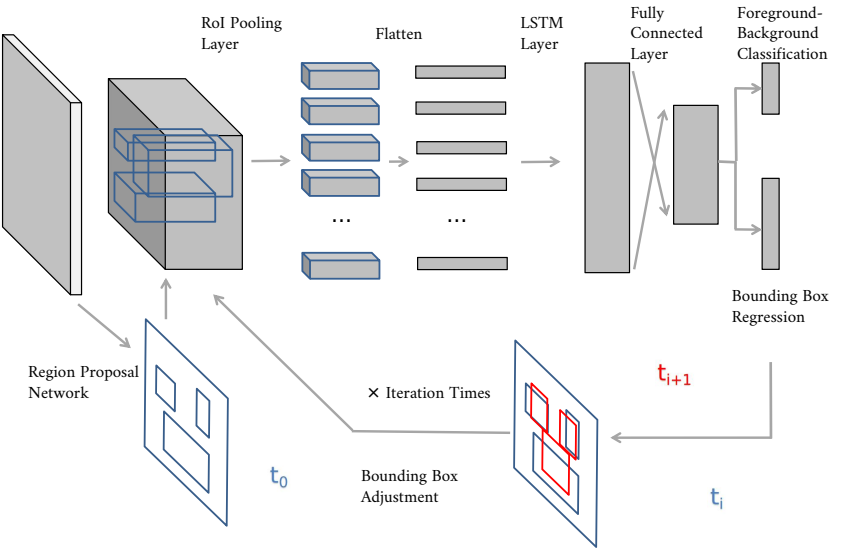


Figure 1: LSTM refinement module. Blue boxes are initial proposal boxes calculated from Region Proposal Network. 3D feature vectors are cropped from CNN feature map according to proposal boxes. Each vector is resized to equal shape by RoI pooling layer. After that we flatten 3D into 1D and simultaneously give them into LSTM layer as inputs. After decoding, we will get refined boxes which are in red. In next iteration, red boxes will then be treated as initial proposals to be refined. Such process will repeat t times until t meets the predefined iteration time.

feature map. Alternatively, SSD [25] performs regression and classification directly while it concatenates the proposals from different-level feature maps.

Typically one-stage detection models outperform two-stage models in speed but underperform them in terms of the accuracy of their detection [22]. We argue that the reasons why two-stage models yield more accurate results are primarily (1) in the first stage they provide class balanced proposals and (2) they apply more regression stages which tend to improve the localization.

Aside from one and two-stage frameworks, many works proposed multi-stage methods [2, 10, 31], which normally conduct even more bounding box regression or classification stages than most two-stage methods and have achieved improved results. Therefore, in this work, we propose a new multi-stage bounding box regression module that is flexible to regression steps. The main contributions of this paper can be summarized as follows:

- Bounding box proposals are iteratively refined by a Recurrent Neural Network [35], which is an LSTM [17] used in this work.
- The number of regression steps can be arbitrary, and it can be different at training and testing. Higher detection accuracy will be achieved with more regression steps; otherwise, detection speed can be improved by applying fewer regression steps.
- Our proposed module can be naturally applied into different two-stage detection frameworks usually by simply replacing the original regression heads.

We extensively evaluate our method on PASCAL VOC [8] and MS COCO [21] datasets

with base frameworks R-FCN [6] and FPN [22] using both ResNet-50 and ResNet-101. The experimental results demonstrate that our model surpasses R-FCN and FPN by a large margin and also outperforms existing state-of-the-art method Cascade R-CNN.

2 Related Works

One-Stage & Two-Stage Object Detection. In recent years, Faster-RCNN [28] has made great progress in improving both the speed and accuracy of object detection. Faster-RCNN introduced a framework that combines region proposal detector and region-wise classifier into a single network. Based on Faster-RCNN many other methods have been proposed to improve detection accuracy or computational speed. These include R-FCN, which proposed a position-sensitive convolution to replace the fully connected layer to improve detection efficiency, and FPN [22], which concatenated detection proposals from multiple feature map layers to further improve accuracy. On the other hand, one-stage methods such as SSD [25] can be viewed as a region proposal network that directly predicts classification and perform bounding box regression at different feature map levels. RetinaNet [23] used a focal loss to balance the foreground and background class proportion and achieved better results.

Multi-Stage Object Detection. Because normally, two-stage frameworks achieve higher detection accuracy than one-stage frameworks, many works have proposed multi-stage object detection frameworks to achieve even higher accuracy. In [20], the authors recursively sent output proposals back to the regression head, and this approach achieved relatively better mAP. [37] added an anchor refinement module to the original SSD framework for including one more bounding box refinement step than normal one-stage methods. AttractioNet [10] proposed an object location refinement module that iteratively optimizes proposal locations. Cascade R-CNN [2] repeatedly cascaded the regression head of some two-stage detection frameworks and achieved state-of-the-art results. An iterative refinement method proposed in [8] that first determined a search region by uniting overlapped regions, and then they applied a divide-and-conquer search within the search region. In addition, for multi-stage bounding box refinement, R-FCN-3000 [31] proposed a decoupled multi-stage classification framework in which each object is conducted two-step classification (super-class classification and sub-class classification), and the results showed that the detection accuracy was also improved.

Object detection with RNN. The convolutional neural network has been widely used in most object detection frameworks. This network can either be employed for feature extraction [14] or for regression and classification [6, 7]. On the other hand, recurrent neural networks have achieved great success in the neuro-linguistic programming [13, 32] field. And, many works have shown that RNNs are also applicable to object detection tasks. CTPN [34] used an LSTM [17] to encode sequential context information and showed it could reduce false detection and recover missed text proposals. The authors of [24] proposed an efficient video object detection framework which integrated a Convolutional LSTM with the SSD framework. Then, the LSTM module could encode the temporal context at each time step to refine the inputs. In [3], a CNN representation of the image crop was used as input; and this model was able to predict object shapes by a Convolutional LSTM layer.

3 LSTM-based Proposal Refinement Network

In this section, we introduce the structure of our LSTM based proposal refinement network. Figure 1 shows a high-level overview of the whole network. First, a set of candidate boxes that have high probabilities of containing an object are generated by a region proposal network. Each candidate box is then used to crop feature vector from the CNN feature map. These vectors are fed into the LSTM layer at each iteration step. Meanwhile, the hidden state of the LSTM layer is updated.

The output from the LSTM layer is fully-connected to two branches: one predicts the offsets of previous candidate bounding boxes, and the other predicts the objectness score. After decoding, we obtain refined proposal boxes that are considered to be new candidate boxes for the next iteration step. The overall flow is repeated multiple times until a predefined total iteration time is reached. Then, we conduct non-max suppression to eliminate highly overlapped bounding boxes. Finally, the remaining boxes are assigned classification scores by the classifier. The proposed module and the classifier can be trained jointly, and they can share all convolution layers.

3.1 Candidate Boxes Selection

Iterative bounding box regression is relatively high-computation. Hence, we only refine the bounding boxes with high objectness scores to speed up the refinement process. Meanwhile, we retain boxes with all different objects within an image to keep location universality.

Instead of manually selecting seed boxes as [10], we use region proposal network (RPN) [28] to select the promising bounding box candidates for further refinement. Each candidate is supplied an objectness score S_O (the higher S_O indicates that an object is more likely to be contained in the box), and four coordinates $\{x_1, y_1, x_2, y_2\}$. We use non-max suppression to eliminate highly overlapped boxes. After that, we choose boxes with the top-K S_O as candidate boxes for further refinement.

3.2 Iterative Bounding Box Regression

Iterative regression tasks can be viewed as a repetitious re-sampling procedure that aims to find the optimal hypothetical distribution. The Cascade R-CNN model [2] cascaded additional two detection heads with the base detection framework. Also, it increased the IoU threshold at each regression step during training, enabling proposals to be improved iteratively at each stage. However, this method requires the fixed regression steps at both training and testing time, which may lead to over-fitting. Also, it fails to back-propagate the whole proposal optimization in an end-to-end way, and thus its proposal optimization is conducted via multiple separate detection heads rather than a single module. AttractioNet [10] used a single CNN regressor to optimize proposals iteratively under the same IoU threshold at all training steps. However, since the bounding box distributions change significantly at each regression step [2], this single regressor may result in sub-optimal results as it is trained with a fixed IoU threshold.

Instead, we think the entire proposal optimization can be regarded as a sequential procedure, in which proposals from the previous iteration will impact on optimization in the next iteration. Also, the proposal optimizer needs to be adaptive to each iteration, and it should be able to back-propagate to optimize through this sequential procedure. Hence, we use Recur-

Algorithm 1: Iterative Bounding Box Regression

Input : Feature map F, Candidate Boxes B_0
Output : Refined Boxes B_T
List of Bounding Box Offsets $O = \{O_1, O_2, ..., O_T\}$,
List of Objectness Scores $S = \{S_1, S_2, ..., S_T\}$
h_0 = zeros states
Offsets = $\emptyset$, Scores = $\emptyset$
for $t \leftarrow 1$ **to** T **do**
  V_t = **RoIP**(F, B_{t-1}) // RoI-pooling operation
  O_t, S_t, h_t = $\Phi(V_t, h_{t-1})$
  B_t = **DecB**(O_t, B_{t-1}) // Decoding bounding box operation
  $O = O \cup O_t$
  $S = S \cup S_t$
end
B_T = **NMS**(B_T, S_T) // Non-max suppression operation
Return : B_T, O, S

rent Neural Networks, more specifically LSTM in this work, to implement our refinement module.

For each given bounding box B, we use the RoI-Pooling layer to crop a fixed-shape 3D feature from a given feature map. Each feature has $Height(H) \times Weight(W) \times Channel(C)$ shape. The 3D feature is then reshaped to 1D vector before fed into the LSTM layer. The LSTM layer contains M (=128) hidden units, and a fully-connected layer followed by the output. The key modules of LSTM are shown in Equation 1. Here x_t are cropped feature vectors, while h_t are hidden states. U represents the input-to-state parameters while W represents the hidden-to-state parameters, and i, f, o and C_t denote input, forget, output gates and cell state respectively. $\odot$ represents the element-wise product. Then, two FC layers connect the outputs from previous LSTM layer with two prediction heads, including one for predicting bounding box offset $\Delta = \{\delta_x, \delta_y, \delta_w, \delta_h\}$ and the other for predicting objectness score S.

$$
\begin{aligned}
i_t &= \text{sigmoid}(x_t U^i + h_{t-1} W^i) \\
f_t &= \text{sigmoid}(x_t U^f + h_{t-1} W^f) \\
o_t &= \text{sigmoid}(x_t U^o + h_{t-1} W^o) \\
C_t &= \text{sigmoid}(f_t \odot C_{t-1} + i_t \odot \tanh(x_t U^g + h_{t-1} W^g)) \\
h_t &= \tanh(C_t) \odot o_t
\end{aligned}
\tag{1}
$$

During each iteration, we update the LSTM hidden states and proposals. Each refined proposal is used to crop new feature map for the refinement in the next iteration. The whole process continues until the iteration time t reaches the predefined threshold T. Intermediate results such as bounding box offsets and objectness scores are stored for subsequent loss calculation. And, before forwarding the refined bounding boxes to the classifier, non-max suppression is applied to eliminate highly overlapped proposals.

The overall refinement procedure is shown in Algorithm 1, where F represents convolutional feature map and B_0 represents the candidate boxes from RPN. We initialize LSTM hidden state from a zero state, and it is updated at each iteration. The symbol Φ represents the LSTM layer. We skip a description of the fully-connected layer in algorithm description.

3.3 Classification

Besides the detection part, a classification module needs to be applied to finalize the whole object detection procedure. In this paper, we adopt Position-Sensitive score maps with Position-Sensitive RoI pooling for conducting classification when using R-FCN as our backbone; and we use a fully-connected layer as the final classifier when using FPN as our backbone. To achieve better classification results and conduct fair comparisons to vanilla R-FCN and FPN results, we employ online hard example mining (OHEM) [29] when training the classification head. We select the proposal boxes with the top-K classification loss, and then we only perform back-propagation on those selected proposal boxes.

3.4 Loss Function

The loss function for iterative bounding box regression and classification is defined as follows,

$$L_I(\boldsymbol{x},\boldsymbol{g}) = \sum_{t=1}^{T}\Big(\frac{1}{N_p}\sum_{i=1} L_{cls}(h(x_i^t, b_i^t), y_i^t) + \frac{\lambda}{N_p}[y_i^t \geq 1]\sum_{i=1} L_{loc}(f(x_i^t, b_i^t), g)\Big) \tag{2}$$

where T represents the total iteration times while t represents t-th iteration. $\boldsymbol{x}$ is the set of feature vectors, while x_i^t is the feature map cropped by bounding box b_i^t at t-th iteration. $\boldsymbol{g}$ is the set of ground truth boxes. N_p is the number of proposals. h is the objectness classifier and f is the regressor.

Apart from the loss of iterative bounding box, we also jointly train location and classification loss from RPN and the classification loss from Position-sensitive classifier [6].

4 Experiments

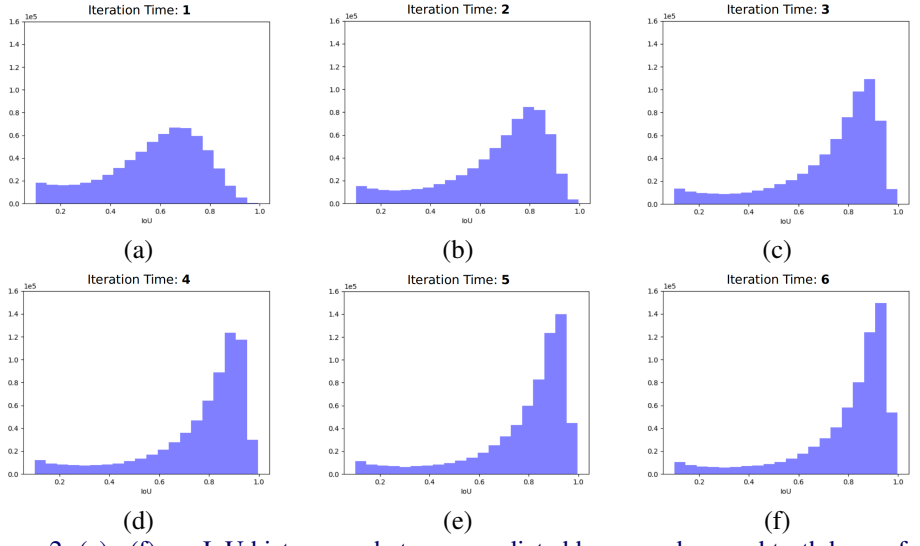
4.1 Setting

Dataset We conduct the experiments on Pascal VOC and MS-COCO 2017 datasets. For Pascal VOC, we train our models on a combined set consisting of the VOC2007 trainval and the VOC2012 trainval sets, which contain ~16K images and evaluate the trained models on VOC2007 test set which contains 4,952 images. Analogously, we train the models on MS-COCO 2017 ~118K training images and evaluate the models on the 5k validation image set.

Evaluation Metric We use the evaluation metric, the same as [21], average precision (AP) evaluated @IoU $\in$ [0.5, 0.95] with an interval of 0.05 for evaluating the models trained on both datasets.

4.2 Implementation Details

We use both ResNet-50 and ResNet-101 as the backbone networks for our experiments. We pre-process the images by resizing them isotropically to have a shorter side of 600 pixels on Pascal VOC dataset and have a shorter side of 800 pixels on COCO dataset. At training, the only data augmentation technique used is left-right flipping, without further bells and whistles.



(a) (b) (c) (d) (e) (f)

Figure 2: (a) ~(f) are IoU histograms between predicted boxes and ground truth boxes from 1st iteration to 6th iteration. IoU distributions are approaching closer to 0.9 which implies higher bounding box regression quality.

Anchors We use five scales with areas of $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ and three aspect ratios $\{1:2, 1:1, 2:1\}$. Anchors that exceeded the image size, will be clipped.

NMS Threshold The IoU threshold for RPN non-max suppression is set to 0.7; then, we choose the top 800 proposal boxes to perform further refinement. In some extreme cases, where the number of proposal boxes is less than 800 after non-max suppression, we do zero-padding to satisfy the batch size requirement. The IoU threshold for LSTM non-max suppression is set as 0.8 to eliminate highly overlapped bounding boxes only.

Foreground Threshold Figure 2 shows that the mean IoUs of predicted boxes increase during each iteration. Therefore, we increase the foreground threshold value u, which indicates a predicted box as a positive sample if its IoU with the ground truth box is great than or equal to u. We choose $u = \{0.5, 0.6, 0.7\}$ in our experiments because the maximum iteration time T is set to 3 at training.

Configurations We use a M-SGD optimizer with momentum value 0.9. The initial learning rate is set to 0.001, and is switched to 0.0001 at 120K iteration on Pascal VOC and at 520K iterations on MS COCO. We select top 800 proposals for iterative refinement and sample them into 1:1 positive-negative ratio.

Code Implementation Our algorithm is implemented by TensorFlow [1, 4]. We failed to find a TensorFlow version R-FCN whose performance matches the results in R-FCN paper [5]. Therefore, we start from an R-FCN baseline with lower accuracy. Nevertheless, by incorporating our module, the final method achieves comparable or better performance than other competitors, which demonstrates the strong efficacy of our proposed module.

Iteration stage	1	2	3	4	5	6	7	8
Test speed	0.081s	0.097s	0.117s	0.136s	0.157s	0.176s	0.198s	0.216s
AP@0.5-0.95	48.4	52.9	**53.6**	53.5	53.5	53.2	53.3	53.2
AP@0.5	74.8	76.8	**77.0**	77.0	76.9	76.7	76.6	76.7
AP@0.75	52.7	57.4	57.6	**57.7**	57.4	57.5	57.6	57.5

Table 1: AP results under different numbers of testing iterations. Best result in **bold**.

Method	Backbone	AP@0.5-0.95	AP@0.5	AP@0.75
R-FCN [5]	ResNet-50	44.8	77.5	46.8
Cascade R-CNN [2]	ResNet-50	51.8	**78.5**	57.1
LSTM+R-FCN	ResNet-50	**53.6**	77.0	**57.6**
R-FCN [5]	ResNet-101	49.4	**79.8**	53.2
Cascade R-CNN [2]	ResNet-101	54.2	79.6	59.2
LSTM+R-FCN	ResNet-101	**56.0**	78.5	**61.4**

Table 2: Object detection performance on PASCAL VOC benchmark of different methods. Best result in **bold**.

4.3 Experiments

4.3.1 Iteration Stages

We plot histograms of the IoUs between the predicted boxes and the ground truth boxes at different iteration stages in Figure 2. The model is trained on VOC2007+2012 trainval set with ResNet-50 and tested on VOC2007 test set. All bounding box proposals before conducting non-max suppression are used to calculate the IoUs. From these plots, we can see that bounding boxes are progressively refined towards higher IoUs at each stage, which means the proposals become closer to the ground truth. The performance improves rapidly at the first and second iterations and saturates at the fourth iteration.

Table 1 shows the APs@0.5-0.95, APs@0.5, APs@0.75 and test speeds with different iteration numbers. From Table 1, we can see that the results are consistent with the observation in Figure 2, i.e., the AP begins to saturate during the third and fourth iterations and then decreases marginally. We also observed the same phenomenon when using ResNet-101. Thus, all the subsequent results reported here were tested using three iterations. Meanwhile, we can see that the time cost of our model during testing time is proportional to the number of refinement iterations at testing. It takes 117ms for our model to detect one image by three iterations. Also, higher detection speed can be achieved with fewer iterations.

4.3.2 Evaluation on Pascal VOC

In this section, we first evaluate our model on Pascal VOC using the introduced metrics. We compare our method with the baseline R-FCN network and the state-of-the-art multi-stage method Cascade R-CNN using both ResNet-50 and ResNet-101 [14] backbones.

From the results in Table 2, we can see that our method outperforms the baseline R-FCN by a large margin under AP@0.5-0.95 and AP@0.75 with both backbones. Meanwhile, under AP@0.5-0.95, we method has better mAP (1.8% improvement) compared to Cascade R-CNN using both backbones. Besides, our method achieves 2.2% higher mAP than Cas-

Method	Backbone	AP	AP@0.5	AP@0.75	AP@S	AP@M	AP@L
R-FCN [6]	ResNet-50	27.0	48.7	26.9	9.8	30.9	40.4
Cascade R-CNN [2]	ResNet-50	31.1	**49.8**	32.8	10.4	34.4	**48.5**
LSTM+R-FCN	ResNet-50	**32.0**	48.3	**34.6**	**12.3**	**36.4**	47.1
R-FCN [6]	ResNet-101	30.3	**52.2**	30.8	12.0	34.7	44.3
Cascade R-CNN	ResNet-101	33.3	52.0	35.2	11.8	37.2	**51.1**
LSTM+R-FCN	ResNet-101	**34.1**	50.5	**37.0**	**13.7**	**38.9**	51.0
FPN [22]	ResNet-50	36.5	58.6	39.2	20.8	40.0	47.8
Cascade R-CNN*	ResNet-50	40.3	59.4	43.7	22.9	43.7	**54.1**
LSTM+FPN	ResNet-50	**41.3**	**59.8**	**44.4**	**23.9**	**44.3**	53.9
FPN [22]	ResNet-101	38.5	60.6	41.7	22.1	41.9	51.1
Cascade R-CNN*	ResNet-101	42.7	61.6	46.6	23.8	46.2	57.4
LSTM+FPN	ResNet-101	**43.6**	**62.3**	**47.1**	**25.4**	**47.4**	**57.8**

Table 3: Object detection performance on COCO benchmark of different methods. Cascade R-CNN denoted by "*" uses FPN as base architecture while Cascade R-CNN without "*" uses R-FCN as base architecture. Best result in **bold**.

cade R-CNN when evaluated under AP@0.75 using ResNet-101. Overall, this shows that our model has better detection accuracy and improved detection quality than those competitors.

4.3.3 Evaluation on MS COCO

We also evaluate our proposed method on MS COCO 2017 dataset and incorporate our LSTM module into both R-FCN [6] and FPN [22]. Moreover, we compare our method to Cascade R-CNN using R-FCN [6] and FPN [22] respectively for fair comparisons. The top part of Table 3 shows the results when using R-FCN as the base framework. We can see that our model outperforms R-FCN with 4.7% and 3.5% AP when using ResNet-50 and ResNet-101, respectively. Compared to Cascade R-CNN, our method achieves better performance under AP@0.5-0.95 and AP@0.75, which is consistent to results on Pascal VOC.

The bottom part of Table 3 shows the results when using FPN as the base framework. Compared to the results using R-FCN, the performance of all methods is improved under all metrics. Using this more advanced base framework, our method works more effectively, consequently outperforming FPN and Cascade R-CNN at almost all different metrics. Furthermore, it is interesting to see our model is good at detecting small- and medium-size objects, i.e., harder tasks, with higher AP than other competitors in all cases.

4.3.4 Further Analysis

Visualization of mAP under different IoUs To further demonstrate the detection quality improved by our method, we plot the mAPs of each model under each IoU thresholds in Figure 3. All models were trained and tested on Pascal VOC using ResNet-101. From this plot, we can find that all models have almost the same mAP at 0.5, 0.55 and 0.6. However, the mAP of vanilla R-FCN begins to drop quickly when IoU is greater than 0.6. Our method begins to outperform Cascade R-CNN when the IoU exceeds 0.7, and the improvement margin becomes larger under higher IoUs. Especially, our method gets 2.5-time better mAP than Cascade R-CNN when the IoU equals to 0.95. More qualitative detection results are shown

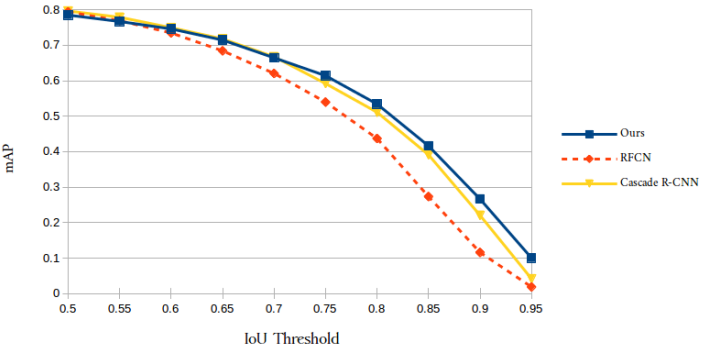


Figure 3: Our model surpasses vanilla R-FCN when IoU is larger than 0.6. And our model continuously outperforms Cascade R-CNN when IoU is higher than 0.75.

Method	Backbone	AP@0.5-0.95	AP@0.5	AP@0.75
MLP+RFCN	ResNet-50	47.0	74.2	50.4
LSTM+RFCN	ResNet-50	53.6	77.0	57.6

Table 4: Comparison between LSTM layer and MLP layer

in **Supplementary**.

Ablation Study To demonstrate the merit of our LSTM based regression module, we conduct the experiments on Pascal VOC replacing the LSTM module with an MLP module, as shown in Table 4. The MLP module has the equivalent intermediate dimension as our LSTM module. Its first layer encodes feature maps with shape $H \times W \times C$ into a 128D vector, and then its output layer conducts the bounding box regression and predicts the foreground-background classification scores. The results in Table 4 show that the LSTM module surpasses MLP module with noticeable margins, especially LSTM based R-FCN gains 5.2% mAP improvement compared with MLP based R-FCN under AP@0.5-0.95. This shows the advantage of our LSTM module compared with the standard MLP module. Our LSTM module contains regression information from previous steps via cell memory and hidden states, which may be more suited to iterative bounding box refinement than an MLP module.

5 Conclusion

In this paper, we proposed a new iterative proposal refinement module in which proposals are iteratively improved through an LSTM module until convergence. This module can be applied to R-FCN and FPN without excessive tuning or structure modifying and can naturally replace the regression head of most two-stage frameworks. The experimental results demonstrated that by incorporating our LSTM-based refinement module, it achieved better detection performance on both Pascal VOC and MS COCO benchmarks than vanilla R-FCN and FPN and yielded a higher detection quality than the state-of-the-art method Cascade R-CNN.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[2] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018.

[3] Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Annotating object instances with a polygon-rnn. In *CVPR*, 2017.

[4] Xinlei Chen and Abhinav Gupta. An implementation of faster rcnn with study for region sampling. *arXiv*, 2017.

[5] Kai-Wen Cheng, Yie-Tarng Chen, and Wen-Hsien Fang. Improved object detection with iterative localization refinement in convolutional neural networks. *TCSVT*, 2018.

[6] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

[7] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017.

[8] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010.

[9] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv*, 2017.

[10] Spyros Gidaris and Nikos Komodakis. Attend refine repeat: Active box proposal generation via in-out localization. In *BMVC*, 2016.

[11] Ross Girshick. Fast r-cnn. In *ICCV*, 2015.

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[13] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*. Springer, 2016.

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[18] Borui Jiang, Ruixuan Luo, Jiayuan Mao, Tete Xiao, and Yuning Jiang. Acquisition of localization confidence for accurate object detection. In *ECCV*, 2018.

[19] Tao Kong, Fuchun Sun, Anbang Yao, Huaping Liu, Ming Lu, and Yurong Chen. Ron: Reverse connection with objectness prior networks for object detection. In *CVPR*, 2017.

[20] Jianan Li, Xiaodan Liang, Jianshu Li, Yunchao Wei, Tingfa Xu, Jiashi Feng, and Shuicheng Yan. Multistage object detection with group recursive learning. *Transactions on Multimedia*, 2018.

[21] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*. Springer, 2014.

[22] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[23] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[24] Mason Liu and Menglong Zhu. Mobile video object detection with temporally-aware feature maps. In *CVPR*, 2018.

[25] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*. Springer, 2016.

[26] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *CVPR*, 2017.

[27] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[28] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[29] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, 2016.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014.

[31] Bharat Singh, Hengduo Li, Abhishek Sharma, and Larry S Davis. R-fcn-3000 at 30fps: Decoupling detection and classification. In *CVPR*, 2018.

[32] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *ISCA*, 2012.

[33] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *NIPS*, 2013.

[34] Zhi Tian, Weilin Huang, Tong He, Pan He, and Yu Qiao. Detecting text in natural image with connectionist text proposal network. In *ECCV*. Springer, 2016.

[35] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1989.

[36] Sergey Zagoruyko, Adam Lerer, Tsung-Yi Lin, Pedro O Pinheiro, Sam Gross, Soumith Chintala, and Piotr Dollár. A multipath network for object detection. *arXiv*, 2016.

[37] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. In *CVPR*, 2018.