Faster R-CNN with Region Proposal Refinement

Peng Yuan
Electrical Engineering Department
Stanford University
pengy@stanford.edu

Yangxin Zhong
Computer Science Department
Stanford University
yangxin@stanford.edu

Yang Yuan
Computer Science Department
Stanford University
yyuan16@stanford.edu

Abstract

State-of-the-art object detection methods are in two major groups: region proposal based methods like Faster R-CNN [11], and regression based methods like YOLO [10]. In this work, we focus on improving the region proposal part of Faster R-CNN by introducing iterative region proposal refinement and LSTM region proposal refinement. An iterative region proposal refinement can iteratively refine region proposal based on previous output, and an LSTM region proposal refinement is of similar structure but added an LSTM layer to carry hidden information between different iterations. We trained and tested our model on PASCAL VOC 2007 dataset, improved the mean average precision (mAP) from 0.6702 to 0.6774. Experiments also show that our method can iteratively improve the detection results along multiple refinements.

1. Introduction

Object detection is a computer vision task that aims to detect instances of semantic objects of certain classes in digital images (and videos). Given an image, object detection system detects what objects are in it and where they locate. It plays an important role in face detection, self-driving cars, video surveillance and many other applications.

Recent years, deep Convolutional Neural Network (CNN) bring significant improvement to object detection task over traditional methods [6, 12]. CNN based object detection algorithms can be divided into two major categories: region proposal based algorithm like R-CNN [4], Fast R-CNN [3], and Faster R-CNN [11], and regression based algorithm like YOLO [10] and SSD [7]. In this project, we investigate the region proposal based detection algorithm and try to refine region proposals for multiple iterations. More specifically, our goal is to improve the performance of Faster R-CNN [11].

Inspired by RefineNet [9] and LSTM [5], we propose methods to get better region proposals through multiple-iteration refinement, and tested our method on PASCAL VOC 2007 dataset [2].

2. Related Work

2.1. Region Proposal Based Algorithms

The most representative region proposal based mode is Faster R-CNN [11], which originates from R-CNN [4], and fast R-CNN [3].

R-CNN. Region-based Convolutional Neural Network, or R-CNN, was proposed not long after the emergence of CNN and it greatly improved the detection performance in terms of mean average precision (mAP) compared to models without deep CNNs [4]. The system consists of three parts. The first part generates region proposals using selective search [14]. The second part is feature extraction through CNN, in each proposed region. The third part is classification by SVMs. Although R-CNN is an innovative and effective model, it has many defects: ad hoc training objectives, expensive training both in space and time, and worst, long detection time [3, 11].

Fast R-CNN. Fast R-CNN builds on the work of R-CNN and improves training and testing speed while increasing the detection accuracy [3]. The speed-up comes from sharing CNN computation among all region proposals. In Fast R-CNN, an image is put into a CNN to create a convolution feature map first and then an Region of Interest (RoI) pooling layer extracts a feature vector for each region proposal [3]. The feature vectors are fed into fully-connected layers and finally the model produces softmax class probability estimates and bounding boxes for each detected ob-

jects. Fast R-CNN significantly reduces training and testing time, but the region proposals are still made by traditional methods [3], which costs a long time for pre-processing.

Faster R-CNN. To solve the bottleneck problem of region proposals in Fast R-CNN, Faster R-CNN is proposed, which makes region proposals by neural network instead [11]. It is composed of two modules. The first module is a Region Proposal Network(RPN), which proposes regions for the second module, Fast R-CNN detector, to inspect. In RPN module, a small network slides over the convolution feature map with multiple anchors at each sliding-window location. The RPN outputs a bounding boxes (region proposals) and predicted class as Fast R-CNN module does. A four-step alternating training is adopted to share features for both modules [11].

2.2. Regression Based Algorithms

YOLO and SSD. You Only Look Once (YOLO) reframe the detection problem as a regression problem, so as Single Shot MultiBox Detector (SSD), and they do not have region proposals involved [10, 7]. Instead, they are based on pre-defined grid and use a CNN to simultaneously predicts bounding boxes and class confidence, making it extremely fast compared to methods based on region proposals [10, 7].

2.3. Iterative Refinement

As reported in RefineNet [9], iterative refinement on the proposed regions can significantly improve the mAP of the detection result. RefineNet is based on ZF Net [16] and achieved close performance to Faster R-CNN, while running ten times faster. Inspired by the paper, we propose iterative refinement based on Fast R-CNN and adding LSTM [5] to it, achieved further improvement on its performance in object detecting task.

3. Method

3.1. Faster R-CNN and Motivation

One of the state-of-the-art object detection models is Faster R-CNN [11]. The architecture of Faster R-CNN is shown in Figure 1. Given an image, we first employ a pre-trained deep convolutional neural network, such as VGG [13], to extract feature maps from it. Then, they use a Region Proposal Network (RPN), which consists of two convolutional layers, to detect the regions that might contain object in the feature maps (image). Then the network employ a RoI pooling layer [3] to crop and resize the the feature maps according to these region proposals. The new feature maps of each region are then used for classification and finer bounding box regression through three fully connected layers.

One issue of this Faster R-CNN architecture is that the region proposals output from RPN are not very accurate.

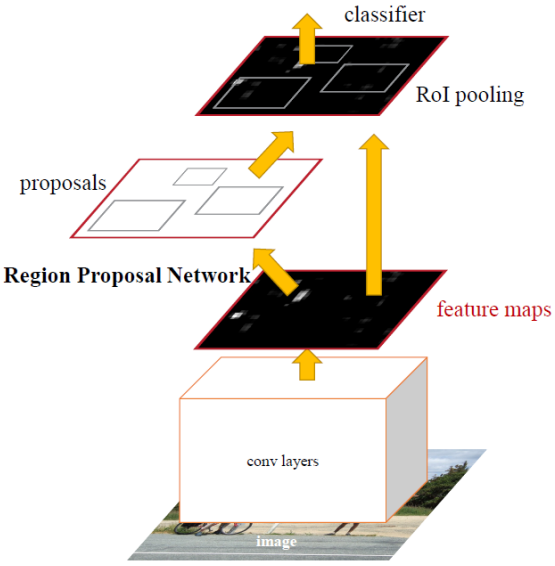

Figure 1. Architecture of Faster R-CNN. (This figure is from the original paper [11].)

This is due to RPN using some hard coded anchors with fixed scales and aspects to guess the potential regions. Although RPN also has a bounding box regression part, it only aims to give general boxes that may contain any kind of objects. Without class specific knowledge, these boxes won't be very accurate. We argue that the error of classification and final bounding box regression might be partly caused by error of proposed region. Suppose we can have finer region proposals, the accuracy of classification and final bounding box regression may be further improved.

3.2. Iterative Region Proposal Refinement Model

A nature better region proposal will be the regressed bounding box since it is designed to refine the rough region output by RPN using class specific knowledge encoded in the network. Since regressed bounding box is one of the outputs of the whole Faster R-CNN network, we cannot use it as our region proposal at the beginning. However, once we get the finer bounding box, we can start another round of classification and bounding box regression, using the regressed bounding box in previous round as the region proposals in the new round. This process can keep going for several iterations. In this iterative way, we can refine the region proposals again and again and might get a better result after each iteration.

The model of Faster R-CNN with iterative region proposal refinement is shown in Figure 2. In the first iteration, it's exactly the same as Faster R-CNN: extract feature maps from image by VGG, pass them to RPN to get region proposals, employ RoI pooling layer to crop and resize new feature maps for each proposal, and use a three-layer fully

connected network to get the final class scores and bounding box regression for each class.

After the first iteration, for each proposed region, we select the regressed bounding box with the maximum class score as the region proposal in the second iteration. And the rest of second iteration is the same as the first iteration: we use RoI pooling layer to crop and resize feature maps for each proposal, classify and regress the bounding box using new feature maps. Note that for the input of RoI pooling layer, we reuse the feature maps in first iteration and there is no need for recalculation. Also, we reuse parameters of the three-layer fully connected network in different iterations. After the second iteration, we can repeat the same process for the third iteration, and so on. Figure 2 is a demonstration of iteration number = 3.

Our iterative refinement model can be represented by the following equations. For the feature maps extraction and each initial RPN region proposal, we have

$$f = VGG(image)$$

$$RP_1 = RPN(f)$$

Then suppose we set iteration number = T, then for each iteration i, the model can be represented as

$$\text{for } i = 1, 2, ...T,$$
$$\begin{cases} r_i = RoIPooling(f, RP_i) \\ scores_i,\ boxes_i = FC^3(r_i) \\ RP_{i+1} = boxes_{i, \arg\max_{j \in \{0...C\}} scores_{ij}} \\ loss_i = loss_{cross-entropy}(scores_i, c_i) \\ \qquad + \lambda[c_i \neq bg] loss_{smooth_{L_1}}(boxes_i, b_i) \end{cases}$$

where FC^3 denotes the three-layer fully connected network, C denotes the number of classes, c_i denotes the class label for the proposed region RP_i, b_i denotes the ground truth bounding box of object in this region, $[c_i \neq bg] = 1$ if the region is not a background and contains object, otherwise $[c_i \neq bg] = 0$, and λ is the weight parameter to balance classification loss and bounding box regression loss.

During training, for the classification, we use softmax-cross entropy loss, and for the bounding box regression, we use smooth L_1 norm loss [3]. We will get a $loss_i$ in each iteration. The final loss will be the sum of all iterations (loss from RPN might also be added to the final loss). And in test time, we use the scores and bounding boxes from the last iteration as the final output:

$$\begin{cases} loss_{final} = \sum_{i=1}^{T} loss_i \\ output = scores_T,\ boxes_T \end{cases}$$

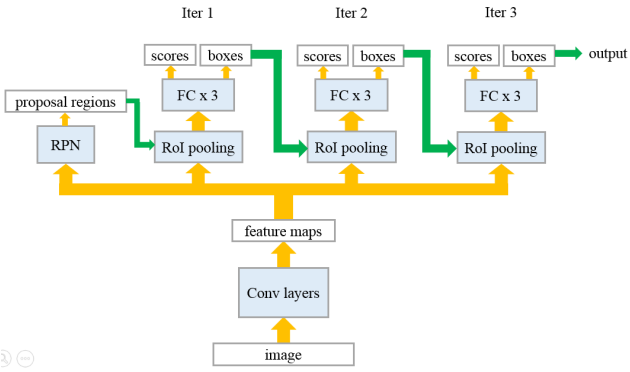

Figure 2. Architecture of Faster R-CNN with iterative region proposal refinement.

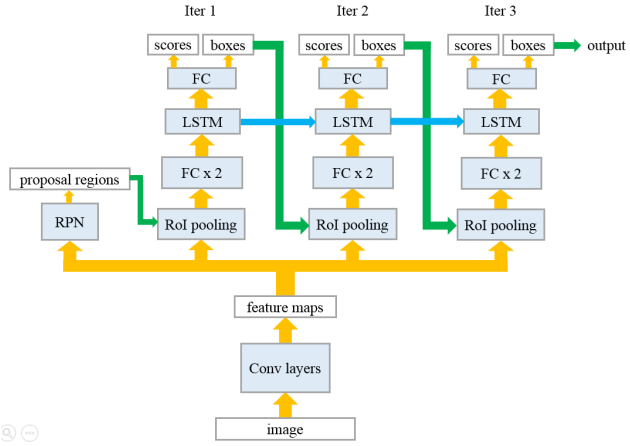

Figure 3. Architecture of Faster R-CNN with LSTM region proposal refinement.

Note that above are the equations for only one region proposal. If RPN yields K region proposals at the very beginning, we will do the same thing for each proposal, and average $loss_{final}$ of all K proposals as the final loss.

3.3. LSTM Region Proposal Refinement Model

One issue of the iterative refinement model is the gradient of loss can not be backpropagated from the later iteration to the earlier iterations. Since the error of classification and bounding box regression of later iteration might be caused by errors of earlier iterations, we hope those earlier errors can also be corrected by gradient backpropagation. However, this cannot be done by iterative refinement model since RoI layer cannot backpropagate the gradient to the region proposal coordinates. This is because RoI layer uses these coordinates to crop the feature maps, but the "crop" operation is not differentiable with respect to coordinates. As a result, in Figure 2 the gradient of each loss in iterative model can only be propagated downward (yellow arrows)

but cannot be propagated to the left (green arrows).

Inspired by Reccurrent Neural Network (RNN), which can pass useful information to later time steps and also propagate the gradient backward to the previous time steps, we propose a LSTM region proposal refinement model, where LSTM is one of RNN models [5]. The architecture of our model is shown in Figure 3. The different between iterative and LSTM refinement models is that we add a LSTM layer right before the final fully connected (output) layer, and pass the hidden states of LSTM to the next iteration (blue arrows). As a result, if the iteration number = T, then in each iteration i, the equations become

$$\text{for } i = 1, 2, ...T,$$

$$\begin{cases} r_i = RoIPooling(f, RoI_i) \\ a_i = FC^2(r_i) \\ h_i = LSTM(h_{i-1}, a_i) \\ scores_i,\ boxes_i = FC(h_i) \\ RoI_{i+1} = boxes_{i, \arg\max_{j \in \{0...C\}} scores_{ij}} \\ loss_i = loss_{cross-entropy}(scores_i, c_i) \\ \qquad + \lambda[c_i \neq bg] loss_{smooth_{L_1}}(boxes_i, b_i) \end{cases}$$

where h_i is the hidden state of LSTM at iteration i, and the input of the LSTM layer is h_{i-1}, the hidden state of previous iteration, and a_i, the output of two-layer fully connected network in current iteration. The rest of this model is the same as iterative refinement model.

One benefit of adding a LSTM layer is that the hidden state of previous iteration can contain information that is useful to improve classification and bounding box regression results in current iteration. Another benefit is that we can now backpropagate the gradient of loss from the later iterations to the earlier ones (through the blue arrows in Figure 3), since LSTM is differentiable with respect to the previous hidden state. As a result, if the error of prediction in current iteration comes from the errors of previous iterations, LSTM gives it a chance to correct those errors by gradient backpropagation.

Actually, this is an unusual use of LSTM/RNN model. RNNs are more often used in sequential input data such as text, audio and video. The hidden state of RNN was proved to have the ability to capture the temporal information of data at previous time steps. Since the meaning of current frame in sequential data is often related to the frames previous to it, RNN makes a great success in encoding features of sequential data.

Although our data is image, which is not sequential data, but the way we refine the region proposals and make progress step by step also contains temporal information. We can see the iterations as multiple guesses. In each iteration, we look into the guessed region of the image and get some information that is useful to decide a better guess next time. This way of sequential processing of non-sequential data is often called attention or glimpse model, which has been used in both multiple object detection [1] and question answering [15]. In the paper [15], they also use a LSTM to refine the answer span for a question for multiple times.

4. Experiment

We implemented our models based on a PyTorch implementation of Faster R-CNN [8] and trained our iterative refinement model and LSTM refinement model on VOC 2007 train and validation set, and tested our models on VOC 2007 test set. We use mean average precision (mAP), which is widely used for object detection task, as the metric to evaluate our model.

4.1. Test Results

The overall results is shown in Table 1. We found that most models with refinement perform better than the original Faster R-CNN model, which shows that region proposal refinement can improve the detection results. As we explore more different hyper-parameters, we found that the iterative refinement in general performs better than the LSTM refinement. The best model of iterative refinement gives 0.6774 mAP and the best of LSTM model gives 0.6743 mAP, compared to 0.6702 mAP of the original Faster R-CNN that we trained. We also notice that to train a better refinement model, the learning rate should start from a relative small value. In addition, training that includes loss from RPN gives slightly better results over those do not. As for the number of iterations of the refinement, training with 3 iterations and testing with 3 iterations achieves best performance, suggesting that the iteration number is not the larger the better.

The detailed test results of our best iterative refinement and LSTM refinement is in Table 2. The iterative refinement and LSTM refinement have close performance over many classes, and both gives better AP than the original model in most of the categories.

4.2. Visualization

To better understand our iterative refinement model and to analyze its pros and cons, we visualize some object detection results in Figure 4, where refinement with iteration 1, 2 and 3 are shown in bounding box with color orange, yellow and green respectively.

We can observe iterative improvement of the bounding box predictions in some images. More concretely: 1) in images with single object where the original region proposal only covers a part of the object: the refinement can usually enlarge the bounding box to cover the entire object, as shown in 2A, 3A, 3B and 5A in Figure 4. 2) in images with single object where the original region proposal is already very good: the refinement usually makes minor changes to

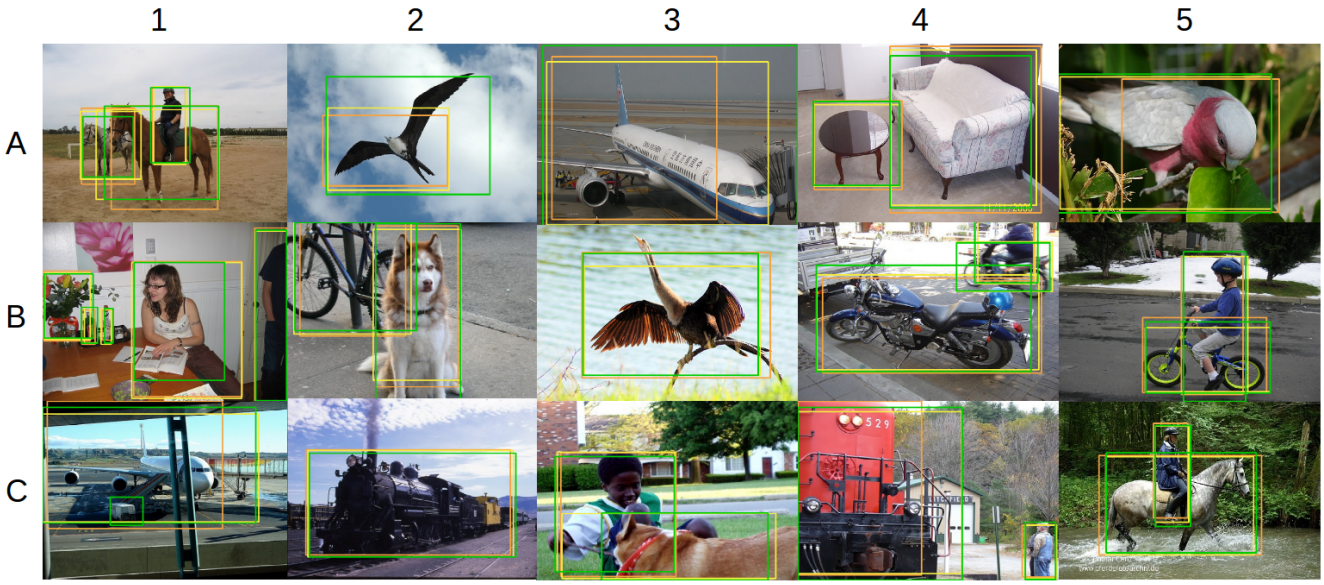

Figure 4. Visualization of iterative refinement model (Iteration 1, 2, and 3 are denoted by orange, yellow, and green)

model	**parameters**				**mAP**
	train max_iter	**test max_iter**	**includes RPN loss**	**learning rate**	
original model	-	-	-	-	0.6702
iterative refinement	2	2	Y	0.0001	0.6580
iterative refinement	2	2	Y	0.00001	0.6707
iterative refinement	3	3	N	0.00001	0.6744
iterative refinement	3	2	Y	0.00001	0.6772
iterative refinement	3	3	Y	0.00001	**0.6774**
iterative refinement	3	4	Y	0.00001	0.6760
LSTM refinement	2	2	Y	0.0001	0.6582
LSTM refinement	3	2	Y	0.00001	0.6702
LSTM refinement	3	3	Y	0.00001	0.6596
LSTM refinement	4	2	Y	0.00001	0.6712
LSTM refinement	4	3	Y	0.00001	0.6723
LSTM refinement	7	2	Y	0.00001	**0.6743**
LSTM refinement	7	3	Y	0.00001	0.6557

Table 1. Overall results. Trained with 100K step, learning rate decay to 1/10 at 50K and 80K step. Size of each fully connected layer: 4096, hidden units number of LSTM: 1024.

model	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
original model	0.6702	0.680	**0.779**	0.660	0.548	0.470	0.758	0.794	0.786	0.456	**0.724**	0.654	0.743	0.803	0.697	0.761	0.391	0.652	0.624	0.750	0.680
iterative refinement	**0.6774**	0.691	0.710	0.672	**0.581**	**0.494**	0.764	**0.797**	**0.792**	0.441	0.717	**0.688**	**0.754**	0.806	0.706	**0.762**	**0.412**	**0.667**	**0.657**	0.755	**0.682**
LSTM refinement	0.6743	**0.700**	0.773	**0.678**	0.515	0.478	**0.768**	0.789	0.775	**0.470**	0.715	0.670	0.753	**0.807**	**0.745**	0.755	0.368	0.659	0.646	**0.758**	0.665

Table 2. Test results of different models across different classes on VOC 2007

the bounding box, as shown in 2C. 3) in images with multiple object where region proposal for some objects are too large, iterative refinement can sometimes shrink it to the correct size (as in 1A, 1B and 4A) and sometimes fail to do so. 4) most importantly, iterative refinement can sometimes find objects that was not originally detected, as shown in 1C (where the truck under the plane is not detected until the third refinement iteration) and 3C (where the dog is not detected until the second iteration).

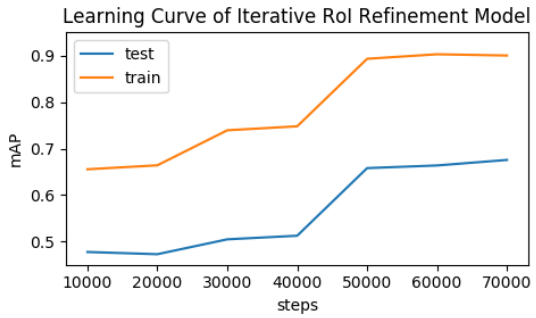

Figure 5. Learning curve of Faster R-CNN with iterative region proposal refinement (learning rate decay at 40K and 60K step).

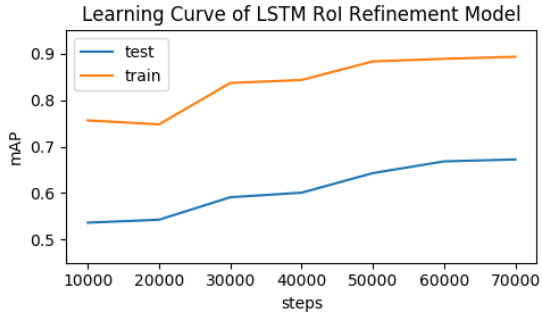

Figure 6. Learning curve of Faster R-CNN with LSTM region proposal refinement (learning rate decay at 40K and 60K step).

4.3. Discussion

RefineNet [9] applies a similar iterative refinement based on ZF Net and reported significant improvement on mAP. However, our iterative refinement and LSTM refinement did not improve the mAP as large as expected, and we analyzed the possible reasons in depth in this section.

Over-fitting issue

Figure 5 and Figure 6 are the learning curves for iterative refinement and the LSTM refinement respectively. Both learning curves show large gap between the learn and test mAP, suggesting that the model may have over-fitting issue. We tried larger drop-out rate and adding regularization, but neither could bring down the gap while not decrease the test mAP. However, we believe that our refinement method will achieve better performance once the overfitting issue is solved.

Iterative length is too short for LSTM

From Table 4.1 we see that iterative region proposal refinement works better than LSTM refinement, this may due to that such iterative length (2-7) is too short for LSTM to show its strength. Another comparison among LSTM refinement shows that training with larger number of iterations can bring up the test mAP, which further suggests that the iterative length may be too short for a LSTM.

Magnify both good and bad detections

Another reason contributes to the overall underperformance is that our refinement model takes the prediction of previous stage as input, therefore it will magnify both good and bad predictions. More concretely, if the original region proposal is reasonably good, then the refinement part can refine the detection bounding box and improve the detection accuracy. However, if the original region proposal is bad (or totally wrong in some cases), then the refinement model will further magnify the error, since the refinement part is somewhat repeating the region proposal task and a failure in the original task will lead to a failure in the subsequent tasks.

5. Conclusion

To conclude, we introduced two refinement methods, iterative and LSTM refinement, to Faster R-CNN model and improve the performance of it from 0.6702 to 0.6774 mAP in object detection task on PASCAL VOC 2007 dataset.

In the future, our methods can be further improved by 1) finding a method to make the RoI pooling layer able to backpropagate gradient through different iterations, so that we can further improve our iterative model; and 2) solving the potential over-fitting issue.

References

[1] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *CoRR*, abs/1412.7755, 2014.

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

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

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

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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

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

[8] Longcw. Faster rcnn with pytorch. `https://github.com/longcw/faster_rcnn_pytorch`, 2017.

[9] R. N. Rajaram, E. Ohn-Bar, and M. M. Trivedi. Refinenet: Iterative refinement for accurate object localization. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1528–1533, Nov 2016.

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

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

[12] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

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

[15] C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[16] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.