Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 3598316, 7 pages
https://doi.org/10.1155/2018/3598316

Research Article

Object Detection Based on Fast/Faster RCNN Employing Fully Convolutional Architectures

Yun Ren,[1] Changren Zhu,[1] and Shunping Xiao[2]

[1] *ATR National Lab, National University of Defense Technology, Changsha 410073, China*
[2] *State Key Lab of Complex Electromagnetic Environment Effects on Electronics and Information System, National University of Defense Technology, Changsha 410073, China*

Correspondence should be addressed to Yun Ren; renyun_nudt@163.com

Received 23 October 2017; Accepted 12 December 2017; Published 9 January 2018

Academic Editor: Francesco Marotti de Sciarra

Modern object detectors always include two major parts: a feature extractor and a feature classifier as same as traditional object detectors. The deeper and wider convolutional architectures are adopted as the feature extractor at present. However, many notable object detection systems such as Fast/Faster RCNN only consider simple fully connected layers as the feature classifier. In this paper, we declare that it is beneficial for the detection performance to elaboratively design deep convolutional networks (ConvNets) of various depths for feature classification, especially using the fully convolutional architectures. In addition, this paper also demonstrates how to employ the fully convolutional architectures in the Fast/Faster RCNN. Experimental results show that a classifier based on convolutional layer is more effective for object detection than that based on fully connected layer and that the better detection performance can be achieved by employing deeper ConvNets as the feature classifier.

1. Introduction

Modern object detectors [1, 2] always include two major parts: a feature extractor and a feature classifier as same as traditional object detectors. These two parts are thought to be mutually independent in the traditional object detectors while they can be considered to be a unified course in the modern object detectors. The feature extractor in traditional object detection methods is usually a hand-engineered descriptor, such as SIFT [3] and HOG [4]. At the same time, the feature classifier is usually a linear SVM [5], a nonlinear boosted classifier [6], or an additive kernel SVM [7]. However, the deep ConvNets have dominated the feature extractor of the modern object detectors in various application scenarios [8–11]. Aside from being capable of representing higher-level semantics, ConvNets are also more robust to variance in scale and thus facilitate recognition from features computed on a single input scale.

The successful RCNN [12] method applies high-capacity convolutional neural networks to extract a fixed-length feature vector from each region which is fed to a set of class-specific linear SVMs. It firstly pretrains the network by supervision for image classification with abundant data and then fine-tunes the network for detection where data is scarce. In fact, it only can be considered a hybrid of traditional detectors and deep ConvNets. Although its feature extractor is replaced by pretrained deep ConvNets, the classifier still uses a traditional model which is a set of class-specific linear SVMs. SPPnet [13] is also a hybrid model using convolutional layers to extract full-image features followed by a set of class-specific binary linear SVMs like RCNN. What is different is that the spatial pyramid pooling layer proposed by SPPnet enables feature extraction in arbitrary windows from the deep convolutional feature maps.

Fast RCNN [14] and Faster RCNN [15] make further evolution on the pipeline of object detection. Following the pioneering RCNN, Fast/Faster RCNN uses convolutional layers, initialized with discriminative pretraining for ImageNet [16] classification, to extract region-independent features followed by a regionwise multilayer perceptron (MLP) for classification. Besides, they jointly optimize a softmax classifier and bounding-box regressors, rather than training a softmax classifier, SVMs, and regressors in three separate stages. Nevertheless, this strategy ending with MLP classifier architectures is memory hog.

Based on a thorough study of regionwise feature classifier in Fast/Faster RCNN, our main work and contributions include the following three aspects. Firstly, we notably prove that the input size of the prevailing fully convolutional architectures must satisfy certain condition due to the concatenation of the convolutional layer and the pooling layer. Secondly, based on the detailed analysis of these fully convolutional architectures, we put forward how to employ recent state-of-the-art image classification networks such as ResNet [17] and various versions of GoogleNet [18, 19] which are by design fully convolutional into Fast/Faster RCNN detection systems. Finally, we adopt the idea of skip connection analogous to the hybrid of PVANET [20] and FPN [21] that combines several intermediate outputs. Consequently, the low-level visual features and high-level semantic features can be taken into account at the same time.

In the remainder of this paper, we derive a general formula for accurately designing the input size of the various fully convolutional networks in which the convolutional layer and the pool layer are concatenated (Section 2) and propose an efficient architecture of skip connection stemming from PVANET and FPN (Section 3). Finally, we provide abundant experimental results on VOC2007 benchmarks in Fast/Faster RCNN detection systems employing various fully convolutional networks including/excluding the architecture of skip connection, with detailed settings for training and testing (Section 4).

2. Deriving the Condition of the Input Size Based on Fully Convolutional Architectures

Beginning with LeNet-5 [22], convolutional neural networks have typically had a standard structure which includes some stacked convolutional layers optionally followed by local response normalization and pooling layer and ends with two 4096-d fully connected (fc) layers. Variants of this basic design are prevailing in the image classification literature, which have acquired the best results on MNIST [23], CIFAR [24], and most notably the ILSVRC competition. For larger datasets such as ImageNet, the latest trend has been to increase the depth and width of by design fully convolutional CNN architecture, whose fully connected layers are replaced by the global average pooling layer, while using dropout [25] to deal with the problem of overfitting and batch normalization [26] to accelerate deep network training by reducing internal covariate shift. Meanwhile, a latest pooling technique called Mean-Max Pooling is introduced in DPN, which can improve the performance of a well-trained CNN in the testing phase without any noticeable computational overhead.

These prevailing fully convolutional networks, such as ResNet and GoogleNet and their updated versions, have the effective stride, namely, 2^5 pixels, as same as the ZF [27]/VGG [28] networks. In other words, the effective stride increases two times by each stage. The difference is the way of reducing feature map size by half. The ResNet/GoogleNet specially designs a residual/inception building block while the ZF/VGG only adopts the max pooling layer. As we know, Faster RCNN system can take an image of any size as input. Feeding images with varying sizes is owed not only to the proposed RoI pooling layer but also to the architecture of the ConvNets such as ZF/VGG which are stacked with the max pooling layers and the convolutional layers.

Table 1 has illustrated the detailed network architectures of various CNN networks. As you can see, the special inception/residual block is designed to reduce the feature map size in the fully convolutional networks. Furthermore, the parameters of the special reduction blocks are precisely designed to enable the alignment of feature map size in the concatenation layer. If we want to employ the fully convolutional networks in Fast/Faster RCNN system, we present that the input size in these various ConvNets must satisfy certain condition. Taking the Inception_v3 architecture as an example, we calculate each feature map size according to the model parameters of convolutional and pooling layer. Based on the CAFFE [29] framework, the output size of each convolutional and pooling layer can be calculated precisely by the following two formulas:

$$\begin{aligned}&\text{output\_size}_{\text{conv}}\\&\quad=\left\lfloor\frac{\text{input\_size}+2*\text{pad}-[\text{dilation}*(\text{kernel\_size}-1)+1]}{\text{stride}}\right\rfloor\\&\qquad+1\\&\text{output\_size}_{\text{pool}}=\left\lceil\frac{\text{input\_size}+2*\text{pad}-\text{kernel\_size}}{\text{stride}}\right\rceil+1,\end{aligned}\tag{1}$$

where $\lfloor\ \rfloor$ and $\lceil\ \rceil$ are denoted as the floor and ceil function, respectively.

The detailed Inception_v3 architecture is defined in Table 2 according to the original paper. Because the output of the pooling layer will be concatenated with the outputs of the convolutional layers by the end of the inception block, these outputs must ensure the same feature map size. We can derive a general formula for accurately designing the input size of Inception_v3 by reversing-inference method, which is taken in two stages.

In consideration of the alignment issue of the feature map size in an inception block, we first calculate the input size of the reduction module consisting of a special inception block whose detailed parameters can be found in Table 2. For one reduction module, presumably the output size is x_{out} which represents a known positive integer and the input size is x_{in} which is unknown. Then we can establish a link between the output size and the input size by (1):

$$\begin{gathered}x_{\text{out}}=\left\lfloor\frac{x_{\text{in}}-3}{2}\right\rfloor+1,\\x_{\text{out}}=\left\lceil\frac{x_{\text{in}}-3}{2}\right\rceil+1,\\x_{\text{out}},x_{\text{in}}\in N^{+},\end{gathered}\tag{2}$$

where N^{+} is denoted as a set of positive integers. Therefore, we can obtain the solution of the equation set; that is, $x_{\text{in}}=x_{\text{out}}+1$. Assuming that Inception_v3 has m reduction modules, we can easily derive the relationship between the output size x_0

Table 1: ConvNet configurations. The convolutional layer parameters are denoted as "layer/receptive field size-stride." s2 stands for the case that the stride of the layer is 2 while stride 1 is omitted. All the receptive field size of convolutional layer is 3×3 which is omitted in ZF/VGG. LRN and GAP are short for local response normalization layer and global average pool layer while Inc and Res are the abbreviations for inception and residual building block, respectively. The ReLU activation function is not shown for brevity. And the batch normalization layer only included in the second three architectures is not shown as well.

ZF	VGG	GoogleNet	Inception_v2	Inception_v3	ResNet_50
Conv1-s2	Conv1_1	Conv1/7 × 7-s2	Conv1/7 × 7-s2	Conv1/3 × 3-s2	Conv1/7 × 7-s2
LRN	Conv1_2	MaxPool-s2	MaxPool-s2	Conv2/3 × 3	MaxPool-s2
MaxPool-s2	MaxPool-s2	LRN	Conv2/1 × 1	Conv3/3 × 3	Res2a
Conv2-s2	Conv2_1	Conv2/1 × 1	Conv2/3 × 3	MaxPool-s2	Res2b
LRN	Conv2_2	Conv2/3 × 3	MaxPool-s2	Conv4/1 × 1	Res2c
MaxPool-s2	MaxPool-s2	LRN	Inc_3a	Conv4/3 × 3	Res3a-s2
Conv3	Conv3_1	MaxPool-s2	Inc_3b	MaxPool-s2	Res3b
Conv4	Conv3_2	Inc_3a	Inc_3c-s2	Inc_a1	Res3c
Conv5	Conv3_3	Inc_3b	Inc_4a	Inc_a2	Res3d
MaxPool-s2	MaxPool-s2	MaxPool-s2	Inc_4b	Inc_a3	Res4a-s2
Fc_4096	Conv4_1	Inc_4a	Inc_4c	Inc_a-s2	Res4b
Fc_4096	Conv4_2	Inc_4b	Inc_4d	Inc_b1	Res4c
	Conv4_3	Inc_4c	Inc_4e-s2	Inc_b2	Res4d
	MaxPool-s2	Inc_4d	Inc_5a	Inc_b3	Res4e
	Conv5_1	Inc_4e	Inc_5b	Inc_b4	Res4f
	Conv5_2	MaxPool-s2	GAP	Inc_b-s2	Res5a-s2
	Conv5_3	Inc_5a		Inc_c1	Res5b
	MaxPool-s2	Inc_5b		Inc_c2	Res5c
	Fc_4096	GAP		GAP	GAP
	Fc_4096				
Fc_1000					
Softmax					

Table 2: The detailed feature size of each layer in the Inception_v3 architecture.

Layers	Input size	Pad	Kernel size	Output size
Conv1-s2	$2^5(n+1)+p\ (p=5\sim12)$	0	3 × 3	$2^4n+r\ (r=18\sim21)$
Conv2	$2^4n+r\ (r=18\sim21)$	0	3 × 3	$2^4n+r\ (r=16\sim19)$
Conv3	$2^4n+r\ (r=16\sim19)$	1	3 × 3	$2^4n+r\ (r=16\sim19)$
MaxPool-s2	$2^4n+r\ (r=16\sim19)$	0	3 × 3	2^3n+8 or 2^3n+9
Conv4_1	2^3n+8 or 2^3n+9	0	1 × 1	2^3n+8 or 2^3n+9
Conv4_2	2^3n+8 or 2^3n+9	0	3 × 3	2^3n+6 or 2^3n+7
MaxPool-s2	2^3n+6 or 2^3n+7	0	3 × 3	2^2n+3
Inc_a1~Inc_a3	2^2n+3	1	3 × 3	2^2n+3
Inc_a-s2				
Conv	2^2n+3	0	3 × 3	$2n+1$
Pool	2^2n+3	0	3 × 3	$2n+1$
Inc_b1~Inc_b4	$2n+1$	1	3 × 3	$2n+1$
Inc_b-s2				
Conv	$2n+1$	0	3 × 3	n
Pool	$2n+1$	0	3 × 3	n
Inc_c1~Inc_c2	n	1	3 × 3	n
GAP				
Fc_1000				
Softmax				

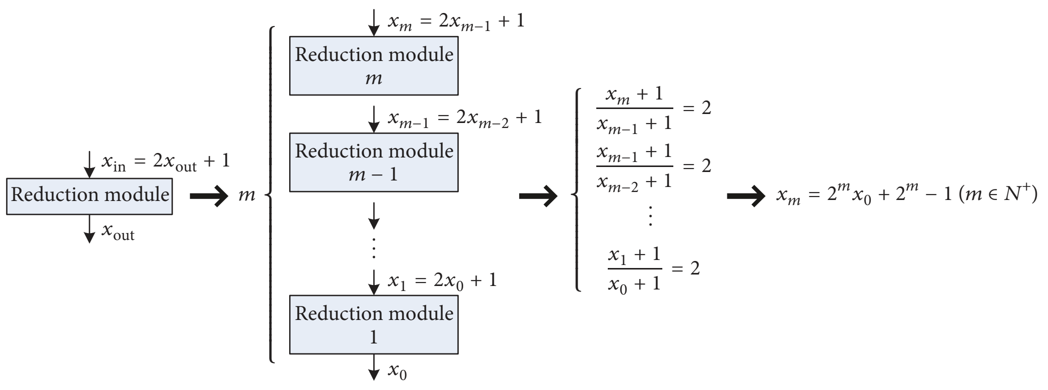

FIGURE 1: The derivation process of the initial input size of m reduction modules.

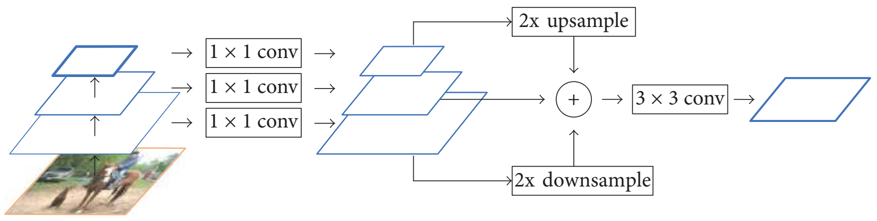

FIGURE 2: The skip-layer connection architecture.

of the final reduction module and the input size x_m of the first reduction module which is $x_m = 2^m x_0 + 2^m - 1\ (m \in N^+)$. The detailed derivation process has been shown in Figure 1.

The remaining part of Inception_v3 only consists of some stacked convolutional layers and pooling layers which are in front of the Inception_v3 architecture. Similarly, we can derive the relationship between the input size x_{image} and the output size x_m of the part by (1).

$$x_{\text{image}} = 2^3 x_m + r \quad r \in \{13, 14, \ldots, 19, 20\}. \tag{3}$$

Finally, we can obtain a general formula about the input size of Inception_v3 architecture which is as follows:

$$x_{\text{image}} = 2^{m+3}\left(x_0 + 1\right) + p \quad p \in \{5, 6, \ldots, 11, 12\}. \tag{4}$$

The detailed input size of each layer for Inception_v3 is shown in Table 2 where m equals 2. It is worth noting that the dilation of each layer is set to 1 which is omitted in Table 2. In the same way, we can derive the formula belonging to the input size of other ConvNets. From here we can see that their input sizes must be ensured to satisfy certain conditions when these fully convolutional networks are employed by the Fast/Faster RCNN system.

3. Skip-Layer Connections

Many studies have shown that multiscale representation and its combination are effective in many recent deep learning tasks. In essence, multiscale representation is a skip-layer connection method combining fine-grained details with highly abstracted information in feature extraction layer, which contributes to the following region proposal and classification network. Our skip-layer connection architecture is more like a combination of the observation from PVANET and FPN.

Our design choice combines the last and two intermediate layers whose scales of feature map are two times and four times the last layer, respectively. The backbone ConvNet computes a feature hierarchy including feature maps at five scales with a scaling step of 2. The feature maps of some layers have the same scale and we say these layers are in the same network stage. As we know, most ConvNets have five network stages which have strides of $\{2, 2^2, 2^3, 2^4, 2^5\}$ pixels with respect to the input image. Our multiscale features can be obtained by combining three network stages whose strides are $\{2^3, 2^4, 2^5\}$ pixels. Besides, the output of the last layer of each stage is chosen as our reference set of feature maps.

There are four steps to obtain our multiscale features as shown in Figure 2. To combine multilevel maps at the same resolution, different sampling strategies are carried out for different stages. The stage with stride 2^3 is downscaled by 3×3 MaxPool with stride 2 while the stage with stride 2^5 is upscaled by channelwise deconvolution whose weights are fixed as bilinear interpolation. For the purpose of merging the features of the three stages by elementwise addition, a 1×1 convolutional layer is used to adjust their channel dimensions to a fixed size which is set as 256 in our experiments. At last, a 3×3 convolution layer on the merged feature map is appended to generate the final feature map, which is to reduce the aliasing effect of upsampling.

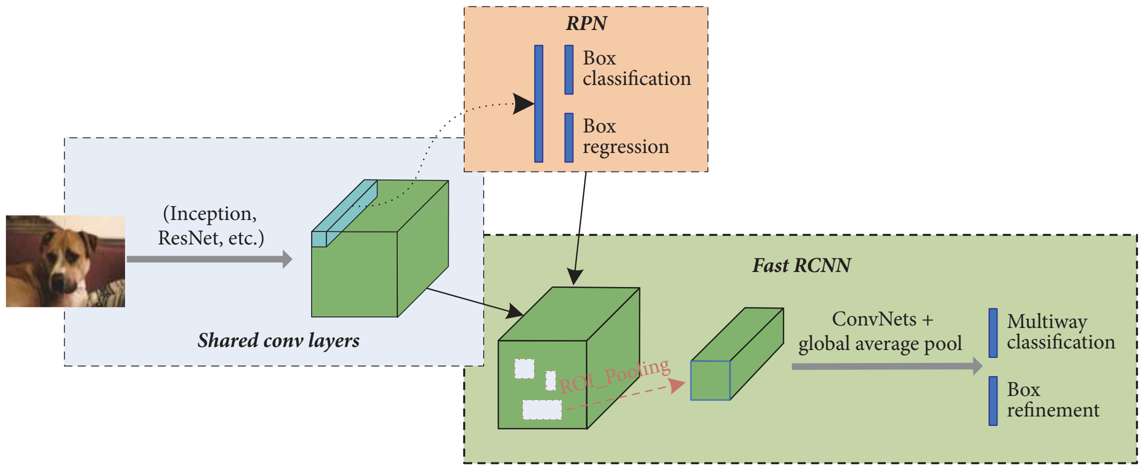

Figure 3: Faster RCNN diagram using fully convolutional networks.

4. Experiments

4.1. Experimental Setup. As we know, Faster RCNN system includes two stages shown in Figure 3. The first stage called the region proposal network (RPN) processes the input of arbitrary size by some shared convolutional layers known as feature extractor. And then the extracted features output by the last shared convolutional layer are slid over with a 3×3 spatial window by a small network. Each sliding window is then mapped to a lower-dimensional feature which is fed into two sibling 1×1 convolutional layers for box regression and box classification, respectively. The second stage includes a detection network where Fast RCNN is adopted. The proposals generated by the RPN and the shared convolutional features are fed into the RoI pooling layer followed by the remaining layers of the backbone ConvNet in order to predict a class and class-specific box refinement for each proposal.

We only perform all our experiments on the 20 category PASCAL VOC2007 detection dataset. Our code is based on the official Faster RCNN code written in MATLAB which also includes the reimplementation of Fast RCNN. As a common practice, all network backbones used are pretrained on the 1000-class ImageNet dataset and then fine-tuned on the detection dataset. We investigate the pretrained GoogleNet, Inception_v2, Inception_v3, and ResNet-50 models that are publicly available. From the above analyses, any latest network can be similarly used as a backbone ConvNet in Fast/Faster RCNN, provided that the issue on the input size is solved.

All parameters related to Fast/Faster RCNN were set as in the original work except that the shorter edge of each input image was resized to be 587. What is noteworthy is that the last max pooling layer of ZF/VGG is replaced by a RoI pooling layer in the original Fast/Faster RCNN, which leads to an effective output stride of 2^4 instead of 2^5. To put all ConvNets on an equal foot, we slightly modify the original models by modifying the last network stage to have stride 1 instead of 2. Furthermore, the atrous [30] convolution is used in the last network stage to compensate for reduced stride. Except for GoogleNet, the batch normalization, whose parameters are frozen to be those estimated during ImageNet pretraining, is used after convolutional layers. All experiments were performed on Intel i7-6700K CPU and NVIDIA GTX1080 GPU.

4.2. Experiments on ConvNets for Fast/Faster RCNN System. RoI pooling is used to pool regionwise features from the shared convolutional feature maps. And it generates a fixed-size feature map for each proposal replacing the last pooling layer while ZF/VGG is the backbone network. Naturally, the remaining fc layers are used as regionwise classifier typically. Differing from ZF and VGG, other ConvNets replace the fc layers with a global average pooling layer. Therefore, we must choose the layer to insert RoI pooling layer for these ConvNets. Based on our observation, we choose the last network stage to insert RoI pooling layer to ensure the effective output stride of 2^4. Besides, we set the output size of RoI pooling as 7×7 like VGG in all cases. Because three layers are included in the last network stage, we perform experiments on different layers of various depths for Fast/Faster RCNN system and get some exciting results shown in Table 3. According to the experimental results, we intuitively make an argument that a deeper convolution-based regionwise classifier is more effective but more time-consuming and is in general orthogonal to more powerful and deeper feature map. Moreover, we also perform original Fast RCNN experiments based on ZF/VGG and get the detection mAP of 58.9 and 68.2, respectively, which are significantly improved by Inception_v3 and ResNet_50. Besides, the results also indicate that the fully convolutional networks have smaller final model size due to lack of fully connected layers.

As discussed above, our skip-layer connection is used in the feature extraction stage by combining three network stages whose strides are $\{2^3, 2^4, 2^5\}$ pixels. Then all the stages of the fully convolutional networks are considered as a feature extractor. According to the previous experiment results, the final detection mAP is much less than others when the RoI pooling layer is only followed by the global average pooling layer. Consequently, we replace the global average pooling

Table 3: The results of Fast RCNN using convolution-based regionwise classifier. "Inc" indicates an inception block and "Res" indicates a residual block. CRC is short for convolution-based regionwise classifier. The training time refers to the consuming time of 20 iterations.

Network	CRC	Training time	Model size	mAP (%)
GoogleNet	Inc_5a, 5b, GAP	15 s	23.2 MB	65.8
	Inc_5b, GAP	7 s		62.9
	GAP	5 s		41.2
Inception_v2	Inc_4e, 5a, 5b, GAP	17 s	39.3 MB	67.0
	Inc_5a, 5b, GAP	14 s		66.6
	Inc_5b, GAP	10 s		65.6
	GAP	7 s		42.9
Inception_v3	Inc_b, c1, c2, GAP	27 s	84.1 MB	69.8
	Inc_c1, c2, GAP	23 s		69.4
	Inc_c2, GAP	18 s		68.9
	GAP	12 s		49.7
ResNet_50	Res_5a, 5b, 5c, GAP	17 s	90.7 MB	71.5
	Res_5b, 5c, GAP	13 s		70.9
	Res_5c, GAP	10 s		69.5
	GAP	8 s		50.4

Table 4: The results of Fast RCNN with skip-layer connections.

Network	Training time	Model size	mAP (%)
GoogleNet	6 s	129 MB	64.0
Inception_v2	11 s	145 MB	65.2
Inception_v3	16 s	191 MB	68.9
ResNet_50	14 s	200 MB	70.8

layer with two 1024-d fc layers. They are randomly initialized by the Xavier method due to having no pretrained fc layers available. Each fc layer is followed by a ReLU layer and a dropout layer with the dropout ratio of 0.25. Similarly, we perform experiments on different backbone ConvNets with our skip-connection layers for Fast RCNN system. The experiment results are shown in Table 4, which indicates our skip-connection architecture is fairly effective and the skip-connection architecture has almost identical performance as the third convolution-based regionwise classifier. Obviously, the fc layers account for most of the final model compared to convolutional layers. Besides, we assume that the performance can be improved further while using more fc layers.

In fact, Faster RCNN innovatively merges the proposed RPN and Fast RCNN into a single network by sharing their convolutional features. Therefore, we only perform some experiments on the above best case for Faster RCNN. Due to a lack of time, we only experiment on the ResNet_50 and get the detection mAP of 73.1 which is boosted by almost two percentage points in comparison with Fast RCNN. What is more, we try to run some experiments on the convolution-based regionwise classifier where our skip-layer connection is used in the feature extraction stage by combining three network stages whose strides are $\{2^2, 2^3, 2^4\}$ pixels. To our great pity, the obtained detection mAP is fairly low, only 62.4 and 65.9 for Inception_v2 and ResNet_50, respectively. Therefore, we put forward an argument that combining higher feature is more effective.

5. Conclusions

In this paper, we have presented how to use the prevailing fully convolutional architectures in the notable object detection systems such as Fast/Faster RCNN. Specifically, we have derived a general formula for accurately designing the input size of the various fully convolutional networks in which the convolutional layer and pooling layer are concatenated with their strides being greater than 1 and have proposed an efficient architecture of skip connection to accelerate the training process. It is worth noting that our experiments are only performed on the VOC2007 set for some reason. We strongly believe that the results clearly can be boosted by a large margin as using more training data. We believe that our theoretical analysis and experiments can bring in some insights into how to employ other CNN architectures in single-stage or two-stage object detection systems. Besides, we will leverage the Faster RCNN whose backbone ConvNet is replaced with the ResNet_50 to detect small objects in optical remote sensing image by accurately modifying the strides in the future work.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] J. Huang, V. Rathod, C. Sun et al., "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '17)*, pp. 3296-3297, Honolulu, HI, USA, July 2017.

[2] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun, "Object detection networks on convolutional feature maps," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 7, pp. 1476–1481, 2017.

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, vol. 1, pp. 886–893, June 2005.

[5] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[6] X. Wang, M. Yang, S. Zhu, and Y. Lin, "Regionlets for generic object detection," in *Proceedings of the 2013 14th IEEE International Conference on Computer Vision (ICCV '13)*, pp. 17–24, December 2013.

[7] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013.

[8] S. Yan, Y. Xia, J. S. Smith, W. Lu, and B. Zhang, "Multiscale Convolutional Neural Networks for Hand Detection," *Applied Computational Intelligence and Soft Computing*, vol. 2017, pp. 1–13, 2017.

[9] Y. Sun, L. Zhu, G. Wang, and F. Zhao, "Multi-input convolutional neural network for flower grading," *Journal of Electrical & Computer Engineering*, vol. 2017, Article ID 9240407, 8 pages, 2017.

[10] Y. Xu, G. Yu, Y. Wang, X. Wu, and Y. Ma, "Car detection from low-altitude UAV imagery with the faster R-CNN," *Journal of Advanced Transportation*, vol. 2017, Article ID 2823617, 11 pages, 2017.

[11] L. Zhou, Q. Li, G. Huo, and Y. Zhou, "Image classification using biomimetic pattern recognition with convolutional neural networks features," *Computational Intelligence and Neuroscience*, vol. 2017, Article ID 3792805, 12 pages, 2017.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the 27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14)*, pp. 580–587, Columbus, Ohio, USA, June 2014.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Proceedings of the European Conference on Computer Vision*, vol. 8691, pp. 346–361, Springer, 2014.

[14] R. Girshick, "Fast R-CNN," in *Proceedings of the 15th IEEE International Conference on Computer Vision (ICCV '15)*, pp. 1440–1448, December 2015.

[15] 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*, pp. 91–99, 2015.

[16] J. Deng, W. Dong, and R. Socher, "ImageNet: a large-scale hierarchical image database," in *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR' 09)*, pp. 248–255, IEEE, Miami, FL, USA, June 2009.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*, pp. 770–778, July 2016.

[18] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '15)*, pp. 1–9, Boston, MA, USA, June 2015.

[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*, pp. 2818–2826, July 2016.

[20] K. Kim H, S. Hong, B. Roh et al., "PVANET: deep but lightweight neural networks for real-time object detection," *Computer Vision and Pattern Recognition*, 2016, https://arxiv.org/abs/1608.08021.

[21] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," Computer Vision and Pattern Recognition, 2016, https://arxiv.org/abs/1612.03144.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] Y. LeCun, C. Cortes, and C. J. C. Burges, "MNIST handwritten digit database," *AT & T Labs*, p. 2, 2010, http://yann.lecun.com/exdb/mnist.

[24] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 dataset," 2014, http://www.cs.toronto.edu/kriz/cifar.html.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML '15)*, pp. 448–456, July 2015.

[27] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proceedings of the European conference on computer vision*, vol. 8689 of *Lecture Notes in Computer Science*, pp. 818–833, Springer, 2014.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Computer Vision and Pattern Recognition*, 2014, https://arxiv.org/abs/1409.1556.

[29] Y. Jia, E. Shelhamer, J. Donahue et al., "Caffe: convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678, ACM, Orlando, FL, USA, November 2014.

[30] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *Computer Vision and Pattern Recognition*, 2015, https://arxiv.org/abs/1511.07122.