2018 24th International Conference on Pattern Recognition (ICPR)
Beijing, China, August 20-24, 2018

ThinNet: An Efficient Convolutional Neural Network for Object Detection

Sen Cao, Yazhou Liu[*], Changxin Zhou, Quansen Sun
School of Computer Science and Engineering
Nanjing University of Science and Technology
Nanjing, China
csaimd@foxmail.com, {yazhouliu[*], cxzhou
sunquansen}@njust.edu.cn

Lasang Pongsak, Sheng Mei Shen
Core Technology Group
Panasonic R&D Center Singapore
Singapore, Singapore
{pongsak.lasang, shengmei.shen} @sg.panasonic.com

***Abstract*—Great advances have been made for the deep networks, but relatively high memory and computation requirements limit their applications in the embedded device. In this paper, we introduce a class of efficient network architecture named ThinNet mainly for object detection applications on memory and computation limited platforms. The new architecture is based on two proposed modules: Front module and Tinier module. The Front module reduce the information loss from raw input images by utilizing more convolution layers with small size filters. The Tinier module use pointwise convolution layers before conventional convolution layer to decrease model size and computation, while ensuring the detection accuracy. Experimental evaluations on ImageNet classification and PASCAL VOC object detection datasets demonstrate the superior performance of ThinNet over other popular models. Our pretrained classification model(ThinNet_C) attains the same top-1 and top-5 performance as the classic AlexNet but only with 1/50th the parameters. The detection model also obtains significant improvements over other detection methods, while requiring smaller model size to achieve high performance.**

Keywords—deep learning; object detection; ThinNet

I. Introduction

Convolutional Neural Networks (CNNs) have produced impressive performance improvements in many computer vision tasks, such as image classification [1, 4-7], object detection[8] [11] [14]and image segmentation[15]. Since AlexNet [1] popularized deep convolutional neural networks by winning the ImageNet Challenge ILSVRC 2012, many innovative CNNs network structures have been proposed. Szegedy et al. [7] propose an “Inception” module which is comprised of a number of different sized filters. He [4] propose ResNet with skip connection over multiple layers. Huang [5] propose DenseNet with direct connection from any layer to all subsequent layers, which enables training very deep networks with more than 200 layers. Due to these excellent network structures, the quality of many vision tasks has been growing at a dramatic pace. Among them, object detection is one of the most benefited areas because of its wide applications in intelligent monitoring, security system, and autonomous driving.

In order to achieve higher accuracy, the general trend is to make deeper and more complicated networks. Recent evidence reveals that network depth is of crucial importance, and the leading results [4, 5, 7] on the challenging ImageNet dataset all exploit “very deep” models. However, these advances to improve accuracy are not necessarily making networks more efficient with respect to size and speed. There has been rising interest in building small and efficient neural networks in the recent literature [10, 13, 16]. Generally, given equivalent accuracy, the lightweight models with fewer parameters have several advantages: (1) small models can be trained faster with less data due to its smaller parameter space. (2) small models can be deployed on the platforms with low power and computation cost. (3) Small models can be exported from cloud to client with less overhead.

In this paper, the basic principle is: pursuing the competitive accuracy in very limited computational and memory cost, which may be used for computationally and memory limited platforms such as drones, robots, and phones. We develop a class of efficient model called ThinNet. *Fig. 1* illustrates the overview of the detection framework and the proposed ThinNet architecture that consists of Front module and Tinier module. Front module reduce information from raw input images and Tinier module’s role is to decrease parameters and computation without reducing accuracy. We evaluate our models on the challenging ImageNet classification [17] and PASCAL VOC object detection [18] tasks. A series of comparative experiments shows the effectiveness of our design principles and the better performance over other network structures.

The rest of the paper is organized as follows. In section 2, we review related works; in section 3, we detail the Front module and Tinier module, and describe ThinNet architecture; in section 4, we describe comparative experiments on benchmark datasets. Finally, we conclude in section 5.

II. ReLate WoRks

A. Object Detect Framework

In the past few years, mainly due to the advances of deep learning, more specifically convolutional neural networks [1, 4, 5, 7], the performance of object detection has been progressing at a dramatic pace. RCNN [11] firstly to show that the CNNs can lead to dramatically higher object detection performance on PASCAL VOC as compared to systems based on traditional methods. R-CNN requires high computational costs since each region is processed by the CNNs network separately. Fast R-

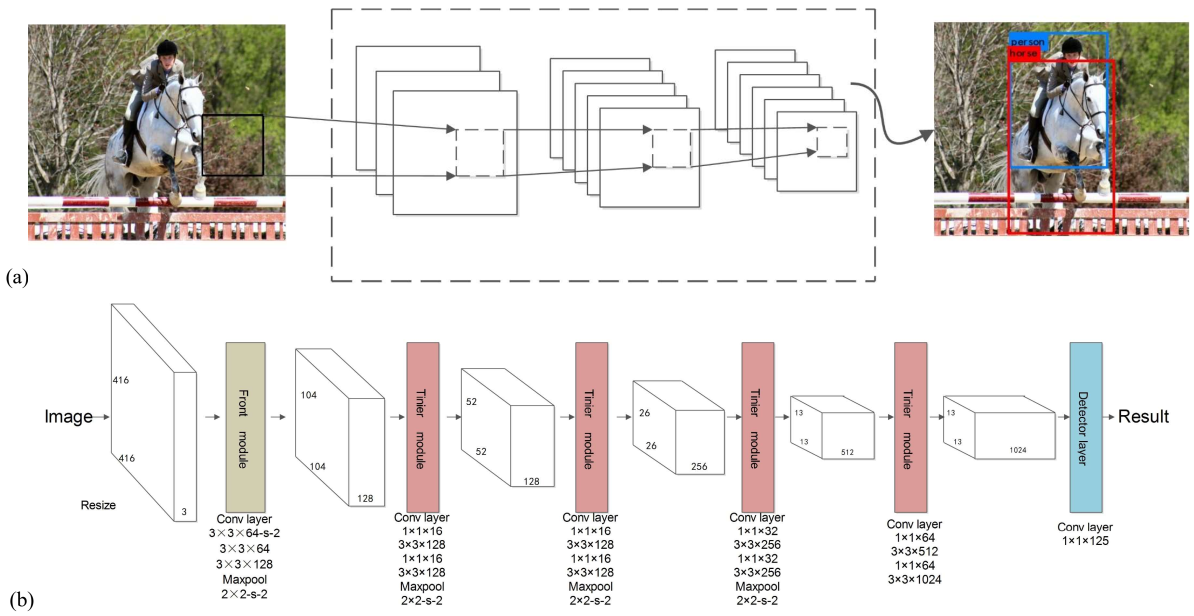

Fig.1. Overview of object detection system(a) and network architecture(b).

CNN [19] and Faster R-CNN [2] improve the efficiency by sharing computation and using neural networks to generate the region proposals.

RCNN, Fast RCNN, Faster RCNN and other improved methods based on RCNN achieve excellent object detection accuracy by using a deep CNNs to classify object proposals. They can be summarized as region proposal based methods. However, these region-proposal based methods have several notable drawbacks: 1) Training is a multi-stage pipeline; 2) Training is expensive in space and time; 3) Object detection is slow. Thus, some real-time object detection methods are proposed, such as YOLO [8] and SSD [14]. They use a single feed-forward convolutional network to directly predict bounding boxes and associated class probabilities. Comparing with the region-based methods, these methods no longer requires a second per-region classification operation so that it is extremely fast.

B. Network Architecture Design

In order to achieve higher accuracy, building deeper and larger convolutional neural networks (CNNs) is a primary trend. As in [6], Simonyan and Zisserman investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition task and propose the popular VGG net which reaches 19 layers. He [4] propose residual learning blocks with skip connection, which enables training very deep networks with more than 100 layers. DenseNet [5] with direct connection from any layer to all subsequent layers have more than 200 layers. Larger networks perform well on expensive, GPU-based machines, however, deeper network architectures are often unsuitable for smaller devices with limited memory and computation power.

Recently, the increasing demands of running high quality deep neural networks on computationally and memory limited devices encourage the study on lightweight model designs. For example, Han et al. [10] proposed "deep compression" with three stage pipelines: pruning, trained quantization and Huffman coding, that work together to reduce the storage requirement of neural networks. Andrew G et al. [16] propose MobileNet based on a streamlined architecture that uses depth wise separable convolutions to build lightweight deep neural networks. These different approaches can be generally categorized into either compressing pretrained networks or training small networks directly.

Thanks to these excellent detection methods and network structures, they provide a lot of inspiration for designing our own model. we designed Front module and Tinier module as the backbone architecture of the ThinNet, and use 1×1 convolution layers to reduce the parameters and computation. They will be introduced in more detail in next section. ThinNet is built upon the YOLOv2 [12] framework, and it frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

III. METHOD

The overview of object detection system is illustrated in *Fig. 1(a)*. The detection system can be divided into three steps: resizing the input images to 416×416; running the ThinNet on the image and predicting multiple bounding boxes and class probabilities for those boxes; selecting the test results based on the confidence of the model. The ThinNet architecture consists of three parts: Front module, Tinier module and Detector layer. The full network is shown in *Fig. 1(b)*. We elaborate each component and corresponding design principles in the following.

A. Strategy of decraasing parameters

It is known that the number of parameters in a convolutional layer is related to the number of input channels, the size of filters,

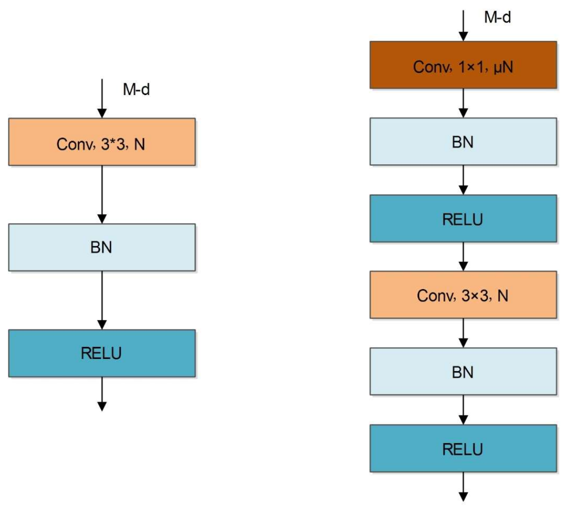

Fig.2. Left: conventional 3 × 3 convolution layer with batch normalization and ReLU. Right: adding 1×1 convolution layer before conventional convolution layer.

and the number of filters. Consider a convolution layer that is comprised entirely of 3×3 filters, the total quantity of parameters in this layer is computed as:

$$T_l = M_l \times N_l \times 3 \times 3 \quad (1)$$

Where T_l is the total quantity of parameters in the l^{th} layer, M_l stands for the channels of input feature map, N_l is the number of filters, and also denotes the number of output channels. Therefore, according to (1), there are two strategies to reduce the number of parameters and computational complexity: (1) decrease the number of 3×3 filters, which is referred to as N_l; (2) decrease the number of input channels to the 3x3 convolution layer, which is referred to as M_l. We choose the second strategy, reducing M_l, because directly reducing the number of 3×3 filters in a lightweight model will make the accuracy drop dramatically and it is difficult to find the optimal solution.

After our exploration, we reduce the parameters by using a 1×1 convolution(also called pointwise convolution in [16]) before each 3×3 convolution. *Fig. 2* contrasts a layer with regular convolution, batch normalization and ReLU nonlinearity to our proposed layer with regular convolution, 1×1 convolution as well as batch normalization and ReLU after each convolutional layer. The 1×1 convolution layers in our method have two important effects:

1) *Adding non-linearity*

In conventional CNNs, convolution layers take inner product of the linear filter and the underlying receptive field followed by a nonlinear activation function at every local portion of the input. This linear convolution is sufficient for abstraction when the instances of the latent concepts are linearly separable. However, the features we wish to extract are highly non-linear in general. The 1×1 convolution before the conventional 3×3 convolution corresponds to a linear combination of different channels for each pixel. Since the input of the convolution layer is a three-dimensional structure with multiple channels instead of a planar structure, the network model can greatly achieve the spatial information fusion of the feature maps from different channels. This operation is efficient to increase the non-linearity of the decision function and the representational power of neural networks.

2) *Decreasing the quantity of parameter*

In the deep convolution neural network, there is redundancy in the output feature maps. The method used in [20] is to design a threshold function to evaluate the importance of output feature maps, and pruning unimportant channels. We use another method that decreasing the input dimensions through the 1×1 convolution, not only to reduce the model parameters, but also to reduce the impact of the redundancy of the feature map. In order to decrease the input dimensions we introduce a very simple parameters μ. For a given 3×3 convolution layer and the parameter μ, the number of input channels M becomes αM. Comparing the quantity of parameters in *Fig. 2*:

Standard 3×3 convolutional layer (*Fig. 2(left)*): M×N×3×3

Ours (*Fig. 2(right)*):M× μM ×1×1+μM ×N×3×3

where μ=0.25 in our experiments, and we get 8 to 9 times less parameters than standard convolution.

B. *Tinier module and Front module*

With the trend of designing very deep CNNs, state-of-the-art networks [13, 21, 22] usually consist of repeated building blocks or modules with the same structure. Generally, carefully designed blocks are able to achieve better performance with low theoretical complexity. So, in order to take advantages of module structure, we proposed the novel Tinier module and Front module. As shown in *Fig. 3(right)*, a Tinier module is comprised of two 1×1 convolution layers and two 3×3 convolution layers, where the liberal use of 1×1 layer is responsible for reducing dimensions. Each convolution layer is followed by batch normalization and ReLU operations. Batch normalization was used in many state-of-the-art models since it was introduced. It leads to significant improvements in convergence while eliminating the need for other forms of regularization.

Inspired by Inception-v3 [21] and v4 [22], we define Front module as a stack of three 3×3 convolution layers followed by

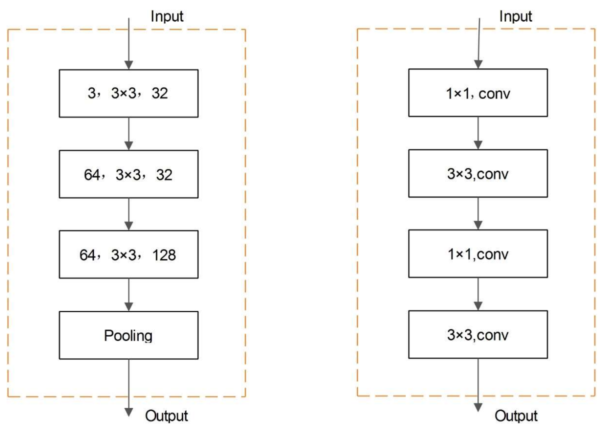

Fig.3. Overview of Front module (left) and Tinier module(right)

TABLE I. DETAILED INFORMATION ON THINNET ARCHITECTURE

Type	Output size	Filter size/stride	$N_{1\times1}$	$N_{3\times3}$	Parameter
input image	416×416×3				
Front module	104×104×128	$\begin{bmatrix}3\times3,32,2\\3\times3,32,1\\3\times3,64,1\end{bmatrix}$			112320
Tinier1	104×104×64	$\begin{bmatrix}1\times1/1\\3\times3/1\end{bmatrix}\times2$	16	128	40960
Tinier2	52×52×128	$\begin{bmatrix}1\times1/1\\3\times3/1\end{bmatrix}\times2$	32	256	159744
Tinier3	26×26×256	$\begin{bmatrix}1\times1/1\\3\times3/1\end{bmatrix}\times2$	64	512	636976
Tinier4	13×13×1024	$\begin{bmatrix}1\times1/1\\3\times3/1\end{bmatrix}\times2$	128	1024	3155904
Detector layer	13×13×125	1×1,125,1			12800
					3.6M(total)

($N_{1\times1}$ and $N_{3\times3}$ represent the number of 1×1 filter and 3×3 filter respectively.)

a 2×2 max pooling layer as shown in *Fig. 3(left)*. The first convolution layer works with stride = 2 and the other two are with stride = 1. We find that adding this simple Front module can evidently improve the detection performance in our experiments. A possible explanation is that, compared with the ResNet and DenseNet (7×7 convolution layer, stride = 2 followed by a 3×3 max pooling layer, stride = 2), the downsmapling is applied at the end of Front module, therefore convolution layers have large feature maps. Our intuition is that large feature maps (due to delayed downsampling) can evidently improve the detection performance because of reducing the information loss from raw input images.

C. ThinNet Architecture

1) Details of ThinNet

Built on Front module and Tinier module, the overall ThinNet architecture is shown in *Fig. 1(b)*. In our experiments on VOC, we use the ThinNet with one Front module and four Tinier modules. The beginning of the network is a Front module, followed by a stack of four Tinier modules. The final detector layer is a 1×1 convolution layer with linear activation. ThinNet performs max-pooling with a stride of 2 after every Tinier module except the last module. The full ThinNet architecture is detailed in *Table I*. In the first Tinier module, the number of filters used in the two 3×3 convolutional layers is 64 and then the number of channels are doubled after every pooling step. Through our experimental exploration, we set the number of 1×1 filters to 1/4 (μ=0.25) of the output channels from the previous module for each Tinier module. As shown in the last column of *Table 1*, we also calculated parameters of ThinNet parameters. The total parameters of ThinNet is 3.6M, less than other small models, such as Fast YOLO and Tiny YOLO, which are 27.1M and 15.8M, respectively.

2) Visual explanation for ThinNet

In order to have a more intuitive explanation about what have been learned by different modules of the network, we generate the heat map of different blocks of the proposed ThinNet using Grad_CAM [23]. As shown in *Fig. 4*, it is clearly that different layers of network architecture have learned different key information. Furthermore, as the number of network layers deepens, key information begins to converge. This visualization process helps us to identify dataset bias and lend insight into failures of current, showing that seemingly unreasonable predictions have reasonable explanations.

IV. EXPERIMENTS

In this section, we evaluate the proposed ThinNet on public available datasets: ImageNet classification and PASCAL VOC detection. First, we compare ThinNet_C (pretrained classification model on ImageNet) with some popular classification models. Then, we compare ThinNet with several well-known small detection models in terms of accuracy and model size.

A. Dataset Description and Implementation Details

We conduct experiments on the widely used ImageNet 2012 classification [17], PASCAL VOC 2007, 2012 detection [18] datasets. The ImageNet 2012 classification dataset consists 1.2 million images for training and 50000 for validation, from 1000 categories. The PASCAL VOC detection benchmark consists of VOC2007 trainval, OVC2007 test, VOC2012 trainval, VOC2012 test. There are twenty classes of natural scene images in VOC dataset, such as person, dog, cat, boat and etc.

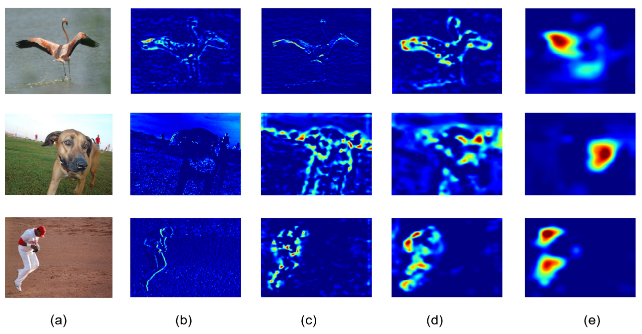

Fig.4. Visual explanation of key information learned at different layers: (a)original images; (b)Heat map after Front module; (c)Heat map after Tinier1; (d)Heat map after Tinier2; (e)Heat map after Tinier3.

TABLE II. THINNET_C COMPARISON TO POPULAR MODELS

Model	Top-1	Top-5	Size
AlexNet [1]	57.0	80.3	238MB
AlexNet_SVD [3]	56.0	79.4	56MB
AlexNet_purning [9]	57.2	80.3	27MB
Deep compression [10]	57.2	80.3	6.9MB
SqueezeNet [13]	57.5	80.3	4.8MB
SqueezeNet_Bynass [13]	58.8	82.0	7.7MB
MobileNetv1_0.50_160 [16]	59.5	82.5	24.5MB
MobileNetv1_0.50_128 [16]	56.2	79.6	24.5MB
MobileNetv1_0.25_224 [16]	50.0	75.0	11.4MB
ThinNet-C(ours)	58.7	81.7	4MB

(The top-1 and top-5 error rates (%, smaller number represents better performance))

We modify ThinNet to pretrain on ImageNet dataset [17] by removing the last Tinier module and Detector layer, and addone 1 × 1 convolutional layer with 1000 filters. We train the ThinNet_C (ThinNet for Classification) on the standard ImageNet classification dataset for 16 epochs using stochastic gradient descent with a starting learning rate of 0.1, polynomial rate decay with a power of 4, weight decay of 0.0005 and momentum of 0.9.

Our detection models are trained based on the union of VOC 2007 trainval and VOC2012 trainval ("07+12") detection benchmark [18], consisting of about 16k trainval images. Then we comprehensively evaluate our models on VOC2007 test set (4952 images). Our detection model is trained from pretrained modle(ThinNet-C) with SGD solver on NVidia TitanX GPU. We use a batch size of 64. This batch size requires a small amount of GPU memory so the proposed model can be easily trained within a limited amount of time. We train the network for 160 epochs with a starting learning rate of 10^{-3}, dividing it by 10 at 60 and 90 epochs. Following [14], we use a weight decay of 0.0005 and a momentum of 0.9.

B. Classification on ImageNet

In order to achieve better performance, we pretrain the ThinNet on the ILSVRC 2012 classification dataset [17]. The classification model is called ThinNet_C, which is a little different from the detection model. This model is designed to be small but powerful. It attains the same top-1 and top-5 performance as the classic AlexNet but with less parameters. The results of ThinNet_C are compared with five popular classification models which cover both original larger model and state-of-the-art lightweight model, including Alexnet [1], AlexNet_SVD , AlexNet_pruning [9], Deep Compression [10], SqueezeNet [13], MobileNet [16].

Tabel II shows the performance of different models in terms of top1 and top5 accuracy rates. Among them, AlexNet_SVD, AlexNet_purning and Deep Compression are compression methods on AlexNet, which prune the weights but not change the original AlexNet architecture. Squeezenet and MobileNet, focusing on efficient novel network design for embedded devices, both achieve state-of-the-art results on small models. It is clearly that ThinNet_C achieve comparable performance while requiring significantly fewer parameters than other models. For example, comparing with original AlexNet, ThinNet_C obtain higher accuracy while being 50× smaller than AlexNet, which is better than compression methods. Compared to SqueezeNet and MobileNet, the accuracy is slightly lower, but ThinNet is more competitive in terms of model size.

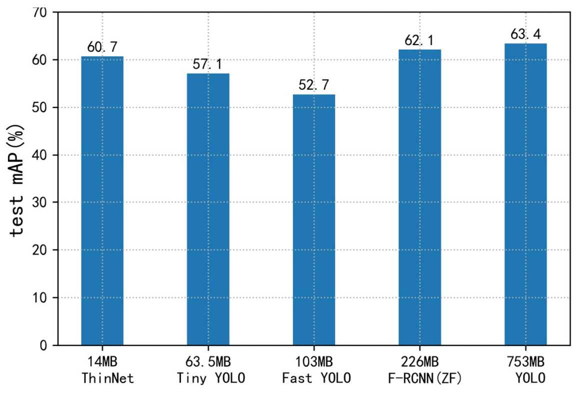

Fig.5. Tinier module(right).ThinNet comparison to diffierent models in terms of accuracy and model size

C. Detection Results on Pascal VOC

ThinNet are compared to several well-known small detection models in terms of computation cost and model size in this section. In order to make a fair comparison, we select the lightweight models from the-state-of-the-arts. The baseline methods are: Faster-RCNN(ZF) [2](the fast ZF model is used and is referred to as F-RCNN(ZF)), Fast YOLO [8], YOLO [8], Tiny YOLO [12]. F-RCNN(ZF) [2] has 5 convolutional layers and 3 fully connected layers, which is the smallest model used in Faster RCNN. Fast YOLO and YOLO are the smallest model and the model with highest accuracy in YOLOv1 [8], respectively. In addition to these, we also compared ThinNet with Tiny YOLO, which is the fastest network in Yolov2 [12]. *Table III* shows the comparative performance of ThinNet versus above small detection models, measured by AP (average precision) values of each object category. mAP values are also computed. Combining *Table III* and *Fig. 5*, it is clear that the overall performance of our ThinNet model is better than the other networks. On the VOC 2007 test set, ThinNet scores 60.7%, closer to F-RCNN(ZF) while its model size is 16 times smaller than ZF net. Comparing ThinNet with Tiny YOLO and

TABLE III. PASCAL VOC 2007 TEST DETECTION RESULTS COMPARISON USING DIFFERENT NETWORK ARCHITECTURES

Model	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
F-RCNN(ZF) [2]	62.1	66.8	67.4	54.5	46.6	29.5	73.8	67.2	84.4	37.9	51.7	78.3	76.9	87.2	71.0	59.9	34.9	48.4	71.2	81.5	52.5
Fast YOLO [8]	52.7	59.7	67.7	39.9	33.5	13.6	62.6	62.0	72.5	25.6	46.7	59.8	67.4	77.3	66	55.9	21.3	45.9	57.3	70.8	47.9
YOLOv1 [8]	63.4	73.4	72.3	47.5	46.5	19.3	77.9	75.9	79.6	38.9	56.4	74.9	75.9	86.6	79.7	68.0	30.7	53.4	72.2	81.8	56.7
Tiny-YOLO [12]	57.1	64.0	74.7	48.7	41.8	18.2	70.1	69.5	72,7	33.4	53.9	58.5	63.6	73.8	71.4	61.6	25.6	56.1	52.8	72.9	60.1
ThinNet	60.7	64.3	75.6	55.8	40.9	29.6	767.3	73.2	74.7	43.6	70.0	48.6	71.2	77.7	77.4	66.5	32.0	64.3	56.9	68.8	59.8

Fast YOLO, our ThinNet is more accurate with only 14 MB. *Fig. 6* show some detection examples on VOC2007 test set.

V. Conclusion

We have proposed a novel network architecture named ThinNet for object detection tasks. This new architecture is mainly based on the Front module and Tinier module to improve the quality with less parameters. The advantage of our model is that it can be de deployed on some platform with limited memory and computation. Experimental evaluations on the publicly available ImageNet classification and PASCAL VOC detection datasets demonstrate that the presented network model outperform than several popular models with smaller model size.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China under Grant 61672286

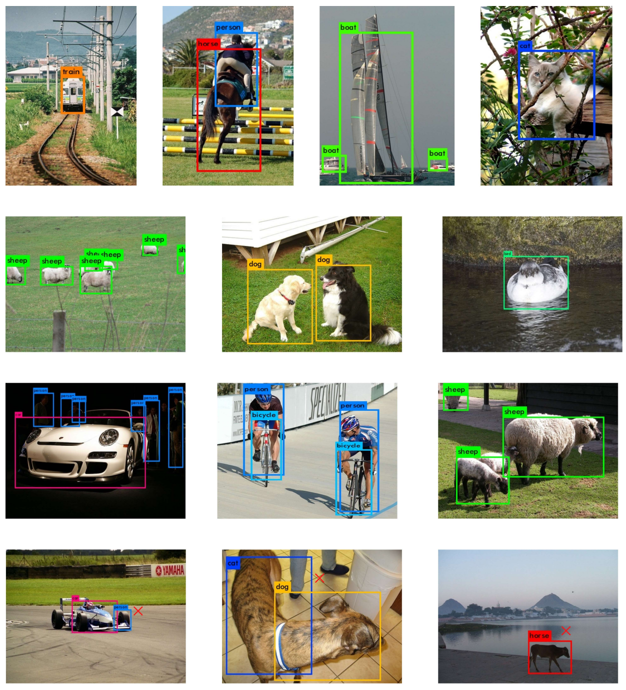

Fig.6. Selected examples of object detection results on PASCAL VOC2007 test set using the ThinNet.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in Neural Information Processing Systems, 2012, pp. 1097-1105.

[2] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 2017.

[3] E. L. Denton, W. Zaremba, J. Bruna, Y. Lecun, and R. Fergus, “Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation,” in Neural Information Processing Systems, 2014, pp. 1269-1277.

[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in Computer Vision and Pattern Recognition, 2016, pp. 770-778.

[5] G. Huang, Z. Liu, K. Q. Weinberger, and L. V. Der Maaten, “Densely Connected Convolutional Networks,” in Computer Vision and Pattern Recognition, 2017.

[6] K. Simonyan, and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in International Conference on Learning Representations, 2015.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in Computer Vision and Pattern Recognition, 2015, pp. 1-9.

[8] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in Computer Vision and Pattern Recognition, 2016, pp. 779-788.

[9] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” in Neural Information Processing Systems, 2015, pp. 1135-1143.

[10] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” in International Conference on Learning Representations, 2016.

[11] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” in Computer Vision and Pattern Recognition, 2014, pp. 580-587.

[12] J. Redmon, and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” in Computer Vision and Pattern Recognition, 2017.

[13] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,” in International Conference on Learning Representations, 2017.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, “SSD: Single Shot MultiBox Detector,” in European Conference on Computer Vision, 2016, pp. 21-37.

[15] F. Yu, and V. Koltun, “Multi-Scale Context Aggregation by Dilated Convolutions,” in International Conference on Learning Representations, 2016.

[16] A. G, Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” in Computer Vision and Pattern Recognition, 2017.

[17] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. S. Bernstein, “ImageNet Large Scale Visual Recognition Challenge,” International Journal of Computer Vision, vol. 115, no. 3, pp. 211-252, 2015.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, “The Pascal Visual Object Classes (VOC) Challenge,” International Journal of Computer Vision, vol. 88, no. 2, pp. 303-338, 2010.

[19] R. B. Girshick, “Fast R-CNN,” in International Conference on Computer Vision, 2015, pp. 1440-1448.

[20] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning Efficient Convolutional Networks through Network Slimming,” in International Conference on Computer Vision, 2017.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in Computer Vision and Pattern Recognition, 2016, pp. 2818-2826.

[22] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” in National Conference on Artificial Intelligence, 2016, pp. 4278-4284.

[23] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization,” in International Conference on Computer Vision, 2017, pp. 618-626.