Paper:

Real-Time Hand Detection Based on YOLOv3

Yihui Xie, Zhongjian Dai, Zhiyang Jia, Yaping Dai*

School of Automation, Beijing Institute of Technology
5 Zhongguancun South Street, Haidian District, Beijing 100081, P .R. China
E-mail: daiyaping@bit.edu.cn

Abstract. In this paper, a rapid detection method of human hand is proposed, which can determine the position of human hand in real time. Since YOLOv3 shows good accuracy in the field of object detection, a detection method based on YOLOv3 is proposed. Because of the larger model after training and the high demand for the computing power of the platform, the model is difficult to be applied to embedded devices. In order to solve the problem of large model, a new network slimming method is proposed in this paper. The method combines channel pruning and layer pruning to compress the model, and finally the precision is picked up by fine-tuning. The sparsity training step for the model is required before the pruning operation and this step can be applied to any typical CNNs or fully connected networks. The experimental results demonstrate that the mAP (Mean Average Precision) reaches 0.76, and the inference speed reaches 9.0 ms. Therefore, the proposed algorithm is capable of real-time detection.

Keywords: Object Detection, YOLOv3, Hand Detection, Network Slimming.

1. INTRODUCTION

With the dramatic improvement in computing power, a large number of papers on deep neural networks have been published. Especially in 2012, the AlexNet network [1] built by hinton's team had performed well in the ImageNet competition. Since then convolutional neural network (CNN) models have received widespread attention, and researchers have begun to focus on how to apply deep learning to object detection. Currently, the mainstream object detection algorithm is based on the deep learning model, and it can be primarily divided into two families: (1) two-stage detectors that divide the detection problem into two stages,such as R-CNN [2], Fast R-CNN [3], (2) one-stage detectors, such as YOLO [4], SSD [5], YOLOv2 [6], YOLOv3 [7] and CornerNet [8]. Two-stage detectors first generates a series of candidate boxes for the sample, followed by region classifiers to classify the samples. One-stage detectors directly produces the category probability and position coordinate values of the object without the region classification step. Although one-stage detectors are better than two-stage detectors in detection speed, they are not good enough in detection effect and accuracy.

The object detection algorithm was mainly achieved by sliding window and region detection before the YOLO algorithm. The network input of the YOLO algorithm is the whole picture, and the output layer is predicted based on the location and category of the bounding box. YOLOv2 added anchor box to YOLOv1, meanwhile YOLO2 introduced a method called passthrough layer that preserved some details in the feature map. YOLOv3 adopted multiscale prediction, and replaced softmax function with logistic regression and threshold. Because of the good performance of YOLOv3 algorithm in detection accuracy and detection speed, YOLOv3 is currently one of the best object detection algorithms in the world.

The performance of YOLOv3 is excellent. However, the trained model has some problems. For instance, the model size is very large, which is a burden on embedded devices. The convoluntion operations are extremely large on high-resolution images. Hence a large model can take several minutes to process a single picture on an embedded device, making it impractical to use in real-world applications. To solve these problems, [9] showed an acceleration method for the CNN model. This method directly removed convolutional cores that have little impact on CNN's accuracy, greatly reduced computational costs. But the accuracy of the slimmed-down model had also dropped dramatically. In 2017, [10] proposed a new method of network slimming. This method could be applied directly to the modern CNN network framework, introduced minimal costs in the training process and eliminated the need for other special hardware and software accelerators.

In this paper, we address the problem of real-time detection of the hand. YOLOv3 algorithm is used to detect the hand and determine the position of the hand. Then we reduce the size of the model and improve the speed of the model detection by compressing the network. Hand detection can be used on embedded devices, so the model needs to have a smaller size. In addition, to achieve real-time detection of hand, the equipment on the model size and detection speed are required.

The hand detection method we propose is based on the YOLOv3 algorithm.It mainly consists of three steps: sparsity training, pruning and fine-tuning. The sparsity training step is to achieve channel-level sparsity on the model. The resulting network is essentially a thin network, which can be quickly reasoned on the convolutional CNN platform. In the pruning step, we introduce the scale factor γ in the BN layer and then cut off the channel corresponding to the small scale factor. After completing these two steps, the accuracy of the model decreases. So in the final step, we fine-tune the network to improve the accuracy of the model.

2. HAND DETECTION METHOD

2.1. Bounding Box

2.1.1. Prior Box

YOLOv1 direct predicted width and height for detection box using logistic regression, so the effect was not good. Therefore, YOLOv2 used logical regression based on the variation value of the prior box. In this way, the learning difficulty of the network was reduced, and the overall accuracy had been improved. The technique of the prior box in YOLOv2 was still used in YOLOv3, and YOLOv3 used k-means to cluster the label boxes in the dataset, and got 9 boxes at the center point of the category as prior boxes. The prior box can only be affected by the width and height of the detection box.

2.1.2. Bounding Box Prediction

YOLOv3 predicted bounding boxes using dimension clusters as anchor boxes. In our experiment, the predictions correspond to:

$$b_x = \sigma(t_x) + c_x \tag{1}$$

$$b_y = \sigma(t_y) + c_y \tag{2}$$

$$b_w = p_w e^{t_w} \tag{3}$$

$$b_h = p_h e^{t_h} \tag{4}$$

where t_x, t_y, t_w, t_h denotes four coordinates predicted by the network. (c_x, c_y) denotes the offset between the cell and the top left corner of the image. p_w, p_h denotes width and height of the bounding box prior. σ is an activation function, and the sigmoid function is used in this paper.

The value should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. After this calculation, b_x also need to be multiplied by the sample rate to get the t_x and t_y.

2.2. Network Framework

2.2.1. Backbone

YOLOv3 used a network structure that evolved from Darknet-19 in YOLOv2 to Darknet-53 that deepened the number of network layers. Darknet-53 handled 78 graphs per second, much slower than Darknet-19, but much faster than the same accuracy ResNet[11]. YOLO's entire network drawed on the essence of Resnet, Densenet[12], and FPN[13], and it incorporated all the techniques that were currently most effective in object detection.Our structure is based on Darknet-53, as shown in **Fig 1**.

Unlike other frameworks, Darknet builds a network architecture that is generated by parsing cfg files, rather than stacking directly through code.

	Type	Filters	Size	Output
	Convolutional	32	3×3	416×416
	Convolutional	64	3×3×2	208×208
1×	Convolutional	32	1×1	
	Convolutional	64	3×3	
	Residual			208×208
	Convolutional	128	3×3/2	104×104
2×	Convolutional	64	1×1	
	Convolutional	128	3×3	
	Residual			104×104
	Convolutional	256	3×3/2	52×52
8×	Convolutional	128	1×1	
	Convolutional	256	3×3	
	Residual			52×52
	Convolutional	512	3×3/2	26×26
8×	Convolutional	256	1×1	
	Convolutional	512	3×3	
	Residual			26×26
	Convolutional	1024	3×3/2	13×13
4×	Convolutional	512	1×1	
	Convolutional	1024	3×3	
	Residual			13×13
	Avgpool		Global	
	Connected		1000	

Fig. 1. Network structure.

2.2.2. Structure

YOLOv3 has only convolution layers and controls the size of the output feature map by adjusting the convolutional steps. Therefore, the network has no special limit on the size of the input picture.The input and output of YOLOv3 is shown in **Fig 2**, and Yolov3 draws on the idea of pyramid feature maps, where small-size feature maps are used to detect large objects, while large-size feature maps detect small objects. In **Fig 2**, the width and height of the input image is 416 × 416, three feature maps with different scale are generated after convolution (CNN): (13 × 13), (26 × 26), (52 × 52). Next, we downsample the first feature map by 32 ×, downsampling the second feature map by 16 ×, and downsampling the third feature map by 8 ×. A total of predictions generated (13 × 13 + 26 × 26 + 52 × 52) × 3 = 10647 predictions. The prediction box is a series of prediction vectors generated by the network, each prediction vector has C+1+4 parameters, where C is the number of categories, 1 is a confidence level, and 4 is the prediction box position parameter (including the center point coordinates, the border width and

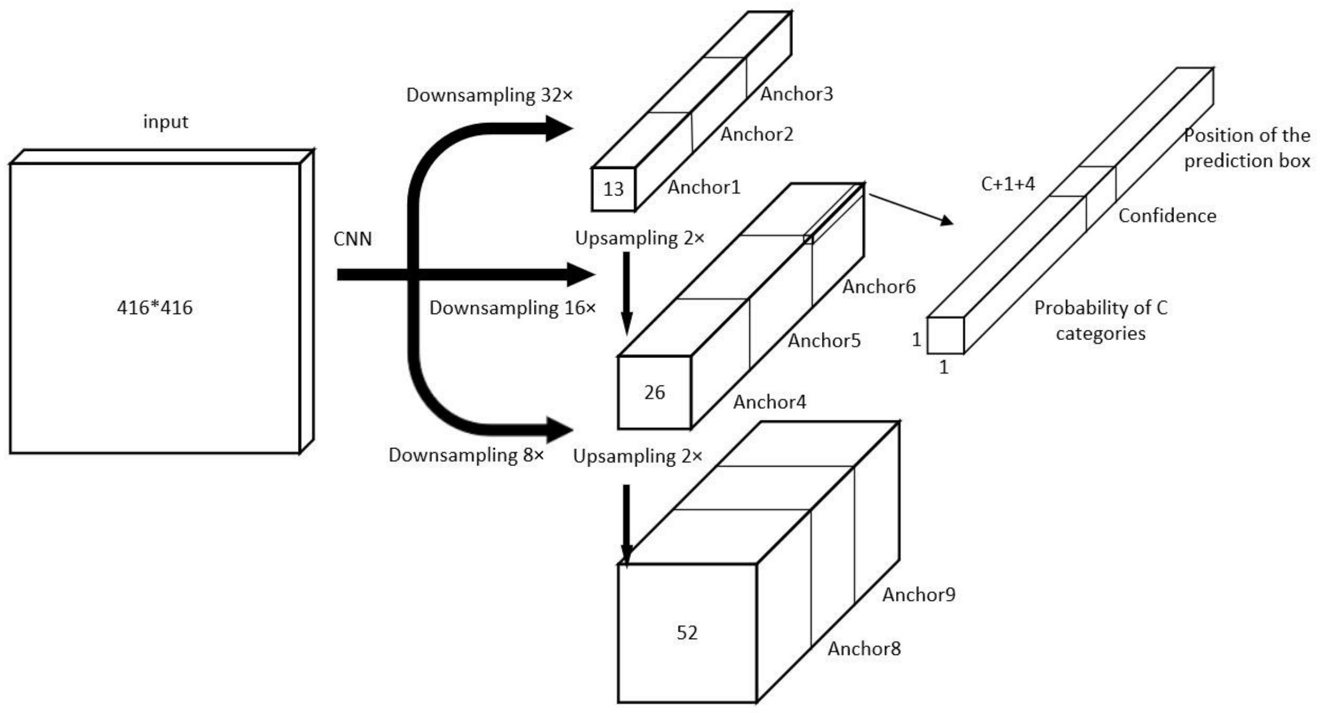

Fig. 2. Input and output of YOLOv3.

the border height). The role of the upsampling layer is to generate large-size images by interpolating and other methods. For instance, the upsampling layer transforms the image of 8 $\times$ 8 to 16 $\times$ 16 using the nearest neighbor interpolation algorithm, and the upsampling layer does not change the number of channels in the feature plot.

2.3. Pruning Optimization

YOLOv3 has reached a considerable level in the field of object detection. However, the model requires high computing power and storage space for the mobile platform, so it is impractical to apply YOLOv3 directly to embedded devices.

2.3.1. Batch normalization

The batch normalization (BN) layer is often used in CNN networks, usually behind the convolution layer and before the activation function. The BN layer reduces the overfitting of parameters during training.Each sample in the batch is associated with another sample in the batch. BN layer uses mini-batch statics to normalize convolutional features. BN layer performs the following transformation:

$$z = \gamma \times \frac{t-\mu}{\sqrt{\sigma^2+\varepsilon}} + \beta \tag{5}$$

where t and z are the input and output of a BN layer, μ and σ are mean and standard deviation values in a mini-batch, γ and β are trainable scale factor and bias.

Due to the influence of γ and β hyper parameters, the output data of each layer will not change much each time. This leads to a trend in the gradient decline of each layer.As a result, the BN layer accelerates convergence.

2.3.2. Pruning method

Pruning the model is a worthwhile approach to compressing the model. Paper [10] proposed a pruning method that used the scale factor as a measure of network contribution.In this method, the objective function is defined as:

$$L = \sum_{(x,y)} l(f(x,W),y) + \lambda \sum_{\gamma \in \Gamma} g(\gamma) \tag{6}$$

where (x,y) denotes the input of the network, W denotes the trainable weights, the first sum-term denotes the normal training loss of a CNN, $g(\cdot)$ is a penalty on the scaling factors, γ is a scaling factor, and λ balances the two terms.

Two pruning methods we focus on are as follows, (1) channel pruning [14] and (2) layer pruning. Pruning unimportant channels in convolutional layers is called channel pruning. Layer pruning strategy is derived from the previous channel pruning strategy. It sorts the mean of gamma on the each layer of shortcut, and cuts off the shortcut and convolution layers corresponding to the Minimum value of γ. In this paper we choose $g(s) = |s|$, and extracts the scaling factor from the network in the BN layer and adds the L1 regularization term on the original loss function. Thus the network realizes the sparse training and regains the sorting relationship of the scale factor. It prunes the network structure which scaling factor is trend to 0.

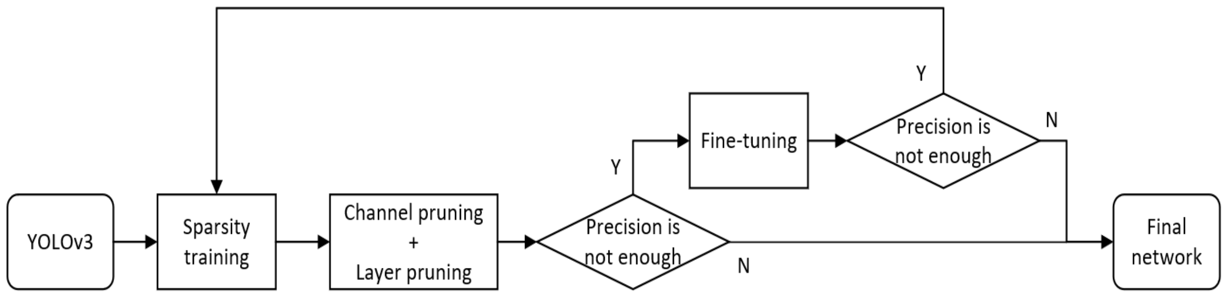

Fig. 3. The process of network optimization.

2.3.3. Network slimming process

Fig 3 shows the specific network optimization process. Since sparse networks can produce considerable acceleration of model size compression and inference,the first step is basic training and sparse training on the original network [15]. In sparse training, changing parameter scale in sparse training affects the accuracy of the model. To keep sparse models high-precision while achieving high sparseness, we need to choose a suitable scale parameter(The default is 0.001) and sparse strategy. In this paper, the scale parameter is always constant at 0.001 throughout the sparse training process, which increases the compression of the model.

After the first step mentioned above, the accuracy of the model has decreased. We determine the proportion and size of pruning based on the change in the γ of the bn layer. In this step, we combine channel pruning and layer pruning methods to prune the model. The effect of pruning depends on the sparse steps, and the different pruning strategies and threshold settings perform differently after the pruning. In general, pruning can damage model accuracy, at this time the model after the pruning needs to be finetuned to allow accuracy to pick up.

3. EXPERIMENTS

In order to enable the model to be applied on embedded devices, we used a dataset with a volume of 238.9MB. The dataset is a hand dataset introduced in paper [16], and it annotates a total of 13050 hand instances. In this paper, all experiments are implemented on a server with linux operating system. The hardware and software configuration of the server is shown in **Table 1**.

Because of computing power and model size, we set the batch size to 16, epoch to 100 during the basic training process. The resulting baseline model has a mAP of 0.80. In order to prepare for the pruning, we set the scale parameter to a constant value of 0.001, and set the epoch to 300 during the sparse process. Then we get a sparse model with a mAP value of 0.78. After getting the change in γ of the BN layer, many of the scaling factors in the model we get tend to be 0. Then we combine channel pruning and layer pruning methods to prune the model. We cut off the channel for the scaling factor near zero. At the same time, the average of each layer is sorted, taking the smallest layer cutting. In this process, it inevitably causes a decrease in the accuracy of the model. In this paper, we set the right parameters, select the right training strategy, and finally make the accuracy pick up by fine-tuning. **Table 2** shows the accuracy and speed of the model.

Table 1. Configuration.

Name	Configuration
CPU	Intel(R) Core(TM) i9-7900X,3.30GHz
GPU	NVIDIA 1080TI
System	Ubuntu16.04
GPU acceleration library	CUDA9.0,cuDNN7.0.5
Framework	Pytorch(backend TensorFlow)

Hand detection is an important part of service robot system, and it provides the basis for human-computer interaction. The results of our algorithm running on a personal computer are shown in the **Fig 4**. As can be seen from the visual interface, changing the shape of the hand does not affect the results of the detection.

4. CONCLUSION

In this paper, YOLOv3 is applied to the detection of human hands, and the experiment uses the hand dataset proposed in paper [16]. Speed and accuracy are two main purposes we focus on. Channel pruning and layer pruning are proposed for YOLOv3 network model, so that the model can be detected in real time on embedded devices. Before the pruning process, we compress the model with the sparsity training to prepare for the pruning. In addi-

Table 2. Experimental results.

Metric	Baseline	Sparsity training	Pruning Optimization	fine-tuning
mAP	0.80	0.78	0.69	**0.76**
Parameters (million)	62.6	62.6	8.5	**8.5**
Inference speed (ms)	18.2	18.2	9.0	**9.0**
Size of model (MB)	238.9	238.9	32.6	**32.6**

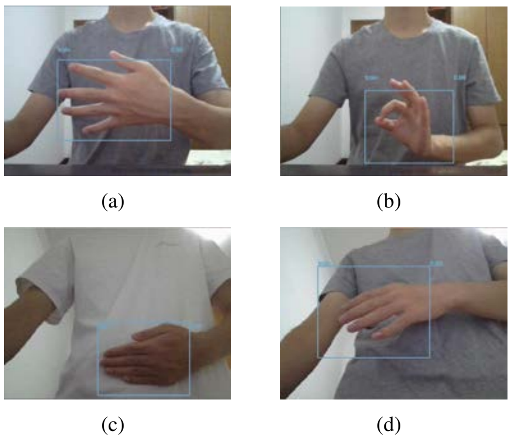

Fig. 4. The hand detection effect of different postures.

tion, we have introduced two pruning methods: (1) channel pruning and (2) layer pruning. Channel pruning is performed based on the γ parameter of the BN layer. Layer pruning is derived from channel pruning strategy, cutting off the shortcut layer of the network. We combine channel pruning and layer pruning to prune the YOLOv3 network in this paper. After the pruning process, the accuracy of the model decreases, so we fine-tune the network. Network slimming greatly reduces the input model, and improves the detection speed of the model. The mAP value of the detection algorithm can reache 0.76 and the size of the pruned model is 32.6MB. Considering future work, the detection accuracy of the proposed method is not high enough which need to be solved later.

Acknowledgements

This work was supported by the National Talents Foundation under Grant No. WQ20141100.

References:

[1] Krizhevsky A, Sutskever I, Hinton G E, et al. ImageNet Classification with Deep Convolutional Neural Networks. neural information processing systems, 2012: 1097-1105.

[2] Girshick R, Donahue J, Darrell T, et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. computer vision and pattern recognition, 2014: 580-587.

[3] Girshick R. Fast R-CNN. international conference on computer vision, 2015: 1440-1448.

[4] Redmon J, Divvala S K, Girshick R, et al. You Only Look Once: Unified, Real-Time Object Detection. computer vision and pattern recognition, 2016: 779-788.

[5] Liu W, Anguelov D, Erhan D, et al. SSD: Single Shot MultiBox Detector. european conference on computer vision, 2016: 21-37.

[6] Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger. computer vision and pattern recognition, 2017: 6517-6525.

[7] Redmon J, Farhadi A. YOLOv3: An Incremental Improvement. arXiv: Computer Vision and Pattern Recognition, 2018.

[8] Law H, Deng J. CornerNet: Detecting Objects as Paired Keypoints. european conference on computer vision, 2018: 765-781.

[9] Li H, Kadav A, Durdanovic I, et al. Pruning Filters for Efficient ConvNets. arXiv: Computer Vision and Pattern Recognition, 2016.

[10] Liu Z, Li J, Shen Z, et al. Learning Efficient Convolutional Networks through Network Slimming. international conference on computer vision, 2017: 2755-2763.

[11] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition. computer vision and pattern recognition, 2016: 770-778.

[12] Huang G, Liu Z, Der Maaten L V, et al. Densely Connected Convolutional Networks. computer vision and pattern recognition, 2017: 2261-2269.

[13] Lin T, Dollar P, Girshick R, et al. Feature Pyramid Networks for Object Detection. computer vision and pattern recognition, 2017: 936-944.

[14] He Y, Zhang X, Sun J, et al. Channel Pruning for Accelerating Very Deep Neural Networks. international conference on computer vision, 2017: 1398-1406.

[15] Han S, Pool J, Tran J, et al. Learning both weights and connections for efficient neural networks. neural information processing systems, 2015: 1135-1143.

[16] Mittal A, Zisserman A, Torr P H, et al. Hand detection using multiple proposals. british machine vision conference, 2011: 1-11.