Scaled-YOLOv4: Scaling Cross Stage Partial Network

Chien-Yao Wang[1], Alexey Bochkovskiy[2], and Hong-Yuan Mark Liao[1,3],
[1]Institute of Information Science, Academia Sinica, Taiwan
[2]Intel Intelligent Systems Lab
[3]Department of Computer Science and Information Engineering, Providence University, Taiwan
kinyiu@iis.sinica.edu.tw, alexeyab84@gmail.com, liao@iis.sinica.edu.tw

Abstract

We show that the YOLOv4 object detection neural network based on the CSP approach, scales both up and down and is applicable to small and large networks while maintaining optimal speed and accuracy. We propose a network scaling approach that modifies not only the depth, width, resolution, but also structure of the network. YOLOv4-large model achieves state-of-the-art results: 55.5% AP (73.4% AP$_{50}$) for the MS COCO dataset at a speed of ~16 FPS on Tesla V100, while with the test time augmentation, YOLOv4-large achieves 56.0% AP (73.3 AP$_{50}$). To the best of our knowledge, this is currently the highest accuracy on the COCO dataset among any published work. The YOLOv4-tiny model achieves 22.0% AP (42.0% AP$_{50}$) at a speed of ~443 FPS on RTX 2080Ti, while by using TensorRT, batch size = 4 and FP16-precision the YOLOv4-tiny achieves 1774 FPS.

1. Introduction

The deep learning-based object detection technique has many applications in our daily life. For example, medical image analysis, self-driving vehicles, business analytics, and face identification all rely on object detection. The computing facilities required for the above applications maybe cloud computing facilities, general GPU, IoT clusters, or single embedded device. In order to design an effective object detector, model scaling technique is very important, because it can make object detector achieve high accuracy and real-time inference on various types of devices.

The most common model scaling technique is to change the depth (number of layers in a neural network) and width (number of filters in a layer) of the backbone, and then train neural networks suitable for different devices. For example among the ResNet [11] series, ResNet-152 and ResNet-101 are often used in cloud server GPUs, ResNet-50 and ResNet-34 are often used in personal computer GPUs, and ResNet-18 and ResNet-10 can be used in low-end embed-
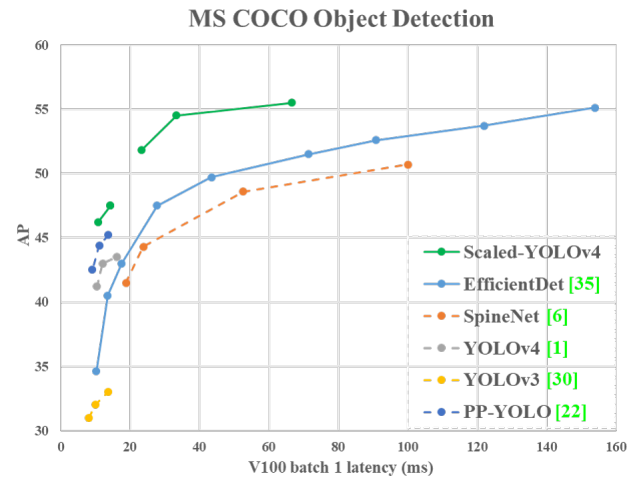

Figure 1: Comparison of the proposed scaled-YOLOv4 and other state-of-the-art object detectors. The dashed line means only latency of model inference, while the solid line include model inference and post-processing.

ded systems. In [2], Cai *et al.* try to develop techniques that can be applied to various device network architectures with only training once. They use techniques such as decoupling training and search and knowledge distillation to decouple and train several sub-nets, so that the entire network and sub-nets are capable of processing target tasks. Tan *et al.* [34] proposed using network architecture search (NAS) technique to perform compound scaling width, depth, and resolution on EfficientNet-B0. They use this initial network to search for the best convolutional neural network (CNN) architecture for a given amount of computation and set it as EfficientNet-B1, and then use linear scale-up technique to obtain EfficientNet-B2 to EfficientNet-B7. Radosavovic *et al.* [27] summarized and added constraints from the vast parameter search space AnyNet, and then designed RegNet to find optimal depth, bottleneck ratio, and width increase rate of a CNN. In addition, there are NAS and model scaling methods specifically proposed for object detection [6, 35].

Through analysis of state-of-the-art object detectors [1, 3, 6, 26, 35, 40, 44], we found that CSPDarknet53, which is the backbone of YOLOv4 [1], matches almost all optimal architecture features obtained by network architecture search technique [27]. Therefore, we developed model scaling technique based on YOLOv4 and proposed scaled-YOLOv4. The proposed scaled-YOLOv4 turned out with excellent performance, as illustrated in Figure 1. In the proposed scaled-YOLOv4, we discussed the upper and lower bounds of linear scaling up/down models, and respectively analyzed the issues that need to be paid attention to in model scaling for small models and large models. Thus, we are able to systematically develop YOLOv4-large and YOLOv4-tiny models. Scaled-YOLOv4 can achieve the best trade-off between speed and accuracy, and is able to perform real-time object detection on 15 FPS, 30 FPS, and 60 FPS movies, as well as embedded systems.

We summarize the contributions of this paper : (1) design a powerful model scaling method for small model, which can systematically balance the computation cost and memory bandwidth of a light CNN; (2) design a simple yet effective strategy for scaling a large object detector; (3) analyze the relations among all model scaling factors and then perform model scaling based on most advantageous group partitions; (4) experiments have confirmed that the FPN structure is inherently a once-for-all structure; and (5) we make use of the above methods to develop YOLOv4-tiny and YOLOv4-large.

2. Related work

2.1. Real-time object detection

Object detectors is mainly divided into one-stage object detectors [28, 29, 30, 21, 18, 24] and two-stage object detectors [10, 9, 31]. The output of one-stage object detector can be obtained after only one CNN operation. As for two-stage object detector, it usually feeds the high score region proposals obtained from the first-stage CNN to the second-stage CNN for final prediction. The inference time of one-stage object detectors and two-stage object detectors can be expressed as $T_{one} = T_{1^{st}}$ and $T_{two} = T_{1^{st}} + mT_{2^{nd}}$, where m is the number of region proposals whose confidence score is higher than a threshold. Today's popular real-time object detectors are almost one-stage object detectors. One-stage object detectors mainly have two kinds: anchor-based [30, 18] and anchor-free [7, 13, 14, 36]. Among all anchor-free approaches, CenterNet [46] is very popular because it does not require complicated post-processing, such as Non-Maximum Suppression (NMS). At present, the more accurate real-time one-stage object detectors are anchor-based EfficientDet [35], YOLOv4 [1], and PP-YOLO [22]. In this paper, we developed our model scaling methods based on YOLOv4 [1].

2.2. Model scaling

Traditional model scaling method is to change the depth of a model, that is to add more convolutional layers. For example, the VGGNet [32] designed by Simonyan *et al.* stacks additional convolutional layers in different stages, and also uses this concept to design VGG-11, VGG-13, VGG-16, and VGG-19 architectures. The subsequent methods generally follow the same methodology for model scaling. For the ResNet [11] proposed by He *et al.*, depth scaling can construct very deep networks, such as ResNet-50, ResNet-101, and ResNet-152. Later, Zagoruyko *et al.* [43] thought about the width of the network, and they changed the number of kernel of convolutional layer to realize scaling. They therefore design wide ResNet (WRN) , while maintaining the same accuracy. Although WRN has higher amount of parameters than ResNet, the inference speed is much faster. The subsequent DenseNet [12] and ResNeXt [41] also designed a compound scaling version that puts depth and width into consideration. As for image pyramid inference, it is a common way to perform augmentation at run time. It takes an input image and makes a variety of different resolution scaling, and then input these distinct pyramid combinations into a trained CNN. Finally, the network will integrate the multiple sets of outputs as its ultimate outcome. Redmon *et al.* [30] use the above concept to execute input image size scaling. They use higher input image resolution to perform fine-tune on a trained Darknet53, and the purpose of executing this step is to get higher accuracy.

In recent years, network architecture search (NAS) related research has been developed vigorously, and NAS-FPN [8] has searched for the combination path of feature pyramid. We can think of NAS-FPN as a model scaling technique which is mainly executed at the stage level. As for EfficientNet [34], it uses compound scaling search based on depth, width, and input size. The main design concept of EfficientDet [35] is to disassemble the modules with different functions of object detector, and then perform scaling on the image size, width, #BiFPN layers, and #box/class layer. Another design that uses NAS concept is SpineNet [6], which is mainly aimed at the overall architecture of fish-shaped object detector for network architecture search. This design concept can ultimately produce a scale-permuted structure. Another network with NAS design is RegNet [27], which mainly fixes the number of stage and input resolution, and integrates all parameters such as depth, width, bottleneck ratio and group width of each stage into depth, initial width, slope, quantize, bottleneck ratio, and group width. Finally, they use these six parameters to perform compound model scaling search. The above methods are all great work, but few of them analyze the relation between different parameters. In this paper, we will try to find a method for synergistic compound scaling based on the design requirements of object detection.

3. Principles of model scaling

After performing model scaling for the proposed object detector, the next step is to deal with the quantitative factors that will change, including the number of parameters with qualitative factors. These factors include model inference time, average precision, etc. The qualitative factors will have different gain effects depending on the equipment or database used. We will analyze and design for quantitative factors in 3.1. As for 3.2 and 3.3, we will design qualitative factors related to tiny object detector running on low-end device and high-end GPUs respectively.

3.1. General principle of model scaling

When designing the efficient model scaling methods, our main principle is that when the scale is up/down, the lower/higher the quantitative cost we want to increase/decrease, the better. In this section, we will show and analyze various general CNN models, and try to understand their quantitative costs when facing changes in (1) image size, (2) number of layers, and (3) number of channels. The CNNs we chose are ResNet, ResNext, and Darknet.

For the k-layer CNNs with b base layer channels, the computations of ResNet layer is $k*[\text{conv}(1 \times 1, b/4) \to \text{conv}(3 \times 3, b/4) \to \text{conv}(1 \times 1, b)]$, and that of ResNext layer is $k*[\text{conv}(1 \times 1, b/2) \to \text{gconv}(3 \times 3/32, b/2) \to \text{conv}(1 \times 1, b)]$. As for the Darknet layer, the amount of computation is $k*[\text{conv}(1 \times 1, b/2) \to \text{conv}(3 \times 3, b)]$. Let the scaling factors that can be used to adjust the image size, the number of layers, and the number of channels be α, β, and γ, respectively. When these scaling factors vary, the corresponding changes on FLOPs are summarized in Table 1.

Table 1: FLOPs of different computational layers with different model scalng factors.

Model	original	size α	depth β	width γ
Res layer	$r = 17whkb^2/16$	$\alpha^2 r$	βr	$\gamma^2 r$
ResX layer	$x = 137whkb^2/128$	$\alpha^2 x$	βx	$\gamma^2 x$
Dark layer	$d = 5whkb^2$	$\alpha^2 d$	βd	$\gamma^2 d$

It can be seen from Table 1 that the scaling size, depth, and width cause increase in the computation cost. They respectively show square, linear, and square increase.

The CSPNet [37] proposed by Wang *et al.* can be applied to various CNN architectures, while reducing the amount of parameters and computations. In addition, it also improves accuracy and reduces inference time. We apply it to ResNet, ResNeXt, and Darknet and observe the changes in the amount of computations, as shown in Table 2.

From the figures shown in Table 2, we observe that after converting the above CNNs to CSPNet, the new architecture can effectively reduce the amount of computations (FLOPs)

Table 2: FLOPs of different computational layers with/without CSP-ization.

Model	original	to CSP
Res layer	$17whkb^2/16$	$whb^2(3/4 + 13k/16)$
ResX layer	$137whkb^2/128$	$whb^2(3/4 + 73k/128)$
Dark layer	$5whkb^2$	$whb^2(3/4 + 5k/2)$

on ResNet, ResNeXt, and Darknet by 23.5%, 46.7%, and 50.0%, respectively. Therefore, we use CSP-ized models as the best model for performing model scaling.

3.2. Scaling Tiny Models for Low-End Devices

For low-end devices, the inference speed of a designed model is not only affected by the amount of computation and model size, but more importantly, the limitation of peripheral hardware resources must be considered. Therefore, when performing tiny model scaling, we must also consider factors such as memory bandwidth, memory access cost (MACs), and DRAM traffic. In order to take into account the above factors, our design must comply with the following principles:

Make the order of computations less than $O(whkb^2)$**:** Lightweight models are different from large models in that their parameter utilization efficiency must be higher in order to achieve the required accuracy with a small amount of computations. When performing model scaling, we hope the order of computation can be as low as possible. In Table 3, we analyze the network with efficient parameter utilization, such as the computation load of DenseNet and OS-ANet [15], where g means growth rate.

Table 3: FLOPs of Dense layer and OSA layer.

Model	FLOPs
Dense layer	$whgbk + whg^2k(k-1)/2$
OSA layer	$whbg + whg^2(k-1)$

For general CNNs, the relationship among g, b, and k listed in Table 3 is $k << g < b$. Therefore, the order of computation complexity of DenseNet is $O(whgbk)$, and that of OSANet is $O(max(whbg, whkg^2))$. The order of computation complexity of the above two is less than $O(whkb^2)$ of the ResNet series. Therefore, we design our tiny model with the help of OSANet, which has a smaller computation complexity.

Minimize/balance size of feature map: In order to get the best trade-off in terms of computing speed, we propose a new concept, which is to perform gradient truncation between computational block of the CSPOSANet. If we apply the original CSPNet design to the DenseNet or ResNet architectures, because the jth layer output of these two architectures is the integration of the 1st to $(j-1)^{th}$ layer

outputs, we must treat the entire computational block as a whole. Because the computational block of OSANet belongs to the PlainNet architecture, making CSPNet from any layer of a computational block can achieve the effect of gradient truncation. We use this feature to re-plan the b channels of the base layer and the kg channels generated by computational block, and split them into two paths with equal channel numbers, as shown in Table 4.

Table 4: Number of channel of OSANet, CSPOSANet, and CSPOSANet with partial in computational block (PCB).

layer ID	original	CSP	partial in CB
1	$b \to g$	$g \to g$	$g \to g$
2	$g \to g$	$g \to g$	$g \to g$
...	$g \to g$	$g \to g$	$g \to g$
k	$g \to g$	$g \to g$	$g \to g$
T	$(b+kg)$ $\to (b+kg)/2$	$kg \to kg$	$(b+kg)/2$ $\to (b+kg)/2$

When the number of channel is $b + kg$, if one wants to split these channels into two paths, the best partition is to divide it into two equal parts, i.e. $(b + kg)/2$. When we actually consider the bandwidth τ of the hardware, if software optimization is not considered, the best value is $ceil((b+kg)/2\tau) \times \tau$. The CSPOSANet we designed can dynamically adjust the channel allocation.

Maintain the same number of channels after convolution: For evaluating the computation cost of low-end device, we must also consider power consumption, and the biggest factor affecting power consumption is memory access cost (MAC). Usually the MAC calculation method for a convolution operation is as follows:

$$MAC = hw(C_{in} + C_{out}) + KC_{in}C_{out} \qquad (1)$$

where h, w, C_{in}, C_{out}, and K represent, respectively, the height and width of feature map, the channel number of input and output, and the kernel size of convolutional filter. By calculating geometric inequalities, we can derive the smallest MAC when $C_{in} = C_{out}$ [23].

Minimize Convolutional Input/Output (CIO): CIO [4] is an indicator that can measure the status of DRAM IO. Table 5 lists the CIO of OSA, CSP, and our designed CSPOSANet.

Table 5: CIO of OSANet, CSPOSANet, and CSPOSANet with PCB.

original	CSP	partial in CB
$bg + (k-1)g^2 + (b+kg)^2/2$	$kg^2 + (kg)^2$	$kg^2 + (b+kg)^2/4$

When $kg > b/2$, the proposed CSPOSANet can obtain the best CIO.

3.3. Scaling Large Models for High-End GPUs

Since we hope to improve the accuracy and maintain the real-time inference speed after scaling up the CNN model, we must find the best combination among the many scaling factors of object detector when performing compound scaling. Usually, we can adjust the scaling factors of an object detector's input, backbone, and neck. The potential scaling factors that can be adjusted are summarized as Table 6.

Table 6: Model scaling factors of different parts of object detectors.

Part	Scaling Factor
Input	$size^{input}$
Backbone	$width^{backbone}$, $depth^{backbone}$, #$stage^{backbone}$
Neck	$width^{neck}$, $depth^{neck}$, #$stage^{neck}$

The biggest difference between image classification and object detection is that the former only needs to identify the category of the largest component in an image, while the latter needs to predict the position and size of each object in an image. In one-stage object detector, the feature vector corresponding to each location is used to predict the category and size of an object at that location. The ability to better predict the size of an object basically depends on the receptive field of the feature vector. In the CNN architecture, the thing that is most directly related to receptive field is the stage, and the feature pyramid network (FPN) architecture tells us that higher stages are more suitable for predicting large objects. In Table 7, we illustrate the relations between receptive field and several parameters.

Table 7: Effect of receptive field caused by different model scaling factors.

Scaling factor	Effect of receptive field
$size^{input}$	no effect.
width	no effect.
depth	one more $k \times k$ conv layer, increases $k - 1$.
#stage	one more stage, receptive field doubled.

From Table 7, it is apparent that width scaling can be independently operated. When the input image size is increased, if one wants to have a better prediction effect for large objects, he/she must increase the depth or number of stages of the network. Among the parameters listed in Table 7, the compound of $\{size^{input}, \#stage\}$ turns out with the best impact. Therefore, when performing scaling up, we first perform compound scaling on $size^{input}$, #stage, and then according to real-time requirements, we further perform scaling on depth and width respectively.

4. Scaled-YOLOv4

In this section, we put our emphasis on designing scaled YOLOv4 for general GPUs, low-end GPUs, and high-end GPUs.

4.1. CSP-ized YOLOv4

YOLOv4 is designed for real-time object detection on general GPU. In this sub-section, we re-design YOLOv4 to YOLOv4-CSP to get the best speed/accuracy trade-off.

Backbone: In the design of CSPDarknet53, the computation of down-sampling convolution for cross-stage process is not included in a residual block. Therefore, we can deduce that the amount of computation of each CSPDarknet stage is $whb^2(9/4+3/4+5k/2)$. From the formula deduced above, we know that CSPDarknet stage will have a better computational advantage over Darknet stage only when $k > 1$ is satisfied. The number of residual layer owned by each stage in CSPDarknet53 is 1-2-8-8-4 respectively. In order to get a better speed/accuracy trade-off, we convert the first CSP stage into original Darknet residual layer.
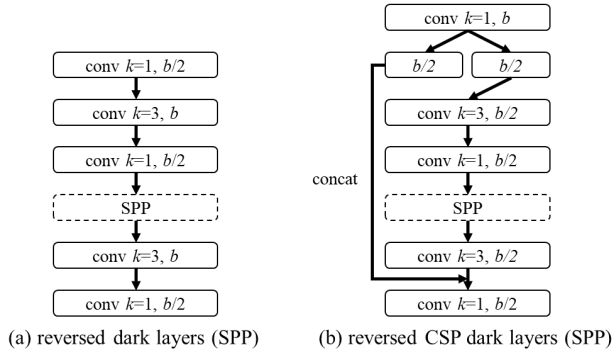

Figure 2: Computaional blocks of reversed Dark layer (SPP) and reversed CSP dark layers (SPP).

Neck: In order to effectively reduce the amount of computation, we CSP-ize the PAN [20] architecture in YOLOv4. The computation list of a PAN architecture is illustrated in Figure 2(a). It mainly integrates the features coming from different feature pyramids, and then passes through two sets of reversed Darknet residual layer without shortcut connections. After CSP-ization, the architecture of the new computation list is shown in Figure 2(b). This new update effectively cuts down 40% of computation.

SPP: The SPP module was originally inserted in the middle position of the first computation list group of the neck. Therefore, we also inserted SPP module in the middle position of the first computation list group of the CSPPAN.

4.2. YOLOv4-tiny

YOLOv4-tiny is designed for low-end GPU device, the design will follow principles mentioned in section 3.2.
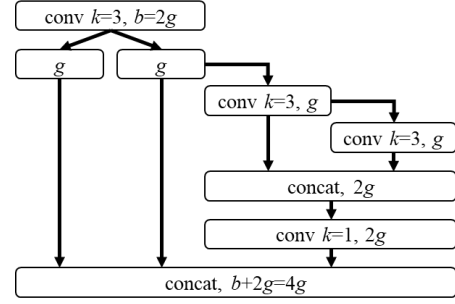
Figure 3: Computational block of YOLOv4-tiny.

We will use the CSPOSANet with PCB architecture to form the backbone of YOLOv4. We set $g = b/2$ as the growth rate and make it grow to $b/2 + kg = 2b$ at the end. Through calculation, we deduced $k = 3$, and its architecture is shown in Figure 3. As for the number of channels of each stage and the part of neck, we follow the design of YOLOv3-tiny.

4.3. YOLOv4-large

YOLOv4-large is designed for cloud GPU, the main purpose is to achieve high accuracy for object detection. We designed a fully CSP-ized model YOLOv4-P5 and scaling it up to YOLOv4-P6 and YOLOv4-P7.

Figure 4 shows the structure of YOLOv4-P5, YOLOv4-P6, and YOLOv4-P7. We designed to perform compound scaling on $size^{input}$, #stage. We set the depth scale of each stage to $2^{d_{s_i}}$, and d_s to [1, 3, 15, 15, 7, 7, 7]. Finally, we further use inference time as constraint to perform additional width scaling. Our experiments show that YOLOv4-P6 can reach real-time performance at 30 FPS video when the width scaling factor is equal to 1. For YOLOv4-P7, it can reach real-time performance at 16 FPS video when the width scaling factor is equal to 1.25.

5. Experiments

We use MSCOCO 2017 object detection dataset to verify the proposed scaled-YOLOv4. We do not use ImageNet pre-trained models, and all scaled-YOLOv4 models are trained from scratch and the adopted tool is SGD optimizer. The time used for training YOLOv4-tiny is 600 epochs, and that used for training YOLOv4-CSP is 300 epochs. As for YOLOv4-large, we execute 300 epochs first and then followed by using stronger data augmentation method to train 150 epochs. As for the Lagrangian multiplier of hyper-parameters, such as anchors of learning rate, the degree of different data augmentation methods, we use k-means and genetic algorithms to determine. All details related to hyper-parameters are elaborated in Appendix.

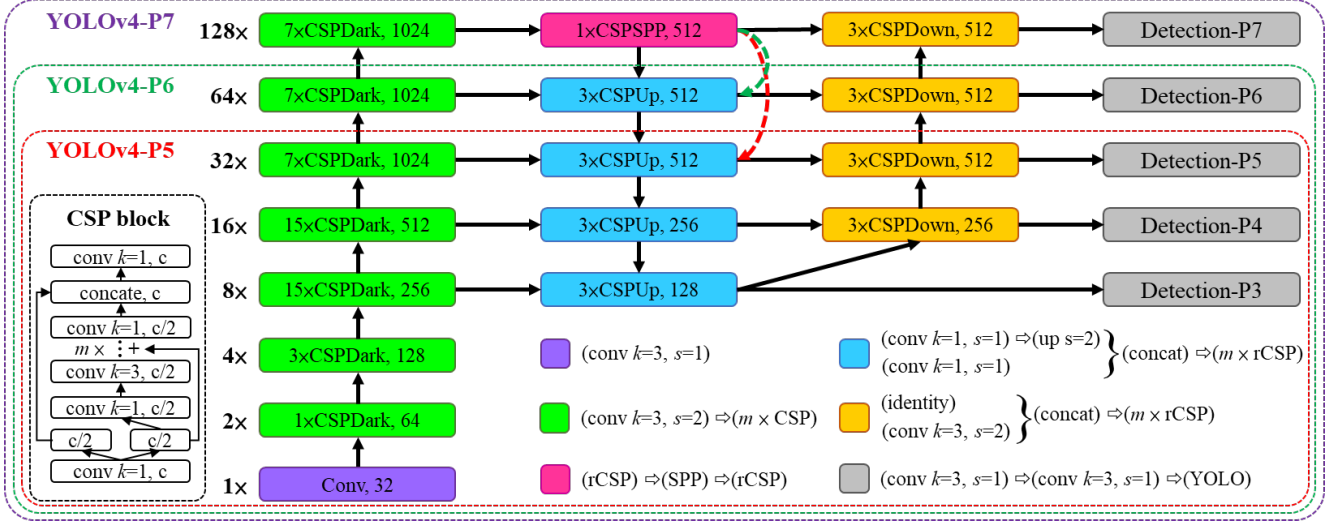Figure 4: Architecture of YOLOv4-large, including YOLOv4-P5, YOLOv4-P6, and YOLOv4-P7. The dashed arrow means replace the corresponding CSPUp block by CSPSPP block.

Table 8: Ablation study of CSP-ized models @608×608.

Backbone	Neck	Act.	#Param.	FLOPs	Batch 8 FPS	AP^{val}
D53	FPNSPP	Leaky	63M	142B	208	43.5%
D53	FPNSPP	Mish	63M	142B	196	45.3%
CD53s	**CFPNSPP**	Leaky	43M	97B	222	45.7%
CD53s	**CFPNSPP**	Mish	43M	97B	208	46.3%
D53	PANSPP	Leaky	78M	160B	196	46.5%
D53	PANSPP	Mish	78M	160B	185	46.9%
CD53s	**CPANSPP**	Leaky	53M	109B	208	46.9%
CD53s	**CPANSPP**	Mish	53M	109B	200	47.5%

Table 9: Ablation study of partial at different position in computational block.

Backbone	Neck	FLOPs	FPS_{TX2}	AP^{val}
tinyCD53s	tinyFPN	7.0B	30	22.2%
COSA-1x3x	tinyFPN	7.6B	38	**22.5%**
COSA-2x2x	tinyFPN	6.9B	42	22.0%
COSA-3x1x	tinyFPN	**6.3B**	**46**	21.2%

Table 10: Ablation study of training schedule with/without fine-tuning.

Model	scratch	finetune	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
YOLOv4-P5	300	-	50.5%	68.9%	55.2%
YOLOv4-P5	300	150	51.7%	70.3%	56.7%
YOLOv4-P6	300	-	53.4%	71.5%	58.5%
YOLOv4-P6	300	150	54.4%	72.7%	59.5%
YOLOv4-P7	300	-	54.6%	72.4%	59.7%
YOLOv4-P7	300	150	55.3%	73.3%	60.4%

5.1. Ablation study on CSP-ized model

In this sub-section, we will CSP-ize different models and analyze the impact of CSP-ization on the amount of parameters, computations, throughput, and average precision. We use Darknet53 (D53) as backbone and choose FPN with SPP (FPNSPP) and PAN with SPP (PANSPP) as necks to design ablation studies. In Table 8 we list the AP^{val} results after CSP-izing different DNN models. We use LeakyReLU (Leaky) and Mish activation function respectively to compare the amount of used parameters, computations, and throughput. Experiments are all conducted on COCO minval dataset and the resulting APs are shown in the last column of Table 8.

From the data listed in Table 8, it can be seen that the CSP-ized models have greatly reduced the amount of parameters and computations by 32%, and brought improvements in both Batch 8 throughput and AP. If one wants to maintain the same frame rate, he/she can add more layers or more advanced activation functions to the models after CSP-ization. From the figures shown in Table 8, we can see that both CD53s-CFPNSPP-Mish, and CD53s-CPANSPP-Leaky have the same batch 8 throughput with D53-FPNSPP-Leaky, but they respectively have 1% and 1.6% AP improvement with lower computing resources. From the above improvement figures, we can see the huge advantages brought by model CSP-ization. Therefore, we decided to use CD53s-CPANSPP-Mish, which results in the highest AP in Table 8 as the backbone of YOLOv4-CSP.

5.2. Ablation study on YOLOv4-tiny

In this sub-section, we design an experiment to show how flexible can be if one uses CSPNet with partial func-

Table 11: Comparison of state-of-the-art object detectors.

Method	Backbone	Size	FPS	AP	AP$_{50}$	AP$_{75}$	AP$_S$	AP$_M$	AP$_L$
EfficientDet-D0 [35]	EfficientNet-B0 [34]	512	97*	**34.6%**	**53.0%**	**37.1%**	**12.4%**	**39.0%**	**52.7%**
YOLOv4-CSP	**CD53s**	512	97/93*	**46.2%**	**64.8%**	**50.2%**	**24.6%**	**50.4%**	**61.9%**
EfficientDet-D1 [35]	EfficientNet-B1 [34]	640	74*	40.5%	59.1%	43.7%	18.3%	45.0%	57.5%
YOLOv4-CSP	**CD53s**	640	73/70*	**47.5%**	**66.2%**	**51.7%**	**28.2%**	**51.2%**	**59.8%**
YOLOv3-SPP [30]	D53 [30]	608	73	36.2%	60.6%	38.2%	20.6%	37.4%	46.1%
YOLOv3-SPP ours	D53 [30]	608	73	42.9%	62.4%	46.6%	25.9%	45.7%	52.4%
PP-YOLO [22]	R50-vd-DCN [22]	608	73	45.2%	65.2%	49.9%	26.3%	47.8%	57.2%
YOLOv4 [1]	CD53 [1]	608	62	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%
YOLOv4 ours	CD53 [1]	608	62	45.5%	64.1%	49.5%	27.0%	49.0%	56.7%
EfficientDet-D2 [35]	EfficientNet-B2 [34]	768	57*	43.0%	62.3%	46.2%	22.5%	47.0%	58.4%
RetinaNet [18]	S49s [6]	640	53	41.5%	60.5%	44.6%	23.3%	45.0%	58.0%
ASFF [19]	D53 [30]	608*	46	42.4%	63.0%	47.4%	25.5%	45.7%	52.3%
YOLOv4-P5	**CSP-P5**	896	43/41*	**51.8%**	**70.3%**	**56.6%**	**33.4%**	**55.7%**	**63.4%**
RetinaNet [18]	S49 [6]	640	42	44.3%	63.8%	47.6%	25.9%	47.7%	61.1%
EfficientDet-D3 [35]	EfficientNet-B3 [34]	896	36*	47.5%	66.2%	51.5%	27.9%	51.4%	62.0%
YOLOv4-P6	**CSP-P6**	1280	32/30*	**54.5%**	**72.6%**	**59.8%**	**36.8%**	**58.3%**	**65.9%**
ASFF[19]	D53 [30]	800*	29	43.9%	64.1%	49.2%	27.0%	46.6%	53.4%
SM-NAS: E2 [42]	-	800*600	25	40.0%	58.2%	43.4%	21.1%	42.4%	51.7%
EfficientDet-D4 [35]	EfficientNet-B4 [34]	1024	23*	49.7%	68.4%	53.9%	30.7%	53.2%	63.2%
SM-NAS: E3 [42]	-	800*600	20	42.8%	61.2%	46.5%	23.5%	45.5%	55.6%
RetinaNet [18]	S96 [6]	1024	19	48.6%	68.4%	52.5%	32.0%	52.3%	62.0%
ATSS [45]	R101 [11]	800*	18	43.6%	62.1%	47.4%	26.1%	47.0%	53.6%
YOLOv4-P7	**CSP-P7**	1536	17/16*	**55.5%**	**73.4%**	**60.8%**	**38.4%**	**59.4%**	**67.7%**
RDSNet [39]	R101 [11]	600	17	36.0%	55.2%	38.7%	17.4%	39.6%	49.7%
CenterMask [16]	R101-FPN [17]	-	15	44.0%	-	-	25.8%	46.8%	54.9%
EfficientDet-D5 [35]	EfficientNet-B5 [34]	1280	14*	51.5%	70.5%	56.7%	33.9%	54.7%	64.1%
ATSS [45]	R101-DCN [5]	800*	14	46.3%	64.7%	50.4%	27.7%	49.8%	58.4%
SABL [38]	R101 [11]	-	13	43.2%	62.0%	46.6%	25.7%	47.4%	53.9%
CenterMask [16]	V99-FPN [16]	-	13	46.5%	-	-	28.7%	48.9%	57.2%
EfficientDet-D6 [35]	EfficientNet-B6 [34]	1408	11*	52.6%	71.5%	57.2%	34.9%	56.0%	65.4%
RDSNet [39]	R101 [11]	800	11	38.1%	58.5%	40.8%	21.2%	41.5%	48.2%
RetinaNet [18]	S143 [6]	1280	10	50.7%	70.4%	54.9%	33.6%	53.9%	62.1%
SM-NAS: E5 [42]	-	1333*800	9.3	45.9%	64.6%	49.6%	27.1%	49.0%	58.0%
EfficientDet-D7 [35]	EfficientNet-B6 [34]	1536	8.2*	53.7%	72.4%	58.4%	35.8%	57.0%	66.3%
ATSS [45]	X-32x8d-101-DCN [5]	800*	7.0	47.7%	66.6%	52.1%	29.3%	50.8%	59.7%
EfficientDet-D7x [35]	EfficientNet-B7 [34]	1536	6.5*	55.1%	**74.3%**	59.9%	37.2%	57.9%	**68.0%**
TSD [33]	R101 [11]	-	5.3*	43.2%	64.0%	46.9%	24.0%	46.3%	55.8%

[1] FPS value with * means overall latency, which include model inference and post-processing.

[2] APs value with **bold font** means the value is higher than all method which has higher FPS.

tions in computational blocks. We also compare with CSP-Darknet53, in which we perform linear scaling down on width and depth. The results are shown in Table 9.

From the figures shown in Table 9, we can see that the designed PCB technique can make the model more flexible, because such a design can be adjusted according to actual needs. From the above results, we also confirmed that linear scaling down does have its limitation. It is apparent that when under limited operating conditions, the residual addition of tinyCD53s becomes the bottleneck of inference speed, because its frame rate is much lower than the COSA architecture with the same amount of computations. Meanwhile, we also see that the proposed COSA can get a higher AP. Therefore, we finally chose COSA-2x2x which received the best speed/accuracy trade-off in our experiment as the YOLOv4-tiny architecture.

5.3. Ablation study on YOLOv4-large

In Table 10 we show the AP obtained by YOLOv4 models in training from scratch and fine-tune stages.

5.4. Scaled-YOLOv4 for object detection

We compare with other real-time object detectors, and the results are shown in Table 11. The values marked in bold in the [AP, AP$_{50}$, AP$_{75}$, AP$_S$, AP$_M$, AP$_L$] items indicate that model is the best performer in the corresponding item. We can see that all scaled YOLOv4 models, including YOLOv4-CSP, YOLOv4-P5, YOLOv4-P6, YOLOv4-P7, are Pareto optimal on all indicators. When we compare YOLOv4-CSP with the same accuracy of EfficientDet-D3 (47.5% vs 47.5%), the inference speed is 1.9 times. When YOLOv4-P5 is compared with EfficientDet-D5 with

the same accuracy (51.8% vs 51.5%), the inference speed is 2.9 times. The situation is similar to the comparisons between YOLOv4-P6 vs EfficientDet-D7 (54.5% vs 53.7%) and YOLOv4-P7 vs EfficientDet-D7x (55.5% vs 55.1%). In both cases, YOLOv4-P6 and YOLOv4-P7 are, respectively, 3.7 times and 2.5 times faster in terms of inference speed. All scaled-YOLOv4 models reached state-of-the-art results.

The results of test-time augmentation (TTA) experiments of YOLOv4-large models are shown in Table 12. YOLOv4-P5, YOLOv4-P6, and YOLOv4-P7 gets 1.1%, 0.7%, and 0.5% higher AP, respectively, after TTA is applied.

Table 12: Results of YOLOv4-large models with test-time augmentation (TTA).

Model	AP	AP_{50}	AP_{75}
YOLOv4-P5 with TTA	52.9%	70.7%	58.3%
YOLOv4-P6 with TTA	55.2%	72.9%	60.5%
YOLOv4-P7 with TTA	56.0%	73.3%	61.4%

We then compare the performance of YOLOv4-tiny with that of other tiny object detectors, and the results are shown in Table 13. It is apparent that YOLOv4-tiny achieves the best performance in comparison with other tiny models.

Table 13: Comparison of state-of-the-art tiny models.

Model	Size	FPS_{1080ti}	FPS_{TX2}	AP
YOLOv4-tiny	416	**371**	**42**	21.7%
YOLOv4-tiny (3l)	320	252	41	**28.7%**
ThunderS146 [25]	320	248	-	23.6%
CSPPeleeRef [37]	320	205	41	23.5%
YOLOv3-tiny [30]	416	368	37	16.6%

Finally, we put YOLOv4-tiny on different embedded GPUs for testing, including Xavier AGX, Xavier NX, Jetson TX2, Jetson NANO. We also use TensorRT FP32 (FP16 if supported) for testing. All frame rates obtained by different models are listed in Table 14. It is apparent that YOLOv4-tiny can achieve real-time performance no matter which device is used. If we adopt FP16 and batch size 4 to test Xavier AGX and Xavier NX, the frame rate can reach 380 FPS and 199 FPS respectively. In addition, if one uses TensorRT FP16 to run YOLOv4-tiny on general GPU RTX 2080ti, when the batch size respectively equals to 1 and 4, the respective frame rate can reach 773 FPS and 1774 FPS, which is extremely fast.

Table 14: FPS of YOLOv4-tiny on embedded devices.

TensorRT.	FPS_{AGX}	FPS_{NX}	FPS_{TX2}	FPS_{NANO}
without	120	75	42	16
with	290	118	100	39

5.5. Scaled-YOLOv4 as naïve once-for-all model

In this sub-section, we design experiments to show that an FPN-like architecture is a naïve once-for-all model. Here we remove some stages of top-down path and detection branch of YOLOv4-P7. YOLOv4-P7\P7 and YOLOv4-P7\P7\P6 represent the model which has removed {P7} and {P7, P6} stages from the trained YOLOv4-P7. Figure 5 shows the AP difference between pruned models and original YOLOv4-P7 with different input resolution.
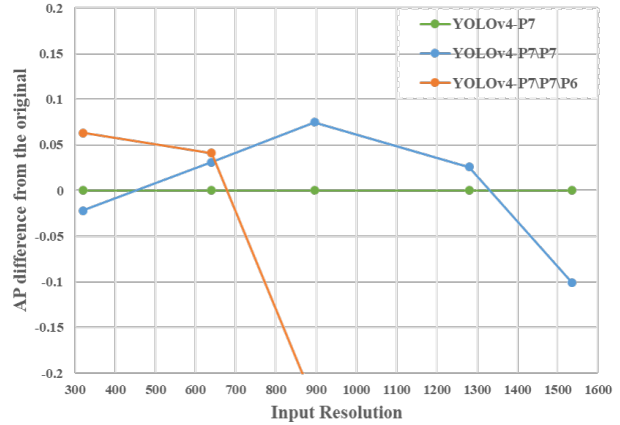

Figure 5: YOLOv4-P7 as "once-for-all" model.

We can find that YOLOv4-P7 has the best AP at high resolution, while YOLOv4-P7\P7 and YOLOv4-P7\P7\P6 have the best AP at middle and low resolution, respectively. This means that we can use sub-nets of FPN-like models to execute the object detection task well. Moreover, we can perform compound scale-down the model architectures and input size of an object detector to get the best performance.

6. Conclusions

We show that the YOLOv4 object detection neural network based on the CSP approach, scales both up and down and is applicable to small and large networks. So we achieve the highest accuracy 56.0% AP on test-dev COCO dataset for the model YOLOv4-large, extremely high speed 1774 FPS for the small model YOLOv4-tiny on RTX 2080Ti by using TensorRT-FP16, and optimal speed and accuracy for other YOLOv4 models.

7. Acknowledgements

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 2, 7

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 1

[3] Jiale Cao, Hisham Cholakkal, Rao Muhammad Anwer, Fahad Shahbaz Khan, Yanwei Pang, and Ling Shao. D2Det: Towards high quality object detection and instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11485–11494, 2020. 2

[4] Ping Chao, Chao-Yang Kao, Yu-Shan Ruan, Chien-Hsiang Huang, and Youn-Long Lin. HarDNet: A low memory traffic network. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019. 4

[5] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 764–773, 2017. 7

[6] Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V Le, and Xiaodan Song. SpineNet: Learning scale-permuted backbone for recognition and localization. *arXiv preprint arXiv:1912.05027*, 2019. 1, 2, 7

[7] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. CenterNet: Keypoint triplets for object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 6569–6578, 2019. 2

[8] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7036–7045, 2019. 2

[9] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 2

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014. 2

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 2, 7

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017. 2

[13] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018. 2

[14] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. CornerNet-Lite: Efficient keypoint based object detection. *arXiv preprint arXiv:1904.08900*, 2019. 2

[15] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. An energy and GPU-computation efficient backbone network for real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop)*, 2019. 3

[16] Youngwan Lee and Jongyoul Park. CenterMask: Real-time anchor-free instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 7

[17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017. 7

[18] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017. 2, 7

[19] Songtao Liu, Di Huang, and Yunhong Wang. Learning spatial fusion for single-shot object detection. *arXiv preprint arXiv:1911.09516*, 2019. 7

[20] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8759–8768, 2018. 5

[21] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 21–37, 2016. 2

[22] Xiang Long, Kaipeng Deng, Guanzhong Wang, Yang Zhang, Qingqing Dang, Yuan Gao, Hui Shen, Jianguo Ren, Shumin Han, Errui Ding, et al. PP-YOLO: An effective and efficient implementation of object detector. *arXiv preprint arXiv:2007.12099*, 2020. 2, 7

[23] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNetV2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 4

[24] Siyuan Qiao, Liang-Chieh Chen, and Alan Yuille. DetectoRS: Detecting objects with recursive feature pyramid and switchable atrous convolution. *arXiv preprint arXiv:2006.02334*, 2020. 2

[25] Zheng Qin, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng, and Jian Sun. ThunderNet: Towards real-time generic object detection. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019. 8

[26] Han Qiu, Yuchen Ma, Zeming Li, Songtao Liu, and Jian Sun. BorderDet: Border feature for dense object detection. In *Pro-

ceedings of the European Conference on Computer Vision (ECCV), pages 549–564. Springer, 2020. 2

[27] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10428–10436, 2020. 1, 2

[28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. 2

[29] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017. 2

[30] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 2, 7, 8

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 91–99, 2015. 2

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[33] Guanglu Song, Yu Liu, and Xiaogang Wang. Revisiting the sibling head in object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11563–11572, 2020. 7

[34] Mingxing Tan and Quoc V Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2019. 1, 2, 7

[35] Mingxing Tan, Ruoming Pang, and Quoc V Le. EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1, 2, 7

[36] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 9627–9636, 2019. 2

[37] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A new backbone that can enhance learning capability of CNN. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop)*, 2020. 3, 8

[38] Jiaqi Wang, Wenwei Zhang, Yuhang Cao, Kai Chen, Jiangmiao Pang, Tao Gong, Jianping Shi, Chen Change Loy, and Dahua Lin. Side-aware boundary localization for more precise object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 403–419. Springer, 2020. 7

[39] Shaoru Wang, Yongchao Gong, Junliang Xing, Lichao Huang, Chang Huang, and Weiming Hu. RDSNet: A new deep architecture for reciprocal object detection and instance segmentation. *arXiv preprint arXiv:1912.05070*, 2019. 7

[40] Xinjiang Wang, Shilong Zhang, Zhuoran Yu, Litong Feng, and Wayne Zhang. Scale-equalizing pyramid convolution for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13359–13368, 2020. 2

[41] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1492–1500, 2017. 2

[42] Lewei Yao, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. SM-NAS: Structural-to-modular neural architecture search for object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020. 7

[43] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 2

[44] Hongkai Zhang, Hong Chang, Bingpeng Ma, Naiyan Wang, and Xilin Chen. Dynamic R-CNN: Towards high quality object detection via dynamic training. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 260–275. Springer, 2020. 2

[45] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 7

[46] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019. 2