Scaled-YOLOv4

CSPDarknet53, which is the backbone of YOLOv4, matches almost all optimal architecture features obtained by network architecture search technique. The depth of CSPDarknet53, bottleneck ratio, width growth ratio between stages are 65, 1, and 2, respectively. Therefore, we developed model scaling technique based on YOLOv4 and proposed scaled-YOLOv4. The proposed scaled-YOLOv4 turned out with excellent performance.

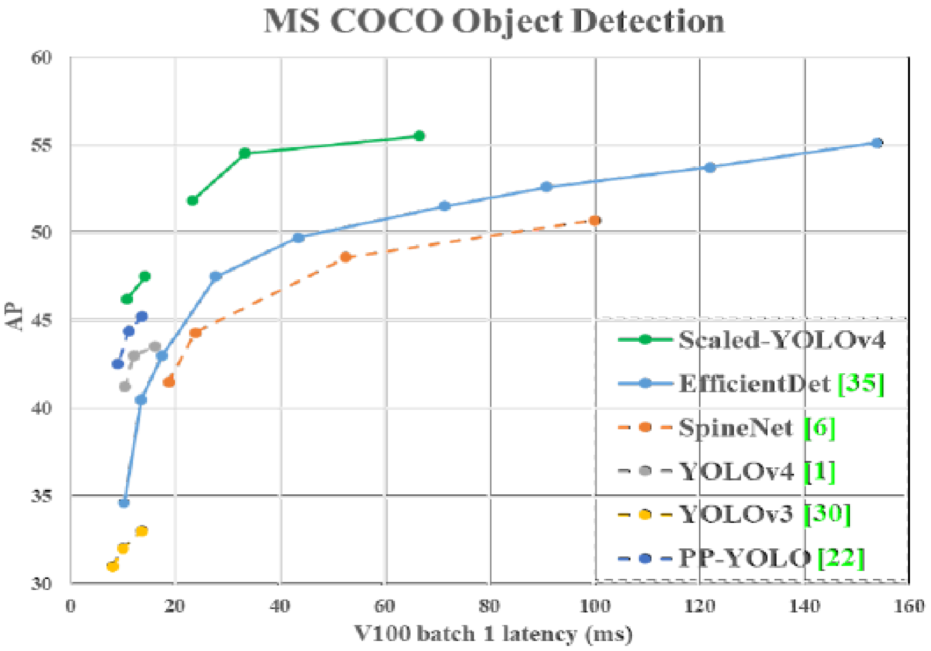

Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. The dashed line means only latency of model inference, while the solid line include model inference and post-processing.

The design procedure of scaled-YOLOv4 is as follows. 1) we re-design YOLOv4 and propose YOLOv4-CSP and 2) based on YOLOv4-CSP we developed scaled-YOLOv4. 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 16 FPS, 30 FPS, and 60 FPS movies, as well as embedded systems.

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.

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.

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.

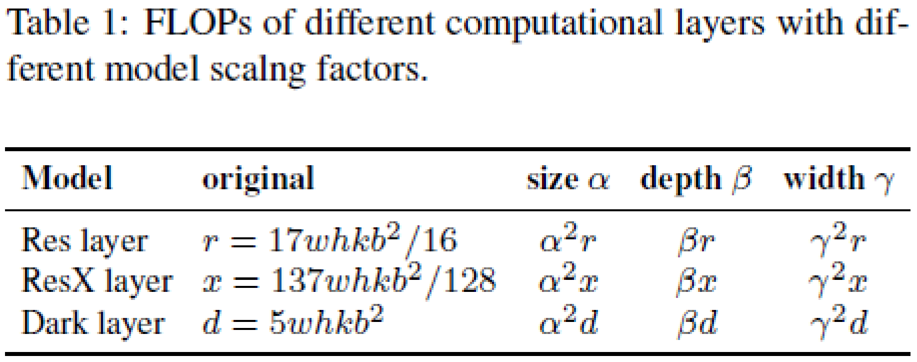

Table 1: FLOPs of different computational layers with different model scalng factors.

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	$\gamma^2 d$

The CSPNet 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.

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) 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.

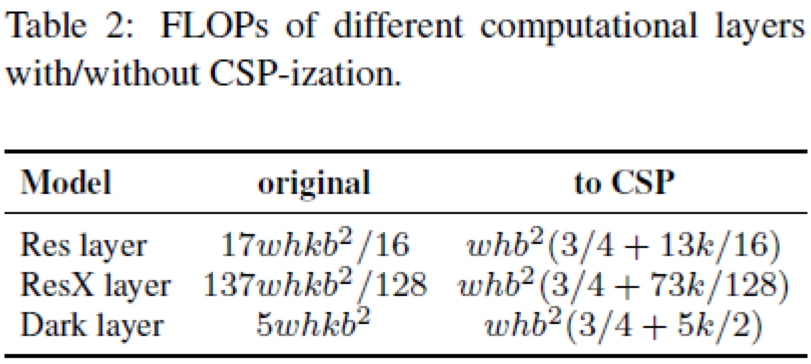

Table 2: FLOPs of different computational layers with/without CSP-ization.

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	$5whkb^2$	$whb^2(3/4 + 5k/2)$

Scaling Tiny Models for LowEnd 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.

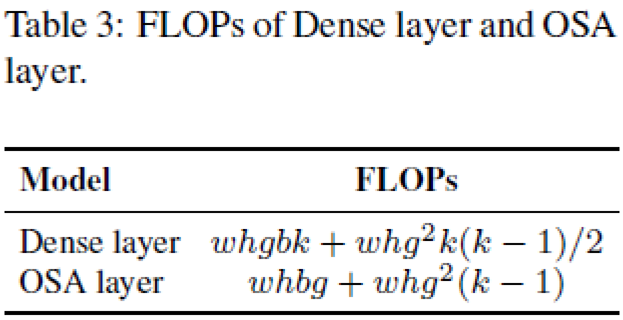

Table 3: FLOPs of Dense layer and OSA layer.

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

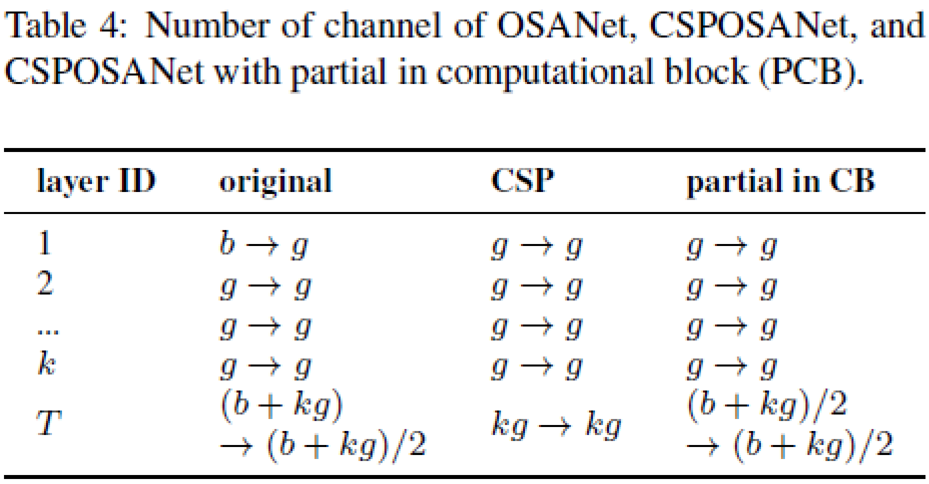

Table 4: Number of channel of OSANet, CSPOSANet, and CSPOSANet with partial in computational block (PCB).

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	$(b+kg)$ $\to (b+kg)/2$	$kg \to kg$	$(b+kg)/2$ $\to (b+kg)/2$

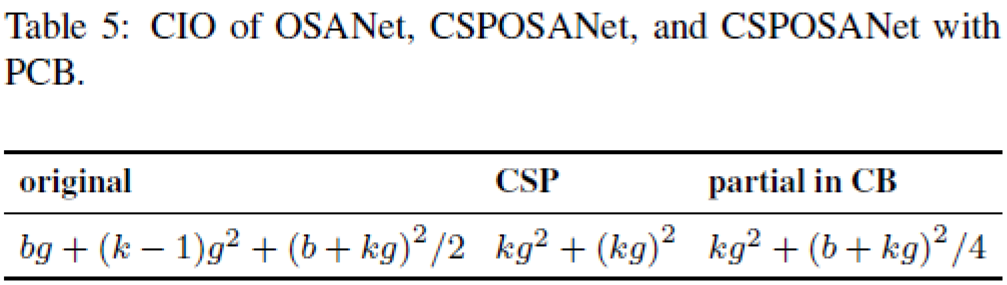

Table 5: CIO of OSANet, CSPOSANet, and CSPOSANet with PCB.

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

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.

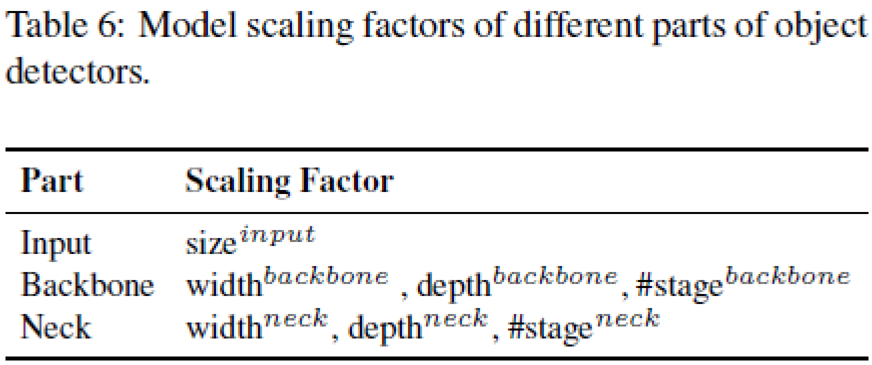

Table 6: Model scaling factors of different parts of object detectors.

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.

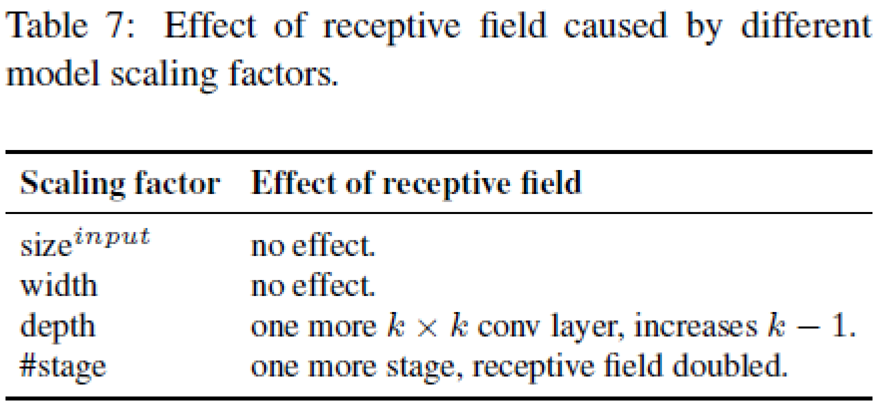

Table 7: Effect of receptive field caused by different model scaling factors.

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.

Scaled-YOLOv4

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 $whb^2(9/4+3/4+5k/2)$. In order to get a better speed/accuracy trade-off, we convert the first CSP stage into original Darknet residual layer.

- 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.

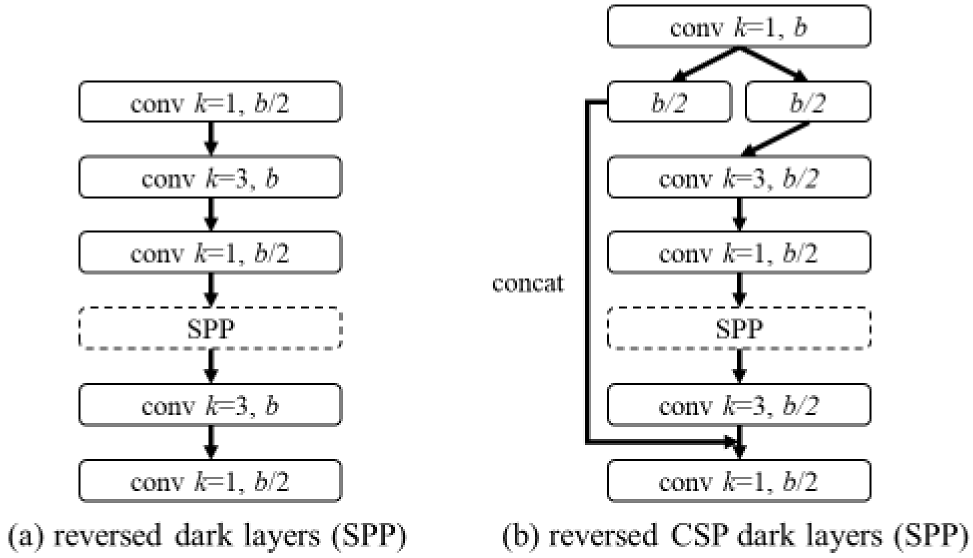

Figure 2: Computaional blocks of reversed Dark layer (SPP) and reversed CSP dark layers (SPP).

- 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.

2- YOLOv4-tiny:

YOLOv4-tiny is designed for low-end GPU device. 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. As for the number of channels of each stage and the part of neck, we follow the design of YOLOv3-tiny.

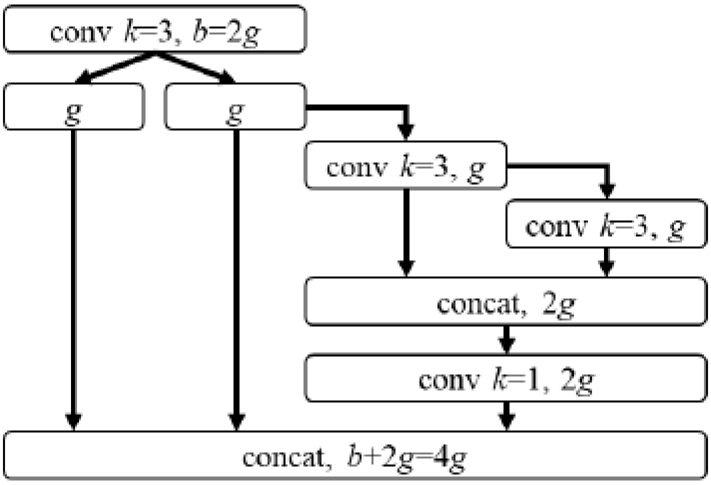

Figure 3: Computational block of YOLOv4-tiny.

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.

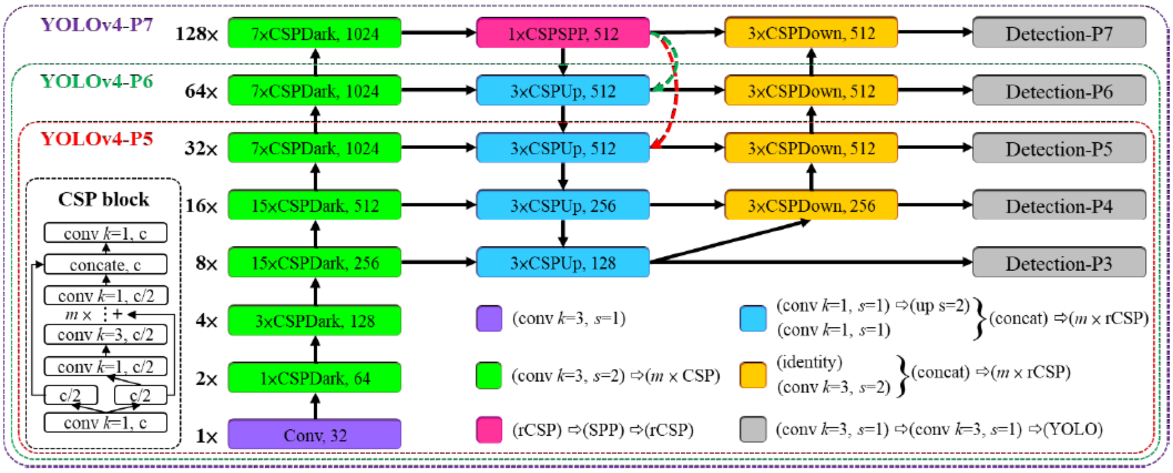

Figure 4: Architecture of YOLOv4-large, including YOLOv4-P5, YOLOv4-P6, and YOLOv4-P7. The dashed arrow means replace the corresponding CSPUp block by CSPSPP block.

Experiment

Dataset: MSCOCO 2017

We do not use ImageNet pre-trained models, and all scaled-YOLOv4 models are trained from scratch and the adopted tool is SGD optimizer.

Time for training YOLOv4-tiny: 600 epochs

Time for training YOLOv4-CSP: 300 epochs

Time for training YOLOv4-large: executed 300 epochs first and then followed by using stronger data augmentation method to train 150 epochs.

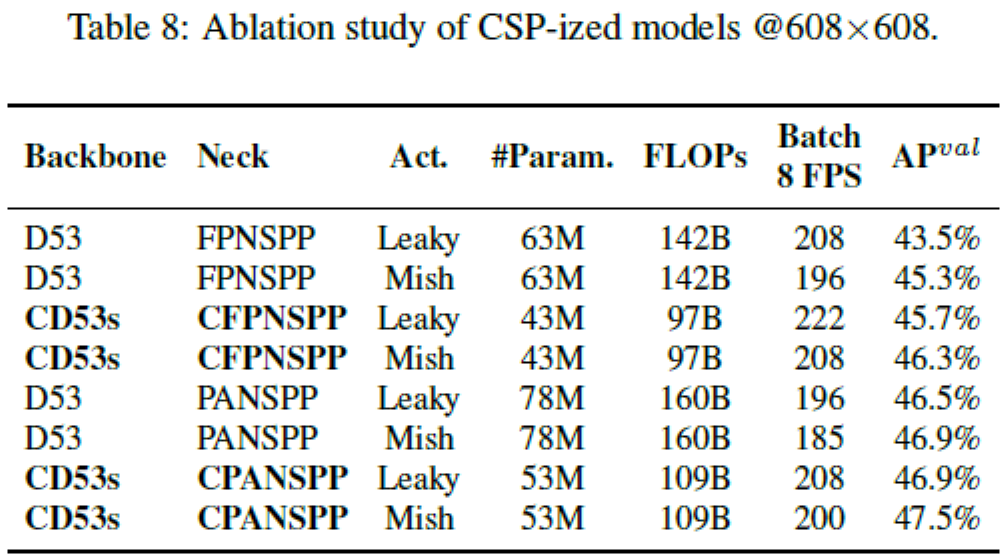

Table 8: Ablation study of CSP-ized models @608×608.

Backbone	Neck	Act.	#Param.	FLOPs	Batch 8 FPS	$\mathbf{AP}^{val}$
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%

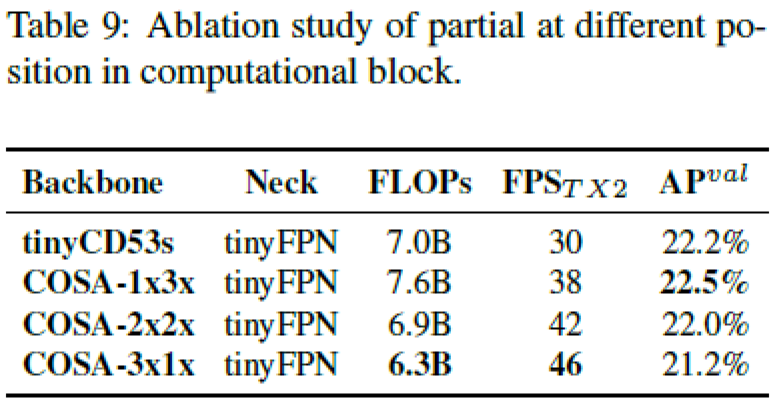

Table 9: Ablation study of partial at different position in computational block.

Backbone	Neck	FLOPs	$\mathbf{FPS}_{TX2}$	$\mathbf{AP}^{val}$
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%

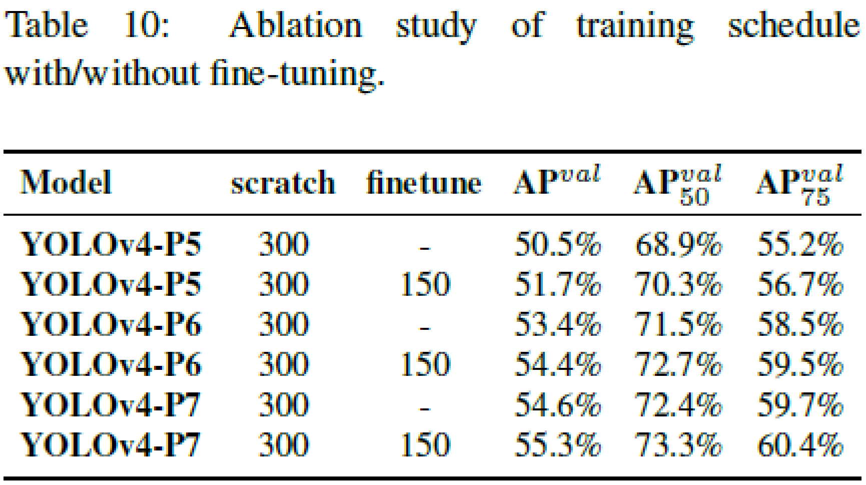

Table 10: Ablation study of training schedule with/without fine-tuning.

Model	scratch	finetune	$\mathbf{AP}^{val}$	$\mathbf{AP}^{val}_{50}$	$\mathbf{AP}^{val}_{75}$
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%

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%
ATSS [45]	X-64x4d-101-DCN [5]	800*	6.9	47.7%	66.5%	51.9%	29.7%	50.8%	59.4%
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%

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%

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%

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

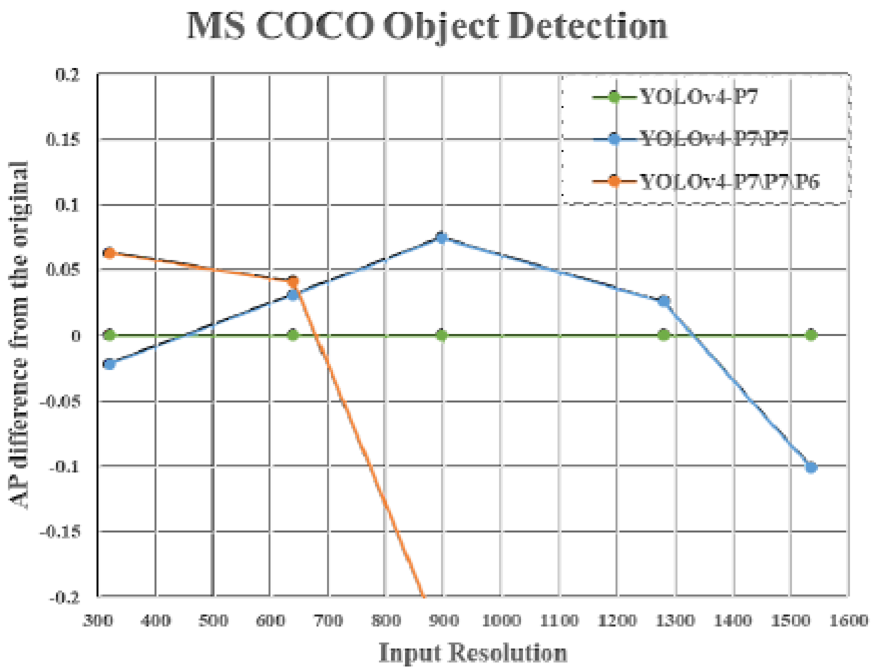

Figure 5: YOLOv4-P7 as “once-for-all” model.