PP-YOLOv2: A Practical Object Detector

Xin Huang[1,2], Xinxin Wang[1], Wenyu Lv[1], Xiaying Bai[1], Xiang Long[1]
Kaipeng Deng[1], Qingqing Dang[1], Shumin Han[1], Qiwen Liu[1], Xiaoguang Hu[1]
Dianhai Yu[1], Yanjun Ma[1], Osamu Yoshie[2]
koushin@toki.waseda.jp, {wangxinxin08, lvwenyu01, baixiaying, longxiang
dengkaipeng, dangqingqing, hanshumin, liuqiwen, huxiaoguang
yudianhai, mayanjun02}@baidu.com, yoshie@waseda.jp
[1]Baidu Inc. [2]Waseda University

Abstract

*Being effective and efficient is essential to an object detector for practical use. To meet these two concerns, we comprehensively evaluate a collection of existing refinements to improve the performance of PP-YOLO while almost keep the infer time unchanged. This paper will analyze a collection of refinements and empirically evaluate their impact on the final model performance through incremental ablation study. Things we tried that didn't work will also be discussed. By combining multiple effective refinements, we boost PP-YOLO's performance from 45.9% mAP to 49.5% mAP on COCO2017 test-dev. Since a significant margin of performance has been made, we present PP-YOLOv2. In terms of speed, PP-YOLOv2 runs in 68.9FPS at 640x640 input size. Paddle inference engine with TensorRT, FP16-precision and batch size = 1 further improves PP-YOLOv2's infer speed, which achieves 106.5 FPS. Such a performance surpasses existing object detectors with roughly the same amount of parameters (*i.e.*, YOLOv4-CSP, YOLOv5l). Besides, PP-YOLOv2 with ResNet101 achieves 50.3% mAP on COCO2017 test-dev. Source code is at* https://github.com/PaddlePaddle/PaddleDetection.

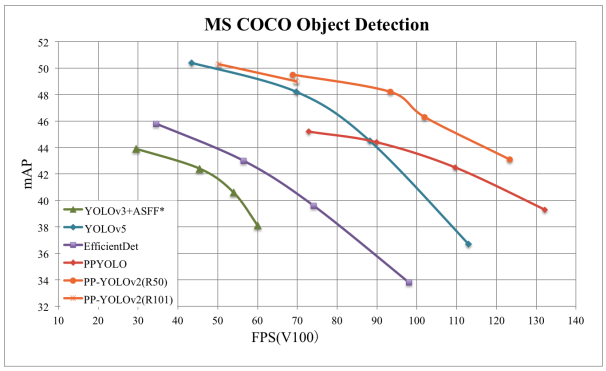

Figure 1. Comparison of the proposed PP-YOLOv2 and other object detectors. With a similar FPS, PP-YOLOv2 outperforms YOLOv5l by 1.3% mAP. Besides, when we replace PP-YOLOv2's backbone from ResNet50 to ResNet101, PP-YOLOv2 achieves comparable performance with YOLOv5x while it is 15.9% faster than YOLOv5x. The data is recorded in Table 2.

1. Introduction

Object detection is a critical component of various real-world applications such as self-driving cars, face recognition, and person re-identification. In recent years, the performance of object detectors has been rapidly improved with the rise of deep convolutional neural networks (CNNs) [23, 8, 10]. Although, recent works focus on novel detection pipeline (*i.e.*, Cascade RCNN [2] and HTC [3]), sophisticated network architecture design (DetectoRS [19] and CBNET [15]) push forward the state-of-the-art object detection approaches, YOLOv3 [20] is still one of the most widely used detector in industry. Because, in various practical applications, not only the computation resources are limited, but also the software support is insufficient. Without necessary technique support, two stage object detector(*e.g.* Faster RCNN [21], Cascade RCNN [2]) may excruciatingly slow. Meanwhile, a significant gap exists between the accuracy of YOLOv3 and two stage object detectors. Therefore, how to improve the effectiveness of YOLOv3 while maintaining the inference speed is an essential problem for practical use. To simultaneously satisfy two concerns, we add a bunch of refinements that almost not increase the infer time to improve the overall performance of the PP-YOLO [16]. To note that, although a huge number of approaches claim to improve object detector's accuracy independently, in practice, some methods are not effective when combined. Therefore, practical testing of combina-

tions of such tricks is required. We follow the incremental manner to evaluate their effectiveness one by one. All our experiments are implemented based on PaddlePaddle[1] [17].

In fact, this paper is more like a TECH REPORT, which tells you how to build PP-YOLOv2 step by step. Theoretical justification of the failure cases is also involved. To this end, we achieve a better balance between effectiveness (49.5% mAP) and efficiency (69 FPS), surpassing existing robust detectors with roughly the same amount of parameters such as YOLOv4-CSP [26] and YOLOv5l[2]. Hopefully, our experience in building PP-YOLOv2 can help developers and researchers to think deeper in implementing object detectors for practical applications.

2. Revisit PP-YOLO

In this section, we will perform the implementation of our baseline model specifically.

Pre-Processing. Apply Mixup Training [27] with a weight sampled from $Beta(\alpha, \beta)$ distribution where $\alpha = 1.5, \beta = 1.5$. Then, RandomColorDistortion, RandomExpand, RandCrop and RandomFlip are applied one by one with probability 0.5. Next, Normalize RGB channels by subtracting 0.485, 0.456, 0.406 and dividing by 0.229, 0.224, 0.225, respectively. Finally, The input size is evenly drawn from [320, 352, 384, 416, 448, 480, 512, 544, 576, 608].

Baseline Model. Our baseline model is PP-YOLO which is an enhanced version of YOLOv3. Specifically, it first replaces the backbone to ResNet50-vd[9]. After that a total of 10 tricks which can improve the performance of YOLOv3 almost without losing efficiency are added to YOLOv3 such as Deformable Conv [5], SSLD [4], CoordConv [12], DropBlock [6], SPP [7] and so on. The architecture of PP-YOLO is presented in the paper [16].

Training Schedule. On COCO *train2017*, the network is trained with stochastic gradient descent (SGD) for 500K iterations with a minibatch of 96 images distributed on 8 GPUs. The learning rate is linearly increased from 0 to 0.005 in 4K iterations, and it is divided by 10 at iteration 400K and 450K, respectively. Weight decay is set as 0.0005, and momentum is set as 0.9. Gradient clipping is adopted to stable the training procedure.

3. Selection of Refinements

Path Aggregation Network. Detecting objects at different scales is a fundamental challenge in object detection. In practice, a detection neck is developed for building high-level semantic feature maps at all scales. In PP-YOLO, FPN is adopted to compose bottom-up paths. Recently, several FPN variants have been proposed to enhance the ability of pyramid representation. For example, BiFPN [24], PAN [14], RFP [19] and so on. We follow the design of PAN to aggregate the top-down information. The detailed structure of PAN is shown in Fig. 2.

Mish Activation Function. Mish activation function [18] has been proved effective in many practical detectors, such as YOLOv4 and YOLOv5. They adopt the mish activation function in the backbone. However, we prefer to use pre-trained parameters because we have a powerful model which achieves 82.4% top-1 accuracy on ImageNet. To keep the backbone unchanged, we apply the mish activation function in the detection neck instead of the backbone.

Larger Input Size. Increasing the input size enlarges the area of objects. Thus, information of the objects on a small scale will be preserved easier than before. As a result, performance will be increased. However, a larger input size occupies more memory. To apply this trick, we need to decrease batch size. To be more specific, we reduce the batch size from 24 images per GPU to 12 images per GPU and expand the largest input size from 608 to 768. The input size is evenly drawn from [320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768].

IoU Aware Branch. In PP-YOLO, IoU aware loss is calculated in a soft weight format which is inconsistent with the original intention. Therefore, we apply a soft label format. Here is the IoU aware loss:

$$loss = -t * \log(\sigma(p)) - (1 - t) * \log(1 - \sigma(p)) \quad (1)$$

where t indicates the IoU between the anchor and its matched ground-truth bounding box, p is the raw output of IoU aware branch, $\sigma(\cdot)$ refers to the sigmoid activation function. To note that only positive samples' IoU aware loss is computed. By replacing the loss function, IoU aware branch works better than before.

4. Experiments

4.1. Dataset

COCO [11] is a widely used benchmark in the field of object detection. In this work, we train all our models on the COCO *train2017* which consists of 118k images across 80 classes. For evaluation, we evaluate our results on the COCO *minival* which consists of 5k testing images. Our evaluation metric also follows the standard COCO style mean Average Precision (mAP).

4.2. Ablation Studies

In this subsection, we present the effectiveness of each module in an incremental manner. Results are shown in Table 1, where infer time and FPS only consider the influence

[1] https://github.com/PaddlePaddle/Paddle
[2] https://github.com/ultralytics/yolov5

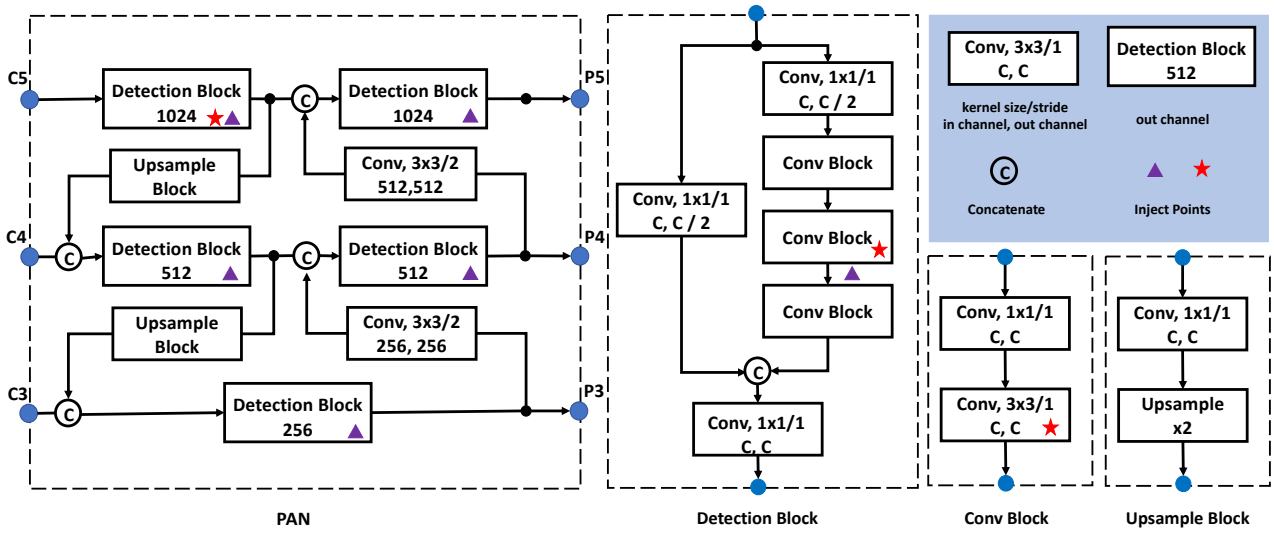

Figure 2. The architecture of PP-YOLOv2's detection neck.

	Methods	mAP	Parameters	GFLOPs	infer time	FPS
A	PP-YOLO	45.1	45 M	45.1	13.7 ms†	72.9
B	A + PAN + MISH	47.1	54 M	52.0	14.0 ms	71.4
C	B + input size 640	47.7	54 M	57.6	14.5 ms	68.9
D	C + Larger input size	48.3	54 M	57.6	14.5 ms	68.9
E	D + IoU Aware Branch	49.1	54 M	57.6	14.5 ms	68.9

Table 1. The ablation study of refinements on the MS-COCO minival split. "†" indicates the result includes bounding box decode time(1~2ms).

of the model in FP32-precision which does not include result decoder and NMS following YOLOv4[1].

A. First of all, we follow the original design of PP-YOLO to build our baseline. Since the heavy pre-processing on the CPU slows down the training, we decrease the images per GPU from 24 to 12. Reducing batch size drops mAP by 0.2%. Training settings are described in section 2 entirely.

A → B. The first refinement with a positive effect on PP-YOLO that we found was PAN. To stable the training process, we add several skip connections to our PAN module. The detailed structure of PAN is shown in Fig. 2. We can see that PAN and FPN are a group of symmetrical structures. When we perform it with Mish, it boosts the performance from 45.1% mAP to 47.1% mAP. Although model B is slightly slower than model A, such a significant gain promotes us to adopt PAN in our final model. For more details, please refer to our code.

B → C. Since the input size of YOLOv4 and YOLOv5 during evaluation is 640, we increase training and evaluation input size to 640 to build a fair comparison. The performance increases 0.6% mAP.

C → D. Keep increasing the input size should benefit more. However, it is impossible to use Larger Input Size and Larger Batch Size together. We train the model D with 12 images per GPU and Larger Input Size. It increases the mAP by 0.6% which brings more gains than Larger Batch Size. Therefore, we choose Larger Input Size in the final practice. The input size is evenly drawn from [320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768].

D → E. In the training phase, the modified IoU aware loss performs better than before. In the former version, the value of IoU aware loss will drop to 1e-5 in hundreds of iterations during training. After we modified the IoU aware loss, its value and the value of IoU loss are in the same order of magnitude, which is reasonable. After using this strategy, the mAP of model E increases to 49.1% without any loss of efficiency.

4.3. Comparison With Other State-of-the-Art Detectors

Comparison of the results on MS-COCO test split with other state-of-the-art object detectors is shown in Figure

Method	Backbone	Size	FPS (V100)		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
			w/o TRT	with TRT						
YOLOv3 + ASFF* [13]	Darknet-53	320	60	-	38.1%	57.4%	42.1%	16.1%	41.6%	53.6%
YOLOv3 + ASFF* [13]	Darknet-53	416	54	-	40.6%	60.6%	45.1%	20.3%	44.2%	54.1%
YOLOv3 + ASFF* [13]	Darknet-53	608	45.5	-	42.4%	63.0%	47.4%	25.5%	45.7%	52.3%
YOLOv3 + ASFF* [13]	Darknet-53	800	29.4	-	43.9%	64.1%	49.2%	27.0%	46.6%	53.4%
EfficientDet-D0 [24]	Efficient-B0	512	98.0^{+}	-	33.8%	52.2%	35.8%	12.0%	38.3%	51.2%
EfficientDet-D1 [24]	Efficient-B1	640	74.1^{+}	-	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
EfficientDet-D2 [24]	Efficient-B2	768	56.5^{+}	-	43.0%	62.3%	46.2%	22.5%	47.0%	58.4%
EfficientDet-D2 [24]	Efficient-B3	896	34.5^{+}	-	45.8%	65.0%	49.3%	26.6%	49.4%	59.8%
YOLOv4 [1]	CSPDarknet-53	416	96	164.0*	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4 [1]	CSPDarknet-53	512	83	138.4*	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4 [1]	CSPDarknet-53	608	62	105.5*	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%
YOLOv4-CSP [26]	Modified CSPDarknet53	512	97	-	46.2%	64.8%	50.2%	24.6%	50.4%	61.9%
YOLOv4-CSP [26]	Modified CSPDarknet53	640	73	-	47.5%	66.2%	51.7%	28.2%	51.2%	59.8%
YOLOv5s	-	640	113*	-	36.7%	55.4%	-	-	-	-
YOLOv5m	-	640	88.2*	-	44.5%	63.1%	-	-	-	-
YOLOv5l	-	640	69.8*	-	48.2%	66.9%	-	-	-	-
YOLOv5x	-	640	43.4*	-	50.4%	68.8%	-	-	-	-
PP-YOLO [16]	ResNet50-vd-dcn	320	132.2†	242.2†	39.3%	59.3%	42.7%	16.7%	41.4%	57.8%
PP-YOLO [16]	ResNet50-vd-dcn	416	109.6†	215.4†	42.5%	62.8%	46.5%	21.2%	45.2%	58.2%
PP-YOLO [16]	ResNet50-vd-dcn	512	89.9†	188.4†	44.4%	64.6%	48.8%	24.4%	47.1%	58.2%
PP-YOLO [16]	ResNet50-vd-dcn	608	72.9†	155.6†	45.9%	65.2%	49.9%	26.3%	47.8%	57.2%
PP-YOLOv2	ResNet50-vd-dcn	320	123.3	152.9	43.1%	61.7%	46.5%	19.7%	46.3%	61.8%
PP-YOLOv2	ResNet50-vd-dcn	416	102	145.1	46.3%	65.1%	50.3%	23.9%	50.2%	62.2%
PP-YOLOv2	ResNet50-vd-dcn	512	93.4	141.2	48.2%	67.1%	52.7%	27.7%	52.1%	62.1%
PP-YOLOv2	ResNet50-vd-dcn	608	72.1	109.9	49.2%	68.0%	54.1%	29.9%	52.8%	61.5%
PP-YOLOv2	ResNet50-vd-dcn	640	68.9	106.5	49.5%	68.2%	54.4%	30.7%	52.9%	61.2%
PP-YOLOv2	ResNet101-vd-dcn	512	69.8	116.8	49.0%	67.8%	53.8%	28.7%	53.0%	63.5%
PP-YOLOv2	ResNet101-vd-dcn	640	50.3	87.0	50.3%	69.0%	55.3%	31.6%	53.9%	62.4%

Table 2. Comparison of the speed and accuracy of different object detectors on the MS-COCO (test-dev 2017). We compare the results with batch size = 1, without tensorRT (w/o TRT) or with tensorRT(with TRT). Results marked by "+" are updated results from the corresponding official code base. Results marked by "*" are test in our environment using official code and model. "†" indicates the result includes bounding box decode time(1~2ms). The backbone of YOLOv5 has not been named yet, so we leave it blank.

1 and Table 2. We compare our method with YOLOv4-CSP and YOLOv5l because they have roughly the same amount of parameters as our model. It clearly shows that PP-YOLOv2 outperforms these two methods. With a similar FPS, PP-YOLOv2 outperforms YOLOv4-CSP by 2% mAP and surpasses YOLOv5l by 1.3% mAP. Besides, when we replace PP-YOLOv2's backbone from ResNet50 to ResNet101, PP-YOLOv2 achieves comparable performance with YOLOv5x while it is 15.9% faster than YOLOv5x. Therefore, we can draw a conclusion that compared with other state-of-the-art methods, our PP-YOLOv2 has certain advantages in the balance of speed and accuracy.

Moreover, PP-YOLOv2 is implemented based on PaddlePaddle. As a deep learning framework, PaddlePaddle not only supports model implementation but also pays attention to model deployment. With official support, adapting TensorRT for PP-YOLOv2 is much easier than other detectors. Specifically, the Paddle inference engine with TensorRT, FP16-precision, and batch size = 1 further improves PP-YOLOv2's infer speed. The speed-up ratios for PP-YOLOv2(R50) and PP-YOLOv2(R101) are 54.6% and 73%, respectively.

5. Things We Tried That Didn't Work

Since it takes about 80 hours for training PP-YOLO with 8 V100 GPUs on COCO *train2017*, we involve COCO *minitrain* [22] to speed up our analysis on ablation studies. COCO *minitrain* is a subset of the COCO *train2017*, containing 25K images. On COCO *minitrain*, the total iterations is 90K. We divide the learning rate by 10 at iteration 60k. Other settings are the same as training on COCO *train2017*.

We tried lots of stuff while we were working on PP-YOLOv2. Some of them have a positive effect on COCO *minitrain* while hinders the performance when training on COCO *train2017*. Due to the inconsistency, someone may doubt the experimental conclusion on COCO *minitrain*.The reason why we use COCO *minitrain* is that we want to seek refinements with universal features, which means that they should be useful on different scale datasets. It is also important to figure out the reason why they failed. Therefore, we discuss some of them in this section.

Cosine Learning Rate Decay. Different from linear step learning rate decay, cosine learning rate decay is exponen-

tially decaying the learning rate. Therefore, the change of the learning rate is smooth, which benefits the training process. We follow the formula in Bag of Tricks [9] to set learning rate at each epoch. Although cosine learning rate decay achieves better performance on COCO *minitrain*, it is sensitive to hyper-parameters such as initial learning rate, the number of warm up steps, and the ending learning rate. We tried several sets of hyper-parameters. However, we didn't observe a positive effect on COCO *train2017* eventually.

Backbone Parameter Freezing. When fine-tuning the ImageNet pre-trained parameters on downstream tasks, freezing parameters in the first two stages is a common practice. Since our pre-trained ResNet50-vd is much powerful than others(82.4% Top1 accuracy versus 79.3% Top1 accuracy), we are more motivated to adopt this strategy. On COCO *minitrain*, parameter freezing brings 1mAP gain, however, on COCO *train2017* it decreases mAP by 0.8% . A possible reason for the inconsistency phenomena was speculated to be the different sizes of the two training sets. COCO *minitrain* is a fifth of COCO *train2017*. The ability to generalization of parameters that are trained on a small dataset may be worse than pre-trained parameters.

SiLU Activation Function. We tried using SiLU [25] instead of Mish in detection neck. This increases 0.3% mAP on COCO *minitrain* but drops 0.5% mAP on COCO *train2017*. We are not sure about the reason.

6. Conclusions

This paper presents some updates to PP-YOLO, which forms a high-performance object detector called PP-YOLOv2. PP-YOLOv2 achieves a better balance between speed and accuracy than other famous detectors, such as YOLOv4 and YOLOv5. In this paper, we explore a bunch of tricks and show how to combine these tricks on the PP-YOLO detector and demonstrate their effectiveness. Moreover, with PaddlePaddle's official support, the gap between model development and production deployment is narrowed. We hope this paper can help developers and researchers get better performance in practical scenes.

7. Acknowledgements

This work was supported by the National Key Research and Development Project of China (2020AAA0103500).

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. 3, 4

[2] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018. 1

[3] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4974–4983, 2019. 1

[4] Cheng Cui, Ruoyu Guo, Yuning Du, Dongliang He, Fu Li, Zewu Wu, Qiwen Liu, Shilei Wen, Jizhou Huang, Xiaoguang Hu, Dianhai Yu, Errui Ding, and Yanjun Ma. Beyond self-supervision: A simple yet effective network distillation alternative to improve backbones, 2021. 2

[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*, pages 764–773, 2017. 2

[6] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. *arXiv preprint arXiv:1810.12890*, 2018. 2

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. 2

[8] 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*, pages 770–778, 2016. 1

[9] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019. 2, 5

[10] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 1

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2

[12] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *arXiv preprint arXiv:1807.03247*, 2018. 2

[13] Songtao Liu, Di Huang, and Yunhong Wang. Learning spatial fusion for single-shot object detection. *arXiv preprint arXiv:1911.09516*, 2019. 4

[14] 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*, pages 8759–8768, 2018. 2

[15] Yudong Liu, Yongtao Wang, Siwei Wang, TingTing Liang, Qijie Zhao, Zhi Tang, and Haibin Ling. Cbnet: A novel composite backbone network architecture for object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11653–11660, 2020. 1

[16] 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. 1, 2, 4

[17] Yanjun Ma, Dianhai Yu, Tian Wu, and Haifeng Wang. Paddlepaddle: An open-source deep learning platform from industrial practice. *Frontiers of Data and Domputing*, 1(1):105–115, 2019. 2

[18] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 4, 2019. 2

[19] 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. 1, 2

[20] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 1

[21] 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*, pages 91–99, 2015. 1

[22] Alexander Sheshkus, Anastasia Ingacheva, Vladimir Arlazarov, and Dmitry Nikolaev. Houghnet: neural network architecture for vanishing points detection. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 844–849. IEEE, 2019. 4

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[24] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020. 2, 4

[25] Aili Wang, Xin He, Pedram Ghamisi, and Yushi Chen. Lidar data classification using morphological profiles and convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters*, 15(5):774–778, 2018. 5

[26] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network, 2020. 2, 4

[27] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 2

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. 3, 4

[2] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018. 1

[3] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4974–4983, 2019. 1

[4] Cheng Cui, Ruoyu Guo, Yuning Du, Dongliang He, Fu Li, Zewu Wu, Qiwen Liu, Shilei Wen, Jizhou Huang, Xiaoguang Hu, Dianhai Yu, Errui Ding, and Yanjun Ma. Beyond self-supervision: A simple yet effective network distillation alternative to improve backbones, 2021. 2

[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*, pages 764–773, 2017. 2

[6] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. *arXiv preprint arXiv:1810.12890*, 2018. 2

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. 2

[8] 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*, pages 770–778, 2016. 1

[9] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019. 2, 5

[10] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 1

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2

[12] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *arXiv preprint arXiv:1807.03247*, 2018. 2

[13] Songtao Liu, Di Huang, and Yunhong Wang. Learning spatial fusion for single-shot object detection. *arXiv preprint arXiv:1911.09516*, 2019. 4

[14] 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*, pages 8759–8768, 2018. 2

[15] Yudong Liu, Yongtao Wang, Siwei Wang, TingTing Liang, Qijie Zhao, Zhi Tang, and Haibin Ling. Cbnet: A novel composite backbone network architecture for object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11653–11660, 2020. 1

[16] 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 ef-

ficient implementation of object detector. *arXiv preprint arXiv:2007.12099*, 2020. 1, 2, 4

[17] Yanjun Ma, Dianhai Yu, Tian Wu, and Haifeng Wang. Paddlepaddle: An open-source deep learning platform from industrial practice. *Frontiers of Data and Domputing*, 1(1):105–115, 2019. 2

[18] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 4, 2019. 2

[19] 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. 1, 2

[20] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 1

[21] 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*, pages 91–99, 2015. 1

[22] Alexander Sheshkus, Anastasia Ingacheva, Vladimir Arlazarov, and Dmitry Nikolaev. Houghnet: neural network architecture for vanishing points detection. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 844–849. IEEE, 2019. 4

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[24] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020. 2, 4

[25] Aili Wang, Xin He, Pedram Ghamisi, and Yushi Chen. Lidar data classification using morphological profiles and convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters*, 15(5):774–778, 2018. 5

[26] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network, 2020. 2, 4

[27] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 2