PackDet: Packed Long-Head Object Detector

Kun Ding[1], Guojin He[1], Huxiang Gu[2,3], Zisha Zhong[2], Shiming Xiang[2], and Chunhong Pan[2]

[1] Aerospace Information Research Institute, Chinese Academy of Sciences
kding1225@gmail.com, hegj@radi.ac.cn
[2] National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China
{smxiang,chpan}@nlpr.ia.ac.cn, zisha.zhong@ia.ac.cn
[3] Beijing EvaVisdom Tech, guhuxiang@evavisdom.com

Abstract. State-of-the-art object detectors exploit multi-branch structure and predict objects at several different scales, although substantially boosted accuracy is acquired, low efficiency is inevitable as fragmented structure is hardware unfriendly. To solve this issue, we propose a packing operator (PackOp) to combine all head branches together at spatial. Packed features are computationally more efficient and allow to use cross-head group normalization (GN) at handy, leading to notable accuracy improvement against the common head-separate GN. All of these are only at the cost of less than 5.7% relative increase on runtime memory and introduction of a few noisy training samples, however, whose side-effects could be diminished by good packing patterns design. With PackOp, we propose a new anchor-free one-stage detector, PackDet, which features a single deeper/longer but narrower head compared to the existing methods: multiple shallow but wide heads. Our best models on COCO `test-dev` achieve better speed-accuracy balance: 35.1%, 42.3%, 44.0%, 47.4% AP with 22.6, 16.9, 12.4, 4.7 FPS using MobileNet-v2, ResNet-50, ResNet-101, and ResNeXt-101-DCN backbone, respectively. Codes will be released.[4]

Keywords: Object detection, anchor-free, packing features, long head.

1 Introduction

Object detection is a task of simultaneously locating and recognizing objects given an image, which serves as a very fundamental task in computer vision and is a building block of many other tasks, *e.g.*, instance segmentation [7]. Due to the advance of deep learning techniques, more finely-annotated data and stronger computational power, object detection has achieved dramatic improvements, both on accuracy [30, 29, 13] and speed [29].

Prevailing detection pipelines use one or more stages of localization and classification, termed one-stage [18, 30] and multi-stage [26, 2] detectors. More stages generally lead to better localization and recognition accuracy at the cost

[4] https://github.com/kding1225/PackDet

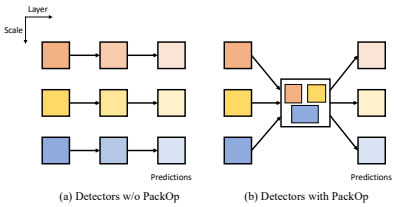


Fig. 1: A concept figure shows how to use PackOp to existing object detectors. PackOp stitches multi-scale features at spatial and performs forwarding all at once, which is faster. However, PackOp blends features at boundary, which will generate noisy data.

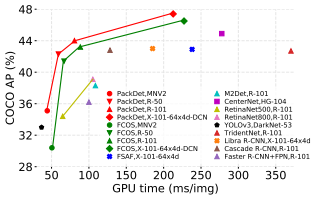


Fig. 2: *Single-model* and *single-scale* performance of different detectors on COCO `test-dev` set. Compared to the state-of-the-art method FCOS, PackDet that integrates PackOp in head can not only improve accuracy, but also decrease inference latency.

of slower speed [2]. At different stages, currently two opposite techniques are developed: anchor-based [26, 38] and anchor-free detectors [30, 43]. The former pre-defines a large number of prior boxes with various sizes to cover the true scale space of objects, while the latter does not use any prior boxes. Enumerating all combinations results in one-stage anchor-based [16], one-stage anchor-free [30], multi-stage anchor-based [26], and multi-stage anchor-free [37, 43] approaches. Considering the superiority of better computational efficiency and generalization ability, one-stage anchor-free detectors have drawn increasing attention [31, 42].

Due to large scale-variation of objects, feature pyramid networks (FPN) and multi-branch structure have become the workhorses of state-of-the-art detectors [15, 16, 37, 30]. FPN fuses and produces a series of feature maps with different scales, whereon objects of a certain range of size are predicted. However, the fragmented multi-branch structure is not hardware friendly and leads to sub-optimal computation efficiency on mainstream deep learning frameworks, *e.g.*, PyTorch, despite of their intensive optimization of asynchronous computation.

To reduce the structure fragments and speed up computation, we develop a packing operator (termed PackOp, ref. Fig. 4 for the implementation) that combines all branches and the associated computations together, avoiding costly for-loops over all heads one by one. A concept figure of utilizing PackOp is shown in Fig. 1, obviously, PackOp can be easily integrated into existing FPN-based detectors. Though PackOp is simple, it could further squeezes the power of modern deep learning frameworks, especially for the cases of using more branches, deeper heads, thinner feature maps and smaller input images (ref. Fig. 8). Surprisingly, it is observed that detecting objects on packed features leads to improved accuracy. We attribute this to the usage of cross-head GN (CH-GN) that is previously ignored and only head-separate GN (HS-GN) is explored. CH-GN works better than HS-GN because of more accurate mean and variance computed across all heads. The side effects incurred by PackOp include increased runtime memory

cost, but which is tiny and could be acceptable, and the incorporation of noisy features, whose negative effect could be mitigated with good packing patterns.

Accordingly, we propose a new anchor-free one-stage detector, called PackDet. Owing to PackOp, the network structure of PackDet is much simpler, as shown in Fig. 3. PackDet first combines multi-scale feature maps into a packed feature map by PackOp, and then feeds the packed features to a shared head, two separate heads (localization and classification head) and the prediction layers, sequentially. A deep but narrow structure of shared head is explored, which consists of many times of repetition of a basic convolution block that contains one convolution, one CH-GN and one activation.

Extensive experiments are conducted on COCO detection dataset to demonstrate the effectiveness of PackDet. Our best models on COCO `test-dev` achieve 35.1%, 42.3%, 44.0%, 47.4% AP with 22.6, 16.9, 12.4, 4.7 FPS on GTX 1080Ti GPU using MobileNet-v2, ResNet-50, ResNet-101, and ResNeXt-101-DCN backbone, respectively, which are better speed-accuracy balance points (ref. Fig. 2) compared to state-of-the-art anchor-free one-stage approaches, *e.g.*, FCOS [30].

2 Related Work

One-stage and Multi-stage Object Detectors. Single-stage detectors use both bounding box regression and multi-class classification once, respectively, without further refinements in modeling. They are usually quite fast in inference and more favorable for mobile applications. Early one-stage detectors are less satisfactory in accuracy despite of striking FPS (frames per second) numbers, such as YOLO [23, 24], SSD [18], DSSD [5]. Subsequent researches of one-stage detectors focus on improving accuracy, while trying to keep the advantage of high efficiency. Related techniques include designing better loss function to solve class-imbalance problem (*e.g.*, RetinaNet [16]), enhancing the multi-scale features (*e.g.*, M2Det [40]), searching network architectures (*e.g.*, NAS-FCOS [31] and EfficientDet [29]). The most common multi-stage detectors use two stages, one is region proposal network and the other is region-wise refinement network. Faster-RCNN [26] is among the earliest methods of this category, and many successors, *e.g.*, R-FCN [3], FPN [15] and TridentNet [13], further strengthen and enrich the two-stage framework. More stages are studied in Cascade R-CNN [2], and significantly improved accuracy is observed, but suffering from longer latency.

Anchor-based and Anchor-free Object Detectors. Anchor boxes serve as scale prior of object distribution and are extensively adopted in both one-stage and multi-stage detectors, such as Fast-RCNN [6], Faster-RCNN [26], YOLO [24], SSD [18]. Anchor boxes can also be learned from data, *e.g.*, MetaAnchor [36], OptAnchor [41]. Pre-defined prior boxes are suspected to have generalization problem and are less computational efficient, thus anchor-free methods begin to resuscitate. Based on feature pyramid networks and other novel ideas, recent work such as FCOS [30], FSAF [43], CenterNet [4], consistently demonstrate that it is possible to drop prior boxes while maintaining a promising result.

Multi-head Object Detectors. To tackle the large scale-variation, state-of-the-art detectors adopt feature pyramid structure (termed *neck*) and detect objects of different scales at the most suitable feature map, unanimously, such as FPN [15], TridentNet [13], SSD [18]. For each feature map, a multi-layer sub-network (termed *head*) is commonly applied to extract task-oriented object-level features. For sake of higher FPS, heads are often limited to a shallow and wide structure, *e.g.*, Faster-RCNN [26], light-head RCNN [14], RetinaNet [16]. By contrast, this work explores deeper and narrower head structure.

Normalization Techniques in Object Detection. Normalization can ease the deep neural networks training and has become a foundation of many algorithms, including object detection. The most commonly-used method is batch normalization (BN) [11], and its variants, freezing BN and synchronized BN [22]. Group norm (GN) [34] is recently proposed as an alternative for BN when batch size is relatively small, which is very appropriate for the tasks like object detection where only small batch is affordable subjected to high image resolution and limited GPU memory. GN is widely adopted in newest detectors [30, 31, 37] to normalize the layers in heads, which are newly-added and untransferable from a pre-trained classification model. However, in these methods, mean and variance are separately computed for different heads.

3 PackDet: Packed Long-Head Object Detector

This section elaborates the proposed packed long-head detector. We first summarize the overall network architecture (Section 3.1), and then detail all parts of PackDet, including packing operator (Section 3.2), head structure (Section 3.3), learning targets (Section 3.4) and loss functions (Section 3.5).

3.1 Network Architecture

As shown in Fig. 3, PackDet has a concise fully convolutional structure, consisting of a backbone, an FPN neck, a packing operator (PackOp), a shared head, a classification head and a localization head. An input image $I \in \mathbb{R}^{3\times H\times W}$ is first fed to the backbone to extract visual features. After that, similar to existing multi-scale detectors [16, 30], three feature maps $C_{l+3}, l \in [0, 3)$ of strides $8, 16, 32$ are combined to generate five features $P_{l+3} \in \mathbb{R}^{C\times h_l\times w_l}, l \in [0, 5)$ of stride $8, 16, 32, 64, 128$, respectively, with h_l the height and w_l the width of P_{l+3}. Here, we denote $[a, b)$ as the set containing all integers no less than a and smaller than b. P_6 and P_7 are obtained by consecutively applying convolution operations on P_5, and $P_3 \sim P_5$ are generated in a top-down manner [16].

Unlike existing methods that construct a branch for per feature map, we first stitch all features together at spatial and obtain a big feature map, denoted as $F^{(p)}$, and only use one main branch subsequently. The packed feature $F^{(p)}$ is passed to a shared head to generate a new feature map $F^{(s)}$. We explore a deep and narrow structure for this shared head, while keeping the spatial size unchanged. $F^{(s)}$ is further fed to two separate heads to generate the classification and localization prediction, respectively.

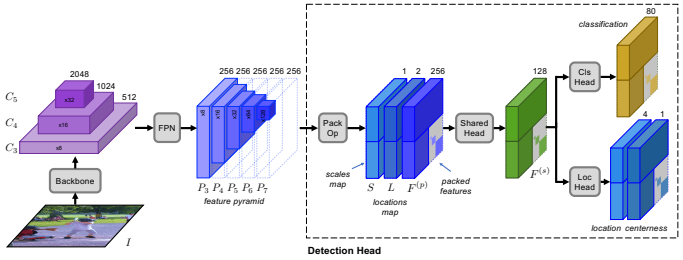


Fig. 3: Overview of PackDet with ResNet-50 as the backbone. PackOp stitches multiple feature maps ($P_3 \sim P_7$) into a patchwork ($F^{(p)}$), see Fig. 4 for more details; shared head and separate heads: Fig. 6; loss computation: Section 3.5.

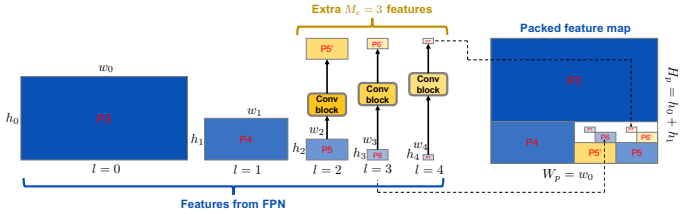


Fig. 4: The procedure of PackOp. Some lines are not drawn in case of cluttering.

3.2 Packing Operator

To reduce the structure fragments in FPN-like head, a packing operator (PackOp) is proposed, which places multi-scale feature maps into a cubic container, *i.e.*, packed feature map, while avoiding overlaps between any two feature maps. This problem is similar to the rectangle packing problem [9] that seeks an enclosing rectangle of minimal area to contain a given set of rectangles without overlap. Rectangle packing is proved to be a NP-completeness problem [12]. Considering the hardness of optimization and the searched solutions may not result in high detection accuracy, we find feasible solutions manually by exploiting the rules found by experiments.

The general case of PackOp with extra M_e (especially, $M_e = 3$) features is explained in Fig. 4. The new features P_5', P_6', P_7' are generated by applying three convolution blocks (*i.e.*, conv-gn-relu) with different initial values to P_5, P_6, P_7, respectively. After that, all the $M_e + 5$ feature maps $F_l, l \in [0, M_e + 5)$ are placed into a big tensor $F^{(p)} \in \mathbb{R}^{C \times H_p \times W_p}$ according to pre-defined placing coordinates, as listed in Table 1. Accordingly, we obtain $H_p = H_0 + H_1$ and $W_p = W_0$. Apart from $F^{(p)}$, PackOp also computes two extra tensors, a scales map $S \in \mathbb{R}^{H_p \times W_p}$ recording the associated scale information of all pixels in $F^{(p)}$, and a locations map $L \in \mathbb{R}^{2 \times H_p \times W_p}$ recording the position of $F^{(p)}$'s pixels mapped back to the original image. S and L are necessary for computing supervision targets and decoding predictions. PackOp can be easily implemented with a few lines of PyTorch code, as is shown in Fig. 5.

l	F_l	source	W_l	H_l	x_l^0	y_l^0	x_l^1	y_l^1
0	F_0	P_3	w_0	h_0	0	0	w_0	h_0
1	F_1	P_4	w_1	h_1	0	h_0	w_1	$h_0{+}h_1$
2	F_2	P_5	w_2	h_2	$w_0{-}w_2$	$h_0{+}h_1{-}h_2$	w_0	$h_0{+}h_1$
3	F_3	P_6	w_3	h_3	$w_0{-}w_2{-}w_3$	$h_0{+}h_1{-}h_2{-}h_3$	$w_0{-}w_2$	$h_0{+}h_1{-}h_2$
4	F_4	P_7	w_4	h_4	$w_0{-}w_2{-}w_3{-}w_4$	$h_0{+}h_1{-}h_2{-}h_3{-}h_4$	$w_0{-}w_2{-}w_3$	$h_0{+}h_1{-}h_2{-}h_3$
5	F_5	P_5'	w_2	h_2	w_1	$h_0{+}h_1{-}h_2$	$w_1{+}w_2$	$h_0{+}h_1$
6	F_6	P_6'	w_3	h_3	$w_0{-}w_3$	$h_0{+}h_1{-}h_2{-}h_3$	w_0	$h_0{+}h_1{-}h_2$
7	F_7	P_7'	w_4	h_4	$w_0{-}w_3{-}w_4$	$h_0{+}h_1{-}h_2{-}h_3{-}h_4$	$w_0{-}w_3$	$h_0{+}h_1{-}h_2{-}h_3$

Table 1: Coordinates of F_l in $F^{(p)}$. $x_l^0, y_l^0, x_l^1, y_l^1$ are top-left horizontal, top-left vertical, bottom-right horizontal, bottom-right vertical coordinate, respectively.

```
def PackOp(xs, boxes, levels, locations, Hp, Wp):
    # xs: feature map F_l, l=0,1,...,M-1, F_l: (N, C, H_l, W_l)
    # boxes: positions in packed feature map, (M, 4)
    # levels: level l for each feature map F_l, (M,)
    # locations: grid points per feature map, (N, 2, H_l, W_l)
    # Hp,Wp: height and width of packed feature map
    N, C = xs[0].shape[:2]
    pack_map = xs[0].new_zeros(N, C, Hp, Wp)
    scales_map = xs[0].new_zeros(N, 1, Hp, Wp)
    locations_map = -xs[0].new_ones(N, 2, Hp, Wp)
    for x, box, l, loc in zip(xs, boxes, levels, locations):
        x0, y0, x1, y1 = box
        pack_map[..., y0:y1, x0:x1] = x
        scales_map[..., y0:y1, x0:x1] = l
        locations_map[..., y0:y1, x0:x1] = loc
    return pack_map, scales_map, locations_map
```

Fig. 5: PyTorch code of PackOp.

As will be shown in our experiments (ref. Fig. 8), the proposed approach has universality in speeding up FPN-like head structure. Although this work explores the efficiency in object detection task, the underlying idea could also be applied to other tasks/methods as long as FPN-like multi-branch structure is adopted. It is worth noting that our work is quite different from architecture search [31, 33], though which could also reduce network latency, as it generally focuses on discovering network structure automatically by optimizing accuracy-speed trade-off. More similar work include jigsaw based unsupervised learning [32] and mixup-based data augmentation [39], but they have distinct differences to our method and are proposed in different contexts and for different applications. The idea of packing has already appeared in earlier work, such as [21, 10]. However, the packing operations therein are performed in image space to make convnet feature pyramids faster to compute. Most recently, mosaic data augmentation is proposed in YOLOv4 [1], which combines 4 training images into one in certain ratios. The underlying idea also shares some similarities to ours, but also being performed in image space.

3.3 Packed Long Head

The overall head structure excluding PackOp is shown in Fig. 6. It contains three parts: shared head, classification head and localization head.

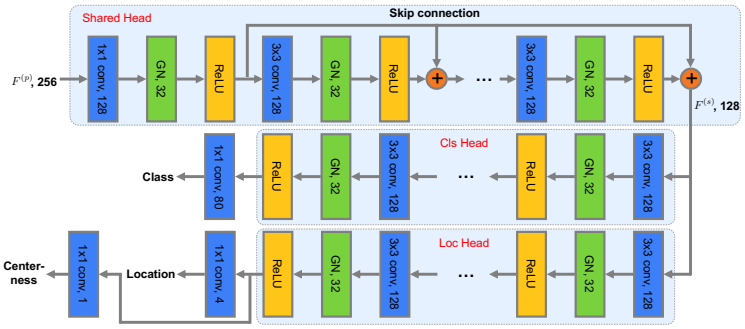


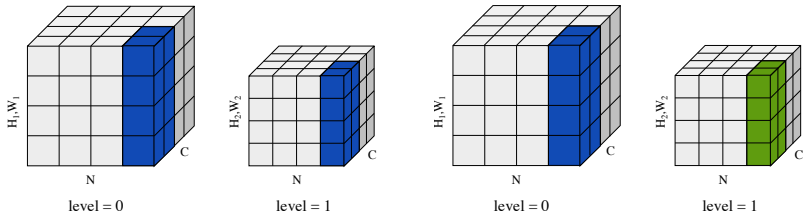
Fig. 6: The head architecture of PackDet (PackOp is not shown).

The shared head takes as an input $F^{(p)} \in \mathbb{R}^{C \times H_p \times W_p}$ and outputs $F^{(s)} \in \mathbb{R}^{C_s \times H_p \times W_p}$ with C_s the channel number of the output feature map. A dimension reduction block $R(\cdot)$ (1x1conv-gn-relu) is first applied to reduce the dimension, and then T_1 convolution blocks with 3×3 convolution $CB_i^1(\cdot), i \in [0, T_1)$ are performed sequentially, *i.e.* $F^{(s)} = CB_{T_1-1}^1(\cdots(CB_0^1(F^{(d)}), F^{(d)}), F^{(d)})$, where $F^{(d)} \triangleq R(F^{(p)})$ is the dimension reduced feature map. Therein, a skip connection from $F^{(d)}$ at each block is adopted. In all blocks, the spatial dimension is kept fixed, as objects should be detected at every positions determined by the locations map L that is intuitively unsuitable for interpolation.

Generally speaking, detection and classification are two quite different tasks and need task-specific features. To this end, two separate heads are stacked on top of the shared head: classification head and localization head. The structure of these two heads are identical to that of the shared head besides removed skip connections and fewer convolution blocks. We perform classification on the output of classification head and regression+centerness prediction on localization head (ref. Section 3.4). For convenience, let T_2 denote the number of convolution blocks used in classification/localization head.

PackDet uses deeper while narrower head structure, that is to say, $T_1 + T_2$ is relatively large, *e.g.*, 14, and C_s is relatively small, *e.g.*, 128. By contrast, existing methods like FCOS [30] and RetinaNet [16] use shallow and wide heads. For instance, FCOS uses 4 convolution blocks (denoted as T) for both classification and localization head, the channel size (denoted as C) is set to be 256. The number of convolution parameters (excluding the last prediction layers) in PackDet's heads is $CC_s + (9T_1 + 18T_2)C_s^2$, and $18TC^2$ in FCOS's heads. The convolution FLOP count of PackDet's and FCOS's heads are proportional to $[CC_s + (9T_1 + 18T_2)C_s^2]H_pW_p$ and $18TC^2 \sum_{l=0}^{4} H_lW_l$, respectively. Considering $\sum_{l=0}^{4} H_lW_l/H_pW_p \approx 0.89$, under the settings $T = 4, T_1 = 12, T_2 = 2, C = 256, C_s = 128$, PackDet's heads should have fewer parameters and FLOPs

Compared to processing all heads separately, PackOp slightly increases runtime memory cost by $\beta = 1 - \sum_{l=0}^{M-1} H_lW_l/H_pW_p$, relatively. For the case $M_e = 0$, β is about 11.2%; for the case $M_e = 3$, it is about 5.7%. While considering back-



(a) CH-GN: cross-head GN (b) HS-GN: head-separate GN

Fig. 7: Comparison between cross-head and head-separate normalization methods with group normalization (GN) taken as an example. Pixels with same color are normalized by the same statistics computed on the values of these pixels.

bone, neck and head as a whole, the relative increment might be smaller. For example, when ResNet-50 serves as the backbone and M_e is 3, the GPU memory cost is only increased by 5.5%, relatively.

PackDet's heads perform convolution operations on packed feature maps, which inevitably incorporates some noisy features by mixing up spatially close features. The noisy data may mess the training procedure, and finally lead to worse performance. However, as will be demonstrated in the experiments, by carefully designing the packing pattern, this side-effect can be largely mitigated.

Due to the usage of packed features, feature normalization at each convolution block is performed across different heads. Fig. 7 compares the differences between cross-head GN (CH-GN) and the common head-separate GN (HS-GN). In CH-GN, the statistics are computed based on values from all scales, which are more accurate and facilitate to get better accuracy (ref. Fig. 9).

3.4 Learning Targets

Considering the ij-th element of L, the associated feature stride is $2^{S_{i,j}+3}$ and the corresponding space location mapped back to the original image is $(L_{0,i,j}, L_{1,i,j}) = (2^{S_{i,j}+3}(j+0.5), 2^{S_{i,j}+3}(i+0.5))$. Following the improved version of FCOS [30], box shrinking strategy is adopted to reduce noisy positive samples. For this end, let $B = (c, x, y, w, h)$ be a ground-truth box, and $B_s = (c, x, y, w', h')$ be a central shrunk version of B, where c is the class id, (x, y) is box center, w and h are the width and height of B, w' and h' are the width and height of B_s, respectively. Pixel (i, j) denotes a positive sample when $x - w'/2 \le L_{0,i,j} \le x + w'/2, y - h'/2 \le L_{1,i,j} \le y + h'/2$ holds, otherwise denotes a negative sample. For a positive sample, let $c^*_{i,j} = c$ be the classification target and $\mathbf{o}^*{}_{i,j} = (o^l, o^t, o^r, o^b)$ be the regression target, where

$$o^l = \frac{1}{2^{S_{i,j}+3}}[L_{0,i,j} - (x - w/2)], \quad o^t = \frac{1}{2^{S_{i,j}+3}}[L_{1,i,j} - (y - h/2)],$$
$$o^r = \frac{1}{2^{S_{i,j}+3}}[(x + w/2) - L_{0,i,j}], \quad o^b = \frac{1}{2^{S_{i,j}+3}}[(y + h/2) - L_{1,i,j}]. \qquad (1)$$

o^l, o^t, o^r, o^b are the stride-normalized distances between $(L_{0,i,j}, L_{1,i,j})$ and the left, top, right, bottom boundaries of B, respectively. Like [30], centerness is also predicted to filter out low-quality boxes. For a position (i, j) that associates to a positive sample, the ground-truth centerness target is defined as $t^*_{ij} = \sqrt{\min(o^l, o^r)/\max(o^l, o^r) \cdot \min(o^t, o^b)/\max(o^t, o^b)}$. At test stage, centerness is used for down-weighting the scores of boxes that are far away from center.

3.5 Loss Functions

Assume that at location (i, j) the predicted class distribution vector is $\mathbf{p}_{i,j}$, the predicted class-agnostic bounding box offset vector is $\mathbf{o}_{i,j}$ and the predicted centerness is $t_{i,j}$, the following multi-task loss is adopted:

$$L(\{\mathbf{p}_{i,j}\}, \{\mathbf{o}_{i,j}\}, \{t_{i,j}\}) = \frac{1}{N_{\text{pos}}} \sum\nolimits_{ij} L_{\text{cls}}(\mathbf{p}_{i,j}, c^*_{i,j}) + \frac{1}{N_{\text{pos}}} \sum\nolimits_{ij} I_{c^*_{i,j}>0} L_{\text{reg}}(\mathbf{o}_{i,j}, \mathbf{o}^*_{i,j}) + \frac{1}{N_{\text{pos}}} \sum\nolimits_{ij} I_{c^*_{i,j}>0} L_{\text{cen}}(t_{i,j}, t^*_{i,j}), \quad (2)$$

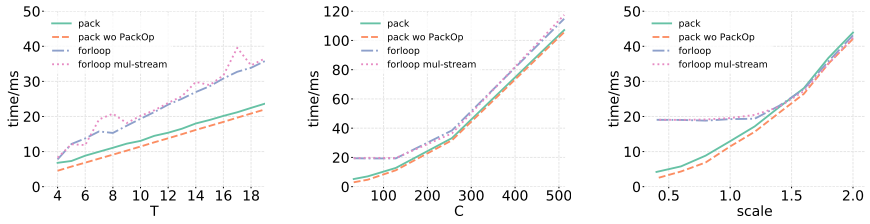
where $L_{\text{cls}}, L_{\text{reg}}, L_{\text{cen}}, N_{\text{pos}}$ denote classification loss, regression loss, centerness loss and the number of positive samples, respectively. Focal loss [16], GIoU loss [27], and binary cross entropy loss are used for $L_{\text{cls}}, L_{\text{reg}}, L_{\text{cen}}$, respectively.

3.6 Implementation Details

Training Details The network initialization follows the tradition in [30, 16]. Please refer to these work for more details. PackDet is trained with stochastic gradient descent (SGD) algorithm. Unless specified, all models are trained use a batch size of 16 and the `1x` training strategy:[5] 1) use 90K iterations and initial learning rate of 0.01; 2) reduce learning rate by a factor 10 at iterations 60K and 80K, respectively. Unless otherwise specified, images are resized to have shorter side being 800 and longer size no more than 1333, meanwhile, only horizontal image flipping is applied. By default, we use $T_1 = 12$, $T_2 = 2$ and $C_s = 128$, which results in a long head with narrow feature maps.

Inference Details At testing time, an image is fed to the network to get three tensors: a class score map ($K \times H_p \times W_p$, $K = 80$ for COCO), a box offset map ($4 \times H_p \times W_p$) and a centerness map ($H_p \times W_p$). We first threshold the element-wise product of class score map and centerness map by 0.05 to obtain some candidate objects. After that, predictions of the same scale are filtered to keep only top-1000 most confidential results. Finally, the predictions from all scales are merged followed by non-maximum suppression (NMS) with a threshold of 0.5 to generate the final results.

[5] https://github.com/facebookresearch/maskrcnn-benchmark



(a) varying T (b) varying C (c) varying scale

Fig. 8: GPU time comparison with varying T, C and scale s. The scales $\{2.0, 1.8, 1.6, 1.4, 1.2, 1.0, 0.8, 0.6, 0.4\}$ are used to resize input tensors. The default value of T, C, s are $10, 128, 1.0$, respectively.

4 Experiments

We conduct experiments on COCO detection benchmark [17]. The `trainval35k` set is used for training, and the `minival` set is used for ablation study. For state-of-the-art comparison, we report COCO AP on the `test-dev` split.

4.1 Ablation Study

Benchmark PackOp's Speed To validate the effectiveness of PackOp as a universal speeder for multi-branch structure, we benchmark the latency of forwarding on a multi-branch network. The network consists of 8 branches and each is a sequence of T convolution blocks (conv-gn-relu). The input tensors for all branches have the same batch size of 2 and channel number C. Their spatial dimensions are $(100, 100)$, $(50, 50)$, $(25, 25)$, $(13, 13)$, $(7, 7)$, $(25, 25)$, $(13, 13)$, and $(7, 7)$, respectively. All convolutions have the same kernel size of $3 \times 3 \times C$, padding of 1, and stride of 1. We consider two methods for forwarding: 1) loop over the branches one by one; 2) pack all branches and run forwarding only once.

We use PyTorch-1.1.0, CUDA-9.0, cuDNN-7.3.0, and perform forwarding on a single GTX 1080Ti GPU. Elapsed time is recorded by `torch.cuda.Event()` and the average time over 30 trials is reported. The results are shown in Fig. 8, where `pack wo PackOp` denotes the packing method excluding the PackOp time, `pack` is the same method counts this time, and `forloop` denotes the branch-by-branch method. Multiple CUDA streams are also tried for `forloop` and the corresponding method is called `forloop mul-stream`. Specifically, two CUDA streams are used as more streams do not make distinct difference. From Fig. 8, the following conclusions can be drawn: 1) PackOp is quite efficient to execute; 2) using multiple CUDA streams takes no effect on speed; 3) the speedup effect is more considerable for deeper but narrower network and smaller spatial size. We have also tried two other deep learning frameworks, TensorFlow and MXNet, and similar results are obtained. Please refer to supplementary for more details.

M_e	tr-pack	ts-pack	ch-gn	time	AP	AP_{50}	AP_{75}	No.
0				67.4	38.9	56.7	42.2	①
0		✓		63.1	22.9	35.9	24.5	②
0			✓	68.9	39.5	57.2	42.7	③
0	✓	✓	✓	64.4	39.7	57.5	43.1	④
3				75.0	39.4	56.8	42.9	⑤
3		✓		65.2	25.9	39.7	28.1	⑥
3			✓	74.0	39.7	57.4	43.0	⑦
3	✓	✓	✓	65.3	40.1	57.5	43.3	⑧

Table 2: The effectiveness of packed head in PackDet for speeding up inference and improving accuracy.

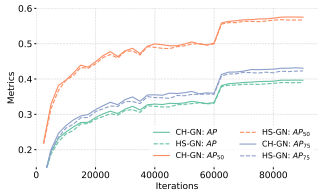


Fig. 9: Validation accuracy comparison between the methods using CH-GN and HS-GN. M_e is set to be 0.

Speeding up PackDet by PackOp To validate the effect of PackOp in speeding up forward computation, we slightly modify the packed head structure as shown in Fig. 6: PackOp is dropped and all the convolution blocks are computed using nested two-level for-loops over layers and branches. The results are listed in Table 2, where `tr-pack` and `ts-pack` denote whether PackOp is used in train and test stage, respectively. When `ch-gn` is checked, it means CH-GN is used, otherwise HS-GN is used. M_e means how many extra feature maps are added, it should be 0 or 3 (ref. Fig. 4). The listed times denote the inference time (in ms). By comparing line 1 to line 4, the relative reduction of inference time is 4.5%. With more branches, this number is larger, *i.e.*, 12.9% for $M_e = 3$ (ref. line 5 and 8 in Table 2). More results with varying input and channel size are given in the supplementary material. As a result, PackOp is indeed capable to speed up the network inference, and the more branches used the larger gain obtained.

The Usefulness of Test Time Packing It is possible to train a multi-branch detector in a traditional manner without using PackOp but use it in test stage. Nevertheless, our experimental results demonstrate that this strategy is sub-optimal. By comparing the results of line 1 and 2 (or line 5 and 6) in Table 2, we see that AP degrades significantly even though testing time also has a large drop, which could be attributed to the bias between training and testing.

Effect of Packed Convolution and Normalization In Table 2, line 3 and 7 correspond to the method without using PackOp, while CH-GN is adopted. This is implemented by first concatenating the spatially flattened feature maps from all scales of the same layer and then applying the naive GN, followed by a split operation. By comparing line 1 to line 3 (or line 5 to line 7), we find that CH-GN improves AP visibly. As the main difference between PackDet and the detectors using multi-branch structure lies in whether packed convolution and normalization are utilized or not, we believe that CH-GN does contribute to the accuracy improvement. However, there is still a small AP margin between the results of line 3 and line 4 (or line 7 and line 8). We conjecture that the

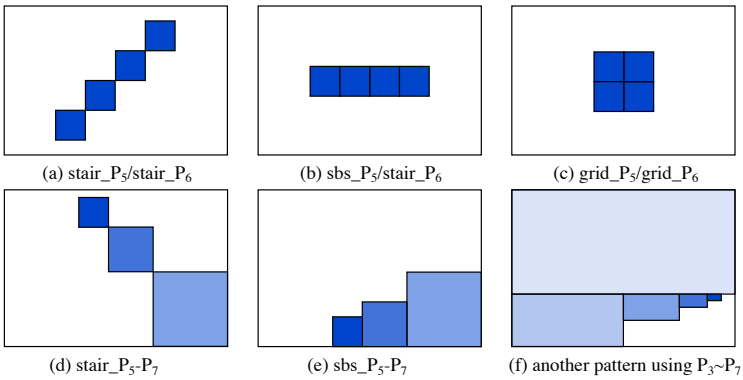


Fig. 10: Different packing patterns being compared. (f) uses $P_3 \sim P_7$ for packing. Different colors denote feature map of different scales. `sbs`: side-by-side.

correlation between the highest three scales are captured, at least in some degree, which are helpful for detecting large objects.

Packing Extra Features In Table 2, by comparing the results in line 4 and 8, using extra three features increases 0.4% absolute AP. Thus, we conclude that the prediction results from different feature maps of the same scale should have some complementarity. How to use more feature maps and if extra improvements can be obtained are two open questions, we leave these for future work.

Packing Patterns Comparison Different packing patterns may lead to different results. For example, to pack feature maps $P_3 \sim P_7$, the pattern shown in Fig. 10(f) is also valid. However, when it is used in PackDet's head, the resultant AP is 39.3%, which is lower than the number 39.7% given in Table 2 (line 4). Thus, it is necessary to study what kind of rules should be followed to design good packing patterns. For this aim, we manually design five patterns for the following experiments, as shown in Fig. 10 (a)-(e). (a)-(c) use features of same scale for packing. Feature map P_5 (or P_6) and its processed versions by different convolution blocks are used. (d) and (e) use P_5, P_6, P_7 for packing.

The results of same scale packing are shown in Table 3(a) and Table 3(b), where `pack` means if all branches are merged for forwarding in both train and test stage, and `ch-gn` denotes if CH-GN is adopted. For separate branches, CH-GN is performed as the manner mentioned before. Table 3(a) uses P_5-series features, corresponding to a stride of 32. The results with different packing patterns are similar and only slightly worse than those without packing operator, implying the introduced noisy samples are unable to degenerate the accuracy a lot. Table 3(b) uses P_6-series features, corresponding to relatively large objects. From the results, we find that different packing patterns have different effect on accuracy. The `grid` pattern leads to more mixed samples at packing boundaries, thus, it gets

the worst result. As for the `stair` pattern, it often acquires the best result. Table 3(c) gives the packing results using $P_3 \sim P_7$. The results of `sbs_P5-P7` and `stair_P5-P7` are similar, and both of them are better than those without PackOp, which could be attributed to the learned correlation between the three feature maps. It is indeed possible as the receptive field is large enough to cover all these feature maps and convolution networks have strong representative ability.

pattern	pack	ch-gn	AP	AP_{50}	AP_{75}
stair_P_5	✓	✓	14.6	22.3	15.5
sbs_P_5	✓	✓	14.2	22.0	15.0
grid_P_5	✓	✓	14.1	21.8	14.8
grid_P_5			14.7	22.5	15.6
grid_P_5		✓	14.8	22.9	15.6

(a)

pattern	pack	ch-gn	AP	AP_{50}	AP_{75}
stair_P_6	✓	✓	12.6	17.9	13.2
sbs_P_6	✓	✓	9.3	13.3	10.0
grid_P_6	✓	✓	4.1	6.6	4.2
grid_P_6			12.7	18.2	13.4
grid_P_6		✓	12.8	18.4	13.5

(b)

pattern	pack	ch-gn	AP	AP_{50}	AP_{75}
sbs_P_5-P_7	✓	✓	24.7	34.4	26.6
stair_P_5-P_7			24.4	34.2	26.3
stair_P_5-P_7		✓	24.4	34.3	26.4
stair_P_5-P_7	✓	✓	24.6	34.5	26.4

(c)

Table 3: Results of different packing patterns using (a) P_5 and its variants, (b) P_6 and its variants, and (c) $P_5 \sim P_7$. `sbs`: side-by-side.

We summarize the rules of thumb for designing good packing patterns as follows: 1) It is not a good idea to put feature maps with the same high scale together; 2) Putting feature maps with the same low scale is feasible; 3) Putting several large-stride feature maps of different scales is helpful for improving accuracy. Reviewing Fig. 4 again, the designed packing pattern uses stair-like structure as much as possible and no feature maps of the same high scale are put near each other. As a result, PackDet could not only enjoy the speedup brought from PackOp, but also minimize its side effects.

4.2 Comparison to State of the Art

We further compare PackDet to state-of-the-art multi-stage and single-stage methods on COCO `test-dev` set. Following previous work, PackDet uses the `2x` learning schedule, which runs 180K iterations with the learning rate change points tuned proportionally w.r.t. the `1x` schedule. In addition, the shorter side of input images are randomly re-scaled to the range $[640, 800]$. Note that, when MobileNet-v2 [28] is used as the backbone, synchronous BN will be adopted as it boosts accuracy remarkably. Unless otherwise specified, M_e is set to be 3, train packing, test packing and CH-GN are all adopted.

The numerical results of different methods are shown in Table 4 and the speed-accuracy curves are plotted in Fig. 2. We can see that PackDet achieves better speed-accuracy trade-offs compared to the state-of-the-art anchor-free one-stage method FCOS. For example, when X-101-64x4d-DCN backbone is adopted, PackDet acquires 47.4% AP with 4.7 FPS, which outperforms FCOS notably, *i.e.*, 46.5% AP with 4.4 FPS. The speed and accuracy superiority of PackDet is more considerable for small backbones. For example, with MobileNet-v2 as the backbone, PackDet improves the baseline FCOS by a large margin, *i.e.*, 4.7% absolute AP and 2.8 FPS increase. These results imply that a single packed deep

Method	Input Size	Backbone	Anchor Free	MS Train	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Multi-Stage:											
Faster R-CNN+FPN [15]	$\sim 1000 \times 600$	R-101			10.0	36.2	59.1	39.0	18.2	39.0	48.2
Cascade R-CNN [2]	$\sim 1300 \times 800$	R-101			7.8	42.8	62.1	46.3	23.7	45.5	55.2
TridentNet [13]	$\sim 1200 \times 800$	R-101		✓	2.7	42.7	63.6	46.5	23.9	46.6	56.6
TridentNet [13]	$\sim 1200 \times 800$	R-101-DCN		✓	1.3	46.8	67.6	51.5	28.0	51.2	60.5
Libra R-CNN [20]	$\sim 1300 \times 800$	X-101-64x4d			5.4	43.0	64.0	47.0	25.3	45.6	54.6
One-Stage:											
YOLOv3 [25]	608×608	DarkNet-53		✓	27.3	33.0	57.9	34.4	18.3	35.4	41.9
RetinaNet500 [16]	$\sim 832 \times 500$	R-101		✓	15.4	34.4	53.1	36.8	14.7	38.5	49.1
RetinaNet800 [16]	$\sim 1300 \times 800$	R-101		✓	9.5	39.1	59.1	42.3	21.8	42.7	50.2
CenterNet [4]	511×511	HG-104	✓	✓	3.6	44.9	62.4	48.1	25.6	47.4	57.4
M2Det [40]	512×512	R-101		✓	9.2	38.8	59.4	41.7	20.5	43.9	53.4
FSAF [43]	$\sim 1300 \times 800$	X-101-64x4d	✓	✓	4.2	42.9	63.8	46.3	26.6	46.2	52.7
FCOS [30]	$\sim 1300 \times 800$	MNV2	✓	✓	19.8	30.4	47.5	32.2	19.1	34.3	33.7
FCOS [30]	$\sim 1300 \times 800$	R-50	✓	✓	15.0	41.4	60.2	44.7	24.8	44.2	50.6
FCOS [30]	$\sim 1300 \times 800$	R-101	✓	✓	11.3	43.2	62.2	46.9	26.1	46.2	53.6
FCOS [30]	$\sim 1300 \times 800$	X-101-64x4d-DCN	✓	✓	4.4	46.5	65.7	50.5	28.9	49.2	58.1
PackDet	$\sim 1300 \times 800$	MNV2	✓	✓	22.6	35.1	51.6	38.1	19.1	37.0	44.0
PackDet	$\sim 1300 \times 800$	R-50	✓	✓	16.9	42.3	60.3	45.9	24.7	45.0	53.7
PackDet$^{\diamond\square}$	$\sim 1300 \times 800$	R-101	✓	✓	12.0	43.5	61.9	47.3	26.4	46.4	53.7
PackDet$^{\square}$	$\sim 1300 \times 800$	R-101	✓	✓	11.2	43.7	61.9	47.6	25.8	46.9	55.3
PackDet	$\sim 1300 \times 800$	R-101	✓	✓	12.4	44.0	62.3	47.8	25.6	47.3	55.7
PackDet	$\sim 1300 \times 800$	X-101-64x4d-DCN	✓	✓	4.7	47.4	66.3	51.5	28.9	50.5	60.1

$\diamond$: $M_c = 0$; $\square$: without train/test packing or CH-GN; `R`: ResNet [8]; `X`: ResNeXt [35]; `MNV2`: MobileNet-v2 [28]; `HG`: Hourglass [19]; `DCN`: Deformable Convolutional Network [44]; `Input Size`: input image size; `Anchor Free`: whether no anchors are used; `MS Train`: whether multi-scale training is used.

Table 4: *Single-model and single-scale* results *vs.* state of the arts on COCO `test-dev` set. FPS is measured on a single GTX 1080Ti GPU card with a batch size of 1.

and narrow head is competitive to multiple shallow and wide heads in terms of both accuracy and speed. Finally, we will argue the accuracy improvements of PackDet are not caused by more model parameters as PackDet has fewer parameters than FCOS. Actually, PackDet could reduce about 0.7M parameters with the same backbone.

5 Conclusions

In this paper, we presented a packing operator called PackOp that puts a group of feature maps into a cubic container. With PackOp, a new detector PackDet was proposed, which packs all branches together for fast forwarding. Beside the speed advantage, the parallized structure is also favorable for handy cross-head normalization that facilitates to get better accuracy. Extensive experiments demonstrated PackOp is effective in speedup and PackDet could get better accuracy-speed trade-off against state of the arts. Future work include searching the packing pattern and applying PackOp to other tasks, such as instance segmentation and keypoints detection.

Acknowledgement

This research was financially supported by National Natural Science Foundation of China (61731022, 91646207) and the Strategic Priority Research Program of the Chinese Academy of Sciences (XDA19090300). We would like to thank Rui Yang and Chaoyi Liu from EvaVisdom Tech for the inspiring discussions. We also thank the anonymous reviewers for their valuable suggestions.

References

1. Bochkovskiy, A., Wang, C.Y., Liao, H.Y.: YOLOv4: Optimal speed and accuracy of object detection. arXiv: 2004.10934 (2020)
2. Cai, Z., Vasconcelos, N.: Cascade R-CNN: Delving into high quality object detection. In: CVPR (2018)
3. Dai, J., Li, Y., He, K., et al.: R-FCN: Object detection via region-based fully convolutional networks. In: NeurIPS (2016)
4. Duan, K., Bai, S., Xie, L., et al.: Centernet: Keypoint triplets for object detection. arXiv: 1904.08189 (2019)
5. Fu, C., Liu, W., Ranga, A., et al.: DSSD: Deconvolutional single shot detector. arXiv: 1701.06659 (2017)
6. Girshick, R.B.: Fast R-CNN. In: ICCV (2015)
7. He, K., Gkioxari, G., Dollár, P., et al.: Mask R-CNN. In: ICCV (2017)
8. He, K., Zhang, X., Ren, S., et al.: Deep residual learning for image recognition. In: CVPR (2016)
9. Huang, E., Korf, R.E.: New improvements in optimal rectangle packing. In: IJCAI (2009)
10. Iandola, F., Moskewicz, M., Karayev, S., et al.: DenseNet: Implementing efficient convnet descriptor pyramids. arXiv: 1404.1869 (2014)
11. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML (2015)
12. Korf, R.E.: Optimal rectangle packing: Initial results. In: ICAPS (2003)
13. Li, Y., Chen, Y., Wang, N., et al.: Scale-aware trident networks for object detection. In: ICCV (2019)
14. Li, Z., Peng, C., Yu, G., et al.: Light-head R-CNN: In defense of two-stage object detector. arXiv: 1711.07264 (2017)
15. Lin, T., Dollár, P., Girshick, R.B., et al.: Feature pyramid networks for object detection. In: CVPR (2017)
16. Lin, T., Goyal, P., Girshick, R.B., et al.: Focal loss for dense object detection. In: ICCV (2017)
17. Lin, T., Maire, M., Belongie, S.J., et al.: Microsoft COCO: Common objects in context. In: ECCV (2014)
18. Liu, W., Anguelov, D., Erhan, D., et al.: SSD: Single shot multibox detector. In: ECCV (2016)
19. Newell, A., Yang, K., Deng, J.: Stacked hourglass networks for human pose estimation. In: ECCV (2016)
20. Pang, J., Chen, K., Shi, J., et al.: Libra R-CNN: Towards balanced learning for object detection. In: CVPR (2019)
21. Papandreou, G., Kokkinos, I., Savalle, P.A.: Untangling local and global deformations in deep convolutional networks for image classification and sliding window detection. arXiv: 1412.0296 (2014)
22. Peng, C., Xiao, T., Li, Z., et al.: MegDet: A large mini-batch object detector. In: CVPR (2018)
23. Redmon, J., Divvala, S.K., Girshick, R.B., et al.: You only look once: Unified, real-time object detection. In: CVPR (2016)
24. Redmon, J., Farhadi, A.: YOLO9000: Better, faster, stronger. In: CVPR (2017)
25. Redmon, J., Farhadi, A.: YOLOv3: An incremental improvement. arXiv: 1804.02767 (2018)

26. Ren, S., He, K., Girshick, R.B., et al.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NeurIPS (2015)
27. Rezatofighi, H., Tsoi, N., Gwak, J., et al.: Generalized intersection over union: A metric and a loss for bounding box regression. In: CVPR (2019)
28. Sandler, M., Howard, A.G., Zhu, M., et al.: MobileNetV2: Inverted residuals and linear bottlenecks. In: CVPR (2018)
29. Tan, M., Pang, R., Le, Q.V.: EfficientDet: Scalable and efficient object detection. arXiv: 1911.09070 (2019)
30. Tian, Z., Shen, C., Chen, H., et al.: FCOS: Fully convolutional one-stage object detection. arXiv: 1904.01355 (2019)
31. Wang, N., Gao, Y., Chen, H., et al.: NAS-FCOS: Fast neural architecture search for object detection. arXiv: 1906.04423 (2019)
32. Wei, C., Xie, L., Ren, X., et al.: Iterative reorganization with weak spatial constraints: Solving arbitrary jigsaw puzzles for unsupervised representation learning. In: CVPR (2019)
33. Wu, B., Dai, X., Zhang, P., et al.: FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: CVPR (2019)
34. Wu, Y., He, K.: Group normalization. In: ECCV (2018)
35. Xie, S., Girshick, R.B., Dollár, P., et al.: Aggregated residual transformations for deep neural networks. In: CVPR (2017)
36. Yang, T., Zhang, X., Li, Z., et al.: MetaAnchor: Learning to detect objects with customized anchors. In: NeurIPS (2018)
37. Yang, Z., Liu, S., Hu, H., et al.: RepPoints: Point set representation for object detection. arXiv: 1904.11490 (2019)
38. Zhang, S., Wen, L., Bian, X., et al.: Single-shot refinement neural network for object detection. In: CVPR (2018)
39. Zhang, Z., He, T., Zhang, H., et al.: Bag of freebies for training object detection neural networks. arXiv: 1902.04103 (2019)
40. Zhao, Q., Sheng, T., Wang, Y., et al.: M2Det: A single-shot object detector based on multi-level feature pyramid network. In: AAAI (2019)
41. Zhong, Y., Wang, J., Peng, J., et al.: Anchor box optimization for object detection. arXiv: 1812.00469 (2018)
42. Zhu, C., Chen, F., Shen, Z., et al.: Soft anchor-point object detection. arXiv: 1911.12448 (2019)
43. Zhu, C., He, Y., Savvides, M.: Feature selective anchor-free module for single-shot object detection. In: CVPR (2019)
44. Zhu, X., Hu, H., Lin, S., et al.: Deformable convnets V2: More deformable, better results. In: CVPR (2019)