arXiv:1912.04260v1 [cs.CV] 9 Dec 2019

# Side-Aware Boundary Localization for More Precise Object Detection

Jiaqi Wang[1] Wenwei Zhang[2] Yuhang Cao[1] Kai Chen[1] Jiangmiao Pang[3] Tao Gong[4]
Jianping Shi[5] Chen Change Loy[2] Dahua Lin[1]
[1]The Chinese University of Hong Kong [2]Nanyang Technological University
[3]Zhejiang University [4] University of Science and Technology of China [5] SenseTime Group Limited
{wj017,dhlin}@ie.cuhk.edu.hk {yhcao6,chenkaidev,pangjiangmiao,gongtao950513}@gmail.com
{wenwei001,ccloy}@ntu.edu.sg shijianping@sensetime.com

## Abstract

*Current object detection frameworks mainly rely on bounding box regression to localize objects. Despite the remarkable progress in recent years, the precision of bounding box regression remains unsatisfactory, hence limiting performance in object detection. We observe that precise localization requires careful placement of each side of the bounding box. However, the mainstream approach, which focuses on predicting centers and sizes, is not the most effective way to accomplish this task, especially when there exists displacements with large variance between the anchors and the targets. In this paper, we propose an alternative approach, named as* ***Side-Aware Boundary Localization (SABL)****, where each side of the bounding box is respectively localized with a dedicated network branch. Moreover, to tackle the difficulty of precise localization in the presence of displacements with large variance, we further propose a two-step localization scheme, which first predicts a range of movement through bucket prediction and then pinpoints the precise position within the predicted bucket. We test the proposed method on both two-stage and single-stage detection frameworks. Replacing the standard bounding box regression branch with the proposed design leads to significant improvements on Faster R-CNN, RetinaNet, and Cascade R-CNN, by 3.0%, 1.6%, and 0.9%, respectively. Code and models will be available at* https://github.com/open-mmlab/mmdetection.

## 1. Introduction

The development of new frameworks for object detection, *e.g.*, *Faster R-CNN* [35], *RetinaNet* [24], and *Cascade R-CNN* [1], has substantially pushed forward the state of the art. All these frameworks, despite the differences in their technical designs, have a common component, namely *bounding box regression*, for object localization.

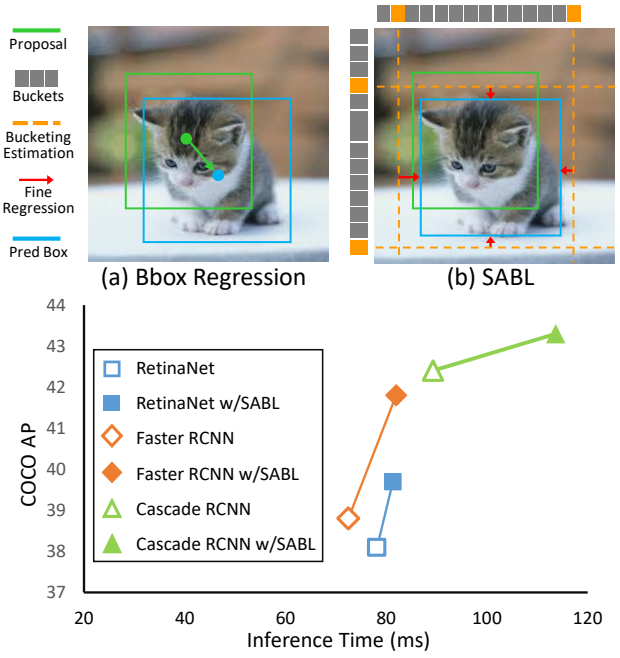



Figure 1: **The Illustration of SABL and its effectiveness**. (a) Common bounding box regression directly predicts displacements from proposals to ground-truth boxes. (b) SABL focuses on object boundaries and localizes them with bucketing scheme via bucketing estimation and fine regression. SABL brings remarkable improvements across RetinaNet, Faster R-CNN, and Cascade R-CNN while maintaining efficiency.

Generally, bounding box regression is trained to align nearby proposals to target objects. In a common design, the bounding box regression branch predicts the offsets of the centers $(\delta x, \delta y)$ together with the relative scaling factor $(\delta w, \delta h)$ based on the features of RoI (Region of Interest). Whereas this design has been shown to be quite effective in previous works, it remains very difficult to *precisely* predict the location of an object when there exists a displacement, with large variance, between the anchor and the target. This

difficulty also limits the overall detection performance.

In recent years, various efforts have been devoted to improving the localization precision, such as cascading the localization process [1, 10, 19], iteratively performing RoI pooling and bounding box regression [12, 11, 35], and treating localization as a procedure to segment grid points [28]. While showing effectiveness in boosting the accuracy of localization, adoption of these methods complicate the detection pipeline, resulting in considerable computational overhead.

In this work, we aim to explore a new approach to object localization that can effectively tackle the difficulty of *precise localization* with lower overhead. Empirically, we observe that when we manually annotate a bounding box for an object, it is often much easier to align each side of the box to the object boundary than to move the box as a whole while tuning the size. Inspired by this observation, we propose a new design, named as **Side-Aware Boundary Localization (SABL)**, where each side of the bounding box is respectively positioned based on its surrounding context. As shown in Figure 1, to improve the localization precision, we devise *bucketing* scheme with two steps. For each side of a bounding box, this scheme divides the target space into multiple *buckets* and then takes two steps to determine its location. Particularly, it first searches for the *correct* bucket, *i.e.*, the one in which the boundary resides, using its centerline as a coarse estimate, and then refines its location by further predicting the offset. This scheme allows very precise localization even in the presence of displacements with large variance. Moreover, to preserve those precisely localized bounding boxes in the non-maximal suppression procedure, we also propose to adjust the classification score based on the bucketing confidences, which leads to further performance gains.

We test the proposed SABL upon various detection frameworks, including two-stage [35], single-stage [24], and cascade [1]. By replacing the bounding box regression branch with the proposed design, we achieves significant improvements while remaining efficiency, *i.e.* 41.8% vs. 38.8% $AP_{box}$, 39.7% vs. 38.1% $AP_{box}$ and 43.3% vs. 42.4% $AP_{box}$ on top of Faster R-CNN, RetinaNet, and Cascade R-CNN, respectively. Moreover, we integrate SABL into Hybrid Task Cascade (HTC) [2], the winning method of COCO Challenge 2018. With bells and whistles, we achieve 57.8% $AP_{box}$ and 51.3% $AP_{mask}$ on *test-dev*, which outperforming the 2018 winning entry by 1.7% $AP_{box}$ and 2.3% $AP_{mask}$, respectively.

## 2. Related Work

**Object Detection.** Recent years have witnessed a dramatic improvement in object detection [4, 9, 14, 36, 42]. The two-stage pipeline [11, 35] has been the leading paradigm in this area. The first stage generates a set of region proposals, and then the second stage classifies and refines the coordinates of proposals by bounding box regression. This design is widely adopted in later two-stage methods [7, 15]. Compared to two-stage approaches, the single-stage pipeline [24, 27, 33, 34] predicts bounding boxes directly. Despite omission of the proposal generation process, single-stage methods [24, 27] require densely distributed anchors produced by sliding window. Recently, some works attempt to use anchor-free methods [20, 22, 37, 40] for object detection. Intuitively, iteratively performing the classification and regression process could effectively improve the detection performance. Therefore, many attempts [1, 10, 19, 29] apply cascade architecture to regress bounding boxes iteratively for progressive refinement.

**Object Localization.** Object localization is one of the crucial and fundamental modules for object detection. A common approach for object localization is to regress the coordinates of a bounding box [7, 11, 27, 35, 39]. This approach is widely adopted in object localization, yet the precision is unsatisfactory due to the large variance of regression target. Aiming for a more accurate localization, Uncertainty [5, 17] or IoU prediction branch [19] is proposed to improve localization accuracy. And some methods [1, 19, 29] directly repeat the bounding box regression multiple times to further improve accuracy. However, such cascading pipeline expenses much more computational overhead to repeatedly adopt RoI pooling and bounding box regression.

There are some methods that try to reformat the object localization process. Grid R-CNN [28] adopts a grid localization mechanism to encode more clues for accurate object detection. It deals with localization as a procedure to segment grid points, which involves a heavy mask prediction process. LocNet [10] predicts probabilities for object borders or locations inside the object's bounding box. However, the resolution of RoI features limits the performance of LocNet because it needs to transfer the probability of pixels into the bounding box location. An iterative localization manner is also adopted in LocNet and restricts its efficiency. On the contrary, our method focuses on the boundaries of object bounding box and composes the localization process for each boundary with an effective and efficient coarse-to-fine bucketing scheme. We also leverage the bucketing estimation confidence to improve the classification results. Adopting RoI-pooling in one pass, SABL achieves substantial gains for object detection. Furthermore, except for two-stage or cascading pipelines, SABL shows its effectiveness on the single-stage pipeline.

## 3. Side-Aware Boundary Localization

Accurate object localization is crucial for object detection. Most current methods directly regress the normalized displacements between proposals and ground-truth boxes. However, this paradigm may not provide satisfactory lo-

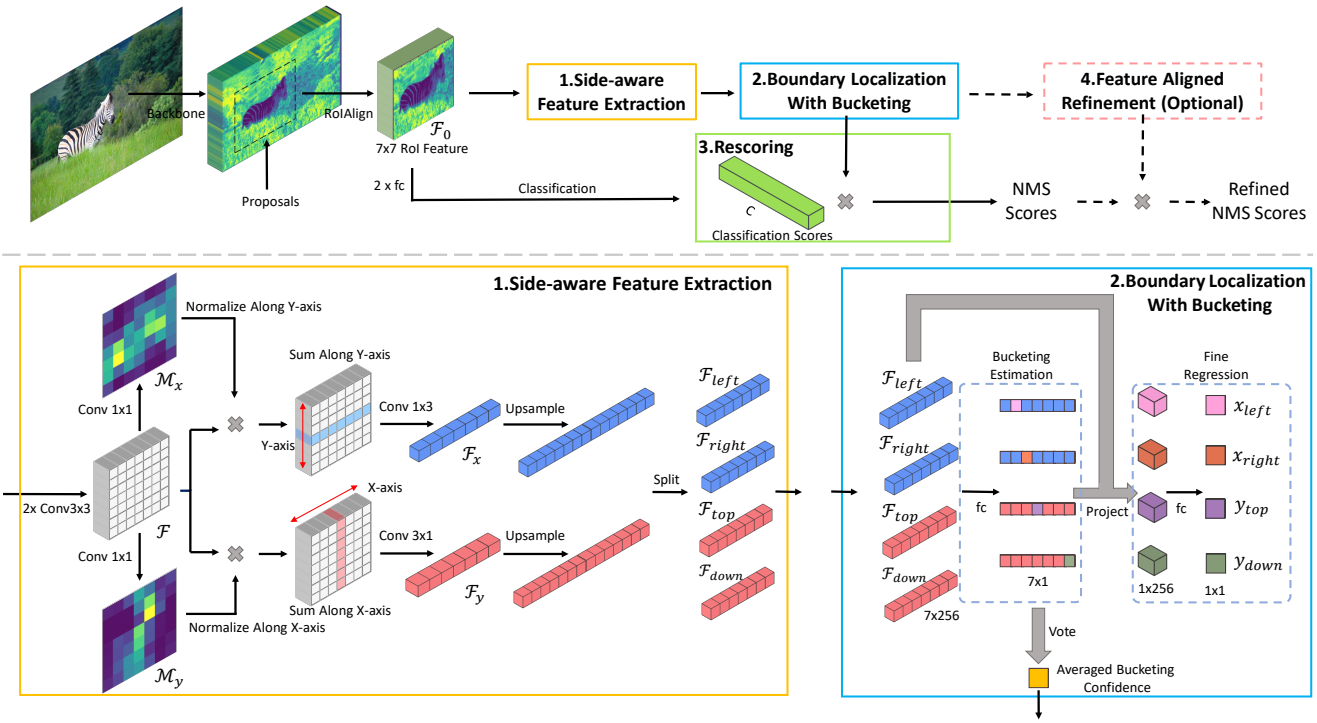



Figure 2: Pipeline of Side-Aware Boundary Localization for the two-stage detector. First, RoI features are aggregated to produce side-aware features in the Side-Aware Feature Extraction module. Second, the Boundary Localization with Bucketing module is performed to localize the boundaries by a two-step *bucketing scheme*. Each boundary is first coarsely estimated into buckets and then finely regressed to more precise localization. Third, the confidences of buckets are adopted to assist the classification scores. Fourth, an optional refinement module is proposed to further improve the localization precision.

calization results in one pass. Some methods [1, 19] attempt to improve localization performance with a cascading pipeline while imposing considerable computational costs. A lightweight as well as effective approach thus becomes necessary.

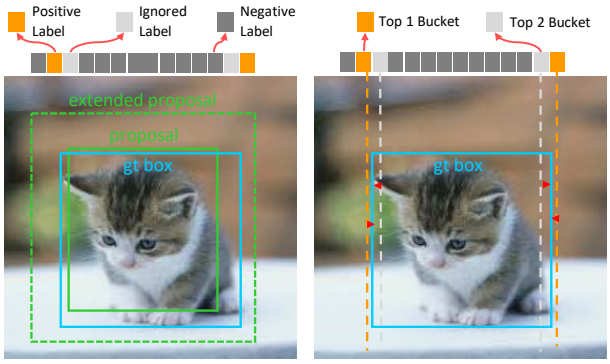
We propose Side-Aware Boundary Localization (SABL) as an alternative for the conventional bounding box regression to locate the objects more accurately. As shown in Figure 2, it first extracts horizontal and vertical features $\mathcal{F}_x$ and $\mathcal{F}_y$ by aggregating the RoI features $\mathcal{F}$ along X-axis and Y-axis, respectively, and then splits $\mathcal{F}_x$ and $\mathcal{F}_y$ into side-aware features $\mathcal{F}_{left}$, $\mathcal{F}_{right}$, $\mathcal{F}_{top}$ and $\mathcal{F}_{down}$. (Section 3.1). Then for each side of a bounding box, SABL first divides the target space into multiple buckets (as shown in Figure 1) and searches for the one where the boundary resides via leveraging the side-aware features. It will refine the boundary location $x_{left}$, $x_{right}$, $y_{top}$ and $y_{down}$ by further predicting their offsets from the bucket's centerline (Section 3.2). Such a two-step *bucketing scheme* could reduce the regression variance and ease the difficulties of prediction. Furthermore, the confidence of estimated buckets could also help to adjust the classification scores and further improve the performance (Section 3.3). The boundary localization results could be further refined by adding an optional boundary refinement module (Section 3.4). Moreover, SABL is also applicable for single-stage detectors with minor modifications (Section 3.5).

### 3.1. Side-Aware Feature Extraction

As shown in Figure 2, we extract side-aware features $\mathcal{F}_{left}$, $\mathcal{F}_{right}$, $\mathcal{F}_{top}$, and $\mathcal{F}_{down}$ based on the $k \times k$ RoI features $\mathcal{F}$ ($k = 7$). Following typical conventions [11, 35, 15], we adopt RoIAlign to obtain the RoI feature of each proposal. Then we utilize two $3 \times 3$ convolution layers to transform it to $\mathcal{F}$. To better capture direction-specific information of the RoI region, we employ the self-attention mechanism to enhance the RoI feature. Specifically, we predict two different attention maps from $\mathcal{F}$ with a $1 \times 1$ convolution, which are then normalized with a softmax function along the Y-axis and X-axis, respectively. Taking the attention maps $\mathcal{M}_x$ and $\mathcal{M}_y$, we aggregate $\mathcal{F}$ to obtain $\mathcal{F}_x$ and $\mathcal{F}_y$ as follows,

$$\begin{aligned} \mathcal{F}_x &= \sum_y \mathcal{F}(y,:) * \mathcal{M}_x(y,:), \\ \mathcal{F}_y &= \sum_x \mathcal{F}(:,x) * \mathcal{M}_y(:,x). \end{aligned} \tag{1}$$

$\mathcal{F}_x$ and $\mathcal{F}_y$ are both a 1-D feature map of shape $1 \times k$ and $k \times 1$, respectively. They are further refined by a $1 \times 3$ or




(a) Bucketing Estimation Label (b) Fine Regression Target

Figure 3: The localization target of SABL for bucketing estimation and fine regression on X-axis. The localization target for Y-axis can be calculated similarly.

$3 \times 1$ convolution layer and upsampled by a scale factor of 2 through a deconvolution layer, resulting in $1 \times 2k$ and $2k \times 1$ features on the horizontal and vertical direction, respectively. Finally, the upsampled features are simply split into two halves, leading to the side-aware features $\mathcal{F}_{left}$, $\mathcal{F}_{right}$, $\mathcal{F}_{top}$ and $\mathcal{F}_{down}$.

### 3.2. Boundary Localization with Bucketing

As shown in the module 2 of Figure 2, we decompose the localization process into a two-step coarse-to-fine *bucketing scheme*: bucketing estimation and fine regression. The candidate region of each object boundary is divided into buckets horizontally and vertically. We first estimate in which bucket the boundary resides and then regress a more accurate boundary localization from this bucket.

**Two-Step Bucketing Scheme.** Given the proposal box $(B_{left}, B_{right}, B_{top}, B_{down})$, we relax the candidate region of boundaries by a factor of $\sigma$ ($\sigma > 1$), with the aim to cover the entire object. The candidate regions are divided into $2k$ buckets on both X-axis and Y-axis, with $k$ buckets corresponding to each boundary. The width of each bucket on X-axis and Y-axis are therefore $l_x = (\sigma B_{right} - \sigma B_{left})/2k$ and $l_y = (\sigma B_{down} - \sigma B_{top})/2k$, respectively. In the coarse step, we adopt a binary classifier to predict whether the boundary is located in or is the closest to the bucket on each side, based on the side-aware features. In the fine step, we apply a regressor to predict the offset from the centerline of the selected bucket and the ground-truth boundary.

**Localization Targets.** There are a coarse estimation and a fine regression branch in the *bucketing scheme* to be trained. We follow the conventional methods [1, 35] for label assigning and proposal sampling. The coarse estimation determines the nearest buckets to the boundaries of a ground-truth bounding box by binary classification. As shown in Figure 3, on each side, the bucket, whose centerline is the nearest to the ground-truth boundary, is labeled as 1 (positive sample), while the others are labeled as 0 (negative samples). To reduce the ambiguity in training, on each side, we ignore the bucket that is the second nearest to the ground-truth boundary because it is hard to be distinguished from the positive one. Similar to the conventional bounding box regression methods [1, 35] that filter out negative RoIs for training, for each side, we ignore negative buckets during training the boundary regressor. To increase the robustness of the fine regression branch, we include both the nearest (labeled as "positive" in the coarse estimation step) bucket and the second nearest (labeled as "ignore" in the coarse estimation step) bucket to train the regressor. The regression target is the displacement between the bucket centerline and the corresponding ground-truth boundary. To ease the training difficulties of regressors, we normalize the target by $l_x$ and $l_y$ on the corresponding axes.

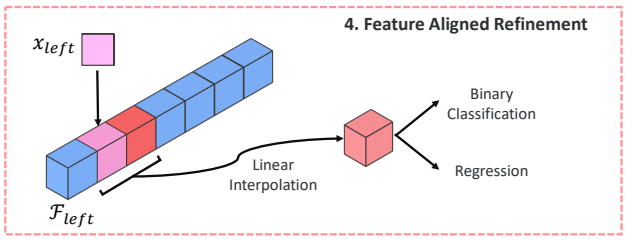



Figure 4: Pipeline of Feature-Aligned Refinement (FAR). Here we take the left boundary of the object for instance.

### 3.3. Bucketing-Guided Rescoring

The bucketing scheme brings a natural benefit, *i.e.*, the coarse estimation confidences can represent the reliability of predicted locations. With the aim to keep the more accurately localized bounding boxes during non-maximal suppression (NMS), we utilize the localization reliability to guide the rescoring. Therefore, SABL naturally averages the bucketing estimation confidence scores of four boundaries. The multi-category classification scores are multiplied by the averaged confidences, and then used for ranking candidates during NMS. The rescoring helps maintain the best box with both high classification confidence and accurate localization.

### 3.4. Feature-Aligned Refinement

The localization results could be further improved by adopting an optional refinement module as shown in Figure 4. Directly cascading the localization head for several times [1, 2] could indeed improve the results but introduce large computational cost. We reuse the side-aware features obtained in the Side-Aware Feature Extraction module to save the computational overhead. Specifically, with the location prediction for each boundary, we could obtain the location-aligned features of the predicted position via linear interpolation on the corresponding side-aware features. A regressor is performed on those location-aligned features to attain a more precise boundary prediction. We also adopt a binary classification to further suppress the NMS scores of

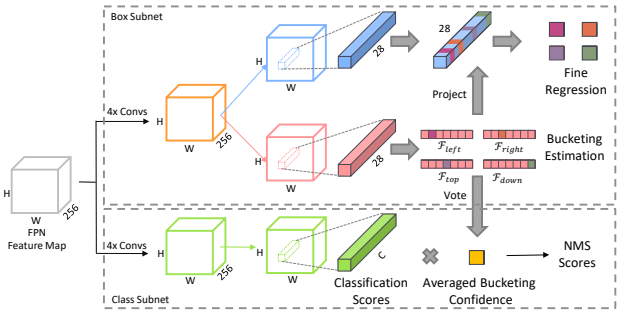



Figure 5: Pipeline of Side-Aware Boundary Localization (SABL) for Single-Stage Detector. Since there is no RoI features, SABL adopts convolution layers to produce feature vector for each location. Then bucketing estimation and fine regression are performed based on the feature vector. Furthermore, bucketing estimation confidence is leveraged to adjust the classification scores as well.

the less precisely localized bounding box. With this slight modification, SABL achieves the same performance with lower computational cost compared to Cascade R-CNN [1]. The proposals, whose IoU w.r.t. ground-truth boxes are bigger than a threshold (*e.g.*, 0.5), are assigned with positive labels in the Boundary Localization with Bucketing module. After localizing their boundaries by Boundary Localization with Bucketing, Feature-Aligned Refinement (FAR) trains two extra branches, *i.e.*, regression and binary classification, on these boundaries. For each side, we assign the boundary whose displacement to the corresponding ground-truth is less than the bucket width $l_x$ and $l_y$ for X-axis and Y-axis, respectively, as a positive sample. Others are assigned as negative samples. The regression target on each side is calculated as the displacement from the positive sample to the ground-truth, and then normalized by $l_x$ and $l_y$ on the corresponding axis.

### 3.5. Application to Single-Stage Detector

SABL could also be applied to single-stage detectors, such as [24], with minor modification. As shown in Figure 5, SABL for the single-stage detector exploits the feature vector at each position rather than the RoI features, which is different from the two-stage pipeline (Sec. 3.1). At each position on feature maps produced by FPN [23], SABL learns to predict and classify one bounding box based its corresponding anchor. The size of this anchor is $\gamma * s$, where $\gamma$ is a hyperparameter ($\gamma = 8$), and $s$ is the stride of the current feature map. To assign the target bounding box for each position, we follow the target assigning procedure in RetinaNet [24]. Specifically, nine anchors on each position are used to search for the ground-truth bounding box where the current location should be assigned to. We assign the ground-truth with the highest IoU to this location during training. This design enables SABL to cover more ground-truth bounding boxes and be better optimized. After using convolution layers to produce the feature vector for each position, the following Boundary Localization with Bucketing and Bucketing-Guided Rescoring remain the same.

## 4. Experiments

### 4.1. Experimental Setting

**Dataset.** We perform experiments on the challenging MS COCO 2017 benchmark [25]. We use the *train* split for training and report the performance on the *val* split for ablation study. Detection results for comparison with mainstream methods [1, 24, 35] are reported on the *test-dev* split.
**Implementation Details.** During training, We follow the 1x training scheduler [13] and use mmdetection [3] as codebase. We train Faster R-CNN [35], Cascade R-CNN [1] and RetinaNet [24] with batch size of 16 for 12 epochs. We apply an initial learning rate of 0.02 for Faster R-CNN, Cascade R-CNN, and 0.01 for RetinaNet. The learning rate will decrease by 0.1 after 8 and 11 epochs, respectively. FPN [23] is used in all experiments by default. The long edge and short edge of images are resized to 1333 and 800 respectively without changing the aspect ratio. The scale factor $\sigma$ is set as 1.7 and 3.0 for Faster R-CNN and RetinaNet, respectively. For Cascade R-CNN, we replace the original bbox head with the proposed SABL, and $\sigma$ for three cascading stages are set as 1.7, 1.5 and 1.3, respectively. $k$ is set as 7 for all experiments if not further specified. And Feature-Aligned Refinement is not applied by default to keep a clear pipeline in ablation study. Detection results are evaluated with the standard COCO metric. The runtime is measured on a single Tesla V100 GPU.
**Training Details.** The proposed framework is optimized in an end-to-end manner. For the two-stage pipeline, the RPN loss $\mathcal{L}_{rpn}$ and classification loss $\mathcal{L}_{cls}$ remain the same as Faster R-CNN [35]. We replace the bounding box regression loss by a bucketing estimation loss $\mathcal{L}_{bucketing}$ and a fine regression loss $\mathcal{L}_{reg}$. When adopting Feature-Aligned Refinement, an extra regression loss $\mathcal{L}_{reg}^{refine}$ and a binary classification loss $\mathcal{L}_{cls}^{refine}$ are involved. Specifically, $\mathcal{L}_{bucketing}$ and $\mathcal{L}_{cls}^{refine}$ adopt Binary Cross-Entropy Loss, $\mathcal{L}_{reg}$ and $\mathcal{L}_{reg}^{refine}$ use Smooth L1 Loss. In summary, a general loss function can be written as follows: $\mathcal{L} = \lambda_1 \mathcal{L}_{rpn} + \mathcal{L}_{cls} + \lambda_2(\mathcal{L}_{bucketing} + \mathcal{L}_{reg}) + \lambda_3(\mathcal{L}_{reg}^{refine} + \mathcal{L}_{cls}^{refine})$, where $\lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 0$ for two-stage pipeline, $\lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 1$ for two-stage pipeline with Feature-Aligned Refinement, $\lambda_1 = 0, \lambda_2 = 1.5, \lambda_3 = 0$ for single stage pipeline.

### 4.2. Results

We show the effectiveness of SABL by applying SABL on RetinaNet, Faster R-CNN and Cascade R-CNN. To be specific, we adopt SABL to Faster R-CNN and RetinaNet as described in Sec. 3. As shown in Table 1, with marginally increased computational cost, SABL improves

Table 1: Comparison to mainstream methods with ResNet-101 FPN backbone on COCO *test-dev* dataset.

| Method | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ | Speed (fps) |
|---|---|---|---|---|---|---|---|
| RetinaNet [24] | 38.1 | 58.5 | 40.8 | 21.2 | 41.5 | 48.2 | 12.8 |
| Faster R-CNN [35] | 38.8 | 60.9 | 42.3 | 22.3 | 42.2 | 48.6 | **13.8** |
| Grid R-CNN [28] | 41.4 | 60.1 | 44.9 | 23.4 | 44.8 | 52.3 | 11.1 |
| Cascade R-CNN [1] | 42.4 | 61.1 | 46.1 | 23.6 | 45.4 | 54.1 | 11.2 |
| RetinaNet w/ SABL | 39.7 | 58.0 | 42.6 | 21.8 | 43.5 | 50.6 | 12.3 |
| Faster R-CNN w/ SABL | 41.8 | 60.2 | 45.0 | 23.7 | 45.3 | 52.7 | 12.2 |
| Faster R-CNN w/ SABL+FAR | 42.4 | 60.3 | 45.6 | 24.0 | 46.0 | 53.5 | 11.8 |
| **Cascade R-CNN w/ SABL** | **43.3** | 60.9 | 46.2 | 23.8 | 46.5 | 55.7 | 8.8 |

Table 2: The effects of each module in our design. 'SAFE', 'BLB', 'LGR', 'FLR' denote Side-Aware Feature Extraction, Boundary Localization with Bucketing, Bucketing-Guided Rescoring, and Feature-Aligned Refinement, respectively.

| SAFE | BLB | BGR | FAR | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|---|---|
| | | | | 37.3 | 58.2 | 40.4 | 22.0 | 41.2 | 47.8 |
| ✓ | | | | 38.5 | 58.2 | 41.6 | 23.0 | 42.5 | 49.5 |
| | ✓ | | | 38.3 | 57.6 | 40.5 | 22.3 | 42.6 | 49.7 |
| | ✓ | ✓ | | 39.0 | 57.5 | 41.9 | 22.6 | 43.2 | 50.9 |
| ✓ | ✓ | | | 39.0 | 57.3 | 41.3 | 22.0 | 43.0 | 50.8 |
| ✓ | ✓ | ✓ | | 39.7 | 57.5 | 42.5 | 22.3 | 43.7 | 51.8 |
| ✓ | ✓ | ✓ | ✓ | 40.1 | 57.6 | 43.1 | 23.4 | 44.3 | 51.9 |

performances of RetinaNet, Faster R-CNN and Cascade R-CNN by 1.6%, 3.0% and 0.9% $AP_{box}$ respectively. Applying the Feature Aligned Refinement as in Sec. 3.4 further improves the performance of Faster R-CNN with SABL by 0.6% $AP_{box}$. The significant performance gains on various object detection architectures show that SABL is a generally efficient and effective bounding box localization method for object detection.

### 4.3. Ablation Study

**Model Design.** We omit different components of SABL on two-stage pipeline to investigate the effectiveness of each component, including Side-Aware Feature Extraction (SAFE), Boundary Localization with Bucketing (BLB), Bucketing-Guided Rescoring (BGR) and Feature-Aligned Refinement (FAR) in Table 2. The official implementation of Faster R-CNN [35] adopts bounding box center offsets and scale factors of size, *i.e.*, $(\delta x, \delta y, \delta w, \delta h)$ as regression targets. In Table 2, we report the performance of a modified Faster R-CNN which replaces the original regression target with offsets of each boundary $(\delta x_1, \delta y_1, \delta x_2, \delta y_2)$. This simple modification improves the performance from 36.4% to 37.3%, which shows localization by each boundary is better than regressing the box as a whole.

*Side-Aware Feature Extraction (SAFE).* In the second row of Table 2, we apply Side-Aware Feature Extraction (SAFE) as described in Sec. 3.1. SAFE focuses on content of the corresponding side and significantly improves the performance from 37.3% to 38.5%. To show that simply adding more parameters will not apparently improve the performance, we also train a Faster RCNN with boundary regression and 4conv1fc head. The 4conv1fc head has four 3x3 convolution layers followed by one fully-connected layer and marginally improves the $AP$ by 0.1%, *i.e.*, from 37.3% to 37.4%.

Table 3: Number of convolution layers for Side-Aware Feature Extraction. 2D Conv indicates the number of 3x3 Convolution layers before $\mathcal{F}$. 1D Conv indicates the number of 1x3 and 3x1 convolution layers before $\mathcal{F}_x$ and $\mathcal{F}_y$, respectively.

| 2D Conv | 1D Conv | AP | 2D Conv | 1D Conv | AP |
|---|---|---|---|---|---|
| 0 | 0 | 23.5 | 2 | 0 | 39.5 |
| 0 | 1 | 38.3 | **2** | **1** | 39.7 |
| 0 | 2 | 38.3 | 2 | 2 | 39.6 |
| 0 | 3 | 38.5 | 2 | 3 | 39.7 |
| 1 | 0 | 36.1 | 3 | 0 | 39.7 |
| 1 | 1 | 39.3 | 3 | 1 | 39.7 |
| 1 | 2 | 39.4 | 3 | 2 | 39.7 |
| 1 | 3 | 39.4 | 3 | 2 | 39.7 |

*Boundary Localization with Bucketing (BLB).* As described in Sec. 3.2, BLB divides the RoI into multiple buckets, it first determines which bucket the boundary resides and takes the centerline of the selected bucket as coarse estimation of boundary. Then it performs fine regression to localize the real boundary precisely. BLB achieves 38.3%, which outperforms the method of directly regressing the four boundaries by 1.0%. Combining BLB with SAFE further achieves 39.0%, which is 0.5% higher than only adopting SAFE.

*Bucketing-Guided Rescoring (BGR).* Bucketing-Guided Rescoring (BGR) is proposed to adjust the classification scores as in Sec. 3.3. The bucketing confidence can naturally be used to represent how confident the model believes a boundary is precisely localized. We average the confidence of selected buckets for four boundaries and multiply it to classification scores before NMS. Applying the BGR on BLB and SAFE + BLB further improves the performance by 0.7% (39.0% *vs*. 38.3% and 39.7% *vs*. 39.0%).

*Feature-Aligned Refinement (FAR).* Feature-Aligned Refinement (FAR) is an optional component in our work. As

Table 4: Comparison of different methods to aggregate the 2D RoI features into 1D features.

| Aggregating Method | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|
| Max Pooling | 39.4 | 57.5 | 42.1 | 22.3 | 43.5 | 51.2 |
| Average Pooling | 39.3 | 57.6 | 42.1 | 23.0 | 43.6 | 50.7 |
| Attention Mask | 39.7 | 57.5 | 42.5 | 22.3 | 43.7 | 51.8 |

Table 5: Comparison of different settings of feature size in Side-Aware Feature Extraction module.

| RoI | Upsample | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|
| 7 | 7 | 39.0 | 57.8 | 41.9 | 23.6 | 42.9 | 50.8 |
| **7** | **14** | 39.7 | 57.5 | 42.5 | 22.3 | 43.7 | 51.8 |
| 7 | 28 | 39.1 | 56.5 | 42.1 | 22.0 | 43.7 | 51.1 |
| 14 | 14 | 39.7 | 57.5 | 42.5 | 22.8 | 44.2 | 50.9 |

described in Sec. 3.4, FAR could help further increase the localization precision. The last row in Table 2 shows FAR can improve the strong performance from 39.7% to 40.1%.

**Side-Aware Feature Extraction.** Side-Aware Feature Extraction (SAFE) is used to aggregate the 2D RoI features to 1D features for X-axis and Y-axis, respectively. Here we perform thorough ablation study for SAFE.

*Parameters.* In SAFE, after performing the RoI pooling, we apply two 3x3 convolution layers to obtain $\mathcal{F}$. One 1x3 and 3x1 convolution layers are adopted after aggregating the 1D features on horizontal and vertical directions to obtain $\mathcal{F}_x$ and $\mathcal{F}_y$, respectively. We investigate the influence of these convolution layer numbers. As shown in Table 3, since less parameters harm the performance and more parameters do not bring extra gains, using two 3x3 convolution layers for $\mathcal{F}$, one 1x3 convolution layer for $\mathcal{F}_x$ and one 3x1 convolution layer for $\mathcal{F}_y$ exhibits a good trade-off between performance and efficiency.

*Feature aggregating method.* We adopt two 1x1 convolution layers to predict the attention masks for X-axis and Y-axis, respectively, and aggregate the RoI features with the normalized attention masks. The max pooling and average pooling are two alternative approaches in this procedure. In Table 4, experimental results reveal that attention masks are more effective than max or average pooling to aggregate RoI features. This attention-guided feature aggregation scheme could adaptively exploit RoI features along one direction with predictable weights, while the rule-based max pooling and average pooling could not.

*Size of Side-Aware Features.* In the Side-Aware Feature Extraction module, we first perform RoI-Pooling and get the RoI features with spatial size $7 \times 7$. The RoI features are aggregated into 1D features with size $1 \times 7$ and $7 \times 1$, and then upsampled to size $1 \times 14$ and $14 \times 1$ by a deconvolution layer. We study RoI features size and upsampled features size. As shown in Table 5, our settings, *i.e.*, RoI size of 7 and upsampled size of 14, achieve the best trade-off between effectiveness and efficiency. A larger or smaller upsampled size will harm the performance. A larger RoI size will not improve the performance meanwhile bring more computational cost to process a larger RoI features.

Table 6: Influence of different localization pipelines. To clarify the effectiveness of Bucketing, **we don't apply the rescoring module**.

| Localization Approach | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|
| Regression | 38.5 | 58.2 | 41.6 | 23.0 | 42.5 | 49.5 |
| Bucketing | 33.8 | 56.9 | 37.6 | 20.5 | 37.8 | 42.5 |
| Iterative Bucketing | 37.6 | 58.5 | 42.1 | 21.4 | 41.1 | 49.2 |
| Bucketing + Regression | 39.0 | 57.3 | 41.3 | 22.0 | 43.0 | 50.8 |

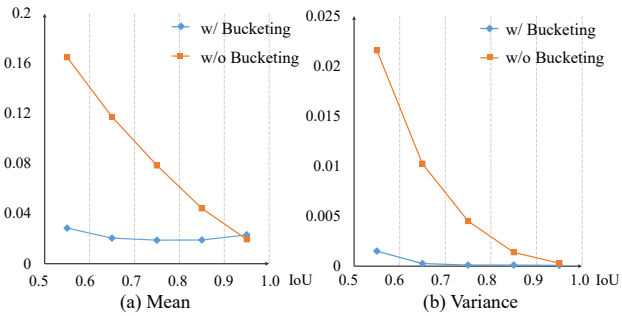



Figure 6: Mean and variance of displacements from proposals to ground-truth boundaries w.r.t. size of the ground-truth boxes with or without bucketing.

**Boundary Localization with Bucketing.** Here we discuss the effectiveness of different designs for localization. In our work, we propose Boundary Localization with Bucketing, which contains a bucketing estimation step and a fine regression step. As shown in Table 6, based on the features provided by Side-Aware Feature Extraction. The proposed 'Bucketing + Regression' achieves better performance than solely adopting 'Regression' or 'Bucketing'. Here 'Regression' indicates applying bounding box regression with boundary target $(\delta x_1, \delta y_1, \delta x_2, \delta y_2)$, and 'Bucketing' indicates to use the centerline of the predicted bucket for each boundary as localization results. It is noteworthy that only applying 'Bucketing' exhibits much inferior performance. The localization precision of 'Bucketing' is limited by the horizontal and vertical bucket numbers, because there exists an inevitable localization error within the width of one bucket. Following LocNet [10], we also try to perform a heavy 'Iterative Bucketing' pipeline where 'Bucketing' step is performed iteratively. Although the performance is improved from 33.8% to 37.6%, it remains inferior to 39.0% of 'Bucketing + Regression'. Figure 6 shows mean and variance of displacements from proposals to ground-truth boxes w.r.t. the size of ground-truth boxes. Without loss of generality, we choose the left boundary to calculate the statistic. The proposals are split into five groups according to their IoU w.r.t. ground-truth, *i.e.*, [0.5, 0.6), [0.6, 0.7), [0.7, 0.8), [0.8, 0.9), [0.9, 1.0). Regression with bucketing exhibits more stable distribution on displacements, which could ease the difficulties of regression and lead to more precise local-

Table 7: Influence of different designs to generate regression targets. 'Ignore' indicates to set the second nearest bucket as ignore label during training the bucketing estimation branch. 'Top2-Reg' denotes to train both the nearest and second nearest centerline of buckets for regression branch.

| Ignore | Top2-Reg | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|
| | | 38.7 | 57.4 | 41.8 | 22.5 | 42.9 | 50.1 |
| ✓ | | 39.1 | 57.6 | 42.3 | 22.0 | 43.3 | 50.9 |
| | ✓ | 39.4 | 57.3 | 42.1 | 22.7 | 43.8 | 50.7 |
| ✓ | ✓ | 39.7 | 57.5 | 42.5 | 22.0 | 43.0 | 50.8 |

Table 8: Influence of different factors $\sigma$ to rescale the basic bounding box to generate localization targets in Faster RCNN w/ SABL.

| $\sigma$ | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|
| 1.1 | 39.2 | 57.6 | 41.9 | 22.9 | 43.3 | 51.1 |
| 1.3 | 39.4 | 57.7 | 42.0 | 22.1 | 43.6 | 51.6 |
| 1.5 | 39.6 | 57.6 | 42.4 | 22.8 | 43.8 | 52.2 |
| **1.7** | 39.7 | 57.5 | 42.5 | 22.3 | 43.7 | 51.8 |
| 1.9 | 39.6 | 57.9 | 42.5 | 22.6 | 43.7 | 51.2 |

ization.

*Target design*. We further study the training targets designs for this module. 1) During training bucketing estimation branch, we ignore the second nearest bucket to ease its ambiguity with the nearest bucket. 2) During training the fine regression branch, buckets with Top-2 displacements to the ground-truth boundaries are trained to reduce the influence of mis-classification in bucketing estimation. As shown in Table 7, two proposed designs bring substantial performance gains. In our study, the classification accuracy of Top-1, Top-2 and Top-3 buckets are 69.3%, 90.0% and 95.7%, respectively. Therefore, we also try to train with Top-3 regression targets, however the final detection performance remains 39.7%.

*Scale factor*. We then study the influence of different scale factors $\sigma$ to extend proposals during generating localization targets. In Faster R-CNN, proposals, whose IoU w.r.t. ground-truth bounding box are larger than 0.5, are used as positive samples. Therefore, $\sigma$ is set as 1.7 to cover the whole objects. As shown in Table 8, 1.7 achieves the best performance. A smaller scale factor will result in difficulties for bucketing, since the boundary may locate outside the buckets. A larger scale factor increases the width of buckets and also shows inferior performance.

**Bucketing-Guided Rescoring.** In Sec. 3.3, we describe the rescoring scheme which leverages the estimation confidence of buckets to guide the classification scores. Table 2 shows it can help improve the performance from 39.0% to 39.7% without extra cost. Here we study its effectiveness on different IoU threshold. Experimental results show it improves $AP_{60}$, $AP_{70}$, $AP_{80}$ and $AP_{90}$ by 0.2%, 1.0%, 1.7% and 0.3%, respectively. There are significant performance gains on relatively high IoU threshold, *i.e.*, $AP_{70}$ and $AP_{80}$. The gains on $AP_{90}$ are limited, because the bucketing scores are just coarse estimation of location. It could not represent the localization very precisely.

Table 9: Effectiveness of different branches in Feature-Aligned Refinement.

| Reg | Binary-Cls | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|
| | | 39.7 | 57.5 | 42.5 | 22.3 | 43.7 | 51.8 |
| ✓ | | 39.8 | 57.5 | 42.6 | 22.4 | 43.9 | 51.9 |
| | ✓ | 39.9 | 57.6 | 42.8 | 22.5 | 44.0 | 52.5 |
| ✓ | ✓ | 40.1 | 57.6 | 43.1 | 23.4 | 44.3 | 51.9 |

Table 10: Influence of different factors in RetinaNet w/ SABL, *i.e.*, scale factor, buckets number and localization loss weight.

| $\sigma$ | Bucket-Num | Weight | AP | $\sigma$ | Bucket-Num | Weight | AP |
|---|---|---|---|---|---|---|---|
| 2 | 7 | 1.5 | 37.3 | 3 | 9 | 1.5 | 37.2 |
| **3** | **7** | **1.5** | 37.4 | 3 | 7 | 0.5 | 35.9 |
| 4 | 7 | 1.5 | 36.9 | 3 | 7 | 0.75 | 36.5 |
| 5 | 7 | 1.5 | 36.5 | 3 | 7 | 1.0 | 36.8 |
| 3 | 3 | 1.5 | 35.7 | 3 | 7 | 1.25 | 37.2 |
| 3 | 5 | 1.5 | 36.9 | 3 | 7 | 1.75 | 37.2 |

**Feature-Aligned Refinement.** We also study the effectiveness of regression and binary classification branches which are described in Sec. 3.4. We evaluate these two branches in Table 9. The results suggest that jointly adopting two branches in Feature-Aligned Refinement can substantially improve the performance from 39.7% to 40.1%.

**SABL for Single-Stage Detector.** For single-stage detector, we take RetinaNet [24] as baseline. In our implementation, we take the bounding box with size of $\gamma * s$ as the basic box for each location. Following RetinaNet, $\gamma$ is 8 and $s$ is the stride of the current feature map. As in two-stage method, we scale proposals by a factor $\sigma$ to generate training targets. Since the single-stage pipeline is different for the two-stage one, we study the suitable scale factors for RetinaNet. The results in Table 10 shows that 3.0 achieves the best result.

Then we study the bucket numbers for the boundary features *i.e.*, how many buckets are necessary to represent the boundaries for accurate localization. The results of different bucket numbers in Table 10 shows that 7 is the best choice.

In our implementation, we also find that the loss weight of localization branch is crucial for superior performance. We perform extensive study for localization loss weight to find the most suitable one. Table 10 shows that adopting localization loss weight of 1.5 achieves the best performance.

**Stronger Results with bells and whistles.** The proposed Side-Aware Boundary Localization (SABL) can be integrated into Hybrid Task Cascade (HTC) [2], which is a state-of-the-art instance segmentation method. With bells and whistles, HTC w/ SABL performs 57.8% $AP_{box}$ and 51.3% $AP_{mask}$ *vs*. 56.0 % $AP_{box}$ and 49.0% $AP_{mask}$ achieved by the winning entry of COCO Detection Challenge 2018 on COCO test-dev.

## 5. Conclusion

In this work, we propose **Side-Aware Boundary Localization (SABL)** to replace the conventional bounding box regression. We extract side-aware features which focus on the content of boundaries for localization. A lightweight two-step *bucketing scheme* is proposed to locate objects accurately based on the side-aware features. We also introduce a rescoring mechanism to leverage the bucketing confidence to keep high-quality bounding boxes. The proposed SABL exhibits consistent and significant performance gains on various object detection pipelines.

## References


[1] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: delving into high quality object detection. In *CVPR*, 2018. 1, 2, 4, 5, 6

[2] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Hybrid task cascade for instance segmentation. In *CVPR*, 2019. 2, 4, 8, 11

[3] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 5

[4] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. DetNAS: Backbone search for object detection. *CoRR*, abs/1903.10979, 2019. 2

[5] Jiwoong Choi, Dayoung Chun, Hyun Kim, and Hyuk-Jae Lee. Gaussian YOLOv3: An accurate and fast object detector using localization uncertainty for autonomous driving. In *ICCV*, 2019. 2

[6] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019. 11

[7] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In *NIPS*, 2016. 2

[8] Haoshu Fang, Jianhua Sun, Runzhong Wang, Minghao Gou, Yonglu Li, and Cewu Lu. InstaBoost: Boosting instance segmentation via probability map guided copy-pasting. *CoRR*, abs/1908.07801, 2019. 11

[9] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. NAS-FPN: learning scalable feature pyramid architecture for object detection. *CoRR*, abs/1904.07392, 2019. 2, 11

[10] Spyros Gidaris and Nikos Komodakis. LocNet: Improving localization accuracy for object detection. In *CVPR*, 2016. 2, 7

[11] Ross Girshick. Fast R-CNN. In *ICCV*, 2015. 2, 3

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 2

[13] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018. 5

[14] Kaiming He, Ross Girshick, and Piotr Dollar. Rethinking ImageNet pre-training. In *ICCV*, 2019. 2

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017. 2, 3, 12

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 11

[17] Yihui He, Chenchen Zhu, Jianren Wang, Marios Savvides, and Xiangyu Zhang. Bounding box regression with uncertainty for accurate object detection. In *CVPR*, 2019. 2

[18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 11

[19] Borui Jiang, Ruixuan Luo, Jiayuan Mao, Tete Xiao, and Yuning Jiang. Acquisition of localization confidence for accurate object detection. In *ECCV*, 2018. 2

[20] Tao Kong, Fuchun Sun, Huaping Liu, Yuning Jiang, and Jianbo Shi. FoveaBox: Beyond anchor-based object detector. *CoRR*, abs/1904.03797, 2019. 2

[21] Tao Kong, Fuchun Sun, Chuanqi Tan, Huaping Liu, and Wenbing Huang. Deep feature pyramid reconfiguration for object detection. In *ECCV*, 2018. 11

[22] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *ECCV*, 2018. 2

[23] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 5, 11, 12

[24] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 1, 2, 5, 6, 8

[25] 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 *ECCV*, 2014. 5, 12

[26] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018. 11, 12

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. In *ECCV*, 2016. 2

[28] Xin Lu, Buyu Li, Yuxin Yue, Quanquan Li, and Junjie Yan. Grid R-CNN. In *CVPR*, 2019. 2, 6

[29] Mahyar Najibi, Mohammad Rastegari, and Larry S Davis. G-cnn: an iterative grid based object detector. In *CVPR*, 2016. 2

[30] Xingang Pan, Xiaohang Zhan, Jianping Shi, Xiaoou Tang, and Ping Luo. Switchable whitening for deep representation learning. *CoRR*, abs/1904.09739, 2019. 11

[31] Jiangmiao Pang, Kai Chen, Jianping Shi, Huajun Feng, Wanli Ouyang, and Dahua Lin. Libra R-CNN: Towards balanced learning for object detection. In *CVPR*, 2019. 11

[32] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. MegDet: A large mini-batch object detector. *CVPR*, 2018. 11

[33] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 2

[34] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. In *CVPR*, 2017. 2

[35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 2, 3, 4, 5, 6, 12

[36] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014. 2

[37] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. *CoRR*, abs/1904.01355, 2019. 2

[38] Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. CARAFE: content-aware reassembly of features. *CoRR*, abs/1905.02188, 2019. 11, 12

[39] Jiaqi Wang, Kai Chen, Shuo Yang, Chen Change Loy, and Dahua Lin. Region proposal by guided anchoring. In *CVPR*, 2019. 2

[40] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, and Chunhua Shen. NAS-FCOS: fast neural architecture search for object detection. *CoRR*, abs/1906.04423, 2019. 2

[41] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 11

[42] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2Det: a single-shot object detector based on multi-level feature pyramid network. In *AAAI*, 2019. 2

[43] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable ConvNets V2: More deformable, better results. In *CVPR*, 2019. 11, 12

## Appendix A. Extensions of SABL in COCO Challenge 2019

Here we demonstrate the whole system with bells and whistles in COCO Challenge 2019, which won the detection track of *no external data*. Applying SABL to Mask R-CNN with ResNet-50 [16] achieves 40.0% $AP_{box}$ and 35.0% $AP_{mask}$. Then we adopt Hybrid Task Cascade (HTC) [2] with the proposed CAFA (in Appendix B) in PAFPN [26] and CARAFE [38] in Mask Head. It achieves 44.3% $AP_{box}$ and 38.4% $AP_{mask}$ compared with 42.1% $AP_{box}$ and 37.3% $AP_{mask}$ of HTC baseline. Our overall system is trained without involving external instance-level annotated data during training. To be specific, it is trained on COCO2017 training split (instance segmentation and stuff annotations) as in [2]. Here we also list other steps and additional modules we used to obtain the final performance. The step-by-step gains brought by different components are illustrated in Table 11.

**SyncBN**. We use Synchronized Batch Normalization [26, 32] in the backbone and heads.

**SW**. We adopt Switchable Whitening (SW) [30] in the backbone and FPN following the original paper.

**DCNv2**. We appply Deformable Convolution v2 [43] in the last three stage (from res3 to res5) of the backbone.

**Multi-scale Training**. We adopt multi-scale training. The scale of short edge is randomly sampled from $[400, 1400]$ per iteration and the scale of long edge is fixed as 1600. The detectors are trained with 20 epoches and the learning rate is decreased by 0.1 after 16 and 19 epoches, respectively.

**SENet-154 with SW**. We tried different larger backbones. SENet-154 [18] with Switchable Whitening (SW) [30] achieves the best single model performance.

**Stronger Augmentation**. We adopt Instaboost [8] as the sixth policy of AutoAugment [6]. Each policy has the same probability to be used for data augmentation during training procedure. The detectors are trained with 48 epoches with such stronger augmentation, and the learning rate is decreased by 0.1 after 40 and 45 epoches, respectively.

**Multi-scale Testing**. We use 5 scales as well as horizontal flip at test time before ensemble. The testing scales are $(600, 900)$, $(800, 1200)$, $(1000, 1500)$, $(1200, 1800)$, $(1400, 2100)$.

**Ensemble**. We use ensemble of models based on five backbone networks. We pretrain SENet-154 w/ SW and SE-ResNext-101 w/ SW on ImageNet-1K image classification dataset and use pretrained weights of ResNeXt-101 32 $\times$ 32d, ResNeXt-101 32 $\times$ 16d [41] and ResNeXt-101 32 $\times$ 8d [41] provided by PyTorch[1].

As shown in Table 12, on COCO 2017 test-dev dataset, our method finally achieves 57.8% $AP_{box}$, 51.3% $AP_{mask}$ with multiple model ensemble and 56.0% $AP_{box}$, 49.4% $AP_{mask}$ with single model. Our result outperforms the 2018 COCO Winner Entry by 1.7% $AP_{box}$ and 2.3% $AP_{mask}$, respectively.

[1] https://pytorch.org/hub/facebookresearch_WSL-Images_resnext/

Table 11: Step by Step results of our method on COCO2017 *val* dataset.

| Methods | scheduler | $AP_{box}$ | $AP_{mask}$ |
|---|---|---|---|
| Mask R-CNN | 1x | 37.3 | 34.2 |
| + SABL | 1x | 40.0 (+2.7) | 35.0 (+0.8) |
| + HTC | 1x | 42.9 (+2.9) | 37.4 (+2.4) |
| + CAFA&CARAFE | 1x | 44.3 (+1.4) | 38.4 (+1.0) |
| + SyncBN | 1x | 45.8 (+1.5) | 39.9 (+1.5) |
| + SW | 1x | 46.1 (+0.3) | 40.0 (+0.1) |
| + Backbone DCNv2 | 1x | 48.2 (+2.1) | 41.7 (+1.7) |
| + Mask Scoring | 1x | 48.3 (+0.1) | 42.4 (+0.7) |
| + MS-Training | 20e | 50.2 (+1.9) | 44.5 (+2.1) |
| + SE154-SW | 20e | 52.7 (+2.5) | 46.1 (+1.6) |
| + AutoAug&InstaBoost | 4x | 54.0 (+1.3) | 47.1 (+1.0) |
| + Multi-Scale Testing | - | 55.3 (+1.3) | 48.4 (+1.3) |
| + Ensemble | - | **57.2** (+1.9) | **50.5** (+2.1) |

Table 12: Results with bells and whistles on COCO2017 *test-dev* dataset.

| Methods | $AP_{box}$ | $AP_{mask}$ |
|---|---|---|
| 2018 Winners Single Model | 54.1 | 47.4 |
| Ours Single Model | 56.0 | 49.4 |
| 2018 Winners Ensemble [2] | 56.0 | 49.0 |
| Ours | **57.8** | **51.3** |

## Appendix B. Content-Aware Feature Aggregation (CAFA)

Many studies [9, 21, 23, 26, 31] have investigated the architecture design for generating the feature pyramid. The approach for fusing low- and high-level features, however, remains much less explored. A typical way for fusing features across different scales is by naïve downsampling or upsampling followed by element-wise summation. While this method is simple, it ignores the underlying content of each scale during the fusion process.

Considering this issue, we propose the **Content-Aware Feature Aggregation** (CAFA) module that facilitates effective fusion and aggregation of multi-scale features in a feature pyramid. To encourage content-aware upsampling, we develop CAFA based on CARAFE [38], a generic operator which is proposed for replacing the conventional upsampling operator. Different from CARAFE, we perform Deformable Convolution v2 [43] after CARAFE while before summing up the upsampled feature maps with the lateral feature maps (see Figure 7). Further, the conventional convolution layers are replaced by Deformable Convolution

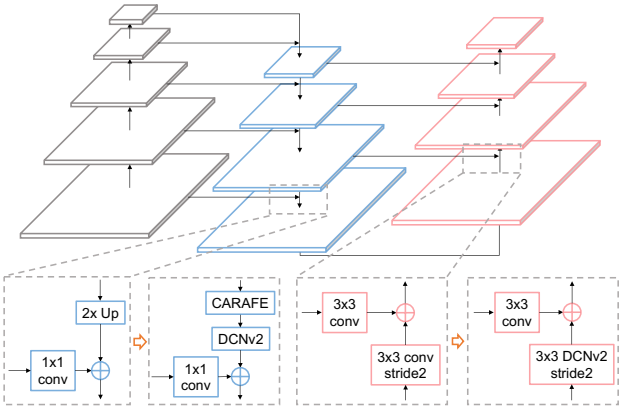



Figure 7: Modification by CAFA on PAFPN [26]. We use CARAFE [38] and DCNv2 [43] during upsampling and use DCNv2 [43] for downsampling.

Table 13: Effectiveness of CAFA combined with FPN [23] and PAFPN [26] in Mask R-CNN [15].

| Method | Modification | box AP | mask AP |
|---|---|---|---|
| FPN | Baseline | 37.3 | 34.2 |
| | + CAFA | 38.9 | 35.3 |
| PAFPN | Baseline | 37.7 | 34.3 |
| | + CAFA | 40.0 | 36.2 |

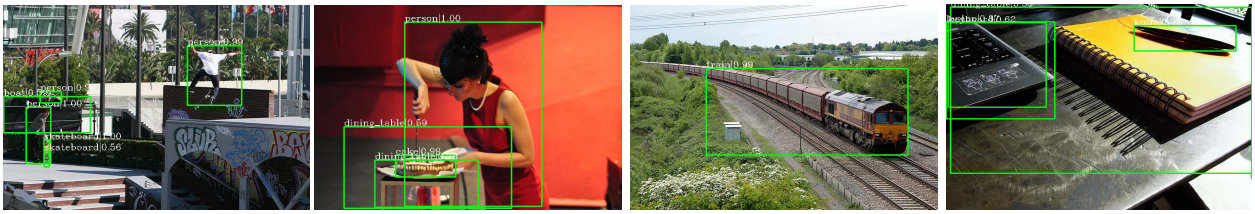
v2 [43] for downsampling. Thanks to this design, CAFA enjoys a larger receptive field and gains improved performance in adapting to instance-specific contents.

As shown in Table 13, we study the effectiveness of CAFA combined with FPN [23] and PAFPN [26] in Mask R-CNN [15]. CAFA brings 1.6% $AP_{box}$, 1.1% $AP_{mask}$ gains on FPN, and 2.3% $AP_{box}$, 1.9% $AP_{mask}$ gains on PAFPN, respectively.

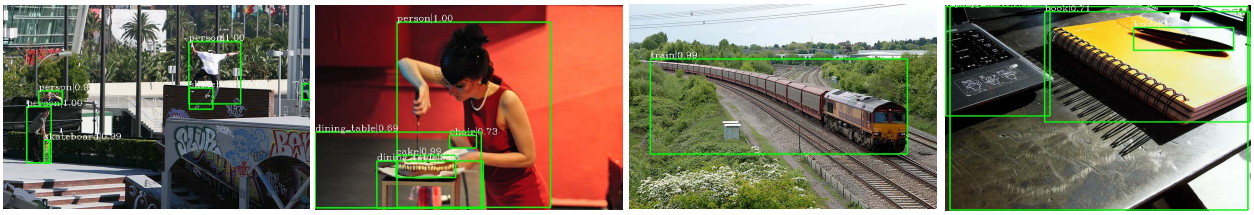
We further evaluate CAFA with NAS-FPN on RetinaNet and it achieves compatible results (39.2% $AP_{box}$ v.s. 39.5% $AP_{box}$ on COCO2017 *val* dataset at $640 \times 640$ scale). While NAS-FPN uses 7 pyramid networks, our CAFA with PAFPN only uses 2 pyramid networks with much simpler pathways (one top-down and one bottom-up) among pyramidal features.

## Appendix C. Visual Results Comparison

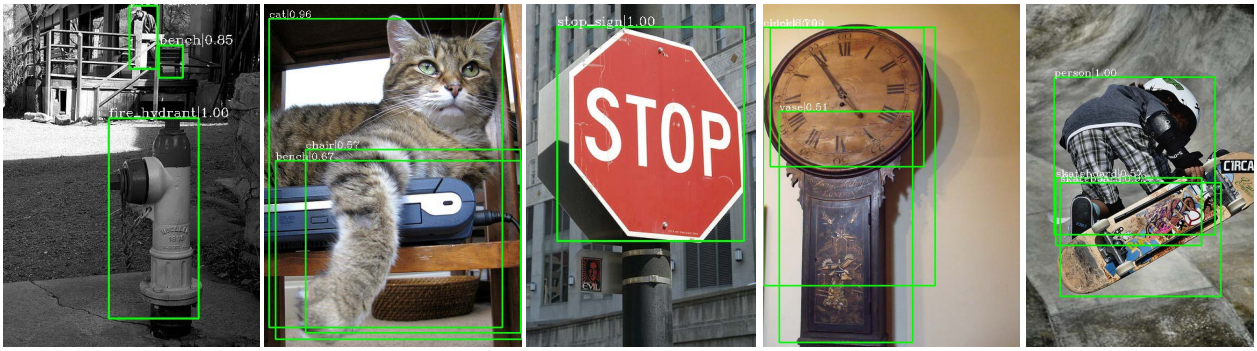
As illustrated in Figure 8, we provide some object detection results comparison between Faster R-CNN [35] baseline and Faster R-CNN w/ SABL on COCO 2017 [25] val. ResNet-101 w/ FPN backbone and 1x training scheduler are adopted in both methods. Faster R-CNN w/ SABL shows more precise localization results than the baseline.



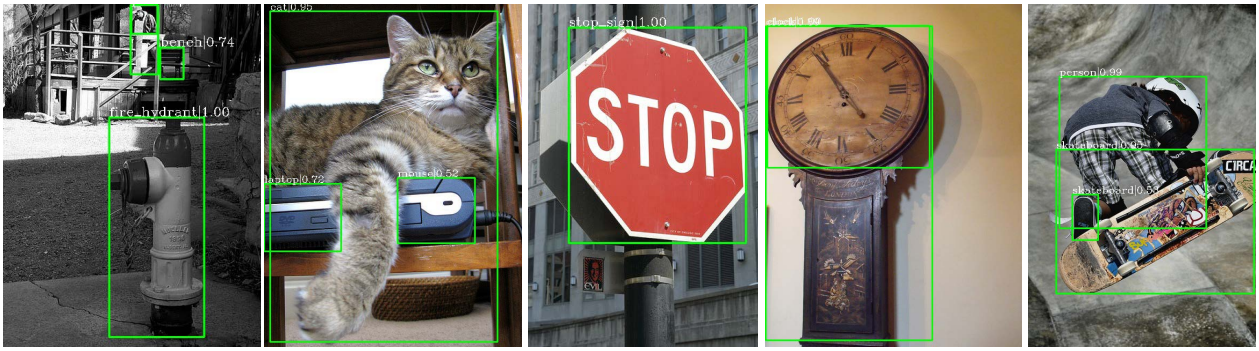
Faster R-CNN



Faster R-CNN w/ SABL



Faster R-CNN



Faster R-CNN w/ SABL

Figure 8: Some comparison of object detection results between Faster R-CNN baseline and Faster R-CNN w/ SABL on COCO 2017 val.