RTMDet: An Empirical Study of Designing Real-Time Object Detectors

Chengqi Lyu[1*] Wenwei Zhang[1,2*] Haian Huang[1] Yue Zhou[1,4] Yudong Wang[1,3]
Yanyi Liu[5] Shilong Zhang[1] Kai Chen[1]

*equal contribution

[1]Shanghai AI Laboratory [2]S-Lab, Nanyang Technological University
[3]School of Electrical and Information Engineering, Tianjin University
[4]Department of Electronic Engineering, Shanghai Jiao Tong University
[5]Northeastern University

{lvchengqi, chenkai}@pjlab.org.cn, wenwei001@e.ntu.edu.sg

Abstract

*In this paper, we aim to design an efficient real-time object detector that exceeds the YOLO series and is easily extensible for many object recognition tasks such as instance segmentation and rotated object detection. To obtain a more efficient model architecture, we explore an architecture that has compatible capacities in the backbone and neck, constructed by a basic building block that consists of large-kernel depth-wise convolutions. We further introduce soft labels when calculating matching costs in the dynamic label assignment to improve accuracy. Together with better training techniques, the resulting object detector, named RTMDet, achieves **52.8% AP on COCO with 300+ FPS** on an NVIDIA 3090 GPU, outperforming the current mainstream industrial detectors. RTMDet achieves the best parameter-accuracy trade-off with tiny/small/medium/large/extra-large model sizes for various application scenarios, and obtains new state-of-the-art performance on real-time instance segmentation and rotated object detection. We hope the experimental results can provide new insights into designing versatile real-time object detectors for many object recognition tasks. Code and models are released at https://github.com/open-mmlab/mmdetection/tree/3.x/configs/rtmdet.*

1. Introduction

Optimal efficiency is always the primary pursuit in object detection, especially for real-world perception in autonomous driving, robotics, and drones. Toward this goal, YOLO series [3, 21, 25, 42, 63–65, 71] explore different model architectures and training techniques to improve the accuracy and efficiency of one-stage object detectors continuously.

In this report, we aim to push the limits of the YOLO series and contribute a new family of ***R**eal-**T**ime **M**odels for object **Det**ection*, named **RTMDet**, which are also capable of doing instance segmentation and rotated object detection that previous works have not explored. The appealing improvements mainly come from better representation with large-kernel depth-wise convolutions and better optimization with soft labels in the dynamic label assignments.

Specifically, we first exploit large-kernel depth-wise convolutions in the basic building block of the backbone and neck in the model, which improves the model's capability of capturing the global context [15]. Because directly placing depth-wise convolution in the building block will increase the model depth thus slowing the inference speed, we further reduce the number of building blocks to reduce the model depth and compensate for the model capacity by increasing the model width. We also observe that putting more parameters in the neck and making its capacity compatible with the backbone could achieve a better speed-accuracy trade-off. The overall modification of the model architectures allows the fast inference speed of RTMDet without relying on model re-parameterizations [42, 71, 84].

We further revisit the training strategies to improve the model accuracy. In addition to a better combination of data augmentations, optimization, and training schedules, we empirically find that existing dynamic label assignment strategies [19, 21] can be further improved by introducing soft targets instead of hard labels when matching ground truth boxes and model predictions. Such a design improves the discrimination of the cost matrix for high-quality matching but also reduces the noise of label assignment, thus improving the model accuracy.

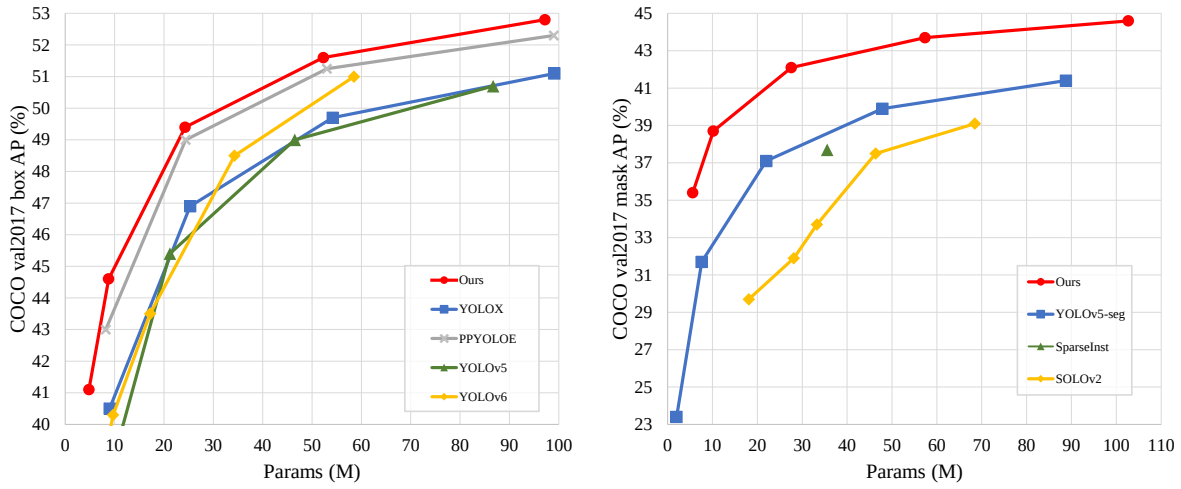

Figure 1. Comparison of parameter and accuracy. (a) Comparison of RTMDet and other state-of-the-art real-time object detectors. (b) Comparison of RTMDet-Ins and other one-stage instance segmentation methods.

RTMDet is generic (通用的) and can be easily extended to instance segmentation and rotated object detection with few modifications. By simply adding a kernel and a mask feature generation head [11,69], RTMDet can perform instance segmentation with only around 10% additional parameters. For rotated object detection, RTMDet only needs to extend the dimension (from 4 to 5) of the box regression layer and switch to a rotated box decoder. We also observe that the pre-training on general object detection datasets [48] is beneficial for rotated object detection in aerial scenarios [81].

We conduct extensive experiments to verify the effectiveness of RTMDet and scale the model size to provide tiny/small/medium/large/extra-large models for various application scenarios. As shown in Fig. 1, RTMDet achieves a better parameter-accuracy trade-off than previous methods and gains superior performance to previous models [3, 21, 25, 65]. Specifically, RTMDet-tiny achieves 41.1% AP at 1020 FPS with only 4.8M parameters. RTMDet-s yields 44.6% AP with 819 FPS, surpassing previous state-of-art small models. When extended to instance segmentation and rotated object detection, RTMDet obtained new state-of-the-art performance on the real-time scenario on both benchmarks, with 44.6% mask AP at 180 FPS on COCO `val` set and 81.33% AP on DOTA v1.0, respectively.

2. Related Work

Efficient neural architecture for object detection. Object detection aims to recognize and localize objects in the scene. For real-time applications, existing works mainly explore anchor-based [47, 50, 64] or anchor-free [70, 98] one-stage detectors, instead of two-stage detectors [5,24,59,66]. To improve the model efficiency, efficient backbone networks and model scaling strategies [3,41,71] and enhancement of multi-scale feature [7, 23, 36, 46, 49, 68] are explored either by handcrafted design or neural architecture search [10, 17, 23, 74]. Recent advances also explore model re-parameterization [14,42,71,84] to improve the inference speed after model deployment. In this paper, we contribute an overall architecture with compatible capacity in the backbone and neck, constructed by a new basic building block with large-kernel depth-wise convolutions toward a more efficient object detector.

Label assignment for object detection. Another dimension to improve the object detector is the design of label assignment and training losses. Pioneer methods [5,47,50,66] use IoU as a matching criterion to compare the ground truth boxes with model predictions or anchors in the label assignment. Later practices [37, 70, 95, 98] further explore different matching criteria such as object centers [70, 98]. Auxiliary detection heads are also explored [62, 71] to speed up and stabilize the training. Inspired by the Hungarian Assignment for end-to-end object detection [6], dynamic label assignment [19–21] are explored to significantly improve the convergence speed and model accuracy. Unlike these strategies that use matching cost functions precisely the same as losses, we propose to use soft labels when calculating the matching costs to enlarge the distinction (差别、区分) between high and low-quality matches, thereby stabilizing training and accelerating convergence.

Instance segmentation. Instance segmentation aims at predicting the per-pixel mask for each object of interest. Pioneer methods explore different paradigms to tackle this task, including mask classification [60, 61], 'Top-Down' [8, 31], and 'Bottom-Up' approaches [1, 39, 58]. Recent attempts perform instance segmentation in one stage with [4, 69] or without bounding boxes [76, 77, 96]. A representative of these attempts is based on dynamic kernels [69, 77, 96], which learn to generate dynamic kernels from either learned

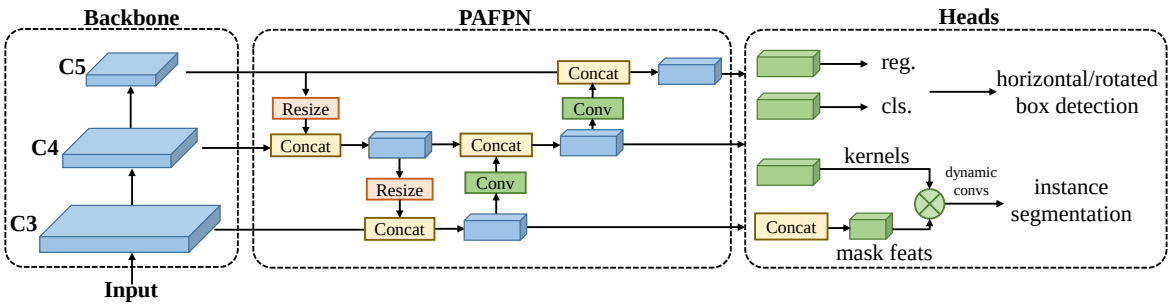

Figure 2. Macro architecture. We use CSP-blocks [72] with large kernel depth-wise convolution layers to build the backbone. The multi-level features, noted as $C3$, $C4$, and $C5$, are extracted from the backbone and then fused in the CSP-PAFPN, which consists of the same block as the backbone. Then, detection heads with shared convolution weights and separated batch normalization (BN) layers are used to predict the classification and regression results for (rotated) bounding box detection. Extra heads can be added to produce dynamic convolution kernels and mask features for the instance segmentation task.

parameters [96] or dense feature maps [69, 77] and use them to conduct convolution with mask feature maps. Inspired by these works, we extend RTMDet by kernel prediction and mask feature heads [69] to conduct instance segmentation.

Rotated object detection. Rotated object detection aims to predict further the orientation of objects in addition to their locations and categories. Based on an existing general object detector (*e.g.*, RetinaNet [47] or Faster R-CNN [66]), different feature extraction networks are proposed to alleviate the feature misalignment [28, 29, 88] caused by object rotations. There are also various representations of rotated boxes explored (*e.g.*, Gaussian distribution [89, 90] and convex set [26, 44]) to ease the rotated bounding box regression task. Orthogonal to these methods, this paper only extends a general object detector with minimal modifications (*i.e.*, adding an angle prediction branch and replacing the GIoU [67] loss by Rotated IoU Loss [97]) and reveals that a high-precision general object detector paves the way for high-precision rotated object detection through the model architecture and the knowledge learned on general detection dataset [48].

3. Methodology

In this work, we build a new family of ***R**eal-**T**ime **M**odels for object **Det**ection*, named **RTMDet**. The macro architecture of RTMDet is a typical one-stage object detector (Sec. 3.1). We improve the model efficiency by exploring the large-kernel convolutions in the basic building block of backbone and neck, and balance the model depth, width, and resolution accordingly (Sec. 3.2). We further explore soft labels in dynamic label assignment strategies and a better combination of data augmentations and optimization strategies to improve the model accuracy (Sec. 3.3). RTMDet is a versatile object recognition framework that can be extended to instance segmentation and rotated object detection tasks with few modifications (Sec. 3.4).

3.1. Macro Architecture

We decompose the macro architecture of a one-stage object detector into the backbone, neck, and head, as shown in Fig. 2. Recent advances of YOLO series [3, 21] typically adopt CSPDarkNet [3] as the backbone architecture, which contains four stages and each stage is stacked with several basic building blocks (Fig. 3.a). The neck takes the multi-scale feature pyramid from the backbone and uses the same basic building blocks as the backbone with bottom-up and top-down feature propogation [46, 49] to enhance the pyramid feature map. Finally, the detection head predicts object bounding boxes and their categories based on the feature map of each scale. Such an architecture generally applies to general and rotated objects, and can be extended for instance segmentation by the kernel and mask feature generation heads [69].

To fully exploit the potential of the macro architecture, we first study more powerful basic building blocks. Then we investigate the computation bottleneck in the architecture and balance the depth, width, and resolution in the backbone and neck.

3.2. Model Architecture

Basic building block. A large effective receptive field in the backbone is beneficial for dense prediction tasks like object detection and segmentation as it helps to capture and model the image context [56] more comprehensively. However, previous attempts (*e.g.*, dilated convolution [92] and non-local blocks [75]) are computationally expensive, limiting their practical use in real-time object detection. Recent studies [15, 52] revisit the use of large-kernel convolutions, showing that one can enlarge the receptive field with a reasonable computational cost through depth-wise convolution [35]. Inspired by these findings, we introduce 5×5 depth-wise convolutions in the basic building block of CSPDarkNet [3] to increase the effective receptive fields (Fig. 3.b). This approach allows for more comprehensive contextual modeling and significantly improves accuracy.

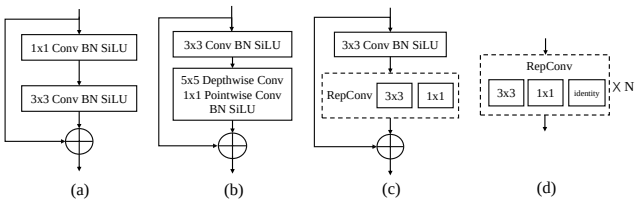

Figure 3. Different basic building blocks. (a) The basic bottleneck block of DarkNet used in [3, 21, 25, 65]. (b) The proposed bottleneck block with a large-kernel depth-wise convolution layer. (c) Bottleneck block of PPYOLO-E [84] that uses re-parameterized convolution. (d) The basic unit of YOLOv6 [42].

It is noteworthy that some recent real-time object detectors [42, 71, 84] explore re-parameterized 3×3 convolutions [14] in the basic building block (Fig. 3.c&d). While the re-parameterized 3×3 convolutions is considered a free lunch to improve accuracy during inference, it also brings side effects such as slower training speed and increased training memory. It also increases the error gap after the model is quantized to lower bits, requiring compensation through re-parameterizing optimizer [13] and quantization-aware training [42]. Large-kernel depth-wise convolution is a simpler and more effective option for the basic building block compared to re-parameterized 3×3 convolution, as they require less training cost and cause less error gaps after model quantization.

Balance of model width and depth. The number of layers in the basic block also increases due to the additional point-wise convolution following the large-kernel depth-wise convolution (Fig. 3.b). This hinders the parallel computation of each layer and thus decreases inference speed. To address this issue, we reduce the number of blocks in each backbone stage and moderately enlarging the width of the block to increase the parallelization and maintain the model capacity, which eventually improves inference speed without sacrificing accuracy.

Balance of backbone and neck. Multi-scale feature pyramid is essential for object detection to detect objects at various scales. To enhance the multi-scale features, previous approaches either use a larger backbone with more parameters or use a heavier neck [36,68] with more connections and fusions among feature pyramid. However, these attempts also increase the computation and memory footprints. Therefore, we adopt another strategy that puts more parameters and computations from backbone to neck by increasing the expansion ratio of basic blocks in the neck to make them have similar capacities, which obtains a better computation-accuracy trade-off.

Shared detection head. Real-time object detectors typically utilize separate detection heads [3, 21, 25, 50, 65] for different feature scales to enhance the model capacity for higher performance, instead of sharing a detection head across multiple scales [47, 70]. We compare different design choices in this paper and choose to share parameters of heads across scales but incorporate different Batch Normalization (BN) layers to reduce the parameter amount of the head while maintaining accuracy. BN is also more efficient than other normalization layers such as Group Normalization [79] because in inference it directly uses the statistics calculated in training.

3.3. Training Strategy

Label assignment and losses. To train the one-stage object detector, the dense predictions from each scale will be matched with ground truth bounding boxes through different label assignment strategies [19, 47, 70]. Recent advances typically adopt dynamic label assignment strategies [6, 20, 21] that use cost functions consistent with the training loss as the matching criterion. However, we find that their cost calculation have some limitations. Hence, we propose a dynamic soft label assignment strategy based on SimOTA [21], and its cost function is formulated as

$$C = \lambda_1 C_{cls} + \lambda_2 C_{reg} + \lambda_3 C_{center}, \tag{1}$$

where C_{cls}, C_{center}, and C_{reg} correspond to the classification cost, region prior cost, and regression cost, respectively, and $\lambda_1 = 1$, $\lambda_2 = 3$, and $\lambda_3 = 1$ are the weights of these three costs by default. The calculation of the three costs is described below.

Previous methods usually utilize binary labels to compute classification cost C_{cls}, which allows a prediction with a high classification score but an incorrect bounding box to achieve a low classification cost and vice versa. To solve this issue, we introduce soft labels in C_{cls} as

$$C_{cls} = CE(P, Y_{soft}) \times (Y_{soft} - P)^2. \tag{2}$$

The modification is inspired by GFL [45] that uses the IoU between the predictions and ground truth boxes as the soft label Y_{soft} to train the classification branch. The soft classification cost in assignment not only reweights the matching costs with different regression qualities but also avoids the noisy and unstable matching caused by binary labels.

When using Generalized IoU [67] as regression cost, the maximum difference between the best match and the worst match is less than 1. This makes it difficult to distinguish high-quality matches from low-quality matches. To make the match quality of different GT-prediction pairs more discriminative, we use the logarithm of the IoU as the regression cost instead of GIoU used in the loss function, which amplifies the cost for matches with lower IoU values. The regression cost C_{reg} is calculated by

$$C_{reg} = -log(IoU). \tag{3}$$

For region cost C_{center}, we use a soft center region cost instead of a fixed center prior [20, 21, 95] to stabilize the matching of the dynamic cost as below

$$C_{center} = \alpha^{|x_{pred} - x_{gt}| - \beta}, \tag{4}$$

where α and β are hyper-parameters of the soft center region. We set $\alpha = 10$, $\beta = 3$ by default.

Cached Mosaic and MixUp. Cross-sample augmentations such as MixUp [94] and CutMix [93] are widely adopted in recent object detectors [3, 21, 25, 42, 71]. These augmentations are powerful but bring two side effects. First, at each iteration, they need to load multiple images to generate a training sample, which introduces more data loading costs and slows the training. Second, the generated training sample is 'noisy' and may not belong to the real distribution of the dataset, which affects the model learning [21].

We improve MixUp and Mosaic with the caching mechanism that reduces the demand for data loading. By utilizing cache, the time cost of mixing images in the training pipeline can be significantly reduced to the level of processing a single image. The cache operation is controlled by the cache length and popping method. A large cache length and random popping method can be regarded as equivalent to the original non-cached MixUp and Mosaic operations. Meanwhile, a small cache length and First-In-First-Out (FIFO) popping method can be seen as similar to the repeated augmentation [2], allowing for the mixing of the same image with different data augmentation operations in the same or contiguous batches.

Two-stage training. To reduce the side effects of 'noisy' samples by strong data augmentations, YOLOX [21] explored a two-stage training strategy, where the first stage uses strong data augmentations, including Mosaic, MixUp, and random rotation and shear, and the second stage use weak data augmentations, such as random resizing and flipping. As the strong augmentation in the initial training stage includes random rotation and shearing that cause misalignment between inputs and the transformed box annotations, YOLOX adds the L1 loss to fine-tune the regression branch in the second stage. To decouple the usage of data augmentation and loss functions, we exclude these data augmentations and increase the number of mixed images to 8 in each training sample in the first training stage of 280 epochs to compensate for the strength of data augmentation. In the last 20 epochs, we switch to Large Scale Jittering (LSJ) [22], allowing for fine-tuning of the model in a domain that is more closely aligned with the real data distributions. To further stabilize the training, we adopt AdamW [55] as the optimizer, which is rarely used in convolutional object detectors but is a default for vision transformers [16].

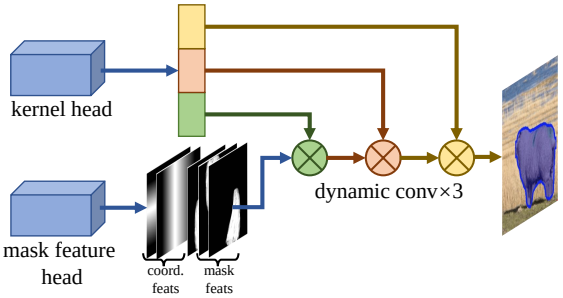

Figure 4. Instance segmentation branch in RTMDet-Ins. The mask feature head has 4 convolution layers and predicts mask features of 8 channels [69] from the multi-level features extracted from neck. Two relative coordinate features are concatenated with the mask features to generate instance masks. The kernel head predicts a 169-dimensional vector for each instance. The vector is divided into three parts (lengths are 88, 72, and 9 respectively), which are used to form the kernels of three dynamic convolution layers.

3.4. Extending to other tasks

Instance segmentation. We enable RTMDet for instance segmentation with a simple modification, denoted as RTMDet-Ins. As illustrated in Figure 4, based on RTMDet, an additional branch is added, consisting of a kernel prediction head and a mask feature head, similar to CondInst [69]. The mask feature head comprises 4 convolution layers that extract mask features with 8 channels from multi-level features. The kernel prediction head predicts a 169-dimensional vector for each instance, which is decomposed into three dynamic convolution kernels to generate instance segmentation masks through interaction with the mask features and coordinate features. To further exploit the prior information inherent in the mask annotations, we use the mass center of the masks when calculating the soft region prior in the dynamic label assignment instead of the box center. We use dice loss [57] as the supervision for the instance masks following typical conventions.

Rotated object detection. Due to the inherent similarity between rotated object detection and general (horizontal) object detection, it only takes 3 steps to adapt RTMDet to a rotated object detector, noted as RTMDet-R: (1) add a 1×1 convolution layer at the regression branch to predict the rotation angle; (2) modify the bounding box coder to support rotated boxes; (3) replace the GIoU loss with Rotated IoU loss. The highly optimized model architecture of RTMDet guarantees high performance of RTMDet-R on the rotated object detection tasks. Moreover, as RTMDet-R shares most parameters of RTMDet, the model weights of RTMDet pre-trained on general detection dataset (*e.g.*, COCO dataset) can serve as a good initialization for rotated object detection.

4. Experiments

4.1. Implementation Details

Object detection and instance segmentation. We conduct experiments on COCO dataset [48], which contains about 118K images in the `train2017` set and 5K images in the `val2017` set for training and validation, respectively. For ablation studies, we trained our models on the `train2017` set for 300 epochs and validated them on the `val2017` set. The hyper-parameters are in Table 1. All our object detection and instance segmentation models are trained on 8 NVIDIA A100 GPUs. We evaluate the model performance on object detection and instance segmentation by bbox AP and mask AP [48], respectively.

During the testing of object detection, the score threshold for filtering bounding boxes is set to 0.001 before non-maximum suppression (NMS), and the top 300 boxes are kept for validation. This setting is consistent with previous studies [25,42,71] for a fair comparison. However, to accelerate the metric computation during ablation studies, we set the score threshold to 0.05 and the number of kept results to 100, which may degrade the accuracy by about 0.3% AP.

Rotated object detection. We conduct experiments on DOTA dataset [81] which contains 2.8K aerial images and 188K instances obtained from different sensors with multiple resolutions. The hyper-parameters are in Table 1. For single-scale training and testing, we crop the original images into 1024×1024 patches with an overlap of 256 pixels. For multi-scale training and testing, the original images are resized with the scale of 0.5, 1.0 and 1.5 and then cropped into 1024×1024 patches with an overlap of 500 pixels. Most of the rotated object detectors are trained by 1 NVIDIA V100 GPU except that the large model uses 2 NVIDIA V100 GPUs. For the evaluation metric, we adopt the same mAP calculation as that in PASCAL VOC2007 [18] but use rotated IoU to calculate the matched objects.

Benchmark settings. The latency of all models is tested in the half-precision floating-point format (FP16) on an NVIDIA 3090 GPU with TensorRT 8.4.3 and cuDNN 8.2.0. The inference batch size is 1.

4.2. Benchmark Results

Object detection. We compare RTMDet with previous real-time object detectors including YOLOv5 [25], YOLOX [21], YOLOv6 [42], YOLOv7 [71], and PPYOLOE [84]. For a fair comparison, all models are trained on 300 epochs without distillation nor pruning and the time of Non-Maximum Suppression (NMS) is not included in the latency calculation.

As shown in Table 2 and Fig. 1 (a), RTMDet achieves a better parameter-accuracy trade-off than previous methods. RTMDet-tiny achieves 41.1% AP with only 4.8M parameters, surpassing other models with a similar size by more than 5% AP. RTMDet-s has a higher accuracy with only half of the parameters and computation costs of YOLOv6-s. RTMDet-m and RTMDet-l also achieve excellent results in similar class models, with 44.6% and 49.4% AP respectively. RTMDet-x yields 52.8% AP with 300+FPS, outperforming the current mainstream detectors. It is worth noting that both [25] and [71] use mask annotation to refine the bounding boxes after data augmentation, resulting in a gain of about 0.3% AP. We achieved superior results without relying on additional information beyond box annotation.

Table 1. **Training settings for object detection, instance segmentation and rotate object detection.**

config	Object detection and instance segmentation	Rotate object detection
optimizer	AdamW [38]	AdamW
base learning rate	0.004	0.00025
weight decay	0.05 (0 for bias and norm [33])	0.05 (0 for bias and norm)
optimizer momentum	0.9	0.9
batch size	256	8
learning rate schedule	Flat-Cosine	Flat-Cosine
training epochs	300	36
warmup iterations	1000	1000
input size	640 ×640	1024 ×1024
augmentation	cached Mosaic and MixUp (first 280 epochs); LSJ [22, 80] (last 20 epochs)	random Flip and Rotate
EMA decay	0.9998	0.9998

Instance segmentation. To evaluate the superiority of our label assignment strategy and loss, we first compare RTMDet-Ins with conventional methods using the standard ResNet50-FPN [46] backbone and the classic multi-scale 3x schedule [9, 80]. We adopt an auxiliary semantic segmentation head for faster convergence speed and a fair comparison with CondInst [69]. RTMDet outperforms CondInst by 1.5% mask AP (the first row in Table 3). However, we do not use the semantic segmentation branch when training RTMDet from scratch with heavy data augmentation because the auxiliary branch brings marginal improvements.

Finally, we trained RTMDet-Ins tiny/s/m/l/x on the COCO dataset using the same data augmentation and optimization hyper-parameters as RTMDet for 300 epochs. RTMDet-Ins-x achieves 44.6% mask AP, surpasses the previous best practice YOLOv5-seg-x [25] by 3.2% AP, and still runs in real-time (second row in Table 3).

Rotated object detection. We compare RTMDet-R with previous state of the arts on the DOTA v1.0 dataset as shown in Table 4. With single-scale training and testing, RTMDet-R-m and RTMDet-R-l achieve 78.24% and 78.85% mAP, respectively, which outperforms almost all

Table 2. **Comparison of RTMDet with previous practices** on the number of parameters, FLOPS, latency, and accuracy on COCO `val2017` set. For a fair comparison, all models are trained for 300 epochs without using extra detection data or knowledge distillation. The inference speeds of all models are measured in the same environment. (LB) means LetterBox resize proposed in [25]. The results of the proposed RTMDet are marked in gray. The best results are in bold

Model	Input shape	Params(M) ↓	FLOPs(G) ↓	Latency(ms) ↓	AP(%) ↑	AP50(%) ↑
YOLOv5-n [25]	640(LB)	1.9	2.3	1.51	28.0	45.7
YOLOX-tiny [21]	416×416	5.1	3.3	0.82	32.8	50.3
YOLOv6-n [42]	640(LB)	4.3	5.6	0.79	35.9	51.2
YOLOv6-tiny	640(LB)	9.7	12.5	0.86	40.3	56.6
RTMDet-tiny	640×640	4.8	8.1	0.98	**41.1**	**57.9**
YOLOv5-s	640(LB)	7.2	8.3	1.63	37.4	56.8
YOLOX-s	640×640	9.0	13.4	1.20	40.5	59.3
YOLOv6-s	640(LB)	17.2	22.1	0.92	43.5	60.4
PPYOLOE-s [84]	640×640	7.9	8.7	1.34	43.0	59.6
RTMDet-s	640×640	8.99	14.8	1.22	**44.6**	**61.9**
YOLOv5-m	640(LB)	21.2	24.5	1.89	45.4	64.1
YOLOX-m	640×640	25.3	36.9	1.68	46.9	65.6
YOLOv6-m	640(LB)	34.3	41.1	1.21	48.5	-
PPYOLOE-m	640×640	23.4	25.0	1.75	49.0	65.9
RTMDet-m	640×640	24.7	39.3	1.62	**49.4**	**66.8**
YOLOv5-l	640(LB)	46.5	54.6	2.46	49.0	67.3
YOLOX-l	640×640	54.2	77.8	2.19	49.7	68.0
YOLOv6-l	640(LB)	58.5	72.0	1.91	51.0	-
YOLOv7 [71]	640(LB)	36.9	52.4	2.63	51.2	-
PPYOLOE-l	640×640	52.2	55.0	2.57	51.4	68.6
RTMDet-l	640×640	52.3	80.2	2.40	**51.5**	**68.8**
YOLOv5-x	640(LB)	86.7	102.9	2.92	50.7	68.9
YOLOX-x	640×640	99.1	141.0	2.98	51.1	69.4
PPYOLOE-x	640×640	98.4	103.3	3.07	52.3	69.5
RTMDet-x	640×640	94.9	141.7	3.10	**52.8**	**70.4**

previous methods. With multi-scale training and testing, RTMDet-R-m and RTMDet-R-l further acheives 80.26% and 80.54% mAP, respectively. Moreover, RTMDet-R-l (COCO pretraining) sets a new record (81.33% mAP) on the DOTA-v1.0 dataset. RTMDet-R also consistently outperforms PPYOLOE-R in all regimes of model sizes with much simpler modifications. Note that RTMDet-R avoids special operators in the architecture to achieve high precision, which makes it can be easily deployed on various hardware. We also compare RTMDet-R with other methods on HRSC2016 [53] and DOTA-v1.5 datasets in the appendix, and RTMDet-R also achieves superior performance.

4.3. Ablation Study of Model Arhitecture

Large kernel matters. We first compare the effectiveness of different kernel sizes in the basic building block of CSPDarkNet [3], with kernel sizes ranging from 3×3 to 7×7. A combination of 3×3 convolution and 5×5 kernel size depth-wise convolution achieves the optimal speed-accuracy trade-off (Table 5a).

Balance of multiple feature scales. Using depth-wise convolution also increases the depth and reduces the inference speed. Thus, we reduce the number of blocks in the 2nd and 3rd stages. As revealed in Table 5b, reducing the number of blocks from 9 to 6 results in a 20% reduction of latency but decreases accuracy by 0.5% AP. To compensate for this loss in accuracy, we incorporate Channel Attention (CA) at the end of each stage, achieving a better speed-accuracy trade-off. Specifically, compared to the detector using 9 blocks in the second and third stages, the accuracy decreases by 0.1% AP, but with a 7% improvement in latency. Overall, our modification successfully reduce the latency of the detector without sacrificing too much accuracy.

Balance of backbone and neck. Following [3, 25, 42, 84], we utilize the same basic block as the backbone for building the neck. We empirically study whether it is more economic to put more computations in the neck. As shown in Table 5c, instead of increasing the complexity of the backbone, making the neck have similar capacity as the backbone can achieve faster speed with similar accuracy in both

Table 3. **Comparison of RTMDet-Ins with previous instance segmentation methods** on the number of parameters, FLOPS, latency, and accuracy on COCO `val2017` set. (LB) means LetterBox resize proposed in [25]. The results of the proposed RTMDet-Ins are marked in gray. The best results are in bold. Different from the object detection model, box NMS and post-processing of top-100 masks are included in the speed measurement

Model	Input shape	Epochs	Params(M) ↓	FLOPs(G) ↓	Latency(ms) ↓	Box AP(%) ↑	Mask AP(%) ↑
SparseInst-R50 [11]	640-853	147	31.6	99.1	-	-	34.2
SOLOv2-R50-FPN [77]	800-1333	36	46.4	253.5	-	-	37.5
CondInst-R50-FPN [69]	800-1333	36	33.9	240.8	-	42.6	38.2
Cascade-R50-FPN [5]	800-1333	36	77.1	403.6	-	44.3	38.5
RTMDet-Ins-R50-FPN	800-1333	36	35.9	295.2	-	**45.3**	**39.7**
YOLOv5n-seg [25]	640(LB)	300	2.0	3.6	1.65	27.6	23.4
YOLOv5s-seg	640(LB)	300	7.6	13.2	1.90	37.6	31.7
YOLOv5m-seg	640(LB)	300	22	35.4	2.71	45.0	37.1
YOLOv5l-seg	640(LB)	300	47.9	73.9	3.44	49.0	39.9
YOLOv5x-seg	640(LB)	300	88.8	132.9	5.10	50.7	41.4
RTMDet-Ins-tiny	640×640	300	5.6	11.8	1.70	40.5	35.4
RTMDet-Ins-s	640×640	300	10.2	21.5	1.93	44.0	38.7
RTMDet-Ins-m	640×640	300	27.6	54.1	2.69	48.8	42.1
RTMDet-Ins-l	640×640	300	57.4	106.6	3.68	51.2	43.7
RTMDet-Ins-x	640×640	300	102.7	182.7	5.31	**52.4**	**44.6**

Table 4. **Comparison of RTMDet-R with previous rotated object detection methods** on the number of parameters, FLOPs, latency, and accuracy on DOTA-v1.0 test set. IN and COCO denote ImageNet pretraining and COCO pretraining. MAE means MAE unsupervised pretraining [30] on the MillionAID [54]. R50 and X50 denote ResNet-50 and ResNeXt-50 (likewise for R101, R152 and X101). Re50 denotes ReResNet-50, RVSA denotes RVSA-ViTAE-B and CRN denotes CSPRepResNet. MS means multi-scale training and testing. DOTA-v1.0 has 15 different object categories: plane (PL), baseball diamond (BD), bridge (BR), ground track field (GTF), small vehicle (SV), large vehicle (LV), ship (SH), tennis court (TC), basketball court (BC), storage tank (ST), soccer ball field (SBF), roundabout (RA), harbor (HA), swimming pool (SP), and helicopter (HC). The AP of each category is listed. The bold fonts indicate the best performance. The results of the proposed RTMDet-R are marked in gray

Method	Pretrain	Backbone	MS	mAP(%)	PL	BD	BR	GTF	SV	LV	SH	TC	BC	ST	SBF	RA	HA	SP	HC
Anchor-based Methods																			
RoI Trans. [12]	IN	R101 [32]	✓	69.56	88.64	78.52	43.44	75.92	68.81	73.68	83.59	90.74	77.27	81.46	58.39	53.54	62.83	58.93	47.67
Gliding Vertex [85]	IN	R101	✓	75.02	89.64	85.00	52.26	77.34	73.01	73.14	86.82	90.74	79.02	86.81	59.55	70.91	72.94	70.86	57.32
CSL [87]	IN	R152	✓	76.17	**90.25**	**85.53**	54.64	75.31	70.44	73.51	77.62	90.84	86.15	86.69	69.60	68.04	73.83	71.10	68.93
R^3Det [88]	IN	R152	✓	76.47	89.80	83.77	48.11	66.77	78.76	83.27	87.84	90.82	85.38	85.51	65.57	62.68	67.53	78.56	72.62
DCL [86]	IN	R152	✓	77.37	89.26	83.60	53.54	72.76	79.04	82.56	87.31	90.67	86.59	86.98	67.49	66.88	73.29	70.56	69.99
S^2ANet [28]	IN	R50	✓	79.42	88.89	83.60	57.74	81.95	79.94	83.19	**89.11**	90.78	84.87	87.81	70.30	68.25	78.30	77.01	69.58
ReDet [29]	IN	Re50 [29]	✓	80.10	88.81	82.48	60.83	80.82	78.34	86.06	88.31	90.87	**88.77**	87.03	68.65	66.90	79.26	79.71	74.67
GWD [89]	IN	R152	✓	80.23	89.66	84.99	59.26	82.19	78.97	84.83	87.70	90.21	86.54	86.85	**73.47**	67.77	76.92	79.22	74.92
KLD [90]	IN	R152	✓	80.63	89.92	85.13	59.19	81.33	78.82	84.38	87.50	89.80	87.33	87.00	72.57	71.35	77.12	79.34	78.68
Oriented RCNN [83]	IN	R50	✓	80.87	89.84	85.43	61.09	79.82	79.71	85.35	88.82	90.88	86.68	87.73	72.21	70.80	82.42	78.18	74.11
RoI Trans. + KFIoU [91]	IN	Swin-tiny [51]	✓	80.93	89.44	84.41	**62.22**	**82.51**	80.10	**86.07**	88.68	**90.90**	87.32	**88.38**	72.80	**71.95**	78.96	74.95	75.27
Oriented RCNN	MAE	RVSA [73]	✓	**81.18**	89.40	83.94	59.76	82.10	**81.73**	85.32	88.88	90.86	85.69	87.65	63.70	69.94	**84.72**	**84.16**	**79.90**
Anchor-free Methods																			
CFA [27]	IN	R152	✓	76.67	89.08	83.20	54.37	66.87	81.23	80.96	87.17	90.21	84.32	86.09	52.34	69.94	75.52	80.76	67.96
DAFNe [40]	IN	R101	✓	76.95	89.40	86.27	53.70	60.51	82.04	81.17	88.66	90.37	83.81	87.27	53.93	69.38	75.61	81.26	70.86
SASM [34]	IN	RX101 [82]	✓	77.19	88.41	83.32	54.00	74.34	80.87	84.10	88.04	90.74	82.85	86.26	63.96	66.78	78.40	73.84	61.97
Oriented RepPoints [44]	IN	Swin-tiny		77.63	89.11	82.32	56.71	74.95	80.70	83.73	87.67	90.81	87.11	85.85	63.60	68.60	75.95	73.54	63.76
PPYOLOE-R-s	IN	CRN-s [84]		73.82	88.80	79.24	45.92	66.88	80.41	82.95	88.20	90.61	82.91	86.37	55.80	64.11	65.09	79.50	50.43
PPYOLOE-R-s	IN	CRN-s	✓	79.42	88.93	83.95	56.60	79.40	82.57	85.89	88.64	90.87	87.82	87.54	68.94	63.46	76.66	79.19	70.87
PPYOLOE-R-m	IN	CRN-m		77.64	89.23	79.92	51.14	72.94	81.86	84.56	88.68	90.85	86.85	87.48	59.16	68.34	73.78	81.72	68.10
PPYOLOE-R-m	IN	CRN-m	✓	79.71	88.63	84.45	56.27	79.12	**83.52**	**86.16**	88.77	90.81	88.01	**88.39**	70.41	61.44	77.65	77.70	74.30
PPYOLOE-R-l	IN	CRN-l		78.14	89.18	81.00	54.01	70.22	81.85	85.16	88.81	90.81	86.99	88.01	62.87	67.87	76.56	79.13	69.65
PPYOLOE-R-l	IN	CRN-l	✓	80.02	88.40	84.75	58.91	76.35	83.13	86.10	88.79	90.87	88.74	87.71	67.71	68.44	77.92	76.17	76.35
PPYOLOE-R-x	IN	CRN-x		78.28	**89.49**	79.70	55.04	75.59	82.40	85.20	88.35	90.76	85.69	87.70	63.17	69.52	77.09	75.08	69.38
PPYOLOE-R-x	IN	CRN-x	✓	80.73	88.45	84.46	**60.57**	77.70	83.34	85.36	**88.97**	90.78	88.53	87.47	69.26	65.96	77.86	81.36	**80.93**
RTMDet-R-tiny	IN	Ours-tiny		75.36	89.21	80.03	47.88	69.73	82.05	83.33	88.63	**90.91**	86.31	86.85	59.94	62.30	74.23	71.97	57.03
RTMDet-R-tiny	IN	Ours-tiny	✓	79.82	87.89	85.70	55.83	81.28	81.47	85.12	88.91	90.88	88.15	87.96	67.29	68.59	77.71	80.50	69.96
RTMDet-R-s	IN	Ours-s		76.93	89.18	80.45	52.09	71.35	81.55	84.05	88.79	90.89	87.83	86.98	59.58	62.28	75.90	81.96	61.04
RTMDet-R-s	IN	Ours-s	✓	79.98	88.16	86.09	56.80	78.79	80.62	85.06	88.64	90.82	86.90	86.70	66.23	70.22	78.17	81.71	74.58
RTMDet-R-m	IN	Ours-m		78.24	89.17	84.65	53.92	74.67	81.48	83.99	88.71	90.85	87.43	87.20	59.39	66.68	77.71	**82.40**	65.28
RTMDet-R-m	IN	Ours-m	✓	80.26	87.10	85.83	56.30	80.28	80.04	84.67	88.22	90.88	88.49	87.57	70.74	69.99	78.35	80.88	74.53
RTMDet-R-l	IN	Ours-l		78.85	89.43	84.21	55.20	75.06	80.81	84.53	**88.97**	90.90	87.38	87.25	63.09	67.87	78.09	80.78	69.13
RTMDet-R-l	IN	Ours-l	✓	80.54	88.36	84.96	57.33	80.46	80.58	84.88	88.08	90.90	86.32	87.57	69.29	70.61	78.63	80.97	79.24
RTMDet-R-l	COCO	Ours-l	✓	**81.33**	88.01	**86.17**	58.54	**82.44**	81.30	84.82	88.71	90.89	**88.77**	87.37	**71.96**	**71.18**	**81.23**	81.40	77.13

Table 5. **Ablation studies of model architecture** on COCO `val2017` set. The proposed setting is marked in gray

(a) Speed-accuracy trade-off of kernel size

Kernel Size	Params. ↓	GFLOPs ↓	Latency ↓	AP(%) ↑
3×3	50.80M	79.61G	2.10ms	50.0
5×5	50.92M	79.70G	2.11ms	50.9
7×7	51.10M	80.34G	2.73ms	51.1

(b) Speed-accuracy trade-off of the number of blocks

Num. Blocks	Params. ↓	GFLOPs ↓	Latency ↓	AP(%) ↑
3-9-9-3	53.40M	86.28G	2.60ms	51.4
3-6-6-3	50.92M	79.70G	2.11ms	50.9
3-6-6-3 w/CA	52.30M	79.90G	2.40ms	51.3

(c) Ablation study of backbone and neck proportions

Model Size	Backbone	Neck	Params. ↓	GFLOPs ↓	Latency ↓	AP(%) ↑
Small	47%	45%	8.54M	15.76G	**1.21ms**	**43.9**
Small	63%	29%	9.01M	15.85G	1.37ms	43.7
Large	47%	45%	50.92M	79.70G	**2.11ms**	50.9
Large	63%	29%	57.43M	93.73G	2.57ms	**51.0**

(d) Design of the detection head

Head Type	Params. ↓	GFLOPs ↓	Latency ↓	AP(%) ↑
Shared Head	52.32M	80.23G	2.44ms	48.0
Totally Separate	57.03M	80.23G	2.44ms	51.2
Separate BN	52.32M	80.23G	2.44ms	**51.3**

Table 6. **Ablation studies of label assignment** on COCO `val2017` set. The proposed setting is marked in gray

(a) Ablation study of dynamic soft label assignment with ResNet50 1x schedule

Soft cls. cost	Soft ctr. prior	Log IoU cost	AP(%) ↑
			39.9
✓			40.3
✓	✓		40.8
✓	✓	✓	**41.3**

(b) Comparison with other label assignment with ResNet50 1x schedule

Method	AP(%) ↑
ATSS [95]	39.2
PAA [37]	40.4
OTA [20]	40.7
TOOD [19] (w/o T-Head)	40.7
Ours	**41.3**

(c) Comparison with SimOTA label assignment on RTMDet-s with the same losses and other training strategies

Method	AP(%) ↑
SimOTA	43.2
Ours	**44.5**

small and large real-time detectors.

Detection head. In Table 5d, we compare different sharing strategies of the detection head for multi-scale features. The results show that incorporating Batch Normalization (BN) into a shared-weight detection head causes a performance drop because of the statistical differences between different feature scales. Using different detection heads for different feature scales can solve this issue but significantly increases the parameter numbers. Using the same weights for different feature scales but different BN statistics yields the best parameter-accuracy trade-off.

4.4. Ablation Study of Training Strategy

Label assignment. We then verify the effectiveness of each component in the proposed dynamic soft label assignment strategy. Following previous conventions, we use SimOTA [21] as our baseline and employ the Focal-Loss [47] and GIoU [67], which are the same as the training losses, as the cost matrix. As shown in Table 6a, our baseline version can achieve an AP of 39.9% on ResNet-50. Introducing IoU as a soft label in the classification cost improves the accuracy by 0.4% AP, reaching 40.3% AP. Replacing the fixed 3×3 center prior with a softened center prior further improves the accuracy to 40.8% AP. By replacing the GIoU cost with the logarithm IoU cost, the model obtains 41.3% AP.

The proposed label assignment strategy surpasses other high-performance strategies by 0.5% AP on the same model architecture with the same losses (Table 6b). When trained with a longer training schedule and stronger data augmentation, the proposed dynamic soft label assignment, together with the losses, surpasses SimOTA by 1.3% AP on RTMDet-s (Table 6c).

Data augmentation. We then study different combinations of data augmentations at different training stages. The first and second training stage takes 280 and 20 epochs, respectively. When the data augmentation is the same in these two stages, it essentially forms a one-stage training. Following YOLOX, the range of random resizing in Mosaic for tiny and small models is $(0.5, 2.0)$, while $(0.1, 2.0)$ is used for larger models. As shown in Table 7a, using large-scale jittering (LSJ) [22] in all the stages is 0.4% AP better than using MixUp and Mosaic. The effect of cached Mosaic and MixUp is consistent with the original ones when there are sufficient cached samples. Still, the cache mechanism speeds up Mosaic and MixUp by $\sim 3.6\times$ and $\sim 1.5\times$, respectively (Table 7b).

Using LSJ in the second stage instead of Mosaic and MixUp brings 2% AP and 1.5% improvement for RTMDet-s and RTMDet-l, respectively. This indicates that Mosaic and MixUp is a stronger augmentation than LSJ but also introduces more noise in training, which should be thrown in the second stage. We also observe that if the cache size

Table 7. **Ablation studies of data augmentation** on COCO `val2017` set. The proposed settings are marked in gray

(a) Comparison with large-scale jittering (LSJ) and Mosaic & MixUp of different setting. "small" means using a small cache size and FIFO popping method

Model	Data Aug. in 1st stage	Data Aug. in 2nd stage	AP(%) ↑
RTMDet-s	LSJ [22]	LSJ	42.3
	Mosaic & MixUp	Mosaic & MixUp	41.9
	Cached Mosaic & MixUp	Cached Mosaic & MixUp	41.9
	Cached Mosaic & MixUp	LSJ	43.9
	Cached(small) Mosaic & MixUp	LSJ	**44.2**
RTMDet-l	LSJ	LSJ	46.7
	Mosaic & MixUp	Mosaic & MixUp	49.8
	Cached Mosaic & MixUp	Cached Mosaic & MixUp	49.8
	Cached Mosaic & MixUp	LSJ	**51.3**
	Cached(small) Mosaic & MixUp	LSJ	51.1

(b) The speed comparison with vanilla and cached Mosaic & MixUp

	Use cache	ms/100imgs ↓
Mosaic		87.1
Mosaic	✓	**24.0**
MixUp		19.3
MixUp	✓	**12.4**

(c) Comparison with data augmentation in YOLOX

Model	Data Aug.	AP(%) ↑
RTMDet-s	YOLOX	42.9
RTMDet-s	Ours	**44.2**
RTMDet-l	YOLOX	50.6
RTMDet-l	Ours	**51.3**

was reduced to around 10 images and the First-In-First-Out (FIFO) popping method was applied, it is possible to mix the same image with different data augmentation operations in the same batch, which may have similar effects as repeated augmentation [2] and can slightly improve tiny and small models (by approximately 0.5% AP).

Compared with YOLOX, we avoid random rotation and shearing in the first training stage because they cause misalignment between box annotations and the inputs. Instead, we increase the number of mixed images from 5 to 8 in each training sample to keep the strength of data augmentation in the first stage. Overall, the new combination of data augmentation explored in this paper is consistently better than those of YOLOX at different model sizes 7c.

Optimization strategy. We finally conduct experiments on the optimization strategies. The results in Table 8 indicate that SGD leads to unstable convergence progress with heavy data augmentation in training. As a result, we selected AdamW with a 0.05 weight decay and Cosine Annealing LR as our baseline. To avoid overfitting in the early or middle training progress due to the quickly reduced learning rate by Cosine Annealing, we adapt a flat-cosine approach, where a fixed learning rate is used in the first half of training epochs, and cosine annealing is then used in the second half. This modification improves the performance by 0.3% AP. Furthermore, inhibiting weight decay on normalization layers and biases following previous practices [33] brings 0.9% AP improvement. Finally, applying a pre-trained ImageNet backbone through the RSB [78] training strategy leads to a further 0.3% increase in AP. The above-mentioned tricks synergically yield significant improvements of 1.5% AP.

Table 8. **Ablation study of optimization strategy** based on RTMDet-s. The proposed setting is marked in gray

Optimizer	AP(%) ↑
SGD + CosineLR	unstable
AdamW + CosineLR	43.0
AdamW + Flat CosineLR	43.3
+ w/o norm&bias decay	44.2
+ RSB pretrain	**44.5**

4.5. Step-by-step Results

As demonstrated in Table 9, we have made successive modifications to YOLOX-s. By modifying the optimization strategy, the model accuracy is improved by 0.4%. The new architecture that has a similar capacity of backbone and neck, constructed by the new basic building block with large-kernel depth-wise convolutions, improves the model accuracy by 1.2% AP with marginal latency costs. Using a detection head with shared weights reduces the number of parameters significantly without hurting the accuracy. Subsequent enhancements to the label assignment strategy and training losses boost the performance by 1.1% AP. The new combination of data augmentations and the pre-training of backbone leads to 1.3% AP and 0.3% AP improvement, respectively. The synergy of these modifications results in the RTMDet-s, which outperforms the baseline by 4.3% AP.

Table 9. Step-by-step improvements from YOLOX-s baseline to RTMDet-s. The proposed setting is marked in gray

Model	Params(M) ↓	FLOPs(G) ↓	Latency(ms) ↓	AP(%) ↑
YOLOX baseline	9.0M	13.4G	1.20ms	40.2
+ AdamW & Flat CosineLR	9.0M	13.4G	1.20ms	40.6(+0.4)
+ New architecture	10.07M(+1.07)	14.8G(+1.4)	1.22ms	41.8(+1.2)
+ SepBNHead	8.89M(-1.18)	14.8G	1.22ms	41.8(+0.0)
+ Label Assign & Loss	8.89M	14.8G	1.22ms	42.9(+1.1)
+ Imporved data augmentations	8.89M	14.8G	1.22ms	44.2(+1.3)
+ RSB pretrained backbone	8.89M	14.8G	1.22ms	**44.5(+0.3)**

5. Conclusion

In this paper, we empirically and comprehensively study each critical component in real-time object detectors, including model architectures, label assignment, data augmentations, and optimization. We further explore minimal adaptations of a high-precision real-time object detector for real-time instance segmentation and rotated object detection. The findings in the study result in a new family ***R**eal-**T**ime **M**odels for object **Det**ection*, named **RTMDet**, and its derivatives for different object recognition tasks. RTMDet demonstrates a superior trade-off between accuracy and speed in industrial-grade applications, with different model sizes for different object recognition tasks. We hope RTMDet, with the experimental results, can pave the way for future research and industrial development of real-time object recognition tasks.

A. Appendix

A.1. Benchmark Results

Comparison with PPYOLOE-R. We further compare RTMDet-R with PPYOLOE-R in detail, and RTMDet-R is more competitive in accuracy and inference speed, as shown in Table A1. More surprisingly, RTMDet-R-m and RTMDet-R-l surpass PPYOLOE-R-l and PPYOLOE-R-x while being 18.5% and 20.8% faster, respectively. Code and models of RTMDet-R are released at MMRotate [99].

Results on DOTA-v1.5. We further verify the effectiveness of RTMDet-R on DOTA-v1.5 dataset. DOTA-v1.5 contains the same images as DOTA-v1.0 but annotates extremely small instances (less than 10 pixels) with 215k instances added, which makes it more challenging. For the DOTA-v1.5 dataset, we use 4 NVIDIA A100 GPUs for training. Since we find that COCO pretraining significantly improves the results on DOTA-v1.5, we use COCO pretraining by default. Other settings are consistent with DOTA-v1.0. As shown in Table A2, RTMDet-R-l surpasses the previous best method ReDet [29] by 1.32% mAP.

Results on HRSC2016. We also verify RTMDet-R on HRSC2016 dataset, a ship detection dataset containing 1K images and a total of 2.9K ships collected from Google Earth. For the HRSC2016 dataset, we do not change the aspect ratios of images. We train all the models with 108 epochs for HRSC2016 dataset. Other settings are consistent with those for DOTA-v1.0. RTMDet-R also obtains a new state-of-the-art performance and achieves 90.6% mAP_{07} (Table A3).

References

[1] Min Bai and Raquel Urtasun. Deep watershed transform for instance segmentation. In *cvpr*, pages 5221–5229, 2017. 2

[2] Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multigrain: a unified image embedding for classes and instances. *arXiv preprint arXiv:1902.05509*, 2019. 5, 10

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. 1, 2, 3, 4, 5, 7

[4] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9156–9165. IEEE Computer Society, 2019. 2

[5] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *CVPR*, 2018. 2, 8

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 2, 4

[7] Kai Chen, Yuhang Cao, Chen Change Loy, Dahua Lin, and Christoph Feichtenhofer. Feature pyramid grids. *arXiv: Computer Vision and Pattern Recognition*, 2020. 2

[8] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2, 12

[9] 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

Table A1. **Comparison of RTMDet-R with PPYOLOE-R** on the number of parameters, FLOPs, latency, and accuracy. The inference speeds of all models are measured in the same environment. The results of the proposed RTMDet-R is marked in gray. The best results are in bold

Model	Input shape	Params(M) ↓	FLOPs(G) ↓	Latency(ms) ↓	mAP(%) ↑
RTMDet-R-tiny	1024	4.88	20.45	3.04	**75.36**
PPYOLOE-R-s	1024	8.24	22.16	3.38	73.82
RTMDet-R-s	1024	8.86	37.62	3.44	**76.93**
PPYOLOE-R-m	1024	24.66	65.40	5.26	77.64
RTMDet-R-m	1024	24.67	99.76	5.79	**78.24**
PPYOLOE-R-l	1024	55.17	145.88	7.09	78.14
RTMDet-R-l	1024	52.27	204.21	8.20	**78.85**
PPYOLOE-R-x	1024	104.18	275.41	10.36	78.28

Table A2. **Comparison with state-of-the-art methods on DOTA v1.5 dataset.** MS means multi-scale training and testing. For the DOTA-v1.5 dataset, we use 4 NVIDIA A100 GPUs for training. Since we find that COCO pretraining significantly improves the results on DOTA-v1.5, we use COCO pretraining by default. DOTA-v1.5 has 16 different object categories: plane (PL), baseball diamond (BD), bridge (BR), ground track field (GTF), small vehicle (SV), large vehicle (LV), ship (SH), tennis court (TC), basketball court (BC), storage tank (ST), soccer ball field (SBF), roundabout (RA), harbor (HA), swimming pool (SP), helicopter (HC) and container crane (CC). The AP of each category is listed. The bold fonts indicate the best performance. The results of the proposed RTMDet-R are marked in gray

Method	MS	PL	BD	BR	GTF	SV	LV	SH	TC	BC	ST	SBF	RA	HA	SP	HC	CC	mAP
RetinaNet-OBB [47]		71.43	77.64	42.12	64.65	44.53	56.79	73.31	90.84	76.02	59.96	46.95	69.24	59.65	64.52	48.06	0.83	59.16
FR-OBB [66]		71.89	74.47	44.45	59.87	51.28	69.98	79.37	90.78	77.38	67.50	47.75	69.72	61.22	65.28	60.47	1.54	62.00
MASK RCNN [31]		76.84	73.51	49.90	57.80	51.31	71.34	79.75	90.46	74.21	66.07	46.21	70.61	63.07	64.46	57.81	9.42	62.67
HTC [8]		77.80	73.67	51.40	63.99	51.54	73.31	80.31	90.48	75.12	67.34	48.51	70.63	64.84	64.48	55.87	5.15	63.40
DAFNe	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	71.99
OWSR [43]	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	74.90
ReDet		79.20	82.81	51.92	71.41	52.38	75.73	80.92	90.83	75.81	68.64	49.29	72.03	73.36	70.55	63.33	11.53	66.86
ReDet	✓	88.51	86.45	**61.23**	81.20	**67.60**	**83.65**	**90.00**	90.86	84.30	75.33	**71.49**	72.06	78.32	74.73	76.10	46.98	76.80
RTMDet-R-tiny		77.79	83.03	48.45	73.37	59.33	81.30	88.89	90.88	80.73	76.26	51.81	71.59	75.81	75.19	54.36	20.01	69.30
RTMDet-R-tiny	✓	88.14	83.09	51.80	77.54	65.99	82.22	89.81	90.88	80.54	81.34	64.64	71.51	77.13	76.32	72.11	46.67	74.98
RTMDet-R-s		80.05	84.36	50.65	72.04	59.54	81.79	89.22	**90.90**	83.07	76.27	56.82	72.13	76.25	77.04	65.66	32.84	71.79
RTMDet-R-s	✓	88.14	85.82	52.90	82.09	65.58	81.83	89.78	90.82	83.31	82.47	68.51	70.93	**78.00**	75.77	73.09	47.32	76.02
RTMDet-R-m		80.34	86.00	54.02	72.98	63.21	82.09	89.46	90.87	85.12	76.69	63.12	72.14	77.91	76.04	71.57	32.24	73.36
RTMDet-R-m	✓	89.07	**86.71**	52.57	82.47	66.13	82.55	89.77	90.88	84.39	83.34	69.51	73.03	77.82	75.98	**80.21**	42.00	76.65
RTMDet-R-l		80.73	84.79	54.09	76.30	63.56	83.06	89.77	90.89	86.65	76.98	63.68	70.31	78.11	75.91	75.09	31.20	73.82
RTMDet-R-l	✓	**89.31**	86.38	55.09	**83.17**	66.11	82.44	89.85	90.84	**86.95**	**83.76**	68.35	**74.36**	77.60	**77.39**	77.87	**60.37**	**78.12**

Table A3. Comparison with state-of-the-art methods on HRSC2016 dataset. mAP_{07} and mAP_{12} indicate that the results were evaluated under VOC2007 and VOC2012 metrics (%), respectively. We report both results for fair comparison. The results of the proposed RTMDet-R is marked in gray. The results of the proposed RTMDet is marked in gray. The best results are in bold

Model	Input shape	mAP_{07}	mAP_{12}
RoI Trans.	512-800	86.20	-
Gliding Vertex	512-800	88.20	-
R^3Det	800×800	89.26	96.01
GWD	800×800	89.85	97.37
CSL	800×800	89.62	96.10
S^2ANet	512-800	90.17	95.01
ReDet	512-800	90.46	**97.63**
Oriented RCNN	800-1333	90.50	97.60
RTMDet-R-tiny	800×800	**90.60**	97.10

mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 6

[10] 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

[11] Tianheng Cheng, Xinggang Wang, Shaoyu Chen, Wenqiang Zhang, Qian Zhang, Chang Huang, Zhaoxiang Zhang, and Wenyu Liu. Sparse instance activation for real-time instance segmentation. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2022. 2, 8

[12] Jian Ding, Nan Xue, Yang Long, Gui-Song Xia, and Qikai Lu. Learning roi transformer for oriented object detection in aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2849–2858, 2019. 8

[13] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Kaiqi Huang, Jungong Han, and Guiguang Ding. Re-parameterizing your optimizers rather than architectures. 2022. 4

[14] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. RepVGG: Making VGG-style

convnets great again. In *CVPR*, 2021. 2, 4

[15] Xiaohan Ding, Xiangyu Zhang, Yizhuang Zhou, Jungong Han, Guiguang Ding, and Jian Sun. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. *CoRR*, abs/2203.06717, 2022. 1, 3

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*. OpenReview.net, 2021. 5

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

[18] Mark Everingham, Luc Van Gool, Christopher Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 06 2010. 6

[19] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R. Scott, and Weilin Huang. TOOD: task-aligned one-stage object detection. In *ICCV*, 2021. 1, 2, 4, 9

[20] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and Jian Sun. Ota: Optimal transport assignment for object detection. In *CVPR*, 2021. 2, 4, 5, 9

[21] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: Exceeding YOLO series in 2021. *CoRR*, abs/2107.08430, 2021. 1, 2, 3, 4, 5, 6, 7, 9

[22] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin Cubuk, Quoc Le, Barret Zoph, Google Research, Brain Team, and U Berkeley. Simple copy-paste is a strong data augmentation method for instance segmentation. 2022. 5, 6, 9, 10

[23] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019. 2

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

[25] glenn jocher et al. yolov5. https://github.com/ultralytics/yolov5, 2021. 1, 2, 4, 5, 6, 7, 8

[26] Zonghao Guo, Chang Liu, Xiaosong Zhang, Jianbin Jiao, Xiangyang Ji, and Qixiang Ye. Beyond bounding-box: Convex-hull feature adaptation for oriented and densely packed object detection. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8788–8797, 2021. 3

[27] Zonghao Guo, Chang Liu, Xiaosong Zhang, Jianbin Jiao, Xiangyang Ji, and Qixiang Ye. Beyond bounding-box: Convex-hull feature adaptation for oriented and densely packed object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8792–8801, 2021. 8

[28] Jiaming Han, Jian Ding, Jie Li, and Gui-Song Xia. Align deep features for oriented object detection. *IEEE Transactions on Geoscience and Remote Sensing*, 2021. 3, 8

[29] Jiaming Han, Jian Ding, Nan Xue, and Gui-Song Xia. Redet: A rotation-equivariant detector for aerial object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2786–2795, 2021. 3, 8, 11

[30] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022. 8

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

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 8

[33] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019. 6, 10

[34] Liping Hou, Ke Lu, Jian Xue, and Yuqiu Li. Shape-adaptive selection and measurement for oriented object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022. 8

[35] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 3

[36] Yiqi Jiang, Zhiyu Tan, Junyan Wang, Xiuyu Sun, Ming Lin, and Hao Li. GiraffeDet: A heavy-neck paradigm for object detection. In *ICLR*. OpenReview.net, 2022. 2, 4

[37] Kang Kim and Hee Seok Lee. Probabilistic anchor assignment with iou prediction for object detection. In *ECCV*, 2020. 2, 9

[38] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[39] Alexander Kirillov, Evgeny Levinkov, Bjoern Andres, Bogdan Savchynskyy, and Carsten Rother. Instancecut: from edges to instances with multicut. In *cvpr*, pages 5008–5017, 2017. 2

[40] Steven Lang, Fabrizio Ventola, and Kristian Kersting. Dafne: A one-stage anchor-free deep model for oriented object detection. *arXiv preprint arXiv:2109.06148*, 2021. 8

[41] Youngwan Lee, Joong won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. An energy and gpu-computation efficient backbone network for real-time object detection. *computer vision and pattern recognition*, 2019. 2

[42] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. Yolov6: A single-stage object detection framework for industrial applications. *CoRR*, abs/2209.02976, 2022. 1, 2, 4, 5, 6, 7

[43] Chengzheng Li, Chunyan Xu, Zhen Cui, Dan Wang, Zequn Jie, Tong Zhang, and Jian Yang. Learning object-wise semantic representation for detection in remote sensing imagery. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 12

[44] Wentong Li, Yijie Chen, Kaixuan Hu, and Jianke Zhu. Oriented reppoints for aerial object detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1819–1828, 2022. 3, 8

[45] Xiang Li, Chengqi Lv, Wenhai Wang, Gang Li, Lingfeng Yang, and Jian Yang. Generalized focal loss: Towards efficient representation learning for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–14, 2022. 4

[46] 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. 2, 3, 6

[47] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 2, 3, 4, 9, 12

[48] 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. 2, 3, 6

[49] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. *computer vision and pattern recognition*, 2018. 2, 3

[50] 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, 4

[51] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pages 9992–10002. IEEE, 2021. 8

[52] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *CoRR*, abs/2201.03545, 2022. 3

[53] Zikun Liu, Hongzhen Wang, Lubin Weng, and Yiping Yang. Ship rotated bounding box space for ship extraction from high-resolution optical satellite images with complex backgrounds. *IEEE Geoscience and Remote Sensing Letters*, 13(8):1074–1078, 2016. 7

[54] Yang Long, Gui-Song Xia, Shengyang Li, Wen Yang, Michael Ying Yang, Xiao Xiang Zhu, Liangpei Zhang, and Deren Li. On creating benchmark dataset for aerial image interpretation: Reviews, guidances, and million-aid. *IEEE Journal of selected topics in applied earth observations and remote sensing*, 14:4205–4230, 2021. 8

[55] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*. OpenReview.net, 2019. 5

[56] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *NeurIPS*, 2016. 3

[57] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. IEEE, 2016. 5

[58] Davy Neven, Bert De Brabandere, Marc Proesmans, and Luc Van Gool. Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth. In *cvpr*, pages 8837–8845, 2019. 2

[59] 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. 2

[60] Pedro O. Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. 2015. 2

[61] Pedro O. Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In *ECCV*, 2016. 2

[62] RangiLyu. NanoDet-Plus: Super fast and high accuracy lightweight anchor-free object detection model. https://github.com/RangiLyu/nanodet, 2021. 2

[63] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 1

[64] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. In *CVPR*, 2017. 1, 2

[65] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. 1, 2, 4

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

[67] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. *CVPR*, 2019. 3, 4, 9

[68] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and efficient object detection. In *CVPR*, 2020. 2, 4

[69] Zhi Tian, Chunhua Shen, and Hao Chen. Conditional convolutions for instance segmentation. In *European conference on computer vision*, pages 282–298. Springer, 2020. 2, 3, 5, 6, 8

[70] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. *CoRR*, abs/1904.01355, 2019. 2, 4

[71] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *CoRR*, abs/2207.02696, 2022. 1, 2, 4, 5, 6, 7

[72] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn. *CVPR*, 2019. 3

[73] Di Wang, Qiming Zhang, Yufei Xu, Jing Zhang, Bo Du, Dacheng Tao, and Liangpei Zhang. Advancing plain vision transformer towards remote sensing foundation model. *IEEE Transactions on Geoscience and Remote Sensing*, pages 1–1, 2022. 8

[74] 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

[75] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, pages 7794–7803, 2018. 3

[76] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. SOLO: Segmenting objects by locations. In *eccv*. Springer, 2020. 2

[77] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. SOLOv2: Dynamic, faster and stronger. In *NeurIPS*, 2020. 2, 3, 8

[78] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. *arXiv: Computer Vision and Pattern Recognition*, 2021. 10

[79] Yuxin Wu and Kaiming He. Group normalization. *ECCV*, 2018. 4

[80] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 6

[81] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Dota: A large-scale dataset for object detection in aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3974–3983, 2018. 2, 6

[82] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 8

[83] Xingxing Xie, Gong Cheng, Jiabao Wang, Xiwen Yao, and Junwei Han. Oriented r-cnn for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3520–3529, 2021. 8

[84] Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao Chang, Cheng Cui, Kaipeng Deng, Guanzhong Wang, Qingqing Dang, Shengyu Wei, Yuning Du, and Baohua Lai. PP-YOLOE: An evolved version of YOLO. *CoRR*, abs/2203.16250, 2022. 1, 2, 4, 6, 7, 8

[85] Yongchao Xu, Mingtao Fu, Qimeng Wang, Yukang Wang, Kai Chen, Gui-Song Xia, and Xiang Bai. Gliding vertex on the horizontal bounding box for multi-oriented object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(4):1452–1459, 2020. 8

[86] Xue Yang, Liping Hou, Yue Zhou, Wentao Wang, and Junchi Yan. Dense label encoding for boundary discontinuity free rotation detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 15819–15829, 2021. 8

[87] Xue Yang and Junchi Yan. Arbitrary-oriented object detection with circular smooth label. In *European Conference on Computer Vision*, pages 677–694. Springer, 2020. 8

[88] Xue Yang, Junchi Yan, Ziming Feng, and Tao He. R3det: Refined single-stage detector with feature refinement for rotating object. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3163–3171, 2021. 3, 8

[89] Xue Yang, Junchi Yan, Qi Ming, Wentao Wang, Xiaopeng Zhang, and Qi Tian. Rethinking rotated object detection with gaussian wasserstein distance loss. In *International Conference on Machine Learning*, pages 11830–11841. PMLR, 2021. 3, 8

[90] Xue Yang, Xiaojiang Yang, Jirui Yang, Qi Ming, Wentao Wang, Qi Tian, and Junchi Yan. Learning high-precision bounding box for rotated object detection via kullback-leibler divergence. *Advances in Neural Information Processing Systems*, 34, 2021. 3, 8

[91] Xue Yang, Yue Zhou, Gefan Zhang, Jitui Yang, Wentao Wang, Junchi Yan, Xiaopeng Zhang, and Qi Tian. The kfiou loss for rotated object detection. *arXiv preprint arXiv:2201.12558*, 2022. 8

[92] Fisher Yu, Vladlen Koltun, and Thomas A. Funkhouser. Dilated residual networks. In *CVPR*, 2017. 3

[93] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, pages 6023–6032, 2019. 5

[94] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 5

[95] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. In *CVPR*, pages 9759–9768, 2020. 2, 5, 9

[96] Wenwei Zhang, Jiangmiao Pang, Kai Chen, and Chen Change Loy. K-Net: Towards unified image segmentation. In *NeurIPS*, 2021. 2, 3

[97] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, and Ruigang Yang. Iou loss for 2d/3d object detection. In *2019 International Conference on 3D Vision (3DV)*, pages 85–94, 2019. 3

[98] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv: Computer Vision and Pattern Recognition*, 2019. 2

[99] Yue Zhou, Xue Yang, Gefan Zhang, Jiabao Wang, Yanyi Liu, Liping Hou, Xue Jiang, Xingzhao Liu, Junchi Yan, Chengqi Lyu, Wenwei Zhang, and Kai Chen. Mmrotate: A rotated object detection benchmark using pytorch. In *Proceedings of the 30th ACM International Conference on Multimedia*, page 7331–7334, 2022. 11