YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications

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 Xiaolin Wei
Meituan Inc.
{lichuyi, lilulu05, jianghongliang02, wengkaiheng, gengyifei,liliang58, kezaidan, liqingyuan02, chengmeng05, nieweiqiang, liyiduo, zhangbo97, liangyufei, zhoulinyuan, xuxiaoming04, chuxiangxiang, weixiaoming, weixiaolin02}@meituan.com

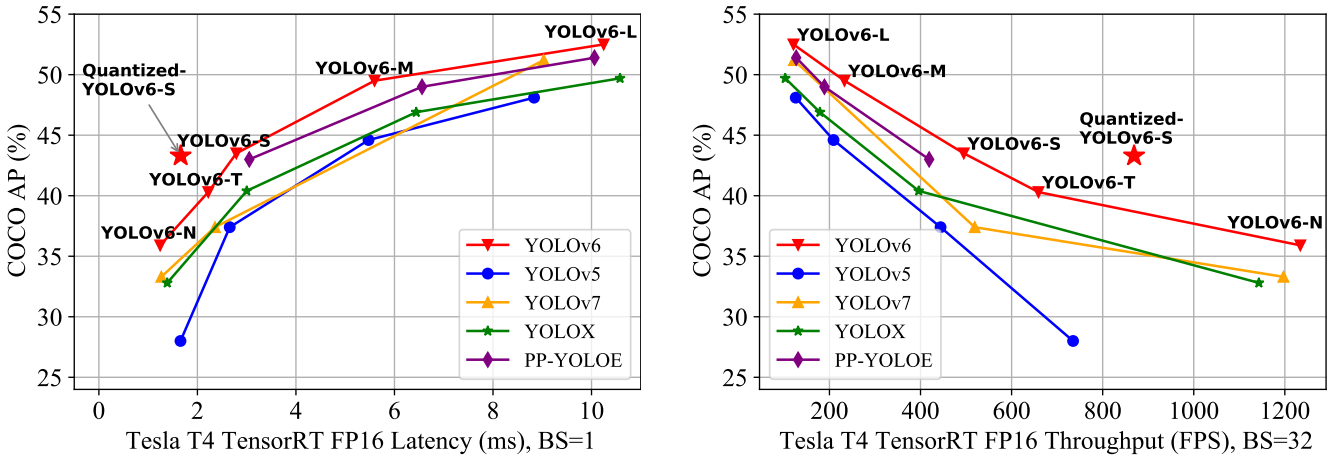


Figure 1: Comparison of state-of-the-art efficient object detectors. Both latency and throughput (at a batch size of 32) are given for a handy reference. All models are test with TensorRT 7 except that the quantized model is with TensorRT 8.

Abstract

For years, YOLO series have been de facto industry-level standard for efficient object detection. The YOLO community has prospered overwhelmingly to enrich its use in a multitude of hardware platforms and abundant scenarios. In this technical report, we strive to push its limits to the next level, stepping forward with an unwavering mindset for industry application. Considering the diverse requirements for speed and accuracy in the real environment, we extensively examine the up-to-date object detection advancements either from industry or academy. Specifically, we heavily assimilate ideas from recent network design, training strategies, testing techniques, quantization and optimization methods. On top of this, we integrate our thoughts and practice to build a suite of deployment-ready networks at various scales to accommodate diversified use cases. With the generous permission of YOLO authors, we name it YOLOv6. We also express our warm welcome to users and contributors for further enhancement. For a glimpse of performance, our YOLOv6-N hits 35.9% AP on COCO dataset at a throughput of 1234 FPS on an NVIDIA Tesla T4 GPU. YOLOv6-S strikes 43.5% AP at 495 FPS, outperforming other mainstream detectors at the same scale (YOLOv5-S, YOLOX-S and PPYOLOE-S). Our quantized version of YOLOv6-S even brings a new state-of-the-art 43.3% AP at 869 FPS. Furthermore, YOLOv6-M/L also achieves better accuracy performance (i.e., 49.5%/52.3%) than other detectors with the similar inference speed. We carefully conducted experiments to validate the effectiveness of each component. Our code is made available at https://github.com/meituan/YOLOv6.

* Equal contributions.
† Corresponding author.

1. Introduction

YOLO series have been the most popular detection frameworks in industrial applications, for its excellent balance between speed and accuracy. Pioneering works of YOLO series are YOLOv1-3 [32–34], which blaze a new trail of one-stage detectors along with the later substantial improvements. YOLOv4 [1] reorganized the detection framework into several separate parts (backbone, neck and head), and verified bag-of-freebies and bag-of-specials at the time to design a framework suitable for training on a single GPU. At present, YOLOv5 [10], YOLOX [7], PPYOLOE [44] and YOLOv7 [42] are all the competing candidates for efficient detectors to deploy. Models at different sizes are commonly obtained through scaling techniques.

In this report, we empirically observed several important factors that motivate us to refurnish the YOLO framework: **(1)** Reparameterization from RepVGG [3] is a superior technique that is not yet well exploited in detection. We also notice that simple model scaling for RepVGG blocks becomes impractical, for which we consider that the elegant consistency of the network design between small and large networks is unnecessary. The plain single-path architecture is a better choice for small networks, but for larger models, the exponential growth of the parameters and the computation cost of the single-path architecture makes it infeasible; **(2)** Quantization of reparameterization-based detectors also requires meticulous treatment, otherwise it would be intractable to deal with performance degradation due to its heterogeneous configuration during training and inference. **(3)** Previous works [7, 10, 42, 44] tend to pay less attention to deployment, whose latencies are commonly compared on high-cost machines like V100. There is a hardware gap when it comes to real serving environment. Typically, low-power GPUs like Tesla T4 are less costly and provide rather good inference performance. **(4)** Advanced domain-specific strategies like label assignment and loss function design need further verifications considering the architectural variance; **(5)** For deployment, we can tolerate the adjustments of the training strategy that improve the accuracy performance but not increase inference costs, such as *knowledge distillation*.

With the aforementioned observations in mind, we bring the birth of YOLOv6, which accomplishes so far the best trade-off in terms of accuracy and speed. We show the comparison of YOLOv6 with other peers at a similar scale in Fig. 1. To boost inference speed without much performance degradation, we examined the cutting-edge quantization methods, including post-training quantization (PTQ) and quantization-aware training (QAT), and accommodate them in YOLOv6 to achieve the goal of deployment-ready networks.

We summarize the main aspects of YOLOv6 as follows:

- We refashion a line of networks of different sizes tailored for industrial applications in diverse scenarios. The architectures at different scales vary to achieve the best speed and accuracy trade-off, where small models feature a plain single-path backbone and large models are built on efficient multi-branch blocks.
- We imbue YOLOv6 with a self-distillation strategy, performed both on the classification task and the regression task. Meanwhile, we dynamically adjust the knowledge from the teacher and labels to help the student model learn knowledge more efficiently during all training phases.
- We broadly verify the advanced detection techniques for label assignment, loss function and data augmentation techniques and adopt them selectively to further boost the performance.
- We reform the quantization scheme for detection with the help of RepOptimizer [2] and channel-wise distillation [36], which leads to an ever-fast and accurate detector with 43.3% COCO AP and a throughput of 869 FPS at a batch size of 32.

2. Method

The renovated design of YOLOv6 consists of the following components, *network design*, *label assignment*, *loss function*, *data augmentation*, *industry-handy improvements*, and *quantization and deployment*:

- **Network Design:** *Backbone*: Compared with other mainstream architectures, we find that RepVGG [3] backbones are equipped with more feature representation power in small networks at a similar inference speed, whereas it can hardly be scaled to obtain larger models due to the explosive growth of the parameters and computational costs. In this regard, we take RepBlock [3] as the building block of our small networks. For large models, we revise a more efficient CSP [43] block, named *CSPStackRep Block*. *Neck*: The neck of YOLOv6 adopts PAN topology [24] following YOLOv4 and YOLOv5. We enhance the neck with RepBlocks or CSPStackRep Blocks to have RepPAN. *Head*: We simplify the decoupled head to make it more efficient, called *Efficient Decoupled Head*.
- **Label Assignment:** We evaluate the recent progress of label assignment strategies [5, 7, 18, 48, 51] on YOLOv6 through numerous experiments, and the results indicate that TAL [5] is more effective and training-friendly.
- **Loss Function:** The loss functions of the mainstream anchor-free object detectors contain *classification loss*,

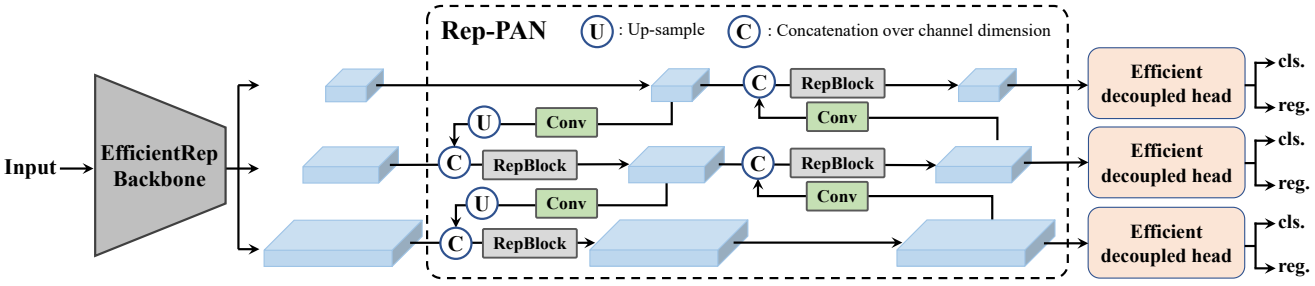


Figure 2: The YOLOv6 framework (N and S are shown). Note for M/L, RepBlocks is replaced with CSPStackRep.

box regression loss and *object loss*. For each loss, we systematically experiment it with all available techniques and finally select VariFocal Loss [50] as our classification loss and SIoU [8]/GIoU [35] Loss as our regression loss.

- **Industry-handy improvements:** We introduce additional common practice and tricks to improve the performance including *self-distillation* and *more training epochs*. For self-distillation, both classification and box regression are respectively supervised by the teacher model. The distillation of box regression is made possible thanks to DFL [20]. In addition, the proportion of information from the soft and hard labels is dynamically declined via cosine decay, which helps the student selectively acquire knowledge at different phases during the training process. In addition, we encounter the problem of the impaired performance without adding extra gray borders at evaluation, for which we provide some remedies.

- **Quantization and deployment:** To cure the performance degradation in quantizing reparameterization-based models, we train YOLOv6 with RepOptimizer [2] to obtain PTQ-friendly weights. We further adopt QAT with channel-wise distillation [36] and graph optimization to pursue extreme performance. Our quantized YOLOv6-S hits a new state of the art with 42.3% AP and a throughput of 869 FPS (batch size=32).

2.1. Network Design

A one-stage object detector is generally composed of the following parts: a backbone, a neck and a head. The backbone mainly determines the feature representation ability, meanwhile, its design has a critical influence on the inference efficiency since it carries a large portion of computation cost. The neck is used to aggregate the low-level physical features with high-level semantic features, and then build up pyramid feature maps at all levels. The head consists of several convolutional layers, and it predicts final detection results according to multi-level features assembled by the neck. It can be categorized as *anchor-based* and *anchor-free*, or rather *parameter-coupled head* and *parameter-decoupled head* from the structure's perspective.

In YOLOv6, based on the principle of hardware-friendly network design [3], we propose two scaled re-parameterizable backbones and necks to accommodate models at different sizes, as well as an efficient decoupled head with the hybrid-channel strategy. The overall architecture of YOLOv6 is shown in Fig. 2.

2.1.1 Backbone

As mentioned above, the design of the backbone network has a great impact on the effectiveness and efficiency of the detection model. Previously, it has been shown that multi-branch networks [13, 14, 38, 39] can often achieve better classification performance than single-path ones [15, 37], but often it comes with the reduction of the parallelism and results in an increase of inference latency. On the contrary, plain single-path networks like VGG [37] take the advantages of high parallelism and less memory footprint, leading to higher inference efficiency. Lately in RepVGG [3], a structural re-parameterization method is proposed to decouple the training-time multi-branch topology with an inference-time plain architecture to achieve a better speed-accuracy trade-off.

Inspired by the above works, we design an efficient re-parameterizable backbone denoted as *EfficientRep*. For small models, the main component of the backbone is RepBlock during the training phase, as shown in Fig. 3 (a). And each RepBlock is converted to stacks of 3 $\times$ 3 convolutional layers (denoted as RepConv) with ReLU activation functions during the inference phase, as shown in Fig. 3 (b). Typically a 3$\times$3 convolution is highly optimized on mainstream GPUs and CPUs and it enjoys higher computational density. Consequently, EfficientRep Backbone sufficiently

utilizes the computing power of the hardware, resulting in a significant decrease in inference latency while enhancing the representation ability in the meantime.

However, we notice that with the model capacity further expanded, the computation cost and the number of parameters in the single-path plain network grow exponentially. To achieve a better trade-off between the computation burden and accuracy, we revise a CSPStackRep Block to build the backbone of medium and large networks. As shown in Fig. 3 (c), CSPStackRep Block is composed of three 1×1 convolution layers and a stack of sub-blocks consisting of two RepVGG blocks [3] or RepConv (at training or inference respectively) with a residual connection. Besides, a *cross stage partial* (CSP) connection is adopted to boost performance without excessive computation cost. Compared with CSPRepResStage [45], it comes with a more succinct outlook and considers the balance between accuracy and speed.

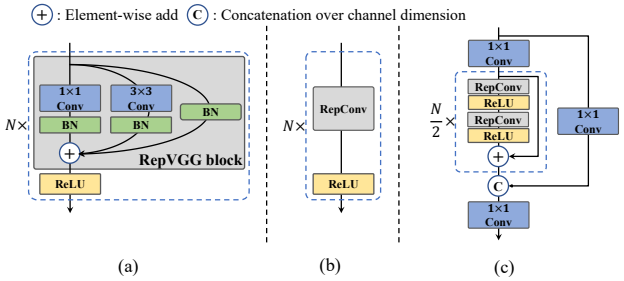


Figure 3: (a) RepBlock is composed of a stack of RepVGG blocks with ReLU activations at training. (b) During inference time, RepVGG block is converted to RepConv. (c) CSPStackRep Block comprises three 1×1 convolutional layers and a stack of sub-blocks of double RepConvs following the ReLU activations with a residual connection.

2.1.2 Neck

In practice, the feature integration at multiple scales has been proved to be a critical and effective part of object detection [9, 21, 24, 40]. We adopt the modified PAN topology [24] from YOLOv4 [1] and YOLOv5 [10] as the base of our detection neck. In addition, we replace the CSP-Block used in YOLOv5 with RepBlock (for small models) or CSPStackRep Block (for large models) and adjust the width and depth accordingly. The neck of YOLOv6 is denoted as Rep-PAN.

2.1.3 Head

Efficient decoupled head The detection head of YOLOv5 is a coupled head with parameters shared between the classification and localization branches, while its counterparts in FCOS [41] and YOLOX [7] decouple the two branches, and additional two 3×3 convolutional layers are introduced in each branch to boost the performance.

In YOLOv6, we adopt a *hybrid-channel* strategy to build a more efficient decoupled head. Specifically, we reduce the number of the middle 3×3 convolutional layers to only one. The width of the head is jointly scaled by the width multiplier for the backbone and the neck. These modifications further reduce computation costs to achieve a lower inference latency.

Anchor-free Anchor-free detectors stand out because of their better generalization ability and simplicity in decoding prediction results. The time cost of its post-processing is substantially reduced. There are two types of anchor-free detectors: anchor point-based [7, 41] and keypoint-based [16, 46, 53]. In YOLOv6, we adopt the anchor point-based paradigm, whose box regression branch actually predicts the distance from the anchor point to the four sides of the bounding boxes.

2.2. Label Assignment

Label assignment is responsible for assigning labels to predefined anchors during the training stage. Previous work has proposed various label assignment strategies ranging from simple IoU-based strategy and inside ground-truth method [41] to other more complex schemes [5, 7, 18, 48, 51].

SimOTA OTA [6] considers the label assignment in object detection as an optimal transmission problem. It defines positive/negative training samples for each ground-truth object from a global perspective. SimOTA [7] is a simplified version of OTA [6], which reduces additional hyper-parameters and maintains the performance. SimOTA was utilized as the label assignment method in the early version of YOLOv6. However, in practice, we find that introducing SimOTA will slow down the training process. And it is not rare to fall into unstable training. Therefore, we desire a replacement for SimOTA.

Task alignment learning Task Alignment Learning (TAL) was first proposed in TOOD [5], in which a unified metric of classification score and predicted box quality is designed. The IoU is replaced by this metric to assign object labels. To a certain extent, the problem of the misalignment of tasks (classification and box regression) is alleviated.

The other main contribution of TOOD is about the task-aligned head (T-head). T-head stacks convolutional layers to build interactive features, on top of which the Task-Aligned Predictor (TAP) is used. PP-YOLOE [45] improved T-head by replacing the layer attention in T-head with the

lightweight ESE attention, forming ET-head. However, we find that the ET-head will deteriorate the inference speed in our models and it comes with no accuracy gain. Therefore, we retain the design of our Efficient decoupled head.

Furthermore, we observed that TAL could bring more performance improvement than SimOTA and stabilize the training. Therefore, we adopt TAL as our default label assignment strategy in YOLOv6.

2.3. Loss Functions

Object detection contains two sub-tasks: *classification* and *localization*, corresponding to two loss functions: *classification loss* and *box regression loss*. For each sub-task, there are various loss functions presented in recent years. In this section, we will introduce these loss functions and describe how we select the best ones for YOLOv6.

2.3.1 Classification Loss

Improving the performance of the classifier is a crucial part of optimizing detectors. Focal Loss [22] modified the traditional cross-entropy loss to solve the problems of class imbalance either between positive and negative examples, or hard and easy samples. To tackle the inconsistent usage of the quality estimation and classification between training and inference, Quality Focal Loss (QFL) [20] further extended Focal Loss with a joint representation of the classification score and the localization quality for the supervision in classification. Whereas VariFocal Loss (VFL) [50] is rooted from Focal Loss [22], but it treats the positive and negative samples asymmetrically. By considering positive and negative samples at different degrees of importance, it balances learning signals from both samples. Poly Loss [17] decomposes the commonly used classification loss into a series of weighted polynomial bases. It tunes polynomial coefficients on different tasks and datasets, which is proved better than Cross-entropy Loss and Focal Loss through experiments.

We assess all these advanced classification losses on YOLOv6 to finally adopt VFL [50].

2.3.2 Box Regression Loss

Box regression loss provides significant learning signals localizing bounding boxes precisely. L1 loss is the original box regression loss in early works. Progressively, a variety of well-designed box regression losses have sprung up, such as IoU-series loss [8, 11, 35, 47, 52, 52] and probability loss [20].

IoU-series Loss IoU loss [47] regresses the four bounds of a predicted box as a whole unit. It has been proved to be effective because of its consistency with the evaluation metric. There are many variants of IoU, such as GIoU [35], DIoU [52], CIoU [52], α-IoU [11] and SIoU [8], etc, forming relevant loss functions. We experiment with GIoU, CIoU and SIoU in this work. And SIoU is applied to YOLOv6-N and YOLOv6-T, while others use GIoU.

Probability Loss Distribution Focal Loss (DFL) [20] simplifies the underlying continuous distribution of box locations as a discretized probability distribution. It considers ambiguity and uncertainty in data without introducing any other strong priors, which is helpful to improve the box localization accuracy especially when the boundaries of the ground-truth boxes are blurred. Upon DFL, DFLv2 [19] develops a lightweight sub-network to leverage the close correlation between distribution statistics and the real localization quality, which further boosts the detection performance. However, DFL usually outputs 17$\times$ more regression values than general box regression, leading to a substantial overhead. The extra computation cost significantly hinders the training of small models. Whilst DFLv2 further increases the computation burden because of the extra sub-network. In our experiments, DFLv2 brings similar performance gain to DFL on our models. Consequently, we only adopt DFL in YOLOv6-M/L. Experimental details can be found in Section 3.3.3.

2.3.3 Object Loss

Object loss was first proposed in FCOS [41] to reduce the score of low-quality bounding boxes so that they can be filtered out in post-processing. It was also used in YOLOX [7] to accelerate convergence and improve network accuracy. As an anchor-free framework like FCOS and YOLOX, we have tried object loss into YOLOv6. Unfortunately, it doesn't bring many positive effects. Details are given in Section 3.

2.4. Industry-handy improvements

The following tricks come ready to use in real practice. They are not intended for a fair comparison but steadily produce performance gain without much tedious effort.

2.4.1 More training epochs

Empirical results have shown that detectors have a progressing performance with more training time. We extended the training duration from 300 epochs to 400 epochs to reach a better convergence.

2.4.2 Self-distillation

To further improve the model accuracy while not introducing much additional computation cost, we apply the clas-

sical knowledge distillation technique minimizing the KL-divergence between the prediction of the teacher and the student. We limit the teacher to be the student itself but pretrained, hence we call it self-distillation. Note that the KL-divergence is generally utilized to measure the difference between data distributions. However, there are two sub-tasks in object detection, in which only the classification task can directly utilize knowledge distillation based on KL-divergence. Thanks to DFL loss [20], we can perform it on box regression as well. The knowledge distillation loss can then be formulated as:

$$L_{KD} = KL(p_t^{cls}||p_s^{cls}) + KL(p_t^{reg}||p_s^{reg}), \quad (1)$$

where p_t^{cls} and p_s^{cls} are class prediction of the teacher model and the student model respectively, and accordingly p_t^{reg} and p_s^{reg} are box regression predictions. The overall loss function is now formulated as:

$$L_{total} = L_{det} + \alpha L_{KD}, \quad (2)$$

where L_{det} is the detection loss computed with predictions and labels. The hyperparameter α is introduced to balance two losses. In the early stage of training, the soft labels from the teacher are easier to learn. As the training continues, the performance of the student will match the teacher so that the hard labels will help students more. Upon this, we apply *cosine weight decay* to α to dynamically adjust the information from hard labels and soft ones from the teacher. We conducted detailed experiments to verify the effect of self-distillation on YOLOv6, which will be discussed in Section 3.

2.4.3 Gray border of images

We notice that a half-stride gray border is put around each image when evaluating the model performance in the implementations of YOLOv5 [10] and YOLOv7 [42]. Although no useful information is added, it helps in detecting the objects near the edge of the image. This trick also applies in YOLOv6.

However, the extra gray pixels evidently reduce the inference speed. Without the gray border, the performance of YOLOv6 deteriorates, which is also the case in [10, 42]. We postulate that the problem is related to the gray borders padding in Mosaic augmentation [1, 10]. Experiments on turning mosaic augmentations off during last epochs [7] (aka. fade strategy) are conducted for verification. In this regard, we change the area of gray border and resize the image with gray borders directly to the target image size. Combining these two strategies, our models can maintain or even boost the performance without the degradation of inference speed.

2.5. Quantization and Deployment

For industrial deployment, it has been common practice to adopt quantization to further speed up runtime without much performance compromise. Post-training quantization (PTQ) directly quantizes the model with only a small calibration set. Whereas quantization-aware training (QAT) further improves the performance with the access to the training set, which is typically used jointly with distillation. However, due to the heavy use of re-parameterization blocks in YOLOv6, previous PTQ techniques fail to produce high performance, while it is hard to incorporate QAT when it comes to matching fake quantizers during training and inference. We here demonstrate the pitfalls and our cures during deployment.

2.5.1 Reparameterizing Optimizer

RepOptimizer [2] proposes gradient re-parameterization at each optimization step. This technique also well solves the quantization problem of reparameterization-based models. We hence reconstruct the re-parameterization blocks of YOLOv6 in this fashion and train it with RepOptimizer to obtain PTQ-friendly weights. The distribution of feature map is largely narrowed (e.g. Fig. 4, more in B.1), which greatly benefits the quantization process, see Sec 3.5.1 for results.

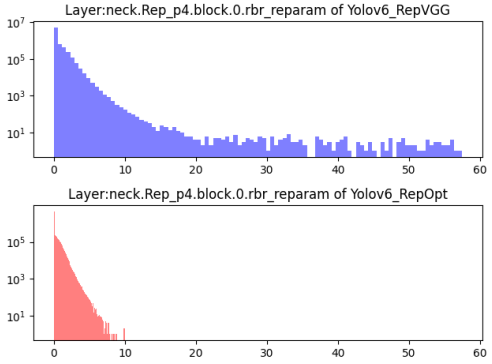


Figure 4: Improved activation distribution of YOLOv6-S trained with RepOptimizer.

2.5.2 Sensitivity Analysis

We further improve the PTQ performance by partially converting quantization-sensitive operations into float computation. To obtain the sensitivity distribution, several metrics are commonly used, mean-square error (MSE), signal-noise ratio (SNR) and cosine similarity. Typically for comparison, one can pick the output feature map (after the activation of a certain layer) to calculate these metrics with and without quantization. As an alternative, it is also viable to

compute validation AP by switching quantization on and off for the certain layer [29].

We compute all these metrics on the YOLOv6-S model trained with RepOptimizer and pick the top-6 sensitive layers to run in float. The full chart of sensitivity analysis can be found in B.2.

2.5.3 Quantization-aware Training with Channel-wise Distillation

In case PTQ is insufficient, we propose to involve quantization-aware training (QAT) to boost quantization performance. To resolve the problem of the inconsistency of fake quantizers during training and inference, it is necessary to build QAT upon the RepOptimizer. Besides, channel-wise distillation [36] (later as CW Distill) is adapted within the YOLOv6 framework, shown in Fig. 5. This is also a self-distillation approach where the teacher network is the student itself in FP32-precision. See experiments in Sec 3.5.1.

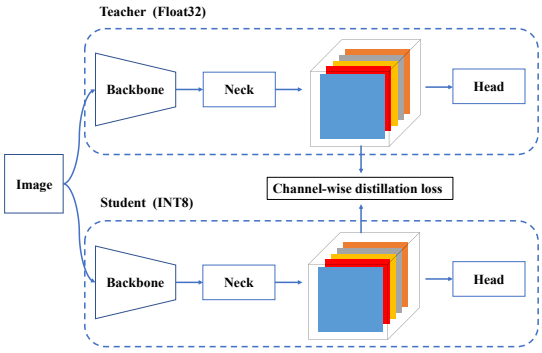


Figure 5: Schematic of YOLOv6 channel-wise distillation in QAT.

3. Experiments

3.1. Implementation Details

We use the same optimizer and the learning schedule as YOLOv5 [10], *i.e.* stochastic gradient descent (SGD) with momentum and cosine decay on learning rate. Warm-up, grouped weight decay strategy and the exponential moving average (EMA) are also utilized. We adopt two strong data augmentations (Mosaic [1,10] and Mixup [49]) following [1,7,10]. A complete list of hyperparameter settings can be found in our released code. We train our models on the COCO 2017 [23] training set, and the accuracy is evaluated on the COCO 2017 validation set. All our models are trained on 8 NVIDIA A100 GPUs, and the speed performance is measured on an NVIDIA Tesla T4 GPU with TensorRT version 7.2 unless otherwise stated. And the speed performance measured with other TensorRT versions or on other devices is demonstrated in Appendix A.

3.2. Comparisons

Considering that the goal of this work is to build networks for industrial applications, we primarily focus on the speed performance of all models after deployment, including throughput (FPS at a batch size of 1 or 32) and the GPU latency, rather than FLOPs or the number of parameters. We compare YOLOv6 with other state-of-the-art detectors of YOLO series, including YOLOv5 [10], YOLOX [7], PPYOLOE [45] and YOLOv7 [42]. Note that we test the speed performance of all official models with FP16-precision on the same Tesla T4 GPU with TensorRT [28]. The performance of YOLOv7-Tiny is re-evaluated according to their open-sourced code and weights at the input size of 416 and 640. Results are shown in Table 1 and Fig. 1. Compared with YOLOv5-N/YOLOv7-Tiny (input size=416), our YOLOv6-N has significantly advanced by 7.9%/2.6% respectively. It also comes with the best speed performance in terms of both throughput and latency. Compared with YOLOX-S/PPYOLOE-S, YOLOv6-S can improve AP by 3.0%/0.4% with higher speed. We compare YOLOv5-S and YOLOv7-Tiny (input size=640) with YOLOv6-T, our method is 2.9% more accurate and 73/25 FPS faster with a batch size of 1. YOLOv6-M outperforms YOLOv5-M by 4.2% higher AP with a similar speed, and it achieves 2.7%/0.6% higher AP than YOLOX-M/PPYOLOE-M at a higher speed. Besides, it is more accurate and faster than YOLOv5-L. YOLOv6-L is 2.8%/1.1% more accurate than YOLOX-L/PPYOLOE-L under the same latency constraint. We additionally provide a faster version of YOLOv6-L by replacing SiLU with ReLU (denoted as YOLOv6-L-ReLU). It achieves 51.7% AP with a latency of 8.8 ms, outperforming YOLOX-L/PPYOLOE-L/YOLOv7 in both accuracy and speed.

3.3. Ablation Study

3.3.1 Network

Backbone and neck We explore the influence of single-path structure and multi-branch structure on backbones and necks, as well as the channel coefficient (denoted as CC) of CSPStackRep Block. All models described in this part adopt TAL as the label assignment strategy, VFL as the classification loss, and GIoU with DFL as the regression loss. Results are shown in Table 2. We find that the optimal network structure for models at different sizes should come up with different solutions.

For YOLOv6-N, the single-path structure outperforms the multi-branch structure in terms of both accuracy and speed. Although the single-path structure has more FLOPs and parameters than the multi-branch structure, it could

Method	Input Size	AP^{val}	AP^{val}_{50}	FPS (bs=1)	FPS (bs=32)	Latency (bs=1)	Params	FLOPs
YOLOv5-N [10]	640	28.0%	45.7%	602	735	1.7 ms	1.9 M	4.5 G
YOLOv5-S [10]	640	37.4%	56.8%	376	444	2.7 ms	7.2 M	16.5 G
YOLOv5-M [10]	640	45.4%	64.1%	182	209	5.5 ms	21.2 M	49.0 G
YOLOv5-L [10]	640	49.0%	67.3%	113	126	8.8 ms	46.5 M	109.1 G
YOLOX-Tiny [7]	416	32.8%	50.3%*	717	1143	1.4 ms	5.1 M	6.5 G
YOLOX-S [7]	640	40.5%	59.3%*	333	396	3.0 ms	9.0 M	26.8 G
YOLOX-M [7]	640	46.9%	65.6%*	155	179	6.4 ms	25.3 M	73.8 G
YOLOX-L [7]	640	49.7%	68.0%*	94	103	10.6 ms	54.2 M	155.6 G
PPYOLOE-S [45]	640	43.1%	59.6%	327	419	3.1 ms	7.9 M	17.4 G
PPYOLOE-M [45]	640	49.0%	65.9%	152	189	6.6 ms	23.4 M	49.9 G
PPYOLOE-L [45]	640	51.4%	68.6%	101	127	10.1 ms	52.2 M	110.1 G
YOLOv7-Tiny [42]	416	33.3%*	49.9%*	787	1196	1.3 ms	6.2 M	5.8 G
YOLOv7-Tiny [42]	640	37.4%*	55.2%*	424	519	2.4 ms	6.2 M	13.7 G*
YOLOv7 [42]	640	51.2%	69.7%	110	122	9.0 ms	36.9 M	104.7 G
YOLOv6-N	640	35.9%	51.2%	802	1234	1.2 ms	4.3 M	11.1 G
YOLOv6-T	640	40.3%	56.6%	449	659	2.2 ms	15.0 M	36.7 G
YOLOv6-S	640	43.5%	60.4%	358	495	2.8 ms	17.2 M	44.2 G
YOLOv6-M‡	640	49.5%	66.8%	179	233	5.6 ms	34.3 M	82.2 G
YOLOv6-L-ReLU‡	640	51.7%	69.2%	113	149	8.8 ms	58.5 M	144.0 G
YOLOv6-L‡	640	52.5%	70.0%	98	121	10.2 ms	58.5 M	144.0 G

Table 1: Comparisons with other YOLO-series detectors on COCO 2017 *val*. FPS and latency are measured in FP16-precision on a Tesla T4 in the same environment with TensorRT. All our models are trained for 300 epochs without pre-training or any external data. Both the accuracy and the speed performance of our models are evaluated with the input resolution of 640×640. '‡' represents that the proposed self-distillation method is utilized. '∗' represents the re-evaluated result of the released model through the official code.

run faster due to a relatively lower memory footprint and a higher degree of parallelism. For YOLOv6-S, the two block styles bring similar performance. When it comes to larger models, multi-branch structure achieves better performance in accuracy and speed. And we finally select multi-branch with a channel coefficient of 2/3 for YOLOv6-M and 1/2 for YOLOv6-L.

Furthermore, we study the influence of width and depth of the neck on YOLOv6-L. Results in Table 3 show that the slender neck performs 0.2% better than the wide-shallow neck with the similar speed.

Combinations of convolutional layers and activation functions YOLO series adopted a wide range of activation functions, ReLU [27], LReLU [25], Swish [31], SiLU [4], Mish [26] and so on. Among these activation functions, SiLU is the most used. Generally speaking, SiLU performs with better accuracy and does not cause too much extra computation cost. However, when it comes to industrial applications, especially for deploying models with TensorRT [28] acceleration, ReLU has a greater speed advantage because of its fusion into convolution.

Models	Block	CC	AP^{val}	FPS (bs=32)	Params	FLOPs
YOLOv6-N	RepBlock	-	35.2%	1237	4.3M	11.1G
	CSPStackRep Block	1/2	32.7%	1257	2.3M	5.6G
YOLOv6-S	RepBlock	-	43.2%	499	17.2M	44.2G
	CSPStackRep Block	1/2	43.4%	511	11.5M	27.7G
YOLOv6-M	RepBlock	-	47.9%	137	67.1M	175.6G
	CSPStackRep Block	2/3	48.1%	237	34.3M	82.2G
	CSPStackRep Block	1/2	47.3%	237	27.7M	68.4G
YOLOv6-L	CSPStackRep Block	2/3	50.1%	149	54.7M	142.7G
	CSPStackRep Block	1/2	50.1%	151	58.5M	144.0G

Table 2: Ablation study on backbones and necks. YOLOv6-L here is equipped with ReLU.

Moreover, we further verify the effectiveness of combinations of RepConv/ordinary convolution (denoted as Conv) and ReLU/SiLU/LReLU in networks of different sizes to achieve a better trade-off. As shown in Table 4, Conv with SiLU performs the best in accuracy while the combination of RepConv and ReLU achieves a better trade-off. We suggest users adopt RepConv with ReLU in latency-sensitive applications. We choose to use RepConv/ReLU combi-

Width	Depth	AP^{val}	FPS (bs=32)	Params	FLOPs
[192, 384, 768]	5	50.8%	123	58.6 M	144.7 G
[128, 256, 512]	12	51.0%	122	58.5 M	144.0 G

Table 3: Ablation study on the neck settings of YOLOv6-L. SiLU is selected as the activation function.

Model	Conv.	Act.	AP^{val}	FPS (bs=32)
YOLOv6-N	Conv	SiLU	**36.6%**	963
	RepConv	SiLU	36.5%	971
	Conv	ReLU	34.8%	**1246**
	RepConv	ReLU	35.2%	1233
	Conv	LReLU	35.4%	983
	RepConv	LReLU	35.6%	975
YOLOv6-M	Conv	SiLU	**48.9%**	180
	RepConv	SiLU	**48.9%**	180
	Conv	ReLU	47.7%	235
	RepConv	ReLU	48.1%	**236**
	Conv	LReLU	48.0%	185
	RepConv	LReLU	48.1%	187

Table 4: Ablation study on combinations of different types of convolutional layers (denoted as Conv.) and activation layers (denoted as Act.).

nation in YOLOv6-N/T/S/M for higher inference speed and use the Conv/SiLU combination in the large model YOLOv6-L to speed up training and improve performance.

Miscellaneous design We also conduct a series of ablation on other network parts mentioned in Section 2.1 based on YOLOv6-N. We choose YOLOv5-N as the baseline and add other components incrementally. Results are shown in Table 5. Firstly, with decoupled head (denoted as DH), our model is 1.4% more accurate with 5% increase in time cost. Secondly, we verify that the anchor-free paradigm is 51% faster than the anchor-based one for its 3× less predefined anchors, which results in less dimensionality of the output. Further, the unified modification of the backbone (EfficientRep Backbone) and the neck (Rep-PAN neck), denoted as EB+RN, brings 3.6% AP improvements, and runs 21% faster. Finally, the optimized decoupled head (hybrid channels, HC) brings 0.2% AP and 6.8% FPS improvements in accuracy and speed respectively.

3.3.2 Label Assignment

In Table 6, we analyze the effectiveness of mainstream label assign strategies. Experiments are conducted on YOLOv6-N. As expected, we observe that SimOTA and TAL are the best two strategies. Compared with the ATSS, SimOTA can increase AP by 2.0%, and TAL brings 0.5% higher AP than SimOTA. Considering the stable training and better accuracy performance of TAL, we adopt TAL as our label assignment strategy.

In addition, the implementation of TOOD [5] adopts ATSS [51] as the warm-up label assignment strategy during the early training epochs. We also retain the warm-up strategy and further make some explorations on it. Details are shown in Table 7, and we can find that without warm-up or warmed up by other strategies (i.e., SimOTA) it can also achieve the similar performance.

DH	AF	EB+RN	HC	AP^{val}	FPS (bs=32)
✗	✗	✗	✗	28.0%	672
✓	✗	✗	✗	29.4%	637
✓	✓	✗	✗	30.7%	962
✓	✓	✓	✗	34.3%	1163
✓	✓	✓	✓	**34.5%**	**1242**

Table 5: Ablation study on all network designs in an incremental way. FPS is tested with FP16-precision and batch-size=32 on Tesla T4 GPUs.

Method	AP^{val}
ATSS [51]	32.5%
SimOTA [7]	34.5%
TAL [5]	**35.0%**
DW [18]	33.4%
ObjectBox [48]	30.1%

Table 6: Comparisons of label assignment methods.

Warmup strategy	AP^{val}
w/o	34.9%
ATSS [51]	**35.0%**
SimOTA [7]	34.9%

Table 7: Comparisons of label assignment methods in warm-up stage.

3.3.3 Loss functions

In the object detection framework, the loss function is composed of a classification loss, a box regression loss and an optional object loss, which can be formulated as follows:

$$L_{det} = L_{cls} + \lambda L_{reg} + \mu L_{obj}, \tag{3}$$

where L_{cls}, L_{reg} and L_{obj} are classification loss, regression loss and object loss. λ and μ are hyperparameters.

Model	Classification Loss	AP^{val}
YOLOv6-N	Focal Loss [22]	35.0%
	Poly Loss [17]	34.0%
	QFL [20]	**35.4**%
	VFL [50]	35.2%
YOLOv6-S	Focal Loss [22]	42.9%
	Poly Loss [17]	41.5%
	QFL [20]	43.1%
	VFL [50]	**43.2**%
YOLOv6-M	Focal Loss [22]	48.0%
	Poly Loss [17]	46.9%
	QFL [20]	48.0%
	VFL [50]	**48.1**%

Table 8: Ablation study on classification loss functions.

In this subsection, we evaluate each loss function on YOLOv6. Unless otherwise specified, the baselines for YOLOv6-N, YOLOv6-S and YOLOv6-M are 35.0%, 42.9% and 48.0% trained with TAL, Focal Loss and GIoU Loss.

Classification Loss We experiment Focal Loss [22], Poly loss [17], QFL [20] and VFL [50] on YOLOv6-N/S/M. As can be seen in Table 8, VFL brings 0.2%/0.3%/0.1% AP improvements on YOLOv6-N/S/M respectively compared with Focal Loss. We choose VFL as the classification loss function.

Regression Loss IoU-series and probability loss functions are both experimented with on YOLOv6-N/S/M.

The latest IoU-series losses are utilized in YOLOv6-N/S/M. Experiment results in Table 9 show that SIoU Loss outperforms others for YOLOv6-N and YOLOv6-T, while CIoU Loss performs better on YOLOv6-M.

For probability losses, as listed in Table 10, introducing DFL can obtain 0.2%/0.1%/0.2% performance gain for YOLOv6-N/S/M respectively. However, the inference speed is greatly affected for small models. Therefore, DFL is only introduced in YOLOv6-M/L.

Object Loss Object loss is also experimented with YOLOv6, as shown in Table 11. From Table 11, we can see that object loss has negative effects on YOLOv6-N/S/M networks, where the maximum decrease is 1.1% AP on YOLOv6-N. The negative gain may come from the conflict between the object branch and the other two branches in TAL. Specifically, in the training stage, IoU between predicted boxes and ground-truth ones, as well as classification scores are used to jointly build a metric as the criteria to assign labels. However, the introduced object branch extends the number of tasks to be aligned from two to three, which obviously increases the difficulty. Based on the experimental results and this analysis, the object loss is then discarded in YOLOv6.

Model	Loss	AP^{val}
YOLOv6-N	GIoU [35]	35.1%
	CIoU [52]	35.1%
	SIoU [8]	**35.5**%
YOLOv6-S	GIoU [35]	43.1%
	CIoU [52]	43.1%
	SIoU [8]	**43.3**%
YOLOv6-M	GIoU [35]	48.2%
	CIoU [52]	**48.3**%
	SIoU [8]	48.1%

Table 9: Ablation study on IoU-series box regression loss functions. The classification loss is VFL [50].

Method	Loss	AP^{val}	FPS (bs=32)
YOLOv6-N	w/o	35.0%	**1226**
	DFL [20]	**35.2**%	1022
	DFLv2 [19]	**35.2**%	819
YOLOv6-S	w/o	42.9%	**486**
	DFL [20]	**43.0**%	461
	DFLv2 [19]	**43.0**%	422
YOLOv6-M	w/o	48.0%	233
	DFL [20]	48.2%	**236**
	DFLv2 [19]	**48.3**%	226

Table 10: Ablation study on probability loss functions.

Method	Object Loss	AP^{val}
YOLOv6-N	✗	**35.0**%
	✓	33.9%
YOLOv6-S	✗	**42.9**%
	✓	41.4%
YOLOv6-M	✗	**48.0**%
	✓	46.5%

Table 11: Effectiveness of object loss.

3.4. Industry-handy improvements

More training epochs In practice, more training epochs is a simple and effective way to further increase the accuracy. Results of our small models trained for 300 and 400 epochs are shown in Table 12. We observe that training for longer epochs substantially boosts AP by 0.4%, 0.6%, 0.5% for YOLOv6-N, T, S respectively. Considering the acceptable cost and the produced gain, it suggests that training for

Model	300 epochs	400 epochs
YOLOv6-N	35.9%	36.3%
YOLOv6-T	40.3%	40.9%
YOLOv6-S	43.4%	43.9%

Table 12: Experiments of more training epochs on small models.

400 epochs is a better convergence scheme for YOLOv6.

Cls.	Reg.	Weight Decay	AP^{val}
✗	✗	✗	51.0%
✓	✗	✗	51.4%
✓	✓	✗	51.7%
✓	✓	✓	**52.3**%

Table 13: Ablation study on the self-distillation.

Self-distillation We conducted detailed experiments to verify the proposed self-distillation method on YOLOv6-L. As can be seen in Table 13, applying the self-distillation only on the classification branch can bring 0.4% AP improvement. Furthermore, we simply perform the self-distillation on the box regression task to have 0.3% AP increase. The introduction of weight decay boosts the model by 0.6% AP.

Gray border of images In Section 2.4.3, we introduce a strategy to solve the problem of performance degradation without extra gray borders. Experimental results are shown in Table 14. In these experiments, YOLOv6-N and YOLOv6-S are trained for 400 epochs and YOLOv6-M for 300 epochs. It can be observed that the accuracy of YOLOv6-N/S/M is lowered by 0.4%/0.5%/0.7% without Mosaic fading when removing the gray border. However, the performance degradation becomes 0.2%/0.5%/0.5% when adopting Mosaic fading, from which we find that, on the one hand, the problem of performance degradation is mitigated. On the other hand, the accuracy of small models (YOLOv6-N/S) is improved whether we pad gray borders or not. Moreover, we limit the input images to 634×634 and add gray borders by 3 pixels wide around the edges (more results can be found in Appendix C). With this strategy, the size of the final images is the expected 640×640. The results in Table 14 indicate that the final performance of YOLOv6-N/S/M is even 0.2%/0.3%/0.1% more accurate with the final image size reduced from 672 to 640.

Model	Image Size	Border Size	Close Mosaic	AP^{val}
YOLOv6-N	640	16	✗	36.0%
	640	16	✓	**36.2**%
	640	0	✗	35.6%
	640	0	✓	36.0%
	634	3	✓	**36.2**%
YOLOv6-S	640	16	✗	43.4%
	640	16	✓	**43.9**%
	640	0	✗	42.9%
	640	0	✓	43.4%
	634	3	✓	43.7%
YOLOv6-M	640	16	✗	48.4%
	640	16	✓	48.4%
	640	0	✗	47.7%
	640	0	✓	47.9%
	634	3	✓	**48.5**%

Table 14: Experimental results about the strategies for solving the problem of the performance degradation without extra gray border.

3.5. Quantization Results

We take YOLOv6-S as an example to validate our quantization method. The following experiment is on both two releases. The baseline model is trained for 300 epochs.

3.5.1 PTQ

The average performance is substantially improved when the model is trained with RepOptimizer, see Table 15. RepOptimizer is in general faster and nearly identical.

Model	FP32 AP^{val}	PTQ INT8 AP^{val}
(v1.0)		
YOLOv6-S	42.4	35.0
YOLOv6-S w/ RepOptimizer	42.4	40.9
(v2.0)		
YOLOv6-S	43.4	41.3
YOLOv6-S w/ RepOptimizer	43.1	42.6

Table 15: PTQ performance of YOLOv6s trained with RepOptimizer.

3.5.2 QAT

For v1.0, we apply fake quantizers to non-sensitive layers obtained from Section 2.5.2 to perform quantization-aware training and call it partial QAT. We compare the result with

full QAT in Table 16. Partial QAT leads to better accuracy with a slightly reduced throughput.

Model	INT8 AP^{val}	FPS
YOLOv6-S	35.0	556
YOLOv6-S + RepOpt + Partial QAT + CW Distill	42.3	503
YOLOv6-S + RepOpt + QAT + CW Distill	42.1	528

Table 16: QAT performance of YOLOv6-S (v1.0) under different settings.

Due to the removal of quantization-sensitive layers in v2.0 release, we directly use full QAT on YOLOv6-S trained with RepOptimizer. We eliminate inserted quantizers through graph optimization to obtain higher accuracy and faster speed. We compare the distillation-based quantization results from PaddleSlim [30] in Table 17. Note our quantized version of YOLOv6-S is the fastest and the most accurate, also see Fig. 1.

Model	AP^{val}	$FPS^{bs=1}$	$FPS^{bs=32}$
YOLOv5-S [30]	36.9	502†	N/A
YOLOv6-S* [30]	41.3	579†	N/A
YOLOv7-Tiny [30]	37.0	512†	N/A
YOLOv6-S (FP16)	43.4	377†	541†
YOLOv6-S (Our QAT strategy)	43.3	596†	869†

Table 17: QAT performance of YOLOv6-S (v2.0) compared with other quantized detectors. '*': based on v1.0 release. '†': We tested with TensorRT 8 on Tesla T4 with a batch size of 1 and 32.

4. Conclusion

In a nutshell, with the persistent industrial requirements in mind, we present the current form of YOLOv6, carefully examining all the advancements of components of object detectors up to date, meantime instilling our thoughts and practices. The result surpasses other available real-time detectors in both accuracy and speed. For the convenience of the industrial deployment, we also supply a customized quantization method for YOLOv6, rendering an ever-fast detector out-of-box. We sincerely thank the academic and industrial community for their brilliant ideas and endeavors. In the future, we will continue expanding this project to meet higher standards and more demanding scenarios.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 2, 4, 6, 7

[2] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Kaiqi Huang, Jungong Han, and Guiguang Ding. Re-parameterizing your optimizers rather than architectures. *arXiv preprint arXiv:2205.15242*, 2022. 2, 3, 6

[3] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13733–13742, 2021. 2, 3, 4

[4] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. 8

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

[6] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and Jian Sun. Ota: Optimal transport assignment for object detection. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 303–312, 2021. 4

[7] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021. 2, 4, 5, 6, 7, 8, 9, 15

[8] Zhora Gevorgyan. Siou loss: More powerful learning for bounding box regression. *arXiv preprint arXiv:2205.12740*, 2022. 3, 5, 10

[9] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7036–7045, 2019. 4

[10] Jocher Glenn. YOLOv5 release v6.1. https://github.com/ultralytics/yolov5/releases/tag/v6.1, 2022. 2, 4, 6, 7, 8, 15

[11] Jiabo He, Sarah Erfani, Xingjun Ma, James Bailey, Ying Chi, and Xian-Sheng Hua. α-iou: A family of power intersection over union losses for bounding box regression. *Advances in Neural Information Processing Systems*, 34:20230–20242, 2021. 5

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 3

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 3

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 3

[16] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European conference on computer vision (ECCV)*, pages 734–750, 2018. 4

[17] Zhaoqi Leng, Mingxing Tan, Chenxi Liu, Ekin Dogus Cubuk, Xiaojie Shi, Shuyang Cheng, and Dragomir Anguelov. Polyloss: A polynomial expansion perspective of classification loss functions. *arXiv preprint arXiv:2204.12511*, 2022. 5, 10

[18] Shuai Li, Chenhang He, Ruihuang Li, and Lei Zhang. A dual weighting label assignment scheme for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9387–9396, June 2022. 2, 4, 9

[19] Xiang Li, Wenhai Wang, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss v2: Learning reliable localization quality estimation for dense object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11632–11641, 2021. 5, 10

[20] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. *Advances in Neural Information Processing Systems*, 33:21002–21012, 2020. 3, 5, 6, 10

[21] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 4

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 5, 10

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 7

[24] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018. 2, 4

[25] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013. 8

[26] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 4(2):10–48550, 2019. 8

[27] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010. 8

[28] NVIDIA. TensorRT. https://developer.nvidia.com/tensorrt, 2018. 7, 8

[29] NVIDIA. pytorch-quantization's documentation. https://docs.nvidia.com/deeplearning/tensorrt/pytorch-quantization-toolkit/docs/index.html, 2021. 7

[30] PaddleSlim. PaddleSlim documentation. https://github.com/PaddlePaddle/PaddleSlim/tree/develop/example/auto_compression/pytorch_yolo_series, 2022. 12

[31] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. 8

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2

[33] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. 2

[34] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 2

[35] 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. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 658–666, 2019. 3, 5, 10

[36] Changyong Shu, Yifan Liu, Jianfei Gao, Zheng Yan, and Chunhua Shen. Channel-wise knowledge distillation for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5311–5320, 2021. 2, 3, 7

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 3

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 3

[39] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 3

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

[41] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proc. Int. Conf. Computer Vision (ICCV)*, 2019. 4, 5

[42] 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. *arXiv preprint arXiv:2207.02696*, 2022. 2, 6, 7, 8, 15

[43] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020. 2

[44] Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao Chang, Cheng Cui, Kaipeng Deng, Guanzhong Wang, Qingqing Dang, Shengyu Wei, Yuning Du, et al. Pp-yoloe: An evolved version of yolo. *arXiv preprint arXiv:2203.16250*, 2022. 2

[45] Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao Chang, Cheng Cui, Kaipeng Deng, Guanzhong Wang, Qingqing Dang, Shengyu Wei, Yuning Du, et al. Pp-yoloe: An evolved version of yolo. *arXiv preprint arXiv:2203.16250*, 2022. 4, 7, 8, 15

[46] Ze Yang, Shaohui Liu, Han Hu, Liwei Wang, and Stephen Lin. Reppoints: Point set representation for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9657–9666, 2019. 4

[47] Jiahui Yu, Yuning Jiang, Zhangyang Wang, Zhimin Cao, and Thomas Huang. Unitbox: An advanced object detection network. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 516–520, 2016. 5

[48] Mohsen Zand, Ali Etemad, and Michael A. Greenspan. Objectbox: From centers to boxes for anchor-free object detection. *ArXiv*, abs/2207.06985, 2022. 2, 4, 9

[49] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 7

[50] Haoyang Zhang, Ying Wang, Feras Dayoub, and Niko Sunderhauf. Varifocalnet: An iou-aware dense object detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8514–8523, 2021. 3, 5, 10

[51] 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*, 2020. 2, 4, 9

[52] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020. 5, 10

[53] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 4

A. Detailed Latency and Throughput Benchmark

A.1. Setup

Unless otherwise stated, all the reported latency is measured on an NVIDIA Tesla T4 GPU with TensorRT version 7.2.1.6. Due to the large variance of the hardware and software settings, we re-measure latency and throughput of all the models under the same configuration (both hardware and software). For a handy reference, we also switch TensorRT versions (Table 18) for consistency check. Latency on a V100 GPU (Table 19) is included for a convenient comparison. This gives us a full spectrum view of state-of-the-art detectors.

A.2. T4 GPU Latency Table with TensorRT 8

See Table 18. The throughput of YOLOv6 models still emulates their peers.

Method	FPS (bs=1)	FPS (bs=32)	Latency (bs=1)
YOLOv5-N [10]	702	843	1.4 ms
YOLOv5-S [10]	433	515	2.3 ms
YOLOv5-M [10]	202	235	4.9 ms
YOLOv5-L [10]	126	137	7.9 ms
YOLOX-Tiny [7]	766	1393	1.3 ms
YOLOX-S [7]	313	489	2.6 ms
YOLOX-M [7]	159	204	5.3 ms
YOLOX-L [7]	104	117	9.0 ms
PPYOLOE-S [45]	357	493	2.8 ms
PPYOLOE-M [45]	163	210	6.1 ms
PPYOLOE-L [45]	110	145	9.1 ms
YOLOv7-Tiny [42]	464	568	2.1 ms
YOLOv7 [42]	128	135	7.6 ms
YOLOv6-N	810	1323	1.2 ms
YOLOv6-T	469	677	2.1 ms
YOLOv6-S	362	522	2.7 ms
YOLOv6-M	180	241	5.6 ms
YOLOv6-L-ReLU	111	145	9.0 ms
YOLOv6-L	105	131	9.6 ms

Table 18: YOLO-series comparison of latency and throughput on a T4 GPU with a higher version of TensorRT (8.2).

A.3. V100 GPU Latency Table

See Table 19. The speed advantage of YOLOv6 is largely maintained.

A.4. CPU Latency

We evaluate the performance of our models and other competitors on a 2.6 GHz Intel Core i7 CPU using OpenCV Deep Neural Network (DNN), as shown in Table 20.

Method	FPS (bs=1)	FPS (bs=32)	Latency (bs=1)
YOLOv5-N [10]	709	1891	1.3 ms
YOLOv5-S [10]	571	1354	1.6 ms
YOLOv5-M [10]	332	685	2.9 ms
YOLOv5-L [10]	215	426	4.5 ms
YOLOX-Tiny [7]	739	3271	1.3 ms
YOLOX-S [7]	515	1254	1.9 ms
YOLOX-M [7]	308	605	3.2 ms
YOLOX-L [7]	189	370	5.2 ms
PPYOLOE-S [45]	435	1117	2.2 ms
PPYOLOE-M [45]	263	583	3.8 ms
PPYOLOE-L [45]	194	415	5.1 ms
YOLOv7-Tiny [42]	647	1269	1.5 ms
YOLOv7 [42]	229	425	4.3 ms
YOLOv6-N	752	2031	1.0 ms
YOLOv6-T	604	1724	1.4 ms
YOLOv6-S	507	1546	1.7 ms
YOLOv6-M	287	750	3.4 ms
YOLOv6-L-ReLU	198	476	5.0 ms
YOLOv6-L	175	416	5.6 ms

Table 19: YOLO-series comparison of latency and throughput on a V100 GPU. We measure all models at FP16-precision with the input size 640×640 in the exact same environment.

Method	Input	Latency (bs=1)
YOLOv5-N [10]	640	118.9 ms
YOLOv5-S [10]	640	202.2 ms
YOLOX-Tiny [7]	416	144.2 ms
YOLOX-S [7]	640	164.6 ms
YOLOX-M [7]	640	357.9 ms
YOLOv7-Tiny [42]	640	137.5 ms
YOLOv6-N	640	70.0 ms
YOLOv6-T	640	128.1 ms
YOLOv6-S	640	163.4 ms

Table 20: YOLO-series comparison of latency on a typical CPU. We measure all models at FP32-precision with the input size 640×640 in the exact same environment.

B. Quantization Details

B.1. Feature Distribution Comparison

We illustrate the feature distribution of more layers that are much alleviated after trained with RepOptimizer, see

Fig. 6.

B.2. Sensitivity Analysis Results

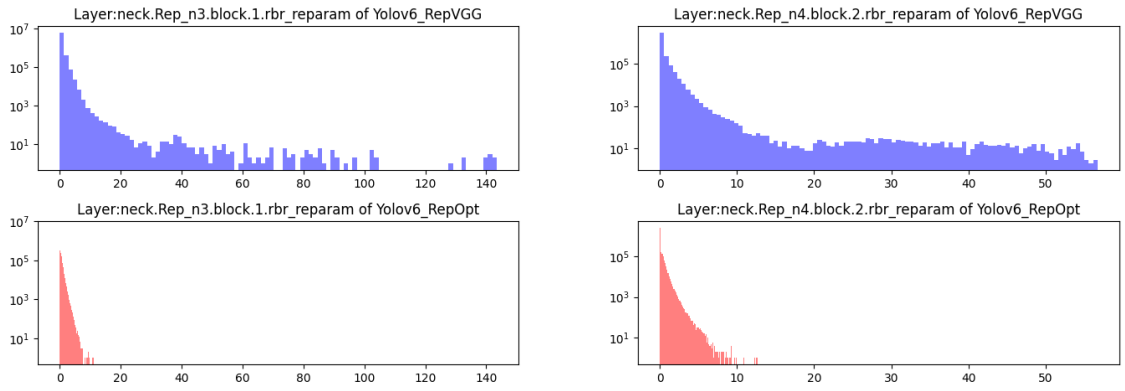
See Fig. 7, we observe that SNR and Cosine similarity gives highly correlated results. However, directly evaluating AP produces a different panorama. Nevertheless, in terms of final quantization performance, MSE is the closest to direct AP evaluation, see Table 21.

Model	AP^{val}
MSE	41.5
Cosine Similarity	41.1
SNR	41.1
Direct AP Evaluation	42.0

Table 21: Partial post-training quantization performance w.r.t. difference sensitivity metrics.

C. Analysis of Gray Border

To analyze the effect of the gray border, we further explore different border settings with the loaded images resized to different sizes and padded to 640×640. For example, when the image size is 608, and the border size is set to 16, and so on. In addition, we alleviate a problem of the information misalignment between pre-processing and post-processing via a simple adjustment. Results are shown in Fig. 8. We can observe that each model achieves the best AP with a different border size. Additionally, compared with an input size of 640, our models get about 0.3% higher AP on average if the input image size locates in the range from 632 to 638.



(a) (b)

Figure 6: More examples of better optimized layers in YOLOv6s that are otherwise hard to quantize.

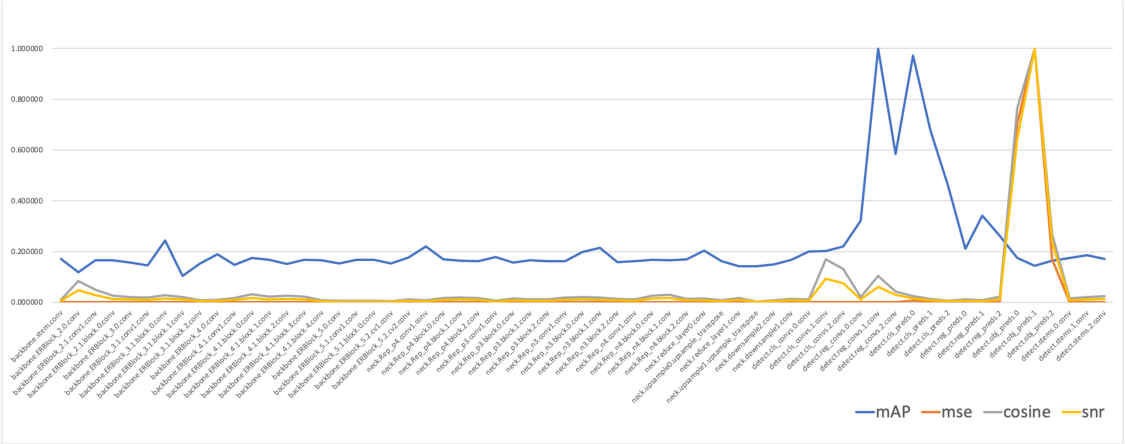
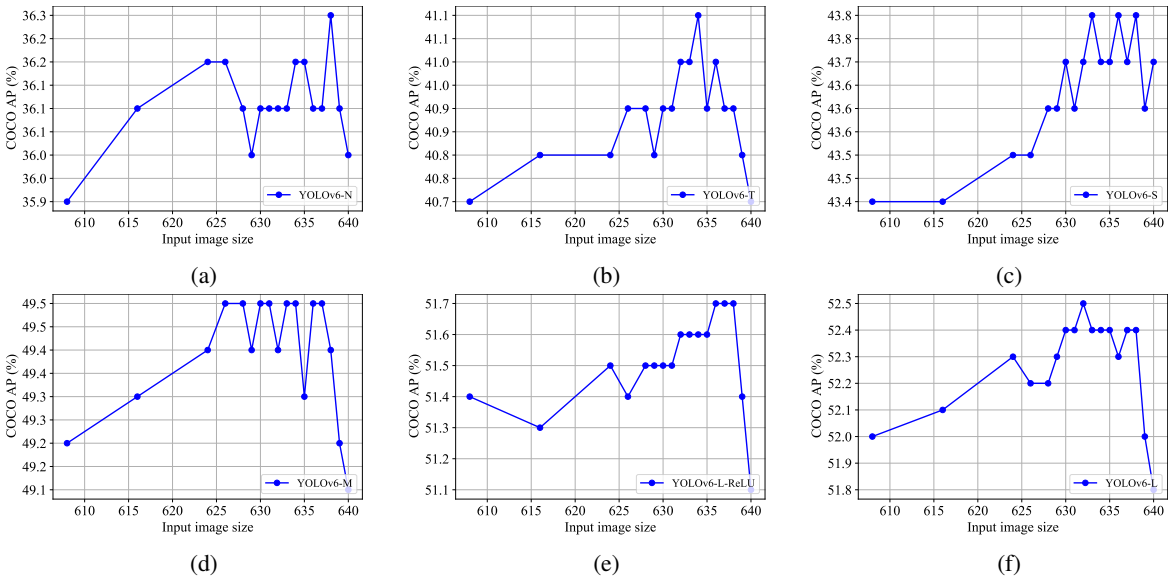


Figure 7: Quantization sensitivity analysis of all layers in YOLOv6s trained with RepOptimizer.



(a) (b) (c)

(d) (e) (f)

Figure 8: Analysis of the gray border problem.

YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

Chien-Yao Wang[1], Alexey Bochkovskiy, and Hong-Yuan Mark Liao[1]
[1]Institute of Information Science, Academia Sinica, Taiwan
kinyiu@iis.sinica.edu.tw, alexeyab84@gmail.com, and liao@iis.sinica.edu.tw

Abstract

YOLOv7 surpasses all known object detectors in both speed and accuracy in the range from 5 FPS to 160 FPS and has the highest accuracy 56.8% AP among all known real-time object detectors with 30 FPS or higher on GPU V100. YOLOv7-E6 object detector (56 FPS V100, 55.9% AP) outperforms both transformer-based detector SWIN-L Cascade-Mask R-CNN (9.2 FPS A100, 53.9% AP) by 509% in speed and 2% in accuracy, and convolutional-based detector ConvNeXt-XL Cascade-Mask R-CNN (8.6 FPS A100, 55.2% AP) by 551% in speed and 0.7% AP in accuracy, as well as YOLOv7 outperforms: YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, Deformable DETR, DINO-5scale-R50, ViT-Adapter-B and many other object detectors in speed and accuracy. Moreover, we train YOLOv7 only on MS COCO dataset from scratch without using any other datasets or pre-trained weights. Source code is released in https://github.com/WongKinYiu/yolov7.

1. Introduction

Real-time object detection is a very important topic in computer vision, as it is often a necessary component in computer vision systems. For example, multi-object tracking [94, 93], autonomous driving [40, 18], robotics [35, 58], medical image analysis [34, 46], etc. The computing devices that execute real-time object detection is usually some mobile CPU or GPU, as well as various neural processing units (NPU) developed by major manufacturers. For example, the Apple neural engine (Apple), the neural compute stick (Intel), Jetson AI edge devices (Nvidia), the edge TPU (Google), the neural processing engine (Qualcomm), the AI processing unit (MediaTek), and the AI SoCs (Kneron), are all NPUs. Some of the above mentioned edge devices focus on speeding up different operations such as vanilla convolution, depth-wise convolution, or MLP operations. In this paper, the real-time object detector we proposed mainly hopes that it can support both mobile GPU and GPU devices from the edge to the cloud.

In recent years, the real-time object detector is still developed for different edge device. For example, the development of MCUNet [49, 48] and NanoDet [54] focused on producing low-power single-chip and improving the inference speed on edge CPU. As for methods such as YOLOX [21] and YOLOR [81], they focus on improving the inference speed of various GPUs. More recently, the development of real-time object detector has focused on the design of efficient architecture. As for real-time object detectors that can be used on CPU [54, 88, 84, 83], their design is mostly based on MobileNet [28, 66, 27], ShuffleNet [92, 55], or GhostNet [25]. Another mainstream real-time object detectors are developed for GPU [81, 21, 97], they mostly use ResNet [26], DarkNet [63], or DLA [87], and then use the CSPNet [80] strategy to optimize the architecture. The development direction of the proposed methods in this paper are different from that of the current mainstream real-time object detectors. In addition to architecture optimization, our proposed methods will focus on the optimization of the training process. Our focus will be on some optimized modules and optimization methods which may strengthen the training cost for improving the accuracy of object detection, but without increasing the inference cost. We call the proposed modules and optimization methods trainable bag-of-freebies.

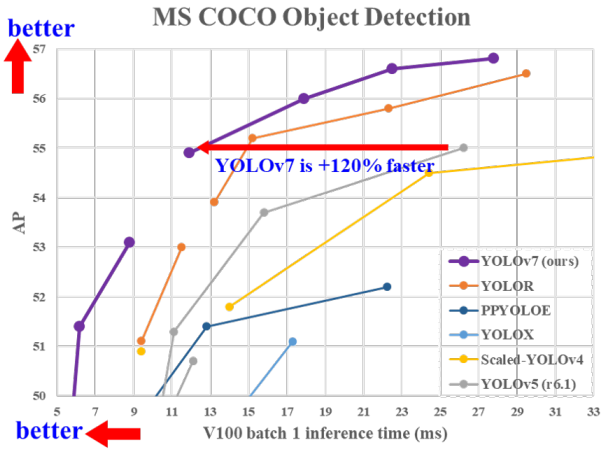


Figure 1: Comparison with other real-time object detectors, our proposed methods achieve state-of-the-arts performance.

Recently, model re-parameterization [13, 12, 29] and dynamic label assignment [20, 17, 42] have become important topics in network training and object detection. Mainly after the above new concepts are proposed, the training of object detector evolves many new issues. In this paper, we will present some of the new issues we have discovered and devise effective methods to address them. For model re-parameterization, we analyze the model re-parameterization strategies applicable to layers in different networks with the concept of gradient propagation path, and propose planned re-parameterized model. In addition, when we discover that with dynamic label assignment technology, the training of model with multiple output layers will generate new issues. That is: "How to assign dynamic targets for the outputs of different branches?" For this problem, we propose a new label assignment method called coarse-to-fine lead guided label assignment.

The contributions of this paper are summarized as follows: (1) we design several trainable bag-of-freebies methods, so that real-time object detection can greatly improve the detection accuracy without increasing the inference cost; (2) for the evolution of object detection methods, we found two new issues, namely how re-parameterized module replaces original module, and how dynamic label assignment strategy deals with assignment to different output layers. In addition, we also propose methods to address the difficulties arising from these issues; (3) we propose "extend" and "compound scaling" methods for the real-time object detector that can effectively utilize parameters and computation; and (4) the method we proposed can effectively reduce about 40% parameters and 50% computation of state-of-the-art real-time object detector, and has faster inference speed and higher detection accuracy.

2. Related work

2.1. Real-time object detectors

Currently state-of-the-art real-time object detectors are mainly based on YOLO [61, 62, 63] and FCOS [76, 77], which are [3, 79, 81, 21, 54, 85, 23]. Being able to become a state-of-the-art real-time object detector usually requires the following characteristics: (1) a faster and stronger network architecture; (2) a more effective feature integration method [22, 97, 37, 74, 59, 30, 9, 45]; (3) a more accurate detection method [76, 77, 69]; (4) a more robust loss function [96, 64, 6, 56, 95, 57]; (5) a more efficient label assignment method [99, 20, 17, 82, 42]; and (6) a more efficient training method. In this paper, we do not intend to explore self-supervised learning or knowledge distillation methods that require additional data or large model. Instead, we will design new trainable bag-of-freebies method for the issues derived from the state-of-the-art methods associated with (4), (5), and (6) mentioned above.

2.2. Model re-parameterization

Model re-parametrization techniques [71, 31, 75, 19, 33, 11, 4, 24, 13, 12, 10, 29, 14, 78] merge multiple computational modules into one at inference stage. The model re-parameterization technique can be regarded as an ensemble technique, and we can divide it into two categories, i.e., module-level ensemble and model-level ensemble. There are two common practices for model-level re-parameterization to obtain the final inference model. One is to train multiple identical models with different training data, and then average the weights of multiple trained models. The other is to perform a weighted average of the weights of models at different iteration number. Module-level re-parameterization is a more popular research issue recently. This type of method splits a module into multiple identical or different module branches during training and integrates multiple branched modules into a completely equivalent module during inference. However, not all proposed re-parameterized module can be perfectly applied to different architectures. With this in mind, we have developed new re-parameterization module and designed related application strategies for various architectures.

2.3. Model scaling

Model scaling [72, 60, 74, 73, 15, 16, 2, 51] is a way to scale up or down an already designed model and make it fit in different computing devices. The model scaling method usually uses different scaling factors, such as resolution (size of input image), depth (number of layer), width (number of channel), and stage (number of feature pyramid), so as to achieve a good trade-off for the amount of network parameters, computation, inference speed, and accuracy. Network architecture search (NAS) is one of the commonly used model scaling methods. NAS can automatically search for suitable scaling factors from search space without defining too complicated rules. The disadvantage of NAS is that it requires very expensive computation to complete the search for model scaling factors. In [15], the researcher analyzes the relationship between scaling factors and the amount of parameters and operations, trying to directly estimate some rules, and thereby obtain the scaling factors required by model scaling. Checking the literature, we found that almost all model scaling methods analyze individual scaling factor independently, and even the methods in the compound scaling category also optimized scaling factor independently. The reason for this is because most popular NAS architectures deal with scaling factors that are not very correlated. We observed that all concatenation-based models, such as DenseNet [32] or VoVNet [39], will change the input width of some layers when the depth of such models is scaled. Since the proposed architecture is concatenation-based, we have to design a new compound scaling method for this model.

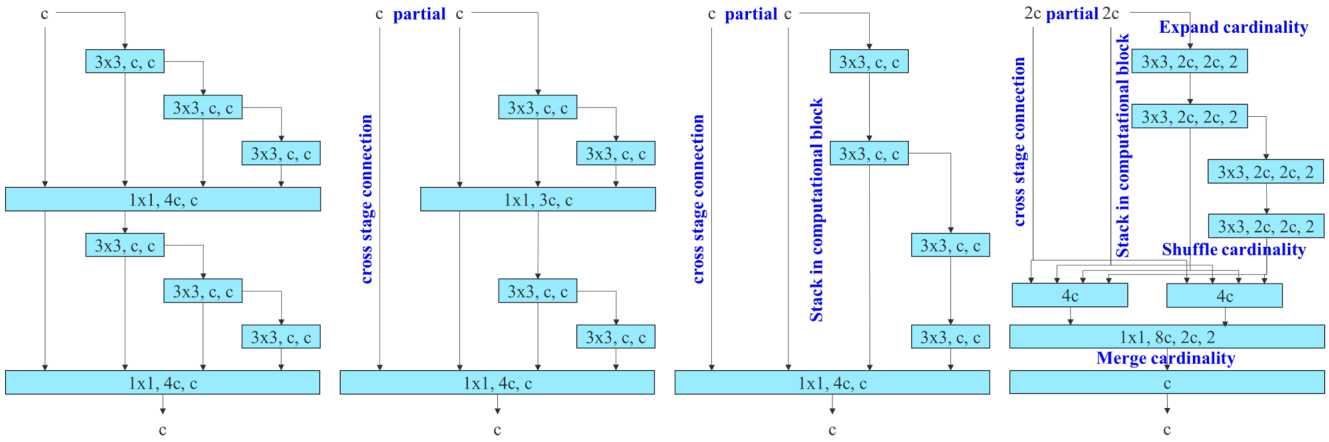


Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmission path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in a shuffle and merge cardinality manner. This way of operation can enhance the features learned by different feature maps and improve the use of parameters and calculations.

3. Architecture

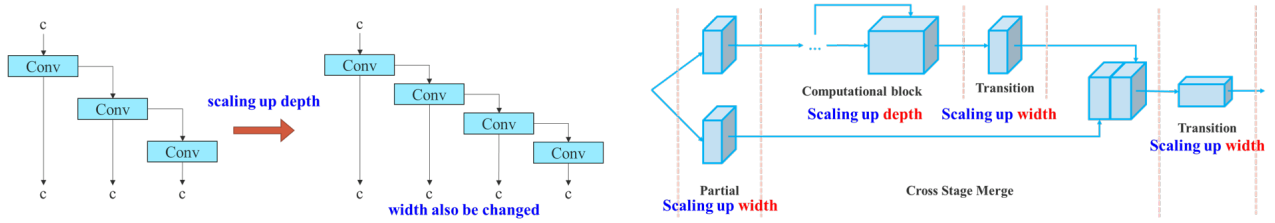
3.1. Extended efficient layer aggregation networks

In most of the literature on designing the efficient architectures, the main considerations are no more than the number of parameters, the amount of computation, and the computational density. Starting from the characteristics of memory access cost, Ma *et al*. [55] also analyzed the influence of the input/output channel ratio, the number of branches of the architecture, and the element-wise operation on the network inference speed. Dollár *et al*. [15] additionally considered activation when performing model scaling, that is, to put more consideration on the number of elements in the output tensors of convolutional layers. The design of CSPVoVNet [79] in Figure 2 (b) is a variation of VoVNet [39]. In addition to considering the aforementioned basic designing concerns, the architecture of CSPVoVNet [79] also analyzes the gradient path, in order to enable the weights of different layers to learn more diverse features. The gradient analysis approach described above makes inferences faster and more accurate. ELAN [1] in Figure 2 (c) considers the following design strategy – "How to design an efficient network?." They came out with a conclusion: By controlling the shortest longest gradient path, a deeper network can learn and converge effectively. In this paper, we propose Extended-ELAN (E-ELAN) based on ELAN and its main architecture is shown in Figure 2 (d).

Regardless of the gradient path length and the stacking number of computational blocks in large-scale ELAN, it has reached a stable state. If more computational blocks are stacked unlimitedly, this stable state may be destroyed, and the parameter utilization rate will decrease. The proposed E-ELAN uses expand, shuffle, merge cardinality to achieve the ability to continuously enhance the learning ability of the network without destroying the original gradient path. In terms of architecture, E-ELAN only changes the architecture in computational block, while the architecture of transition layer is completely unchanged. Our strategy is to use group convolution to expand the channel and cardinality of computational blocks. We will apply the same group parameter and channel multiplier to all the computational blocks of a computational layer. Then, the feature map calculated by each computational block will be shuffled into g groups according to the set group parameter g, and then concatenate them together. At this time, the number of channels in each group of feature map will be the same as the number of channels in the original architecture. Finally, we add g groups of feature maps to perform merge cardinality. In addition to maintaining the original ELAN design architecture, E-ELAN can also guide different groups of computational blocks to learn more diverse features.

3.2. Model scaling for concatenation-based models

The main purpose of model scaling is to adjust some attributes of the model and generate models of different scales to meet the needs of different inference speeds. For example the scaling model of EfficientNet [72] considers the width, depth, and resolution. As for the scaled-YOLOv4 [79], its scaling model is to adjust the number of stages. In [15], Dollár *et al*. analyzed the influence of vanilla convolution and group convolution on the amount of parameter and computation when performing width and depth scaling, and used this to design the corresponding model scaling method.



(a) concatenation-based model (b) scaled-up concatenation-based model (c) compound scaling up depth and width for concatenation-based model

Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on concatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width of the subsequent transmission layer to increase. Therefore, we propose (c), that is, when performing model scaling on concatenation-based models, only the depth in a computational block needs to be scaled, and the remaining of transmission layer is performed with corresponding width scaling.

The above methods are mainly used in architectures such as PlainNet or ResNet. When these architectures are in executing scaling up or scaling down, the in-degree and out-degree of each layer will not change, so we can independently analyze the impact of each scaling factor on the amount of parameters and computation. However, if these methods are applied to the concatenation-based architecture, we will find that when scaling up or scaling down is performed on depth, the in-degree of a translation layer which is immediately after a concatenation-based computational block will decrease or increase, as shown in Figure 3 (a) and (b).

It can be inferred from the above phenomenon that we cannot analyze different scaling factors separately for a concatenation-based model but must be considered together. Take scaling-up depth as an example, such an action will cause a ratio change between the input channel and output channel of a transition layer, which may lead to a decrease in the hardware usage of the model. Therefore, we must propose the corresponding compound model scaling method for a concatenation-based model. When we scale the depth factor of a computational block, we must also calculate the change of the output channel of that block. Then, we will perform width factor scaling with the same amount of change on the transition layers, and the result is shown in Figure 3 (c). Our proposed compound scaling method can maintain the properties that the model had at the initial design and maintains the optimal structure.

4. Trainable bag-of-freebies

4.1. Planned re-parameterized convolution

Although RepConv [13] has achieved excellent performance on the VGG [68], when we directly apply it to ResNet [26] and DenseNet [32] and other architectures, its accuracy will be significantly reduced. We use gradient flow propagation paths to analyze how re-parameterized convolution should be combined with different network. We also designed planned re-parameterized convolution accordingly.

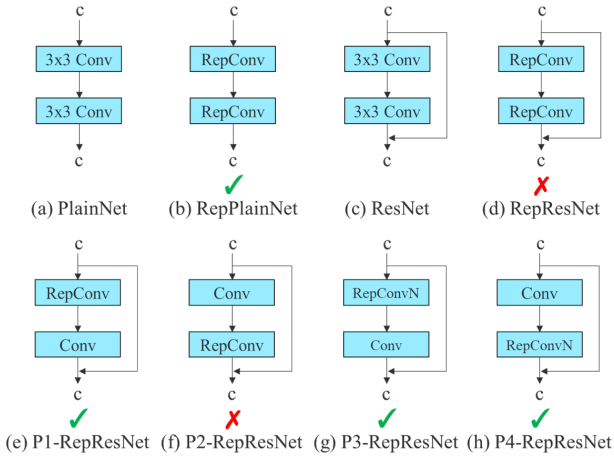
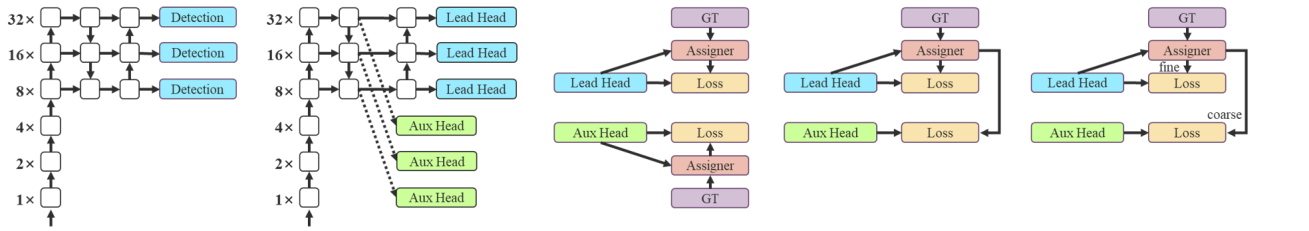


Figure 4: Planned re-parameterized model. In the proposed planned re-parameterized model, we found that a layer with residual or concatenation connections, its RepConv should not have identity connection. Under these circumstances, it can be replaced by RepConvN that contains no identity connections.

RepConv actually combines 3×3 convolution, 1×1 convolution, and identity connection in one convolutional layer. After analyzing the combination and corresponding performance of RepConv and different architectures, we find that the identity connection in RepConv destroys the residual in ResNet and the concatenation in DenseNet, which provides more diversity of gradients for different feature maps. For the above reasons, we use RepConv without identity connection (RepConvN) to design the architecture of planned re-parameterized convolution. In our thinking, when a convolutional layer with residual or concatenation is replaced by re-parameterized convolution, there should be no identity connection. Figure 4 shows an example of our designed "planned re-parameterized convolution" used in PlainNet and ResNet. As for the complete planned re-parameterized convolution experiment in residual-based model and concatenation-based model, it will be presented in the ablation study session.



(a) Normal model (b) Model with auxiliary head (c) Independent assigner (d) Lead guided assigner (e) Coarse-to-fine lead guided assigner

Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be elaborated in Apendix.

4.2. Coarse for auxiliary and fine for lead loss

Deep supervision [38] is a technique that is often used in training deep networks. Its main concept is to add extra auxiliary head in the middle layers of the network, and the shallow network weights with assistant loss as the guide. Even for architectures such as ResNet [26] and DenseNet [32] which usually converge well, deep supervision [70, 98, 67, 47, 82, 65, 86, 50] can still significantly improve the performance of the model on many tasks. Figure 5 (a) and (b) show, respectively, the object detector architecture "without" and "with" deep supervision. In this paper, we call the head responsible for the final output as the lead head, and the head used to assist training is called auxiliary head.

Next we want to discuss the issue of label assignment. In the past, in the training of deep network, label assignment usually refers directly to the ground truth and generate hard label according to the given rules. However, in recent years, if we take object detection as an example, researchers often use the quality and distribution of prediction output by the network, and then consider together with the ground truth to use some calculation and optimization methods to generate a reliable soft label [61, 8, 36, 99, 91, 44, 43, 90, 20, 17, 42]. For example, YOLO [61] use IoU of prediction of bounding box regression and ground truth as the soft label of objectness. In this paper, we call the mechanism that considers the network prediction results together with the ground truth and then assigns soft labels as "label assigner."

Deep supervision needs to be trained on the target objectives regardless of the circumstances of auxiliary head or lead head. During the development of soft label assigner related techniques, we accidentally discovered a new derivative issue, i.e., "How to assign soft label to auxiliary head and lead head ?" To the best of our knowledge, the relevant literature has not explored this issue so far. The results of the most popular method at present is as shown in Figure 5 (c), which is to separate auxiliary head and lead head, and then use their own prediction results and the ground truth to execute label assignment. The method proposed in this paper is a new label assignment method that guides both auxiliary head and lead head by the lead head prediction. In other words, we use lead head prediction as guidance to generate coarse-to-fine hierarchical labels, which are used for auxiliary head and lead head learning, respectively. The two proposed deep supervision label assignment strategies are shown in Figure 5 (d) and (e), respectively.

Lead head guided label assigner is mainly calculated based on the prediction result of the lead head and the ground truth, and generate soft label through the optimization process. This set of soft labels will be used as the target training model for both auxiliary head and lead head. The reason to do this is because lead head has a relatively strong learning capability, so the soft label generated from it should be more representative of the distribution and correlation between the source data and the target. Furthermore, we can view such learning as a kind of generalized residual learning. By letting the shallower auxiliary head directly learn the information that lead head has learned, lead head will be more able to focus on learning residual information that has not yet been learned.

Coarse-to-fine lead head guided label assigner also used the predicted result of the lead head and the ground truth to generate soft label. However, in the process we generate two different sets of soft label, i.e., coarse label and fine label, where fine label is the same as the soft label generated by lead head guided label assigner, and coarse label is generated by allowing more grids to be treated as positive target by relaxing the constraints of the positive sample assignment process. The reason for this is that the learning ability of an auxiliary head is not as strong as that of a lead head, and in order to avoid losing the information that needs to be learned, we will focus on optimizing the recall of auxiliary head in the object detection task. As for the output of lead head, we can filter the high precision results from the high recall results as the final output. However, we must note that if the additional weight of coarse label is close to

Table 1: Comparison of baseline object detectors.

Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLOv4 [3]	64.4M	142.8G	640	49.7%	68.2%	54.3%	32.9%	54.8%	63.7%
YOLOR-u5 (r6.1) [81]	46.5M	109.1G	640	50.2%	68.7%	54.6%	33.2%	55.5%	63.7%
YOLOv4-CSP [79]	52.9M	120.4G	640	50.3%	68.6%	54.9%	34.2%	55.6%	65.1%
YOLOR-CSP [81]	52.9M	120.4G	640	50.8%	69.5%	55.3%	33.7%	56.0%	65.4%
YOLOv7	36.9M	104.7G	640	**51.2%**	**69.7%**	**55.5%**	**35.2%**	**56.0%**	**66.7%**
improvement	-43%	-15%	-	+0.4	+0.2	+0.2	+1.5	=	+1.3
YOLOR-CSP-X [81]	96.9M	226.8G	640	52.7%	**71.3%**	57.4%	36.3%	57.5%	68.3%
YOLOv7-X	71.3M	189.9G	640	**52.9%**	71.1%	**57.5%**	**36.9%**	**57.7%**	**68.6%**
improvement	-36%	-19%	-	+0.2	-0.2	+0.1	+0.6	+0.2	+0.3
YOLOv4-tiny [79]	6.1	6.9	416	24.9%	42.1%	25.7%	8.7%	28.4%	39.2%
YOLOv7-tiny	6.2	5.8	416	**35.2%**	**52.8%**	**37.3%**	**15.7%**	**38.0%**	**53.4%**
improvement	+2%	-19%	-	+10.3	+10.7	+11.6	+7.0	+9.6	+14.2
YOLOv4-tiny-3l [79]	8.7	5.2	320	30.8%	47.3%	32.2%	**10.9%**	31.9%	51.5%
YOLOv7-tiny	6.2	3.5	320	**30.8%**	**47.3%**	**32.2%**	10.0%	**31.9%**	**52.2%**
improvement	-39%	-49%	-	=	=	=	-0.9	=	+0.7
YOLOR-E6 [81]	115.8M	683.2G	1280	55.7%	73.2%	60.7%	40.1%	**60.4%**	69.2%
YOLOv7-E6	97.2M	515.2G	1280	**55.9%**	**73.5%**	**61.1%**	**40.6%**	60.3%	**70.0%**
improvement	-19%	-33%	-	+0.2	+0.3	+0.4	+0.5	-0.1	+0.8
YOLOR-D6 [81]	151.7M	935.6G	1280	56.1%	73.9%	61.2%	**42.4%**	60.5%	69.9%
YOLOv7-D6	154.7M	806.8G	1280	56.3%	73.8%	61.4%	41.3%	60.6%	70.1%
YOLOv7-E6E	151.7M	843.2G	1280	**56.8%**	**74.4%**	**62.1%**	40.8%	**62.1%**	**70.6%**
improvement	=	-11%	-	+0.7	+0.5	+0.9	-1.6	+1.6	+0.7

that of fine label, it may produce bad prior at final prediction. Therefore, in order to make those extra coarse positive grids have less impact, we put restrictions in the decoder, so that the extra coarse positive grids cannot produce soft label perfectly. The mechanism mentioned above allows the importance of fine label and coarse label to be dynamically adjusted during the learning process, and makes the optimizable upper bound of fine label always higher than coarse label.

4.3. Other trainable bag-of-freebies

In this section we will list some trainable bag-of-freebies. These freebies are some of the tricks we used in training, but the original concepts were not proposed by us. The training details of these freebies will be elaborated in the Appendix, including (1) Batch normalization in conv-bn-activation topology: This part mainly connects batch normalization layer directly to convolutional layer. The purpose of this is to integrate the mean and variance of batch normalization into the bias and weight of convolutional layer at the inference stage. (2) Implicit knowledge in YOLOR [81] combined with convolution feature map in addition and multiplication manner: Implicit knowledge in YOLOR can be simplified to a vector by pre-computing at the inference stage. This vector can be combined with the bias and weight of the previous or subsequent convolutional layer. (3) EMA model: EMA is a technique used in mean teacher [75], and in our system we use EMA model purely as the final inference model.

5. Experiments

5.1. Experimental setup

We use Microsoft COCO dataset to conduct experiments and validate our object detection method. All our experiments did not use pre-trained models. That is, all models were trained from scratch. During the development process, we used train 2017 set for training, and then used val 2017 set for verification and choosing hyperparameters. Finally, we show the performance of object detection on the test 2017 set and compare it with the state-of-the-art object detection algorithms. Detailed training parameter settings are described in Appendix.

We designed basic model for edge GPU, normal GPU, and cloud GPU, and they are respectively called YOLOv7-tiny, YOLOv7, and YOLOv7-W6. At the same time, we also use basic model for model scaling for different service requirements and get different types of models. For YOLOv7, we do stack scaling on neck, and use the proposed compound scaling method to perform scaling-up of the depth and width of the entire model, and use this to obtain YOLOv7-X. As for YOLOv7-W6, we use the newly proposed compound scaling method to obtain YOLOv7-E6 and YOLOv7-D6. In addition, we use the proposed E-ELAN for YOLOv7-E6, and thereby complete YOLOv7-E6E. Since YOLOv7-tiny is an edge GPU-oriented architecture, it will use leaky ReLU as activation function. As for other models we use SiLU as activation function. We will describe the scaling factor of each model in detail in Appendix.

Table 2: Comparison of state-of-the-art real-time object detectors.

Model	#Param.	FLOPs	Size	FPS	AP^{test} / AP^{val}	AP^{test}_{50}	AP^{test}_{75}	AP^{test}_{S}	AP^{test}_{M}	AP^{test}_{L}
YOLOX-S [21]	9.0M	26.8G	640	102	40.5% / 40.5%	-	-	-	-	-
YOLOX-M [21]	25.3M	73.8G	640	81	47.2% / 46.9%	-	-	-	-	-
YOLOX-L [21]	54.2M	155.6G	640	69	50.1% / 49.7%	-	-	-	-	-
YOLOX-X [21]	99.1M	281.9G	640	58	51.5% / 51.1%	-	-	-	-	-
PPYOLOE-S [85]	7.9M	17.4G	640	208	43.1% / 42.7%	60.5%	46.6%	23.2%	46.4%	56.9%
PPYOLOE-M [85]	23.4M	49.9G	640	123	48.9% / 48.6%	66.5%	53.0%	28.6%	52.9%	63.8%
PPYOLOE-L [85]	52.2M	110.1G	640	78	51.4% / 50.9%	68.9%	55.6%	31.4%	55.3%	66.1%
PPYOLOE-X [85]	98.4M	206.6G	640	45	52.2% / 51.9%	69.9%	56.5%	33.3%	56.3%	66.4%
YOLOv5-N (r6.1) [23]	1.9M	4.5G	640	159	- / 28.0%	-	-	-	-	-
YOLOv5-S (r6.1) [23]	7.2M	16.5G	640	156	- / 37.4%	-	-	-	-	-
YOLOv5-M (r6.1) [23]	21.2M	49.0G	640	122	- / 45.4%	-	-	-	-	-
YOLOv5-L (r6.1) [23]	46.5M	109.1G	640	99	- / 49.0%	-	-	-	-	-
YOLOv5-X (r6.1) [23]	86.7M	205.7G	640	83	- / 50.7%	-	-	-	-	-
YOLOR-CSP [81]	52.9M	120.4G	640	106	51.1% / 50.8%	69.6%	55.7%	31.7%	55.3%	64.7%
YOLOR-CSP-X [81]	96.9M	226.8G	640	87	53.0% / 52.7%	71.4%	57.9%	33.7%	57.1%	66.8%
YOLOv7-tiny-SiLU	6.2M	13.8G	640	286	38.7% / 38.7%	56.7%	41.7%	18.8%	42.4%	51.9%
YOLOv7	36.9M	104.7G	640	161	51.4% / 51.2%	69.7%	55.9%	31.8%	55.5%	65.0%
YOLOv7-X	71.3M	189.9G	640	114	53.1% / 52.9%	71.2%	57.8%	33.8%	57.1%	67.4%
YOLOv5-N6 (r6.1) [23]	3.2M	18.4G	1280	123	- / 36.0%	-	-	-	-	-
YOLOv5-S6 (r6.1) [23]	12.6M	67.2G	1280	122	- / 44.8%	-	-	-	-	-
YOLOv5-M6 (r6.1) [23]	35.7M	200.0G	1280	90	- / 51.3%	-	-	-	-	-
YOLOv5-L6 (r6.1) [23]	76.8M	445.6G	1280	63	- / 53.7%	-	-	-	-	-
YOLOv5-X6 (r6.1) [23]	140.7M	839.2G	1280	38	- / 55.0%	-	-	-	-	-
YOLOR-P6 [81]	37.2M	325.6G	1280	76	53.9% / 53.5%	71.4%	58.9%	36.1%	57.7%	65.6%
YOLOR-W6 [81]	79.8G	453.2G	1280	66	55.2% / 54.8%	72.7%	60.5%	37.7%	59.1%	67.1%
YOLOR-E6 [81]	115.8M	683.2G	1280	45	55.8% / 55.7%	73.4%	61.1%	38.4%	59.7%	67.7%
YOLOR-D6 [81]	151.7M	935.6G	1280	34	56.5% / 56.1%	74.1%	61.9%	38.9%	60.4%	68.7%
YOLOv7-W6	70.4M	360.0G	1280	84	54.9% / 54.6%	72.6%	60.1%	37.3%	58.7%	67.1%
YOLOv7-E6	97.2M	515.2G	1280	56	56.0% / 55.9%	73.5%	61.2%	38.0%	59.9%	68.4%
YOLOv7-D6	154.7M	806.8G	1280	44	56.6% / 56.3%	74.0%	61.8%	38.8%	60.1%	69.5%
YOLOv7-E6E	151.7M	843.2G	1280	36	56.8% / 56.8%	74.4%	62.1%	39.3%	60.5%	69.0%

[1] Our FLOPs is calaculated by rectangle input resolution like 640 × 640 or 1280 × 1280.
[2] Our inference time is estimated by using letterbox resize input image to make its long side equals to 640 or 1280.

5.2. Baselines

We choose previous version of YOLO [3, 79] and state-of-the-art object detector YOLOR [81] as our baselines. Table 1 shows the comparison of our proposed YOLOv7 models and those baseline that are trained with the same settings.

From the results we see that if compared with YOLOv4, YOLOv7 has 75% less parameters, 36% less computation, and brings 1.5% higher AP. If compared with state-of-the-art YOLOR-CSP, YOLOv7 has 43% fewer parameters, 15% less computation, and 0.4% higher AP. In the performance of tiny model, compared with YOLOv4-tiny-31, YOLOv7-tiny reduces the number of parameters by 39% and the amount of computation by 49%, but maintains the same AP. On the cloud GPU model, our model can still have a higher AP while reducing the number of parameters by 19% and the amount of computation by 33%.

5.3. Comparison with state-of-the-arts

We compare the proposed method with state-of-the-art object detectors for general GPUs and Mobile GPUs, and the results are shown in Table 2. From the results in Table 2 we know that the proposed method has the best speed-accuracy trade-off comprehensively. If we compare YOLOv7-tiny-SiLU with YOLOv5-N (r6.1), our method is 127 fps faster and 10.7% more accurate on AP. In addition, YOLOv7 has 51.4% AP at frame rate of 161 fps, while PPYOLOE-L with the same AP has only 78 fps frame rate. In terms of parameter usage, YOLOv7 is 41% less than PPYOLOE-L. If we compare YOLOv7-X with 114 fps inference speed to YOLOv5-L (r6.1) with 99 fps inference speed, YOLOv7-X can improve AP by 3.9%. If YOLOv7-X is compared with YOLOv5-X (r6.1) of similar scale, the inference speed of YOLOv7-X is 31 fps faster. In addition, in terms of the amount of parameters and computation, YOLOv7-X reduces 22% of parameters and 8% of computation compared to YOLOv5-X (r6.1), but improves AP by 2.2%.

If we compare YOLOv7 with YOLOR using the input resolution 1280, the inference speed of YOLOv7-W6 is 8 fps faster than that of YOLOR-P6, and the detection rate is also increased by 1% AP. As for the comparison between YOLOv7-E6 and YOLOv5-X6 (r6.1), the former has 0.9% AP gain than the latter, 45% less parameters and 63% less computation, and the inference speed is increased by 47%. YOLOv7-D6 has close inference speed to YOLOR-E6, but improves AP by 0.8%. YOLOv7-E6E has close inference speed to YOLOR-D6, but improves AP by 0.3%.

5.4. Ablation study

5.4.1 Proposed compound scaling method

Table 3 shows the results obtained when using different model scaling strategies for scaling up. Among them, our proposed compound scaling method is to scale up the depth of computational block by 1.5 times and the width of transition block by 1.25 times. If our method is compared with the method that only scaled up the width, our method can improve the AP by 0.5% with less parameters and amount of computation. If our method is compared with the method that only scales up the depth, our method only needs to increase the number of parameters by 2.9% and the amount of computation by 1.2%, which can improve the AP by 0.2%. It can be seen from the results of Table 3 that our proposed compound scaling strategy can utilize parameters and computation more efficiently.

Table 3: Ablation study on proposed model scaling.

Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
base (v7-X light)	47.0M	125.5G	640	51.7%	70.1%	56.0%
width only (1.25 w)	73.4M	195.5G	640	52.4%	70.9%	57.1%
depth only (2.0 d)	69.3M	187.6G	640	52.7%	70.8%	57.3%
compound (v7-X)	71.3M	189.9G	640	**52.9%**	**71.1%**	**57.5%**
improvement	-	-	-	+1.2	+1.0	+1.5

5.4.2 Proposed planned re-parameterized model

In order to verify the generality of our proposed planed re-parameterized model, we use it on concatenation-based model and residual-based model respectively for verification. The concatenation-based model and residual-based model we chose for verification are 3-stacked ELAN and CSPDarknet, respectively.

In the experiment of concatenation-based model, we replace the 3×3 convolutional layers in different positions in 3-stacked ELAN with RepConv, and the detailed configuration is shown in Figure 6. From the results shown in Table 4 we see that all higher AP values are present on our proposed planned re-parameterized model.

In the experiment dealing with residual-based model, since the original dark block does not have a 3×3 convolution block that conforms to our design strategy, we additionally design a reversed dark block for the experiment, whose architecture is shown in Figure 7. Since the CSPDarknet with dark block and reversed dark block has exactly the same amount of parameters and operations, it is fair to compare. The experiment results illustrated in Table 5 fully confirm that the proposed planned re-parameterized model is equally effective on residual-based model. We find that the design of RepCSPResNet [85] also fit our design pattern.

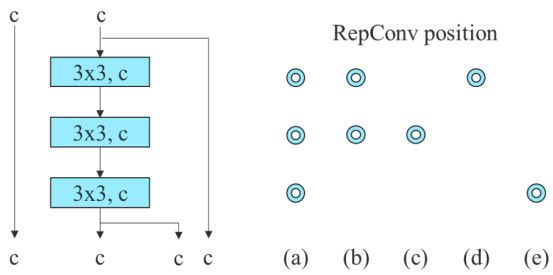


Figure 6: Planned RepConv 3-stacked ELAN. Blue circles are the position we replace Conv by RepConv.

Table 4: Ablation study on planned RepConcatenation model.

Model	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
base (3-S ELAN)	52.26%	70.41%	56.77%	35.81%	57.00%	67.59%
Figure 6 (a)	52.18%	70.34%	56.90%	35.71%	56.83%	67.51%
Figure 6 (b)	52.30%	70.30%	**56.92%**	35.76%	56.95%	67.74%
Figure 6 (c)	**52.33%**	**70.56%**	56.91%	**35.90%**	**57.06%**	67.50%
Figure 6 (d)	52.17%	70.32%	56.82%	35.33%	**57.06%**	**68.09%**
Figure 6 (e)	52.23%	70.20%	56.81%	35.34%	56.97%	66.88%

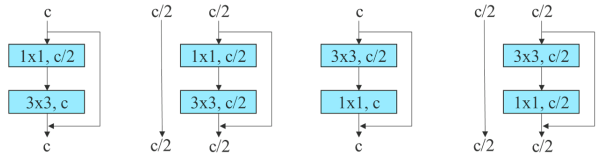


Figure 7: Reversed CSPDarknet. We reverse the position of 1×1 and 3×3 convolutional layer in dark block to fit our planned re-parameterized model design strategy.

Table 5: Ablation study on planned RepResidual model.

Model	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
base (YOLOR-W6)	54.82%	72.39%	59.95%	39.68%	59.38%	**68.30%**
RepCSP	54.67%	72.50%	59.58%	40.22%	**59.61%**	67.87%
RCSP	54.36%	71.95%	59.54%	40.15%	59.02%	67.44%
RepRCSP	**54.85%**	**72.51%**	**60.08%**	**40.53%**	59.52%	68.06%
base (YOLOR-CSP)	50.81%	69.47%	55.28%	33.74%	**56.01%**	65.38%
RepRCSP	**50.91%**	**69.54%**	**55.55%**	**34.44%**	55.74%	**65.46%**

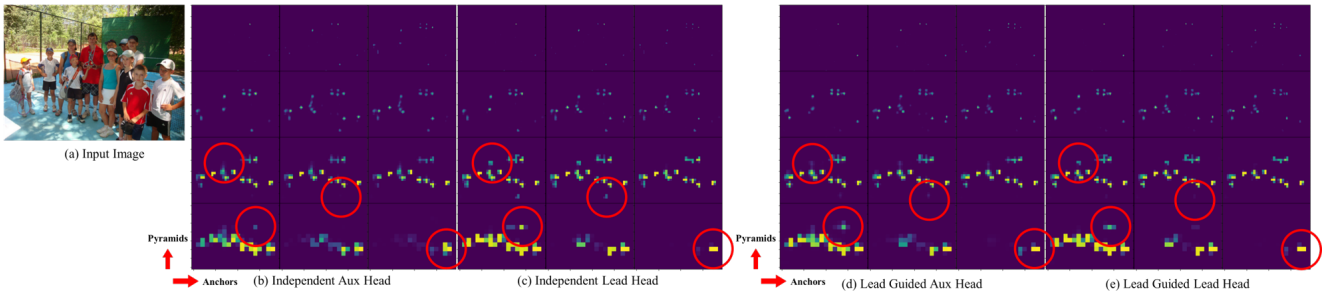


Figure 8: Objectness map predicted by different methods at auxiliary head and lead head.

5.4.3 Proposed assistant loss for auxiliary head

In the assistant loss for auxiliary head experiments, we compare the general independent label assignment for lead head and auxiliary head methods, and we also compare the two proposed lead guided label assignment methods. We show all comparison results in Table 6. From the results listed in Table 6, it is clear that any model that increases assistant loss can significantly improve the overall performance. In addition, our proposed lead guided label assignment strategy receives better performance than the general independent label assignment strategy in AP, AP_{50}, and AP_{75}. As for our proposed coarse for assistant and fine for lead label assignment strategy, it results in best results in all cases. In Figure 8 we show the objectness map predicted by different methods at auxiliary head and lead head. From Figure 8 we find that if auxiliary head learns lead guided soft label, it will indeed help lead head to extract the residual information from the consistant targets.

Table 6: Ablation study on proposed auxiliary head.

Model	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
base (v7-E6)	1280	55.6%	73.2%	60.7%
independent	1280	55.8%	73.4%	60.9%
lead guided	1280	55.9%	73.5%	61.0%
coarse-to-fine lead guided	1280	**55.9%**	**73.5%**	**61.1%**
improvement	-	+0.3	+0.3	+0.4

In Table 7 we further analyze the effect of the proposed coarse-to-fine lead guided label assignment method on the decoder of auxiliary head. That is, we compared the results of with/without the introduction of upper bound constraint. Judging from the numbers in the Table, the method of constraining the upper bound of objectness by the distance from the center of the object can achieve better performance.

Table 7: Ablation study on constrained auxiliary head.

Model	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
base (v7-E6)	1280	55.6%	73.2%	60.7%
aux without constraint	1280	55.9%	73.5%	61.0%
aux with constraint	1280	**55.9%**	**73.5%**	**61.1%**
improvement	-	+0.3	+0.3	+0.4

Since the proposed YOLOv7 uses multiple pyramids to jointly predict object detection results, we can directly connect auxiliary head to the pyramid in the middle layer for training. This type of training can make up for information that may be lost in the next level pyramid prediction. For the above reasons, we designed partial auxiliary head in the proposed E-ELAN architecture. Our approach is to connect auxiliary head after one of the sets of feature map before merging cardinality, and this connection can make the weight of the newly generated set of feature map not directly updated by assistant loss. Our design allows each pyramid of lead head to still get information from objects with different sizes. Table 8 shows the results obtained using two different methods, i.e., coarse-to-fine lead guided and partial coarse-to-fine lead guided methods. Obviously, the partial coarse-to-fine lead guided method has a better auxiliary effect.

Table 8: Ablation study on partial auxiliary head.

Model	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
base (v7-E6E)	1280	56.3%	74.0%	61.5%
aux	1280	56.5%	74.0%	61.6%
partial aux	1280	**56.8%**	**74.4%**	**62.1%**
improvement	-	+0.5	+0.4	+0.6

6. Conclusions

In this paper we propose a new architecture of real-time object detector and the corresponding model scaling method. Furthermore, we find that the evolving process of object detection methods generates new research topics. During the research process, we found the replacement problem of re-parameterized module and the allocation problem of dynamic label assignment. To solve the problem, we propose the trainable bag-of-freebies method to enhance the accuracy of object detection. Based on the above, we have developed the YOLOv7 series of object detection systems, which receives the state-of-the-art results.

7. Acknowledgements

The authors wish to thank National Center for High-performance Computing (NCHC) for providing computational and storage resources.

Table 9: More comparison (batch=1, no-TRT, without extra object detection training data)

Model	#Param.	FLOPs	Size	FPS^{V100}	AP^{test} / AP^{val}	AP$^{test}_{50}$	AP$^{test}_{75}$
YOLOv7-tiny-SiLU	6.2M	13.8G	640	286	**38.7% / 38.7%**	**56.7%**	**41.7%**
PPYOLOE-S [85]	7.9M	17.4G	640	208	**43.1% / 42.7%**	**60.5%**	**46.6%**
YOLOv7	36.9M	104.7G	640	161	**51.4% / 51.2%**	**69.7%**	**55.9%**
YOLOv5-N (r6.1) [23]	1.9M	4.5G	640	159	- / 28.0%	-	-
YOLOv5-S (r6.1) [23]	7.2M	16.5G	640	156	- / 37.4%	-	-
PPYOLOE-M [85]	23.4M	49.9G	640	123	48.9% / 48.6%	66.5%	53.0%
YOLOv5-N6 (r6.1) [23]	3.2M	18.4G	1280	123	- / 36.0%	-	-
YOLOv5-S6 (r6.1) [23]	12.6M	67.2G	1280	122	- / 44.8%	-	-
YOLOv5-M (r6.1) [23]	21.2M	49.0G	640	122	- / 45.4%	-	-
YOLOv7-X	71.3M	189.9G	640	114	**53.1% / 52.9%**	**71.2%**	**57.8%**
YOLOR-CSP [81]	52.9M	120.4G	640	106	51.1% / 50.8%	69.6%	55.7%
YOLOX-S [21]	9.0M	26.8G	640	102	40.5% / 40.5%	-	-
YOLOv5-L (r6.1) [23]	46.5M	109.1G	640	99	- / 49.0%	-	-
YOLOv5-M6 (r6.1) [23]	35.7M	200.0G	1280	90	- / 51.3%	-	-
YOLOR-CSP-X [81]	96.9M	226.8G	640	87	53.0% / 52.7%	**71.4%**	**57.9%**
YOLOv7-W6	70.4M	360.0G	1280	84	**54.9% / 54.6%**	**72.6%**	**60.1%**
YOLOv5-X (r6.1) [23]	86.7M	205.7G	640	83	- / 50.7%	-	-
YOLOX-M [21]	25.3M	73.8G	640	81	47.2% / 46.9%	-	-
PPYOLOE-L [85]	52.2M	110.1G	640	78	51.4% / 50.9%	68.9%	55.6%
YOLOR-P6 [81]	37.2M	325.6G	1280	76	53.9% / 53.5%	71.4%	58.9%
YOLOX-L [21]	54.2M	155.6G	640	69	50.1% / 49.7%	-	-
YOLOR-W6 [81]	79.8G	453.2G	1280	66	**55.2% / 54.8%**	**72.7%**	**60.5%**
YOLOv5-L6 (r6.1) [23]	76.8M	445.6G	1280	63	- / 53.7%	-	-
YOLOX-X [21]	99.1M	281.9G	640	58	51.5% / 51.1%	-	-
YOLOv7-E6	97.2M	515.2G	1280	56	**56.0% / 55.9%**	**73.5%**	**61.2%**
YOLOR-E6 [81]	115.8M	683.2G	1280	45	55.8% / 55.7%	73.4%	61.1%
PPYOLOE-X [85]	98.4M	206.6G	640	45	52.2% / 51.9%	69.9%	56.5%
YOLOv7-D6	154.7M	806.8G	1280	44	**56.6% / 56.3%**	**74.0%**	**61.8%**
YOLOv5-X6 (r6.1) [23]	140.7M	839.2G	1280	38	- / 55.0%	-	-
YOLOv7-E6E	151.7M	843.2G	1280	36	**56.8% / 56.8%**	**74.4%**	**62.1%**
YOLOR-D6 [81]	151.7M	935.6G	1280	34	56.5% / 56.1%	**74.1%**	**61.9%**
F-RCNN-R101-FPN+ [5]	60.0M	246.0G	1333	20	- / 44.0%	-	-
Deformable DETR [100]	40.0M	173.0G	-	19	- / 46.2%	-	-
Swin-B (C-M-RCNN) [52]	145.0M	982.0G	1333	11.6	- / 51.9%	-	-
DETR DC5-R101 [5]	60.0M	253.0G	1333	10	- / 44.9%	-	-
EfficientDet-D7x [74]	77.0M	410.0G	1536	6.5	55.1% / 54.4%	72.4%	58.4%
Dual-Swin-T (C-M-RCNN) [47]	113.8M	836.0G	1333	6.5	- / 53.6%	-	-
ViT-Adapter-B [7]	122.0M	997.0G	-	4.4	- / 50.8%	-	-
Dual-Swin-B (HTC) [47]	235.0M	-	1600	2.5	**58.7% / 58.4%**	-	-
Dual-Swin-L (HTC) [47]	453.0M	-	1600	1.5	**59.4% / 59.1%**	-	-

Model	#Param.	FLOPs	Size	FPS^{A100}	AP^{test} / AP^{val}	AP$^{test}_{50}$	AP$^{test}_{75}$
DN-Deformable-DETR [41]	48.0M	265.0G	1333	23.0	- / 48.6%	-	-
ConvNeXt-B (C-M-RCNN) [53]	-	964.0G	1280	11.5	- / 54.0%	73.1%	58.8%
Swin-B (C-M-RCNN) [52]	-	982.0G	1280	10.7	- / 53.0%	71.8%	57.5%
DINO-5scale (R50) [89]	47.0M	860.0G	1333	10.0	- / 51.0%	-	-
ConvNeXt-L (C-M-RCNN) [53]	-	1354.0G	1280	10.0	- / 54.8%	73.8%	59.8%
Swin-L (C-M-RCNN) [52]	-	1382.0G	1280	9.2	- / 53.9%	72.4%	58.8%
ConvNeXt-XL (C-M-RCNN) [53]	-	1898.0G	1280	8.6	- / 55.2%	74.2%	59.9%

8. More comparison

YOLOv7 surpasses all known object detectors in both speed and accuracy in the range from 5 FPS to 160 FPS and has the highest accuracy 56.8% AP test-dev / 56.8% AP min-val among all known real-time object detectors with 30 FPS or higher on GPU V100. YOLOv7-E6 object detector (56 FPS V100, 55.9% AP) outperforms both transformer-based detector SWIN-L Cascade-Mask R-CNN (9.2 FPS A100, 53.9% AP) by 509% in speed and 2% in accuracy, and convolutional-based detector ConvNeXt-XL Cascade-Mask R-CNN (8.6 FPS A100, 55.2% AP) by 551% in speed and 0.7% AP in accuracy, as well as YOLOv7 outperforms: YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, Deformable DETR, DINO-5scale-R50, ViT-Adapter-B and many other object detectors in speed and accuracy. More over, we train YOLOv7 only on MS COCO dataset from scratch without using any other datasets or pre-trained weights.

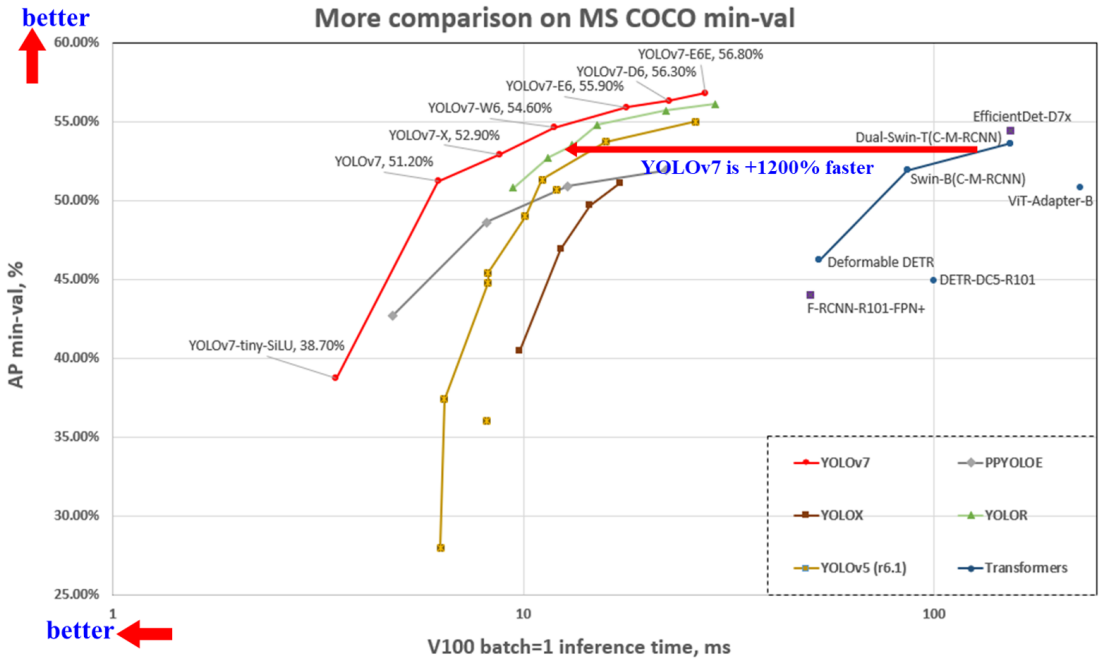


Figure 9: Comparison with other object detectors.

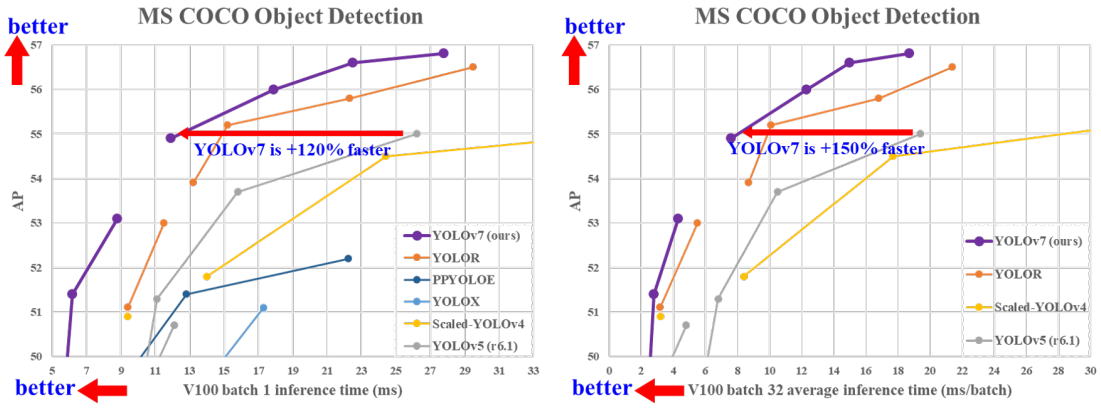


Figure 10: Comparison with other real-time object detectors.

Table 10: Comparison of different setting.

Model	Presicion	IoU threshold	AP^{val}
YOLOv7-X	FP16 (default)	0.65 (default)	**52.9%**
YOLOv7-X	FP32	0.65	**53.0%**
YOLOv7-X	FP16	0.70	**53.0%**
YOLOv7-X	FP32	0.70	**53.1%**
improvement	-	-	+0.2%

* Similar to meituan/YOLOv6 and PPYOLOE, our model could get higher AP when set higher IoU threshold.

The maximum accuracy of the YOLOv7-E6E (56.8% AP) real-time model is +13.7% AP higher than the current most accurate meituan/YOLOv6-s model (43.1% AP) on COCO dataset. Our YOLOv7-tiny (35.2% AP, 0.4 ms) model is +25% faster and +0.2% AP higher than meituan/YOLOv6-n (35.0% AP, 0.5 ms) under identical conditions on COCO dataset and V100 GPU with batch=32.

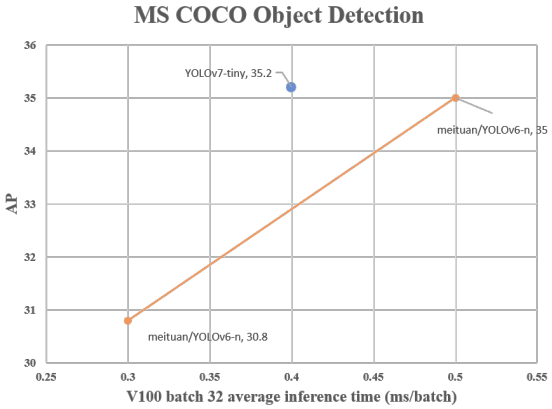


Figure 11: Comparison with other real-time object detectors.

References

[1] anonymous. Designing network design strategies. *anonymous submission*, 2022. 3

[2] Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting ResNets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021. 2

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 2, 6, 7

[4] Yue Cao, Thomas Andrew Geddes, Jean Yee Hwa Yang, and Pengyi Yang. Ensemble deep learning in bioinformatics. *Nature Machine Intelligence*, 2(9):500–508, 2020. 2

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 213–229, 2020. 10

[6] Kean Chen, Weiyao Lin, Jianguo Li, John See, Ji Wang, and Junni Zou. AP-loss for accurate one-stage object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 43(11):3782–3798, 2020. 2

[7] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions. *arXiv preprint arXiv:2205.08534*, 2022. 10

[8] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 502–511, 2019. 5

[9] Xiyang Dai, Yinpeng Chen, Bin Xiao, Dongdong Chen, Mengchen Liu, Lu Yuan, and Lei Zhang. Dynamic head: Unifying object detection heads with attentions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7373–7382, 2021. 2

[10] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Kaiqi Huang, Jungong Han, and Guiguang Ding. Re-parameterizing your optimizers rather than architectures. *arXiv preprint arXiv:2205.15242*, 2022. 2

[11] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. ACNet: Strengthening the kernel skeletons for powerful CNN via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1911–1920, 2019. 2

[12] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10886–10895, 2021. 2

[13] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. RepVGG: Making VGG-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13733–13742, 2021. 2, 4

[14] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[15] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 924–932, 2021. 2, 3

[16] Xianzhi Du, Barret Zoph, Wei-Chih Hung, and Tsung-Yi Lin. Simple training strategies and model scaling for object detection. *arXiv preprint arXiv:2107.00057*, 2021. 2

[17] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R Scott, and Weilin Huang. TOOD: Task-aligned one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3490–3499, 2021. 2, 5

[18] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaeser, Fabian Timm, Werner Wiesbeck, and Klaus Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1341–1360, 2020. 1

[19] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018. 2

[20] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and Jian Sun. OTA: Optimal transport assignment for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 303–312, 2021. 2, 5

[21] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: Exceeding YOLO series in 2021. *arXiv preprint arXiv:2107.08430*, 2021. 1, 2, 7, 10

[22] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7036–7045, 2019. 2

[23] Jocher Glenn. YOLOv5 release v6.1. https://github.com/ultralytics/yolov5/releases/tag/v6.1, 2022. 2, 7, 10

[24] Shuxuan Guo, Jose M Alvarez, and Mathieu Salzmann. ExpandNets: Linear over-parameterization to train compact convolutional networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1298–1310, 2020. 2

[25] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. GhostNet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1580–1589, 2020. 1

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceed-*

ings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016. 1, 4, 5

[27] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1314–1324, 2019. 1

[28] 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. *arXiv preprint arXiv:1704.04861*, 2017. 1

[29] Mu Hu, Junyi Feng, Jiashen Hua, Baisheng Lai, Jianqiang Huang, Xiaojin Gong, and Xiansheng Hua. Online convolutional re-parameterization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[30] Miao Hu, Yali Li, Lu Fang, and Shengjin Wang. A^2-FPN: Attention aggregation based feature pyramid network for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15343–15352, 2021. 2

[31] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *International Conference on Learning Representations (ICLR)*, 2017. 2

[32] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017. 2, 4, 5

[33] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018. 2

[34] Paul F Jaeger, Simon AA Kohl, Sebastian Bickelhaupt, Fabian Isensee, Tristan Anselm Kuder, Heinz-Peter Schlemmer, and Klaus H Maier-Hein. Retina U-Net: Embarrassingly simple exploitation of segmentation supervision for medical object detection. In *Machine Learning for Health Workshop*, pages 171–183, 2020. 1

[35] Hakan Karaoguz and Patric Jensfelt. Object detection approach for robot grasp detection. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4953–4959, 2019. 1

[36] Kang Kim and Hee Seok Lee. Probabilistic anchor assignment with iou prediction for object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 355–371, 2020. 5

[37] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6399–6408, 2019. 2

[38] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial Intelligence and Statistics*, pages 562–570, 2015. 5

[39] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 0–0, 2019. 2, 3

[40] Buyu Li, Wanli Ouyang, Lu Sheng, Xingyu Zeng, and Xiaogang Wang. GS3D: An efficient 3d object detection framework for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1019–1028, 2019. 1

[41] Feng Li, Hao Zhang, Shilong Liu, Jian Guo, Lionel M Ni, and Lei Zhang. DN-DETR: Accelerate detr training by introducing query denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13619–13627, 2022. 10

[42] Shuai Li, Chenhang He, Ruihuang Li, and Lei Zhang. A dual weighting label assignment scheme for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9387–9396, 2022. 2, 5

[43] Xiang Li, Wenhai Wang, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss v2: Learning reliable localization quality estimation for dense object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11632–11641, 2021. 5

[44] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:21002–21012, 2020. 5

[45] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring plain vision transformer backbones for object detection. *arXiv preprint arXiv:2203.16527*, 2022. 2

[46] Zhuoling Li, Minghui Dong, Shiping Wen, Xiang Hu, Pan Zhou, and Zhigang Zeng. CLU-CNNs: Object detection for medical images. *Neurocomputing*, 350:53–59, 2019. 1

[47] Tingting Liang, Xiaojie Chu, Yudong Liu, Yongtao Wang, Zhi Tang, Wei Chu, Jingdong Chen, and Haibin Ling. CBNetV2: A composite backbone network architecture for object detection. *arXiv preprint arXiv:2107.00420*, 2021. 5, 10

[48] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Memory-efficient patch-based inference for tiny deep learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:2346–2358, 2021. 1

[49] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. MCUNet: Tiny deep learning on IoT devices. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:11711–11722, 2020. 1

[50] Yuxuan Liu, Lujia Wang, and Ming Liu. YOLOStereo3D: A step back to 2D for efficient stereo 3D detection. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 13018–13024, 2021. 5

[51] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong,

et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[52] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022, 2021. 10

[53] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11976–11986, 2022. 10

[54] Rangi Lyu. NanoDet-Plus. https://github.com/RangiLyu/nanodet/releases/tag/v1.0.0-alpha-1, 2021. 1, 2

[55] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 1, 3

[56] Kemal Oksuz, Baris Can Cam, Emre Akbas, and Sinan Kalkan. A ranking-based, balanced loss function unifying classification and localisation in object detection. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:15534–15545, 2020. 2

[57] Kemal Oksuz, Baris Can Cam, Emre Akbas, and Sinan Kalkan. Rank & sort loss for object detection and instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3009–3018, 2021. 2

[58] Shuvo Kumar Paul, Muhammed Tawfiq Chowdhury, Mircea Nicolescu, Monica Nicolescu, and David Feil-Seifer. Object detection and pose estimation from rgb and depth data for real-time, adaptive robotic grasping. In *Advances in Computer Vision and Computational Biology*, pages 121–142. 2021. 1

[59] Siyuan Qiao, Liang-Chieh Chen, and Alan Yuille. DetectoRS: Detecting objects with recursive feature pyramid and switchable atrous convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10213–10224, 2021. 2

[60] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10428–10436, 2020. 2

[61] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. 2, 5

[62] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017. 2

[63] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 1, 2

[64] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, 2019. 2

[65] Byungseok Roh, JaeWoong Shin, Wuhyun Shin, and Saehoon Kim. Sparse DETR: Efficient end-to-end object detection with learnable sparsity. *arXiv preprint arXiv:2111.14330*, 2021. 5

[66] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018. 1

[67] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Object detection from scratch with deep supervision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(2):398–412, 2019. 5

[68] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4

[69] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, et al. Sparse R-CNN: End-to-end object detection with learnable proposals. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14454–14463, 2021. 2

[70] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 5

[71] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. 2

[72] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, pages 6105–6114, 2019. 2, 3

[73] Mingxing Tan and Quoc Le. EfficientNetv2: Smaller models and faster training. In *International Conference on Machine Learning (ICML)*, pages 10096–10106, 2021. 2

[74] Mingxing Tan, Ruoming Pang, and Quoc V Le. EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10781–10790, 2020. 2, 10

[75] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017. 2, 6

[76] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proceed-*

ings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 9627–9636, 2019. 2

[77] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: A simple and strong anchor-free object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(4):1922–1933, 2022. 2

[78] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. An improved one millisecond mobile backbone. *arXiv preprint arXiv:2206.04040*, 2022. 2

[79] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-YOLOv4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13029–13038, 2021. 2, 3, 6, 7

[80] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A new backbone that can enhance learning capability of CNN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 390–391, 2020. 1

[81] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. You only learn one representation: Unified network for multiple tasks. *arXiv preprint arXiv:2105.04206*, 2021. 1, 2, 6, 7, 10

[82] Jianfeng Wang, Lin Song, Zeming Li, Hongbin Sun, Jian Sun, and Nanning Zheng. End-to-end object detection with fully convolutional network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15849–15858, 2021. 2, 5

[83] Bichen Wu, Chaojian Li, Hang Zhang, Xiaoliang Dai, Peizhao Zhang, Matthew Yu, Jialiang Wang, Yingyan Lin, and Peter Vajda. FBNetv5: Neural architecture search for multiple tasks in one run. *arXiv preprint arXiv:2111.10007*, 2021. 1

[84] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin, Gabriel Bender, Yongzhe Wang, Pieter-Jan Kindermans, Mingxing Tan, Vikas Singh, and Bo Chen. MobileDets: Searching for object detection architectures for mobile accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3825–3834, 2021. 1

[85] Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao Chang, Cheng Cui, Kaipeng Deng, Guanzhong Wang, Qingqing Dang, Shengyu Wei, Yuning Du, et al. PP-YOLOE: An evolved version of YOLO. *arXiv preprint arXiv:2203.16250*, 2022. 2, 7, 8, 10

[86] Zetong Yang, Yin Zhou, Zhifeng Chen, and Jiquan Ngiam. 3D-MAN: 3D multi-frame attention network for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1863–1872, 2021. 5

[87] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2403–2412, 2018. 1

[88] Guanghua Yu, Qinyao Chang, Wenyu Lv, Chang Xu, Cheng Cui, Wei Ji, Qingqing Dang, Kaipeng Deng, Guanzhong Wang, Yuning Du, et al. PP-PicoDet: A better real-time object detector on mobile devices. *arXiv preprint arXiv:2111.00902*, 2021. 1

[89] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. DINO: DETR with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*, 2022. 10

[90] Haoyang Zhang, Ying Wang, Feras Dayoub, and Niko Sunderhauf. VarifocalNet: An IoU-aware dense object detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8514–8523, 2021. 5

[91] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9759–9768, 2020. 5

[92] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018. 1

[93] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. BYTETrack: Multi-object tracking by associating every detection box. *arXiv preprint arXiv:2110.06864*, 2021. 1

[94] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. FAIRMOT: On the fairness of detection and re-identification in multiple object tracking. *International Journal of Computer Vision*, 129(11):3069–3087, 2021. 1

[95] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-IoU loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 12993–13000, 2020. 2

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

[97] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 1, 2

[98] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. UNet++: A nested U-Net architecture for medical image segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, 2018. 5

[99] Benjin Zhu, Jianfeng Wang, Zhengkai Jiang, Fuhang Zong, Songtao Liu, Zeming Li, and Jian Sun. AutoAssign: Differentiable label assignment for dense object detection. *arXiv preprint arXiv:2007.03496*, 2020. 2, 5

[100] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable transformers for end-to-end object detection. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. 10

YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information

Chien-Yao Wang[1,2], I-Hau Yeh[2], and Hong-Yuan Mark Liao[1,2,3]
[1]Institute of Information Science, Academia Sinica, Taiwan
[2]National Taipei University of Technology, Taiwan
[3]Department of Information and Computer Engineering, Chung Yuan Christian University, Taiwan
kinyiu@iis.sinica.edu.tw, ihyeh@emc.com.tw, and liao@iis.sinica.edu.tw

Abstract

Today's deep learning methods focus on how to design the most appropriate objective functions so that the prediction results of the model can be closest to the ground truth. Meanwhile, an appropriate architecture that can facilitate acquisition of enough information for prediction has to be designed. Existing methods ignore a fact that when input data undergoes layer-by-layer feature extraction and spatial transformation, large amount of information will be lost. This paper will delve into the important issues of data loss when data is transmitted through deep networks, namely information bottleneck and reversible functions. We proposed the concept of programmable gradient information (PGI) to cope with the various changes required by deep networks to achieve multiple objectives. PGI can provide complete input information for the target task to calculate objective function, so that reliable gradient information can be obtained to update network weights. In addition, a new lightweight network architecture – Generalized Efficient Layer Aggregation Network (GELAN), based on gradient path planning is designed. GELAN's architecture confirms that PGI has gained superior results on lightweight models. We verified the proposed GELAN and PGI on MS COCO dataset based object detection. The results show that GELAN only uses conventional convolution operators to achieve better parameter utilization than the state-of-the-art methods developed based on depth-wise convolution. PGI can be used for variety of models from lightweight to large. It can be used to obtain complete information, so that train-from-scratch models can achieve better results than state-of-the-art models pre-trained using large datasets, the comparison results are shown in Figure 1. The source codes are at: https://github.com/WongKinYiu/yolov9.

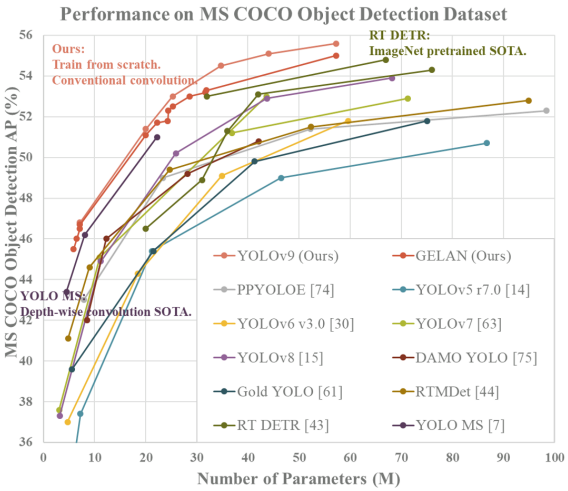
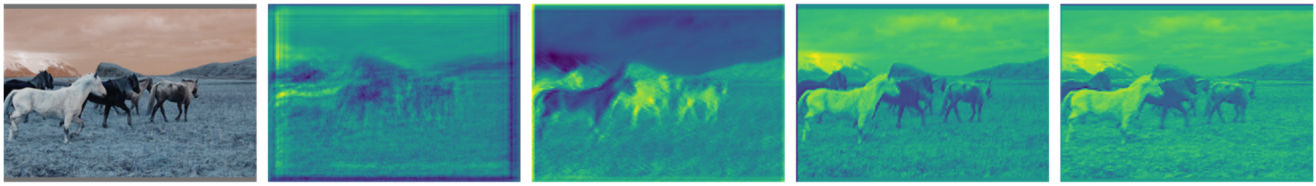


Figure 1. Comparisons of the real-time object detecors on MS COCO dataset. The GELAN and PGI-based object detection method surpassed all previous train-from-scratch methods in terms of object detection performance. In terms of accuracy, the new method outperforms RT DETR [43] pre-trained with a large dataset, and it also outperforms depth-wise convolution-based design YOLO MS [7] in terms of parameters utilization.

1. Introduction

Deep learning-based models have demonstrated far better performance than past artificial intelligence systems in various fields, such as computer vision, language processing, and speech recognition. In recent years, researchers in the field of deep learning have mainly focused on how to develop more powerful system architectures and learning methods, such as CNNs [21–23, 42, 55, 71, 72], Transformers [8, 9, 40, 41, 60, 69, 70], Perceivers [26, 26, 32, 52, 56, 81, 81], and Mambas [17, 38, 80]. In addition, some researchers have tried to develop more general objective functions, such as loss function [5, 45, 46, 50, 77, 78], label assignment [10, 12, 33, 67, 79] and auxiliary supervision [18, 20, 24, 28, 29, 51, 54, 68, 76]. The above studies all try to precisely find the mapping between input and target tasks. However, most past approaches have ignored that input data may have a non-negligible amount of information loss during the feedforward process. This loss of information can lead to biased gradient flows, which are subsequently used to update the model. The above problems can result in deep networks to establish incorrect associations between targets and inputs, causing the trained model to produce incorrect predictions.



(a) Input Image **(b) PlainNet** **(c) ResNet** **(d) CSPNet** **(e) GELAN**

Figure 2. Visualization results of random initial weight output feature maps for different network architectures: (a) input image, (b) PlainNet, (c) ResNet, (d) CSPNet, and (e) proposed GELAN. From the figure, we can see that in different architectures, the information provided to the objective function to calculate the loss is lost to varying degrees, and our architecture can retain the most complete information and provide the most reliable gradient information for calculating the objective function.

In deep networks, the phenomenon of input data losing information during the feedforward process is commonly known as information bottleneck [59], and its schematic diagram is as shown in Figure 2. At present, the main methods that can alleviate this phenomenon are as follows: (1) The use of reversible architectures [3, 16, 19]: this method mainly uses repeated input data and maintains the information of the input data in an explicit way; (2) The use of masked modeling [1, 6, 9, 27, 71, 73]: it mainly uses reconstruction loss and adopts an implicit way to maximize the extracted features and retain the input information; and (3) Introduction of the deep supervision concept [28,51,54,68]: it uses shallow features that have not lost too much important information to pre-establish a mapping from features to targets to ensure that important information can be transferred to deeper layers. However, the above methods have different drawbacks in the training process and inference process. For example, a reversible architecture requires additional layers to combine repeatedly fed input data, which will significantly increase the inference cost. In addition, since the input data layer to the output layer cannot have a too deep path, this limitation will make it difficult to model high-order semantic information during the training process. As for masked modeling, its reconstruction loss sometimes conflicts with the target loss. In addition, most mask mechanisms also produce incorrect associations with data. For the deep supervision mechanism, it will produce error accumulation, and if the shallow supervision loses information during the training process, the subsequent layers will not be able to retrieve the required information. The above phenomenon will be more significant on difficult tasks and small models.

To address the above-mentioned issues, we propose a new concept, which is programmable gradient information (PGI). The concept is to generate reliable gradients through auxiliary reversible branch, so that the deep features can still maintain key characteristics for executing target task. The design of auxiliary reversible branch can avoid the semantic loss that may be caused by a traditional deep supervision process that integrates multi-path features. In other words, we are programming gradient information propagation at different semantic levels, and thereby achieving the best training results. The reversible architecture of PGI is built on auxiliary branch, so there is no additional cost. Since PGI can freely select loss function suitable for the target task, it also overcomes the problems encountered by mask modeling. The proposed PGI mechanism can be applied to deep neural networks of various sizes and is more general than the deep supervision mechanism, which is only suitable for very deep neural networks.

In this paper, we also designed generalized ELAN (GELAN) based on ELAN [65], the design of GELAN simultaneously takes into account the number of parameters, computational complexity, accuracy and inference speed. This design allows users to arbitrarily choose appropriate computational blocks for different inference devices. We combined the proposed PGI and GELAN, and then designed a new generation of YOLO series object detection system, which we call YOLOv9. We used the MS COCO dataset to conduct experiments, and the experimental results verified that our proposed YOLOv9 achieved the top performance in all comparisons.

We summarize the contributions of this paper as follows:

1. We theoretically analyzed the existing deep neural network architecture from the perspective of reversible function, and through this process we successfully explained many phenomena that were difficult to explain in the past. We also designed PGI and auxiliary reversible branch based on this analysis and achieved excellent results.
2. The PGI we designed solves the problem that deep supervision can only be used for extremely deep neural network architectures, and therefore allows new lightweight architectures to be truly applied in daily life.
3. The GELAN we designed only uses conventional convolution to achieve a higher parameter usage than the depth-wise convolution design that based on the most advanced technology, while showing great advantages of being light, fast, and accurate.
4. Combining the proposed PGI and GELAN, the object detection performance of the YOLOv9 on MS COCO dataset greatly surpasses the existing real-time object detectors in all aspects.

2. Related work

2.1. Real-time Object Detectors

The current mainstream real-time object detectors are the YOLO series [2, 7, 13–15, 25, 30, 31, 47–49, 61–63, 74, 75], and most of these models use CSPNet [64] or ELAN [65] and their variants as the main computing units. In terms of feature integration, improved PAN [37] or FPN [35] is often used as a tool, and then improved YOLOv3 head [49] or FCOS head [57, 58] is used as prediction head. Recently some real-time object detectors, such as RT DETR [43], which puts its fundation on DETR [4], have also been proposed. However, since it is extremely difficult for DETR series object detector to be applied to new domains without a corresponding domain pre-trained model, the most widely used real-time object detector at present is still YOLO series. This paper chooses YOLOv7 [63], which has been proven effective in a variety of computer vision tasks and various scenarios, as a base to develop the proposed method. We use GELAN to improve the architecture and the training process with the proposed PGI. The above novel approach makes the proposed YOLOv9 the top real-time object detector of the new generation.

2.2. Reversible Architectures

The operation unit of reversible architectures [3, 16, 19] must maintain the characteristics of reversible conversion, so it can be ensured that the output feature map of each layer of operation unit can retain complete original information. Before, RevCol [3] generalizes traditional reversible unit to multiple levels, and in doing so can expand the semantic levels expressed by different layer units. Through a literature review of various neural network architectures, we found that there are many high-performing architectures with varying degree of reversible properties. For example, Res2Net module [11] combines different input partitions with the next partition in a hierarchical manner, and concatenates all converted partitions before passing them backwards. CBNet [34, 39] re-introduces the original input data through composite backbone to obtain complete original information, and obtains different levels of multi-level reversible information through various composition methods. These network architectures generally have excellent parameter utilization, but the extra composite layers cause slow inference speeds. DynamicDet [36] combines CBNet [34] and the high-efficiency real-time object detector YOLOv7 [63] to achieve a very good trade-off among speed, number of parameters, and accuracy. This paper introduces the DynamicDet architecture as the basis for designing reversible branches. In addition, reversible information is further introduced into the proposed PGI. The proposed new architecture does not require additional connections during the inference process, so it can fully retain the advantages of speed, parameter amount, and accuracy.

2.3. Auxiliary Supervision

Deep supervision [28, 54, 68] is the most common auxiliary supervision method, which performs training by inserting additional prediction layers in the middle layers. Especially the application of multi-layer decoders introduced in the transformer-based methods is the most common one. Another common auxiliary supervision method is to utilize the relevant meta information to guide the feature maps produced by the intermediate layers and make them have the properties required by the target tasks [18, 20, 24, 29, 76]. Examples of this type include using segmentation loss or depth loss to enhance the accuracy of object detectors. Recently, there are many reports in the literature [53, 67, 82] that use different label assignment methods to generate different auxiliary supervision mechanisms to speed up the convergence speed of the model and improve the robustness at the same time. However, the auxiliary supervision mechanism is usually only applicable to large models, so when it is applied to lightweight models, it is easy to cause an under parameterization phenomenon, which makes the performance worse. The PGI we proposed designed a way to reprogram multi-level semantic information, and this design allows lightweight models to also benefit from the auxiliary supervision mechanism.

3. Problem Statement

Usually, people attribute the difficulty of deep neural network convergence problem due to factors such as gradient vanish or gradient saturation, and these phenomena do exist in traditional deep neural networks. However, modern deep neural networks have already fundamentally solved the above problem by designing various normalization and activation functions. Nevertheless, deep neural networks still have the problem of slow convergence or poor convergence results.

In this paper, we explore the nature of the above issue further. Through in-depth analysis of information bottleneck, we deduced that the root cause of this problem is that the initial gradient originally coming from a very deep network has lost a lot of information needed to achieve the goal soon after it is transmitted. In order to confirm this inference, we feedforward deep networks of different architectures with initial weights, and then visualize and illustrate them in Figure 2. Obviously, PlainNet has lost a lot of important information required for object detection in deep layers. As for the proportion of important information that ResNet, CSPNet, and GELAN can retain, it is indeed positively related to the accuracy that can be obtained after training. We further design reversible network-based methods to solve the causes of the above problems. In this section we shall elaborate our analysis of information bottleneck principle and reversible functions.

3.1. Information Bottleneck Principle

According to information bottleneck principle, we know that data X may cause information loss when going through transformation, as shown in Eq. 1 below:

$$I(X, X) \geq I(X, f_\theta(X)) \geq I(X, g_\phi(f_\theta(X))), \quad (1)$$

where I indicates mutual information, f and g are transformation functions, and θ and ϕ are parameters of f and g, respectively.

In deep neural networks, $f_\theta(\cdot)$ and $g_\phi(\cdot)$ respectively represent the operations of two consecutive layers in deep neural network. From Eq. 1, we can predict that as the number of network layer becomes deeper, the original data will be more likely to be lost. However, the parameters of the deep neural network are based on the output of the network as well as the given target, and then update the network after generating new gradients by calculating the loss function. As one can imagine, the output of a deeper neural network is less able to retain complete information about the prediction target. This will make it possible to use incomplete information during network training, resulting in unreliable gradients and poor convergence.

One way to solve the above problem is to directly increase the size of the model. When we use a large number of parameters to construct a model, it is more capable of performing a more complete transformation of the data. The above approach allows even if information is lost during the data feedforward process, there is still a chance to retain enough information to perform the mapping to the target. The above phenomenon explains why the width is more important than the depth in most modern models. However, the above conclusion cannot fundamentally solve the problem of unreliable gradients in very deep neural network. Below, we will introduce how to use reversible functions to solve problems and conduct relative analysis.

3.2. Reversible Functions

When a function r has an inverse transformation function v, we call this function reversible function, as shown in Eq. 2.

$$X = v_\zeta(r_\psi(X)), \quad (2)$$

where ψ and ζ are parameters of r and v, respectively. Data X is converted by reversible function without losing information, as shown in Eq. 3.

$$I(X, X) = I(X, r_\psi(X)) = I(X, v_\zeta(r_\psi(X))). \quad (3)$$

When the network's transformation function is composed of reversible functions, more reliable gradients can be obtained to update the model. Almost all of today's popular deep learning methods are architectures that conform to the reversible property, such as Eq. 4.

$$X^{l+1} = X^l + f_\theta^{l+1}(X^l), \quad (4)$$

where l indicates the l-th layer of a PreAct ResNet and f is the transformation function of the l-th layer. PreAct ResNet [22] repeatedly passes the original data X to subsequent layers in an explicit way. Although such a design can make a deep neural network with more than a thousand layers converge very well, it destroys an important reason why we need deep neural networks. That is, for difficult problems, it is difficult for us to directly find simple mapping functions to map data to targets. This also explains why PreAct ResNet performs worse than ResNet [21] when the number of layers is small.

In addition, we tried to use masked modeling that allowed the transformer model to achieve significant breakthroughs. We use approximation methods, such as Eq. 5, to try to find the inverse transformation v of r, so that the transformed features can retain enough information using sparse features. The form of Eq. 5 is as follows:

$$X = v_\zeta(r_\psi(X) \cdot M), \quad (5)$$

where M is a dynamic binary mask. Other methods that are commonly used to perform the above tasks are diffusion model and variational autoencoder, and they both have the function of finding the inverse function. However, when we apply the above approach to a lightweight model, there will be defects because the lightweight model will be under parameterized to a large amount of raw data. Because of the above reason, important information $I(Y, X)$ that maps data X to target Y will also face the same problem. For this issue, we will explore it using the concept of information bottleneck [59]. The formula for information bottleneck is as follows:

$$I(X, X) \geq I(Y, X) \geq I(Y, f_\theta(X)) \geq ... \geq I(Y, \hat{Y}). \quad (6)$$

Generally speaking, $I(Y, X)$ will only occupy a very small part of $I(X, X)$. However, it is critical to the target mission. Therefore, even if the amount of information lost in the feedforward stage is not significant, as long as $I(Y, X)$ is covered, the training effect will be greatly affected. The lightweight model itself is in an under parameterized state, so it is easy to lose a lot of important information in the feedforward stage. Therefore, our goal for the lightweight model is how to accurately filter $I(Y, X)$ from $I(X, X)$. As for fully preserving the information of X, that is difficult to achieve. Based on the above analysis, we hope to propose a new deep neural network training method that can not only generate reliable gradients to update the model, but also be suitable for shallow and lightweight neural networks.

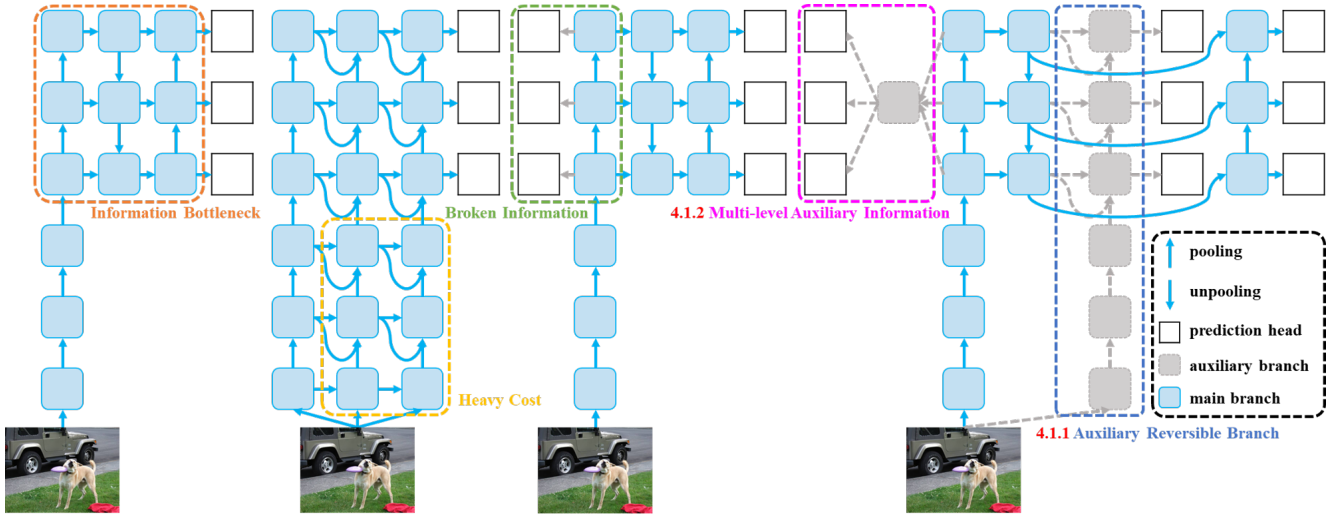


Figure 3. PGI and related network architectures and methods. (a) Path Aggregation Network (PAN)) [37], (b) Reversible Columns (RevCol) [3], (c) conventional deep supervision, and (d) our proposed Programmable Gradient Information (PGI). PGI is mainly composed of three components: (1) main branch: architecture used for inference, (2) auxiliary reversible branch: generate reliable gradients to supply main branch for backward transmission, and (3) multi-level auxiliary information: control main branch learning plannable multi-level of semantic information.

4. Methodology

4.1. Programmable Gradient Information

In order to solve the aforementioned problems, we propose a new auxiliary supervision framework called Programmable Gradient Information (PGI), as shown in Figure 3 (d). PGI mainly includes three components, namely (1) main branch, (2) auxiliary reversible branch, and (3) multi-level auxiliary information. From Figure 3 (d) we see that the inference process of PGI only uses main branch and therefore does not require any additional inference cost. As for the other two components, they are used to solve or slow down several important issues in deep learning methods. Among them, auxiliary reversible branch is designed to deal with the problems caused by the deepening of neural networks. Network deepening will cause information bottleneck, which will make the loss function unable to generate reliable gradients. As for multi-level auxiliary information, it is designed to handle the error accumulation problem caused by deep supervision, especially for the architecture and lightweight model of multiple prediction branch. Next, we will introduce these two components step by step.

4.1.1 Auxiliary Reversible Branch

In PGI, we propose auxiliary reversible branch to generate reliable gradients and update network parameters. By providing information that maps from data to targets, the loss function can provide guidance and avoid the possibility of finding false correlations from incomplete feedforward features that are less relevant to the target. We propose the maintenance of complete information by introducing reversible architecture, but adding main branch to reversible architecture will consume a lot of inference costs. We analyzed the architecture of Figure 3 (b) and found that when additional connections from deep to shallow layers are added, the inference time will increase by 20%. When we repeatedly add the input data to the high-resolution computing layer of the network (yellow box), the inference time even exceeds twice the time.

Since our goal is to use reversible architecture to obtain reliable gradients, "reversible" is not the only necessary condition in the inference stage. In view of this, we regard reversible branch as an expansion of deep supervision branch, and then design auxiliary reversible branch, as shown in Figure 3 (d). As for the main branch deep features that would have lost important information due to information bottleneck, they will be able to receive reliable gradient information from the auxiliary reversible branch. These gradient information will drive parameter learning to assist in extracting correct and important information, and the above actions can enable the main branch to obtain features that are more effective for the target task. Moreover, the reversible architecture performs worse on shallow networks than on general networks because complex tasks require conversion in deeper networks. Our proposed method does not force the main branch to retain complete original information but updates it by generating useful gradient through the auxiliary supervision mechanism. The advantage of this design is that the proposed method can also be applied to shallower networks.

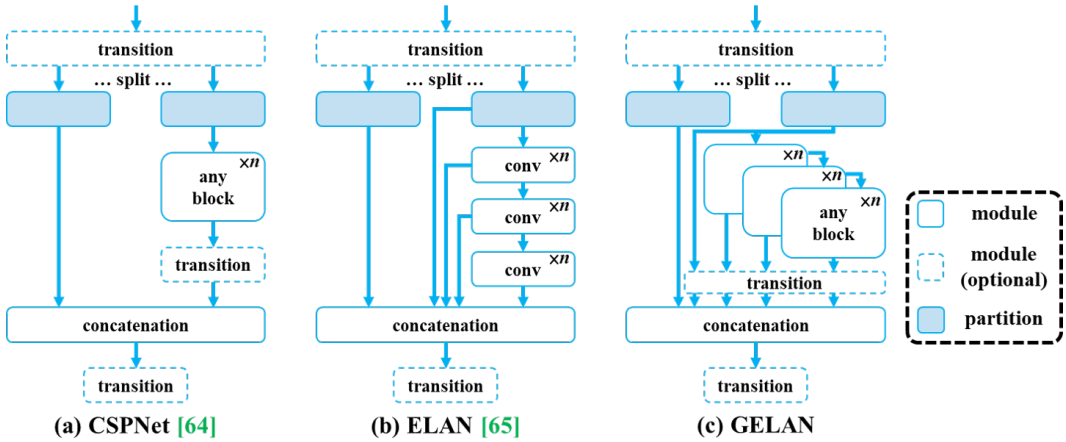


Figure 4. The architecture of GELAN: (a) CSPNet [64], (b) ELAN [65], and (c) proposed GELAN. We imitate CSPNet and extend ELAN into GELAN that can support any computational blocks.

Finally, since auxiliary reversible branch can be removed during the inference phase, the inference capabilities of the original network can be retained. We can also choose any reversible architectures in PGI to play the role of auxiliary reversible branch.

4.1.2 Multi-level Auxiliary Information

In this section we will discuss how multi-level auxiliary information works. The deep supervision architecture including multiple prediction branch is shown in Figure 3 (c). For object detection, different feature pyramids can be used to perform different tasks, for example together they can detect objects of different sizes. Therefore, after connecting to the deep supervision branch, the shallow features will be guided to learn the features required for small object detection, and at this time the system will regard the positions of objects of other sizes as the background. However, the above deed will cause the deep feature pyramids to lose a lot of information needed to predict the target object. Regarding this issue, we believe that each feature pyramid needs to receive information about all target objects so that subsequent main branch can retain complete information to learn predictions for various targets.

The concept of multi-level auxiliary information is to insert an integration network between the feature pyramid hierarchy layers of auxiliary supervision and the main branch, and then uses it to combine returned gradients from different prediction heads, as shown in Figure 3 (d). Multi-level auxiliary information is then to aggregate the gradient information containing all target objects, and pass it to the main branch and then update parameters. At this time, the characteristics of the main branch's feature pyramid hierarchy will not be dominated by some specific object's information. As a result, our method can alleviate the broken information problem in deep supervision. In addition, any integrated network can be used in multi-level auxiliary information. Therefore, we can plan the required semantic levels to guide the learning of network architectures of different sizes.

4.2. Generalized ELAN

In this Section we describe the proposed new network architecture – GELAN. By combining two neural network architectures, CSPNet [64] and ELAN [65], which are designed with gradient path planning, we designed generalized efficient layer aggregation network (GELAN) that takes into account lighweight, inference speed, and accuracy. Its overall architecture is shown in Figure 4. We generalized the capability of ELAN [65], which originally only used stacking of convolutional layers, to a new architecture that can use any computational blocks.

5. Experiments

5.1. Experimental Setup

We verify the proposed method with MS COCO dataset. All experimental setups follow YOLOv7 AF [63], while the dataset is MS COCO 2017 splitting. All models we mentioned are trained using the train-from-scratch strategy, and the total number of training times is 500 epochs. In setting the learning rate, we use linear warm-up in the first three epochs, and the subsequent epochs set the corresponding decay manner according to the model scale. As for the last 15 epochs, we turn mosaic data augmentation off. For more settings, please refer to Appendix.

5.2. Implimentation Details

We built general and extended version of YOLOv9 based on YOLOv7 [63] and Dynamic YOLOv7 [36] respectively. In the design of the network architecture, we replaced ELAN [65] with GELAN using CSPNet blocks [64] with planned RepConv [63] as computational blocks. We also simplified downsampling module and optimized anchor-free prediction head. As for the auxiliary loss part of PGI, we completely follow YOLOv7's auxiliary head setting. Please see Appendix for more details.

Table 1. Comparison of state-of-the-art real-time object detectors.

Model	#Param. (M)	FLOPs (G)	$AP^{val}_{50:95}$ (%)	AP^{val}_{50} (%)	AP^{val}_{75} (%)	AP^{val}_{S} (%)	AP^{val}_{M} (%)	AP^{val}_{L} (%)
YOLOv5-N r7.0 [14]	1.9	4.5	28.0	45.7	–	–	–	–
YOLOv5-S r7.0 [14]	7.2	16.5	37.4	56.8	–	–	–	–
YOLOv5-M r7.0 [14]	21.2	49.0	45.4	64.1	–	–	–	–
YOLOv5-L r7.0 [14]	46.5	109.1	49.0	67.3	–	–	–	–
YOLOv5-X r7.0 [14]	86.7	205.7	50.7	68.9	–	–	–	–
YOLOv6-N v3.0 [30]	4.7	11.4	37.0	52.7	–	–	–	–
YOLOv6-S v3.0 [30]	18.5	45.3	44.3	61.2	–	–	–	–
YOLOv6-M v3.0 [30]	34.9	85.8	49.1	66.1	–	–	–	–
YOLOv6-L v3.0 [30]	59.6	150.7	51.8	69.2	–	–	–	–
YOLOv7 [63]	36.9	104.7	51.2	69.7	55.9	31.8	55.5	65.0
YOLOv7-X [63]	71.3	189.9	52.9	71.1	51.4	36.9	57.7	68.6
YOLOv7-N AF [63]	3.1	8.7	37.6	53.3	40.6	18.7	41.7	52.8
YOLOv7-S AF [63]	11.0	28.1	45.1	61.8	48.9	25.7	50.2	61.2
YOLOv7 AF [63]	43.6	130.5	53.0	70.2	57.5	35.8	58.7	68.9
YOLOv8-N [15]	3.2	8.7	37.3	52.6	–	–	–	–
YOLOv8-S [15]	11.2	28.6	44.9	61.8	–	–	–	–
YOLOv8-M [15]	25.9	78.9	50.2	67.2	–	–	–	–
YOLOv8-L [15]	43.7	165.2	52.9	69.8	57.5	35.3	58.3	69.8
YOLOv8-X [15]	68.2	257.8	53.9	71.0	58.7	35.7	59.3	70.7
DAMO YOLO-T [75]	8.5	18.1	42.0	58.0	45.2	23.0	46.1	58.5
DAMO YOLO-S [75]	12.3	37.8	46.0	61.9	49.5	25.9	50.6	62.5
DAMO YOLO-M [75]	28.2	61.8	49.2	65.5	53.0	29.7	53.1	66.1
DAMO YOLO-L [75]	42.1	97.3	50.8	67.5	55.5	33.2	55.7	66.6
Gold YOLO-N [61]	5.6	12.1	39.6	55.7	–	19.7	44.1	57.0
Gold YOLO-S [61]	21.5	46.0	45.4	62.5	–	25.3	50.2	62.6
Gold YOLO-M [61]	41.3	87.5	49.8	67.0	–	32.3	55.3	66.3
Gold YOLO-L [61]	75.1	151.7	51.8	68.9	–	34.1	57.4	68.2
YOLO MS-N [7]	4.5	17.4	43.4	60.4	47.6	23.7	48.3	60.3
YOLO MS-S [7]	8.1	31.2	46.2	63.7	50.5	26.9	50.5	63.0
YOLO MS [7]	22.2	80.2	51.0	68.6	55.7	33.1	56.1	66.5
GELAN-S (Ours)	7.1	26.4	46.7	63.0	50.7	25.9	51.5	64.0
GELAN-M (Ours)	20.0	76.3	51.1	67.9	55.7	33.6	56.4	67.3
GELAN-C (Ours)	25.3	102.1	52.5	69.5	57.3	35.8	57.6	69.4
GELAN-E (Ours)	57.3	189.0	55.0	71.9	60.0	38.0	60.6	70.9
YOLOv9-S (Ours)	7.1	26.4	46.8	63.4	50.7	26.6	56.0	64.5
YOLOv9-M (Ours)	20.0	76.3	51.4	68.1	56.1	33.6	57.0	68.0
YOLOv9-C (Ours)	25.3	102.1	53.0	70.2	57.8	36.2	58.5	69.3
YOLOv9-E (Ours)	57.3	189.0	55.6	72.8	60.6	40.2	61.0	71.4

5.3. Comparison with state-of-the-arts

Table 1 lists comparison of our proposed YOLOv9 with other train-from-scratch real-time object detectors. Overall, the best performing methods among existing methods are YOLO MS-S [7] for lightweight models, YOLO MS [7] for medium models, YOLOv7 AF [63] for general models, and YOLOv8-X [15] for large models. Compared with lightweight and medium model YOLO MS [7], YOLOv9 has about 10% less parameters and 5∼15% less calculations, but still has a 0.4∼0.6% improvement in AP. Compared with YOLOv7 AF, YOLOv9-C has 42% less parameters and 22% less calculations, but achieves the same AP (53%). Compared with YOLOv8-X, YOLOv9-E has 16% less parameters, 27% less calculations, and has significant improvement of 1.7% AP. The above comparison results show that our proposed YOLOv9 has significantly improved in all aspects compared with existing methods.

On the other hand, we also include ImageNet pretrained model in the comparison, and the results are shown in Figure 5. We compare them based on the parameters and the amount of computation respectively. In terms of the number of parameters, the best performing large model is RT DETR [43]. From Figure 5, we can see that YOLOv9 using conventional convolution is even better than YOLO MS using depth-wise convolution in parameter utilization. As for the parameter utilization of large models, it also greatly surpasses RT DETR using ImageNet pretrained model. Even better is that in the deep model, YOLOv9 shows the huge advantages of using PGI. By accurately retaining and extracting the information needed to map the data to the target, our method requires only 66% of the parameters while maintaining the accuracy as RT DETR-X.

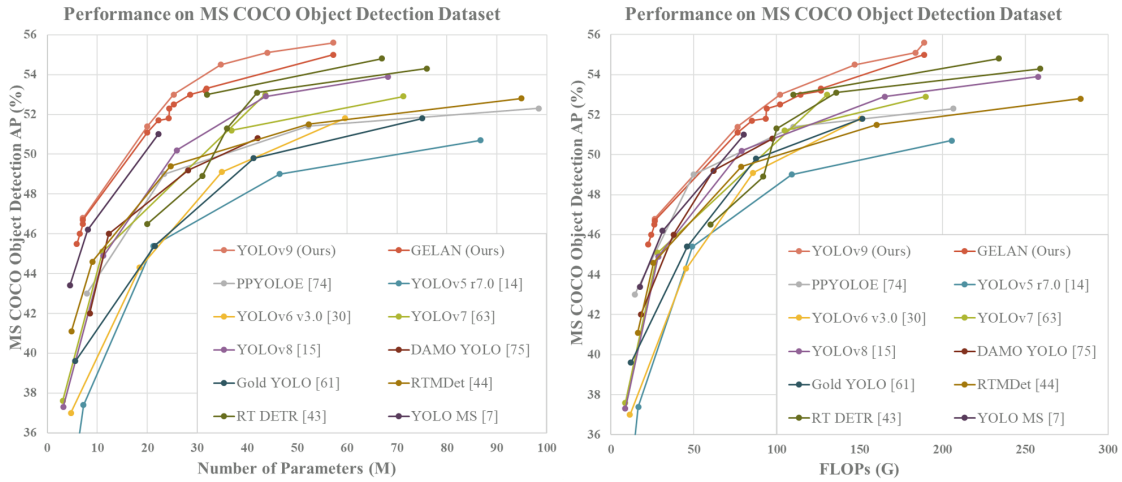


Figure 5. Comparison of state-of-the-art real-time object detectors. The methods participating in the comparison all use ImageNet as pre-trained weights, including RT DETR [43], RTMDet [44], and PP-YOLOE [74], etc. The YOLOv9 that uses train-from-scratch method clearly surpasses the performance of other methods.

As for the amount of computation, the best existing models from the smallest to the largest are YOLO MS [7], PP YOLOE [74], and RT DETR [43]. From Figure 5, we can see that YOLOv9 is far superior to the train-from-scratch methods in terms of computational complexity. In addition, if compared with those based on depth-wise convolution and ImageNet-based pretrained models, YOLOv9 is also very competitive.

5.4. Ablation Studies

5.4.1 Generalized ELAN

For GELAN, we first do ablation studies for computational blocks. We used Res blocks [21], Dark blocks [49], and CSP blocks [64] to conduct experiments, respectively. Table 2 shows that after replacing convolutional layers in ELAN with different computational blocks, the system can maintain good performance. Users are indeed free to replace computational blocks and use them on their respective inference devices. Among different computational block replacements, CSP blocks perform particularly well. They not only reduce the amount of parameters and computation, but also improve AP by 0.7%. Therefore, we choose CSP-ELAN as the component unit of GELAN in YOLOv9.

Table 2. Ablation study on various computational blocks.

Model	CB type	#Param.	FLOPs	$AP^{val}_{50:95}$
GELAN-S	Conv	6.2M	23.5G	44.8%
GELAN-S	Res [21]	5.4M	21.0G	44.3%
GELAN-S	Dark [49]	5.7M	21.8G	44.5%
GELAN-S	CSP [64]	5.9M	22.4G	45.5%

[1] CB type nedotes as computational block type.
[2] -S nedotes small size model.

Next, we conduct ELAN block-depth and CSP block-depth experiments on GELAN of different sizes, and display the results in Table 3. We can see that when the depth of ELAN is increased from 1 to 2, the accuracy is significantly improved. But when the depth is greater than or equal to 2, no matter it is improving the ELAN depth or the CSP depth, the number of parameters, the amount of computation, and the accuracy will always show a linear relationship. This means GELAN is not sensitive to the depth. In other words, users can arbitrarily combine the components in GELAN to design the network architecture, and have a model with stable performance without special design. In Table 3, for YOLOv9-{S,M,C}, we set the pairing of the ELAN depth and the CSP depth to {{2, 3}, {2, 1}, {2, 1}}.

Table 3. Ablation study on ELAN and CSP depth.

Model	$\mathbf{D}_{ELAN}$	$\mathbf{D}_{CSP}$	#Param.	FLOPs	$AP^{val}_{50:95}$
GELAN-S	2	1	5.9M	22.4G	45.5%
GELAN-S	2	2	6.5M	24.4G	46.0%
GELAN-S	3	1	7.1M	26.3G	46.5%
GELAN-S	2	3	7.1M	26.4G	46.7%
GELAN-M	2	1	20.0M	76.3G	51.1%
GELAN-M	2	2	22.2M	85.1G	51.7%
GELAN-M	3	1	24.3M	93.5G	51.8%
GELAN-M	2	3	24.4M	94.0G	52.3%
GELAN-C	1	1	18.9M	77.5G	50.7%
GELAN-C	2	1	25.3M	102.1G	52.5%
GELAN-C	2	2	28.6M	114.4G	53.0%
GELAN-C	3	1	31.7M	126.8G	53.2%
GELAN-C	2	3	31.9M	126.7G	53.3%

[1] $\mathbf{D}_{ELAN}$ and $\mathbf{D}_{CSP}$ respectively nedotes depth of ELAN and CSP.
[2] -{S, M, C} indicate small, medium, and compact models.

5.4.2 Programmable Gradient Information

In terms of PGI, we performed ablation studies on auxiliary reversible branch and multi-level auxiliary information on the backbone and neck, respectively. We designed auxiliary reversible branch ICN to use DHLC [34] linkage to obtain multi-level reversible information. As for multi-level auxiliary information, we use FPN and PAN for ablation studies and the role of PFH is equivalent to the traditional deep supervision. The results of all experiments are listed in Table 4. From Table 4, we can see that PFH is only effective in deep models, while our proposed PGI can improve accuracy under different combinations. Especially when using ICN, we get stable and better results. We also tried to apply the lead-head guided assignment proposed in YOLOv7 [63] to the PGI's auxiliary supervision, and achieved much better performance.

Table 4. Ablation study on PGI of backbone and neck.

Model	$G_{backbone}$	G_{neck}	$AP^{val}_{50:95}$	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
GELAN-C	–	–	52.5%	35.8%	57.6%	**69.4%**
GELAN-C	PFH	–	52.5%	35.3%	58.1%	68.9%
GELAN-C	FPN	–	52.6%	35.3%	58.1%	68.9%
GELAN-C	–	ICN	52.7%	35.3%	58.4%	68.9%
GELAN-C	FPN	ICN	52.8%	35.8%	58.2%	69.1%
GELAN-C	ICN	–	**52.9%**	35.2%	**58.7%**	68.6%
GELAN-C	LHG-ICN	–	**53.0%**	**36.3%**	58.5%	69.1%
GELAN-E	–	–	55.0%	38.0%	60.6%	70.9%
GELAN-E	PFH	–	55.3%	38.3%	60.3%	71.6%
GELAN-E	FPN	–	**55.6%**	**40.2%**	61.0%	71.4%
GELAN-E	PAN	–	55.5%	39.0%	**61.1%**	71.5%
GELAN-E	FPN	ICN	**55.6%**	39.8%	60.9%	**71.9%**

[1] $\mathbf{D}_{ELAN}$ and $\mathbf{D}_{CSP}$ respectively nedotes depth of ELAN and CSP.
[2] LHG indicates lead head guided training proposed by YOLOv7 [63].

We further implemented the concepts of PGI and deep supervision on models of various sizes and compared the results, these results are shown in Table 5. As analyzed at the beginning, introduction of deep supervision will cause a loss of accuracy for shallow models. As for general models, introducing deep supervision will cause unstable performance, and the design concept of deep supervision can only bring gains in extremely deep models. The proposed PGI can effectively handle problems such as information bottleneck and information broken, and can comprehensively improve the accuracy of models of different sizes. The concept of PGI brings two valuable contributions. The first one is to make the auxiliary supervision method applicable to shallow models, while the second one is to make the deep model training process obtain more reliable gradients. These gradients enable deep models to use more accurate information to establish correct correlations between data and targets.

Table 5. Ablation study on PGI.

Model	$AP^{val}_{50:95}$		AP^{val}_{50}		AP^{val}_{75}	
GELAN-S	46.7%		63.0%		**50.7%**	
+ DS	46.5%	-0.2	62.9%	-0.1	50.5%	-0.2
+ PGI	**46.8%**	+0.1	**63.4%**	+0.4	**50.7%**	=
GELAN-M	51.1%		67.9%		55.7%	
+ DS	51.2%	+0.1	**68.2%**	+0.3	55.7%	=
+ PGI	**51.4%**	+0.3	68.1%	+0.2	**56.1%**	+0.4
GELAN-C	52.5%		69.5%		57.3%	
+ DS	52.5%	=	69.9%	+0.4	57.1%	-0.2
+ PGI	**53.0%**	+0.5	**70.3%**	+0.8	**57.8%**	+0.5
GELAN-E	55.0%		71.9%		60.0%	
+ DS	55.3%	+0.3	72.3%	+0.4	60.2%	+0.2
+ PGI	**55.6%**	+0.6	**72.8%**	+0.9	**60.6%**	+0.6

[1] DS indicates deep supervision.
[2] -{S, M, C, E} indicate small, medium, compact, and extended models.

Finally, we show in the table the results of gradually increasing components from baseline YOLOv7 to YOLOv9-E. The GELAN and PGI we proposed have brought all-round improvement to the model.

Table 6. Ablation study on GELAN and PGI.

Model	#Param.	FLOPs	$AP^{val}_{50:95}$	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLOv7 [63]	36.9	104.7	51.2%	31.8%	55.5%	65.0%
+ AF [63]	43.6	130.5	53.0%	35.8%	58.7%	68.9%
+ GELAN	41.2	126.4	53.2%	36.2%	58.5%	69.9%
+ DHLC [34]	57.3	189.0	55.0%	38.0%	60.6%	70.9%
+ PGI	57.3	189.0	55.6%	40.2%	61.0%	71.4%

5.5. Visualization

This section will explore the information bottleneck issues and visualize them. In addition, we will also visualize how the proposed PGI uses reliable gradients to find the correct correlations between data and targets. In Figure 6 we show the visualization results of feature maps obtained by using random initial weights as feedforward under different architectures. We can see that as the number of layers increases, the original information of all architectures gradually decreases. For example, at the 50^{th} layer of the PlainNet, it is difficult to see the location of objects, and all distinguishable features will be lost at the 100^{th} layer. As for ResNet, although the position of object can still be seen at the 50^{th} layer, the boundary information has been lost. When the depth reached to the 100^{th} layer, the whole image becomes blurry. Both CSPNet and the proposed GELAN perform very well, and they both can maintain features that support clear identification of objects until the 200^{th} layer. Among the comparisons, GELAN has more stable results and clearer boundary information.

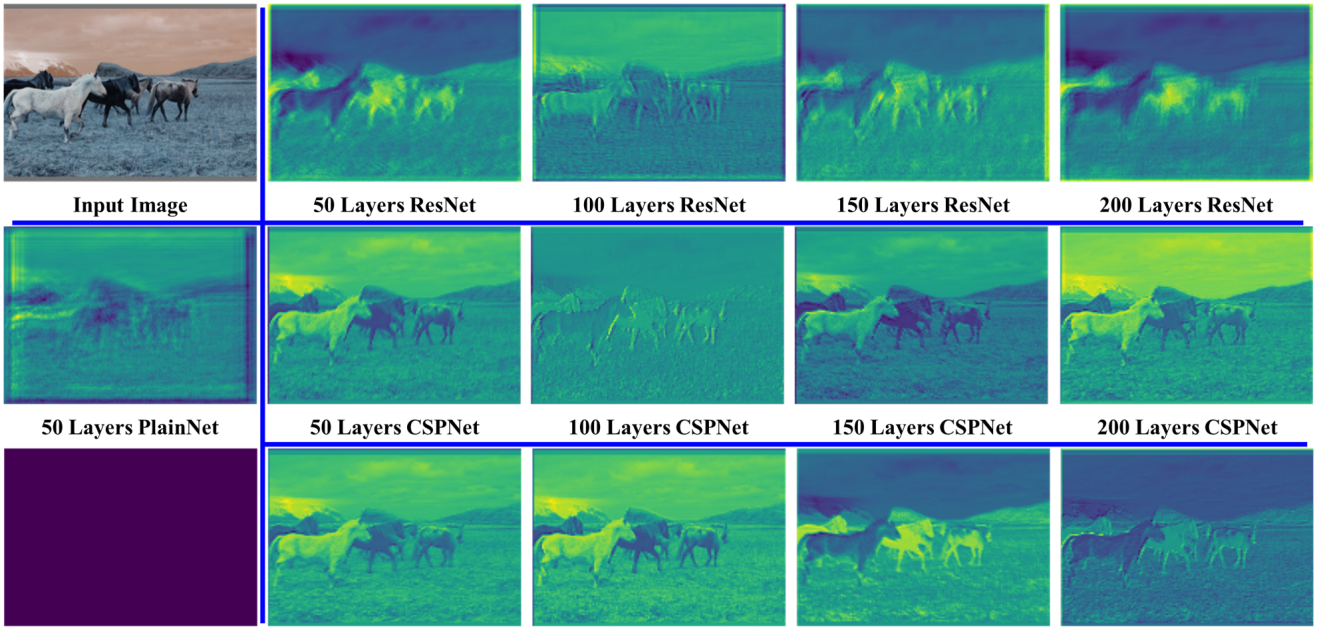


Figure 6. Feature maps (visualization results) output by random initial weights of PlainNet, ResNet, CSPNet, and GELAN at different depths. After 100 layers, ResNet begins to produce feedforward output that is enough to obfuscate object information. Our proposed GELAN can still retain quite complete information up to the 150^{th} layer, and is still sufficiently discriminative up to the 200^{th} layer.

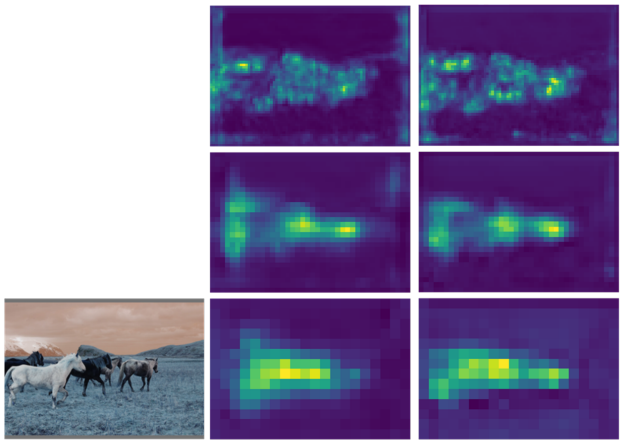


Figure 7. PAN feature maps (visualization results) of GELAN and YOLOv9 (GELAN + PGI) after one epoch of bias warm-up. GELAN originally had some divergence, but after adding PGI's reversible branch, it is more capable of focusing on the target object.

Figure 7 is used to show whether PGI can provide more reliable gradients during the training process, so that the parameters used for updating can effectively capture the relationship between the input data and the target. Figure 7 shows the visualization results of the feature map of GELAN and YOLOv9 (GELAN + PGI) in PAN bias warm-up. From the comparison of Figure 7(b) and (c), we can clearly see that PGI accurately and concisely captures the area containing objects. As for GELAN that does not use PGI, we found that it had divergence when detecting object boundaries, and it also produced unexpected responses in some background areas. This experiment confirms that PGI can indeed provide better gradients to update parameters and enable the feedforward stage of the main branch to retain more important features.

6. Conclusions

In this paper, we propose to use PGI to solve the information bottleneck problem and the problem that the deep supervision mechanism is not suitable for lightweight neural networks. We designed GELAN, a highly efficient and lightweight neural network. In terms of object detection, GELAN has strong and stable performance at different computational blocks and depth settings. It can indeed be widely expanded into a model suitable for various inference devices. For the above two issues, the introduction of PGI allows both lightweight models and deep models to achieve significant improvements in accuracy. The YOLOv9, designed by combining PGI and GELAN, has shown strong competitiveness. Its excellent design allows the deep model to reduce the number of parameters by 49% and the amount of calculations by 43% compared with YOLOv8, but it still has a 0.6% AP improvement on MS COCO dataset.

7. Acknowledgements

The authors wish to thank National Center for High-performance Computing (NCHC) for providing computational and storage resources.

References

[1] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. BEiT: BERT pre-training of image transformers. In *International Conference on Learning Representations (ICLR)*, 2022. 2

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 3

[3] Yuxuan Cai, Yizhuang Zhou, Qi Han, Jianjian Sun, Xiangwen Kong, Jun Li, and Xiangyu Zhang. Reversible column networks. In *International Conference on Learning Representations (ICLR)*, 2023. 2, 3, 5

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 213–229, 2020. 3

[5] Kean Chen, Weiyao Lin, Jianguo Li, John See, Ji Wang, and Junni Zou. AP-loss for accurate one-stage object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 43(11):3782–3798, 2020. 1

[6] Yabo Chen, Yuchen Liu, Dongsheng Jiang, Xiaopeng Zhang, Wenrui Dai, Hongkai Xiong, and Qi Tian. SdAE: Self-distillated masked autoencoder. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 108–124, 2022. 2

[7] Yuming Chen, Xinbin Yuan, Ruiqi Wu, Jiabao Wang, Qibin Hou, and Ming-Ming Cheng. YOLO-MS: rethinking multi-scale representation learning for real-time object detection. *arXiv preprint arXiv:2308.05480*, 2023. 1, 3, 7, 8

[8] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. DaVIT: Dual attention vision transformers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 74–92, 2022. 1

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021. 1, 2

[10] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R Scott, and Weilin Huang. TOOD: Task-aligned one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3490–3499, 2021. 1

[11] Shang-Hua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip Torr. Res2Net: A new multi-scale backbone architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 43(2):652–662, 2019. 3

[12] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and Jian Sun. OTA: Optimal transport assignment for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 303–312, 2021. 1

[13] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: Exceeding YOLO series in 2021. *arXiv preprint arXiv:2107.08430*, 2021. 3

[14] Jocher Glenn. YOLOv5 release v7.0. https://github.com/ultralytics/yolov5/releases/tag/v7.0, 2022. 3, 7

[15] Jocher Glenn. YOLOv8 release v8.1.0. https://github.com/ultralytics/ultralytics/releases/tag/v8.1.0, 2024. 3, 7

[16] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 2, 3

[17] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. 1

[18] Chaoxu Guo, Bin Fan, Qian Zhang, Shiming Xiang, and Chunhong Pan. AugFPN: Improving multi-scale feature learning for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12595–12604, 2020. 1, 3

[19] Qi Han, Yuxuan Cai, and Xiangyu Zhang. RevColV2: Exploring disentangled representations in masked image modeling. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2, 3

[20] Zeeshan Hayder, Xuming He, and Mathieu Salzmann. Boundary-aware instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5696–5704, 2017. 1, 3

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 4, 8

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 630–645. Springer, 2016. 1, 4

[23] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017. 1

[24] Kuan-Chih Huang, Tsung-Han Wu, Hung-Ting Su, and Winston H Hsu. MonoDTR: Monocular 3D object detection with depth-aware transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4012–4021, 2022. 1, 3

[25] Lin Huang, Weisheng Li, Linlin Shen, Haojie Fu, Xue Xiao, and Suihan Xiao. YOLOCS: Object detection based on dense channel compression for feature spatial solidification. *arXiv preprint arXiv:2305.04170*, 2023. 3

[26] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning (ICML)*, pages 4651–4664, 2021. 1

[27] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, volume 1, page 2, 2019. 2

[28] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial Intelligence and Statistics*, pages 562–570, 2015. 1, 2, 3

[29] Alex Levinshtein, Alborz Rezazadeh Sereshkeh, and Konstantinos Derpanis. DATNet: Dense auxiliary tasks for object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1419–1427, 2020. 1, 3

[30] Chuyi Li, Lulu Li, Yifei Geng, Hongliang Jiang, Meng Cheng, Bo Zhang, Zaidan Ke, Xiaoming Xu, and Xiangxiang Chu. YOLOv6 v3.0: A full-scale reloading. *arXiv preprint arXiv:2301.05586*, 2023. 3, 7, 2, 4

[31] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. YOLOv6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022. 3

[32] Hao Li, Jinguo Zhu, Xiaohu Jiang, Xizhou Zhu, Hongsheng Li, Chun Yuan, Xiaohua Wang, Yu Qiao, Xiaogang Wang, Wenhai Wang, et al. Uni-perceiver v2: A generalist model for large-scale vision and vision-language tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2691–2700, 2023. 1

[33] Shuai Li, Chenhang He, Ruihuang Li, and Lei Zhang. A dual weighting label assignment scheme for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9387–9396, 2022. 1

[34] Tingting Liang, Xiaojie Chu, Yudong Liu, Yongtao Wang, Zhi Tang, Wei Chu, Jingdong Chen, and Haibin Ling. CBNet: A composite backbone network architecture for object detection. *IEEE Transactions on Image Processing (TIP)*, 2022. 3, 9

[35] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017. 3

[36] Zhihao Lin, Yongtao Wang, Jinhe Zhang, and Xiaojie Chu. DynamicDet: A unified dynamic architecture for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6282–6291, 2023. 3, 6, 2, 4

[37] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8759–8768, 2018. 3, 5

[38] Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, and Yunfan Liu. Vmamba: Visual state space model. *arXiv preprint arXiv:2401.10166*, 2024. 1

[39] Yudong Liu, Yongtao Wang, Siwei Wang, TingTing Liang, Qijie Zhao, Zhi Tang, and Haibin Ling. CBNet: A novel composite backbone network architecture for object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 11653–11660, 2020. 3

[40] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1

[41] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022, 2021. 1

[42] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11976–11986, 2022. 1

[43] Wenyu Lv, Shangliang Xu, Yian Zhao, Guanzhong Wang, Jinman Wei, Cheng Cui, Yuning Du, Qingqing Dang, and Yi Liu. DETRs beat YOLOs on real-time object detection. *arXiv preprint arXiv:2304.08069*, 2023. 1, 3, 7, 8, 2, 4

[44] Chengqi Lyu, Wenwei Zhang, Haian Huang, Yue Zhou, Yudong Wang, Yanyi Liu, Shilong Zhang, and Kai Chen. RTMDet: An empirical study of designing real-time object detectors. *arXiv preprint arXiv:2212.07784*, 2022. 8, 2, 3, 4

[45] Kemal Oksuz, Baris Can Cam, Emre Akbas, and Sinan Kalkan. A ranking-based, balanced loss function unifying classification and localisation in object detection. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:15534–15545, 2020. 1

[46] Kemal Oksuz, Baris Can Cam, Emre Akbas, and Sinan Kalkan. Rank & sort loss for object detection and instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3009–3018, 2021. 1

[47] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. 3

[48] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017. 3

[49] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 3, 8

[50] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, 2019. 1

[51] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Object detection from scratch with deep supervision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(2):398–412, 2019. 1, 2

[52] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation.

In *Conference on Robot Learning (CoRL)*, pages 785–799, 2023. 1

[53] Peize Sun, Yi Jiang, Enze Xie, Wenqi Shao, Zehuan Yuan, Changhu Wang, and Ping Luo. What makes for end-to-end object detection? In *International Conference on Machine Learning (ICML)*, pages 9934–9944, 2021. 3

[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 1, 2, 3

[55] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. 1

[56] Zineng Tang, Jaemin Cho, Jie Lei, and Mohit Bansal. Perceiver-VL: Efficient vision-and-language modeling with iterative latent attention. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 4410–4420, 2023. 1

[57] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9627–9636, 2019. 3

[58] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: A simple and strong anchor-free object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(4):1922–1933, 2022. 3

[59] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Information Theory Workshop (ITW)*, pages 1–5, 2015. 2, 4

[60] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. MaxVIT: Multi-axis vision transformer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 459–479, 2022. 1

[61] Chengcheng Wang, Wei He, Ying Nie, Jianyuan Guo, Chuanjian Liu, Kai Han, and Yunhe Wang. Gold-YOLO: Efficient object detector via gather-and-distribute mechanism. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 3, 7, 2, 4

[62] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-YOLOv4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13029–13038, 2021. 3

[63] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7464–7475, 2023. 3, 6, 7, 9, 1

[64] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A new backbone that can enhance learning capability of CNN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 390–391, 2020. 3, 6, 8

[65] Chien-Yao Wang, Hong-Yuan Mark Liao, and I-Hau Yeh. Designing network design strategies through gradient path analysis. *Journal of Information Science and Engineering (JISE)*, 39(4):975–995, 2023. 2, 3, 6

[66] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. You only learn one representation: Unified network for multiple tasks. *Journal of Information Science & Engineering (JISE)*, 39(3):691–709, 2023. 2, 3, 4

[67] Jianfeng Wang, Lin Song, Zeming Li, Hongbin Sun, Jian Sun, and Nanning Zheng. End-to-end object detection with fully convolutional network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15849–15858, 2021. 1, 3

[68] Liwei Wang, Chen-Yu Lee, Zhuowen Tu, and Svetlana Lazebnik. Training deeper convolutional networks with deep supervision. *arXiv preprint arXiv:1505.02496*, 2015. 1, 2, 3

[69] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 568–578, 2021. 1

[70] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. PVT v2: Improved baselines with pyramid vision transformer. *Computational Visual Media*, 8(3):415–424, 2022. 1

[71] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. ConvNeXt v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16133–16142, 2023. 1, 2

[72] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1492–1500, 2017. 1

[73] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. SimMIM: A simple framework for masked image modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9653–9663, 2022. 2

[74] Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao Chang, Cheng Cui, Kaipeng Deng, Guanzhong Wang, Qingqing Dang, Shengyu Wei, Yuning Du, et al. PP-YOLOE: An evolved version of YOLO. *arXiv preprint arXiv:2203.16250*, 2022. 3, 8, 2, 4

[75] Xianzhe Xu, Yiqi Jiang, Weihua Chen, Yilun Huang, Yuan Zhang, and Xiuyu Sun. DAMO-YOLO: A report on real-time object detection design. *arXiv preprint arXiv:2211.15444*, 2022. 3, 7, 2, 4

[76] Renrui Zhang, Han Qiu, Tai Wang, Ziyu Guo, Ziteng Cui, Yu Qiao, Hongsheng Li, and Peng Gao. MonoDETR: Depth-guided transformer for monocular 3D object detection. In

Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 9155–9166, 2023. 1, 3

[77] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-IoU loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 12993–13000, 2020. 1

[78] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, and Ruigang Yang. IoU loss for 2D/3D object detection. In *International Conference on 3D Vision (3DV)*, pages 85–94, 2019. 1

[79] Benjin Zhu, Jianfeng Wang, Zhengkai Jiang, Fuhang Zong, Songtao Liu, Zeming Li, and Jian Sun. AutoAssign: Differentiable label assignment for dense object detection. *arXiv preprint arXiv:2007.03496*, 2020. 1

[80] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024. 1

[81] Xizhou Zhu, Jinguo Zhu, Hao Li, Xiaoshi Wu, Hongsheng Li, Xiaohua Wang, and Jifeng Dai. Uni-perceiver: Pre-training unified architecture for generic perception for zero-shot and few-shot tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16804–16815, 2022. 1

[82] Zhuofan Zong, Guanglu Song, and Yu Liu. DETRs with collaborative hybrid assignments training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6748–6758, 2023. 3

Appendix

A. Implementation Details

Table 1. Hyper parameter settings of YOLOv9.

hyper parameter	value
epochs	500
optimizer	SGD
initial learning rate	0.01
finish learning rate	0.0001
learning rate decay	linear
momentum	0.937
weight decay	0.0005
warm-up epochs	3
warm-up momentum	0.8
warm-up bias learning rate	0.1
box loss gain	7.5
class loss gain	0.5
DFL loss gain	1.5
HSV saturation augmentation	0.7
HSV value augmentation	0.4
translation augmentation	0.1
scale augmentation	0.9
mosaic augmentation	1.0
MixUp augmentation	0.15
copy & paste augmentation	0.3
close mosaic epochs	15

The training parameters of YOLOv9 are shown in Table 1. We fully follow the settings of YOLOv7 AF [63], which is to use SGD optimizer to train 500 epochs. We first warm-up for 3 epochs and only update the bias during the warm-up stage. Next we step down from the initial learning rate 0.01 to 0.0001 in linear decay manner, and the data augmentation settings are listed in the bottom part of Table 1. We shut down mosaic data augmentation operations on the last 15 epochs.

Table 2. Network configurations of YOLOv9.

Index	Module	Route	Filters	Depth	Size	Stride
0	Conv	–	64	–	3	2
1	Conv	0	128	–	3	2
2	CSP-ELAN	1	256, 128, 64	2, 1	–	1
3	DOWN	2	256	–	3	2
4	CSP-ELAN	3	512, 256, 128	2, 1	–	1
5	DOWN	4	512	–	3	2
6	CSP-ELAN	5	512, 512, 256	2, 1	–	1
7	DOWN	6	512	–	3	2
8	CSP-ELAN	7	512, 512, 256	2, 1	–	1
9	SPP-ELAN	8	512, 256, 256	3, 1	–	1
10	Up	9	512	–	–	2
11	Concat	10, 6	1024	–	–	1
12	CSP-ELAN	11	512, 512, 256	2, 1	–	1
13	Up	12	512	–	–	2
14	Concat	13, 4	1024	–	–	1
15	CSP-ELAN	14	256, 256, 128	2, 1	–	1
16	DOWN	15	256	–	3	2
17	Concat	16, 12	768	–	–	1
18	CSP-ELAN	17	512, 512, 256	2, 1	–	1
19	DOWN	18	512	–	3	2
20	Concat	19, 9	1024	–	–	1
21	CSP-ELAN	20	512, 512, 256	2, 1	–	1
22	Predict	15, 18, 21	–	–	–	–

The network topology of YOLOv9 completely follows YOLOv7 AF [63], that is, we replace ELAN with the proposed CSP-ELAN block. As listed in Table 2, the depth parameters of CSP-ELAN are represented as ELAN depth and CSP depth, respectively. As for the parameters of CSP-ELAN filters, they are represented as ELAN output filter, CSP output filter, and CSP inside filter. In the downsampling module part, we simplify CSP-DOWN module to DOWN module. DOWN module is composed of a pooling layer with size 2 and stride 1, and a Conv layer with size 3 and stride 2. Finally, we optimized the prediction layer and replaced top, left, bottom, and right in the regression branch with decoupled branch.

Table 3. Comparison of state-of-the-art object detectors with different training settings.

	Model	#Param. (M)	FLOPs (G)	$AP_{50:95}$ (%)	AP_{50} (%)	AP_{75} (%)	AP_S (%)	AP_M (%)	AP_L (%)
Train-from-scratch	**Dy-YOLOv7 [36]**	–	181.7	53.9	72.2	58.7	35.3	57.6	66.4
	Dy-YOLOv7-X [36]	–	307.9	55.0	73.2	60.0	36.6	58.7	68.5
	YOLOv9-S (Ours)	7.1	26.4	46.8	63.4	50.7	26.6	56.0	64.5
	YOLOv9-M (Ours)	20.0	76.3	51.4	68.1	56.1	33.6	57.0	68.0
	YOLOv9-C (Ours)	25.3	102.1	53.0	70.2	57.8	36.2	58.5	69.3
	YOLOv9-E (Ours)	34.7	147.1	54.5	71.7	59.2	38.1	59.9	70.3
	YOLOv9-E (Ours)	44.0	183.9	55.1	72.3	60.7	38.7	60.6	71.4
	YOLOv9-E (Ours)	57.3	189.0	55.6	72.8	60.6	40.2	61.0	71.4
ImageNet Pretrained	**RTMDet-T [44]**	4.8	12.6	41.1	57.9	–	–	–	–
	RTMDet-S [44]	9.0	25.6	44.6	61.9	–	–	–	–
	RTMDet-M [44]	24.7	78.6	49.4	66.8	–	–	–	–
	RTMDet-L [44]	52.3	160.4	51.5	68.8	–	–	–	–
	RTMDet-X [44]	94.9	283.4	52.8	70.4	–	–	–	–
	PPYOLOE-S [74]	7.9	14.4	43.0	60.5	46.6	23.2	46.4	56.9
	PPYOLOE-M [74]	23.4	49.9	49.0	66.5	53.0	28.6	52.9	63.8
	PPYOLOE-L [74]	52.2	110.1	51.4	68.9	55.6	31.4	55.3	66.1
	PPYOLOE-X [74]	98.4	206.6	52.3	69.5	56.8	35.1	57.0	68.6
	RT DETR-L [43]	32	110	53.0	71.6	57.3	34.6	57.3	71.2
	RT DETR-X [43]	67	234	54.8	73.1	59.4	35.7	59.6	72.9
	RT DETR-R18 [43]	20	60	46.5	63.8	–	–	–	–
	RT DETR-R34 [43]	31	92	48.9	66.8	–	–	–	–
	RT DETR-R50M [43]	36	100	51.3	69.6	–	–	–	–
	RT DETR-R50 [43]	42	136	53.1	71.3	57.7	34.8	58.0	70.0
	RT DETR-R101 [43]	76	259	54.3	72.7	58.6	36.0	58.8	72.1
	Gold YOLO-S [61]	21.5	46.0	45.5	62.2	–	–	–	–
	Gold YOLO-M [61]	41.3	57.5	50.2	67.5	–	–	–	–
	Gold YOLO-L [61]	75.1	151.7	52.3	69.6	–	–	–	–
Knowledge Distillation	**YOLOv6-N v3.0 [30]**	4.7	11.4	37.5	53.1	–	–	–	–
	YOLOv6-S v3.0 [30]	18.5	45.3	45.0	61.8	–	–	–	–
	YOLOv6-M v3.0 [30]	34.9	85.8	50.0	66.9	–	–	–	–
	YOLOv6-L v3.0 [30]	59.6	150.7	52.8	70.3	–	–	–	–
	DAMO YOLO-T [75]	8.5	18.1	43.6	59.4	46.6	23.3	47.4	61.0
	DAMO YOLO-S [75]	16.3	37.8	47.7	63.5	51.1	26.9	51.7	64.9
	DAMO YOLO-M [75]	28.2	61.8	50.4	67.2	55.1	31.6	55.3	67.1
	DAMO YOLO-L [75]	42.1	97.3	51.9	68.5	56.7	33.3	57.0	67.6
	Gold YOLO-N [61]	5.6	12.1	39.9	55.9	–	–	–	–
	Gold YOLO-S [61]	21.5	46.0	46.1	63.3	–	–	–	–
	Gold YOLO-M [61]	41.3	57.5	50.9	68.2	–	–	–	–
	Gold YOLO-L [61]	75.1	151.7	53.2	70.5	–	–	–	–
Complex Setting	**Gold YOLO-S [61]**	21.5	46.0	46.4	63.4	–	–	–	–
	Gold YOLO-M [61]	41.3	57.5	51.1	68.5	–	–	–	–
	Gold YOLO-L [61]	75.1	151.7	53.3	70.9	–	–	–	–
	YOLOR-CSP [66]	52.9	120.4	52.8	71.2	57.6	–	–	–
	YOLOR-CSP-X [66]	96.9	226.8	54.8	73.1	59.7	–	–	–
	PPYOLOE+-S [74]	7.9	14.4	43.7	60.6	47.9	23.2	46.4	56.9
	PPYOLOE+-M [74]	23.4	49.9	49.8	67.1	54.5	31.8	53.9	66.2
	PPYOLOE+-L [74]	52.2	110.1	52.9	70.1	57.9	35.2	57.5	69.1
	PPYOLOE+-X [74]	98.4	206.6	54.7	72.0	59.9	37.9	59.3	70.4

B. More Comparison

We compare YOLOv9 to state-of-the-art real-time object detectors trained with different methods. It mainly includes four different training methods: (1) train-from-scratch: we have completed most of the comparisons in the text. Here are only list of additional data of DynamicDet [36] for comparisons; (2) Pretrained by ImageNet: this includes two methods of using ImageNet for supervised pretrain and self-supervised pretrain; (3) knowledge distillation: a method to perform additional self-distillation after training is completed; and (4) a more complex training process: a combination of steps including pretrained by ImageNet, knowledge distillation, DAMO-YOLO and even additional pretrained large object detection dataset. We show the results in Table 3. From this table, we can see that our proposed YOLOv9 performed better than all other methods. Compared with PPYOLOE+-X trained using ImageNet and Objects365, our method still reduces the number of parameters by 55% and the amount of computation by 11%, and improving 0.4% AP.

Table 4. Comparison of state-of-the-art object detectors with different training settings (sorted by number of parameters).

Model	#Param. (M)	FLOPs (G)	$AP^{val}_{50:95}$ (%)	AP^{val}_{50} (%)	AP^{val}_{75} (%)	AP^{val}_{S} (%)	AP^{val}_{M} (%)	AP^{val}_{L} (%)
YOLOv6-N v3.0 [30] (D)	4.7	11.4	**37.5**	53.1	–	–	–	–
RTMDet-T [44] (I)	4.8	12.6	**41.1**	57.9	–	–	–	–
Gold YOLO-N [61] (D)	5.6	12.1	39.9	55.9	–	–	–	–
YOLOv9-S (S)	7.1	26.4	**46.8**	63.4	50.7	26.6	56.0	64.5
PPYOLOE+-S [74] (C)	7.9	14.4	43.7	60.6	47.9	23.2	46.4	56.9
PPYOLOE-S [74] (I)	7.9	14.4	43.0	60.5	46.6	23.2	46.4	56.9
DAMO YOLO-T [75] (D)	8.5	18.1	43.6	59.4	46.6	23.3	47.4	61.0
RTMDet-S [44] (I)	9.0	25.6	44.6	61.9	–	–	–	–
DAMO YOLO-S [75] (D)	16.3	37.8	**47.7**	63.5	51.1	26.9	51.7	64.9
YOLOv6-S v3.0 [30] (D)	18.5	45.3	45.0	61.8	–	–	–	–
RT DETR-R18 [43] (I)	20	60	46.5	63.8	–	–	–	–
YOLOv9-M (S)	20.0	76.3	**51.4**	68.1	56.1	33.6	57.0	68.0
Gold YOLO-S [61] (C)	21.5	46.0	46.4	63.4	–	–	–	–
Gold YOLO-S [61] (D)	21.5	46.0	46.1	63.3	–	–	–	–
Gold YOLO-S [61] (I)	21.5	46.0	45.5	62.2	–	–	–	–
PPYOLOE+-M [74] (C)	23.4	49.9	49.8	67.1	54.5	31.8	53.9	66.2
PPYOLOE-M [74] (I)	23.4	49.9	49.0	66.5	53.0	28.6	52.9	63.8
RTMDet-M [44] (I)	24.7	78.6	49.4	66.8	–	–	–	–
YOLOv9-C (S)	25.3	102.1	**53.0**	70.2	57.8	36.2	58.5	69.3
DAMO YOLO-M [75] (D)	28.2	61.8	50.4	67.2	55.1	31.6	55.3	67.1
RT DETR-R34 [43] (I)	31	92	48.9	66.8	–	–	–	–
RT DETR-L [43] (I)	32	110	53.0	71.6	57.3	34.6	57.3	71.2
YOLOv9-E (S)	34.7	147.1	**54.5**	71.7	59.2	38.1	59.9	70.3
YOLOv6-M v3.0 [30] (D)	34.9	85.8	50.0	66.9	–	–	–	–
RT DETR-R50M [43] (I)	36	100	51.3	69.6	–	–	–	–
Gold YOLO-M [61] (C)	41.3	57.5	51.1	68.5	–	–	–	–
Gold YOLO-M [61] (D)	41.3	57.5	50.9	68.2	–	–	–	–
Gold YOLO-M [61] (I)	41.3	57.5	50.2	67.5	–	–	–	–
RT DETR-R50 [43] (I)	42	136	53.1	71.3	57.7	34.8	58.0	70.0
DAMO YOLO-L [75] (D)	42.1	97.3	51.9	68.5	56.7	33.3	57.0	67.6
YOLOv9-E (S)	44.0	183.9	**55.1**	72.3	60.7	38.7	60.6	71.4
PPYOLOE+-L [74] (C)	52.2	110.1	52.9	70.1	57.9	35.2	57.5	69.1
PPYOLOE-L [74] (I)	52.2	110.1	51.4	68.9	55.6	31.4	55.3	66.1
RTMDet-L [44] (I)	52.3	160.4	51.5	68.8	–	–	–	–
YOLOR-CSP [66] (C)	52.9	120.4	52.8	71.2	57.6	–	–	–
YOLOv9-E (S)	57.3	189.0	**55.6**	72.8	60.6	40.2	61.0	71.4
YOLOv6-L v3.0 [30] (D)	59.6	150.7	52.8	70.3	–	–	–	–
RT DETR-X [43] (I)	67	234	54.8	73.1	59.4	35.7	59.6	72.9
Gold YOLO-L [61] (C)	75.1	151.7	53.3	70.9	–	–	–	–
Gold YOLO-L [61] (D)	75.1	151.7	53.2	70.5	–	–	–	–
Gold YOLO-L [61] (I)	75.1	151.7	52.3	69.6	–	–	–	–
RT DETR-R101 [43] (I)	76	259	54.3	72.7	58.6	36.0	58.8	72.1
RTMDet-X [44] (I)	94.9	283.4	52.8	70.4	–	–	–	–
YOLOR-CSP-X [66] (C)	96.9	226.8	54.8	73.1	59.7	–	–	–
PPYOLOE+-X [74] (C)	98.4	206.6	54.7	72.0	59.9	37.9	59.3	70.4
PPYOLOE-X [74] (I)	98.4	206.6	52.3	69.5	56.8	35.1	57.0	68.6

[1] (S), (I), (D), (C) indicate train-from-scratch, ImageNet pretrained, knowledge distillation, and complex setting, respectively.

Table 4 shows the performance of all models sorted by parameter size. Our proposed YOLOv9 is Pareto optimal in all models of different sizes. Among them, we found no other method for Pareto optimal in models with more than 20M parameters. The above experimental data shows that our YOLOv9 has excellent parameter usage efficiency.

Shown in Table 5 is the performance of all participating models sorted by the amount of computation. Our proposed YOLOv9 is Pareto optimal in all models with different scales. Among models with more than 60 GFLOPs, only ELAN-based DAMO-YOLO and DETR-based RT DETR can rival the proposed YOLOv9. The above comparison results show that YOLOv9 has the most outstanding performance in the trade-off between computation complexity and accuracy.

Table 5. Comparison of state-of-the-art object detectors with different training settings (sorted by amount of computation).

Model	#Param. (M)	FLOPs (G)	$AP^{val}_{50:95}$ (%)	AP^{val}_{50} (%)	AP^{val}_{75} (%)	AP^{val}_{S} (%)	AP^{val}_{M} (%)	AP^{val}_{L} (%)
YOLOv6-N v3.0 [30] (D)	4.7	11.4	**37.5**	53.1	–	–	–	–
Gold YOLO-N [61] (D)	5.6	12.1	**39.9**	55.9	–	–	–	–
RTMDet-T [44] (I)	4.8	12.6	**41.1**	57.9	–	–	–	–
PPYOLOE+-S [74] (C)	7.9	14.4	**43.7**	60.6	47.9	23.2	46.4	56.9
PPYOLOE-S [74] (I)	7.9	14.4	43.0	60.5	46.6	23.2	46.4	56.9
DAMO YOLO-T [75] (D)	8.5	18.1	43.6	59.4	46.6	23.3	47.4	61.0
RTMDet-S [44] (I)	9.0	25.6	**44.6**	61.9	–	–	–	–
YOLOv9-S (S)	7.1	26.4	**46.8**	63.4	50.7	26.6	56.0	64.5
DAMO YOLO-S [75] (D)	16.3	37.8	**47.7**	63.5	51.1	26.9	51.7	64.9
YOLOv6-S v3.0 [30] (D)	18.5	45.3	45.0	61.8	–	–	–	–
Gold YOLO-S [61] (C)	21.5	46.0	46.4	63.4	–	–	–	–
Gold YOLO-S [61] (D)	21.5	46.0	46.1	63.3	–	–	–	–
Gold YOLO-S [61] (I)	21.5	46.0	45.5	62.2	–	–	–	–
PPYOLOE+-M [74] (C)	23.4	49.9	**49.8**	67.1	54.5	31.8	53.9	66.2
PPYOLOE-M [74] (I)	23.4	49.9	49.0	66.5	53.0	28.6	52.9	63.8
Gold YOLO-M [61] (C)	41.3	57.5	**51.1**	68.5	–	–	–	–
Gold YOLO-M [61] (D)	41.3	57.5	50.9	68.2	–	–	–	–
Gold YOLO-M [61] (I)	41.3	57.5	50.2	67.5	–	–	–	–
RT DETR-R18 [43] (I)	20	60	46.5	63.8	–	–	–	–
DAMO YOLO-M [75] (D)	28.2	61.8	50.4	67.2	55.1	31.6	55.3	67.1
YOLOv9-M (S)	20.0	76.3	**51.4**	68.1	56.1	33.6	57.0	68.0
RTMDet-M [44] (I)	24.7	78.6	49.4	66.8	–	–	–	–
YOLOv6-M v3.0 [30] (D)	34.9	85.8	50.0	66.9	–	–	–	–
RT DETR-R34 [43] (I)	31	92	48.9	66.8	–	–	–	–
DAMO YOLO-L [75] (D)	42.1	97.3	**51.9**	68.5	56.7	33.3	57.0	67.6
RT DETR-R50M [43] (I)	36	100	51.3	69.6	–	–	–	–
YOLOv9-C (S)	25.3	102.1	**53.0**	70.2	57.8	36.2	58.5	69.3
RT DETR-L [43] (I)	32	110	53.0	71.6	57.3	34.6	57.3	71.2
PPYOLOE+-L [74] (C)	52.2	110.1	52.9	70.1	57.9	35.2	57.5	69.1
PPYOLOE-L [74] (I)	52.2	110.1	51.4	68.9	55.6	31.4	55.3	66.1
YOLOR-CSP [66] (C)	52.9	120.4	52.8	71.2	57.6	–	–	–
RT DETR-R50 [43] (I)	42	136	**53.1**	71.3	57.7	34.8	58.0	70.0
YOLOv9-E (S)	34.7	147.1	**54.5**	71.7	59.2	38.1	59.9	70.3
YOLOv6-L v3.0 [30] (D)	59.6	150.7	52.8	70.3	–	–	–	–
Gold YOLO-L [61] (C)	75.1	151.7	53.3	70.9	–	–	–	–
Gold YOLO-L [61] (D)	75.1	151.7	53.2	70.5	–	–	–	–
Gold YOLO-L [61] (I)	75.1	151.7	52.3	69.6	–	–	–	–
RTMDet-L [44] (I)	52.3	160.4	51.5	68.8	–	–	–	–
Dy-YOLOv7 [36] (S)	–	181.7	53.9	72.2	58.7	35.3	57.6	66.4
YOLOv9-E (S)	44.0	183.9	**55.1**	72.3	60.7	38.7	60.6	71.4
YOLOv9-E (S)	57.3	189.0	**55.6**	72.8	60.6	40.2	61.0	71.4
PPYOLOE+-X [74] (C)	98.4	206.6	54.7	72.0	59.9	37.9	59.3	70.4
PPYOLOE-X [74] (I)	98.4	206.6	52.3	69.5	56.8	35.1	57.0	68.6
YOLOR-CSP-X [66] (C)	96.9	226.8	54.8	73.1	59.7	–	–	–
RT DETR-X [43] (I)	67	234	54.8	73.1	59.4	35.7	59.6	72.9
RT DETR-R101 [43] (I)	76	259	54.3	72.7	58.6	36.0	58.8	72.1
RTMDet-X [44] (I)	94.9	283.4	52.8	70.4	–	–	–	–
Dy-YOLOv7-X [36] (S)	–	307.9	55.0	73.2	60.0	36.6	58.7	68.5

[1] (S), (I), (D), (C) indicate train-from-scratch, ImageNet pretrained, knowledge distillation, and complex setting, respectively.