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

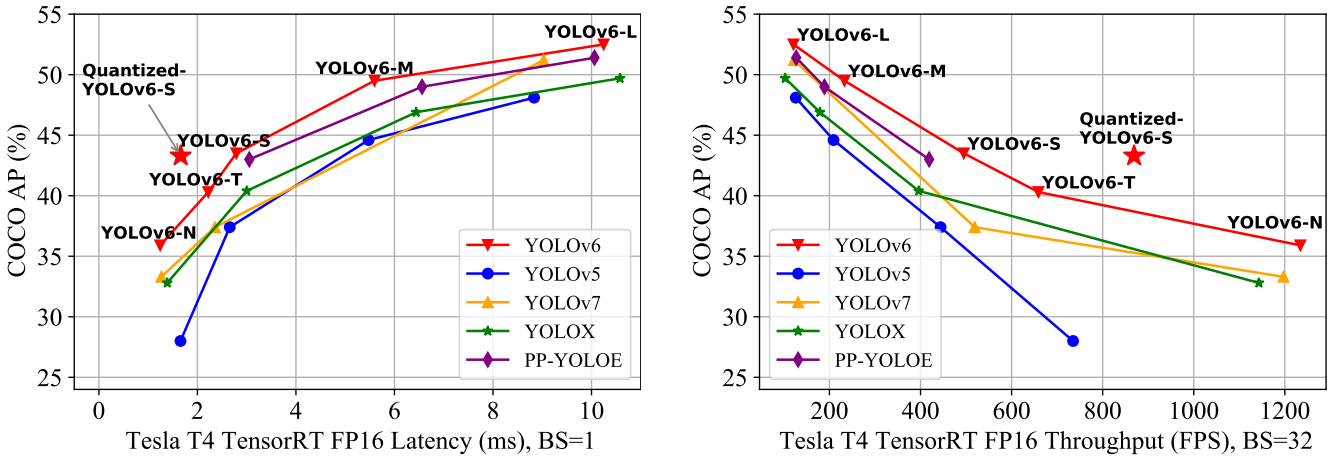


Figure 1: Comparison of state-of-the-art efficient object detectors. Both latency and throughput (at a batch size of 32) are given for a handy reference. All models are test with TensorRT 7 except that the quantized model is with TensorRT 8.

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*,

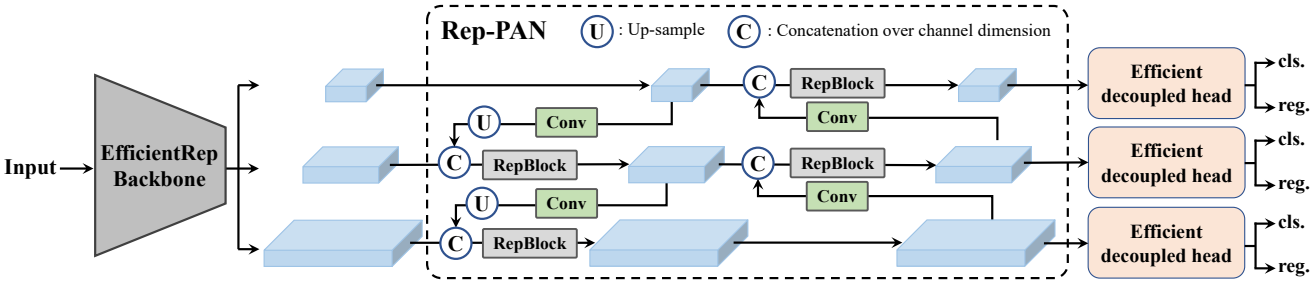


Figure 2: The YOLOv6 framework (N and S are shown). Note for M/L, RepBlocks is replaced with CSPStackRep.

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.

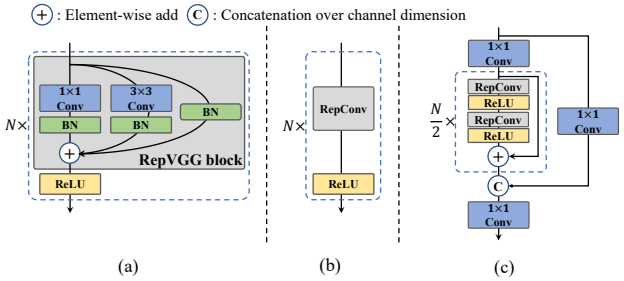


Figure 3: (a) RepBlock is composed of a stack of RepVGG blocks with ReLU activations at training. (b) During inference time, RepVGG block is converted to RepConv. (c) CSPStackRep Block comprises three 1×1 convolutional layers and a stack of sub-blocks of double RepConvs following the ReLU activations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

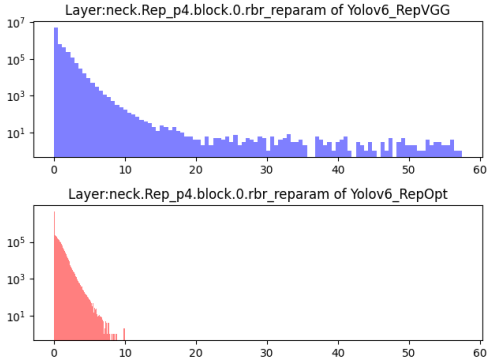


Figure 4: Improved activation distribution of YOLOv6-S trained with RepOptimizer.

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.

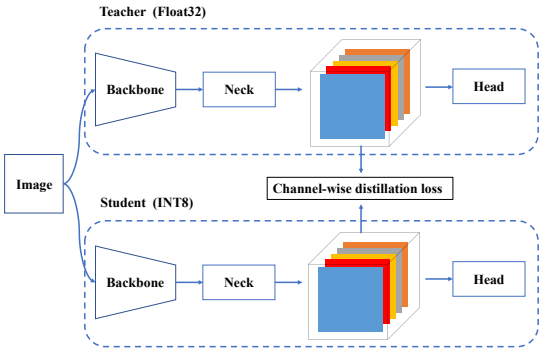


Figure 5: Schematic of YOLOv6 channel-wise distillation in QAT.

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

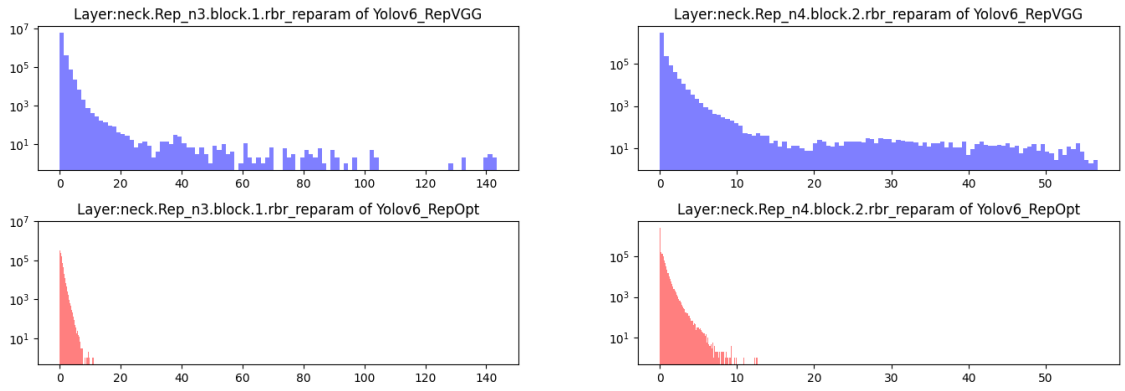
See Fig. 7, we observe that SNR and Cosine similarity gives highly correlated results. However, directly evaluating AP produces a different panorama. Nevertheless, in terms of final quantization performance, MSE is the closest to direct AP evaluation, see Table 21.

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.

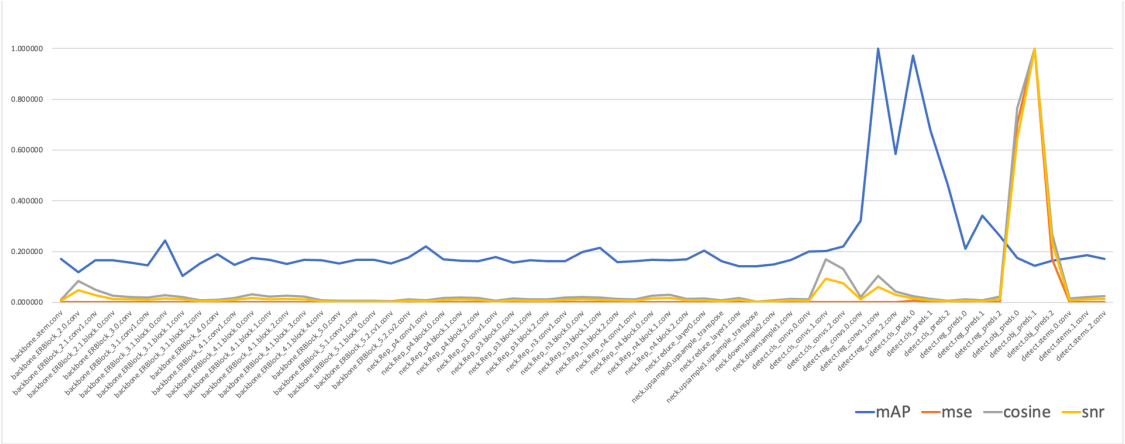
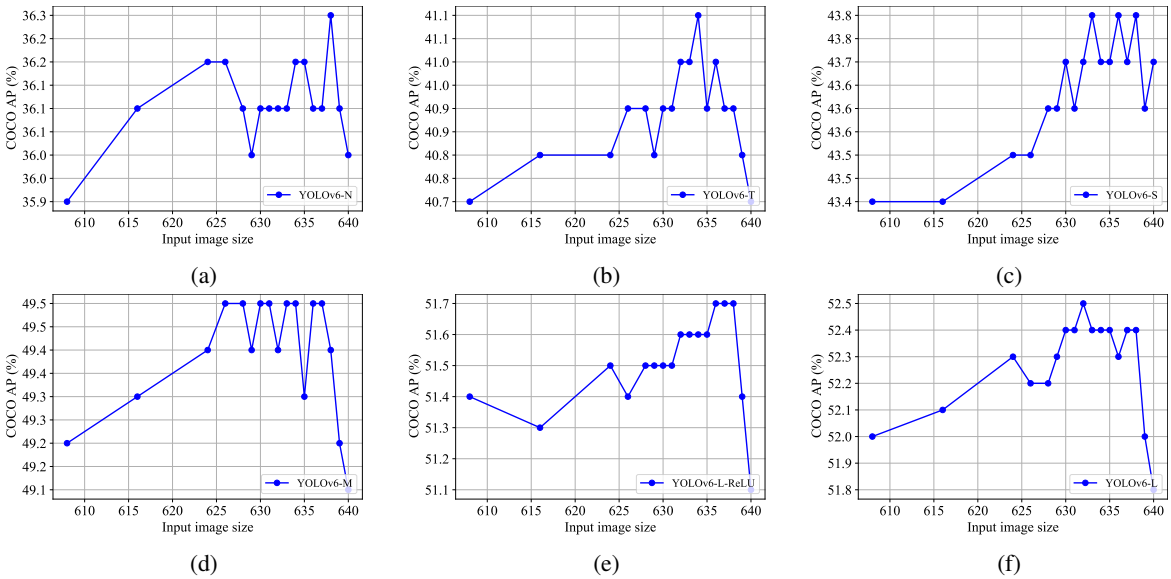


Figure 7: Quantization sensitivity analysis of all layers in YOLOv6s trained with RepOptimizer.



(a) (b) (c)

(d) (e) (f)

Figure 8: Analysis of the gray border problem.