ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240490

BOB-YOLO: Balancing Optimization Binarized YOLO via Module-Wise Latency

Xinyu Liu[a,b,1], Wenqiang Zhou[a,b,1], Zhendong Yu[a,b], Jiaming Yang[a,b], Tao Wang[a,b], Chenwei Tang[a,b,*] and Jiancheng Lv[a,b,*]

[a]College of Computer Science, Sichuan University, Chengdu, 610065, P. R. China
[b]Engineering Research Center of Machine Learning and Industry Intelligence, Ministry of Education, Chengdu 610065, P. R. China

Abstract. When it comes to object detection tasks, YOLO stands out for its impressive speed and efficiency. Nonetheless, deploying YOLO on resource-constrained devices remains a challenge due to its substantial model size and memory requirements. The direct application of conventional binary quantization strategies to YOLO can result in significant accuracy degradation. A prevalent solution is to introduce floating-point shortcuts. However, the increased computational demand and parameter complexity associated with these shortcuts limit their practical deployment on hardware platforms for optimal acceleration. To solve this problem, we propose a binary neural network (BNN) for object detection called BOB-YOLO to achieve a balanced performance in terms of computational speed, model size, and detection accuracy. Our BOB-YOLO fully leverages module-wise latency (MWL) to supervise the latency of floating-point shortcut branches by that of 1-bit trunk branches. This supervision maximizes the information carried by the floating-point data flow in shortcuts while maintaining latency within the limits set by the 1-bit convolution branch, thereby improving parallel computational efficiency. We also introduce the Roofline Model to address these limitations by considering both computational complexity and parameter compression, ensuring high computational intensity. Additionally, we propose a performance evaluation metric P_d, which provides an intuitive description of the trade-off between speed and accuracy, aligning closely with the practical requirements of binary quantization strategies. Extensive experiments on the VOC and COCO datasets demonstrate the significant advantages of our method over state-of-the-art BNN methods.

1 Introduction

Real-time object detection [4] is a crucial task in the field of computer vision, finding extensive applications in various domains such as autonomous driving [2], smart camera [18], and industrial quality inspection [12]. Often, the scenarios for real-time target detection applications are edge computing environments, where computational or storage resources are limited. This necessitates specific demands on model inference speed, storage overhead, and accuracy, highlighting the clear prospects and tangible value of model lightweight techniques in this domain [23]. Recent advancements in 1-bit neural networks, i.e., BNNs, have made quantization technology an efficient and promising method for deep model compression in resource-limited devices [3]. BNNs replace complex convolution with equivalent XNOR and popcount operations, achieving up to 32× memory compression and 58× computational reduction on CPUs [13]. Further research on the behaviors of BNNs in object detection is crucial to extend their usage in academia, and explore their potential in real-world applications [19].

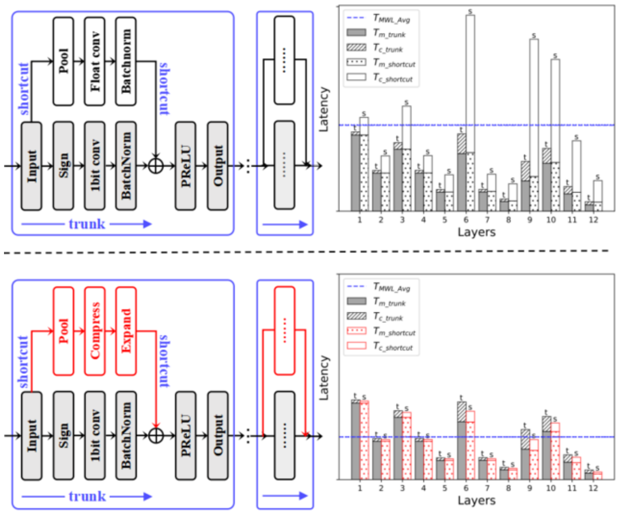


Figure 1. Illustration of the motivation behind MWL (top: commonly used methods, bottom: our method based on MWL). In parallel computations between the 1-bit trunk and floating-point shortcut branches, waiting times occur. Hence, we argue that the current design and evaluation of BNNs may not be optimal for achieving balanced optimization in binarized YOLO.

In recent years, several Binary Neural Network (BNN) algorithms for object detection have been proposed[21, 24]. However, there is a lack of research specifically targeting YOLO. Most studies on binary quantization algorithms for object detection have been binarizing SSD [8] and Faster-RCNN [17], which are based on overparameterized convolutional modules such as VGG and ResNet. In contrast, the latest iteration YOLOv8 architecture outperformed other object detection models in terms of both accuracy and processing speed. It incorporates several innovative features such as massive cross-scale connections, an anchor-less design, a more complex gra-

[*] Corresponding Author.
Email: tangchenwei@scu.edu.cn(C. Tang), lvjiancheng@scu.edu.cn.(J. Lv)
[1] Equal contribution.

dient flow C2f (CSPLayer 2Conv) structure, and a Decoupled-Head design, further enhancing the network's performance, making it an ideal choice for real-time applications. Thus, we chose YOLOv8 as the baseline object detection model.

This paper aims to design a 1-bit YOLO detector that achieves a balanced trade-off between computational speed, model size, and detection accuracy. To achieve this objective, two fundamental questions need to be addressed. The first question is: *Is the well-performing YOLO series still suitable for BNNs?* Directly applying 1-bit quantization to YOLO leads to a substantial loss of information flow, resulting in an unacceptable decrease in detection accuracy. Its poor performance is caused by the low representational capacity. the floating-point shortcuts proposed in Bi-Real Net[10] help enhance the representation of binary modules and mitigate this information flow degradation. However, as shown in Figure 1, we empirically observe that the mixed-precision layers exhibit varying performance in terms of computation and memory access costs. And, these floating-point shortcuts come at the cost of increased computational and parameter complexity, both of which hampers the potential for effective acceleration. Moreover, the existing evaluation metric based on model-wise FLOPs only considers the time cost of sequential calculations and fails to account for the waiting delay introduced by massive parallel computing modules and memory access overhead. Previous models have achieved good results in accelerated computation, but the degree of memory access cost compression is far less than the amount of computation, which leads to a decrease in the operational intensity, and thus can not achieve the ideal acceleration effect in the actual hardware deployment.

In this paper, we address the first question by using the MWL and Roofline Model to guide the design of binary quantization strategies. Based on the Roofline Model, we control the compression of computation and access to a close degree to ensure a high computation intensity. Besides, we propose the MWL to focus on individual modules within the network and account for synchronization and waiting times in parallel computations involving trunk and shortcut branches. By separately measuring the computation and memory access costs of different branches, we calculate the actual delay of a module and account for the time spent waiting for synchronization. Building upon the MWL concept, we utilize the latency of the trunk branch composed of 1-bit convolutions to supervise the latency of shortcuts employing floating-point operations. This supervision ensures that the information carried by the floating-point data flow in the shortcut is maximized to recover the information loss from quantization, while still guaranteeing that the shortcut's latency remains within the limits set by the 1-bit convolution branch. Furthermore, this approach increases the operational intensity of both individual modules and the overall model, ultimately maximizing hardware performance.

Indeed, the inclusion of floating-point operations in BNNs can significantly mitigate the decrease in accuracy. However, these BNNs trade speed for accuracy by incorporating complex computations during training or inference. This brings a new question: *How can we evaluate whether a BNN achieves a trade-off between accuracy and speed using an intuitive quantization metric?* Existing BNNs employ metrics such as FLOPs, model size, and single-inference accuracy on the validation sets to individually assess the performance of quantization strategies. As we know, the goal of BNNs is to minimize FLOPs and model size while maximizing accuracy. These conflicting metrics make it challenging to measure the balance between accuracy and speed. Moreover, real-world application scenarios for BNNs have diverse requirements for speed improvements and tolerances for accuracy degradation. It is natural to believe that if we can find a metric to evaluate the trade-off between speed and accuracy in BNNs would provide more convincing practical applications for 1-bit quantization. Specifically, to address the second question, we propose a performance evaluation metric denoted as P_d, which is based on the requirements of real-time object detection algorithms in real-world scenarios. This metric, P_d, represents the probability of detecting a target within a limited time and is influenced by the computational speed, memory access speed, and detection accuracy of the 1-bit detector. Our contributions are summarized as follows:

- We propose BOB-YOLO as a new BNN framework for YOLO in order to realize performance-balanced quantization in object detection models, which has achieved 76.1% mAP on the VOC dataset and 52.1% mAP on the COCO dataset. It also delivers a remarkable computation acceleration of $28.36\times$ and storage savings of $20.43\times$, significantly surpassing the state-of-the-art 1-bit detectors.
- To the best of our knowledge, we are the first to leverage the MWL to supervise the floating-point shortcuts by the latency of 1-bit trunks, which preserves the speed advantage of 1-bit convolution and increase the operational intensity in the overall model. This strategy also maximizes hardware performance and utilizes the full potential of underlying hardware resources.
- The BOB-YOLO utilises a Roofline Model to ensure high computational intensity. Additionally, a new performance metric, P_d, is proposed to provide a more intuitive description of the trade-off between speed and accuracy. This metric is closely related to the practical requirements of binary quantization strategies.

2 Background

Real-time Object Detection. Real-time object detection algorithms gain significant attention in computer vision due to their extensive applications across various scenarios [20]. Existing algorithms can generally be classified into two types: two-stage and single-stage approaches. The two-stage methods, such as the RCNN series, utilize a region proposal network (RPN) to generate candidate regions and then perform classification on selected regions [17, 6]. Common single-stage methods like SSD [8] and YOLO [16] directly predict across the entire image, allowing for simultaneous localization and classification within a single network. Given the requirement for real-time detection in object detection applications, single-stage methods have become increasingly popular due to their efficiency. Among these methods, the YOLO series has gained significant traction for its outstanding real-time detection capabilities and high accuracy [14, 15, 1, 5]. The latest iteration, YOLOv8[2], incorporates several innovative features such as massive cross-scale connections, an anchor-less design, a more complex gradient flow C2f (CSPLayer_2Conv) structure, and a Decoupled-Head design, further enhancing the network's performance.

Binary Neural Networks. Due to the nearly $32\times$ compression, BNNs experience a catastrophic drop in model accuracy. Consequently, the research trajectory of BNNs has primarily focused on continually enhancing model accuracy. BinaryNet introduced a comprehensive training framework [26]. XNOR-Net proposed the use of scaling factors to compensate for the quantization error arising from weight binarization [13]. Bi-Real Net incorporated floating-point shortcuts both before and after 1-bit convolutions to amplify information representation [10]. ReActNet introduced the RSign and

[2] https://github.com/ultralytics/ultralytics

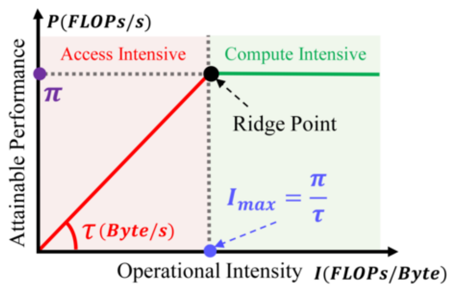


Figure 2. Illustration of the Roofline Model.

RPReLU functions to adjust the activation value distribution before and after convolutions [11]. BiDet employed the information bottleneck principle to eliminate redundant information in higher-level feature maps, thereby boosting recognition accuracy [21]. LWS-Det introduces angular and amplitude loss functions to increase detector capacity, further narrowing the gap with the real-value detector [25]. However, when applying BNN to YOLO, these tricks still fail to achieve balanced performance in detection accuracy, speed, and model compression due to massive cross-scale connections, upsample and down-sample operations.

Roofline Model. The Roofline Model [22] aims to propose an easy-to-understand, visual performance model that offers insights to programmers and architects on improving parallel software and hardware for floating point computations. By comparing computational performance with hardware resource utilization, the Roofline Model provides an intuitive way to analyze performance bottlenecks in computational tasks. As shown in Figure 2, the horizontal and vertical axes are operational intensity (FLOPs/Byte) and attainable performance (FLOPs/s). Given the operational intensity I, hardware memory bandwidth τ, and hardware computational capacity π, the attainable performance P can be calculated as:

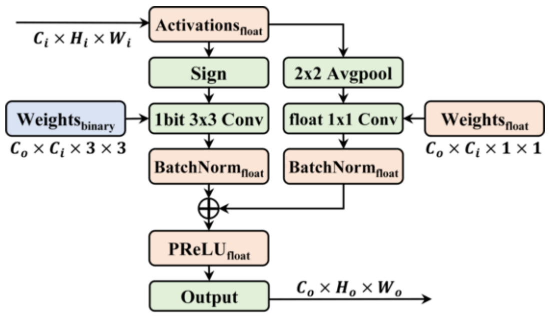
$$P = \min(I \times \tau, \pi). \tag{1}$$

For modules with high computing intensity, their attainable performance is determined by the computing power of the hardware platform and the overall operations of the model. Within the range of hardware computing capabilities, a higher operational intensity of the model leads to increased memory usage rates and correspondingly higher theoretical performance. However, it is worth mentioning that the inference speed is not necessarily faster with a smaller model calculation. Given the memory access MA and the total operations OP, the actual inference time T_{infer} of the operator is as follows:

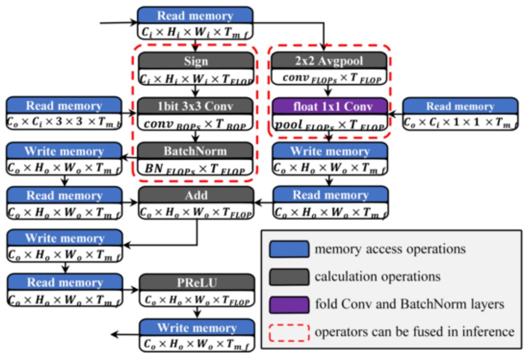
$$T_{infer} = \begin{cases} MA/\tau, & \text{Access Intensive,} \\ OP/\pi, & \text{Compute Intensive.} \end{cases} \tag{2}$$

3 Methodology: shortcut structure based MWL

Existing object detection models are generally constructed by stacking repetitive modules. When devising quantization strategies, parallel computation methods, and computational graph scheduling, a module-wise approach is usually adopted. Building upon this, supplementing with an evaluation of a module's computational efficiency can provide a more comprehensive insight into the model's performance. Taking the most basic module of BNN shown in Figure 3(a) as an example, we provide a detailed introduction to the proposed MWL. First, as shown in Figure 3(b), we transform the basic module into the form of a flowchart including computational and memory access processes.



(a) most basic module of BNN



Read memory
$C_i \times H_i \times W_i \times T_{m\,f}$
Sign
$C_i \times H_i \times W_i \times T_{FLOP}$
2x2 Avgpool
$conv_{FLOPs} \times T_{FLOP}$
Read memory
$C_o \times C_i \times 3 \times 3 \times T_{m\,b}$
1bit 3x3 Conv
$conv_{BOPs} \times T_{BOP}$
float 1x1 Conv
$pool_{FLOPs} \times T_{FLOP}$
Read memory
$C_o \times C_i \times 1 \times 1 \times T_{m\,f}$
Write memory
$C_o \times H_o \times W_o \times T_{m\,f}$
BatchNorm
$BN_{FLOPs} \times T_{FLOP}$
Write memory
$C_o \times H_o \times W_o \times T_{m\,f}$
Read memory
$C_o \times H_o \times W_o \times T_{m\,f}$
Add
$C_o \times H_o \times W_o \times T_{FLOP}$
Read memory
$C_o \times H_o \times W_o \times T_{m\,f}$
Write memory
$C_o \times H_o \times W_o \times T_{m\,f}$
Read memory
$C_o \times H_o \times W_o \times T_{m\,f}$
PReLU
$C_o \times H_o \times W_o \times T_{FLOP}$
Write memory
$C_o \times H_o \times W_o \times T_{m\,f}$
memory access operations
calculation operations
fold Conv and BatchNorm layers
operators can be fused in inference

(b) basic module of BNN in the form of flowchart including computational and memory access processes

Figure 3. The most basic module of BNN and its flowchart form.

Let B denote the number of bytes occupied by the element, then the time T_m taken to read or write an element can be calculated as:

$$T_m = B/\tau. \tag{3}$$

After that, let TB and TF denote the time required for a fixed-bit width integer operation and a floating-point operation, respectively. Nvidia[3] reveals that the hardware computational performance is proportional to the numerical bit-width. The relationship between TB and TF can be expressed as:

$$TB = \frac{1}{32} \times \frac{BW_w \times BW_a}{BW_w + BW_a} \times TF, \tag{4}$$

where BW_w and BW_a represent the bit widths of the weights and activation values, respectively.

Next, we analyze the computation of different layers. As shown in Figure 3(a), the C_i, W_i, H_i, C_o, W_o, H_o denote the channel, width, height of the input feature map and the output feature map, respectively. We can easily determine the computational cost associated with various operators, including convolutional layers, average pooling layers, BatchNorm layers layers, and activation functions.

Let's now compute the MWL for this basic module of BNN. As shown in Figure 3(b), considering the parallel execution of two branches from the node reading input data to the Add node, the time

[3] https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

cost between these two nodes depends on the slower branch. The time costs of the left branch TL and right branch TR are:

$$\begin{aligned} TL =& \frac{9}{8\tau}C_iC_o + \frac{4}{\tau}C_iH_iW_i + \frac{8}{\tau}C_oH_oW_o \\ &+ \left(\frac{9}{64}C_i + 3\right)C_oH_oW_oTF \\ &+ C_iH_iW_iTF, \end{aligned} \tag{5}$$

$$\begin{aligned} TR =& \frac{4}{\tau}C_iC_o + \frac{4}{\tau}C_iH_iW_i + \frac{8}{\tau}C_oH_oW_o \\ &+ \left(\frac{9}{64}C_i + 3\right)C_oH_oW_oTF \\ &+ (C_oC_i + 4C_i + C_o)H_oW_oTF. \end{aligned} \tag{6}$$

Therefore, the total latency of the basic BNN module with 1-bit trunk branch and floating-point shortcut branch can be calculated as:

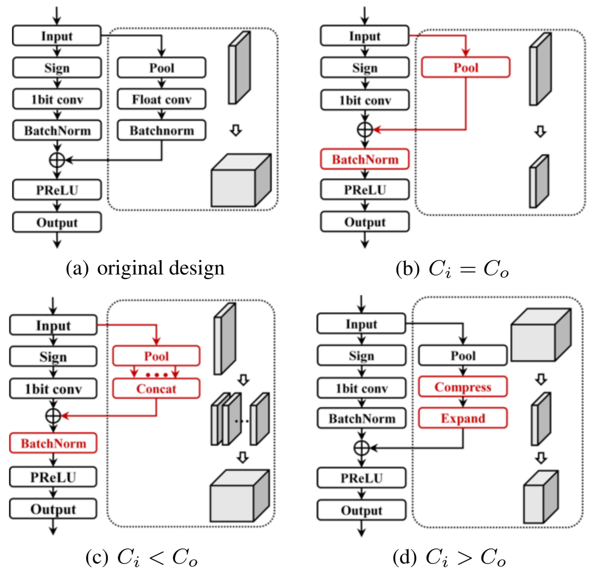
$$MWL = \max(TL, TR) + (TF + \frac{12}{\tau}) \times C_oH_oW_o. \tag{7}$$

Based on the above calculation process, we examined the corresponding convolutional layers within the backbone network of typical object detection models to ascertain whether the computation in the right branch hinders the acceleration effect of the left branch. As shown in Table 1, it is evident that when introducing high-bit-width convolutions in the right branch, the acceleration caused by quantifying the left branch becomes invalid due to the increase in waiting time. Therefore, the structure of cross-layer connections and the mixing of different precisions have an impact on the MWL. It is natural to believe that optimizing this aspect in module design will significantly improve the performance of the model.

Table 1. Illustration of the time overhead of the left and right branches after quantization in convolution modules with various types of inputs and outputs ($t_m = 1/\tau$). The sizes of these inputs and outputs are taken from commonly used object detection networks like SSD, YOLO, etc. For cases where the input and output channels C are the same, the time costs of right branch TR doesn't account for the time of convolution computation and memory. For cases where the height H and width W of input and output are the same, the TR excludes the time for average pooling.

Input-Output ($[\boldsymbol{C}, \boldsymbol{H}, \boldsymbol{W}]$)		**Time Costs** ($\mathbf{10^6}$)
[64, 320, 320]-[128, 160, 160]	**TL**	$52.43t_m + 45.87TF$
	TR	$52.46t_m + \mathbf{219.54}TF$
[128, 160, 160]-[256, 80, 80]	**TL**	$26.25t_m + 37.68TF$
	TR	$26.34t_m + \mathbf{214.63}TF$
[256, 80, 80]-[512, 40, 40]	**TL**	$13.25t_m + 33.58TF$
	TR	$13.63t_m + \mathbf{211.17}TF$
[64, 150, 150]-[128, 150, 150]	**TL**	$28.81t_m + 36.00TF$
	TR	$28.83t_m + \mathbf{187.20}TF$
[128, 75, 75]-[256, 75, 75]	**TL**	$14.43t_m + 30.96TF$
	TR	$14.53t_m + \mathbf{185.76}TF$
[256, 38, 38]-[512, 38, 38]	**TL**	$7.54t_m + 29.20TF$
	TR	$7.91t_m + \mathbf{190.00}TF$
[256, 80, 80]-[256, 40, 40]	**TL**	$9.90t_m + 17.61TF$
	TR	$9.83t_m + 2.45TF$
[512, 40, 40]-[512, 20, 20]	**TL**	$5.21t_m + 16.18TF$
	TR	$4.91t_m + 8.64TF$

To construct the binarized YOLO detector that achieves the optimal trade-off between detection accuracy, speed, and model size, building upon the concept of MWL, we adhere to two principles when designing the binary convolution module for the binarized YOLOv8: 1) compression of the delay on the right branch (floating-point shortcut branch) to match the delay on the left branch (1-bit trunk branch), maximizing the speed advantage of 1-bit convolution, 2) retention of the remaining connections on the right branch to ensure a continuous flow of floating-point data within the network, compensating for significant information loss caused by binary quantization.



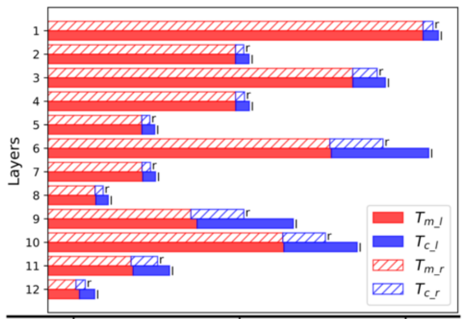
(a) original design (b) $C_i = C_o$
(c) $C_i < C_o$ (d) $C_i > C_o$

Figure 4. Illustration of module designs for different input and output feature maps, where C_i and C_o are the channels of input and output. The 'Compress' and 'Expand' are both operations on the combination of convolution and $BatchNorm$. The dashed boxes depict the feature map size changes under different shortcuts.

Figure 4 illustrates the different strategies we employed in designing the binary convolution, considering various input and output feature map sizes. Specifically, Figure 4(a) illustrates the original module design, where we use pooling and convolution layers to align the input and output sizes. However, as suggested in Table 1, this strategy can negate the acceleration advantages of 1-bit convolution. Figure 4(b) represents the case where the input and output channel numbers are the same. Table 1 indicates that the delay introduced by the pooling layer does not affect the speed of the left branch. Furthermore, in this case, the time delay for computations on the left branch is much greater than that on the right branch. To further enhance the efficiency of parallel computation and increase the overall module's operational intensity, we move the memory-intensive operator $BatchNorm$ after the shortcut connection. Figure 4(c) illustrates the case where the input channel is less than the output channel. In this case, the input feature map is stacked to align with the output feature map, which only increases the time spent on memory read/write operations. Additionally, to maintain a more stable activation distribution, batch normalization calculations are moved after the connection. Figure 4(d) illustrates the case where the input channel is greater than the output channel, which most frequently occurs. Here, channel alignment is achieved by performing two 1×1 convolution operations with batch normalization as follows:

$$\begin{aligned} Comp.(x') &= BN_C(Conv_{C_i \times C \times 1 \times 1}(x)), \\ Expa.(x'') &= BN_{C_0}(Conv_{C \times C_0 \times 1 \times 1}(x')). \end{aligned} \tag{8}$$

The first convolution compresses ($Comp.(\cdot)$) the channel count from C_i to C and the second one expands ($Expa.(\cdot)$) feature maps with



Layers	C_i	H_i	W_i	C_o	H_o	W_o	C
1	64	320	320	128	160	160	4
2	128	160	160	64	160	160	4
3	160	160	160	128	160	160	8
4	128	160	160	256	80	80	8
5	256	80	80	128	80	80	8
6	1024	80	80	256	80	80	16
7	256	80	80	512	40	40	16
8	512	40	40	256	40	40	16
9	2048	40	40	512	40	40	32
10	768	80	80	256	80	80	16
11	768	40	40	512	40	40	32
12	1280	20	20	512	20	20	32

Figure 5. Illustration of the latency in the left and right branches of the typical layers in our model.

C_o output channels. This approach reduces the computational load and access overhead compared to a single convolution, as shown below:

$$\begin{aligned}&OPs(Conv) = 2 \times (C_i \times C_o)HW,\\&OPs(Comp. + Expa.) = 2 \times (C_i + C_o)HWC,\\&Params(Conv) = C_i \times C_o,\\&Params(Comp. + Expa.) = (C_i + C_o) \times C.\end{aligned} \tag{9}$$

The selection of C in our model is determined by the delay on the left branch and optimized based on the Roofline Model (which will be detailed in section 4.1) for binary quantization. This ensures that the acceleration on the left branch remains effective while maximizing the information flow on the right branch. Simultaneously, it enhances the overall computing intensity of the module. Figure 5 demonstrates the choice of C in our model, where T_{m_l}, T_{c_l}, T_{m_r}, and T_{c_r} represent the memory and computational costs in the left and right branches, respectively. From the Figure 5, we can find that the latency in the 1-bit trunk (left branch) and the floating-point shortcut (right branch) are relatively similar, which demonstrates that our method can compress the computational and parameter quantities to the same extent. A more detailed analysis will be presented in the experiments (section 5).

4 Comprehensive consideration for Binary Quantization

4.1 Roofline Model for Binary Quantization

In BNNs, the operations primarily include 1-bit convolution, floating-point convolution, activation functions, batch normalization, and so on. The memory access cost mainly consists of reading parameters and reading/writing intermediate feature maps. Thus, we calculate the operational intensity I_{BNN} for Binary Quantization based on the Roofline Model as follows:

$$I_{BNN} = \frac{BOP_{conv} + FLOP_{all}}{M_{act} + M_{weight}}, \tag{10}$$

where BOP_{conv} and $FLOP_{all}$ denote the binary operations in 1-bit convolution and all floating point operations including convolution, activation functions, batch normalization, and so on. The M_{act} and M_{weight} denote access to floating point activation values and binary weights. It is important to note that when employing binary quantization to compress network structures, we pay attention to the changes in computation and memory access brought about by lightweight module designs. Increasing the operational intensity before reaching the platform's limit is beneficial for accelerating model inference. However, reducing the overall number of operations only becomes effective in speeding up the inference process when the computational intensity exceeds the upper limit. While the convolution operations in BNNs are nearly 64× compressed [10], the utilization of floating-point activation values does not reduce the memory access overhead, leading to an overall decrease in the model's operational intensity. Therefore, during the design phase, it is crucial to focus on enhancing the computational intensity to address this issue. We calculate the objective function of the Roofline Model for Binary Quantization as follows:

$$\begin{cases}\max I_{BNN}, & if \quad I_{BNN} < \pi/\tau,\\ \min(BOP_{conv} + FLOP_{all}), & if \quad I_{BNN} \geq \pi/\tau.\end{cases} \tag{11}$$

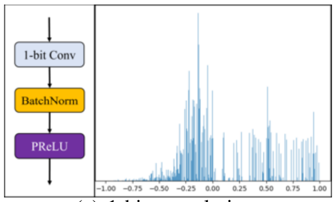
In the deployment of BNNs, the Roofline model serves as a crucial tool for supervision and performance assessment. This model quantitatively evaluates the compatibility between the computational demands of BNN algorithm and the processing capabilities of hardware. Specifically, if the computational density required by the algorithm exceeds the threshold capabilities of the hardware, it may limit the algorithm's ability to achieve its potential performance within a given time frame. Conversely, if the computational density of the algorithm is below the maximum processing capacity of the hardware, it results in underutilization of computational resources, thereby impacting the overall energy efficiency ratio. Therefore, the Roofline model not only guides hardware selection but also promotes the optimization of algorithm design to ensure optimal synergy between hardware and algorithm.

4.2 Novel Metric P_d for Balancing Optimization

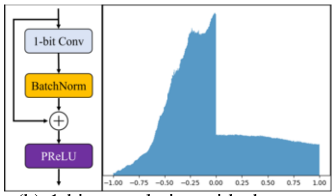
To evaluate the performance of quantized models, it is necessary to consider the real-world requirements of using these models and strike a balance between efficiency and accuracy. When it comes to real-time object detection algorithms, evaluation metrics should reflect the users' experience. Considering the trade-off between efficiency and accuracy, evaluation metrics should comprehensively consider factors such as inference computation, memory access overhead, and accuracy of the model. Based on this, we propose the evaluation metric P_d to assess the balanced performance of quantized models:

$$P_d = 1 - (1 - Acc)^{\lfloor \frac{tl}{l} \rfloor}, \tag{12}$$

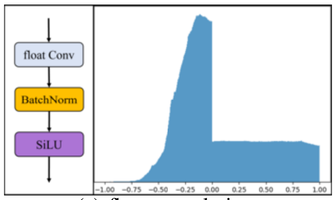
where Acc represents the probability of successfully detecting a target in a single inference. In the experimental process, as the distribution of the validation set approaches the real distribution, this probability closely approximates the model's accuracy on the validation set. The variable tl represents the time limit, while l represents



(a) 1-bit convolution



(b) 1-bit convolution with shortcut



(c) float convolution

Figure 6. Numerical distribution of 1-bit 3 × 3 convolution (a), 1-bit 3 × 3 convolution with float shortcut(b), and float 3×3 convolution (c) in the first layer.

the model's inference latency. P_d represents the probability that the model can detect the target within the specified time limit. From a practical application perspective, P_d should be as close to 100% as possible. This ensures that the model can consistently complete the task within the specified time, meeting the requirements of real-time applications.

Table 2. Comparison of mean average precision (mAP), model size, and FLOPs with the SOTA BNNs on VOC, where trunk means a simple 1-bit convolutional module without shortcut and LearnableBias, shortcut* means shortcut based on MWL.

Framework	Quantization Method	W/A (bit)	$mAP@0.5$ $VOC(\%)\uparrow$	FLOPs (B)↓	Model size (MB)↓
SSD300 VGG-16	Full value	32/32	74.3	31.44	105.16
	ReAct-Net	1/1	68.4	2.73	21.88
	BiDet	1/1	66.0	2.73	21.88
	LWS-Det	1/1	71.4	2.73	21.88
YOLOv8l	Full value	32/32	83.0	82.53	166.3
	trunk	1/1	39.7	1.81	5.81
	trunk + shortcut*	1/1	56.1	2.91	8.14
	trunk + LearnedBias	1/1	69.8	1.81	5.81
	trunk + LearnedBias + shortcut	1/1	79.3	15.23	31.29
	ours(trunk + LearnedBias + shortcut*)	**1/1**	**76.1**	**2.91**	**8.14**
YOLOv8m	Full value	32/32	81.7	39.43	99.4
	trunk	1/1	32.8	1.00	3.32
	trunk + shortcut*	1/1	56.3	1.60	5.19
	trunk + LearnedBias	1/1	67.9	1.00	3.32
	ours(trunk + LearnedBias + shortcut*)	**1/1**	**71.4**	**1.60**	**5.19**

5 Experiments

We conduct extensive experiments on representative datasets: VOC [4] and COCO datasets [7]. The VOC dataset contains 20 classes with a total of 16,551 training images and 4,952 validation images. The COCO dataset consists of images from 80 categories with a total of 118,287 training images and 5,000 validation images. The model is implemented in PyTorch 2.0.1 and trained on NVIDIA GeForce RTX 3090 GPUs. The input image size is standardized to 640 × 640 on both the VOC and COCO datasets. Following [9], we divided the dataset into training and validation sets, and employed the Cross-Validation to determine the hyperparameter settings. We utilize the SGD optimizer with a momentum of 0.937, an initial learning rate of 0.01, a final learning rate of 0.0001, and a weight decay of 0.0005. [4]

5.1 Comparative Results

In this section, we conduct comparative studies to analyze the significance of shortcuts and LearnableBias in the BNN module. Table 2 illustrates the comparison of computation complexity, storage cost, and the mAP. It is evident that LearnableBias substantially enhances the flexibility and generalization ability of the model, achieving a 30.1% and 35.1% increase in the mAP for the Yolov8l and Yolov8m models, respectively. Similarly, shortcuts effectively enhance the model's expressiveness. To better illustrate this, We visualize the numerical distributions of feature maps during the YOLOv8 training process as shown in Figure 6. Compared to a full-precision convolutional module, a simple 1-bit convolutional module has extremely weak representational capability. The use of a float shortcut can effectively mitigate this issue, achieving a representational effect comparable to that of a full-precision convolutional module. However, the 40% additional model size and 60% computational overhead introduced by shortcuts are non-negligible, which reinforces the importance of our work.

In addition, we compare our method with ReAct-net [11], BiDet [21] and LWS-Det [25] in the task of object detection. Based on the MWL and Roofline Model, our method reduces the latency of floating-point shortcuts while increasing their computational density. We used Xilinx's Vitis HLS software toolset to develop, test, and document the execution latency of the shortcut section before and after optimization. Test results showed that the optimized shortcut reduced the execution latency by 63.5%. This allows us to mitigate the impact of a large number of up-sampling, down-sampling, and cross-layer connections in YOLO, which would otherwise increase the number of floating-point shortcuts. As a result, the model's parameter count and computational complexity did not increase significantly. Compared to directly applying binary quantization methods to YOLO, our approach achieved a 5.23× reduction in FLOPs and a 3.84× reduction in model size, while maintaining a marginal ac-

Table 3. Comparison of $mAP@0.5$ and $mAP@[.5, .95]$ with the SOTA BNNs on COCO.

Framework	Quantization	W/A	$mAP@0.5\uparrow$	$mAP@[.5,.95]\uparrow$
SSD300 VGG-16	Full value	32/32	41.2	23.2
	ReAct-Net	1/1	30.0	15.3
	BiDet	1/1	28.3	13.2
	LWS-Det	1/1	32.9	17.1
YOLOv8l	Full value	32/32	65.7	49.3
	ours	**1/1**	**52.1**	**33.7**

Table 4. Parameters, Multiply and add operation (MACs) of full-value convolutions in BNNs using SSD and YOLO framework.

BNN_{SSD}			$BNN_{YOLO(ours)}$		
layers	Params(k)	MACs(M)	layers	Params(k)	MACs(M)
1	1.73	155.52	1	1.73	176.95
2	32.77	184.32	2	2.56	65.54
3	262.14	94.63	3	16.38	104.86
	...			...	
16	442.41	159.71	34	5.14	8.22
17	290.43	7.26	35	8.19	3.28
18	193.62	0.19	45	5.14	2.06
Total	5361.71	2630.82	Total	626.87	1260.71

[4] Code is available at https://github.com/534-quant/BOB-YOLO.

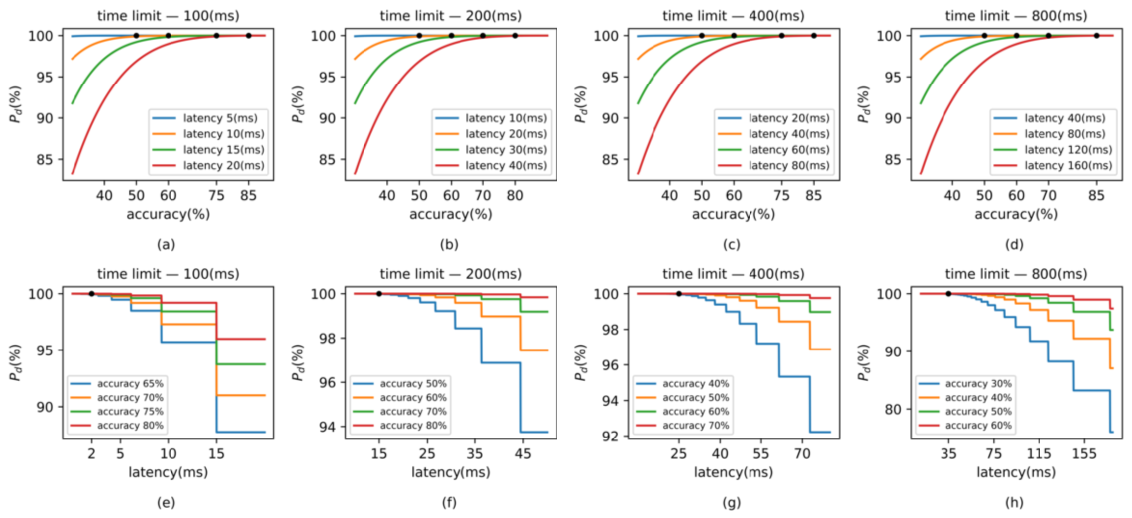


Figure 7. Illustration of the tradeoff between efficiency and accuracy as reflected by P_d under different time limits.

curacy difference of only 3.2%. Compared to full-value models, our method significantly accelerates the computation and saves the storage by $28.36\times$ and $20.43\times$ with the yolov8l detector and $24.64\times$ and $19.15\times$ with the yolov8m detector. Compared with SOTA LWS-Net, our method demonstrates a 4.7% increase in mAP. Remarkably, despite similar FLOPs, our model size is less than half that of LWS-Net. Moreover, our method based on YOLOv8m achieves the same mAP as LWS-Net but with a significantly smaller model size, only a quarter of the size, and nearly half the FLOPs. The COCO dataset is much more challenging for object detection than VOC dataset due to the high diversity and large scale. Table 3 demonstrates the mAP under different IOU thresholds. Compared with SOTA LWS-Det, our method improves the mAP by 12.8% and 13.2% under IOU@0.5 and IOU@[0.5-0.95]. The comparative results shown in Table 2 and Table 3 both demonstrate that our proposed method achieves a balanced performance in binary quantization for object detection models. It effectively reduces computational complexity and model size while maintaining a certain level of detection accuracy.

5.2 Analysis of Balance Performance

We conducted a comprehensive analysis by selecting all layers that utilize floating-point computations from previous 1-bit detectors and our method. We calculated the total number of multiply-accumulate operations and parameters for each of these layers. The results are presented in Table 4. Remarkably, despite having more than twice as many layers, our model exhibits significantly reduced overall operations and parameters compared to previous 1-bit detectors. The total operations are reduced by a factor of $8.55\times$, while the total parameters are reduced by a factor of $2.09\times$. Furthermore, our compression of parameters surpasses the compression of computations, leading to a substantial improvement in the operational intensity of the model. This indicates that in practical hardware inference deployment, our model can effectively utilize hardware resources, resulting in superior performance. These findings highlight the efficiency and resource utilization advantages of our proposed model, making it highly suitable for practical deployment in hardware inference scenarios.

5.3 Analysis of Metric P_d

With respect to the metric P_d, we visualise its effectiveness for the comprehensive evaluation of objective detection models. The first and second rows of Figure 7 are the variation of P_d with single-inference accuracy under a fixed inference latency and accuracy, respectively. The intersection points between curves of different colors represent the minimum model accuracy required to ensure target detection at a specific latency. The columns display the performance variations under four different time limits. As the latency increases, the detection capability of the model decreases significantly, resulting in lower accuracy. Each black dot within a sub-figure represents the conditions that must be met to ensure task completion. For instance, in Figure 7(a), with a time limit of 100 milliseconds (ms), an accuracy exceeding 85% can be achieved with an inference latency of approximately 20 ms. An accuracy above 75% requires a latency of less than 15 ms and 50% accuracy necessitates less than 5 ms. As the time limit increases, the conditions become more relaxed. Notably, when the accuracy decreases by 10%, the impact of increasing latency on performance degradation becomes more pronounced.

6 Conclusions

In this paper, we present several new ideas for supervising the binary quantisation of YOLO to achieve a balanced performance in terms of computational speed, model size and detection accuracy. The MWL-based shortcut mitigates information flow loss and improves parallel computing efficiency, which can also be utilized to analyze the achievable acceleration limits in practical deployment scenarios. The Roofline Model for binary quantization simultaneously considers the compression of computational complexity and parameter count, ensuring high computational intensity. Both shortcut and Roofline Model extremely valuable for the design of BNNs. The metric P_d can effectively reflect the trade-off between speed and accuracy and guide automatic mixed-precision quantization frameworks to find suitable bit widths for each layer. It can even be used as an evaluation metric for other lightweight methods such as pruning and distillation. Experiments prove that BOB-YOLO outperforms SOTA BNNs. Future work will focus on a more in-depth analysis of adapting BNNs to different hardware platforms.

Acknowledgements

This work is supported by the National Nature Science Foundation of China under Grant 62106161, the Fundamental Research Funds for the Central Universities under Grant 1082204112364, the Sichuan University Luzhou Municipal Government Strategic Cooperation Project under Grant 2022CDLZ-8, the Natural Science Foundation of Sichuan under Grant 24NSFSC3404 and 2023NSFSC0474, and the Tianfu Yongxing Laboratory Organized Research Project Funding under Grant 2023CXXM14.

References

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[2] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[3] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[4] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

[5] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[7] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[9] X. Liu, T. Wang, J. Yang, C. Tang, and J. Lv. Mpq-yolo: Ultra low mixed-precision quantization of yolo for edge devices deployment. *Neurocomputing*, 574:127210, 2024. ISSN 0925-2312.

[10] Z. Liu, W. Luo, B. Wu, X. Yang, W. Liu, and K.-T. Cheng. Bi-real net: Binarizing deep network towards real-network performance. *International Journal of Computer Vision*, 128:202–219, 2020.

[11] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pages 143–159. Springer, 2020.

[12] S. S. Martínez, J. G. Ortega, J. G. García, A. S. García, and E. E. Estévez. An industrial vision system for surface quality inspection of transparent parts. *The International Journal of Advanced Manufacturing Technology*, 68:1123–1136, 2013.

[13] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

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

[15] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[16] J. Redmon, S. Divvala, R. Girshick, and A. 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.

[17] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[18] A. C. Sankaranarayanan, A. Veeraraghavan, and R. Chellappa. Object detection, tracking and recognition for multiple smart cameras. *Proceedings of the IEEE*, 96(10):1606–1624, 2008.

[19] Z. Shen, Z. Liu, J. Qin, L. Huang, K.-T. Cheng, and M. Savvides. S2-bnn: Bridging the gap between self-supervised real and 1-bit neural networks via guided distribution calibration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2165–2174, 2021.

[20] P. Viola, M. Jones, et al. Robust real-time object detection. *International journal of computer vision*, 4(34-47):4, 2001.

[21] Z. Wang, Z. Wu, J. Lu, and J. Zhou. Bidet: An efficient binarized object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2049–2058, 2020.

[22] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[23] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4820–4828, 2016.

[24] S. Xu, J. Zhao, J. Lu, B. Zhang, S. Han, and D. Doermann. Layer-wise searching for 1-bit detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5682–5691, June 2021.

[25] S. Xu, J. Zhao, J. Lu, B. Zhang, S. Han, and D. Doermann. Layer-wise searching for 1-bit detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5682–5691, 2021.

[26] Z. Yang, Y. Wang, K. Han, C. Xu, C. Xu, D. Tao, and C. Xu. Searching for low-bit weights in quantized neural networks. *Advances in neural information processing systems*, 33:4091–4102, 2020.