PG-YOLO: A Novel Lightweight Object Detection Method for Edge Devices in Industrial Internet of Things

CHENHANG DONG\textsuperscript{1}, (Student Member, IEEE), CHENGXIN PANG\textsuperscript{2,} (Member, IEEE), ZHENGWEI LI\textsuperscript{3}, (Student Member, IEEE), XINHUA ZENG\textsuperscript{2}, (Member, IEEE), AND XING HU\textsuperscript{3}

\textsuperscript{1}School of Electronics and Information Engineering, Shanghai University of Electric Power, Yangpu, Shanghai 200090, China
\textsuperscript{2}Academy for Engineering and Technology, Fudan University, Shanghai 200433, China
\textsuperscript{3}School of Optical Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China

Corresponding author: Chengxin Pang (chengxin.pang@shiep.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant SGSCJY00GHJS2000014.

ABSTRACT

With the rapid development of Industrial Internet of Things (IIoT) technology, video surveillance devices and video data in IIoT environments are massively growing and increasingly important. Deploying rapidly evolving deep learning-based object detection algorithms in IIoT can improve the efficiency of video data utilization and increase the automation and intelligence of the IIoT. Facing the transmission latency problem of massive video data, the algorithms are better deployed in edge devices. However, due to the large size and high computing power requirements of existing object detection algorithms, it is difficult to deploy them on edge devices with limited performance. Therefore, to solve this problem, we propose a smaller and less computationally intensive object detection algorithm, PG-YOLO. We replaced all the convolutional modules of YOLOV5 with low computational ghost modules and streamlined the backbone network to improve the network performance. Then, we propose a better pruning algorithm to compress the network, and finally improve the accuracy by distillation to obtain PG-YOLO. PG-YOLO is smaller and faster and can be deployed in edge devices. In addition, we conducted experimental validation on the SHWD dataset. The experimental results show that PG-YOLO can compress the volume of the model by 9 times compared with YOLOv5s, and the compressed detection accuracy reaches 0.934, with only 0.1% loss of accuracy. Also, the time required for inference is reduced by 32.7%, and the inference speed is improved by 10 FPS. Compared with other object detection algorithms, PG-YOLO also has advantages.

INDEX TERMS
Deep learning, object detection, edge device, lightweight model, PG-YOLO.

I. INTRODUCTION

As a typical application of the Internet of Things (IoT), Industrial Internet of Things (IIoT) is the implementation of IoT in various manufacturing systems [1] and the introduction of a large number of IoT devices and computing nodes in production lines and manufacturing processes to enable the monitoring of industrial systems [2]. In the process, the massive and continuous growth of surveillance devices and the high-quality video data generated in IIoT environments has occurred. And computer vision and deep learning-based algorithms have shown great potential for video and image processing. In particular, deep learning-based object detection algorithms have shown significant progress in the accuracy and speed of video data processing [3]. Also, deep learning based object detection algorithms can be well suited for many industrial applications such as quality monitoring, classification counting, and security management. Therefore, the integration of deep learning-based object detection into IIoT systems can enhance the automation and intelligence of IIoT systems greatly [4].

Since the deep learning-based object detection algorithm requires high computing power to analyze the data, it is usually deployed in a cloud server with high computing power.
After the video data generated by surveillance devices is transferred to the cloud server, the object detection algorithm then processes it. With the massive increase in the volume of high-quality surveillance video data brought about by the explosive growth of surveillance devices in IIoT environments, the traditional cloud computing model is facing increasing challenges. On the one hand, there is an increasing gap between the data processing capacity of cloud computing and the speed of video data generation. On the other hand, the transfer of large volumes of video data to servers can lead to network congestion and affect IIoT systems that are usually very latency sensitive [5]. While deploying object detection algorithms to the edge is a natural solution, it introduces new challenges due to the limited computational power of edge nodes [6].

In order to obtain higher accuracy, the general trend in object recognition is to build deeper and more complex networks. In particular, the self-attention target detection algorithm containing Transformer structure, these algorithms have further improved the accuracy of recognition [7], [8], [9], [10], [11], [12]. However, the larger networks resulting from increased accuracy do not necessarily improve recognition speed. Due to the limited computational power and storage space of edge nodes, deploying object detection algorithms to the edge requires the algorithm itself to be sufficiently lightweight. Existing deep learning-based object detection algorithms have difficulty meeting this, with most networks being oversized in terms of the number of parameters, computation, and model size. Especially in terms of volume, even some lightweight algorithms are still too large in terms of the number of parameters and the size of the weight files. And some too small models sacrifice accuracy, and the lower accuracy is difficult to meet the demand of practical deployment. To solve this problem, there is an urgent need to lightweight the object detection models based on deep learning. Therefore, we propose PG-YOLO that compresses substantially in size with little loss of accuracy and can meet the practical requirements for deployment at the edge side.

There are three approaches to lightening the neural network currently used for object detection. (1) Choosing an object detection algorithm framework that is advantageous in terms of volume, such as the one-stage algorithm. (2) Using a more compact model (3) Compression of the neural network model. Therefore, this paper combines the above three methods and proposes a more lightweight object detection network model. We have optimized and compressed the network based on the excellent object detection algorithm YOLOv5s. First, we replaced all the convolutional modules in the original network with Ghost module, a much smaller computational and volume module, and optimized the redundant parts of the network structure. Then we obtained the extremely light model PG-YOLO by sparse training and pruning. Finally, we used knowledge distillation to improve the final accuracy of PG-YOLO, so that PG-YOLO combines both lightness and network accuracy.

Due to the great success of deep learning-based image classification, object detection techniques using deep learning have also seen some research progress in recent years, and have been further applied in different fields [13], [14]. The current major deep learning-based object detection frameworks can be divided into two categories: two-stage detectors, such as region-based CNN (R-CNN) [15] and its variants [9], [16]. The other one is one-stage detectors, such as YOLO (You Only Look Once) [7] and its variants [8], [10], [17]. The two-stage detector first uses a region candidate network (RPN) to extract candidate object information, and then the detection network completes the prediction and detection of the location and class of the candidate object, which usually achieves higher localization and object detection accuracy [3].

The one-stage detector does not need to use RPN but generates the object location and category information directly through the network, which is an end-to-end object detection algorithm. Therefore, one-stage detection algorithms are usually faster [7], [18]. Two-stage detection algorithms are more accurate but require a slower detection speed. In contrast, single-stage detection, although slightly less accurate, has advantages in terms of speed and model size that meet the needs of this paper. Therefore, the one-stage detector algorithm will be used as the basis for enhancement in this paper. Among the one-stage detectors, the YOLO series algorithm is representative [3]. The YOLO series algorithms have reached the current mainstream level in terms of speed, quantity and accuracy, with YOLOv4 and YOLOv5 as the most widely used. YOLOv4 has made a lot of improvements based on the previous single-stage algorithm, absorbing many excellent ideas, such as data augmentation, CSP structure, etc. The related algorithms also have a large number of applications in the industry [19], [20]. And then YOLOv4 introduced YOLOv4-tiny optimized for edge-side devices has greatly improved the real-time performance of the model, but there is still room for optimization. YOLOv5 was released later, and it is very similar to YOLOv4 in architecture. While YOLOv4 outperforms YOLOv5 in terms of performance, it is less flexible and faster than YOLOv5. YOLOv5 has been optimized for accuracy since then. This makes YOLOv5s version, the smallest of the four versions of YOLOv5, have a good balance of speed and accuracy. Therefore, this paper will be based on YOLOv5s for improvement.

In addition, the recently proposed Detection Transformer (DETR) [21] is the first fully end-to-end target detector. By utilizing Transformers [22], originally developed for language tasks, the final set of predictions can be output directly without further post-processing. Although DETR performs relatively well in terms of accuracy, it has disadvantages that cannot be ignored. On the one hand, its structure gives it relatively low performance in detecting small objects. On the other hand, and its main drawback, DETR requires a long training time to converge. Therefore a large amount of research work has focused on how to alleviate this problem [23], [24]. There are also some studies that attempt
to introduce Transformer structures and ideas in the original target detection framework [25]. Although these works have made some progress, Transformer-based object detection algorithms still have some distance to go in practical applications.

Due to the need to deploy neural networks on embedded devices, a series of compact models have been proposed in recent years. MobileNet [26] is a cell phone neural network model proposed by Google in 2017. It mainly uses deep separable convolution to reduce the computation. MobileNetV3 [27] is the third version of the MobileNet series, in addition to absorbing the bottleneck structure with reverse residuals in MobileNetV2 [28]. Further optimizations were made to (1) search the entire network architecture using NAS, (2) search the appropriate network width using NetAdapt, (3) introduce an attention mechanism in the SE module, (4) introduce the h-swish activation function, and ShuffleNet introduces a channel shuffle operation to improve the information flow exchange between channel groups. ShuffleNetV2 [29] further considers the actual speed of the object hardware for compact design. In addition, to reduce the computational consumption of neural networks, the [30] proposes the Ghost module to build an efficient convolutional module. This module divides the original convolutional layer into two parts, first using fewer convolutional kernels to generate a small number of internal feature maps, and then a simple linear change operation to further generate ghost feature maps efficiently. From the experimental point of view, GhostNet has the best detection accuracy and compression compared with other models, even surpassing mobilenetV3 [30].

The main model compression methods currently used include knowledge distillation, low-rank decomposition, parameter quantization, and parameter pruning. Knowledge distillation [31] is more complex to train and is used for image classification tasks with most knowledge distillation methods and is not applicable for object detection tasks. The end result of quantization, low-rank decomposition, and pruning are all from pre-trained large models to lightweight small models, reducing the network parameters and computational effort. Among them, pruning makes important filtering for different granularity of units in the network and prunes the unimportant units to maintain the network performance with more flexibility [32].

II. RELATED THEORY

Here we briefly introduce the theory related to this paper.

A. YOLOv5 NETWORK

YOLOv5 is an object detection network proposed by the ultralytics company. It takes both detection efficiency and accuracy into account. It is a real-time and efficient one-stage object detection algorithm. Based on the original YOLO framework, the algorithm optimizes the backbone of feature extraction, the neck network for feature fusion, and the output of prediction head for classification and regression, and integrates the excellent algorithms and models of deep convolution neural networks in recent years. YOLOv5 is available in 4 sizes with different volumes and YOLOv5s have the smallest network volume.

YOLOv5s network mainly uses the Bottleneck CSP structure. The Bottleneck CSP structure is divided into two parts. The first part performs a classical residual structure called the Bottleneck operation, which undergoes The first part performs the Bottleneck operation, which is a classical residual structure that undergoes a 1 × 1 and 3 × 3 convolution operation to add the convolution result to the input. The other part is dimensioned down by 1 × 1 convolution, reducing the number of channels by half. In the YOLOv5 series, the network structure size is controlled by two parameters: depth and width. So there are 4 models of YOLOv5, the difference is the width and depth of the network Fig. 1 shows the network structure of YOLOv5 network.

B. GHOST MODULE

The [30] proposes that well-trained networks usually have rich or even redundant feature map information to ensure the understanding of the input. And for these redundant feature map information can be obtained from other feature maps using an inexpensive linear transformation, which can reduce the overall computational consumption. The specific Ghost module structure is shown in Fig. 2.

In ordinary convolution, the transformation that completes the input $X = h \cdot w \cdot c$ to the output $h' \cdot w' \cdot n$. The number of FLOPs needed in this convolution process is $n \cdot h' \cdot w' \cdot c \cdot k' \cdot k$, which is usually large due to the number of convolution kernels $c$ and the number of channels $n$, which are usually very large (e.g., 256 or 512).

The output feature mappings of a convolutional layer usually contain many redundancies, some of which may be similar to each other. Therefore, a cheap conversion can be made
8 represents the cheap operation. The Ghost module outputs the same number of channels.

The final compression ratio is $S$ and $s=n/m$ which means the smaller the value of $m$, the more computationally compressed it is. However, excessive compression does not guarantee that too many ghost feature maps can contain enough rich features. Therefore, set $S$ in the range of $\{2, 3, 4, 5\}$ and conducted a comparison experiment. When $S$ increases, the FLOP of the Ghost method decreases significantly and the accuracy also decreases gradually. However, when $S = 2$, the Ghost method performs optimally and even slightly better than the original model. Based on this result, this paper also sets $S$ to 2.

**C. MODEL COMPRESSION**

At present, the main methods of model compression include knowledge distillation, low-rank decomposition, parameter quantization, parameter pruning [33], channel pruning [34], and so on. Different methods are applied to different situations [35], [36]. In this paper, channel pruning and knowledge distillation are selected to compress the model.

Channel pruning [37], [38] is based on sparse training of BN layers in the network with a trained model; Then, the network is sorted according to the weights added by sparse training; Finally, the appropriate threshold or pruning rate is set, and if the weight is less than the set range, the channel will be pruned.

Regarding the BN layer i.e., batch normalization has been used as a standard method by most convolutional neural networks to achieve fast convergence and better generalization performance. Also, the way BN normalized activation makes possible a simple and efficient way to count the scaling factors of the channels. After thinning training, the $\gamma$ parameter in the BN layer is used as the scaling factor needed to thin the network, and all the convolutional layers are trained sparsely. With sparse training, we can filter out the channels with low-performance contributions and remove them. Although there is a loss of precision in removing these channels, the loss can be recovered by fine-tuning and will greatly compress the model.

Knowledge distillation is another common approach in model compression and was first applied to classification tasks [31]. Unlike pruning and quantization in model compression, knowledge distillation is done by constructing a small lightweight model and using supervised information from a larger model with better performance to train this small model in order to achieve better performance and accuracy. This large model we call the teacher model, and the small model we call the student model. The teacher model outputs supervised information, and the student model learns to migrate supervised information from the teacher model, a process known as knowledge distillation. The training accuracy of the teacher model is higher than that of the student model, and the larger the difference between the two, the

\[
R_s \approx \frac{n \cdot c}{m \cdot c + n - m} \approx \frac{s \cdot c}{c + s - 1} \approx S
\]
III. NETWORK OPTIMIZATION AND COMPRESSION

In order to solve the problem of insufficient computing power to deploy large networks in mobile and embedded devices, in this paper, we propose a PG-YOLO network by combining relevant theories and considering the weight size, computation, detection speed, and accuracy. Firstly, we use ghost bottleneck to replace all the convolutional modules in YOLOv5s to obtain the initial lightweight network model. We then modified the backbone network by selectively removing inefficient structures from the backbone, and then we use the R-pruning method to further compress the network. We further improve the accuracy of the network by using distillation after completing the compressed network, so that the accuracy of the network remains high after several times of compression.

A. NETWORK OPTIMIZATION

Due to the excellent performance of YOLOv5s, it is chosen as the basis of algorithm improvement to ensure that the algorithm can eventually obtain good accuracy and speed. Ghost module can complete the work of information extraction with less computational power cost and storage space cost through cheap operation and identity. Therefore, replacing the normal convolution module in YOLOv5s with a better performance Ghost convolution module can significantly reduce the size and computation of the network and complete the lightweighting of the network.

The Ghost bottleneck is a stack of ghost modules, the specific structure of which is shown in Fig. 4. On top of the ghost module, the reimaging bottleneck consists mainly of two stacked reimaging modules. The first layer is used as an extension layer to increase the number of channels. The second Ghost module reduces the number of channels to match the shortcut path.

Then, the inputs and outputs of these two Ghost modules are connected using shortcuts and taking the advantage of MobileNetV2, no ReLU is used after the second Ghost module and the other layers apply batch normalization (BN) and ReLU nonlinear activation after each layer. For the case of stride = 2, through the shortcut path by the depthwise convolution between the lower sampled layer and stride = 2, as shown in Fig. 4.

The specific structure of the C3 module in the original YOLOv5 is three normal convolutions and a CSP bottleneck block. We replace the CSP block with a Ghost bottleneck, and the replaced C3 is denoted as C3Ghost. C3Ghost can also perform the function of the C3 module for learning residual features, but it is faster and smaller. In addition, the C3 module in YOLOv5 is an optimization of CSPNet, which differentiates the input layer and convolves only a portion of the features before Concat. The second and third C3 modules of the backbone contain nine bottlenecks to enhance the feature extraction capability. However, we found that the number of these bottlenecks was too large and not cost-effective for model accuracy improvement. Reducing the number of bottlenecks did not result in any loss of accuracy and reduced the number of parameters and the size of the network to some extent. So we reduced the number of Ghost bottlenecks in the second and third C3Ghost of the backbone, and get the LightC3.

YOLO-Ghost is obtained by replacing the convolution module and C3 module of the original YOLOv5 with a Ghost module and LightC3 module. the YOLO-Ghost network can fully combine the accuracy advantage of YOLOv5 and the lightweight advantage of the Ghost module, and achieves the initial compression compared with the original YOLOv5s. Finally, the complete network structure is shown in Fig. 5.

B. NETWORK COMPRESSION

The pruning algorithm is one of the model compression methods, and model compression can reduce the size of the
network significantly. Pruning starts with sparse training [34], sparse training is performed by training the BN layer with L1 regularization so that the weights of the BN layer [37] converge to 0 as much as possible and the sparse weights are redistributed to the other effective layers of the network. The pruning algorithm then removes unnecessary channels based on the weights and never achieves model compression. The pruning algorithm can effectively compress the size of the network while maintaining accuracy as much as possible. Therefore, based on YOLO-Ghost, we use pruning for further network compression. We use YOLO-Ghost as a model compression baseline for sparse training, pruning, and fine-tuning.

Sparse training will directly affect whether the pruned model meets the required performance. The loss function formula of sparse training is as follows:

\[
L = \sum_{(x, y)} l(f(x, w), y) + \lambda \sum_{\gamma \in T} g(\gamma)
\]  

In (3) the first sum represents the original loss function, the second sum represents the new sparse training loss, and \( g(\gamma) \) is the loss function of the scaling factor to evaluate whether the scaling is reasonable. The \( \lambda \) represents the sparse coefficient, which can adjust the sparsity training strength.

By sparse regularization training, we make most of the \( \gamma \) factors of the BN layers in the model (both original and newly added) close to zero, which indicates that the contribution of the channels corresponding to these \( \gamma \) factors is reduced. We then use a smaller scaling factor to prune the network by removing the low-contributing channel incoming and outgoing and the corresponding weight parameters. The scaling factor here is a global threshold across all layers, which is defined as the percentage of bits occupied among all \( \gamma \) factors values. By doing this, we can obtain a narrower network and fewer computational operations.

In general, the number of channels in each layer of the network is designed to be \( 2^N \) to satisfy the computational characteristics of the GPU in order to enable the network to efficiently utilize the computational power of the GPU during the inference calculation. The previous pruning operation prunes the channels at each layer according to the gamma factor of each channel from small to large, and the channels with \( \gamma \) factor less than the threshold are pruned. However, this operation destroys the designed number of channels of the network, which leads to the inefficient use of GPU computing power, so sometimes the small size of the network is slower than the large size of the network in inference. Therefore we propose R-pruning (Recovery Pruning). After ordinary pruning, the number of channels in each layer that does not meet the number of \( 2^N \) is supplemented by restoring the channel with the largest \( \gamma \) factor from the deleted channels until the number of channels in that layer meets \( 2^N \). The optimized pruning process is shown in Fig. 6.

We use the set \( C \) to denote the retained channels after pruning at a certain layer, \( c_i \) denotes the \( i \)th channel, and \( \gamma_i \) denotes the \( \gamma \) coefficient of the \( i \)th channel. Channels larger than the threshold \( \gamma_{TH} \) will be retained. In the optimal pruning algorithm, we recover several channels based on the ordering of \( \gamma \), with the aim that the number of retained channels \( |C'| = 2^N \), \( N \in \mathbb{N}^* \).

\[
\begin{align*}
C &= \{c_1, c_2, \ldots, c_n\} \\
\gamma_1 > \gamma_2 > \ldots > \gamma_n > \gamma_{TH} \\
C' &= \{c_1, \ldots, c_n, c_n+1, \ldots, c_{n+z}\} \\
\gamma_1 > \ldots > \gamma_n > \gamma_{TH} > \gamma_{n+1} > \ldots > \gamma_{n+z}
\end{align*}
\]  

The optimized pruning algorithm has a significant speed advantage in actual deployment compared to the model size after previous pruning, although it will be on the larger side.

A large compression of the model inevitably results in a loss of accuracy, and although fine-tuning will restore some of the performance, there is still a loss of accuracy. Therefore, we use knowledge distillation to improve the accuracy of the model.

We train a teacher model on top of the dataset in advance, and then use the acquired teacher model for supervised training to achieve the distillation purpose when training the student model. Specifically, when the student model is trained, the distillation loss function calculates the difference between the previous output predictions of the two models, and the loss of the student model is added together as the whole training loss to perform a gradient update. Finally, higher

![Figure 6. The process of R-pruning. We recover the channels(orange) with larger scaling factors after pruning so that the number of channels retained after pruning is equal to \( 2^N \).](image-url)
C. EXPERIMENTS AND DISCUSSION

The test environment of this experiment is Ubuntu 16.04 operating system, and Pytorch deep learning framework implements the compression algorithm of the PG-YOLO object detection model. The hardware configuration of the server is NVIDIA RTX 3080 GPU *1 and Intel Core i7 CPU; We used the NVIDIA Jetson TX2 as the development board in order to realize on edge device detection during the testing part of the experiment. The NVIDIA Jetson TX2 is an embedded vision computer system with the 256-core NVIDIA Maxwell GPU model, dual-core Denver2 CPU model, and an 8 G memory size. At the same time, TX2 has the characteristics of low power consumption and high performance, making it very suitable as a device for the deployment of algorithms at the edge.

We choose to validate the algorithm with a helmet of targets that may need to be detected in a practical application scenario of edge devices. Therefore, the experiments use the helmet-related dataset safety-helmet-wearing(SHWD). Pascal VOC is used to label and train the data. According to the actual scene, 7571 pictures are detected, including 5450 training sets, 1515 test sets, and 606 verification sets. Fig. 7 shows the images in the dataset and the corresponding labels.

First of all, we use YOLOv5s to start training on this data set and choose not applicable pre-training weight, the epoch is 100. The results of the training were compared. Then we use the YOLO-Ghost after replacing the Convolution Module for training, and the epoch and other settings are consistent with those of YOLOv5s. Ghost convolution effectively reduces the computational effort of convolution by an inexpensive convolution operation.

Ablation experiments Table 1 show that applying the Ghost convolution module to YOLOv5s can significantly compress the size of the network and reduce the computational expenditure of the network.

The C3 module is the main component of the backbone network in Yolo5, which can accomplish the work of feature extraction with less computation. However, in the original network structure, the second and third C3 modules of backbone have a superposition of 3 Ghost Bottlenecks, and we believe that these Bottlenecks are somewhat redundant. Therefore, we set the number N of these Bottleneck to N={1,2,3,4}, respectively, and then conducted a comparison experiment. The experimental results are presented in Table 2. The experiments show that 3 Bottlenecks in YOLOv5s do have redundancy, and the method performs best when the number of Bottlenecks is 2, even slightly better than the performance of 4 Bottlenecks. We named the best-performing C3 module with N=2 as LightC3.

Sparse training is an important step in the process of pruning. Selecting the appropriate sparse training parameters will affect the performance of the model after pruning. The purpose of choosing the appropriate sparse parameters is to select most of the channels which have a low contribution to the network performance and cut them out in the pruning operation. In this way, a large degree of pruning can be completed without affecting the performance as much as possible. If the sparsity factor is too large or too small, it can not effectively filter channels. As shown in Fig. 8, when the sparsity factor = 0.002, the sparsity strength is small and cannot compress the channel weights to 0. Therefore, when pruning, many channels with non-zero weights are pruned, which affects the performance more obviously. And when the sparsity factor = 0.006, most of the channel weights are compressed to 0, so when pruning, even though a small pruning threshold is chosen, most of the channels are still pruned. When the sparsity factor = 0.004, the sparsity is more appropriate, and most of the channel weights are close to 0. With a suitable pruning threshold, the channels with channel weights of 0 can be selectively pruned.

Fig. 8 shows the distribution of weights after training with different sparse coefficients. The orange color in Fig. 8 shows the weights that are retained and the blue color indicates the...
weights that will be removed because they are close to 0. Through sparse training, most of the channel weights in the BN layer are close to 0.

Next, we begin to prune the sparse training model. The optimized pruning algorithm ensures that the number of channels in each layer of the network meets $2^N$ when pruning, which allows the pruned network to make the most efficient use of the GPU’s computational power and provides a significant advantage in inference speed. The experimental results are shown in Table 3. The network after the pruning algorithm has a small speed improvement. And the network after the R-pruning algorithm has a huge speedup even with a large amount of computation (GFLOPS) because the optimized pruning algorithm makes the number of channels per layer satisfy $2^N$.

After sparse training, we prune the network by different percentages. The network with different pruning thresholds has different model sizes. Generally, the larger the size of the network, the higher the accuracy. But when the volume is greatly reduced, the precision loss can be ignored. At 20% pruning, the accuracy of the model is close to the original yolov5s and the size is reduced to 4MB, corresponding to a decrease in the number of parameters and computational effort. We want to get the model with both volume and accuracy. While the maximum degree of 60% pruning compresses the volume to 1.1 MB, the accuracy loss is large. Therefore, we choose 50% pruning and further improve the accuracy by exponential distillation on this basis.

Finally, we choose YOLOv5L as the teacher network after training on this dataset. The completed training YOLOv5L has a volume of 89 MB and 46.5M parameters, while the mAP reaches 0.94. The pruned network is used as the student network. After distillation, we can see that the accuracy of the network has improved. Detailed ablation experiments are presented in Table 5.

We were able to see that PG-YOLO has an amazingly small size while ensuring accuracy. Likewise, we tested the performance of other prevailing object detection algorithms and prevailing lightweight algorithms. The results are presented in Table 6. The results show that PG-YOLO offers a good balance between accuracy and lightness. Compared to YOLOv4, our method has a very close accuracy and huge volume advantage, while compared to other lightweight algorithms, our method is ahead in both accuracy and volume.

Then we tested it on edge-side devices. We selected some algorithms to test and compare with PG-YOLO on Nvidia tx2. The results are shown in Table 7. PG-YOLO has the fastest speed, with an actual inference speed of 30.3 FPS, which can meet the needs of real operation.

### Table 3. Comparison of optimized pruning algorithms.

| Model       | Prams | GFLOPs | Inference time |
|-------------|-------|--------|----------------|
| YOLOv5s     | 7.0M  | 16.3   | 4.8ms          |
| Pruning YOLO-Ghost | 0.63M | 4.3 | 4.7ms         |
| R-pruning YOLO-Ghost | 0.65M | 4.4 | 2.1ms         |

### Table 4. Comparison of different pruning percentages.

| Model | mAP@0.5 | Prams | GFLOPs | Weights |
|-------|---------|-------|--------|---------|
| 20%   | 0.931   | 1.8M  | 6.2    | 4MB     |
| 50%   | 0.926   | 0.63M | 4.3    | 1.6MB   |
| 60%   | 0.921   | 0.4M  | 3.6    | 1.1MB   |

### Table 5. Ablation experiments.

| Model              | mAP@0.5 | Prams | GFLOPs | Weights |
|--------------------|---------|-------|--------|---------|
| A YOLOv5s          | 0.934   | 7.0M  | 16.3   | 14.4MB  |
| B A+Ghost          | 0.920   | 3.66M | 8.1    | 7.8MB   |
| C B+lightG3        | 0.924   | 3.68M | 8.0    | 7.7MB   |
| D R-pruning YOLOv4 | 0.926   | 0.63M | 4.5    | 1.6MB   |
| E D+ KD            | 0.933   | 0.63M | 4.3    | 1.6MB   |

### Table 6. Comparison of model performance and parameters.

| Model              | mAP@0.5 | Prams | GFLOPs | Weights |
|--------------------|---------|-------|--------|---------|
| YOLOv5s            | 0.934   | 7.0M  | 16.3   | 14.4MB  |
| YOLOv4[8]          | 0.946   | 52.5M | 105.6MB|         |
| DETR[21]           | 0.896   | 41M   | 474MB  |         |
| YOLOv4-tiny        | 0.873   | 6.1M  | 22.4MB |         |
| YOLOX-nano[39]     | 0.921   | 2.25M | 7.7MB  |         |
| PG-YOLO            | 0.933   | 0.63M | 1.6MB  |         |

### Table 7. Actual test comparison.

| Model              | mAP@0.5 | Prams | Weights | Inference time |
|--------------------|---------|-------|---------|----------------|
| YOLOv4[21]         | 0.946   | 52.5M | 105.6MB| 526.3ms        |
| YOLOv4-tiny        | 0.873   | 6.1M  | 22.4MB  | 41.3ms         |
| YOLOX-nano[39]     | 0.921   | 2.25M | 7.7MB   | 40.5ms         |
| YOLOv5s            | 0.934   | 7.0M  | 14.4MB  | 48.9ms         |
| PG-YOLO            | 0.933   | 0.63M | 1.6MB   | 32.9ms         |
Compared to YOLOv5s, PG-YOLO has almost the same accuracy, while the volume is compressed by 8.75 times, the time required for inference is reduced by 32.7%, and the inference speed is improved by 10 FPS. This makes PG-YOLO more suitable for deployment in some edge-side devices with limited computing power and high real-time requirements. Therefore, it can be well applied to the edge devices that generate video data in an IIoT environment. PG-YOLO processes video data directly at the edge devices of IIoT with high accuracy and high real-time to improve the automation and intelligence of the IIoT system and solve the transmission latency problem of massive video data in cloud computing.

IV. CONCLUSION

This paper investigates the problem of real-time processing of large amounts of video data in IIoT environments. video data in IIoT can be greatly automated and intelligent when processed with the help of deep learning-based object detection algorithms. Object detection algorithms need to be deployed on edge devices because the cloud approach cannot meet the real-time performance required by IIoT. However, most of the algorithms face challenges when deployed in edge devices with limited performance. Therefore, this paper designs and validates PG-YOLO for edge devices in IIoT by replacing and optimizing the backbone network and combining it with an improved model compression method. Experiments show that the algorithm PG-YOLO in this paper (1) reduces the model size and floating point operations (FLOPs), (2) reduces the time for inference, and (3) sacrifices almost no accuracy. Therefore, the use of PG-YOLO can meet the deployment in IIoT, i.e., PG-YOLO can be deployed on edge devices with limited performance in IIoT. PG-YOLO solves the practical problems faced by object detection algorithms in edge device applications and achieves high accuracy and real-time performance even on platforms with limited performance and storage space. After that, one of our research directions is to focus on video data processing problems in IIoT problems, such as faster transmission or more efficient compression. On the other hand, the inference speed of the algorithm does not improve exponentially compared to the large compression of the volume, so how to further improve the inference speed is also our research direction.

REFERENCES

[1] H. Xu, W. Yu, D. Griffith, and N. Golmie, “A survey on industrial Internet of Things: A cyber-physical systems perspective,” IEEE Access, vol. 6, pp. 78238–78259, 2018.
[2] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, “Industrial Internet of Things: Challenges, opportunities, and directions,” IEEE Trans. Ind. Informat., vol. 14, no. 11, pp. 4724–4734, Nov. 2018.
[3] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, “Deep learning for generic object detection: A survey,” Int. J. Comput. Vis., vol. 128, no. 2, pp. 261–318, Jan. 2020.
[4] F. Saed, M. J. Ahmed, M. J. Gul, K. J. Hong, A. Paul, and M. S. Kavitha, “A robust approach for industrial small-object detection using an improved faster regional convolutional neural network,” Sci. Rep., vol. 11, no. 1, pp. 1–13, Dec. 2021.
[5] L. Hu, G. Sun, and Y. Ren, “CoEdge: Exploiting the edge-cloud collaboration for faster deep learning,” IEEE Access, vol. 8, pp. 100533–100541, 2020.
[6] T. Hafeez, L. Xu, and G. Mcardle, “Edge intelligence for data handling and predictive maintenance in IIoT,” IEEE Access, vol. 9, pp. 49355–49371, 2021.
[7] I. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 779–788, doi: 10.1109/Cvpr.2016.91.
[8] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” 2020, arXiv:2004.10934.
[9] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 2980–2988.
[10] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 7263–7271.
[11] L. Azziz, M. S. B. H. Salam, U. U. Sheikh, and S. Ayub, “Exploring deep learning-based architecture, strategies, applications and current trends in generic object detection: A comprehensive review,” IEEE Access, vol. 8, pp. 107461–107495, 2020.
[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, 2017, pp. 1–11.
[13] S. Zagoruyko, “End-to-end object detection with transformers,” in Computer Vision—ECCV, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds, Cham, Switzerland: Springer, 2020, pp. 213–229.
[14] R. Zhang, L. Wu, Y. Yang, W. Wu, Y. Chen, and M. Xu, “Multi-camera multi-player tracking with deep player identification in sports video,” Pattern Recognit., vol. 102, Jun. 2020, Art. no. 107260.
[15] R. Girshick, “Fast R-CNN,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 1440–1448.
[16] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 28, 2015, pp. 91–99.
[17] Z. Tian, C. Shen, H. Chen, and T. He, “FCOS: Fully convolutional one-stage object detection,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 9626–9635, doi: 10.1109/ICCV.2019.00972.
[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: Single shot MultiBox detector,” in Proc. Eur. Conf. Comput. Vis. Cham, Switzerland: Springer, 2016, pp. 1–37.
[19] U. Farooq, A. Rehman, T. Khaman, A. Amtullah, M. A. Bou-Rabee, and M. Tariq, “Lightweight deep learning model for weed detection for IoT devices,” in Proc. 2nd Int. Conf. Emerg. Frontiers Electr. Electron. Technol. (ICEFEET), Jun. 2022, pp. 1–5, doi: 10.1109/ICEFEET51821.2022.9847812.
[20] H. Zhao, C. Wang, R. Guo, X. Rong, J. Guo, Q. Yang, L. Yang, Y. Zhao, and Y. Li, “Autonomous live working robot navigation with real-time detection and motion planning system on distribution line,” High Voltage, pp. 1–13, Jun. 2022, doi: 10.1049/hve.12221.
[21] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-end object detection with transformers,” in Computer Vision—ECCV, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds, Cham, Switzerland: Springer, 2020, pp. 779–788.
[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, 2017, pp. 1–11.
[23] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, “Deformable DETR: Deformable transformers for end-to-end object detection,” in Proc. Int. Conf. Learn. Represent., 2020, pp. 1–16.
[24] X. Dui, Y. Chen, J. Yang, P. Zhang, L. Yuan, and L. Zhang, “Dynamic DETR: End-to-end object detection with dynamic attention,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2021, pp. 2988–2997.
[25] X. Zhu, S. Lyu, X. Wang, and Q. Zhao, “TPh-YOLOv5: Improved YOLOv5 based on transformer prediction head for object detection on drone-captured scenarios,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops (ICCVW), Oct. 2021, pp. 2778–2788.
[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Tang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861.
[27] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, “Searching for MobileNetV3,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 1314–1324, doi: 10.1109/ICCV.2019.00140.

[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 4510–4520.

[29] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “ShuffleNet v2: Practical guidelines for efficient CNN architecture design,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 116–131.

[30] J. Yim, D. Joo, J. Bae, and J. Kim, “A gift from knowledge distillation: Fast optimization, network minimization and transfer learning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 7130–7138, doi: 10.1109/CVPR.2017.754.

[31] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural network,” in Proc. 29th Annu. Conf. Neural Inf. Process. Syst. (NIPS), Montreal, QC, Canada, 2015, pp. 1–9.

[32] S. Chen, R. Zhan, W. Wang, and J. Zhang, “Learning slimming SAR ship object detector through network pruning and knowledge distillation,” IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens., vol. 14, pp. 1267–1282, 2021, doi: 10.1109/jstares.2020.3041783.

[33] D. Wu, S. Lv, M. Jiang, and H. Song, “Using channel pruning-based YOLO v4 deep learning algorithm for the real-time and accurate detection of apple flowers in natural environments,” Comput. Electron. Agricult., vol. 178, Nov. 2020, Art. no. 105742, doi: 10.1016/j.compag.2020.105742.

[34] S. O’Keeffe and R. Villing, “Evaluating pruned object detection networks for real-time robot vision,” in Proc. IEEE Int. Conf. Auto. Robot Syst. Competitions (iCARSC), J. Calado, L. Bento, N. Lopes, and P. Oliveira, Eds., Apr. 2018, pp. 91–96.

[35] Y. Sun and Z. Yan, “Image target detection algorithm compression and pruning based on neural network,” Comput. Sci. Inf. Syst., vol. 18, no. 2, pp. 499–516, 2021, doi: 10.2298/csis200316007s.

[36] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 2755–2763.

[37] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 1398–1406.

[38] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, “YOLOX: Exceeding YOLO series in 2021.” 2021, arXiv:2107.08430.

CHENHANG DONG (Student Member, IEEE) was born in 1998. He received the B.S. degree in electrical engineering and its automation from the Henan University of Science and Technology, Luoyang, China, in 2020. He is currently pursuing the M.S. degree in Smart Grid Information and Communication Engineering with the Shanghai University of Electric Power (SUEP), Shanghai, China. His research interests include deep learning, object detection, and model compression.

CHENGXIN PANG (Member, IEEE) received the Ph.D. degree from the University of Technology of Troyes, Troyes, France, in 2008.

His research topics have been first magneto-optics isolator in IEF at Paris-Sud University, Orsay, France, and hybrid silicon photonics at Orange Laboratories, France, and in LCF at IOGS, Palaiseau, France. His research interests include networking systems of artificial intelligence (AI) and artificial intelligence and the Internet of Things (AIoT).

ZHENGWEI LI (Student Member, IEEE) was born in 1997. He received the B.S. degree in communication engineering from the Hunan University of Science and Technology, Xiangtan, China, in 2020. He is currently pursuing the M.S. degree in information and communication engineering with the Shanghai University of Electric Power (SUEP), Shanghai, China.

His research interests include deep learning, computer vision, and object detection.

XINHUA ZENG (Member, IEEE) received the Ph.D. degree in nanotechnology and optical instrumentation from the University of Technology of Troyes, Troyes, France, in 2009.

He was an Assistant Professor and a Lecturer at Shanghai Jiaotong University, Shanghai, China, from 2002 to 2006. He was an Associate Researcher or the Associate Director of the Research Laboratory, Institute of Intelligent Machinery, Chinese Academy of Sciences, Hefei, China. He has been the Director of the Center for Technological Innovation and Engineering, Chinese Academy of Sciences, since 2010. He has been the Director of the Smart Linkage, Institute of Industrial Research, Fudan University, Shanghai, where he has also been the Deputy Director of the Center for Wisdom and Systems, Institute of Engineering and Research, since 2020. He has led more than ten national research projects, applied for 80 Chinese patents, and published more than 70 articles. He has published 70 articles. His research interests include autonomous driving, visual imaging, deep learning, bioinformation acquisition, and intelligent information perception and processing.

Dr. Zeng main academic positions include the Deputy Secretary General of INSAI and a Reviewer of Optics Letters, Optics Express, and China Communications.

XING HU received the Ph.D. degree from Shanghai Jiaotong University, in 2016. He is an Associate Professor with the University of Shanghai for Science and Technology. His research interests include computer vision, signal processing, and intelligent driving.