Academic Journal of Science and Technology
ISSN: 2771-3032 | Vol. 10, No. 2, 2024

A Design of Model Compression Method Based on YOLOv5

Zeyang Wang[1], Huawei Mei[1,*] and Yi Fang[1]

[1]School of Control and Computer Engineering, North China Electric Power University, Baoding 071003, China
[*]Corresponding author Email: fn123@ncepu.edu.cn

Abstract: In response to the requirements for detection accuracy and speed in security detection technology in current smart city construction, a model compression method based on YOLOv5s is proposed, aiming to lighten the original model and detect the opening and closing states of car windows. Firstly, a structured pruning method based on channel weights is employed to perform channel pruning. Then, a fine-tuning training method based on logical distillation is adopted to restore the network detection accuracy. Experimental results on the car window dataset show that compared with the original algorithm, the improved algorithm only loses 0.3% in detection accuracy while reducing the number of parameters and computations to 41.5% and 44.3% of the original model, respectively. This significantly reduces the model complexity and achieves model lightweighting.

Keywords: Object Detection, Model Compression, Pruning, Logical Distillation.

1. Introduction

With the continuous advancement of smart city construction and the rapid development of artificial intelligence technology, intelligent security and personnel intelligent management have gradually become the focus of social attention. Smart cities have installed numerous sensors to capture vast amounts of data, such as surveillance videos, environmental, and transportation data. Facing the massive volume of video surveillance data, diverse personalized security task requirements in various scenarios, and the performance demands of industrialized applications, traditional security technologies can no longer meet current market needs. Compared to traditional security technologies, deep learning-based visual algorithms exhibit significant advantages in network recognition accuracy, efficiency, and model generalization capabilities, making them crucial for obtaining the most useful information from massive datasets. Target detection technology plays a vital role in the field of intelligent security.

Moreover, as edge devices (such as smartphones and embedded systems) become increasingly popular due to their small size, high cost-effectiveness, and low power consumption, more researchers are exploring their computing capabilities. Target detection algorithms can also leverage the advantages of edge computing to enhance the real-time responsiveness of systems and be applied in areas such as autonomous driving, unmanned aerial vehicle (UAV) automatic inspection, and face recognition.

However, as the pursuit of detection accuracy continues to increase, the network structures of target detection algorithms have become increasingly complex, with deeper network layers, leading to a sharp increase in model parameters and computations. This poses higher requirements for devices and makes it difficult to deploy and apply them on edge computing devices with limited computing power and storage, such as smartphones and embedded systems.

Therefore, many experts and scholars have begun to investigate methods for lightweighting deep learning models and deploying lightweight target detection models on edge devices to enable real-time detection tasks. To achieve this goal, this article proposes a model compression design method based on YOLOv5, aiming to lighten the YOLOv5 target detection model without compromising detection accuracy. The following innovations are made in this article: 1) adopting a structured pruning method based on channel weights to prune channels with lower weights and minimal impact on the network model, achieving effective model compression; 2) employing a fine-tuning training method based on logical distillation to restore the detection accuracy reduced by pruning operations. Finally, experiments demonstrate the effectiveness of the proposed model compression method.

2. Related Work

2.1. Deep Learning-Based Object Detection Technique.

Deep learning-based object detection algorithms can be roughly categorized into two main types: 1) two-stage object detection, such as R-CNN[1], Fast R-CNN[2], Faster R-CNN[3], and so on; and 2) one-stage object detection, represented by the YOLO series[4-7]. In two-stage object detection, candidate regions are typically generated first, followed by extracting and encoding feature vectors using deep convolutional networks. Then, the categories and locations of objects within the candidate regions are detected. In contrast, one-stage object detection models eliminate the candidate region generation stage, directly classifying each region of interest as background or a target object. As a result, they offer faster inference speeds and are more suitable for real-time object detection scenarios, albeit with slightly lower accuracy compared to two-stage methods. In applications requiring real-time and rapid responses, one-stage object detection algorithms are generally preferred. In this paper, we focus on improving the single-stage object detection algorithm YOLOv5 to strike a balance between accuracy and detection speed.

2.2. Model Compression Technique.

Due to the vast number of parameters and complex computational processes inherent in deep learning models for

object detection, their deployment on resource-limited hardware platforms is often constrained. To address this challenge, it is necessary to compress the storage space and accelerate the computational processes of deep models. To achieve these goals, common methods include knowledge distillation [8], network pruning [9, 10], model quantization [11, 12], and low-rank decomposition [13, 14].

3. A design of model compression method based on YOLOv5

3.1. YOLOv5.

The YOLOv5 algorithm shares a similar network structure with the YOLO series, consisting primarily of four components: Input, Backbone, Neck, and Head. Due to its smaller model size, YOLOv5s is widely used in scenarios that require lightweight solutions. The structure of the YOLOv5 algorithm is illustrated in Fig. 1.

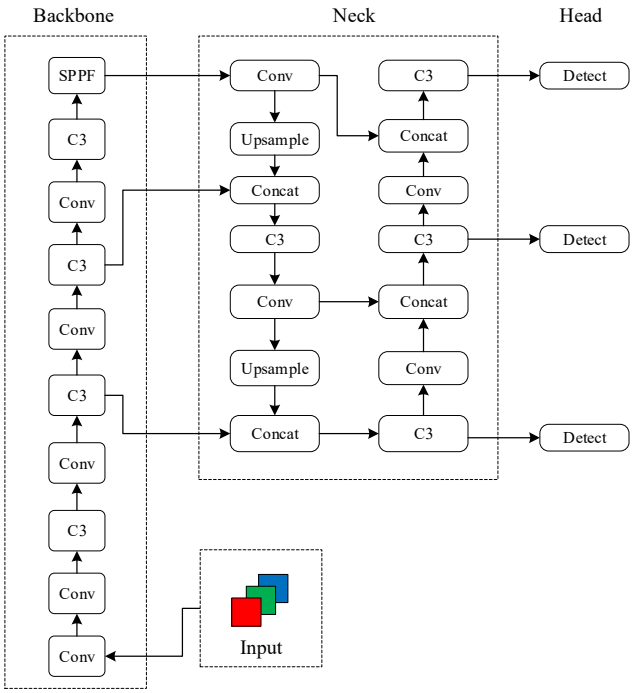


Figure 1. The Network Structure of YOLOv5

The Input section of the YOLOv5 network handles the input of three-channel RGB images. It performs Mosaic data augmentation, adaptive processing of image sizes, and optimization of anchor box calculations. Mosaic data augmentation enriches image information by combining four images into one, scaling the resulting image to a standard size for training. This reduces the model's dependency on batch size and effectively improves detection accuracy. Anchor box calculations compare predicted bounding boxes with ground truth boxes, obtaining the differences and then updating them in reverse to iteratively optimize parameters and approximate the optimal anchor box values.

The Backbone, consisting of CSP and Spatial Pyramid Pooling Fast (SPPF) structures, is primarily responsible for feature extraction in object detection. The CSP module reduces computational complexity and enhances the network's learning performance, having been upgraded to the C3 module in the 6.0 version of YOLOv5. SPPF integrates deep semantic information with shallow semantic information, further improving the network's detection accuracy.

The Neck, the feature fusion layer, employs a Path Aggregation Network (PANet) that combines the FPN layer, which merges deep and shallow features, with a feature pyramid that merges shallow features with deep ones. This fusion enhances the model's feature extraction by integrating information from different network layers in the Backbone and achieving a blend of information from different feature levels.

The Head, the output section, generates a vector that describes the target's category probability, score, and the location of the predicted bounding box. There are three YOLO Head detectors that output feature maps of different scales for object prediction, detecting targets of varying sizes. Each detection head obtains a different vector, which is used to determine the predicted bounding box location and category information of the target in the original image.

While YOLOv5 has achieved impressive results, it still has some limitations, particularly its high computational demand, which makes it difficult to deploy on small embedded devices or mobile devices, resulting in high overall system hardware costs. To make it suitable for edge computing devices, this paper compresses the YOLOv5 model to reduce its complexity and computational requirements while minimizing precision loss.

3.2. Structured Pruning Based on Model Channel Weights.

In neural networks, due to the vast number of parameters, there often exist a significant amount of redundant connections and parameters. These redundant parts contribute little to the model's performance and may even introduce noise and interference. Pruning operations aim to reduce the complexity and computational cost of the model by eliminating these redundant parts while minimizing the impact on performance.

Since unstructured pruning can disrupt the network structure and is not hardware-friendly, this paper adopts structured pruning to preserve the integrity of the network structure. Channel pruning is performed based on the weights of each channel.

Batch Normalization (BN) is a commonly used technique in deep learning that addresses the issue of internal covariate shift. It normalizes the input data to accelerate the convergence of the network and improve training stability. Specifically, BN layers normalize the distribution of input data by standardizing each feature dimension to have a mean of 0 and a variance of 1, thus avoiding gradient vanishing issues caused by deeper network layers and speeding up the convergence of model training.

In YOLOv5, BN layers are typically placed after convolutional layers. Each BN layer contains four learnable parameters: shift parameters β, scale parameters γ, mean, and variance statistics. The shift parameters β and scale parameters γ are used to shift and scale the BN layer, normalizing the distribution of the input data. The mean and variance information are used to record the statistical information of the input data for normalization during the testing phase.

The scale parameters γ of the BN layer's parameters directly affects the output feature vector of the current channel. When the value of γ approaches 0, it indicates that the

current channel has a minimal impact on the network and a low weight. Therefore, the γ of the BN layer can be used as influencing factors for channel pruning to measure the importance of each channel.

For the set of Γ corresponding to each channel γ in the BN layer, L1 regularization is employed to achieve sparse expression. By introducing sparse expression into the network's loss function L for sparse training, the formula is as follows:

$$L = \sum_{(x,y)} l(f(x,W),y) + \lambda \sum_{y\in\Gamma} g(\gamma) \quad (1)$$

Where, x and y represent the input and target of the network, respectively, W represents the trainable weights of the network, $\sum_{(x,y)} l(f(x,W),y)$ represents the original loss function, λ is the balancing factor, and $g(\gamma)$ is the loss for sparse training of the scaling parameter γ, which is implemented using L1 regularization, i.e., $g(\gamma) = |\gamma|$.

During sparse training, keeping the balancing factor λ constant is a commonly used approach. Although it can achieve good sparsity results for γ, it requires a significant amount of time to find the appropriate value of λ. Therefore, a subgradient descent optimization method is adopted to continuously decrease the value of λ during training, allowing γ to gradually approach 0 and achieve sparsity.

After sparse training, the obtained scaling factors γ are sorted. Based on the set pruning ratio, a pruning threshold is determined. If the value of a scaling factor γ is below this threshold, it is considered a less important channel and will be pruned.

The specific process of channel pruning is illustrated in Fig. 2. In the i-th convolutional layer of the original network, each feature channel is connected to a scaling factor γ that has undergone sparse processing. If the values of scaling factors γ corresponding to feature channels Ci2 and Ci3 are below the preset threshold, the convolutional kernels and feature channels associated with these channels will be removed, effectively compressing the model size.

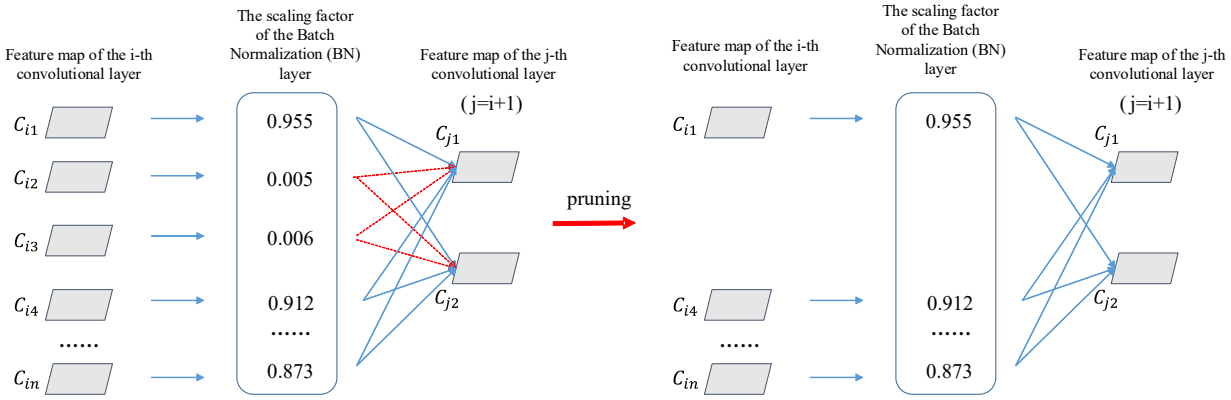


Figure 2. The Process of Channel Pruning

3.3. The design of fine-tuning training method based on logic distillation.

After pruning and compressing the model in Section 3.2, the number of parameters and computational complexity of the model decrease significantly, but there is also a certain loss in model accuracy. At this point, fine-tuning training is needed to restore the accuracy. The commonly used fine-tuning training method is to directly retrain the pruned network to restore its detection accuracy. In this paper, a fine-tuning training method based on logic distillation is designed, where the unpruned network serves as the teacher network and the pruned network serves as the student network. By transferring the logit information from the large neural network to the small neural network, knowledge transfer between the two models is achieved. By adopting the logic distillation method, an auxiliary network is introduced to supervise and assist the fine-tuning training process of the pruned network, thereby fully utilizing richer sample information and ultimately obtaining better detection performance. This method can effectively improve the performance of the pruned network while ensuring the accuracy and generalization ability of the model.

In this paper, logic distillation is used for fine-tuning training after pruning. The algorithm framework is shown in Fig. 3. The algorithm consists of two branches, namely the teacher network branch and the student network branch. The teacher network branch is the original unpruned YOLOv5 object detection algorithm, while the student network branch is the YOLOv5 object detection algorithm after network pruning in Section 3.2. After inputting the detection image, it undergoes feature extraction through both the teacher network and the student network, and finally outputs the image position, category probability, and confidence level of the student network. In the process of logic distillation, the loss function consists of the bounding box loss, category loss, and confidence loss of the student network, as well as the distillation output loss. Using the offline distillation method of the pre-trained teacher network, only the parameters of the student network are updated.

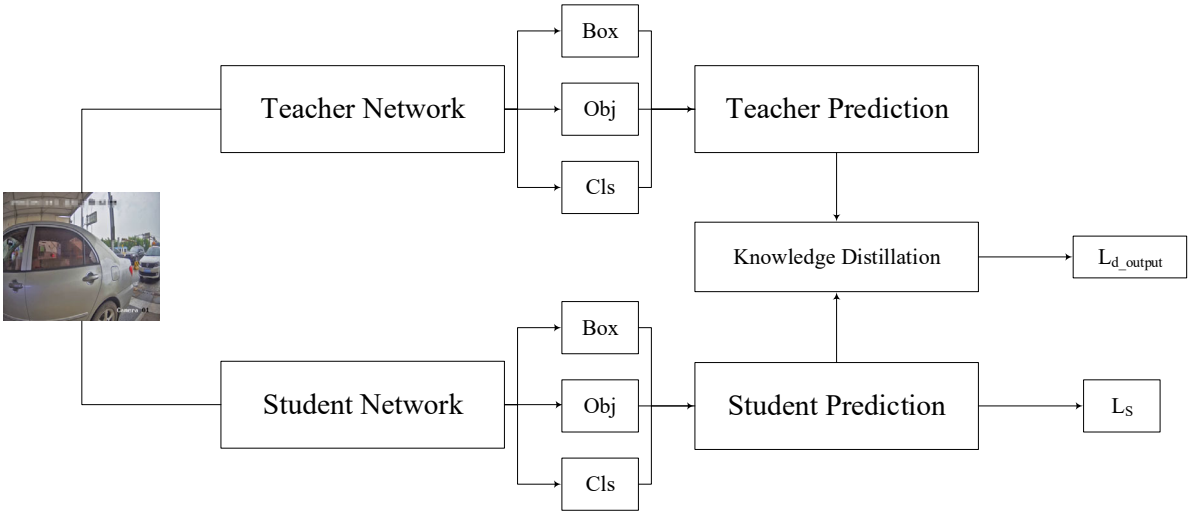


Figure 3. The Framework of Knowledge Distillation Algorithm

As can be seen from Fig. 3, the distillation loss function consists of the loss function of the student network model and the distillation output loss. The formula is as follows:

$$L = L_s + \alpha L_{d\_output} \tag{2}$$

Where, L_s represents the loss function of the student network model, L_{d_output} denotes the distillation output loss, and α is the weight parameter that adjusts the influence of distillation loss within the total loss. The value of α affects the student's tendency to learn from the teacher network, determining whether it prefers to acquire knowledge from the teacher's predictions or directly from the true labels. In this paper, α is set to 10, indicating that the distillation loss will be amplified by a factor of 10.

L_s represents the hard loss of the student model, reflecting the discrepancy between the student's predictions and the true labels. Its numerical value corresponds to the loss function of the original YOLOv5. L_{d_output}, as the core component of logic distillation, calculates the difference between the predictions made by the student model and those made by the teacher model. The formula for L_{d_output} is as follows:

$$L_{d\_output} = h_b L_b + h_c L_c + h_o L_o \tag{3}$$

Where, h_b, h_c, and h_o represent the hyperparameter weight coefficients for the bounding box loss, class loss, and confidence loss, respectively. For the calculation of these three types of losses, the Mean Squared Error (MSE) loss function is employed to measure the differences between the student model and the teacher model in terms of the bounding box, class, and confidence. The formula is as follows:

$$MSE(\bullet) = \frac{1}{m}\sum (y_t^i - y_s^i)^2 \tag{4}$$

Where, m represents the number of samples, y_t^i and y_s^i denote the predicted values of the teacher network and the student network, respectively. Therefore, the formulas for calculating L_b, L_c, and L_t are as follows:

$$L_b = w_b \cdot MSE(y_{t\_b} - y_{s\_b}) \tag{5}$$

$$L_c = w_c \cdot MSE(y_{t\_c} - y_{s\_c}) \tag{6}$$

$$L_o = w_o \cdot MSE(y_{t\_o} - y_{s\_o}) \tag{7}$$

Where, w_b, w_c, and w_o represent the scaling weights for the bounding box loss, the scaling weight for the category loss, and the confidence score output by the teacher model, respectively. The values of w_b and w_c are obtained by expanding w_o to the same shape as the bounding box and prediction box coordinates, ensuring that prediction boxes with higher confidence scores carry greater weight in the calculation of bounding box loss and category loss. Through the above formulas, the final distillation loss function L is derived.

4. Experiment

4.1. Experimental Environment and Dataset Sources.

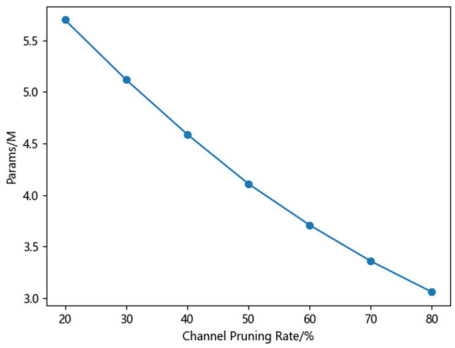
The model training and testing in this paper were conducted on an NVIDIA GeForce RTX 3080 Ti with 12GB of video memory, using the deep learning framework PyTorch 1.10.

For the experiments, YOLOv5s was used as the baseline model. The network parameters were iteratively updated using the SGD method, with the momentum parameter set to 0.937. The initial learning rate (lr) was set to 0.1, the batch size was 32, the input image size was 640*640, and the number of training epochs was 300.

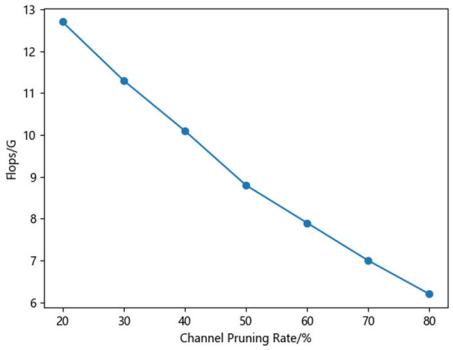
To achieve the detection of open and closed car windows in smart city construction, a custom-made car window dataset was created. This dataset consists of four categories: open front window, closed front window, open rear window, and closed rear window. The dataset contains a total of 10,869 images, with 7,608 images in the training set and 3,261 images in the test set.

4.2. Results of Channel Pruning Experiments.

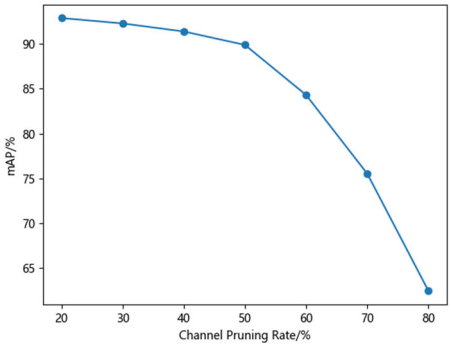
Based on the structured pruning algorithm utilizing model channel weights described in Section 3.2, channel pruning was implemented for the YOLOv5 object detection model. Fig. 4 illustrates the impact of using different pruning rates on the number of parameters, computational complexity, and detection accuracy of the YOLOv5 model.



(a) Trend of Parameter Quantity Change



(b) Trend of Computational Complexity Change



(c) Trend of Detection Accuracy Change

Figure 4. The Framework of Knowledge Distillation Algorithm

As shown in Fig. 4, as the pruning rate continues to increase, the number of parameters and computational complexity of the YOLOv5 model decrease accordingly. When the channel pruning rate is less than 50%, the decrease in detection accuracy is relatively small. However, when the channel pruning rate exceeds 50%, the detection accuracy begins to decline rapidly, and even after retraining and fine-tuning, the accuracy cannot be restored. Therefore, in this paper, a pruning rate of 50% is adopted for subsequent experiments related to knowledge distillation. Table 1 compares the performance of the YOLOv5s model before and after pruning on the custom-made car window state dataset.

Table 1. Comparison of YOLOv5 Model Pruning Performance

Model	mAP50	GFLOPs	Params/M
YOLOv5	0.953	15.8	7.02
50% Channel Pruning	0.899	8.8	4.11

According to Table 1, after applying channel pruning to the YOLOv5 model, the mAP50 decreased by 0.053, while the number of parameters and computational complexity decreased by 41.5% and 44.3%, respectively.

4.3. Results and Analysis of Logical Distillation Experiments.

The results of fine-tuning the pruned YOLOv5 model through retraining and logical distillation-based fine-tuning are presented in Table 2.

Table 2. Comparison of Fine-tuning Performance for Pruned Models

Model	mAP50	GFLOPs	Params/M
50% Channel Pruning for YOLOv5	0.901	8.8	4.11
Retraining Fine-tuning	0.945	8.8	4.11
Logical Distillation Fine-tuning	0.950	8.8	4.11

According to Table 2, after fine-tuning through retraining, the detection accuracy of the model improved by 0.044. When using logical distillation for fine-tuning, the detection accuracy increased by 0.049. Compared to the ordinary fine-

tuning method, the YOLOv5 model achieved a 0.5% increase in accuracy when fine-tuned using logical distillation. Compared to the unpruned YOLOv5 model, the pruned model significantly reduces the number of parameters while only sacrificing 0.3% of detection accuracy, making it lighter and reducing the storage space requirements. Fig. 5 demonstrates the detection results of the optimized model.

Figure 5. Model Detection Results

5. Summary

To reduce model complexity and improve inference speed, this paper proposes a lightweight object detection compression method. First, a structured pruning approach based on channel weights is employed to prune channels with lower weights and minimal impact on the network model in the YOLOv5 model, thereby reducing the number of parameters and computational complexity. Although the detection accuracy of the model slightly decreases after channel pruning, fine-tuning based on logical distillation is then adopted to recover the detection accuracy lost due to pruning. In this process, the unpruned YOLOv5 model serves as the teacher model, while the pruned YOLOv5 model serves as the student model. Experimental results show that the pruned YOLOv5 model achieves a 41.5% reduction in the number of parameters and a 44.3% reduction in computational complexity compared to the unpruned model. After fine-tuning, the detection accuracy of the pruned model is comparable to that of the unpruned model.

In future work, we will also consider the differences in feature representations between the intermediate layers of the student and teacher models to further improve the knowledge distillation algorithm framework.

References

[1] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[2] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

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

[4] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[5] Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

[6] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[7] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[8] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

[9] Luo, J. H., Wu, J., & Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE international conference on computer vision (pp. 5058-5066).

[10] He, Y., Zhang, X., & Sun, J. (2017). Channel pruning for accelerating very deep neural networks. In Proceedings of the IEEE international conference on computer vision (pp. 1389-1397).

[11] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2018). Quantized neural networks: Training neural networks with low precision weights and activations. Journal of Machine Learning Research, 18(187), 1-30.

[12] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

[13] Zhang, X., Zou, J., He, K., & Sun, J. (2015). Accelerating very deep convolutional networks for classification and detection. IEEE transactions on pattern analysis and machine intelligence, 38(10), 1943-1955.

[14] Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., & Lempitsky, V. (2014). Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553.