Noname manuscript No.
(will be inserted by the editor)

A Compression Pipeline for One-Stage Object Detection Model

Zhishan Li [1, *] · Yiran Sun[1, *] · Guanzhong Tian[1] · Lei Xie[1, ✉] · Yong Liu[1] · Hongye Su[1] · Yifan He[2, 3]

Received: date / Accepted: date

Abstract Deep neural networks (DNNs) have strong fitting ability on a variety of computer vision tasks, but they also require intensive computing power and large storage space, which are not always available in portable smart devices. Although a lot of studies have contributed to the compression of image classification networks, there are few model compression algorithms for object detection models. In this paper, we propose a general compression pipeline for one-stage object detection networks to meet the real-time requirements. Firstly, we propose a softer pruning strategy on the backbone to reduce the number of filters. Compared with original direct pruning, our method can maintain the integrity of network structure and reduce the drop of accuracy. Secondly, we transfer the knowledge of the original model to the small model by knowledge distillation to reduce the accuracy drop caused by pruning. Finally, as edge devices are more suitable for integer operations, we further transform the 32-bit floating point model into the 8-bit integer model through quantization. With this pipeline, the model size and inference time are compressed to 10% or less of the original, while the mAP is only reduced by 2.5% or less. We verified that performance of the compression pipeline on the Pascal VOC dataset.

✉Lei Xie
E-mail: leix@iipc.zju.edu.cn

1 State Key Laboratory of Industrial Control Technology and Institute of Cyber-systems and Control, Zhejiang University, Hangzhou, China, 310027
2 Reconova Technologies Co., Ltd, Xiamen, China, 361008
3 Institute of Intelligence Science and Engineering, Shenzhen Polytechnic, Shenzhen, China, 518055
* The authors contribute equally

1 Introduction

Recently, due to the improvement of computing power, convolutional neural networks have become mainstream models for various tasks in computer vision. Along with the constantly increasing accuracy, FLOPs and parameters of the networks are also rising. Unlike large servers in the laboratories that contain multiple GPUs, smart devices, such as mobile phones, drones, or smart cameras for autonomous driving, cannot directly run a variety of high-precision complex DNN models directly. One of solutions is to transfer the data obtained locally to the large server via the Internet, then calculate the result and transfer it back to the local devices. This idea is a common method for the early implementation of neural network models, but the problems brought by this method are the impact of delay and instability of wireless network transmission. Besides, this approach requires lots of expensive communication base stations, which leads to high costs. Another approach is to design more efficient hardware accelerators to reduce the inference time on the local side [1,2,3]. Our method is to design a model compression pipeline, which greatly reduce the size and influence time at the cost of minimal accuracy drop.

There are a lot of redundancies in the DNNs. The existence of unnecessary weights will not only increase the amount of calculation, but also make the model overfit. The proposed model compression pipeline includes the following methods: Pruning, Knowledge Distillation, and Quantization, as shown in **Figure 1**.

Pruning. At present, the most popular pruning algorithm is channel pruning. The advantage of this pruning method is that the overall architecture does not change. What we need to do is to change the number of filters in each convolutional layer. This method is very

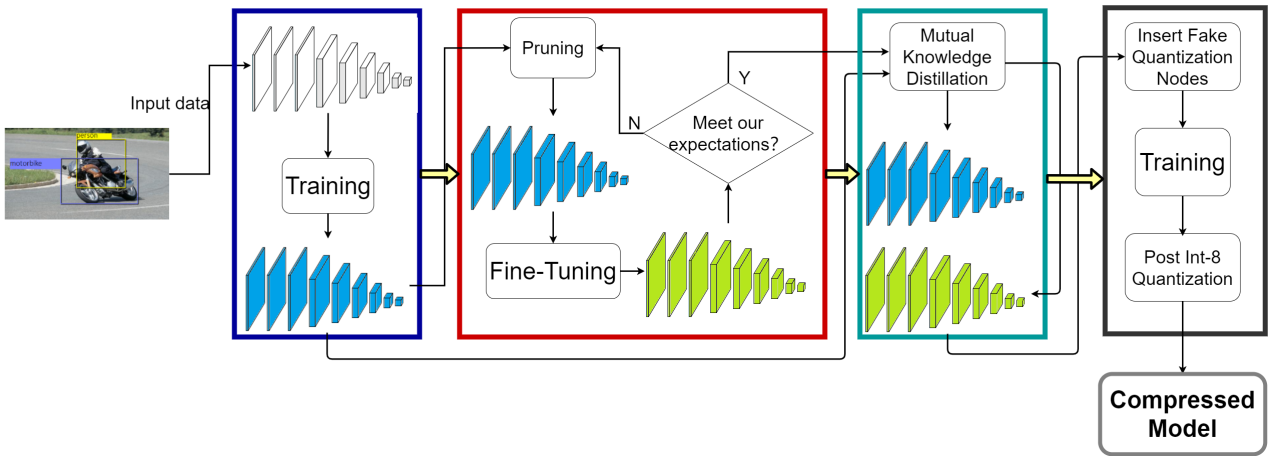


Fig. 1: The compression pipeline for one-stage object detection model. The four blocks represent the process of original training, pruning, knowledge distillation and quantization.

convenient and can be perfectly combined with subsequent knowledge distillation and quantization. However, one of the problems of channel pruning is that some layers may be completely cut off by adopting the global criteria because of the imbalance of weight distribution. This would destroy the original structure and is not conducive to the construction of model and fine-tuning. We have optimized this situation in our pipeline.

Knowledge Distillation. Knowledge distillation is to transfer the knowledge learned by a teacher network (large, complex, and high accuracy) to a student network (lightweight, simple, and low accuracy), so that the student network can learn more knowledge and improve the representation ability. With knowledge distillation the network performance can be improved without changing the network structure. In our pipeline, the original large network is the teacher net, and the pruned network is the student net.

Quantization. Generally speaking, the parameters of neural network models are all 32-bit floating-point (FP-32) numbers. However, a large number of experiments have proved that FP-32 weights are unnecessary[4]. Floating-point weights in the network can be replaced with 8-bit integer (Int-8) parameters, which can greatly reduce the storage space of the model. Post-training quantization is to directly perform the 8-bit quantization operation on the model obtained after traditional training. One problem with this is that it may cause a sharp decline in the accuracy of lightweight network. Another method is quantization-aware training, which is to insert quantization nodes in the network during training. When we input training data into the network, the existence of quantized nodes is taken into account to simulate the forward propagation of Int-8 for quantitative training. After quantization-aware training, we can obtain an Int-8 model with less accuracy drop.

In fact, the effect of using the above algorithms alone is limited. we can use pruning method to obtain a lightweight model that is only 10% to 20% of the original model size, but there will be a great loss of accuracy. Knowledge distillation is a choice to improve the accuracy of slim network. If we design a slim network at will for knowledge distillation, it's difficult to achieve the desired accuracy as well. Quantization can only make the model size become 25% of the original, and cannot continue to reduce the model size. Therefore, we combine the three methods together to achieve the best effect.

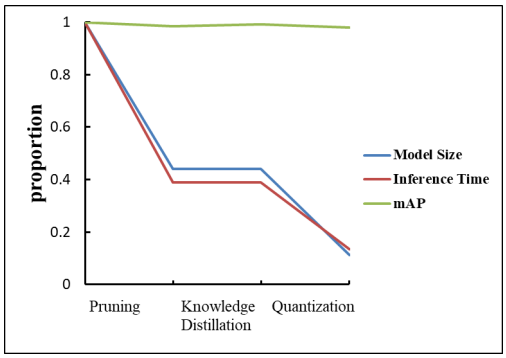
At present, many studies mainly consider pruning and quantization for model compression. These two methods can directly reduce the model size and inference time. However, it would lead to a large drop of accuracy if we directly apply the current image classification pruning algorithm to object detection model and quantized to Int-8 model. The reason is that the fitting ability of pruned model is limited, resulting in a lower accuracy of the quantized lightweight model. Therefore, how to reduce the accuracy loss caused by pruning becomes the top priority. Comparing with other classical pruning methods, our pruning method has obvious advantages in mAP. Our general knowledge distillation strategy can further reduce the loss of precision.

In this paper, we take SSD[5] and YoloV3[6] as research objects and select VGG16[7] as basic backbone. First, we train an original model and then prune it in the channel dimension. As the criterion of pruning, we borrow the idea of Liu *et al.*[8], which uses the scale factors in Batch Normalization[9] layers to measure the importance of channels. On this basis we add another

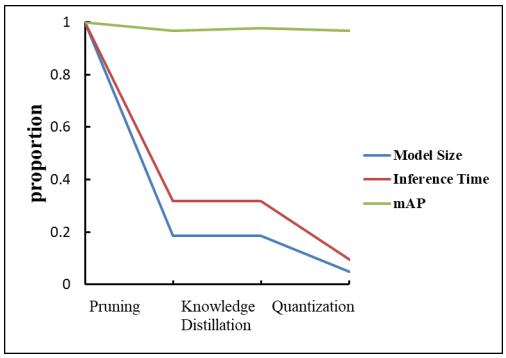
determining factor in each Batch Normalization layer to make the network structure more complete and achieve the purpose of soft pruning. Furthermore, we use an iterative pruning strategy, pruning a small part per time until we get the desired slim model. Experiments show that our pruning strategy can minimize the accuracy degradation. And next, we use the original network and the pruned network for knowledge distillation. In terms of knowledge distillation, our thought is that both the teacher network and the student network have their characteristics, and the teacher network can also learn from the student network what it cannot learn in a traditional training scheme. In this way, they both can achieve better performance through mutual knowledge distillation. We aim to make this series of methods as general as possible and reduce unnecessary process of tuning hyperparameters. This idea first comes from Zhang *et al.*[10], but that strategy is only for image classification. However, the loss function of object detection network is much more complicated than that of image classification. We make further optimization and propose a knowledge distillation loss function for one-stage object detection network. Finally, we use the distilled lightweight model for quantization-aware training and convert it to Int-8 model. The changes of various indicators during the model compression process are shown in **Figure 2**. It can be seen from the figure that the model size and inference time are fully compressed with little sacrifice for the mAP.

In general, the contribution of this paper can be summarized as follows:

- We propose a new channel pruning optimization strategy, which ensures pruning can achieve better performance without destroying the structure of original network.
- We propose a general knowledge distillation method for object detection model, which can alleviate the accuracy drop caused by pruning.
- We propose a model compression pipeline for one-stage object detection network to meet the real-time requirements. For other one-stage networks such as DSSD[11], RefineDet[12], M2Det[13], etc, we can still use the pipeline to facilitate the deployment of them.



(a) SSD



(b) YoloV3

Fig. 2: Changes of model size, inference time and mAP during the process of model compression.

2 Related Work

Object Detection Model. The object detection network can be divided into two types. One is the two-stage type network. This type is mainly represented by RCNN[14], Fast-RCNN[15], and Faster-RCNN[16]. The two-stage network first extracts candidate frames that may have objects, and then fine-tunes the bounding boxes on the candidate frames and determine the classification result. The accuracy of this algorithm is generally higher, but inference time is longer and does not meet our real-time needs. Another type is the one-stage type network, which is represented by SSD and Yolo[17]. Later, Redmon *et al.* proposed YoloV2[18] and YoloV3[6] on the basis of Yolo. The classification of this network and the regression of the bounding box proceed simultaneously. This algorithm can achieve higher speed. On the premise that the backbone is VGG16, the weight storage space of Faster-RCNN exceeds 500M, while neither SSD nor YoloV3 exceeds 150M. In addition, the speed of Faster-RCNN is only one-fifth of SSD and the accuracy is not as good as SSD and YoloV3.

Pruning. Han *et al.*[19] proposed to determine the unimportant connections in the trained network according to the size of the neuron connection weight to reduce the parameters required by the network. Molchanov *et al.*[20] pointed out that choosing an optimal combination from the weight parameters minimizes the loss of the cost function of the pruned model. Li *et al.*[21] proposed that by using the L1 norm digital evaluation criteria to calculate the convolution kernel, some channels

with smaller L1 norms were cut out, and the conclusion was obtained through pruning experiments. Pruning the VGG-16 network based on the L1 norm obtained 34% accelerating, ResNet101 achieved a 38% acceleration. Hu *et al.*[22] considered neurons with an activation value of 0 to be redundant and proposed a statistical-based method to delete units that output zero values for most different inputs and perform alternate retraining. Wei *et al.*[23] introduced the LASSO regular penalty term and constructed different sparse loss functions through different concrete forms to make the model more sparse. He *et al.*[24] pointed out that "smaller-norm-less-important" is inadequate because the distribution deviation of the norm data must be large enough to ensure that it is in a wide range of distributions. The paper proposed that those filters located at or near the geometric median contain redundant information. Therefore, they can be effectively subtracted and replaced with the remaining filters.

Knowledge Distillation. The currently designed knowledge distillation is mainly applied on the image classification model. Hinton[25] first proposed the concept of knowledge distillation. For the first time, the output of the teacher network is treated as a soft target as part of the total loss (the other part is the cross-entropy corresponding to the hard target) to guide the training of the student network to achieve knowledge transfer. The validity of the knowledge distillation method was verified by experiments. Romero *et al.*[26] proposed adding a supervised learning signal in the middle of the deep neural network model, requiring the middle layer activation response of the student model and the teacher model to be as consistent as possible to achieve the purpose of knowledge distillation. Zagoruyko *et al.*[27] proposed that the soft target to guide student network training can be final output or attention maps. In other words, not only the output probability distribution of the teacher and student network is similar, but also the intermediate activation and The activation of the original image is similar, and the effects of the two knowledge transfers can be superimposed. Xu *et al.*[28] proposed that GAN and Knowledge Distillation can be combined and achieved good results. Yim *et al.*[29] proposed the concept of the FSP matrix, which is the relationship between two layers of feature maps of a convolutional network. The knowledge of the teacher network is extracted in the form of several FSP matrices. To minimize the difference between the FSP matrix of the teacher network and the student network, the knowledge is distilled from the teacher to the students.

Quantization. Vanhoucke *et al.*[30] proposed that the model with the weights set to 8 bits can significantly increase the speed of reasoning and not cause much loss of accuracy. Jacob *et al.*[31] proposed Int-8 quantization and introduced a quantization method that uses only integer operations, which is more efficient than floating-point operations. Raghuraman[32] introduced Google's research on model quantification. And Post-quantification and Quantization-aware training were systematically explained for the first time. After quantization, the model takes up 75% of storage space. If the model consists mainly of convolutional layers, the execution speed is increased by 10-50%. Reduced memory and computing power requirements also mean that the power consumption of most models will be significantly reduced.

3 Compression Pipeline

In this section, we elaborate on the compression algorithm, including Pruning, Knowledge Distillation, and Quantization.

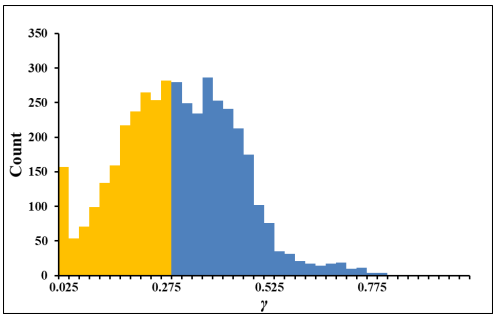
3.1 Pruning

Just as that paper by Liu[8], due to the role of Batch Normalization layers in neural network, the absolute value of scale factor γ of Batch Normalization layers can reflect the importance of each channel. Here, we use some actual statistics to intuitively reflect the impact of γ on the channels of feature maps.

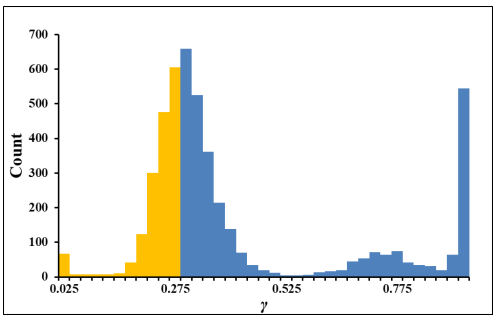
As shown in **Figure 4**, we select the 11th Batch Normalization layers in the SSD network for statistics and comparison. We use the L1 norm to quantify and analyze each channel of feature map. It can be seen that the L1 norm of feature map obtained after the 11th convolution layer is hardly distributed near 0. However, due to the role of subsequent Batch Normalization layer in which many scale factors are distributed around 0, the feature map has a large number of channels with a minimum L1 norm. According to our statistics chart of all channels in the SSD network, as shown in **Figure 3**, it can be seen that there are at least 1/3 of scale factors whose absolute values are less than 0.2, and we can start pruning from this part. We list absolute values of all scale factors in the entire network as G:

$$G = \{|\gamma_1|,\ |\gamma_2|, ..., |\gamma_n|\} \tag{1}$$

where n represents the number of all channels in the entire network. In G, we can find a relatively small threshold by setting a pruning rate and define it as Θ. The scale factor whose absolute value is smaller than Θ can be pruned.



(a) SSD.

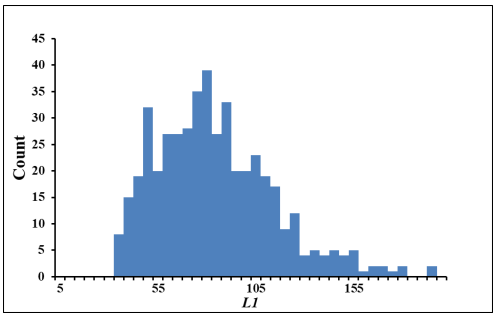


(b) YoloV3.

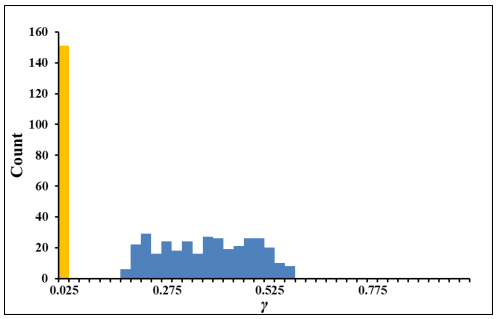
Fig. 3: Distribution of all the scale factors in SSD and YoloV3. The horizontal axis represents the absolute value of the scale factor. The vertical axis represents the number of a certain scale factor. The yellow part indicates that the scale factor is less than Θ, which should be pruned. The following figure is the same.

In order to obtain a better fitting ability, we use an iterative pruning strategy. After multiple iterations of pruning and fine-tuning, the slim network we achieve meets our expectations and we can also acquire a relatively smaller loss of accuracy. This method is proved to be effective in our experiment.

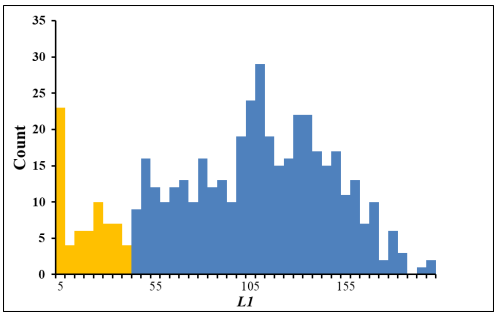
Considering the uneven distribution of scale factors, some convolutional layers may be cut off as pruning continues. This would cause a great loss of accuracy. As shown in **Figure 5**, original pruning would change the structure of network too much. What we want is not to cut off a certain convolutional layer, but only to reduce part of filters per layer. If we change the structure too much, it would cause a large change in the gradient in terms of the chain derivative rules. Eventually, the model would take a longer time to fine-tune, and the accuracy would decrease too much. In addition, for the scale factors in Batch Normalization layers, we need to not only consider their role globally, but also consider their role in current layer. Here, we set another scale factor threshold list ϑ to keep the integrity of network



(a) Feature maps after Conv11.



(b) Scale factors of the 11th Batch Normalization layer.



(c) Feature maps after BN11.

Fig. 4: The effect of the 11th Batch Normalization layer on feature map.

structure:

$$\vartheta = \{\theta_1,\ \theta_2, ..., \theta_N\} \tag{2}$$

where

$$\theta_i = \alpha \times max(\{\gamma_{i1}, \gamma_{i2}, ...\gamma_{ic}\}) \tag{3}$$

N represents the number of convolutional layers and α is greater than 0 and less than 1. For the i-th Batch Normalization layer, we set another threshold θ_i. For example, we can cut out the channel whose corresponding absolute value of scale factor is not only smaller than the Θ but also smaller than the threshold θ_i. This has two purposes. One is that we can keep at least a small part of channels after pruning to prevent some convolutional layers from being completely cut off. We

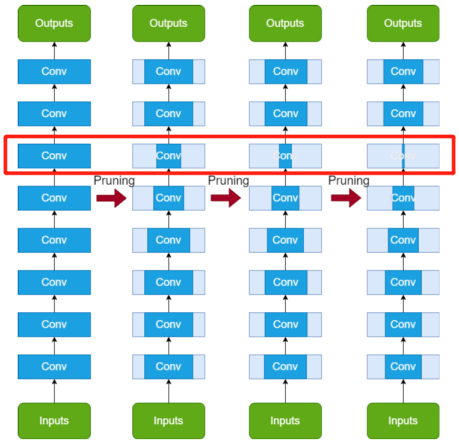


Fig. 5: Problems without ϑ. The length of the blue part represents the number of kernels in convolutional layer. According to the pruning strategy of liu, the 6-th convolutional layer will be completely cut off.

think that the practice of liu causes the relationship between the channels in each layer to be completely destroyed, and our method can alleviate it to some extent. The other is that we can achieve a softer pruning, which improves the accuracy of the pruning network. We verify the effectiveness of this method in experiments. The entire pruning process can be represented by Algorithm 1.

Algorithm 1 Pruning.

Input: *OriginalModel*
Output: *SlimModel*
1: Extract the scale factors of all Batch Normalization layers and list them into $G = \{|\gamma_1|\ |\gamma_2|\ ...\ |\gamma_n|\}$
2: Set the pruning rate and obtain Θ in G
3: Find out the threshold of each BN layer to list into $\vartheta = \{\theta_1\ \theta_2 ...\ \theta_N\}$
4: **for all** γ_j in G **do**
5: Suppose $\gamma_j \in$ the i Batch Normalization layer
6: **if** $\gamma_j < \Theta$ and $\gamma_j < \theta_i$ **then**
7: Keep the channel j
8: **else**
9: Delete the channel j
10: **end if**
11: **end for**
12: Fine-tuning
13: **return** *Slimmodel*;

3.2 Knowledge Distillation

Compared with the original network, the accuracy of the slim model after pruning drops a little. Knowledge distillation is an effective way to make up for the loss of accuracy. The first method of knowledge distillation[25] is to transfer knowledge from a large pretrained network to a small network. In the process of backpropagation of the small network, we can use the following equation to express the final loss function of the small model.

$$Loss_{total} = Loss_{original} + \lambda * Loss_{kd} \tag{4}$$

In the equation, $Loss_{total}$, $Loss_{original}$, $Loss_{kd}$ represent the final loss function, the original loss function and the output difference between the small network and the large network respectively. λ is a weight coefficient that we need to adjust.

The first proposal of mutual learning was from Zhang *et al.*[10]. Mutual learning is the collaborative training of two networks. That is, student network can also guide the training of teacher network. We can use $Net-1$ and $Net-2$ to represent the large network and small network, and the calculation of their total loss in mutual learning is as follows:

$$Loss_{total1} = Loss_{original1} + Loss_{kd1} \tag{5}$$

$$Loss_{total2} = Loss_{original2} + Loss_{kd2} \tag{6}$$

In mutual learning, both $Net-1$ and $Net-2$ participate in the backpropagation process. Compared with the original one-way knowledge distillation, λ in mutual learning is a constant value, which can reduce a lot of tuning time.

In the image classification task, we input $Batchsize$ training data into network and get the output whose shape is $Batchsize \times Classes$. We use cross-entropy as the loss function for optimization. $Kullback-Leibler$ (KL) divergence [33] is often used to describe the difference between two probability distributions. We can input the same data into the two networks and calculate the KL divergence of the output. By using KL divergence as a part of the loss function and backpropagation, we can make the output of two networks tend to be consistent, thus completing the distillation of knowledge. The calculation of KL divergence is as follows:

$$KL(P||Q) = \sum P(x) log \frac{P(x)}{Q(x)} \tag{7}$$

where P and Q represent two different probability distributions and their shape is $Batchsize \times Classes$. We combine KL divergence of the two distributions and original cross-entropy to get total loss.

For SSD, the input is also batch samples, and we can get regression output and classification output. Object detection with SSD is based on prior boxes. We set a series of prior boxes on each pixel of different scale feature maps, and then let each prior box correspond to

coordinate regression and classification. For example, assuming that there are I feature maps for regression and classification output, and the size of the i feature map is $h_i \times w_i$, each pixel of the feature map corresponds to n_i different prior boxes. Then, each prior box on the feature map corresponds to four coordinate values and classification output through convolution. Therefore, the total number of prior boxes is $\sum_i^I h_i \times w_i \times n_i$, and we can use $NumPriorBoxes$ to represent it. Different from image classification task, one image data in SSD corresponds to $NumPriorBoxes$ outputs. The output's shape of classification is $NumPriorBoxes \times Classes$, another is $NumPriorBoxes \times 4$. Even if there are $NumPriorBoxes$ prior boxes in one image data, only positive and negative samples are considered when calculating total loss. The positive sample is the prior boxes whose IOU with GroundTruth is greater than 0.5, and the negative sample is the prior boxes with lower IOU and larger classification loss. Since the coordinates of prior boxes are fixed, the selection of positive samples is always constant for a certain image data regardless of the training state. In contrast, the selection of negative samples is dynamic and related to network parameters. In our experiment, we have found that only distilling knowledge on classification branch can achieve the best effect. Considering that the output of the regression branch is non-discrete, we cannot simply use KL divergence to characterize the difference of their distribution. Chen[34] conducted knowledge distillation on Faster RCNN and proposed a regression-based distillation algorithm, but after testing that method on SSD and YoloV3, we found that the effect is not as good as the effect of knowledge distillation only on classification branch.

In SSD, the definition of total loss can be written as follows:

$$Loss = \frac{1}{N_{pos}}(L_{conf}(p, p_{label}) + \alpha L_{loc}(l, l_{label})) \tag{8}$$

where

$$L_{conf}(p, p_{label}) = \sum_{i=1}^{N_{pos}+N_{neg}} CE(p_i, p_{label}) \tag{9}$$

N_{pos} and N_{neg} denote the number of positives and negative samples. p and p_{label} represent the output of classification and label of each prior box. l and l_{label} represent the output of regression and regression label of each prior box. L_{loc} represents the smooth L1 loss function of the regression part. CE represents the $Cross-Entropy$ loss function. Different from image classification based on each image as the object, the loss function of SSD is based on the prior boxes. Since the choice of positive samples is not directly related to the model, we choose positive samples for knowledge distillation. The negative samples for different networks are different, so we cannot guarantee their consistency. That's the key of knowledge distillation for object detection network.

$$\begin{aligned} L_{conf}(p_1, p_{label}) &= \sum_{i=1}^{N_{pos}+N_{neg}} CE(p_{1_i}, p_{label}) + KL(p_2||p_1) \\ &= \sum_{i=1}^{N_{pos}+N_{neg}} CE(p_{1_i}, p_{label}) + \sum_{j=1}^{N_{pos}} p_{2_j} log(\frac{p_{2_j}}{p_{1_j}}) \end{aligned} \tag{10}$$

$$\begin{aligned} L_{conf}(p_2, p_{label}) &= \sum_{i=1}^{N_{pos}+N_{neg}} CE(p_{2_i}, p_{label}) + KL(p_1||p_2) \\ &= \sum_{i=1}^{N_{pos}+N_{neg}} CE(p_{2_i}, p_{label}) + \sum_{j=1}^{N_{pos}} p_{1_j} log(\frac{p_{1_j}}{p_{2_j}}) \end{aligned} \tag{11}$$

For YoloV3, the loss function of distillation is similar to SSD. The only difference is that YoloV3 has an additional branch of object confidence. So we have to add this branch when designing the knowledge distillation loss function. The calculation formula is consistent with SSD.

A more specific knowledge distillation process is shown in Algorithm 2.

Algorithm 2 Knowledge Distillation.

Input: $OriginalModel(Net1), SlimModel(Net2)$
1: **for all** $iter$ in $Total\ Iters$ **do**
2: Input training data x to two models
$p_1, l_1 = Net1(x)$
$p_2, l_2 = Net2(x)$
3: Calculate total loss of $Net1$ by Equation (5) and (7)
4: Update the weight of $Net1$
5: Re-input same training data to $Net1$
$p_1, l_1 = Net1(x)$
6: Calculate total loss of $Net2$ by Equation (5) and (8)
7: Update the weight of $Net2$
8: **end for**
9: **return** $DistilledOriginalModel, DistilledSlimModel$;

3.3 Quantization

For the over-parameterized neural network model, we can directly perform FP-32 to Int-8 quantization with less accuracy drop. Considering the relatively low robustness of the slim model, we cannot directly convert

it into Int-8 integer model. Quantization-aware training could achieve good performance in lightweight models. We embed fake quantization nodes in identifiable operations. These nodes are used to count the maximum and minimum values of data flowing through this node during training. This training process just simulates the quantization process and we don't implement quantization. Its forward calculation and backpropagation process still use FP-32 for calculation. At the end of training, we convert it into Int-8 model to achieve better performance.

The quantization formula from floating point to the fixed point is as follows:

$$Q = round(\frac{R}{S}) + Z \tag{12}$$

The formula for inverse quantization from fixed point to floating point is as follows:

$$R = (Q - Z) \times S \tag{13}$$

where R is a floating-point value, Q is a quantized fixed-point value, Z is a quantized fixed-point value corresponding to zero floating-point value and S is the minimum scale that can be expressed after fixed-point quantization. *round* is the operation of converting a floating-point type to an integer type. The calculation of S and Z are as follows:

$$S = \frac{R_{max} - R_{min}}{Q_{max} - Q_{min}} \tag{14}$$

and

$$Z = Q_{max} - round(\frac{R_{max}}{S}) \tag{15}$$

where R_{max} represents the largest floating-point value, R_{min} represents the smallest floating-point value, Q_{max} represents the largest fixed-point value (This value is 255 in our method), and Q_{min} represents the smallest fixed-point value (This value is 0 in our method).

In the above calculation process, the *round* step is the core source of quantization loss. We can explain this phenomenon with a simple example. As shown in **Figure 6**, floating point values are linearly mapped to integer values via Equation 9, but there is a deviation in this process which can be reflected when it is mapped back to floating point values. Each convolutional layer has such a deviation, which eventually cause a large decrease in accuracy. In the training process, we insert the quantization node to embed the above process and use the optimizer to optimize the loss due to quantization.

Other than that, each Batch Normalization layer needs to be fused with the convolution layer. The specific algorithm formula is as follows. Changing the weight and offset value makes the new convolution layer not only complete the role of the original convolution but also the work of the Batch Normalization. The calculation of the convolution layer can be expressed as the following formula:

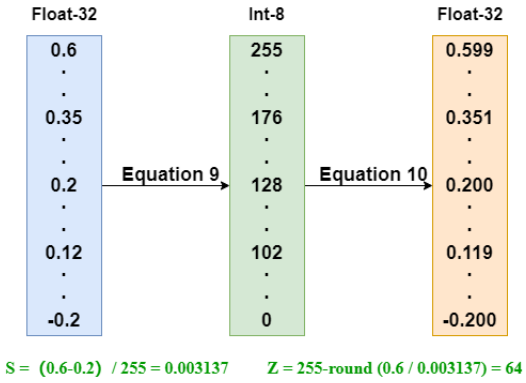


Fig. 6: Example of fake quantization nodes for Quantization-aware training.

$$y_1 = w * x + b \tag{16}$$

where w, x, and b represent the weight, input, and bias of convolutional layer respectively. y_1 represents the output of convolutional layer. Considering the role of the Batch Normalization layer, we can directly get the output after merging as:

$$y = \gamma \times \frac{y_1 - \mu}{\sqrt{\delta^2 + \epsilon}} + \beta \tag{17}$$

$$y = \gamma \times (\frac{(w * x + b) - \mu}{\sqrt{\delta^2 + \epsilon}}) + \beta \tag{18}$$

$$y = \frac{\gamma \times w}{\sqrt{\delta^2 + \epsilon}} * x + \frac{\gamma \times (b - \mu)}{\sqrt{\delta^2 + \epsilon}} + \beta \tag{19}$$

Among them, γ, β represent the scale factor and bias factor of the BN layer. μ and δ represent the mean and variance of y_1. y represents the output of the BN layer. ϵ is to prevent the denominator from being 0 and its value is set to 1e-5. And then, we can get the weight and bias of the final merged convolutional layer.

$$w_{merged} = w \times \frac{\gamma}{\sqrt{\delta^2 + \epsilon}} \tag{20}$$

$$b_{merged} = (b - \mu) \times \frac{\gamma}{\sqrt{\delta^2 + \epsilon}} + \beta \tag{21}$$

The quantization algorithm is shown in Algorithm 3.

Algorithm 3 Quantization.

Input: $DistilledSlimModel$
Output: $Int-8Model$
1: Fold Batch Normalization layers into convolutional layers
2: Insert fake quantization nodes for the Slim Model
3: **for all** *iter* in $Total\ Iters$ **do**
4: Input training samples to the model.
5: Forward propagation.
6: Calculate the total loss.
7: Calculate the gradient and update the parameters.
8: **end for**
9: Convert to Int-8 Slim Model.
10: **return** $Int-8Model$;

4 Experimental Results

In this section, we elaborate on the experiments performed. The object detection networks are SSD and YoloV3, and the backbone is VGG16. We choose VOC dataset as the research object. Our server hardware mainly includes $3 \times NVIDIA\ TITAN\ XP$ GPU (12 GB video memory) and $Intel\ Core\ i7-9700k$ CPU (3.6 GHz). All test parts are carried out on the CPU. Some training parameters are shown in **TABLE 1**.

Table 1: Hyperparameters setting during training

Experiment	Batch Size	Epoch	LR
Original Training	32	1-120	1.E-3
		121-160	1.E-4
		161-200	1.E-5
Pruning (Fine-tuning)	32	1-30	1.E-3
		31-40	1.E-4
		41-50	1.E-5
Knowledge Distillation	32	1-30	1.E-4
		31-50	1.E-5
Quantization-aware Training	32	1-20	1.E-5

4.1 Pruning

As shown in the previous pruning algorithm, firstly we need to train an original model with high accuracy. On this basis, iterative pruning is performed. We only cut a small part at a time, which can minimize accuracy drop. The size of the pruning parts depends on the set threshold Θ in G. In our experiment, we set the pruning ratio of each time as 0.2. The results of the pruning process are shown in the following table.

From **TABLE 2**, we can see that after each pruning, the amount of network parameters is greatly reduced, especially YoloV3. Because the FPN structure of YoloV3 has a high degree of redundancy, the model size of pruned model changes more than SSD. After pruning, model size of YoloV3 is only 20% of original. Considering the mechanical superposition of 3×3 convolutional layers, there are a lot of redundant parts. For such a network with a high degree of redundancy, the method of pruning can cut off unnecessary parts without affecting too much accuracy.

In addition, the reduction of inference time by pruning is obvious. For the sake of fairness, we use CPU for speed measurement. It cound be clearly seen that the inference time of the network also decreases during the pruning process.

As for accuracy, we find that compared to model size, the reduction of accuracy is negligible after the fourth pruning. However, after the fifth pruning, the mAP of both yolov3 and SSD has been greatly reduced while model size does not change much. So we choose the model after the fourth pruning for knowledge distillation and quantization. Also, we mentioned earlier that redundant parameters in the network not only increase the amount of calculation but also affect the accuracy. Whether YoloV3 or SSD, we verify the correctness of this conclusion after the first pruning.

4.2 Knowledge Distillation

We use the pruned network and the original network for knowledge distillation. This process is a little time-consuming. Because normal training only needs one forward propagation and one backpropagation, and knowledge distillation requires three forward propagations and two backpropagations. During network training, we put both networks on the GPU, which undoubtedly requires a lot of video memory. In addition, since we inherit the originally trained parameters before distillation, so it is necessary to ensure that the learning rate is a relatively low value during distillation. The final result is shown in the table below.

From **TABLE 3**, we can see that with the progress of distillation, the accuracy is significantly improved compared to the original, which verifies the effectiveness of the distillation algorithm. Moreover, the accuracy of our original network has also increased significantly, especially YoloV3. After the network distillation after pruning, the mAP has increased by 1.2% compared to the original, which can not be achieved by ordinary distillation methods. The most convenient is that the entire process does not need to adjust hyperparameters.

4.3 Quantization

We insert quantization nodes on the slim model after distillation and then perform quantization-aware training. Considering that the process of the quantization

Table 2: Result of pruning

Model		ModelSize(MB)	RelativeChange	InferenceTime(s)	RelativeChange	mAP(%)	RelativeChange
SSD	Baseline	105.2	0.00%	1.26	0.00%	76.27	0.00%
	Iter-1	77.4	-26.43%	0.97	-23.02%	77.03	1.00%
	Iter-2	62.2	-40.87%	0.73	-42.06%	76.27	0.00%
	Iter-3	52.7	-49.90%	0.55	-56.35%	75.84	-0.56%
	Iter-4	**46.3**	**-55.99%**	**0.485**	**-61.51%**	**75.18**	**-1.43%**
	Iter-5	41.1	-60.93%	0.47	-62.70%	71.78	-5.89%
	Iter-6	39.2	-62.73%	0.46	-63.49%	65.79	-13.74%
YoloV3	Baseline	142.1	0.00%	1.16	0.00%	78.03	0.00%
	Iter-1	97.8	-31.18%	0.68	-41.38%	78.09	0.08%
	Iter-2	68.8	-51.58%	0.56	-51.72%	77.75	-0.36%
	Iter-3	40.3	-71.64%	0.46	-60.34%	76.21	-2.33%
	Iter-4	**26.4**	**-81.42%**	**0.37**	**-68.10%**	**75.44**	**-3.32%**
	Iter-5	17.7	-87.54%	0.32	-72.41%	73.16	-6.24%
	Iter-6	14.7	-89.66%	0.27	-76.72%	70.44	-9.73%

Table 3: Result of knowledge distillation

Model	Network	ModelSize(MB)	Original mAP(%)	Distilled mAP(%)
SSD	Baseline	105.2	76.27	**76.96**
	Pruned	46.3	75.18	**75.65**
YoloV3	Baseline	142.1	78.03	**79.28**
	Pruned	26.4	75.44	**76.21**

and inverse quantization of weights during training requires a lot of computer operation time, so in every training step, quantization-aware training takes longer than ordinary training. After training, we directly convert it to Int-8 model. The result is shown in the following table.

From **TABLE 4**, we can see that the storage size of the final model obtained after quantization is about 1/4 of the original network. This has been greatly reduced in storage size. In addition, comparing to the FP-32 model, our Int-8 model shows a slight decrease in mAP. But comparing to the change in the size of the network storage, this drop is minimal. As for inference time, we can see that the inference speed of both networks has been accelerated by 2-3 times. At the same time, we also conduct a comparative experiment between direct quantization and quantization-aware training, and conclude that the final model trained by quantization-aware training is 4 to 5 percentage points higher. This is exactly in line with our previous thought.

4.4 **Comparative Experiment**

Comparative experiment of pruning. We reproduce other pruning methods on YoloV3 and SSD, including Network-Slimming[8], L1-norm[21] , Rethinking-Pruning[35] and FPGM[24]. Since the above methods were performed on Image classification models, we migrate them to object detection networks. After a series of pruning, we obtain lightweight models and make sure that the parameters of these models are consistent. From **TABLE 5**, we can clearly see that even with the same judgment criteria, the pruned model's mAP obtained by our method is much higher than Network-Slimming. This shows that adding ϑ to the pruning process can not only ensure a more complete network, but also increase the accuracy. Comparing with the latest channel pruning method FPGM, our method also obtains higher accuracy.

Comparison with the original network. The comparison result is shown in **TABLE 6**. Compared with the original model, our model is compressed to the extreme without losing too much accuracy. The final SSD model is compressed to only 11% of original model size, and YoloV3 is compressed to only 4.7% of original. In terms of inference speed, SSD and YoloV3 are only 10% of original. These particularly obvious compression effects are completed with only a loss of up to 3% mAP.

4.5 **Demo**

In order to intuitively show the practicality of this pipeline for model compression, we randomly select three images for testing and make demos as shown in **Figure 7**. There is almost no difference between the original networks and the compressed network in specific tests. However, storage size and inference time of the model we get has been greatly reduced. This effect is what we expect.

Table 4: Result of quantization

Model		ModelSize(MB)	RelativeChange	InferenceTime(s)	RelativeChange	mAP(%)	RelativeChange
SSD	FP32	46.3	0.00%	0.49	0.00%	75.65	0.00%
	Direct-Int8[31]*	11.7	-75.79%	0.17	-65.31%	70.50	-6.81%
	Qat-Int8**	11.7	-75.79%	0.17	-65.31%	**74.70**	**-1.26%**
YoloV3	FP32	26.4	0.00%	0.37	0.00%	76.21	0.00%
	Direct-Int8[31]*	6.7	-74.62%	0.11	-70.27%	70.95	-6.92%
	Qat-Int8**	6.7	-74.62%	0.11	-70.27%	**75.55**	**-0.87%**

* The slim model is quantized to Int-8 model directly without quantization-aware training.
** The slim model is quantized to Int-8 model after quantization-aware training.

Table 5: Comparative experiment of pruning

Model	Method	ModelSize(MB)	RelativeChange	mAP(%)	RelativeChange
SSD	Baseline	105.2	0.00%	76.27	0.00%
	Network-Slimming[8]	46.2	-56.08%	72.87	-4.46%
	L1-Norm[21]	46.2	-56.08%	74.35	-2.52%
	FPGM[24]	48.4	-53.99%	74.17	-2.75%
	Rethinking-Pruning[35]	46.3	-55.99%	60.63	-20.51%
	Our Method	46.3	-55.99%	**75.18**	**-1.43%**
YoloV3	Baseline	142.1	0.00%	78.03	0.00%
	Network-Slimming[8]	26.9	-81.07%	69.85	-10.48%
	L1-Norm[21]	26.2	-81.56%	73.78	-5.45%
	FPGM[24]	30.9	-78.25%	73.85	-5.36%
	Rethinking-Pruning[35]	26.4	-81.42%	64.89	-16.84%
	Our Method	26.4	-81.42%	**75.44**	**-3.32%**

Table 6: Comparison with the original network

Model		ModelSize(MB)	RelativeChange	InferenceTime(s)	RelativeChange	mAP(%)	RelativeChange
SSD	Original	105.2	0.00%	1.26	0.00%	76.27	0.00%
	Compressed	**11.7**	**-88.88%**	**0.17**	**-86.51%**	**74.70**	**-2.06%**
YoloV3	Original	142.1	0.00%	1.16	0.00%	78.03	0.00%
	Compressed	**6.7**	**-95.29%**	**0.11**	**-90.52%**	**75.55**	**-3.18%**

4.6 **Ablation Study**

Comparative experiment of knowledge distillation. We compare our method with the original knowledge distillation method extended by Hinton[25] and Chen's method[34] which considers the distillation of regression. From **TABLE 7**, it can be seen that the accuracy obtained by our method and the method extended by Hinton is higher than Chen's method. This shows that it is better to conduct knowledge distillation only in the classification branch. When we reproduced the other methods, the process of adjusting the hyperparameters is tedious and inappropriate hyperparameters would actually reduce the mAP of the original model. Compared with Hinton's method, we greatly reduce that process on the basis of higher mAP, which is convenient.

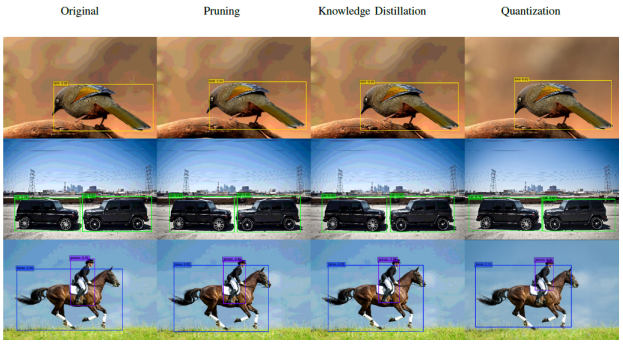
Comparative experiment of different compression strategies. In order to show the effectiveness of our compression pipeline, we compare different model compression strategies. The results are shown in **TABLE 8**. We ensure that the compressed network has almost the same storage size and inference time. From the table, we can find that the direct combination of Network-Slimming pruning algorithm and quantization causes a great loss of accuracy. Using our pruning algorithm can greatly reduce the mAP loss caused by pruning, and then improve the accuracy of the final compressed model. On this basis, we further add knowledge distillation algorithm between pruning and quantization to obtain a lightweight model with stronger fitting ability. The result here represents the thinking process of our attempt to compress one-stage object detection network and the design idea of pipeline.

5 Conclusions

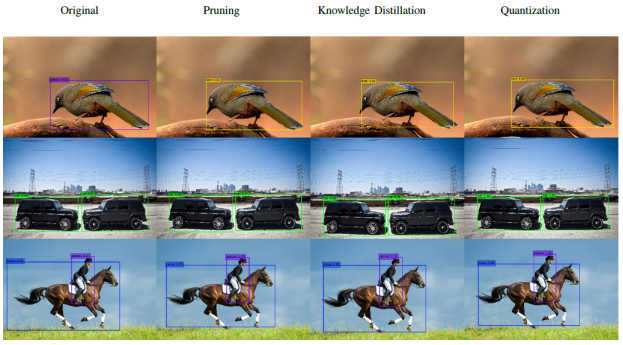
In this work, we proposed a pipeline for compressing the object detection network. This approach includes Pruning, Knowledge Distillation, and Quantization. We pruned the redundant part of the original network and used the mutual knowledge distillation algorithm to compensate for the accuracy decrease. Then, we performed quantization-aware training and got the

Table 7: Comparative experiment of knowledge distillation

Model	Method	Regression	Classification	Mutual	mAP(%)
SSD	Baseline				75.18
	Chen[34]	✓	✓		75.44
	Extend from Hinton[25]		✓		75.55
	Our Method		✓	✓	**75.65**
YoloV3	Baseline				75.44
	Chen[34]	✓	✓		75.67
	Extend from Hinton[25]		✓		76.12
	Our Method		✓	✓	**76.21**



(a) SSD



(b) YoloV3

Fig. 7: Examples of object detection at different moments in the model compression process.

Table 8: Result of different compression strategies

Model	Methods	mAP(%)
SSD	Baseline	76.27
	Network-Slimming[8]+Quantization	71.41
	L1-Norm[21]+Quantization	72.45
	Our Pruning Method+Quantization	74.40
	Our Compression Pipeline	**74.70**
YoloV3	Baseline	78.03
	Network-Slimming[8]+Quantization	68.42
	L1-Norm[21]+Quantization	72.76
	Our Pruning Method+Quantization	75.03
	Our Compression Pipeline	**75.55**

extremely lightweight network. In fact, these three algorithms are independent of each other and we design a pipeline to combine them together. Comparing with the original network, the compressed network has relatively few parameters and less inference time. In addition, if the local equipment has certain competitiveness, we can also combine the compression algorithms at will.

At present, our method is universal for one-stage object detection networks and promotes the application of them to real-time scenarios, such as autonomous driving, pedestrian detection, and so on.

The future research plan is to explore the optimization of the knowledge distillation algorithm of the object network. At present, our knowledge distillation algorithm is mainly developed on the basis of classification and does not take into account the regression branch. If we can propose a knowledge distillation strategy that optimize the regression algorithm, the small model we obtain will possess better performance.

References

1. K. Xu, X. Wang, X. Liu, C. Cao, H. Li, H. Peng, and D. Wang, "A dedicated hardware accelerator for real-time acceleration of yolov2," *Journal of Real-Time Image Processing*, 2020.
2. G. Doménech-Asensi, J. Zapata-Pérez, R. Ruiz-Merino, J. A. Lopez-Alcantud, J. Á. Díaz-Madrid, V. M. Brea, and P. López, "All-hardware sift implementation for real-time vga images feature extraction," *Journal of Real-Time Image Processing*, vol. 17, no. 2, pp. 371–382, 2020.
3. Z. Zhao, X. Kuang, Y. Zhu, Y. Liang, and Y. Xuan, "Combined kernel for fast gpu computation of zernike moments," *Journal of Real-Time Image Processing*, 2020.
4. M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.
5. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.
6. J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
7. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

8. Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.
9. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
10. Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4320–4328.
11. C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "Dssd: Deconvolutional single shot detector," *arXiv preprint arXiv:1701.06659*, 2017.
12. S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4203–4212.
13. Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling, "M2det: A single-shot object detector based on multi-level feature pyramid network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9259–9266.
14. R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.
15. R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 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 *Advances in neural information processing systems*, 2015, pp. 91–99.
17. 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*, 2016, pp. 779–788.
18. J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.
19. S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.
20. P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.
21. H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.
22. H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.
23. W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.
24. Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4340–4349.
25. G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.
26. A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.
27. S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," *arXiv preprint arXiv:1612.03928*, 2016.
28. Z. Xu, Y.-C. Hsu, and J. Huang, "Learning loss for knowledge distillation with conditional adversarial networks," *arXiv preprint arXiv:1709.00513*, 2017.
29. J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning."
30. V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.
31. B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.
32. R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.
33. S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.
34. G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Advances in Neural Information Processing Systems*, 2017, pp. 742–751.
35. Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.