Model Compression Methods for YOLOv5: A Review

Mohammad Jani[a], Jamil Fayyad[b], Younes Al-Younes[a], Homayoun Najjaran[a,1]

[a]*University of Victoria, 800 Finnerty Road, Victoria, V8P 5C2, BC, Canada*
[b]*The University of British Columbia, 3333 University Way, Kelowna, V1V 1V7, BC, Canada*

Abstract

Over the past few years, extensive research has been devoted to enhancing YOLO object detectors. Since its introduction, eight major versions of YOLO have been introduced with the purpose of improving its accuracy and efficiency. While the evident merits of YOLO have yielded to its extensive use in many areas, deploying it on resource-limited devices poses challenges. To address this issue, various neural network compression methods have been developed, which fall under three main categories, namely network pruning, quantization, and knowledge distillation. The fruitful outcomes of utilizing model compression methods, such as lowering memory usage and inference time, make them favorable, if not necessary, for deploying large neural networks on hardware-constrained edge devices. In this review paper, our focus is on pruning and quantization due to their comparative modularity. We categorize them and analyze the practical results of applying those methods to YOLOv5. By doing so, we identify gaps in adapting pruning and quantization for compressing YOLOv5, and provide future directions in this area for further exploration. Among several versions of YOLO, we specifically choose YOLOv5 for its excellent trade-off between recency and popularity in literature. This is the first specific review paper that surveys pruning and quantization methods from an implementation point of view on YOLOv5. Our study is also extendable to newer versions of YOLO as implementing them on resource-limited devices poses the same challenges that persist even today. This paper targets those interested in the practical deployment of model compression methods on YOLOv5, and in exploring different compression techniques that can be used for subsequent versions of YOLO.

Keywords: Model Compression, Pruning, Quantization, YOLO

1. Introduction

As a fundamental problem, object detection has been an active area of research for many years. The main goal of object detection involves identifying and localizing objects of interest from different categories within some given image. Object detection is the basis of many other advanced computer vision tasks [1] ranging from semantic segmentation [2] to object tracking [3] to activity recognition [4]. In recent years, deep learning-based approaches such as Convolutional Neural Networks (CNNs) have achieved state-of-the-art performance in object detection tasks. As a result of the advancements in computational power and cutting-edge algorithms, object detection has become more accurate, enabling a wide range of real-world applications. Compared to classical object detection methods, using CNNs alleviates the problem of feature extraction, classification, and localization in object detection [5, 6, 7, 8, 9, 10].

Typically, Object detection can be done through two methods, namely, single-stage and two-stage detection. While in the former, the algorithm directly predicts the bounding boxes and class probabilities for objects, in the latter, the algorithm first generates a set of region proposals and then classifies those proposals as objects or backgrounds [11]. Unlike Faster R-CNN [6] and R-FCN [12] as two-stage object detection methods, single-stage ones, like YOLO [5], SSD [13], EfficientDet [14], and RetinaNet [15], typically use one Fully Convolutional Neural Network (FCN) to detect objects' classes and spatial locations without intermediate steps.

[*]Corresponding Author
Email addresses: mhmdjani@uvic.ca (Mohammad Jani), jfayyad@mail.ubc.ca (Jamil Fayyad), yalyounes@uvic.ca (Younes Al-Younes), najjaran@uvic.ca (Homayoun Najjaran)

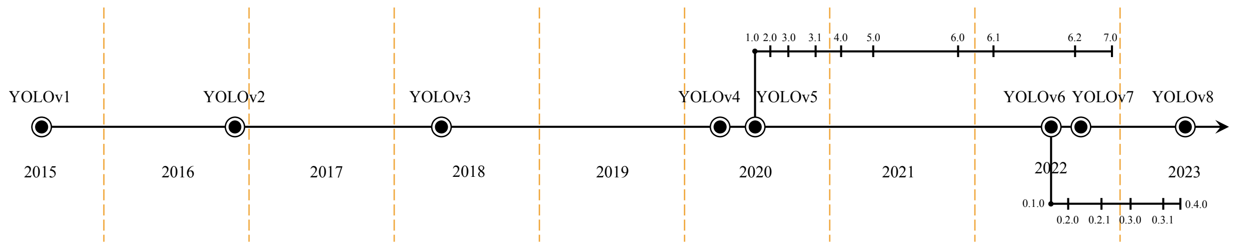


Figure 1: **YOLO release timeline.** YOLOv5 and YOLOv6 have ten and six released variants, respectively.

Among different single-stage object detection methods, YOLO has gained a lot of attention since it was published in 2016. The primary idea behind YOLO is to divide an input image into a grid of cells and predict bounding boxes and class probabilities for each cell. YOLO treats object detection as a regression problem. Also, since it uses a single neural network for object detection and classification, it can be optimized jointly for both tasks, which ultimately enhances the whole detection performance. YOLOv1 was equipped with a simple structure containing 24 convolutional layers and two fully connected layers at the end to deliver probabilities and coordinates [5]. Since its introduction, YOLO has undergone several improvements and variations. In 2017, YOLOv2 (aka YOLO9000) was published with improvements in the performance led by using multi-scale training, anchor boxes, batch normalization, Darknet-19 architecture, and a modified loss function [16]. Following that, Redmon and Farhadi introduced YOLOv3 that employs a feature pyramid network, convolutional layers with anchor boxes, Spatial Pyramid Pooling (SPP) block, Darknet-53 architecture, and an improved loss function [17]. Unlike previous versions, YOLOv4 was introduced by different authors. A. Bochkovskiy et al. enhanced YOLO's performance by utilizing CSPDarknet53 architecture, Bag-of-Freebies, Bag-of-Specials, mish activation function, Weighted-Residual-Connections (WRC), Spatial Pyramid Pooling (SPP), and Path Aggregation Network (PAN) [18].

In 2020, Ultralytics introduced YOLOv5 in five different sizes ranging from nano to extra large [19]. YOLO underwent major modifications ranging from new backbone architecture to automated hyper-parameter optimization. In the Backbone, YOLOv5 utilizes a new CSPDarknet53 structure [20] which is constructed based on Darknet53 with added Cross Stage Partial (CSP) strategy. The design of YOLOv5's Neck takes advantage of CSP-PAN [21] and a faster variation of SPP block (SPPF). The output is generated through the Head that uses the YOLOv3's Head structure. The structure of YOLOv5l is illustrated in Figure 2 in which CSPDarknet53 contains C3 blocks which are the CSP-fused modules. CSP strategy partitions the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy. Therefore, the C3 module can effectively handle redundant gradients while improving the efficiency of information transfer within residual and dense blocks. C3 is a simplified version of BottleNeckCSP and is currently used in the latest variant of YOLOv5. For comparison, the design of C3 and BottleNeckCSP blocks are depicted in Figure 3. Overall, these modifications have enabled YOLOv5 to achieve state-of-the-art performance on several benchmarks for object detection, including the COCO dataset. Also, different model size provides the user with the opportunity to choose based on their need. In 2022, YOLOv6 was released by Meituan, featuring enhancements led by Bi-directional Concatenation (BiC) module, an anchor-aided training (AAT) strategy, and a new backbone and neck design [22]. After a very short period, introduced by the original authors, YOLOv7 [23] was a breakthrough. Wang et al. proposed a Bag-of-Freebies, a compound model scaling method, and an extended ELAN architecture to expand, shuffle, and merge cardinality. The Bag-of-Freebies refers to a planned re-parameterized convolution inspired by ResConv[24], an extra auxiliary head in the middle layers of the network for Deep Supervision [25], and a soft label assigner guiding both the auxiliary head and the lead head by the lead head prediction. Lastly, Ultralytics presented YOLOv8 in 2023 with several alternations in the backbone, neck, and head [26]; A C2f module is used instead of C3; a decoupled head provides the output; and the model directly predicts the center of the object instead of the anchor box. While YOLOv6/7/8 are more featured models, our work focuses on YOLOv5 as more well-established studies have been done on it. However, this investigation can be expanded to newer versions of YOLO, especially YOLOv8.

The current trend in using and expanding over-parameterized models leads to higher accuracy; however, the required Floating-Point Operations (FLOPs) and parameters are dramatically increasing [27]. This issue impedes the deployment of complex models on edge devices owing to memory, power, and computational power limitations. To address this matter, one can employ Cloud Computing (CC). However, running complex models on cloud services

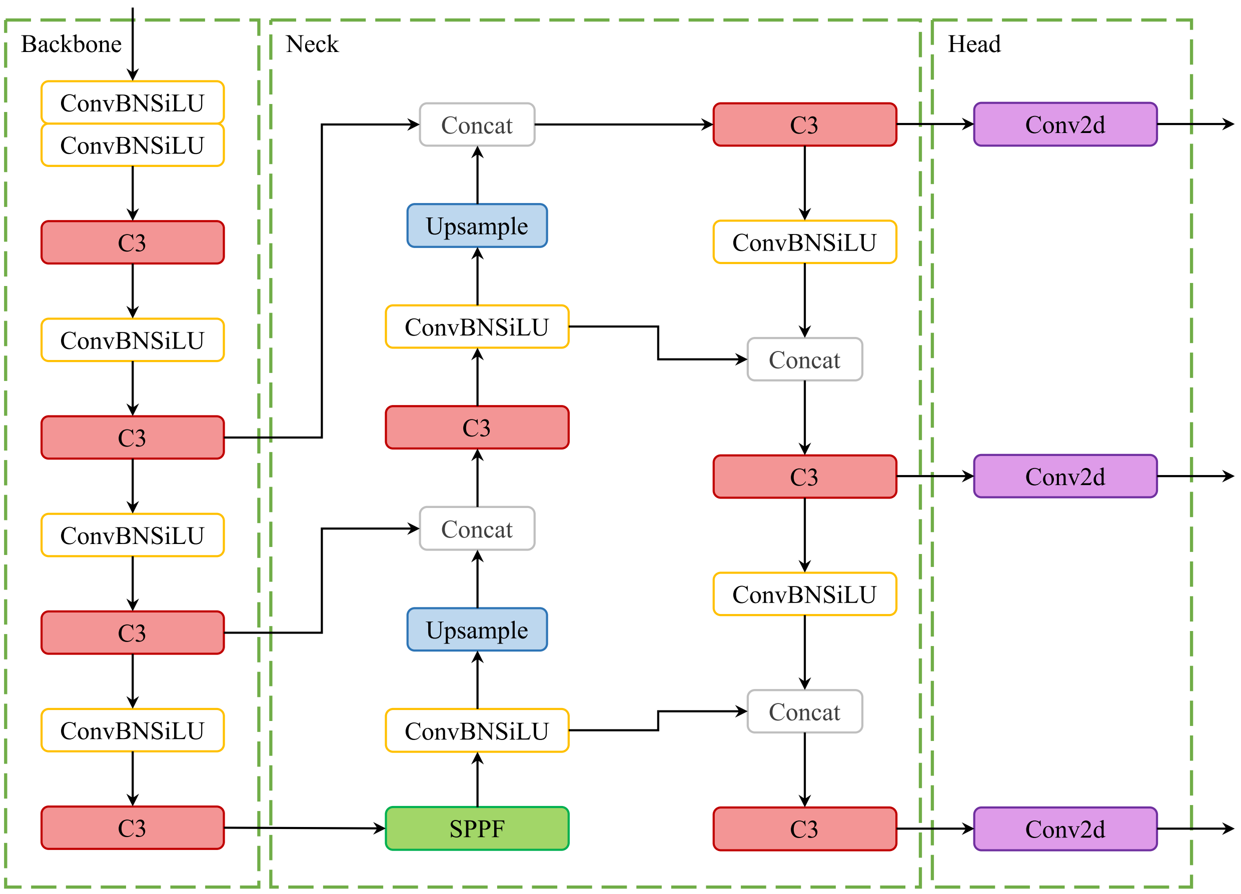
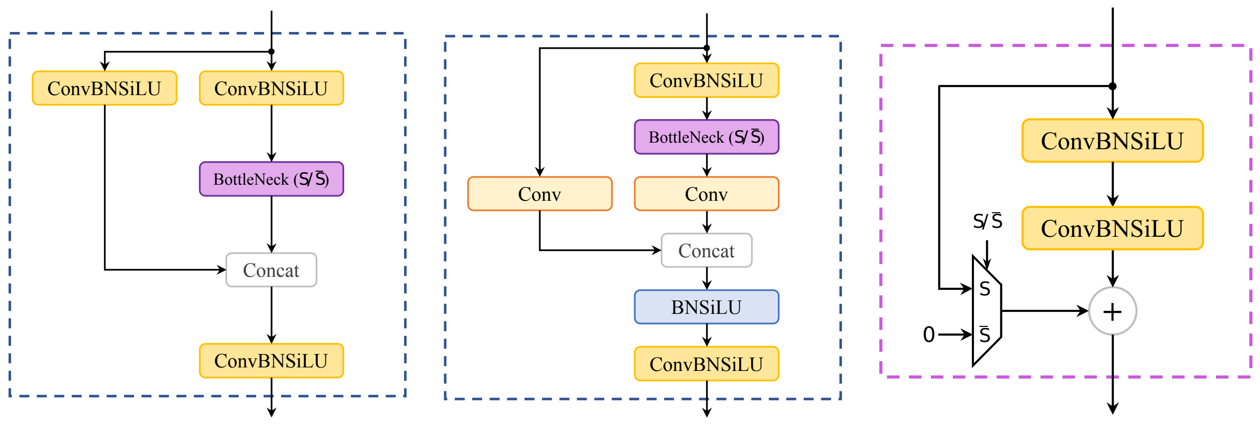


Figure 2: **YOLOv5l architecture.** SPPF represents a computation-efficient version of the Spatial Pyramid Pooling, which was originally implemented in YOLOv3; C3 uses the new CSP-combined module whose details are illustrated in Figure 3.

may not be a feasible option due to 1) the cost of the network: transferring image data to the cloud consumes relatively large network bandwidth; 2) Latency in time-critical tasks: access delay to cloud services is not guaranteed; 3) Accessibility: cloud services rely on the access of devices to wireless communications which can be disrupted in many environmental circumstances [28]. Hence, in many cases, edge computing emerges as a more fruitful resolution. Accordingly, various methods have been introduced to compress neural networks with the purpose of making large models deployable on edge devices. Model compression methods can fall under three categories; pruning, quantization, and knowledge distillation. In pruning, redundant parameters of a model with less importance are removed to obtain a sparse/compact model structure. Quantization involves representing models' activations and weights using lower-precision data types. Finally, knowledge distillation refers to employing a large and accurate model as a teacher to train a small model using soft labels supplied by the teacher model(s) [29, 30].

In this review paper, our focus is on pruning and quantization methods since they are widely used as modular compression techniques while utilizing knowledge distillation requires having two models or modifying the structure of the target network. We review the pruning and quantization methods that have been utilized to compress YOLOv5 in recent years and compare the results in terms of compression terminology. We choose to focus on YOLOv5 as it is the most recent YOLOv5 with enough research on it associated with pruning and quantization. Although the newer versions of YOLO have recently outperformed YOLOv5 in many areas, their applied compression methods are still insufficient to review. Many reviews have been done on neural network compression methods; however, here, we review the actual implementations of such methods on YOLOv5 in real-world situations. We present all the work related to pruning and quantizing YOLOv5, along with their result from different aspects. Generally, the compression results can be expressed regarding changes in memory footprint, power usage, FLOPs, inference time, FPS, accuracy,



(a) C3 structure (b) BottleNeckCSP structure (c) BottleNeck structure

Figure 3: **Structure of C3 and BottleNeckCSP modules**. Using the CSP strategy enables the C3 module to strengthen information flow with residual and dense blocks while addressing redundant gradients. BottleNeck block, which is utilized in C3 and BottleNeckCSP and marked as purple, can have two configurations; $S/\bar{S}$. S denotes the active shortcut connection, while $\bar{S}$ characterizes a simple BottleNeck without any skip connection. C3 blocks of the Backbone use BottleNecks with shortcut connection, whereas those of the Neck does not.

and training time.

The paper is organized as follows: In section 2, various types of pruning techniques are reviewed, and the recent practical research on applying pruning to YOLOv5 is analyzed. Similarly, in section 3 quantization techniques, along with their empirical implementations on YOLOv5, are reviewed. Finally, in section 4, the challenges of compressing YOLOv5 through pruning and quantization are discussed, and possible future directions are provided based on the current gap in this area.

2. Pruning

Neural network pruning was initially proposed in Optimal Brain Damage [31] and Optimal Brain Surgeon [32]. They both rely on a second-order Taylor expansion to estimate parameter importance for pruning. That is, the Hessian matrix should be partially or completely computed in the mentioned methods. However, other criteria can be employed to identify the importance, also called *saliency*, of parameters. Hypothetically, while the best criterion would be an exact evaluation of each parameter's effect on the network, such evaluation is excessively costly to compute. Therefore, other evaluations, including ℓ_n-norm, the mean or standard deviation of feature map activation, batch normalization scaling factor, first-order derivative, and mutual information, can be utilized for saliency analysis. In the following section, we will discuss such saliency evaluation methods. We will not quantify the effectiveness of each scheme here because different works can hardly be compared, and various factors ranging from hyperparameters to learning rate schedules to architecture of implementation impact the results. Instead, we will present the idea behind each criterion and express the results of applying them to compress YOLOv5.

2.1. Saliency Criteria for Pruning

Saliency criteria refer to measures or metrics used to determine the importance or relevance of individual weights, neurons, filters, or a group of weights in a neural network based on certain characteristics or properties of the network. We refer the reader to [33] for a detailed review of saliency criteria.

2.1.1. ℓ_n-norm

Pruning a model based on ℓ_n-norm is the predominantly-used method over the scope of this review paper. Since weight values generally form a normal distribution with zero mean, this is an intuitive approach to select less important individual or structure of weights. The challenge in using this criterion is to define a threshold with which pruning can be performed. Such a threshold can be set statically for the whole network or for each layer. Also, one can

approach it as a dynamic parameter and define schedulers for this threshold. For instance, [34] proposes a method that treats threshold scheduling as an implicit optimization problem and provides threshold schedulers upon using Iterative Shrinkage-Thresholding Algorithm (ISTA).
ℓ_n-norm is usually combined with sparse training of the network to push parameters with the same effect to have similar values [33] (see Section 2.1.3). To do so, ℓ_1 or ℓ_2-regularization is usually added to the cost function, and parameters with a low ℓ_2 norm are pruned after (each step of) training [35, 36].

2.1.2. Feature map activation

When an activation function is used at the end of a layer, its output can be interpreted as the importance of parameters on the prediction. For instance, in case of ReLU function, outputs closer to zero can be considered less salient and be chosen as candidates for pruning. Also, in a broader outlook, the mean or the standard deviation of a tensor of activation can indicate saliency [33, 37].

2.1.3. Batch normalization scaling factor (BNSF)

Although it can be categorized as a fusion between ℓ_1-norm and feature map activation criteria, the BN scaling factor is used predominantly for pruning YOLOv5 and, more generally, for CNNs. Presented by [38], this method introduces a scaling factor γ for each channel and penalizes it during training to obtain a sparse network that can be pruned. The authors proposed the BN scaling factor as the γ needed for network compression. In their method, they penalize γ of the channels using ℓ_1-norm and then prune channels with near-zero scaling factors.

2.1.4. First-order derivative

Unlike the previous criterion, the first-order derivative metrics use the information provided during backpropagation via gradients. This class of criteria may combine information from activations to gradients as expressed in [33, 39].

2.1.5. Mutual information

The Mutual Information (MI) between the parameters of a layer and the prediction or that of another layer can represent saliency. In [40], authors tried to minimize the MI between two hidden layers while maximizing it between the last hidden layer and the prediction.

2.2. Granularity of Pruning

The granularity of pruning defines what kind of parameters of the model are to be pruned. Broadly, pruning can be done either in a structured or unstructured way.

2.2.1. Unstructured pruning

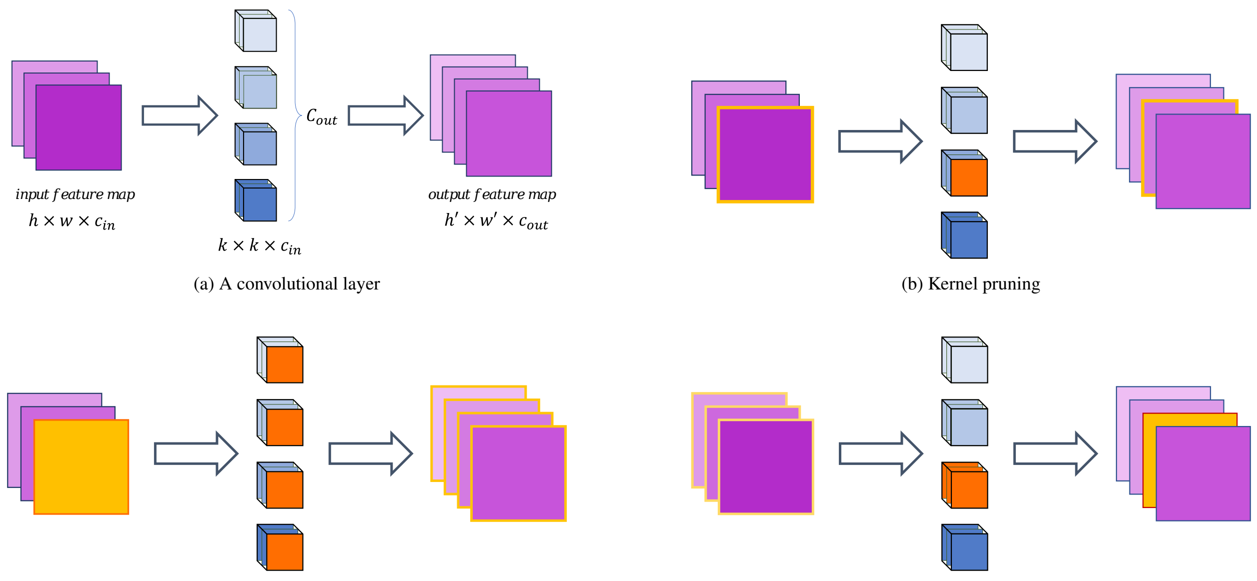
Unstructured or fine-grained pruning refers to when goal parameters for pruning are weights of the model without considering their location in the associated tensor or layer. In weight pruning, unnecessary weights are identified through saliency evaluation and masked out or removed afterward. Since eliminating the weights may impair the model architecture, they are mostly masked out during this process [37]. While masking out the weights instead of removing them increases the memory usage during training, the information of the masked weights can be used at each step to compare the pruned model with the original one. fine-grained pruning is not always beneficial as it requires special hardware to take advantage of such irregular sparse operations [41]. While higher compression ratios could be achieved using unstructured pruning, storing the index of pruned weights may result in higher storage usage [42, 43].

2.2.2. Structured pruning

Unlike the previous category, model weights can be pruned based on their structure. Structured pruning observes patterns in the weights tensors when evaluating their importance so that they can be described with low indexing overhead, such as strides or blocks. In a convolutional layer, j-th channel is obtained through convolving the j-th filter with the input feature map. So groups of parameters such as filters, channels, or kernels can be selected for structured pruning. Figure 4 depicts the difference between these structural pruning paradigms.

Channel-based pruning. It aims at removing the weight filters leading to the *j*-th channel(s) of the output feature map in each layer. Many existing channel pruning techniques utilize the ℓ_2-norm as the criterion for identifying the least important tensor of weights. However, there is a dispute regarding the impact of this process on the overall model structure. In the [44], the authors stated that the process of channel pruning resulted in little damage to the model structure. Conversely, in the [45], it was observed that channel pruning led to a drastic change in the network structure. Nonetheless, it is possible to mitigate structural damage by masking the parameters rather than completely eliminating them. However, this approach may not result in any savings during training since the entire model needs to be stored in memory.

Filter-based pruning. Filter-based pruning eliminates the weights corresponding to the *i*-th channel of the input feature map. That is, pruning a specific kernel, *i*-th, of all filters in a convolutional layer. The structural damage is minimal in this pruning method, and the model can be treated similarly to the original one since the number of output channels remains intact [27]. It is worth mentioning that: 1) channel-based pruning at *l*-th layer is equivalent to filter-based pruning at *(l+1)*th layer, and 2) Filter pruning is not equivalent to filter-based pruning. In filter pruning, one or some of the filters of a layer are pruned which can be categorized as channel-based pruning from the granularity perspective.



input feature map
$h \times w \times c_{in}$
$k \times k \times c_{in}$
C_{out}
output feature map
$h' \times w' \times c_{out}$

(a) A convolutional layer

(b) Kernel pruning

(c) Filter-based pruning

(d) Channel-based pruning

Figure 4: **Pruning granularity.** The dimension of a convolutional layer is shown in (a). Kernel, Filter, and Channel-based pruning are illustrated in (b), (c), and (d), respectively. The orange plane characterizes the pruned parameters, and the yellow bounding indicates the affected features.

Kernel-based pruning. In this category, all parameters over a kernel of a filter in *l*-th layer, which connects the *i*-channel of the input feature map and *j*-th channel of the output feature map, are pruned [46]. This pruning granularity would not impair the model structure. We refer the interested reader to [27] for a thorough survey of pruning granularity.

Regardless of the pruning granularity and the saliency criteria, the pruning process can be performed either in a one-shot manner or iteratively. In one-shot pruning [47, 48, 49], less salient parameters are removed/masked prior to or after training. In post-training pruning, network performance may drop permanently, while iterative pruning, takes the performance drop into consideration and re-trains the network. Compared to one-shot pruning, although iterative pruning is more computation- and time-consuming, it can prevent the accuracy drop or even enhance it in some cases. Moreover, some methods modify the network cost function, such as adding regularization [38, 50], to make the model more suitable for pruning. Consequently, they cannot be used as post-training pruning.

Table 1: Experimental results of pruning applied on YOLOv5 categorized by granularity. Each granularity method is specified by a color. ΔAcc represents the absolute difference in $mAP^{0.5}$ before and after pruning. The rest of the quantitative results represent relative changes. Also, ↻ denotes the iterative pruning, ⊙ denotes the one-shot pruning, and Δ◷ refers to the changes in the inference time.

Paper	V	Task	Dataset	Saliency	Granul	↻/⊙	ΔAcc	ΔParam	ΔSize	ΔFLOPs	ΔFPS/Δ◷
Unstructured Pruning											
[51]	5s	Object Detection	—	SPDY	unstructured	↻	-0.5	—	—	—	50/—
[51]	5m	Object Detection	—	SPDY	unstructured	↻	-1.7	—	—	—	75/—
Channel-Based Pruning											
[52]	5s	Position Detection	Manual	BNSF	channel	↻	-2.3	-45	-48	-55	—/-40
[53]	5n	Fault Detection	Manual	BNSF	channel	↻	-4.8	-72.2	—	—	—/-29.4
[54]	5	Target Detection	Military Aircraft Detection	BNSF	channel	↻	-4.5	—	—	—	560/—
[55]	5s	Fruitlet Detection	Manual	BNSF	channel	↻	0(F1)	-92	-90.5	—	—/-13
[56]	5	Outdoor Obstacles Detection	OBSTACLE	BNSF	channel	↻	-0.4	—	-59.1	—	—/-43.6
[57]	5s	Garlic Detection	Manual	BNSF	channel	↻	-0.1	—	-17	—	—/-1
[58]	5s	Wheat Grain Quality Detection	Manual	—	channel	↻	1	—	—	-86.7	—/—
[59]	5x	Ship Detection	Manual	BNSF	channel	↻	2.3	—	-48.9	—	61.4/—
[60]	5s	Flame Detection	Manual	BNSF	channel	↻	0	-37.8	-37.5	-54.5	10/—
[61]	5s	Satellite Components Recognition	Manual	BNSF	channel	↻	-1.2	—	-66.2	—	—/—
[62]	5s	Helmet-Wearing Detection	—	BNSF	channel	↻	-0.9	-87.3	-86.2	-87.4	53.5/—
[63]	5l	Object Detection	COCO	CDSC-BNSF	channel	↻	-0.9	—	-63.8	-37.4	—/—
[64]	5s	Sewer Defect Detection	Manual	BNSF	channel	↻	-0.5	-81	-79.3	-48.8	—/-34.2
[65]	5s	Tracking and Counting Grape Clusters	Manual	BNSF	channel	↻	-0.2	-73.3	-76.4	-57.6	—/-10.3
[66]	5	Fiber Defect Detection	PASCAL VOC	BNSF	channel	↻	-0.3	—	-29.5	—	—/-18.7
[67]	5s	Pedestrian Detection	VisDrone	BNSF	channel	↻	-0.4	—	-26.3	-11.9	—/-9.3
[68]	5	Blade Defect Detection	Manual	BNSF	channel	↻	7.8	—	-19.6	—	28.3/—
[69]	5l	Object Detection	PASCAL VOC	CRLST	channel	↻	-2.9	-49.2	—	-46.2	—/-35.5
[69]	5l	Object Detection	COCO	CRLST	channel	↻	-3.8	-48.8	—	-46.5	—/-29.4
[69]	5l	Object Detection	VisDrone	CRLST	channel	↻	-1.1	-51.2	—	-46	—/-35.4
[70]	5s	Pedestrian Detection	Manual	BNSF	channel	↻	3.78	-52.3	-51.8	-40.8	57.8/—
[71]	5s	Aerial Image Object Detection	DOTA	BNSF	channel	⊙	-6.46	-58.8	-57.6	-37	—/-17.3
[72]	5l	Object Detection	Manual	NSGA-II	channel	↻	-0.9	-73.9	-73.5	-62.7	59/—
[50]	5m	Object Detection	VisDrone	BNSF & ASR	channel	↻	0.4	-91.2	—	-65.8	—/-65.7
[50]	5s	Object Detection	VisDrone	BNSF & ASR	channel	↻	-0.5	-93.4	—	-74.3	—/-53.2
[73]	5	Tomato Maturity Detection	Manual	—	channel	—	-1.5	-78	-76.4	-84.1	183/-64.9
[74]	5s	Railway Defect Detection	Manual	FPGM	channel	↻	2.19	—	—	—	34.7/—
Hybrid Pruning											
[75]	5l	Object Detection	Manual	—	kernel/layer	—	1.27	—	-86.8	—	—/-58.4
[76]	5s	Fruit Detection	Manual	BNSF	channel/layer	↻	-1.57	—	-72.5	—	32.5/—

2.3. Recent Applied Studies on Pruned YOlOv5

Table 1 represents the recent experimental pruning results on YOLOv5 categorized by the pruning granularity. [51] concentrates on achieving a desired inference time instead of a specific compression ratio. It presents a pruning method, *learned efficient Sparsity Profiles via DYnamic programming search* (SPDY), that can be utilized in both one-shot and iterative schemes. In [72], researchers implemented an algorithm based on Non-Dominated Sorting Genetic Algorithm (NSGA) II that tackles pruning as an optimization problem. That is, how to prune the channels in order to minimize GFLOPs and maximize the $mAP^{0.5}$. [50] introduces an Adaptive Sparsity Regularization (ASR) that renders sparse constraints based on the filter weights. That is, a penalty is assigned to the filters with weak channel outputs in the regularized loss function, instead of directly regulating the loss with the L1-norm of the batch normalization scaling factor. After training, filters with a scaling factor smaller than a global threshold are pruned for all layers, and find-tuning is performed to retrieve the accuracy drop. The work of [52] modifies the YOLOv5 structure by utilizing the PReLU activation function and using Ghost Bottleneck instead of BottleNeckCSP. Thereafter, it prunes the channels, except Ghost Bottleneck, with small batch normalization scaling factor based on the BNSF method (see Section 2.1.3). [61] proposed a feature fusion layer and selective kernel network to improve the channel and kernel attention of the model. It attaches transformer encoder modules to the outputs of the PAN neck to explore the potential of the model's prediction via a self-attention procedure and compresses the model using the BNSF approach before deployment on NVIDIA Jetson Xavier NX. Furthermore, [69] targets a desired number of parameters and FLOPs and utilizes a computation-regularized loss sparse training (CRLST). After sparse training, it iteratively prunes the channels based on their batch normalization scaling factor. Both compressed YOLOv5 models in [69, 61] were deployed on NVIDIA Xavier NX. [74] treats filters as points in space and adopts a Filter Pruning Geometric Median (FPGM) approach to prune filters of a convolutional layer that, unlike ℓ_n-norm criteria, explicitly exploits the mutual relations between filters. It computes the geometric median of a whole layer's weights and prunes the filters which are regarded redundant if their geometric median is close to the geometric median of the layer. [63] discards the upsample, concatenate, and detect layer and prunes the filters similar to the BNSF method, yet it embeds the cosine decay of sparsity coefficient (CDSC-BNFS) and uses a soft mask strategy. [67, 68] makes the model lighter by BNSF channel-based pruning and adds another upsampling level with the BottleNeckCSP module to the neck network to extract more semantic information from small objects. The latter also adds a Convolution Block Attention Module (CBAM) to the output of each BottleNeckCSP module in the neck network before feeding them to the head. The paper [76] considers the mean value of each Conv layer before each shortcut in the BottleNeckCSP and compresses the backbone. It also performs channel-based pruning using the BNSF method. In the following, several recent implementations of pruning on YOLOv5 are listed:

- [71, 53] prunes the network through the BNSF method but combines the fine-tuning with knowledge distillation to save training time while maintaining accuracy.
- The Authors in [54] replaced the CSPDarknet backbone with MobileNetV3 [77] and use TensorRT after pruning the filters with the BNSF method.
- The work of [55, 56, 57, 60, 64] focuses on pruning the filters using BNSF strategy and fine-tuning.
- In [58], the backbone is pruned, and since the desired objects have relatively the same size, the largest feature map of the PAN module is removed. Also, in the neck, a hybrid attention module is proposed to extract the most comprehensive feature from channels.
- [59] employs the t-Distributed Stochastic Neighbour Embedding algorithm to reduce the dimension of anchor frame prediction and fuses it with weighted clustering to predict frame size to achieve a more accurate prediction target frame. Afterward, it prunes the filters through the BNSF method.
- [62] utilizes a multi-spectral channel attention mechanism in the backbone to generate more informative features and improve the model's accuracy for detecting small objects. Then, using the BNSF process, it prunes the filters of the model.
- [65] lightens the model through pruning filters with the BNSF criteria and introduces a soft non-maximum suppression which enables the model to detect overlapping clusters of grapes instead of discarding them.

- [66] combines feature maps from different receptive fields using spatial pyramid dilated convolutions (SPDCs) to integrate defect information at multiple scales. It embeds a channel attention mechanism in the neck to direct more attention to effective feature channels after each concatenation operation. Afterward, it compresses the model via BNSF channel-based pruning with fine-tuning.
- [70] makes many modifications to the YOLOv5 structure, such as appending an attention mechanism to the neck and a context extraction model to the backbone. As for pruning, it removes the filters using the BNSF criteria.
- [75] compress the neck and the backbone of YOLOv5 with layer and kernel pruning.
- [73] replaces the backbone of YOLOv5 with MobileNetV3 and prunes the neck network through channel-based pruning.

Almost 85% of the research on pruning YOLOv5 is done using channel-based pruning, and the rest is associated with other structured and unstructured granularities. The main saliency criterion used for pruning is the BNSF sparse training method which is employed in around 60% of the surveyed papers in our scope, while the rest employed ℓ_1-norm or ℓ_2-norm or proposed a new saliency criterion.

3. Quantization

Neural network quantization aims to represent the weights and activations of a deep neural network with fewer bits than their original precision, which usually is 32-bit single-precision floating-point (FP32). This process is done while the effect on the model's performance/accuracy is kept minimal. Quantization, by exploiting faster hardware integer instructions, can reduce the size of the model and improve inference time [78]. In [30], Gholami et al. surveyed different aspects of neural network quantization, which includes the theoretical details of this topic. Here, we will briefly introduce and discuss the key points.
Without loss of generality, we explain the concepts of quantization on a real finite variable which can represent weights or activations in a neural network. Assuming $r \in \mathbb{R}$ is a finite variable limited to the range of $\mathbb{S}$, we want to map its values to q with a set of discrete numbers that lie in $\mathbb{D} \subset \mathbb{R}$. Before mapping, we may want to clip the range of input r to a smaller set of $\mathbb{C} \subseteq \mathbb{S}$.

3.1. Quantization Interval: Uniform and Non-uniform

Uniform quantization maps r into a set of evenly spaced discrete values, while in non-uniform quantization, the distance between discrete values is not necessarily equal. Through non-uniform quantization, one can better capture the vital information from the weights and activation distributions because, for instance, more closely-spaced steps can be allocated to denser areas of distribution. Consequently, although employing non-uniform quantization requires more design than the uniform approach, it may achieve a lower accuracy drop [79]. Also, since the distribution for weights and activations generally tends to be bell-shaped with long tails, non-uniform quantization can lead to better results [80, 81, 82]. Figure 5 demonstrates the disparity between the aforementioned quantization schemes.

3.2. Static and Dynamic Quantization

For a set of inputs, the clipping range $\mathbb{C} = [a, b]$, where $(a, b) \in \mathbb{R}$, can either be determined dynamically or statically. The former computes the clipping range dynamically for each input, whereas the latter uses a pre-calculated range to clip all the inputs. Dynamic quantization reaches higher accuracy compared to static one, but the computational overhead is significantly high.

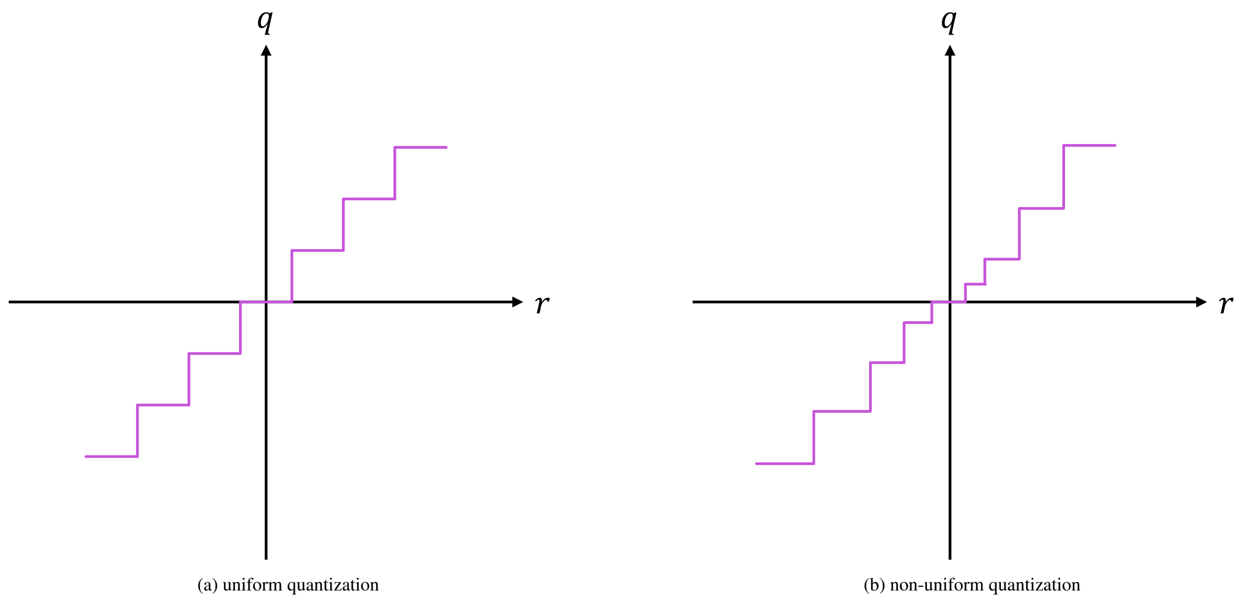


Figure 5: **Quantization Interval.** (a) indicates the uniform quantization, while (b) depicts the non-uniform quantization in which neither the interval in the real number nor the quantization steps are evenly distributed.

3.3. Quantization Scheme: QAT and PTQ

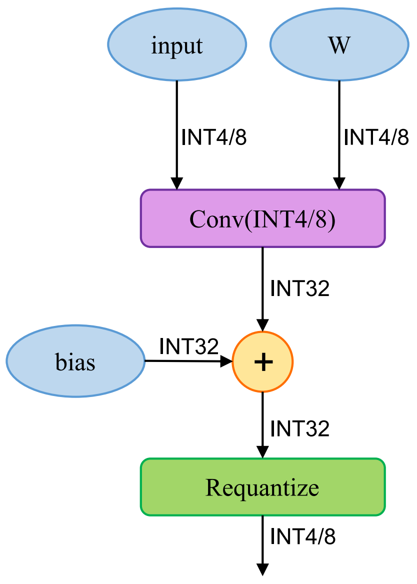
Quantizing a trained model can negatively impact the accuracy of the model due to cumulative numerical errors. To recover this drop in performance, network parameters often need adjustment. Therefore, quantization can be performed in two fashions; Quantization Aware Training (QAT), which refers to retraining the network, or Post Training Quantization (PTQ), which does not include re-training. In QAT, the forward and backward passes of the quantized model are performed in floating points, and network parameters are quantized after each gradient update. On the other hand, PTQ performs quantization and parameter adjustment without re-training the network. Compared to QAT, this method usually suffers from the model's accuracy degradation, but its computational overhead is considerably lower. Generally, PTQ uses a small set of calibration data to optimize the quantization parameters and then quantizes the model [83]. As PTQ relies on minimal information, it is often impossible to achieve lower than 4 or 8 bits precision while maintaining accuracy [84].

3.4. Quantization deployment scheme

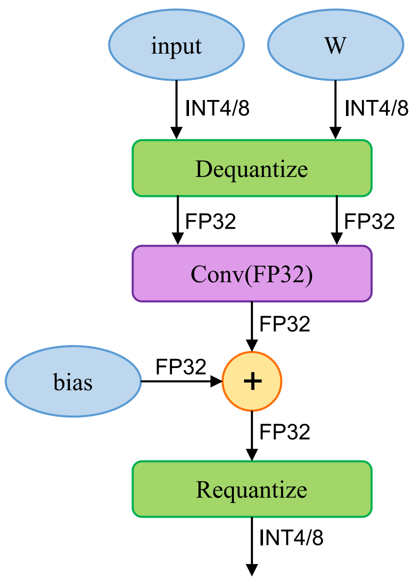
Once the model is quantized, it can be deployed using *fake quantization* (also called simulated quantization) or *integer-only* quantization (also known as fixed-point quantization). In the former, weights and activations are stored in low precision, but all the operations ranging from addition to matrix multiplication are performed in floating point precision. While this approach requires constant dequantizing and quantization before and after floating point operations, it favors the model's accuracy. However, in the latter, operations, as well as weights/activation storing, are performed using low-precision integer arithmetic. In such a fashion, the model can take advantage of fast integer arithmetic enabled by most hardware. Figure 6 illustrates the disparity between PTQ and QAT deployment for a single convolutional layer.

3.5. Recent Applied Studies on Quantized YOLOv5

Table 2 represents the recent practical quantization results on YOLOv5 categorized by quantization scheme. The Authors in [85], proposed a QAT method that dynamically selects rounding mode for weights considering the direction of weight updates during training and adjusts the corresponding gradient. They quantized the model layer-wise while adopting symmetric/asymmetric clipping ranges for the weights and activations of the network, respectively.



(a) Fixed-precision quantization



(b) Simulated quantization

Figure 6: Inference comparison between fixed-precision and simulated quantization in a convolutional layer.

Noise Injection Pseudo Quantization (NIPQ) [86], as a QAT method, initially pre-trains a network with pseudo quantization noise and then quantizes the model in post-training. This approach automatically adjusts the bit-width and quantization interval while also regularizing the sum of the Hessian trace of a neural network. The authors evaluated their method on YOLOv5 and achieved down to 3-bit precision without major degradation in accuracy.
Furthermore, [87] uses ShuffleNetV2 [88] to modify the backbone and reduce the number of layers in the PAN and head network in order to make the model more suitable for mobile devices. It takes advantage of TensorFlow Lite Micro [89] to quantize the weights and activations with 8-bit precision and finally deploys it on an ultra-low-power microcontroller from the STM32 family. The Authors in [90] introduced Deeplite Neutrino, which automatically quantizes a CNN model with lower than 4 bits, and presented Deeplite Runtime as an inference engine, which makes the deployment of ultra-low bit quantized models possible on ARM CPUs. Their QAT method can achieve below 4-bit precision for weights and activations of the network, which was rendered feasible with the design of a custom convolution operator using bitserial computation. That is, the calculation of the dot products of the low bit weight and activation values are done through popcount and bitwise operations. They evaluated their method by means of deploying YOLOv5 on Raspberry Pi 4B. [91] replaces the backbone network with MobileNetV2 [92] and adds a coordinate attention mechanism to it. Before deploying the model on NVIDIA Xavier NX, the authors utilized PyTorch to quantize the model after training through a static scheme with 8-bit precision. Similarly, in [93], Pytorch was employed to quantize the modified YOLOv5 in a fake-quantization fashion with 8-bit precision and static clipping range. The work of [73] uses Nihui Convolutional Neural Network (NCNN) framework to quantize the compressed model after training and deploys it on an actual mobile device with a MediaTek Dimensity processor. [94] proposes a log-scale quantization method that rescales the distribution of activations so that they are suitable for log-scale quantization. This approach minimizes the accuracy drop in YOLOv5 due to log-scale quantization.

Overall, more than half of the reviewed papers used the QAT scheme, which resulted in low precision quantization down to 3 bits. However, none of the PTQ schemes have reached below 8-bit precision. While there exists more research on YOLOv5 conducted using quantization, the focus of the review is to primarily include those with novel quantization methods. Therefore, we excluded the results that merely used TensorRT [95], PyTorch Quantization [96], and ONNX quantization [97] in their implementation.

Table 2: Experimental results of quantization applied on YOLOv5 categorized by deployment scheme. Each scheme method is specified by color. ΔAcc represents the absolute difference in $mAP^{0.5}$ before and after pruning. The rest of the quantitative results represent relative changes. Δ⏲ refers to the changes in the inference time.

Paper	**V**	**Task**	**Dataset**	**Symmetry**	**Interval**	**St/Dyn**	**Fake/IntOnly**	**Precision**	**ΔAcc**	**Δsize**	**ΔFLOPs**	**ΔFPS/Δ⏲**
PTQ												
[73]	5s	Tomato Detection	Manual	—	—	—	—	FP16	-0.9	-51.1	—	18.8/—
[73]	5s	Tomato Detection	Manual	—	—	—	—	Int8	-5.75	-73.7	—	-5/—
[98]	5	Human Pose Estimation	COCO	—	—	—	—	8/16 bits	-1.3	—	—	—
[91]	5	Infrared Object Detection	FLIR ADAS22	—	—	static	—	Int8	-1.1	-75	—	40/—
QAT												
[93]	5s	Infrared Ship Detection	Manual	Asym	uniform	static	fake	Int8	-5	-27	-33	—
[85]	5s	Object Detection	MS-COCO	Asym act/ Sym w	uniform	dynamic	IntOnly	Int4	-3.1	—	—	—
[86]	5s	Object Detection	PASCAL VOC	Asym	uniform	dynamic	—	5W/5A	-1.2	—	—	—
[86]	5s	Object Detection	PASCAL VOC	Asym	uniform	dynamic	—	4W/4A	-2.5	—	—	—
[86]	5s	Object Detection	PASCAL VOC	Asym	uniform	dynamic	—	3W/3A	-5.8	—	—	—
[87]	5	Crowd Counting	Manual	Asym act/ Sym w	—	—	—	Int8	-14	-87.5	—	—/-92.7
[90]	5n	Object Detection	COCO-8class	Asym	uniform	—	bitserial	<Int4	-1	—	—	—/-60.6
Unspecified												
[94]	5	—	—	Asym	nonuniform	—	fake	Int8	-0.9	—	-89.2	—
[99]	5	Object Detection	COCO	—	—	—	—	FP16	—	—	—	—/-60

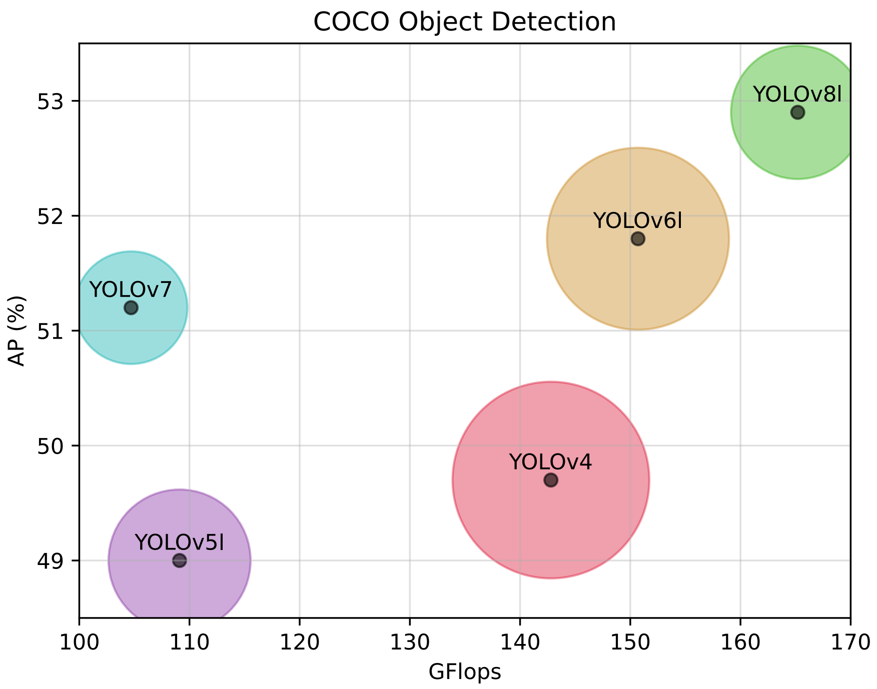


Figure 7: **Performance characteristic of four different YOLO versions.** The horizontal and vertical axes represent the number of GFlops and the average $AP^{0.5}$ on the COCO 2017 validation dataset with an input size of 640. The radius of circles denotes the relative size of the models.

4. Conclusions

Model compression methods have gained significant attention in recent years, and their applications are becoming more specific. In this paper, we presented a review of the commonly used pruning and quantization approaches applied to YOLOv5 and categorized them from different aspects. Through our analysis, we identified gaps in adapting pruning and quantization methods specifically for YOLOv5. This matter is of importance since the application of those compression methods to YOLOv5 requires further exploration and optimization due to many structural interconnections. Our review serves as a resource for researchers interested in the practical deployment of model compression methods, specifically pruning and quantization, on YOLOv5 and its subsequent newer versions. With the advent of YOLOv8, YOLO has experienced a new borderline in terms of performance, shown in Figure 7, which pushes the boundary of object detection and establishes a new SOTA performance. However, the reduction in terms of size and FLOPs is relatively insignificant which necessitates the need for applying pruning and quantization on newer YOLOs when the limitation in hardware is pivotal. Therefore, our review is extendable to subsequent versions of YOLO and can be used as a guideline for researchers interested in compressing any version of YOLO.

4.1. Pruning Challenges and Future Directions

Unlike a regular CNN, pruning YOLOv5 comes with some challenges due to its complex and highly optimized deep neural network architecture. YOLOv5 uses CSP-Darknet53 neural network architecture as the backbone and PANet as the neck, which both consist of many convolutional layers that are tightly interconnected with concatenations. Also, the interconnections between the backbone and the neck add to the complexity of the model. Overall, the structural complexity of these layers hinders the process of removing unnecessary filters without adversely affecting the overall structure performance of the network. Otherwise, the spatial resolution of feature maps related to concatenations would not match. Therefore, some compensations should be made prior to pruning the YOLOv5. For instance, [63] does not consider pruning the upsampling layer, the concatenate layer, and the head of the YOLOv5.

Also, it ignores the shortcut connection in the BottleNeck module to allow the inputs to have a different number of channels. In this regard, more studies should consider filter-based and kernel-based pruning because this strategy of pruning does not change the number of output channels; thus, it simplifies the pruning process.
As represented in Table 1, The current research direction is to utilize the BNSF for sparsity training and channel-based pruning with fine-tuning. However, there is a gap in using one-shot pruning using other saliency criteria. Here we introduce some novel approaches which have not been applied to YOLOv5.
EagleEye [100] treats the pruning process as an optimization problem and states that using the evaluated accuracy might not be a promising criterion to guide the selection of pruning candidates. Therefore, it proposes a random pruning ratio for each layer according to the problem constraints, then prunes the filters based on their ℓ_1-norm. It evaluates the impact of pruning candidates through an adaptive BN-based candidate evaluation module using a sub-sample of training data. In [101], authors introduced the *HRank filter pruning* which iteratively prunes the filters with low-rank activation maps. [102] proposes a *mask diversity* evaluation method to correlate between the structured pruning and the expected accuracy. It also introduces a pruning algorithm called *AdaPrune* that compresses an unstructured sparse model to a fine-grained sparse structured model, which does not need re-training. Similarly, [103] proposes *Variational Bayesian pruning* algorithm that considers the distribution of channels' BN scaling factor instead of deterministically using them similar to section 2.1.

4.2. Quantization Challenges and Future Directions

Although not specific to YOLO, quantizing FP32 to INT8 is not a smooth transformation, and it might hinder the optimality of the results if the gradient landscape is harsh. Also, achieving a low-bit (< 4 bits) precision using PTQ is almost impossible as it most likely shatters the performance of the model. Currently, there is a trend in using ready-to-use quantization modules like TensorRT [95], PyTorch Quantization [96], and ONNX quantization [97], which cannot achieve very low precision as they are limited to 8-bit precision. Nevertheless, such research is not included in this review as our focus is to find novel quantization methods employed on YOLOv5.
Regarding applied studies on quantizing YOLOv5, more research is conducted using QAT with a wide range of precision from 1 bit to 8 bits. However, there is a gap in focusing on accelerating training time as well as inference time, especially because training YOLOv5 on a new dataset is computation- and time-consuming. As a solution, more work can be done employing integer-only quantization since hardware throughput is much higher when operations are performed using integer numbers. For instance, TITAN RTX can perform around 23 times more operations per second when the data type is INT4 rather than FP32 [30]. Additionally, PTQ methods still fall back when lower than 8-bit precision is investigated/needed, which presents the opportunity for future work in this area. Hence, we recommend some approaches that can be applied to YOLOv5 in order to fill the above-mentioned gap. In [104], a PTQ algorithm, AdaRound, is proposed to more efficiently round weights for quantization. It accomplishes SOTA performance with as low as 4-bit precision without a noticeable drop in accuracy ($< 1\%$). Yao et al. proposed *HAWQ-V3* [105], a mixed-precision integer-only quantization method that reaches INT4 or INT4/INT8 uniformly mapped quantization. *AdaQuant* [106] proposed a PTQ quantization scheme to minimize the quantization errors of each layer or block separately by optimizing its parameters based on a calibration set. It can attain SOTA quantization with INT4 precision, which leads to a negligible accuracy drop. The authors of [107] presented a method for quantization that exclusively utilizes integer-based operations and eliminates redundant instructions during inference. [108] evaluates the impact of quantization on loss landscape and introduces a novel PTQ approach that can reach 4-bit precision by minimizing the loss function directly, resulting in almost the full-precision baseline accuracy.

Acknowledgements

We would like to acknowledge the financial support of Sentire and Mathematics of Information Technology and Complex Systems (MITACS) under IT31385 Mitacs Accelerate.

References

[1] A. Salari, A. Djavadifar, X. Liu, H. Najjaran, Object recognition datasets and challenges: A review, Neurocomputing 495 (2022) 129–152.
[2] S. Hao, Y. Zhou, Y. Guo, A brief survey on semantic segmentation with deep learning, Neurocomputing 406 (2020) 302–321.

[3] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, F. Herrera, Deep learning in video multi-object tracking: A survey, Neurocomputing 381 (2020) 61–88.

[4] S. González, J. Sedano, J. R. Villar, E. Corchado, Á. Herrero, B. Baruque, Features and models for human activity recognition, Neurocomputing 167 (2015) 52–60.

[5] J. Redmon, S. Divvala, R. Girshick, 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.

[6] R. Girshick, J. Donahue, T. Darrell, 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.

[7] R. Girshick, Fast r-cnn, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

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

[9] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Communications of the ACM 60 (6) (2017) 84–90.

[10] Z.-Q. Zhao, P. Zheng, S.-T. Xu, X. Wu, Object detection with deep learning: A review, IEEE Transactions on Neural Networks and Learning Systems 30 (2019) 3212–3232. doi:10.1109/TNNLS.2018.2876865.

[11] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, R. Qu, A survey of deep learning-based object detection, IEEE access 7 (2019) 128837–128868.

[12] J. Dai, Y. Li, K. He, J. Sun, R-fcn: Object detection via region-based fully convolutional networks, Advances in neural information processing systems 29 (2016).

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

[14] M. Tan, R. Pang, Q. V. Le, Efficientdet: Scalable and efficient object detection, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 10781–10790.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár, Focal loss for dense object detection, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 2980–2988.

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

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

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

[19] G. Jocher, A. Stoken, A. Chaurasia, J. Borovec, NanoCode012, TaoXie, Y. Kwon, K. Michael, L. Changyu, J. Fang, A. V, Laughing, tkianai, yxNONG, P. Skalski, A. Hogan, J. Nadar, imyhxy, L. Mammana, AlexWang1900, C. Fati, D. Montes, J. Hajek, L. Diaconu, M. T. Minh, Marc, albinxavi, fatih, oleg, wanghaoyang0106, ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support (Oct. 2021). doi:10.5281/zenodo.5563715.

[20] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, I.-H. Yeh, Cspnet: A new backbone that can enhance learning capability of cnn, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, 2020, pp. 390–391.

[21] S. Liu, L. Qi, H. Qin, J. Shi, J. Jia, Path aggregation network for instance segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8759–8768.

[22] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, et al., Yolov6: A single-stage object detection framework for industrial applications, arXiv preprint arXiv:2209.02976 (2022).

[23] C.-Y. Wang, A. Bochkovskiy, H.-Y. M. Liao, YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, arXiv preprint arXiv:2207.02696 (2022).

[24] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, J. Sun, Repvgg: Making vgg-style convnets great again, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021, pp. 13733–13742.

[25] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, Z. Tu, Deeply-supervised nets, in: Artificial intelligence and statistics, PMLR, 2015, pp. 562–570.

[26] G. Jocher, A. Chaurasia, J. Qiu, YOLO by Ultralytics (Jan. 2023).
URL https://github.com/ultralytics/ultralytics

[27] J.-X. Mi, J. Feng, K.-Y. Huang, Designing efficient convolutional neural network structure: A survey, Neurocomputing 489 (2022) 139–156.

[28] J. Chen, X. Ran, Deep learning with edge computing: A review, Proceedings of the IEEE 107 (8) (2019) 1655–1674.

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

[30] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, K. Keutzer, A survey of quantization methods for efficient neural network inference, arXiv preprint arXiv:2103.13630 (2021).

[31] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, Advances in neural information processing systems 2 (1989).

[32] B. Hassibi, D. Stork, G. Wolff, Optimal brain surgeon: Extensions and performance comparisons, Advances in neural information processing systems 6 (1993).

[33] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings (11 2016). doi:10.48550/arxiv.1611.06440.

[34] Y. Chen, Z. Ma, W. Fang, X. Zheng, Z. Yu, Y. Tian, A unified framework for soft threshold pruning, arXiv preprint arXiv:2302.13019 (2023).

[35] H. Zhou, J. M. Alvarez, F. Porikli, Less is more: Towards compact cnns, in: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14, Springer, 2016, pp. 662–677.

[36] M. D. Collins, P. Kohli, Memory bounded deep convolutional networks, arXiv preprint arXiv:1412.1442 (2014).

[37] S. Xu, A. Huang, L. Chen, B. Zhang, Convolutional neural network pruning: A survey, in: 2020 39th Chinese Control Conference (CCC), IEEE, 2020, pp. 7458–7463.

[38] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings

of the IEEE international conference on computer vision, 2017, pp. 2736–2744.
[39] K. Persand, A. Anderson, D. Gregg, Composition of saliency metrics for pruning with a myopic oracle, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2020, pp. 753–759.
[40] B. Dai, C. Zhu, B. Guo, D. Wipf, Compressing neural networks using the variational information bottleneck, in: International Conference on Machine Learning, PMLR, 2018, pp. 1135–1144.
[41] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, J. Ye, Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates, in: Proceedings of the AAAI Conference on Artificial Intelligence, no. 04, 2020, pp. 4876–4883.
[42] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. H. Tan, Z. Li, D. Fan, X. Qian, et al., Non-structured dnn weight pruning—is it beneficial in any platform?, IEEE transactions on neural networks and learning systems 33 (9) (2021) 4930–4944.
[43] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, A. Peste, Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, The Journal of Machine Learning Research 22 (1) (2021) 10882–11005.
[44] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 1389–1397.
[45] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, Y. Tian, Channel pruning via automatic structure search, arXiv preprint arXiv:2001.08565 (2020).
[46] S. Anwar, K. Hwang, W. Sung, Structured pruning of deep convolutional neural networks, J. Emerg. Technol. Comput. Syst. 13 (3) (feb 2017). `doi:10.1145/3005348`.
[47] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, X. Tu, Only train once: A one-shot neural network training and pruning framework, Advances in Neural Information Processing Systems 34 (2021) 19637–19651.
[48] B. Liberatori, C. A. Mami, G. Santacatterina, M. Zullich, F. A. Pellegrino, Yolo-based face mask detection on low-end devices using pruning and quantization, in: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), 2022, pp. 900–905. `doi:10.23919/MIPRO55190.2022.9803406`.
[49] C. Li, Z. Wang, X. Wang, H. Qi, Single-shot channel pruning based on alternating direction method of multipliers, arXiv preprint arXiv:1902.06382 (2019).
[50] T. Shao, D. Shin, Structured pruning for deep convolutional neural networks via adaptive sparsity regularization, in: 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, 2022, pp. 982–987.
[51] E. Frantar, D. Alistarh, Spdy: Accurate pruning with speedup guarantees, in: International Conference on Machine Learning, PMLR, 2022, pp. 6726–6743.
[52] L. Pan, Y. Duan, Y. Zhang, B. Xie, R. Zhang, A lightweight algorithm based on yolov5 for relative position detection of hydraulic support at coal mining faces, Journal of Real-Time Image Processing 20 (2) (2023) 1–12.
[53] Q. Yang, F. Li, H. Tian, H. Li, S. Xu, J. Fei, Z. Wu, Q. Feng, C. Lu, A new knowledge-distillation-based method for detecting conveyor belt defects, Applied Sciences 12 (19) (2022) 10051.
[54] Y. Chen, J. Yang, J. Wang, X. Zhou, J. Zou, Y. Li, An improved yolov5 real-time detection method for aircraft target detection, in: 2022 27th International Conference on Automation and Computing (ICAC), IEEE, 2022, pp. 1–6.
[55] D. Wang, D. He, Channel pruned yolo v5s-based deep learning approach for rapid and accurate apple fruitlet detection before fruit thinning, Biosystems Engineering 210 (2021) 271–281.
[56] Z. Li, K. Qiu, Z. Yu, Channel pruned yolov5-based deep learning approach for rapid and accurate outdoor obstacles detection, arXiv preprint arXiv:2204.13699 (2022).
[57] A. Ding, B. Peng, K. Yang, Y. Zhang, X. Yang, X. Zou, Z. Zhu, Design of a machine vision-based automatic digging depth control system for garlic combine harvester, Agriculture 12 (12) (2022) 2119.
[58] W. Zhao, S. Liu, X. Li, X. Han, H. Yang, Fast and accurate wheat grain quality detection based on improved yolov5, Computers and Electronics in Agriculture 202 (2022) 107426.
[59] J.-C. Zheng, S.-D. Sun, S.-J. Zhao, Fast ship detection based on lightweight yolov5 network, IET Image Processing 16 (6) (2022) 1585–1593.
[60] Y. Zhou, M. Wu, Y. Bai, C. Guo, Flame detection with pruned and knowledge distilled yolov5, in: 2021 5th Asian Conference on Artificial Intelligence Technology (ACAIT), IEEE, 2021, pp. 780–785.
[61] C. Li, G. Zhao, D. Gu, Z. Wang, Improved lightweight yolov5 using attention mechanism for satellite components recognition, IEEE Sensors Journal 23 (1) (2022) 514–526.
[62] C. Sun, S. Zhang, P. Qu, X. Wu, P. Feng, Z. Tao, J. Zhang, Y. Wang, Mca-yolov5-light: A faster, stronger and lighter algorithm for helmet-wearing detection, Applied Sciences 12 (19) (2022) 9697.
[63] J. Zhang, P. Wang, Z. Zhao, F. Su, Pruned-yolo: Learning efficient object detector using model pruning, in: Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part IV, Springer, 2021, pp. 34–45.
[64] Z. Situ, S. Teng, X. Liao, G. Chen, Q. Zhou, Real-time sewer defect detection based on yolo network, transfer learning, and channel pruning algorithm, Journal of Civil Structural Health Monitoring (2023) 1–17.
[65] L. Shen, J. Su, R. He, L. Song, R. Huang, Y. Fang, Y. Song, B. Su, Real-time tracking and counting of grape clusters in the field based on channel pruning with yolov5s, Computers and Electronics in Agriculture 206 (2023) 107662.
[66] Y. Zhang, W. Wang, Q. Liu, Z. Guo, Y. Ji, Research on defect detection in automated fiber placement processes based on a multi-scale detector, Electronics 11 (22) (2022) 3757.
[67] X. Liu, C. Wang, L. Liu, Research on pedestrian detection model and compression technology for uav images, Sensors 22 (23) (2022) 9171.
[68] R. Zhang, C. Wen, Sod-yolo: A small target defect detection algorithm for wind turbine blades based on improved yolov5, Advanced Theory and Simulations 5 (7) (2022) 2100631.
[69] J. Jeon, J. Kim, J.-K. Kang, S. Moon, Y. Kim, Target capacity filter pruning method for optimized inference time based on yolov5 in embedded systems, IEEE Access 10 (2022) 70840–70849.
[70] H. Lv, H. Yan, K. Liu, Z. Zhou, J. Jing, Yolov5-ac: Attention mechanism-based lightweight yolov5 for track pedestrian detection, Sensors 22 (15) (2022) 5903.
[71] Y. Xu, Y. Bai, Compressed yolov5 for oriented object detection with integrated network slimming and knowledge distillation, in: 2022 3rd

International Conference on Information Science, Parallel and Distributed Systems (ISPDS), IEEE, 2022, pp. 394–403.

[72] Y. Nan, H. Zhang, Y. Zeng, J. Zheng, Y. Ge, Faster and accurate green pepper detection using nsga-ii-based pruned yolov5l in the field environment, Computers and Electronics in Agriculture 205 (2023) 107563.

[73] T. Zeng, S. Li, Q. Song, F. Zhong, X. Wei, Lightweight tomato real-time detection method based on improved yolo and mobile deployment, Computers and Electronics in Agriculture 205 (2023) 107625.

[74] X. Wang, J. Zhang, Y. Wang, M. Li, D. Liu, Defect detection of track fasteners based on pruned yolo v5 model, in: 2022 IEEE 11th Data Driven Control and Learning Systems Conference (DDCLS), IEEE, 2022, pp. 391–395.

[75] Q. Song, S. Li, Q. Bai, J. Yang, X. Zhang, Z. Li, Z. Duan, Object detection method for grasping robot based on improved yolov5, Micromachines 12 (11) (2021) 1273.

[76] Z. Wang, L. Jin, S. Wang, H. Xu, Apple stem/calyx real-time recognition using yolo-v5 algorithm for fruit automatic loading system, Postharvest Biology and Technology 185 (2022) 111808.

[77] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al., Searching for mobilenetv3, in: Proceedings of the IEEE/CVF international conference on computer vision, 2019, pp. 1314–1324.

[78] H. Wu, P. Judd, X. Zhang, M. Isaev, P. Micikevicius, Integer quantization for deep learning inference: Principles and empirical evaluation, arXiv preprint arXiv:2004.09602 (2020).

[79] F. Fu, Y. Hu, Y. He, J. Jiang, Y. Shao, C. Zhang, B. Cui, Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript, in: International Conference on Machine Learning, 2020, pp. 3304–3314.

[80] Y. Li, X. Dong, W. Wang, Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks, arXiv preprint arXiv:1909.13144 (2019).

[81] C. Baskin, N. Liss, E. Schwartz, E. Zheltonozhskii, R. Giryes, A. M. Bronstein, A. Mendelson, Uniq: Uniform noise injection for non-uniform qantization of neural networks, ACM Transactions on Computer Systems 37 (1-4) (2021). doi:10.1145/3444943.

[82] K. Yamamoto, Learnable companding quantization for accurate low-bit neural networks, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021, pp. 5029–5038.

[83] O. Weng, Neural network quantization for efficient inference: A survey, arXiv preprint arXiv:2112.06126 (2021).

[84] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, A. Mendelson, Loss aware post-training quantization, Machine Learning 110 (11-12) (2021) 3245–3262.

[85] Q. Wu, Y. Li, S. Chen, Y. Kang, Drgs: Low-precision full quantization of deep neural network with dynamic rounding and gradient scaling for object detection, in: Data Mining and Big Data: 7th International Conference, DMBD 2022, Beijing, China, November 21–24, 2022, Proceedings, Part I, Springer, 2023, pp. 137–151.

[86] S. Park, J. So, J. Shin, E. Park, Nipq: Noise injection pseudo quantization for automated dnn optimization, arXiv preprint arXiv:2206.00820 (2022).

[87] A. Papaioannou, C. S. Kouzinopoulos, D. Ioannidis, D. Tzovaras, An ultra-low-power embedded ai fire detection and crowd counting system for indoor areas, ACM Transactions on Embedded Computing Systems (2023).

[88] N. Ma, X. Zhang, H.-T. Zheng, J. Sun, Shufflenet v2: Practical guidelines for efficient cnn architecture design, in: Proceedings of the European conference on computer vision (ECCV), 2018, pp. 116–131.

[89] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, et al., Tensorflow lite micro: Embedded machine learning for tinyml systems, Proceedings of Machine Learning and Systems 3 (2021) 800–811.

[90] S. Ashfaq, M. AskariHemmat, S. Sah, E. Saboori, O. Mastropietro, A. Hoffman, Accelerating deep learning model inference on arm cpus with ultra-low bit quantization and runtime, arXiv preprint arXiv:2207.08820 (2022).

[91] J. Wang, Q. Song, M. Hou, G. Jin, Infrared image object detection of vehicle and person based on improved yolov5, in: Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data, Springer, 2023, pp. 175–187.

[92] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.

[93] P. Cui, J. Zhang, B. Han, Y. Wu, Performance evaluation and model quantization of object detection algorithm for infrared image, in: Seventh Asia Pacific Conference on Optics Manufacture and 2021 International Forum of Young Scientists on Advanced Optical Manufacturing (APCOM and YSAOM 2021), Vol. 12166, SPIE, 2022, pp. 554–559.

[94] D. Choi, H. Kim, Hardware-friendly log-scale quantization for cnns with activation functions containing negative values, in: 2021 18th International SoC Design Conference (ISOCC), IEEE, 2021, pp. 415–416.

[95] R. Rao, K. Chen, et al., Nvidia tensorrt, https://developer.nvidia.com/tensorrt (2016).

[96] Pytorch quantization, https://pytorch.org/docs/stable/quantization.html.

[97] J. Bai, F. Lu, K. Zhang, et al., Onnx: Open neural network exchange, https://github.com/onnx/onnx (2019).

[98] D. Maji, S. Nagori, M. Mathew, D. Poddar, Yolo-pose: Enhancing yolo for multi person pose estimation using object keypoint similarity loss, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 2637–2646.

[99] J. Fang, Q. Liu, J. Li, A deployment scheme of yolov5 with inference optimizations based on the triton inference server, in: 2021 IEEE 6th International Conference on cloud computing and big data analytics (ICCCBDA), IEEE, 2021, pp. 441–445.

[100] B. Li, B. Wu, J. Su, G. Wang, Eagleeye: Fast sub-net evaluation for efficient neural network pruning, in: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, Springer, 2020, pp. 639–654.

[101] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, L. Shao, Hrank: Filter pruning using high-rank feature map, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 1529–1538.

[102] I. Hubara, B. Chmiel, M. Island, R. Banner, J. Naor, D. Soudry, Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks, Advances in Neural Information Processing Systems 34 (2021) 21099–21111.

[103] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, Q. Tian, Variational convolutional neural network pruning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2780–2789.

[104] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, T. Blankevoort, Up or down? adaptive rounding for post-training quantization, in:

International Conference on Machine Learning, PMLR, 2020, pp. 7197–7206.

[105] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. Mahoney, et al., Hawq-v3: Dyadic neural network quantization, in: International Conference on Machine Learning, PMLR, 2021, pp. 11875–11886.

[106] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, D. Soudry, Accurate post training quantization with small calibration sets, in: International Conference on Machine Learning, PMLR, 2021, pp. 4466–4475.

[107] P. Peng, M. You, W. Xu, J. Li, Fully integer-based quantization for mobile convolutional neural network inference, Neurocomputing 432 (2021) 194–205.

[108] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, A. Mendelson, Loss aware post-training quantization, Machine Learning 110 (11-12) (2021) 3245–3262.