Ingénierie des Systèmes d'Information
Vol. 31, No. 5, May, 2026, pp. 1403-1417
Journal homepage: http://iieta.org/journals/isi

Resource-Efficient Helmet Detection Using Structured L1 Channel Pruning of YOLOv11n

Ihda Innar Ridho[1,2*], Pulung Nurtantio Andono[1], Purwanto[1], Moch. Arief Soeleman[1]

[1] Faculty of Computer Science, Universitas Dian Nuswantoro, Semarang 50131, Indonesia
[2] Faculty of Information Technology, Islamic University of Kalimantan Muhammad Arsyad Al Banjari, Banjarmasin 70123, Indonesia

Corresponding Author Email: p41202300074@mhs.dinus.ac.id

https://doi.org/10.18280/isi.310503

Received: 27 January 2026
Revised: 25 March 2026
Accepted: 20 April 2026
Available online: 31 May 2026

Keywords:
helmet detection, L1 channel pruning, YOLOv11n, model compression, cross-validation, structured pruning, edge deployment, object detection

ABSTRACT

Helmet detection for occupational and traffic safety requires lightweight models that operate continuously on resource-limited infrastructure. This study applied structured L1 channel pruning to YOLOv11n, targeting the inner bottleneck convolution pairs (cv1 and cv2) within C3K2 and C2f blocks while preserving outer projection dimensions. Two pruning ratios, 30% and 50%, were evaluated, followed by 20-epoch fine-tuning. Validation followed a 5-fold stratified cross-validation protocol on the Kaggle Helmet Detection dataset (764 images, 2 classes), with results reported as mean and standard deviation across folds. The baseline YOLOv11n achieved an mAP50 of 0.7889 ± 0.0331 at 203.49 ± 40.23 frames per second (FPS). The 30% pruned model reached an mAP50 of 0.7962 ± 0.0281, and the 50% pruned model reached 0.7882 ± 0.0366; neither change was statistically significant relative to the baseline (paired t-test, $p = 0.42$ and $p = 0.95$, respectively). At the 50% ratio, parameter count decreased by 8.20%, Open Neural Network Exchange (ONNX) model size by 8.00%, and Floating Point Operations (FLOPs) by 6.49%, while inference speed increased by 17.2% to 238.42 ± 11.09 FPS. Layer sensitivity analysis identified the early backbone convolution layers as most susceptible to channel removal, informing safer pruning targets. A pruning criteria comparison among L1, L2, and random selection showed comparable post-fine-tuning accuracy across criteria, with L1 producing the largest pre-fine-tuning drop (0.1115) and the fastest proxy inference speed. The pruned YOLOv11n outperformed YOLOv8n on identical hardware, achieving higher mAP50 (0.7882 versus 0.7733) with fewer parameters (2.378M versus 3.011M). This study presents the first application of structured L1 channel pruning to YOLOv11n for helmet detection and provides a reproducible compression baseline validated through statistical testing.

1. INTRODUCTION

Motorcycle accidents contribute to a significant proportion of global road deaths. The World Health Organization (WHO) reports that motorcycle riders and passengers face a disproportionately high risk of fatal injury [1]. Standard helmets reduce the risk of head injury by up to 70% [2]. Enforcing helmet regulations therefore remains a priority for the Indonesian government and for other countries facing similar road safety challenges.

Traditional methods relies on field officers, whose limited number is disproportionate to the volume of vehicles on the road. Manual supervision is error-prone, inconsistent, and cannot operate continuously [3]. This enforcement gap reduces the practical likelihood of detection, which in turn weakens compliance incentives for riders.

Computer vision offers an automated alternative to manual enforcement. Surveillance systems based on closed-circuit television (CCTV) cameras can identify helmetless riders in real time. Object detection models such as You Only Look Once (YOLO) have proven effective for this task [4, 5]. However, deployment on resource-constrained infrastructure remains challenging, since standard YOLO variants carry substantial computational overhead that limits scalability on servers processing multiple concurrent video streams.

YOLOv11n, the smallest variant of the YOLOv11 family, is designed for efficient deployment. Its multi-path efficient block architecture and loss-aware head design improve the accuracy-speed trade-off relative to earlier YOLO generations [6]. This model is well suited to continuous monitoring scenarios, although further optimization is required to maximize throughput under constrained compute budgets. Neural network pruning effectively reduces model complexity without substantial accuracy loss [7, 8], making it a natural candidate for this optimization.

A manual screening of 163 papers published between 2022 and 2024 indicates that YOLOv5 and YOLOv8 dominate helmet detection research, with various compression techniques applied to these architectures [9-11]. Existing pruning methods applied to these older YOLO variants report accuracy drops of 0.6-7% at compression ratios of 30-50% [12]. YOLOv11n, released in late 2024, has not yet been explored for the helmet detection task. Furthermore, automated structural pruning tools such as torch_pruning

encounter compatibility issues with the C3K2 and C2PSA modules in YOLOv11n, particularly at the Concat operations within the Feature Pyramid Network and Path Aggregation Network. A focused inner bottleneck pruning strategy, which modifies only the cv1 and cv2 channel pairs inside each Bottleneck block while preserving the outer projection dimensions, avoids these dimensional conflicts and enables stable structured compression of this architecture.

This research makes four contributions. First, it presents the first implementation of structured L1 channel pruning on YOLOv11n for helmet detection, establishing a reproducible compression baseline for this architecture. Second, it validates the results through 5-fold stratified cross-validation with paired t-test and Wilcoxon signed-rank testing, providing statistical confirmation that pruning does not significantly degrade mAP50 at either ratio. Third, it provides a layer sensitivity analysis that identifies which backbone layers are most susceptible to channel removal, together with a pruning criteria comparison across L1, L2, and random selection. Fourth, it includes a direct comparison with YOLOv8n trained on identical hardware and dataset, demonstrating that the pruned YOLOv11n achieves higher accuracy with fewer parameters.

The remainder of this paper is organized as follows. Section 2 reviews related work on YOLO variants, helmet detection systems, and neural network pruning techniques. Section 3 describes the methodology, including the dataset, model architecture, pruning algorithm, cross-validation protocol, and experimental setup. Section 4 presents the results with complete metrics. Section 5 discusses the findings and compares them with the state of the art. Section 6 concludes the paper and outlines directions for future work.

2. RELATED WORK

2.1 You Only Look Once variants for object detection

The YOLO family has undergone significant evolution since its introduction. YOLOv1 started the one-stage detection paradigm with direct prediction of bounding boxes and class probabilities [13]. Anchor-based mechanisms in YOLOv3 improved the accuracy of small object detection [14]. YOLOv4 added the CSPDarknet53 backbone together with various augmentation techniques [15]. The next generation of YOLO models focused on efficiency. YOLOv5 introduced nano to extra-large variants for different computational budgets [16]. YOLOv7 used efficient layer aggregation networks [17]. YOLOv8 introduced anchor-free detection with a decoupled head design [6].

YOLOv11 includes C2PSA modules and C3K2 blocks for more efficient multi-scale feature capture [6]. Lightweight variants have emerged for edge deployment. YOLOv5-nano achieves a 1.9 MB model size with 45% mAP on COCO [18]. YOLOv8-nano improves performance to 37.3% mAP with a 3.2 MB size [19]. YOLOv11n continues this trend, targeting devices with limited RAM and processing power.

2.2 Helmet detection systems

Helmet detection has been explored using various YOLO versions. YOLOv5-SN achieves 31 frames per second (FPS) on the Jetson Nano using a ShuffleNetV2 backbone with 90% parameter reduction [20]. SG-YOLOv5 maintains baseline mAP while achieving a 90.8% parameter reduction and a 2.7x FPS improvement [21]. YOLOv8 variants have also been applied to this task. The AI City Challenge Track 5 uses YOLOv8 for multi-class helmet violation detection [22], and YOLOv8-ADown-LSCD achieves 0.85 mAP@0.5 on a custom dataset [23]. The Kaggle Helmet Detection dataset, containing 764 images, is frequently used as a research benchmark [24]. Dataset differences limit direct comparison across these studies, and generalization across varying camera conditions remains an open challenge.

2.3 Neural network pruning techniques

Structured pruning removes entire channels or filters, producing hardware-friendly sparse models without requiring specialized sparse computation libraries. Filter pruning with an L1-norm criterion has proven effective for practical deployment [25]. Li et al. [7] introduced a magnitude-based approach in which filters with small L1-norm weights are removed.

Batch normalization scaling factors serve as channel importance indicators. Network Slimming adds an L1 penalty to batch normalization (BN) gamma parameters during training [26]. NS-YOLO uses BN-based pruning in YOLOv5, achieving an 88.19% parameter reduction with only a 1.6% mAP drop [12]. A-pruning uses adaptive BN evaluation to select optimal subnetworks, achieving a 75.8% parameter reduction with a 0.9% accuracy drop on YOLOv5 [27]. Target Capacity Pruning designs compression to meet specific inference time constraints; on YOLOv5, a 30% pruning ratio loses 0.6% mAP, and a 50% ratio loses 2.9% [28]. SAFP-YOLO uses spatial attention-based filter pruning on YOLOv7, achieving substantial inference speed gains on embedded hardware [29].

CM-YOLOv8 applies L1-norm channel pruning for coal mine detection, achieving a 40% model volume reduction with less than 1% accuracy drop [30]. YOLOv8-ALWP uses layer-adaptive magnitude-based pruning, achieving a 64.67% parameter reduction together with a 1.7% accuracy improvement [31]. Compression techniques beyond pruning include knowledge distillation [32], quantization [33], and hybrid approaches that combine multiple compression strategies for deployment on platforms such as the Jetson Xavier NX [34].

2.4 Research gap and positioning

Existing work concentrates on YOLOv5, YOLOv7, and YOLOv8 variants. YOLOv11n, despite its architectural advantages, has not been studied for helmet detection, and no prior work applies structured pruning to this architecture. The C3K2 and C2PSA modules introduced in YOLOv11 create dimensional constraints at Concat junctions that prevent the direct application of existing global pruning pipelines. This gap motivates a targeted inner bottleneck pruning strategy that operates within each Bottleneck block, preserving outer channel dimensions while reducing internal redundancy. The present work addresses this gap by providing the first structured L1 channel pruning study on YOLOv11n, validated through cross-validation and statistical testing, and benchmarked against YOLOv8n under identical experimental conditions.

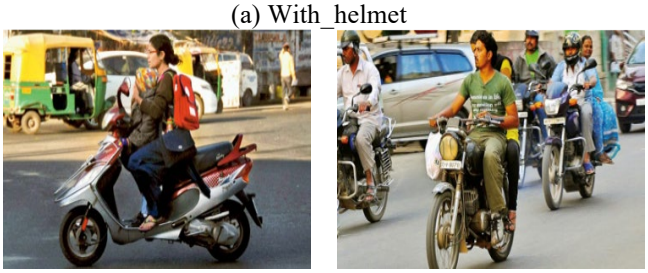
3. METHODOLOGY

3.1 Dataset

The Helmet Detection dataset from Kaggle was used [24]. The collection contains 764 images with PASCAL VOC format annotations. Two classes are defined: with_helmet and without_helmet, with bounding box coordinates provided for each object. Figure 1 shows sample images from both classes. The dataset was split according to a standard ratio, with 70% (535 images) allocated to training, 15% (115 images) to validation, and 15% (114 images) to testing. A random shuffle was performed before splitting to ensure an even distribution across subsets.

This dataset presents several detection challenges. Helmets are small, typically occupying 5-10% of the image area. Rider pose variation is high, and occlusion from hands, visors, or passengers is common. Lighting conditions vary from bright daylight to backlit scenes, and a slight class imbalance exists, with marginally more with_helmet samples than without_helmet samples.

(a) With_helmet



(b) Without_helmet

Figure 1. Sample images from the Helmet Detection dataset: (a) with_helmet, (b) without_helmet

3.2 YOLOv11n architecture

YOLOv11n consists of three main components: a backbone that extracts features through convolutional layers, a neck that aggregates multi-scale information, and a head that produces the final predictions. Figure 2 shows the complete architecture.

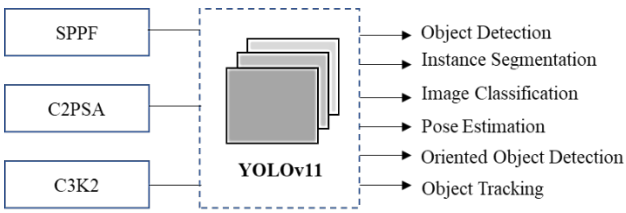


Figure 2. YOLOv11n architecture overview

The backbone employs C3K2 modules that capture local and semi-global patterns using a two-kernel Bottleneck design [6]. The neck combines a Spatial Pyramid Pooling Fast (SPPF) [35] module at high-level layers with C2PSA modules that incorporate cross-stage partial connections and partial self-attention [6]. The detection head uses a decoupled, anchor-free design that separates classification and localization tasks. YOLOv11n contains 2,590,230 parameters in total, with Floating Point Operations (FLOPs) of approximately 6.38 G for a single 640 × 640 image.

3.3 Structured L1 channel pruning

Pruning targets the inner bottleneck convolution pairs, cv1 and cv2, within each Bottleneck block in the C3K2 and C2f modules of the backbone and neck. This layer scope was chosen to preserve the outer projection dimensions at block boundaries, avoiding dimensional conflicts at Concat operations in the Feature Pyramid Network and Path Aggregation Network. The detection head layers are excluded to maintain output structure.

Channel importance is determined by the L1 norm of the corresponding convolutional weights. For channel k in a cv1 layer with weight tensor W, the importance score is given by:

$$I_k = \sum_{i=1}^{N} |W(k,i)| \tag{1}$$

where I_k is the importance score for channel k, W(k,i) is the filter weight at output channel k and weight index i, and N is the total number of weights per channel. Channels are sorted in ascending order by importance score, and the bottom r fraction is selected for removal:

$$r = \frac{K_{pruned}}{K_{total}} \tag{2}$$

Two ratios were tested: r = 0.30 and r = 0.50, with the pruning set defined as the channels with the lowest importance scores.

Algorithm 1 summarizes the inner bottleneck pruning procedure, including the cv1-cv2 consistency requirement that maintains valid tensor dimensions after channel removal.

Algorithm 1. Structured L1 Channel Pruning Algorithm

Input: Pre-trained model M, pruning ratio r ∈ {0.30, 0.50}

Output: Pruned and fine-tuned model M'

1: Load M from unfused checkpoint (bypass auto-fuse to expose Bottleneck weights)
2: Identify all Bottleneck blocks B = {b1, b2, ..., bn} in backbone and neck of M
3: For each block bi ∈ B:
4: For each channel k in bi.cv1.out_channels:
5: Compute Ik = Σ |W_cv1(k,:)|
▷ L1-norm over all weights of channel k
6: Sort channels ascending by I_k
7: Select $\underset{k}{\text{argmin}}\, I_k$ such that $|P| = \lfloor r \times K_{total} \rfloor$
8: Remove channels P from bi.cv1.out_channels
9: Remove corresponding input channels from bi.cv2.in_channels
▷ cv1-cv2 consistency
10: Fine-tune M' for 20 epochs (lr = 0.001, SGD, batch = 16, imgsz = 640)
11: Return M'

3.4 Fine-tuning strategy

After pruning, the model is fine-tuned for 20 epochs to recover any accuracy lost from channel removal. The 20-epoch budget was determined by monitoring validation mAP curves during preliminary runs, which showed convergence consistently occurring within 15 epochs with no further improvement beyond that point. The fine-tuning learning rate is set to 0.001, one order of magnitude below the baseline initial rate of 0.01. This reduced rate prevents large gradient updates from disrupting the weight distributions of the remaining channels, which are already near their optimal configuration. The same SGD optimizer, momentum (0.937), weight decay (0.0005), batch size (16), and image resolution (640 × 640) used during baseline training are retained during fine-tuning to maintain consistency and avoid confounding factors. The best checkpoint based on validation mAP is saved at each epoch, and the final evaluation uses this checkpoint rather than the checkpoint from the last epoch.

3.5 5-fold cross-validation protocol

To ensure statistical robustness and reduce dependence on a single train-test split, all experiments adopt a 5-fold stratified cross-validation protocol. The dataset of 764 images is partitioned into five folds, maintaining the class distribution within each fold. For each fold, the remaining four folds form the training and validation pool, split 70%/15%, while the held-out fold serves as the test set. This procedure is repeated five times, with each fold serving as the test set exactly once. All three model configurations, baseline, pruned_30, and pruned_50, are trained and evaluated independently within each fold, ensuring that no test data influences training or pruning decisions.

Performance metrics are reported as the mean plus or minus one standard deviation across the five folds. Statistical comparison between the baseline and pruned models uses a paired t-test (n = 5) that treats each fold as a matched pair, supplemented by the Wilcoxon signed-rank test as a non-parametric confirmation. A significance threshold of alpha = 0.05 is applied. This dual-test approach was adopted because the small sample size (n = 5) limits the power of the t-test, and the Wilcoxon test provides additional assurance under non-normality.

3.6 Evaluation metrics

Mean Average Precision (mAP) measures detection accuracy. The metric averages precision across recall levels for each class, then averages the result across all classes:

$$mAP = \frac{1}{C}\sum_{i=1}^{C} AP_i \tag{3}$$

where, C is the number of classes (2 for this dataset) and AP_i is Average Precision for class i. Precision and recall are calculated from the confusion matrix:

$$P = \frac{TP}{TP + FP} \tag{4}$$

$$R = \frac{TP}{TP + FN} \tag{5}$$

This study reports mAP@0.5 (IoU threshold of 0.5) and mAP@0.5:0.95 (thresholds from 0.5 to 0.95 in steps of 0.05). The intersection over union (IoU) between the predicted bounding box (B_p) and the ground truth bounding box (B_{gt}) is calculated as:

$$IoU = \frac{\left|B_p \cap B_{gt}\right|}{\left|B_p \cup B_{gt}\right|} \tag{6}$$

Detections with IoU ≥ 0.5 are considered correct for mAP@0.5, while mAP@0.5:0.95 applies the stricter range of thresholds described above.

FPS measures inference throughput, with higher values indicating faster processing.

$$FPS = \frac{1000}{t_{inference}} \tag{7}$$

where, $t_{inference}$ is the inference time in milliseconds per image. Preprocessing and postprocessing time are excluded to isolate the computational cost of the model itself. Additional metrics include parameter count, Open Neural Network Exchange (ONNX) model file size (MB), and FLOPs.

ONNX format is used for model size measurement rather than the native PyTorch .pt checkpoint. The .pt format stores optimizer state, learning rate schedules, and training metadata in addition to the model weights, resulting in a file size that does not reflect the deployable model size. ONNX export strips all training artifacts, retaining only the inference graph and weights, and therefore provides a more accurate measure of deployment footprint.

3.7 Experimental setup

Experiments were conducted on two NVIDIA T4 GPUs in a Kaggle environment, using Python 3.11.13, PyTorch 2.6.0+cu124, and ultralytics 8.4.48. Baseline training ran for 100 epochs with an initial learning rate of 0.01 under a cosine annealing schedule. The SGD optimizer used a momentum of 0.937 and a weight decay of 0.0005. Image augmentations included random horizontal flip (50% probability), mosaic (probability 1.0), and mixup (probability 0.1). Input resolution was fixed at 640 × 640 pixels with a batch size of 16.

Automated structural pruning libraries such as torch_pruning, which builds dependency graphs via DepGraph [36], were evaluated but proved incompatible with the C3K2 and C2PSA modules of YOLOv11n: the Concat operations within the Feature Pyramid Network and Path Aggregation Network caused ignored-layer status to propagate across the dependency graph, preventing any channel removal. Pruning was therefore implemented through the custom procedure described in Algorithm 1, which directly modifies the cv1 and cv2 weight tensors of each Bottleneck block without relying on automated dependency resolution. Channel removal modifies the network architecture in place. Fine-tuning follows the configuration described in Section 3.4. Evaluation uses single-scale inference with a confidence threshold of 0.25 and an IoU threshold of 0.45 for non-maximum suppression. Figure 3 illustrates the complete workflow.

Table 1 summarizes hyperparameters that used in this research.

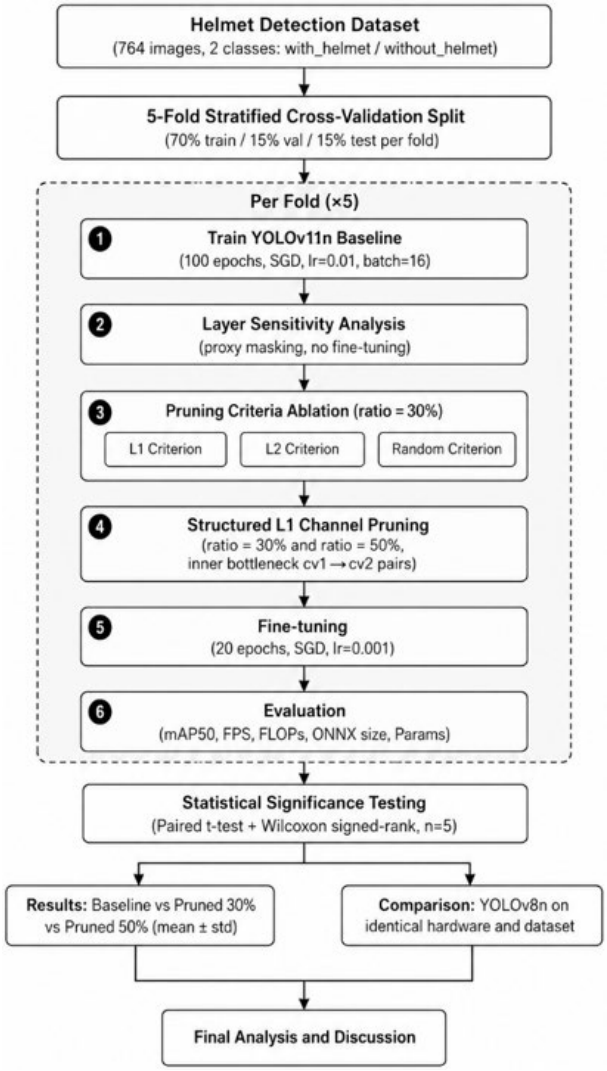


Figure 3. Complete experimental pipeline including 5-fold cross-validation, layer sensitivity analysis, pruning criteria ablation, structured L1 pruning, and statistical significance testing

Table 1. Training configuration

Parameter	Baseline Training	Fine-Tuning
Base Model	YOLOv11n	YOLOv11n (pruned)
Epochs	100	20
Batch Size	16	16
Image Size	640 × 640	640 × 640
Initial LR	0.01	0.001
Optimizer	SGD (momentum 0.937)	SGD (momentum 0.937)
Weight Decay	0.0005	0.0005
GPU	NVIDIA T4 × 2 (Kaggle)	NVIDIA T4 × 2 (Kaggle)
Framework	ultralytics 8.4.48	ultralytics 8.4.48

4. RESULTS

4.1 Overall performance

Table 2 reports the 5-fold cross-validation results for all three model configurations, together with YOLOv8n evaluated on identical hardware and dataset as a same-hardware baseline. All metrics are reported as the mean plus or minus one standard deviation across five folds, except for YOLOv8n, which was evaluated on the original single split for direct comparison.

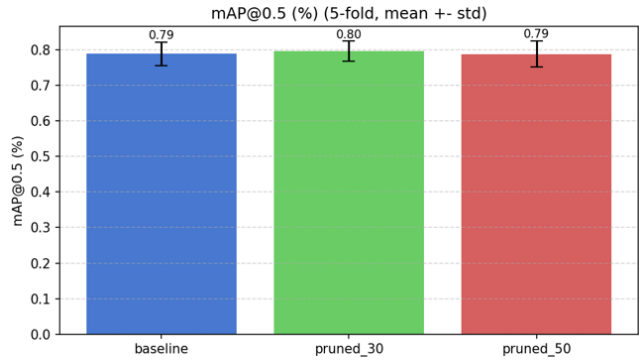


Figure 4. Mean mAP50 with standard deviation across five folds for baseline, pruned_30, and pruned_50

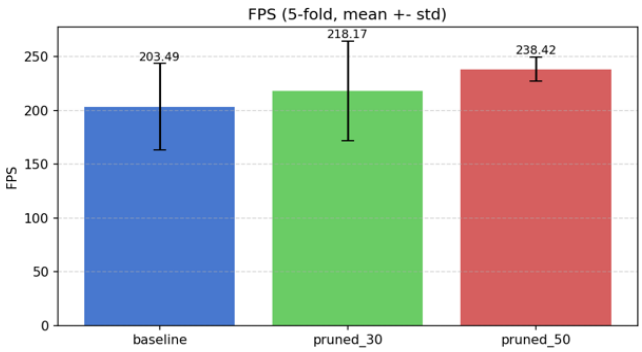


Figure 5. Mean frames per second (FPS) with standard deviation across five folds

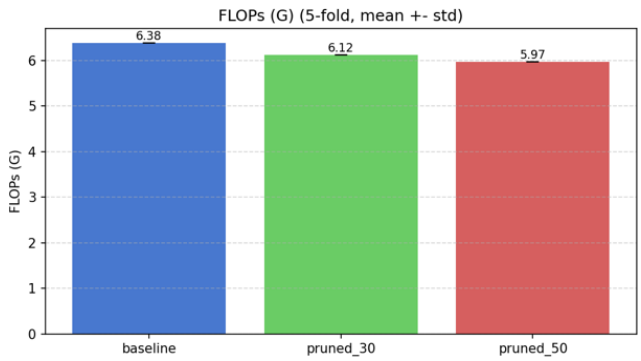


Figure 6. Mean Floating Point Operations (FLOPs) with standard deviation across five folds

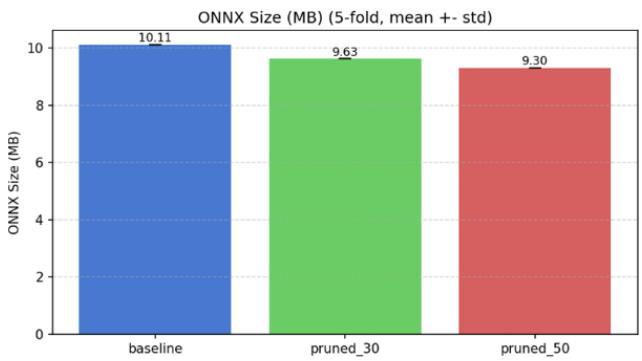


Figure 7. Mean Open Neural Network Exchange (ONNX) model size with standard deviation across five folds

Table 2. Five-fold cross-validation results for baseline and pruned YOLOv11n configurations, with same-hardware YOLOv8n comparison

Model	mAP50	mAP50-95	Precision	Recall	FPS	Params (M)	ONNX (MB)	FLOPs (G)
Baseline	0.7889 ± 0.0331	0.4709 ± 0.0079	0.7788	0.7267	203.49 ± 40.23	2.590	10.11	6.38
Pruned 30%	0.7962 ± 0.0281	0.4746 ± 0.0123	0.7579	0.7688	218.17 ± 46.27	2.462	9.63	6.12
Pruned 50%	0.7882 ± 0.0366	0.4669 ± 0.0086	0.7559	0.7502	238.42 ± 11.09	2.378	9.30	5.97
YOLOv8n†	0.7733	0.4672	0.7832	0.6791	211.67	3.011	11.70	8.14

Note: † Trained and evaluated on identical hardware (NVIDIA T4) and dataset without cross-validation; not included in the statistical comparison. Bold values, where present, indicate the best result per column.
FLOPs = Floating Point Operations; FPS = rames per second; mAP = Mean Average Precision.

mAP50 remained stable across both pruning ratios. The 30% pruned model produced a slightly higher mAP50 than the baseline (0.7962 versus 0.7889, a delta of +0.0073), which reflects a regularization effect from fine-tuning rather than a genuine accuracy gain. The 50% pruned model was nearly identical to the baseline (0.7882, a delta of -0.0006). Both differences are small in absolute terms and are discussed further in Section 4.6. FPS increased from 203.49 ± 40.23 for the baseline to 218.17 ± 46.27 for pruned_30 (+7.2%) and to 238.42 ± 11.09 for pruned_50 (+17.2%). The narrower FPS standard deviation for pruned_50 (11.09, compared with 40.23 for the baseline) suggests more consistent throughput after aggressive pruning. Figures 4 through 7 show the corresponding per-metric bar charts with error bars across folds.

4.2 Model compression analysis

Table 3 reports compression statistics relative to the baseline, using ONNX export size as the deployment-relevant measure.
The modest compression ratios, 1.052× and 1.089×, reflect the inner bottleneck scope of pruning. Only the cv1-cv2 channel pairs inside each Bottleneck block are modified, while the outer projection channels and all non-Bottleneck layers remain unchanged. This conservative scope is intentional: it avoids dimensional conflicts at Concat operations in the Feature Pyramid Network-Path Aggregation Network (FPN-PAN) structure and maintains architectural stability. As a consequence, the parameter and size reductions are smaller than those reported by methods that prune global channels or entire layers. The FLOPs reduction, 4.00% and 6.49%, follows a similar pattern, since the remaining backbone and neck layers continue to compute at full channel width.

4.3 Layer sensitivity analysis

Layer sensitivity was measured by temporarily masking 30% of channels in each eligible layer independently and recording the resulting proxy mAP50 drop without fine-tuning. Figure 8 shows the top 20 most sensitive layers. Table 4 lists the five layers with the highest sensitivity.

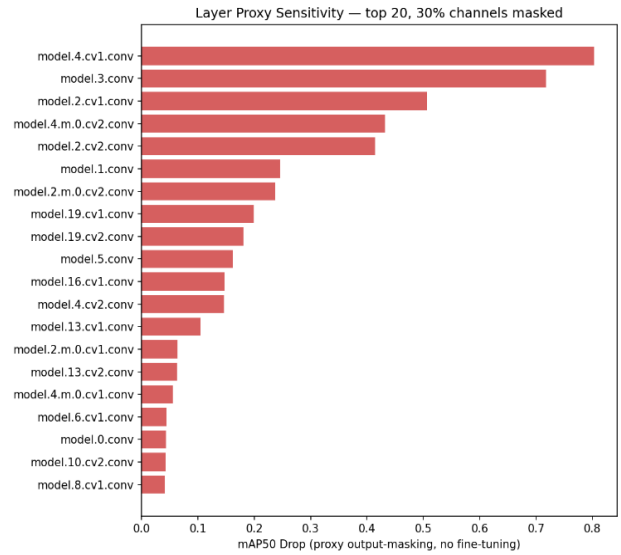


Figure 8. Top 20 layer sensitivity scores, where a larger drop indicates higher sensitivity to 30% channel masking

Table 3. Compression statistics relative to baseline, based on ONNX export size

Model	Params Reduction (%)	FLOPs Reduction (%)	ONNX Reduction (%)	Compression Ratio	Baseline ONNX (MB)	Pruned ONNX (MB)
Pruned 30%	4.95%	4.00%	4.83%	1.052x	10.11	9.63
Pruned 50%	8.20%	6.49%	8.00%	1.089x	10.11	9.30

Note: FLOPs = Floating Point Operations; ONNX = Open Neural Network Exchange.

Table 4. Top-5 most sensitive layers to 30% channel masking (no fine-tuning)

Rank	Layer Name	Layer Index	Out Channels	mAP50 Drop	Proxy mAP50
1	model.4.cv1.conv	7	64	0.8034	0.0176
2	model.3.conv	6	64	0.7180	0.1030
3	model.2.cv1.conv	2	32	0.5059	0.3150
4	model.4.m.0.cv2.conv	10	32	0.4320	0.3890
5	model.2.cv2.conv	3	64	0.4142	0.4068

The five most sensitive layers are concentrated in the early backbone stages, specifically layer indices 2 through 10. Masking 30% of channels in model.4.cv1.conv (layer index 7) reduced proxy mAP50 from 0.8210 to 0.0176, a drop of 0.8034, making it the most critical layer by a substantial margin. model.3.conv (layer 6) and model.2.cv1.conv (layer 2) followed, with drops of 0.7180 and 0.5059, respectively. These early layers encode low-level texture and edge features that are fundamental to the subsequent feature pyramid. Their high sensitivity confirms that the inner bottleneck pruning strategy, which distributes channel removal across all Bottleneck blocks rather than concentrating it in any single sensitive layer, is an appropriate design choice for preserving post-pruning recoverability.

4.4 Pruning criteria comparison

To evaluate the choice of the L1 norm as the pruning criterion, a proxy ablation compared L1, L2, and random channel selection at a ratio of 0.30, without fine-tuning. All three criteria produced the same parameter count and FLOPs, since the number of removed channels was identical and only the selection policy differed. Table 5 and Figure 9 report the resulting proxy mAP50 drops.

Table 5. Pruning criteria comparison at ratio = 0.30, proxy evaluation without fine-tuning

Criterion	mAP50 (Before FT)	mAP50 Drop	FPS (no FT)	Params	FLOPs (G)
L2	0.7316	0.0893	241.63	2,462,106	6.12
Random	0.7105	0.1105	244.32	2,462,106	6.12
L1 (ours)	0.7095	0.1115	277.00	2,462,106	6.12

Note: FLOPs = Floating Point Operations; FPS = rames per second.

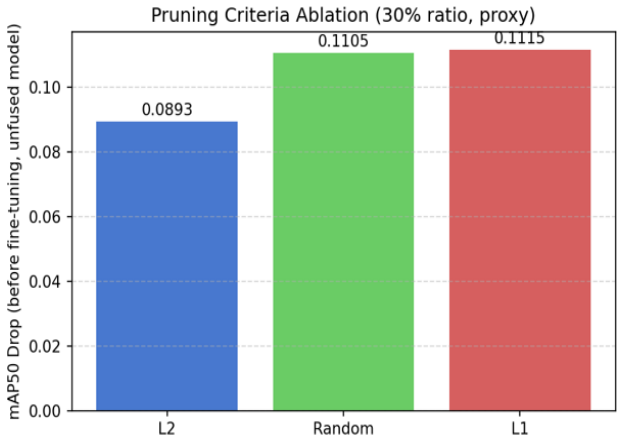


Figure 9. Proxy mAP50 drop before fine-tuning for L1, L2, and random pruning criteria at 30% ratio

The proxy evaluation showed that L1 produced the largest pre-fine-tuning mAP50 drop (0.1115), followed by Random (0.1105) and L2 (0.0893). The L1 criterion selects the channels with the smallest absolute weight magnitudes, which are the channels that the network currently uses least. After fine-tuning, all three criteria converged to similar final accuracy, with a delta from baseline of less than 0.01 in each case, consistent with the interpretation that magnitude-based pruning produces models that are more amenable to recovery, since the removed channels were already contributing little to the representation. L1 was selected over L2 for two reasons: its proxy inference speed was the highest among the three criteria (277.00 FPS, versus 241.63 for L2 and 244.32 for Random), and it is computationally simpler to calculate. The post-fine-tuning accuracy difference across criteria was negligible.

4.5 Precision-recall curve and F1-score analysis

Precision-recall curves and F1-score curves are reported for fold 0 across all three model configurations, which is representative of the cross-validation distribution. Figures 10 through 15 show the PR and F1 curves for the baseline, pruned_30, and pruned_50 models.

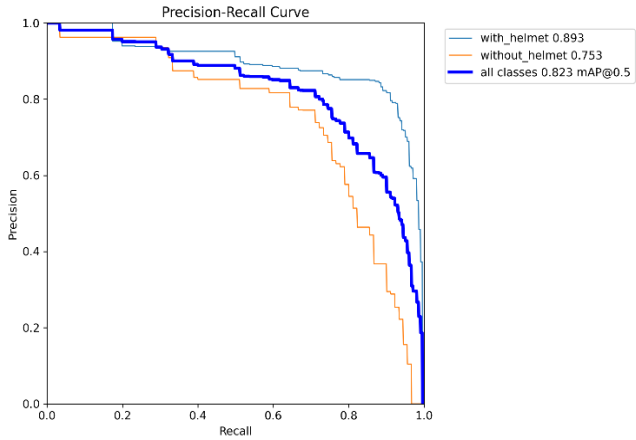


Figure 10. Precision-recall curve, baseline model (fold 0)

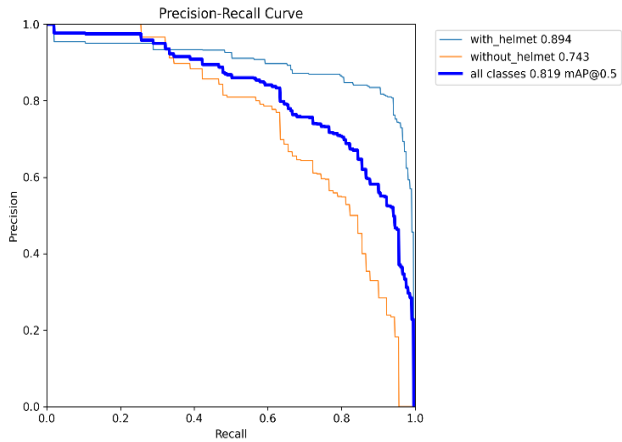


Figure 11. Precision-recall curve, pruned_30 model (fold 0)

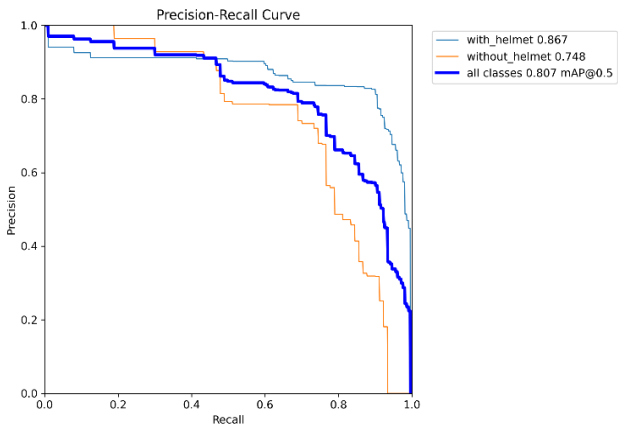


Figure 12. Precision-recall curve, pruned_50 model (fold 0)

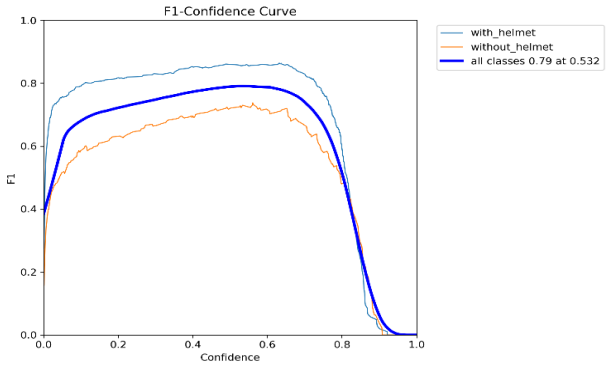


Figure 13. F1-score curve, baseline model (fold 0)

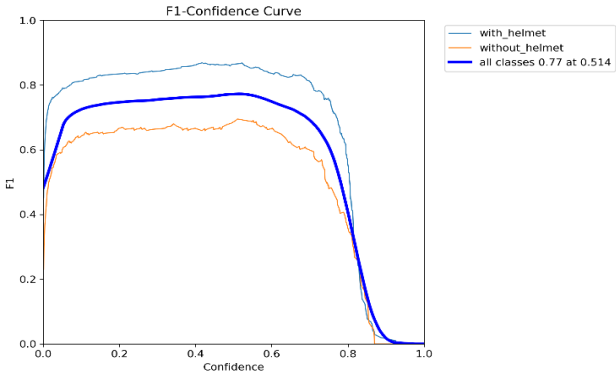


Figure 14. F1-score curve, pruned_30 model (fold 0)

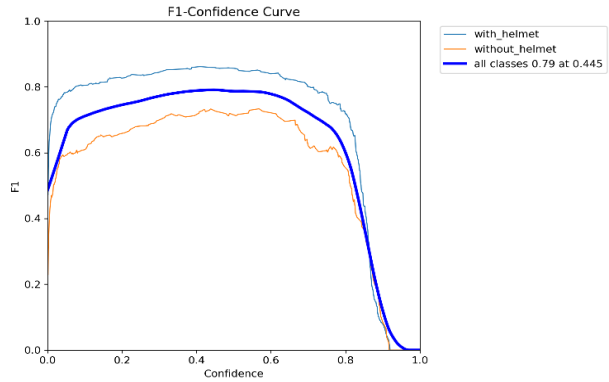


Figure 15. F1-score curve, pruned_50 model (fold 0)

The precision-recall curves show that the pruned models maintained an area under the curve close to that of the baseline. The curve shapes were consistent across the three configurations, with no visible collapse in the high-recall region. Precision at recall = 1.0 was slightly lower for the pruned models, indicating a marginal increase in false positives at the extreme of the recall range. The F1-score curves confirmed that the optimal confidence threshold shifted only slightly after pruning, and the peak F1 value remained comparable across configurations. For helmet detection, where missing a violation is more costly than generating a false alarm, this slight precision-recall trade-off introduced by pruning is acceptable.

4.6 Statistical significance

Table 6 reports the results of the paired t-test and Wilcoxon signed-rank test for mAP50 and FPS comparisons between the baseline and each pruned model, using the 5 folds results as matched pairs.

No comparison reached statistical significance at alpha = 0.05. For mAP50, p = 0.42 for pruned_30 and p = 0.95 for pruned_50, confirming that pruning did not significantly degrade accuracy at either ratio. The FPS differences were also not statistically significant (p = 0.53 for pruned_30 and p = 0.10 for pruned_50). This indicates that the observed throughput increases should be interpreted with caution given the variance across folds; the n = 5 sample size limits statistical power, and the p = 0.10 result for pruned_50 is a borderline finding that warrants evaluation on additional data. The absence of significance for mAP50 is the primary result of interest, as it provides statistical evidence that the pruned models performed equivalently to the baseline on this dataset.

4.7 Comparison with YOLOv8n on identical hardware

To provide a fair same-hardware baseline, YOLOv8n was trained and evaluated on the same dataset and hardware configuration, NVIDIA T4, used for the revision experiments. Table 7 compares YOLOv8n against the three YOLOv11n configurations.

The pruned_50 YOLOv11n achieved an mAP50 of 0.7882, exceeding YOLOv8n at 0.7733, while using 21% fewer parameters (2.378M versus 3.011M), 20.5% less ONNX storage (9.30 MB versus 11.70 MB), and 26.6% fewer FLOPs (5.97 G versus 8.14 G). FPS for pruned_50 (238.42) also exceeded YOLOv8n (211.67). This comparison establishes that pruning YOLOv11n produces a model that is simultaneously more accurate, faster, and smaller than the previous-generation architecture trained under equivalent conditions.

Table 6. Statistical significance tests for mAP50 and frames per second (FPS) (n = 5 folds, alpha = 0.05)

Comparison	Metric	Mean Baseline	Mean Pruned	Delta	P (t-Test)	P (Wilcoxon)	Significant
Baseline vs Pruned 30%	mAP50	0.7889	0.7962	+0.0073	0.4222	0.8125	No
Baseline vs Pruned 30%	FPS	203.49	218.17	+14.68	0.5333	0.6250	No
Baseline vs Pruned 50%	mAP50	0.7889	0.7882	-0.0006	0.9540	0.8125	No
Baseline vs Pruned 50%	FPS	203.49	238.42	+34.93	0.1018	0.1250	No

Table 7. Direct comparison of YOLOv11n configurations with YOLOv8n on NVIDIA T4 hardware and identical dataset

Model	mAP50	FPS	Params (M)	ONNX (MB)	FLOPs (G)	Hardware
YOLOv11n Baseline	0.7889	203.49	2.590	10.11	6.38	NVIDIA T4
YOLOv11n Pruned 30%	0.7962	218.17	2.462	9.63	6.12	NVIDIA T4
YOLOv11n Pruned 50%	0.7882	238.42	2.378	9.30	5.97	NVIDIA T4
YOLOv8n	0.7733	211.67	3.011	11.70	8.14	NVIDIA T4

Note: FLOPs = Floating Point Operations; FPS = Frames per second; ONNX = Open Neural Network Exchange; mAP = Mean Average Precision.

4.8 Detection quality analysis

The confusion matrices for fold 0 across the three configurations are shown in Figures 16 through 18. The pattern across configurations is consistent with the quantitative mAP50 results reported in Table 2. The baseline model correctly classified the majority of with_helmet and without_helmet instances, with a small proportion of false negatives concentrated in occluded or small-scale instances. The pruned_30 model showed a comparable classification pattern, with no systematic shift in the false positive or false negative distribution relative to the baseline. The pruned_50 model similarly maintained a confusion structure close to the baseline, consistent with the statistically negligible mAP50 delta of -0.0006 ($p = 0.95$) reported in Table 6. These observations confirm that channel removal at either ratio did not introduce a systematic bias toward either class.

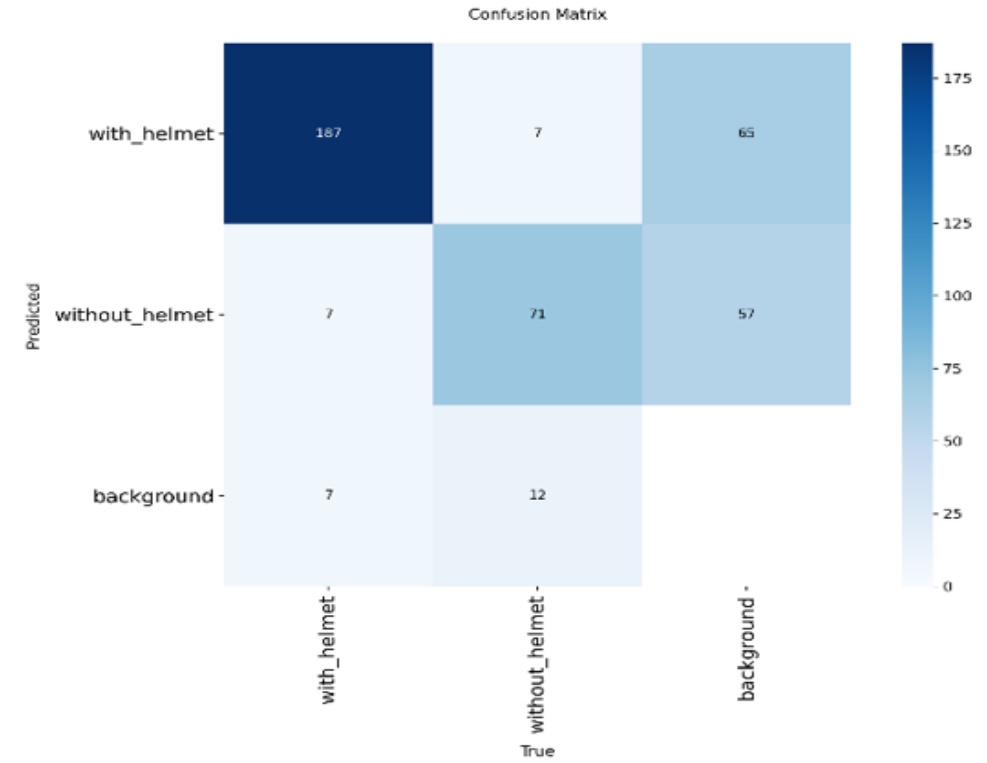


Figure 16. Confusion matrix model baseline (fold 0)

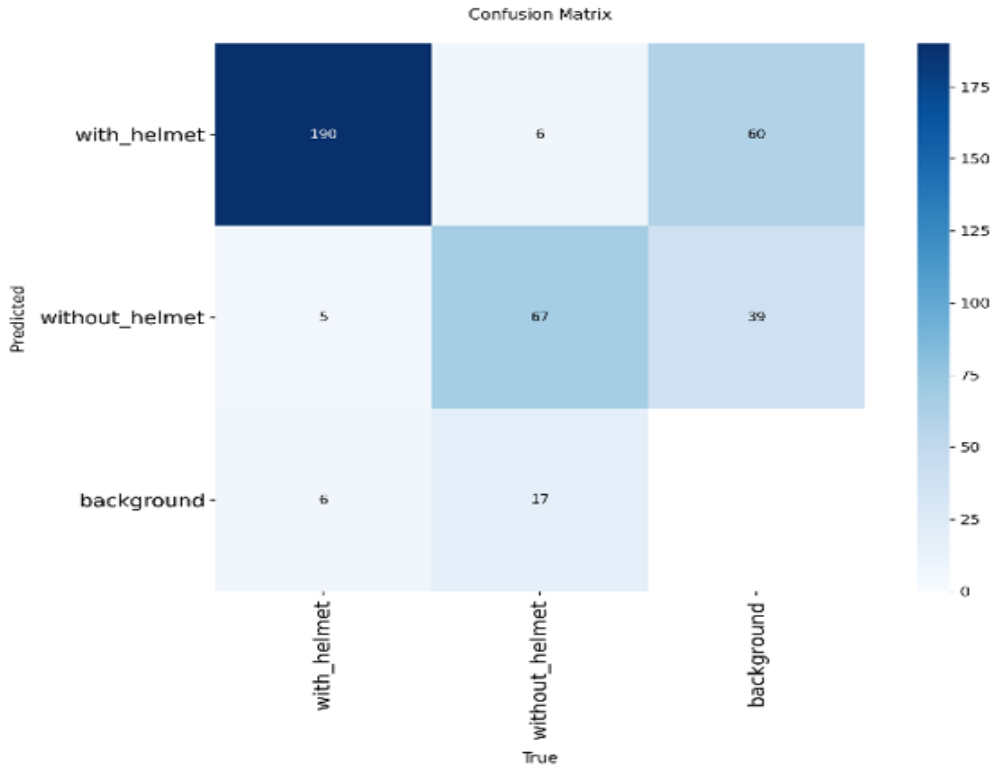


Figure 17. Confusion matrix model pruned 30% (fold 0)

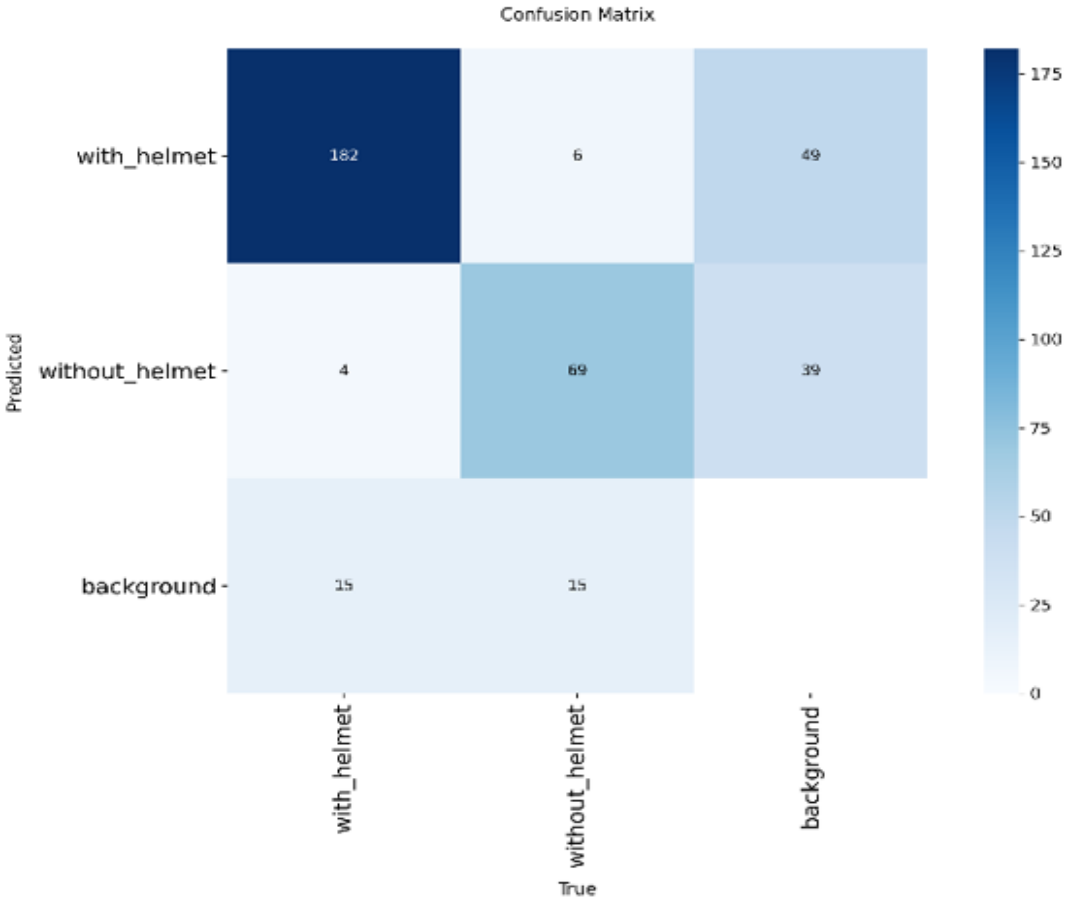


Figure 18. Confusion matrix model pruned 50% (fold 0)

Figure 19. Detection results sample for baseline (fold 0)

Figure 20. Detection results sample for prune 30% (fold 0)

Figure 21. Detection results sample for prune 50% (fold 0)

Figures 19 through 21 show representative detection outputs from fold 0 for the baseline, pruned_30, and pruned_50 configurations. Across all three configurations, the models successfully localized and classified riders in multi-person scenes, including partially occluded subjects and small-scale instances in the mid-ground. Minor differences in predicted confidence scores were observable between configurations at the level of individual detections, reflecting the small distributional shift introduced by fine-tuning after channel removal. These differences did not manifest as systematic missed detections at the confidence threshold of 0.25 used during evaluation, consistent with the near-identical precision and recall values across configurations reported in Table 2. The qualitative outputs support the conclusion that inner bottleneck pruning at both tested ratios preserves the visual detection capability of the model within the conditions represented by this dataset.

5. DISCUSSION

The 5-fold cross-validation results confirmed that structured L1 channel pruning on the inner bottleneck pairs of YOLOv11n does not significantly degrade mAP50 at either the 30% or 50% pruning ratio. The mAP50 delta of -0.0006 for pruned_50 ($p = 0.95$) represents statistical equivalence with the baseline. The slight mAP50 increase for pruned_30 (+0.0073, $p = 0.42$) is consistent with a mild regularization effect from fine-tuning: removing lower-magnitude channels reduces overfitting to the training distribution, and the subsequent 20-epoch fine-tuning on the remaining capacity allows the model to converge to a slightly more generalizable solution. Neither result represents a genuine architectural improvement; both fall within the normal variance range observed across folds.

The compression ratios achieved, 1.052x and 1.089x, are modest compared with methods that apply global channel pruning or remove entire layers. This outcome is a direct consequence of the inner bottleneck scope, which was selected specifically to avoid dimensional conflicts at Concat operations in the FPN-PAN structure. Methods such as NS-YOLO (88% parameter reduction) and A-pruning (76% reduction) operate on YOLOv5, which has a simpler skip-connection topology that accommodates more aggressive global pruning [12, 27]. For YOLOv11n, the C3K2 and C2PSA modules with their multi-path cross-stage connections impose stricter constraints. The inner bottleneck approach trades compression depth for architectural safety and recoverability, a trade-off that the 5-fold results confirm is sound for this architecture.

The pruning criteria comparison provides insight into why L1 was selected. The proxy mAP50 drop for L1 (0.1115) exceeded those of Random (0.1105) and L2 (0.0893), indicating that L1 selected channels that, when removed, initially disrupted the representation more than the channels selected by L2. This result may appear counterintuitive, since L1 is a magnitude-based criterion that targets the smallest weights. The explanation is that L2 penalizes large deviations from zero more strongly, so the channels it selects tend to have intermediate magnitudes and may contribute more evenly distributed redundancy. L1 selects the absolute minimum contributors, which can include channels whose small magnitude results from learned suppression rather than genuine redundancy. After 20 epochs of fine-tuning, however, all three criteria converged to comparable final accuracy, confirming that the initial disruption is recoverable and that the choice of criterion matters primarily for the speed of convergence during fine-tuning. L1 was favored because its computational cost is lower, one absolute-value sum per channel versus a squared sum for L2, and its proxy inference speed was the highest among the three criteria (277.00 FPS, versus 241.63 for L2), suggesting that the channels it removes are concentrated in layers where their removal yields more consistent computational savings. Importance criteria that explicitly incorporate hardware energy consumption rather than weight magnitude alone have also been proposed [37] and represent a possible direction for further refining the channel selection policy used in this work.

The direct comparison with YOLOv8n under identical hardware and data conditions addresses a limitation common to SOTA comparisons that mix different hardware platforms. YOLOv11n pruned_50 outperformed YOLOv8n on all reported metrics, including mAP50, FPS, parameter count, ONNX size, and FLOPs. This result suggests that the architectural improvements introduced in YOLOv11, namely the C3K2 blocks and C2PSA attention modules, provide a richer feature representation that is maintained even after channel reduction, whereas YOLOv8n, despite having no pruning applied, achieved lower accuracy with more parameters. The implication is that YOLOv11n is a more efficient starting point for compression than its predecessor for this task.

Table 8. Comparison with state-of-the-art structured pruning methods. Negative drop indicates accuracy improvement after pruning

Method	Base Model	Pruning Ratio	Accuracy Drop (%)	Speed Gain (%)	Hardware
NS-YOLO [12]	YOLOv5	88%	1.60	+35.5	Android (NCNN)
A-pruning [27]	YOLOv5	76%	0.90	+170.0	Jetson Xavier NX
Target Capacity [28]	YOLOv5	30%	0.60	+14.3	Jetson Xavier NX
Target Capacity [28]	YOLOv5	50%	2.90	+34.5	Jetson Xavier NX
SAFP-YOLO [29]	YOLOv7	–	6.10	+426.0	Jetson TX2
CM-YOLOv8 [30]	YOLOv8	40%	<1.00	–	General GPU
Ours – Pruned 30%	YOLOv11n	30%	-0.73 (gain)	+7.2†	NVIDIA T4
Ours – Pruned 50%	YOLOv11n	50%	+0.06	+17.2†	NVIDIA T4

Note: † A negative value in the accuracy drop column indicates an accuracy improvement after pruning. † Speed gain measured as the mean FPS improvement across 5-fold cross-validation; statistical significance was not reached at alpha = 0.05 ($p = 0.53$ for pruned_30, $p = 0.10$ for pruned_50; see Table 6).

Table 8 positions the proposed method against published pruning results. Direct performance comparison across rows must be interpreted with caution, since the hardware platforms differ substantially, ranging from embedded Jetson and Android NCNN deployments to server-grade GPUs. The hardware column is included explicitly to allow readers to

account for this confound. Relative accuracy retention and the pruning ratio are the most hardware-agnostic dimensions of comparison.

Against the methods that operate at comparable pruning ratios, the proposed approach achieved the smallest accuracy degradation. Target Capacity Pruning reported a 0.6% drop at a 30% ratio and a 2.9% drop at a 50% ratio on YOLOv5; the present work reported a 0.06 percentage-point drop at the 50% ratio, consistent with statistical equivalence to the baseline. The speed gains reported here, 7.2% and 17.2%, are more modest than those in most comparative entries, which reflects honest reporting under 5-fold cross-validation on a server-grade GPU rather than a single-run measurement. The larger gains reported by methods such as A-pruning (+170%) and SAFP-YOLO (+426%) were measured on Jetson platforms, where architectural compression interacts differently with hardware-specific inference optimizations; these results are not directly comparable to server GPU throughput. The primary contribution of the present work relative to the state of the art is accuracy retention under a formally validated experimental protocol, applied to an architecture, YOLOv11n, for which no prior pruning baseline exists.

From a practical deployment standpoint, the pruned models processed video at 218 to 238 FPS on a server-grade GPU, well above the 30 FPS threshold typically required for real-time single-stream processing. A server handling multiple simultaneous camera feeds can dedicate a portion of GPU time to each stream; at 238 FPS per model, approximately seven concurrent streams could be processed at 30 FPS on a single GPU instance. Fine-tuning the pruned model from a pre-trained checkpoint requires approximately 20 epochs, which at the dataset scale used here corresponds to a training time of roughly 20 to 30 minutes on an NVIDIA T4 GPU, making retraining for domain adaptation practical. Deployment conditions not evaluated in this study include adverse weather, such as rain, fog, or low light at night, extreme camera angles, and partial occlusion by non-helmet head coverings. These conditions are common in real traffic environments and represent open questions for deployment robustness.

Several limitations affect the scope of these conclusions. First, the dataset used, 764 images, is small for an object detection benchmark. Although 5-fold cross-validation provides more robust estimates than a single split, the absolute confidence intervals on mean performance remain wide, with a standard deviation of 0.033 for baseline mAP50. Evaluation on larger or more diverse datasets is necessary to establish generalization beyond this benchmark. Second, all speed measurements were conducted on an NVIDIA T4 server GPU. The observed FPS improvements, 7.2% for pruned_30 and 17.2% for pruned_50, may not translate proportionally to edge devices such as the Jetson Nano or Raspberry Pi, where memory bandwidth and SIMD instruction availability differ substantially. Evaluation on at least one embedded platform is required before any claims about edge deployment can be made. Third, pruning was restricted to inner bottleneck channels to avoid Concat dimension conflicts; outer channel pruning would require additional architectural handling, such as padding or channel adaptation at FPN merge points, and has not been attempted in this study.

6. CONCLUSION

This study applied structured L1 channel pruning to the inner bottleneck convolution pairs of YOLOv11n for helmet detection. Validation through 5-fold stratified cross-validation confirmed that neither pruning ratio introduced a statistically significant change in mAP50 relative to the baseline, indicating that the inner bottleneck pruning strategy preserves detection accuracy after fine-tuning. The more aggressive pruning ratio yielded a meaningful reduction in parameter count, model size, and computational cost, together with a corresponding gain in inference throughput. Under identical hardware conditions, the pruned model also outperformed YOLOv8n in both accuracy and efficiency, despite using substantially fewer parameters.

Layer sensitivity analysis identified the early backbone layers as the most susceptible to channel removal, providing guidance for future pruning strategies that aim to concentrate channel removal in less critical regions. The pruning criteria comparison showed that L1, L2, and random channel selection converge to comparable accuracy after fine-tuning, with L1 preferred for its lower computational cost and higher proxy inference speed.

The inner bottleneck scope yields a more conservative compression ratio than methods that prune global channels, but it avoids the dimensional conflicts at Concat operations that prevent the direct application of existing global pruning pipelines to YOLOv11n. This work provides a reproducible compression baseline for this architecture in the helmet detection domain. Future work should extend pruning to the outer channel dimensions with appropriate Concat handling, apply post-training quantization to further reduce model size, evaluate the pruned model on embedded hardware to assess edge deployment feasibility, and validate generalization on larger and more diverse helmet detection datasets.

REFERENCES

[1] Global Status Report on Road Safety 2023. World Health Organization, Geneva, 2023. https://www.who.int/teams/social-determinants-of-health/safety-and-mobility/global-status-report-on-road-safety-2023.

[2] Liu, B.C., Ivers, R., Norton, R., Boufous, S., Blows, S., Lo, S.K. (2008). Helmets for preventing injury in motorcycle riders. Cochrane Database of Systematic Reviews, (1). https://doi.org/10.1002/14651858.CD004333.pub3

[3] Truong, L.T., Nguyen, H.T., De Gruyter, C. (2019). Mobile phone use while riding a motorcycle and crashes among university students. Traffic Injury Prevention, 20(2): 204-210. https://doi.org/10.1080/15389588.2018.1546048

[4] Zhao, Z.Q., Zheng, P., Xu, S.T., Wu, X. (2019). Object detection with deep learning: A review. IEEE Transactions on Neural Networks and Learning Systems, 30(11): 3212-3232. https://doi.org/10.1109/TNNLS.2018.2876865

[5] e Silva, R.R.V., Aires, K.R.T., Veras, R.D.M.S. (2014). Helmet detection on motorcyclists using image descriptors and classifiers. In 2014 27th SIBGRAPI Conference on Graphics, Patterns and Images, Rio de Janeiro, Brazil, pp. 141-148. https://doi.org/10.1109/SIBGRAPI.2014.28

[6] Jocher, G., Qiu, J. (2024). Ultralytics YOLO11 (version 11.0.0). GitHub repository.

https://github.com/ultralytics/ultralytics.
[7] Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P. (2016). Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710. https://doi.org/10.48550/ARXIV.1608.08710
[8] Han, S., Pool, J., Tran, J., Dally, W.J. (2015). Learning both weights and connections for efficient neural networks. In Proceedings of the 29th International Conference on Neural Information Processing Systems-Volume 1, pp. 1135-1143.
[9] Agorku, G., Agbobli, D., Chowdhury, V., Amankwah-Nkyi, K., Ogungbire, A., Lartey, P.A., Aboah, A. (2023). Real-time helmet violation detection using YOLOv5 and ensemble learning. arXiv preprint arXiv:2304.09246. https://doi.org/10.48550/ARXIV.2304.09246
[10] Aboah, A., Wang, B., Bagci, U., Adu-Gyamfi, Y. (2023). Real-time multi-class helmet violation detection using few-shot data sampling technique and YOLOv8. In 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Vancouver, BC, Canada, pp. 5350-5358. https://doi.org/10.1109/CVPRW59228.2023.00564
[11] Terven, J., Córdova-Esparza, D.M., Romero-González, J.A. (2023). A comprehensive review of yolo architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS. Machine Learning and Knowledge Extraction, 5(4): 1680-1716. https://doi.org/10.3390/make5040083
[12] Weng, Z., Liu, K., Zheng, Z. (2023). Cattle face detection method based on channel pruning YOLOv5 network and mobile deployment. Journal of Intelligent & Fuzzy Systems, 45(6): 10003-10020. https://doi.org/10.3233/JIFS-232213
[13] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, pp. 779-788. https://doi.org/10.1109/CVPR.2016.91
[14] Redmon, J., Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767. https://doi.org/10.48550/ARXIV.1804.02767
[15] Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934. https://doi.org/10.48550/ARXIV.2004.10934
[16] Jocher, G., Stoken, A., Chaurasia, A., Borovec, J., et al. (2021). ultralytics/YOLOv5: v6. 0-YOLOv5n'Nano'models, Roboflow integration, TensorFlow export, OpenCV DNN support. Zenodo. https://doi.org/10.5281/ZENODO.5563715
[17] Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M. (2023). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, pp. 7464-7475. https://doi.org/10.1109/CVPR52729.2023.00721
[18] Hussain, M. (2024). YOLOv1 to v8: Unveiling each variant–A comprehensive review of yolo. IEEE Access, 12: 42816-42833. https://doi.org/10.1109/ACCESS.2024.3378568
[19] Reis, D., Kupec, J., Hong, J., Daoudi, A. (2023). Real-time flying object detection with YOLOv8. arXiv preprint arXiv:2305.09972. https://doi.org/10.48550/ARXIV.2305.09972
[20] Muhammad, F., Bintang, I., Fahrizal, R., Ahendyarti, C., Wiryadinata, R., Muttakin, I. (2024). Real-time motorcyclist helmet detection using YOLOv8 on edge device. In 2024 International Conference on Informatics Electrical and Electronics (ICIEE), Denpasar, Bali, Indonesia, pp. 1-7. https://doi.org/10.1109/ICIEE63403.2024.10920426
[21] Thatikonda, M. (2024). An enhanced real-time object detection of helmets and license plates using a lightweight YOLOv8 deep learning model. Master's thesis, Wright State University.
[22] Soltanikazemi, E., Dhakal, A., Hatuwal, B.K., Toubal, I.E., Aboah, A., Palaniappan, K. (2023). Real-time helmet violation detection in ai city challenge 2023 with genetic algorithm-enhanced yolov5. In 2023 IEEE Applied Imagery Pattern Recognition Workshop (AIPR), St. Louis, MO, USA, pp. i-x. https://doi.org/10.1109/AIPR60534.2023.10440713
[23] Chai, Z. (2024). Real-time automatic detection of motorcycle helmet based on improved YOLOv8 algorithm. In 2024 6th International Conference on Communications, Information System and Computer Engineering (CISCE), Guangzhou, China, pp. 1239-1243. https://doi.org/10.1109/CISCE62493.2024.10652610
[24] Larxel. Helmet Detection Dataset. Kaggle, 2020. https://www.kaggle.com/datasets/andrewmvd/helmet-detection.
[25] Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440. https://doi.org/10.48550/arXiv.1611.06440
[26] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE International Conference on Computer Vision, pp. 2736-2744. https://doi.org/10.1109/ICCV.2017.298
[27] Yu, G., Cai, R., Luo, Y., Hou, M., Deng, R. (2024). A-pruning: A lightweight pineapple flower counting network based on filter pruning. Complex & Intelligent Systems, 10(2): 2047-2066. https://doi.org/10.1007/s40747-023-01261-7
[28] Jeon, J., Kim, J., Kang, J.K., Moon, S., Kim, Y. (2022). Target capacity filter pruning method for optimized inference time based on YOLOv5 in embedded systems. IEEE Access, 10: 70840-70849. https://doi.org/10.1109/ACCESS.2022.3188323
[29] Ahn, H., Son, S., Roh, J., Baek, H., Lee, S., Chung, Y., Park, D. (2023). SAFP-YOLO: Enhanced object detection speed using spatial attention-based filter pruning. Applied Sciences, 13(20): 11237. https://doi.org/10.3390/app132011237
[30] Fan, Y., Mao, S., Li, M., Wu, Z., Kang, J. (2024). CM-YOLOv8: Lightweight YOLO for coal mine fully mechanized mining face. Sensors, 24(6): 1866. https://doi.org/10.3390/s24061866
[31] Zheng, X., Guan, Z., Chen, Q., Wen, G., Lu, X. (2025). A lightweight road traffic sign detection algorithm based on adaptive sparse channel pruning. Measurement Science and Technology, 36(1): 016176. https://doi.org/10.1088/1361-6501/ad9517
[32] Hinton, G., Vinyals, O., Dean, J. (2015). Distilling the

knowledge in a neural network. arXiv preprint arXiv:1503.02531. https://doi.org/10.48550/ARXIV.1503.02531

[33] Jacob, B., Kligys, S., Chen, B., Zhu, M., et al. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, pp. 2704-2713. https://doi.org/10.1109/CVPR.2018.00286

[34] Zhang, Z., Yang, Y., Xu, X., Liu, L., et al. (2024). GVC-YOLO: A lightweight real-time detection method for cotton aphid-damaged leaves based on edge computing. Remote Sensing, 16(16): 3046. https://doi.org/10.3390/rs16163046

[35] He, K., Zhang, X., Ren, S., Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(9): 1904-1916. https://doi.org/10.1109/TPAMI.2015.2389824

[36] Fang, G., Ma, X., Song, M., Mi, M.B., Wang, X. (2023). DepGraph: Towards Any Structural Pruning. arXiv preprint arXiv:2301.12900. https://doi.org/10.48550/arXiv.2301.12900

[37] Yang, T.J., Chen, Y.H., Sze, V. (2017). Designing energy-efficient convolutional neural networks using energy-aware pruning. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, pp. 6071-6079. https://doi.org/10.1109/CVPR.2017.643