electronics

Article

Vehicle and Pedestrian Detection Based on Improved YOLOv7-Tiny

Zhen Liang †, Wei Wang †, Ruifeng Meng *,†, Hongyu Yang, Jinlei Wang, He Gao, Biao Li and Jungeng Fan

School of Aviation, Inner Mongolia University of Technology, Hohhot 010021, China; 20221800753@imut.edu.cn (Z.L.); 20211800706@imut.edu.cn (W.W.); 20221800750@imut.edu.cn (H.Y.); 20231800822@imut.edu.cn (J.W.); 20241800848@imut.edu.cn (H.G.); 20230000036@imut.edu.cn (B.L.); baotsinghua@imut.edu.cn (J.F.)

* Correspondence: mrfnmgcn@imut.edu.cn

† These authors contributed equally to this work.

Abstract: To improve the detection accuracy of vehicles and pedestrians in traffic scenes using object detection algorithms, this paper presents modifications, compression, and deployment of the single-stage typical algorithm YOLOv7-tiny. In the model improvement section: firstly, to address the problem of small object missed detection, shallower feature layer information is incorporated into the original feature fusion branch, forming a four-scale detection head; secondly, a Multi-Stage Feature Fusion (MSFF) module is proposed to fully integrate shallow, middle, and deep feature information to extract more comprehensive small object information. In the model compression section: the Layer-Adaptive Magnitude-based Pruning (LAMP) algorithm and the Torch-Pruning library are combined, setting different pruning rates for the improved model. In the model deployment section: the V7-tiny-P2-MSFF model, pruned by 45% using LAMP, is deployed on the embedded platform NVIDIA Jetson AGX Xavier. Experimental results show that the improved and pruned model achieves a 12.3% increase in mAP@0.5 compared to the original model, with parameter volume, computation volume, and model size reduced by 76.74%, 7.57%, and 70.94%, respectively. Moreover, the inference speed of a single image for the pruned and quantized model deployed on Xavier is 9.5 ms.

Keywords: vehicle and pedestrian detection; multi-stage feature fusion; model pruning; NVIDIA Jetson AGX Xavier

check for updates

Citation: Liang, Z.; Wang, W.; Meng, R.; Yang, H.; Wang, J.; Gao, H.; Li, B.; Fan, J. Vehicle and Pedestrian Detection Based on Improved YOLOv7-Tiny. *Electronics* **2024**, *13*, 4010. https://doi.org/10.3390/electronics13204010

Received: 29 August 2024
Revised: 21 September 2024
Accepted: 6 October 2024
Published: 12 October 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In recent years, the global increase in the number of vehicles has raised significant safety concerns. According to the World Health Organization (WHO), road traffic crashes result in the deaths of approximately 1.35 million people each year, and between 20 and 50 million more suffer non-fatal injuries, with many incurring a disability as a result of their injury [1]. In 2022, China alone reported 256,409 traffic accidents, with 157,407 involving vehicles, accounting for 61.4% of all traffic accidents and causing injuries to more than 149,000 people. Statistics indicate that over 90% of road traffic accidents worldwide are caused by human factors, highlighting the urgent need for improved road safety measures and advanced technologies to mitigate these risks [2].

The advent of big data and the rapid rise of artificial intelligence have spurred the development of autonomous driving technology, which is seen as an innovative solution to issues like driver fatigue and operational errors, enhancing driving safety and reducing losses of life and property. Object detection, a key component of the perception module in autonomous driving, is crucial for subsequent planning, decision-making, and control tasks. The primary objective of object detection is to classify and locate targets within images or videos [3]. Traditional vehicle detection methods are primarily based on motion and appearance features. Motion-based methods include frame differencing, optical flow, and background subtraction. Frame differencing detects targets by computing differences

between successive frames [4]. Optical flow assigns velocity vectors to each pixel to form an image motion field, analyzing these vectors to understand motion [5]. Background subtraction involves creating a suitable background model and detecting changes by subtracting the current frame from the background [5]. However, these methods are susceptible to noise and weather conditions and require extensive computational resources.

Appearance-based traditional target detection algorithms focus on selecting regions, extracting features, and determining classifiers. Early machine learning detection methods used handcrafted features combined with classifiers such as Support Vector Machines (SVM) [6]. Classic methods for selecting regions include Scale-Invariant Feature Transform (SIFT) [7] and Histogram of Oriented Gradients (HOG) [8]. In 2008, Felzenszwalb P F et al. introduced the Deformable Part Model (DPM), an effective traditional target detection algorithm that combines HOG and SVM, winning the PASCAL VOC detection challenge for three consecutive years. Despite its effectiveness, DPM is complex and slow in detection [9]. Although traditional feature extraction and classification algorithms have shown acceptable results for vehicle and pedestrian detection in simple traffic scenarios, their performance is limited by environmental factors such as lighting and weather, as well as by detection speed. Consequently, researchers have shifted their focus to deep learning-based object detection algorithms, which offer robust feature extraction and representation capabilities and are widely used in vehicle and pedestrian detection in traffic scenarios.

Deep learning-based object detection methods have now become the most popular object detection methods for image processing. Depending on whether the region of interest needs to be extracted or not, the target detection algorithms are classified into one-level and two-level detectors. In 2014, Girshick R et al. pioneered the application of convolutional neural networks to the object detection algorithm R-CNN (Region-based Convolutional Neural Networks) used to extract the features [10], and the accuracy on the VOC2007 dataset, the accuracy of which skyrocketed by 25 percentage points, which also made target detection take a big step to deep learning; however, it has the drawbacks of complex training and slow detection speed. 2015 Fast R-CNN solves the problem of R-CNN by borrowing the idea of pyramid pooling from SPP-Net, pooling the ROIs in the input image, and fixing the output features to be of uniform size, which solves the problem of R-CNN in the training process of each feature. The CNN is in the training process for each candidate region to repeat the extraction of features to achieve the end-to-end training method, that is, the input of the whole image output results [11,12]. In the same year, Faster R-CNN [13] proposed a Region Proposal Network (RPN), where the generation time of candidate frames was greatly reduced to improve the detection speed. The first stage algorithm has no candidate region generation process and directly performs target regression and classification, the most prominent is its speed, which mainly includes the YOLO series and SSD [14], the YOLO series has become a target detection algorithm that combines both speed and accuracy after iterative updating in several versions.

Object detection in autonomous driving scenarios requires an accurate and fast perception of the surrounding environment; however, perception in complex traffic scenarios requires large and deep models to learn more features, making the models more generalized and the speed bottleneck difficult to break through. Therefore, large and deep models need to be lightweighted to reduce the computation and complexity of the model, so as to improve the inference speed of the model.

There are two main methods for model lightweighting. One is to use lightweight backbone networks directly in model selection: MobileNet [15] uses a depth-separable convolutional optimization network, ShuffleNet [16] uses point-by-point group convolution and channel shuffling, and EfficientNet [17] uniformly adjusts the depth, width, and resolution, all of which are able to balance the latency and accuracy, and have been widely used in mobile and embedded vision. Another approach to model lightweighting is to selectively remove redundant network layers and channels using network pruning techniques. A deep compression method is proposed to reduce the storage requirements of neural networks by a factor of 35 to 49 through pruning, training quantization, and Hoff-

man coding [18]. While maintaining accuracy, a per-channel scaling model compression method is proposed to achieve a parameter compression ratio of 70.6% on the YOLOv3 detector, without compromising accuracy [19]; EagleEye pruning, fine-tuning, and knowledge distillation resulted in a 60.6% reduction in model parameters and 47.9% reduction in computation for YOLOv5s [20]; using a pruning algorithm that combines global channel pruning thresholds and L1 paradigms reduced the number of channels by 84% and computation by 17%, increasing inference speed up to 21ms while maintaining accuracy [21]. Combining model improvement with lightweighting techniques can ensure that the model has a high accuracy without a significant decrease in detection speed, which is of great significance for the practical application of the model.

These studies collectively indicate that deep learning and CNNs play a pivotal role in advancing vehicle detection technologies. Researchers continue to innovate by improving network architectures, integrating multi-sensor data, and optimizing computational efficiency to meet the growing demands of real-time intelligent transportation systems globally. Despite numerous advancements in vehicle and pedestrian detection algorithms, real-world applications of these improved methods are still constrained by the trade-off between model complexity and speed. As model complexity increases, it can learn deeper feature representations, but it also impacts inference speed, leading to a loss of real-time performance. Therefore, designing object detection models requires balancing complexity and inference speed to meet edge deployment and practical application requirements, which remains a significant research focus. The YOLO series of object detection algorithms, through versions v2, v3, v4, v5, and v6 [22–26], have maintained the fast detection philosophy of YOLO while continuously improving accuracy to meet industrial real-time needs.

- This paper focuses on improving the YOLOv7-tiny model [27], proposing an enhancement scheme that adds small target detection layers to form a four-scale detection head and redefines the K-means clustering standard for anchor boxes suitable for our dataset. We incorporate a Multi-Stage Feature Fusion (MSFF) module in the neck to thoroughly fuse shallow, medium, and deep feature information, extracting more detailed small target information;
- We combine the Layer-Adaptive Magnitude-based Pruning (LAMP) algorithm with the Torch-Pruning library, applying various pruning rates to the improved model;
- Finally, we deploy the LAMP-pruned 45%V7-tiny-P2-MSFF model on the NVIDIA Jetson AGX Xavier embedded platform. Training on a large autonomous driving dataset yields a model that can accurately and quickly recognize vehicles from a dashcam perspective, with experimental validation showing excellent detection performance.

The rest of this work is organized as follows: We present the related groundwork in Section 2. Section 3 introduces the overview of YOLOv7-tiny and our improved MSFF-YOLO. Section 4 introduces the experiments and analyzes the results. Finally, we give the conclusions of this paper and prospective future research work. For the reader's convenience, we have included some of the main abbreviations, and their corresponding full forms, used in this paper in Table 1.

Table 1. The main abbreviations and their corresponding full names.

Abbreviation	Full Form
CNN	Convolutional Neural Network
RCNN	Region-based Convolutional Neural Network
SSD	Single Shot Multi Box Detector
FPN	Feature Pyramid Network
PAN	Path Aggregation Network
LAMP	Layer-Adaptive Magnitude-based Pruning
FLOPs	Floating Point Operations

2. Related Works

2.1. CNN-Based Object Detection

Since the advent of deep learning, Convolutional Neural Networks (CNNs) have become crucial for image feature extraction and vehicle detection. Globally, numerous studies have demonstrated significant advancements in this field using various CNN-based approaches [28]. One notable work by Jabar Mahmood and colleagues provides a comprehensive review of vehicle detection and classification using deep learning techniques. They discuss the application of CNNs in traffic monitoring, fleet management, and autonomous driving, emphasizing the need for robust real-time performance. This review highlights the importance of integrating high-core processing units and GPUs to enhance detection accuracy and speed in real-time applications [29]. Another significant contribution is from Moort et al., who provides a detailed overview of CNN architectures for vehicle detection. They emphasize the use of convolutional, pooling, and fully connected layers in CNNs to effectively extract and classify features from input images. This structure allows for more detailed feature extraction, leading to improved detection and classification performance [30]. Additionally, a survey by Wang et al. explores the integration of multi-sensor systems, such as LiDAR, radar, and cameras, to improve vehicle detection in autonomous driving. They propose that combining these sensors can enhance the robustness and accuracy of detection algorithms, especially under various environmental conditions [31].

2.2. Multi-Scale Feature Extraction and Lightweight

To enable more accurate feature extraction and faster inference, Wiseman's [32] approach to dealing with road spatter or hail is by analyzing the amount of change in the picture. Although splashing water or hail can lead to false positives mistaking them for rigid objects, since vehicles tend to avoid these splashes, such false positives from the system do not actually affect judgment. Zhang Y et al. proposed YOLOv7-RAR [33] algorithm, the Res3Unit structure and attention mechanism are used to reconstruct the backbone network of YOLOv7, but lack lightweight optimization, and have slower inference and poorer adaptability. Qixiang Ding et al. proposed GMS-YOLO [34], improving multiscale target detection capabilities through the use of an improved multiscale feature fusion module (GMS module). An adaptive feature fusion method was introduced, excelling in both small and large object detection. Lightweight techniques such as pruning and quantization are employed to reduce the model size, optimizing its mobile deployment performance. Xiaohong Qian et al. proposed LFF-YOLO [35], a lightweight YOLO model for multiscale defect detection. It achieves efficient multiscale defect detection through a lightweight feature fusion network. The multiscale feature fusion mechanism balances the detection performance of large and small objects. The lightweight strategy involves network structure simplification and computation reduction, making it suitable for deployment on mobile devices. Shihang Wang et al. proposed YOLO-SK [36], which improves multiscale feature extraction through the SK module, enhancing detection performance. Lightweighting is achieved by adjusting the architecture to reduce computational complexity and the number of parameters, ensuring real-time performance on embedded devices or mobile platforms. Yue Guo et al. proposed LMSD-YOLO [37], employing feature pyramid (FPN) technology to enhance multiscale object detection capabilities. The lightweight strategy focuses on pruning, weight quantization, and other techniques, enabling efficient model operation on mobile devices or in resource-constrained environments. Daniel Padilla Carrasco et al. proposed T-YOLO [38], optimizing small vehicle detection through the different levels of features in the multiscale convolutional neural network (MSCNN). Lightweighting is achieved through structural simplification and convolution optimization, ensuring its efficient operation on edge devices.

3. Method

3.1. Overview and Key Components of YOLOv7-Tiny

The network structure of YOLOv7-tiny, as shown in Figure 1, is mainly divided into four parts: Input, Backbone, Neck, and Head. In the input part, data augmentation techniques such as MixUp and Mosaic, along with an automatic anchor box clustering method, are employed. The backbone network is simplified from the ELAN-W structure in YOLOv7, where ELAN-t reduces the stacking of two basic modules compared to ELAN-W, and the downsampling factor of the backbone network is maximized at 32. The generated three feature maps are then sent to the feature fusion part. The feature fusion part adopts an FPN+PAN structure, which not only transmits high-level semantic features from top to bottom but also transmits positional features from bottom to top. In the detection head part, YOLOv7 utilizes implicit knowledge from YOLOR, which allows parameter fusion into 3×3 convolutions after training, thereby reparameterizing and converting IDetect to Detect for classification and regression tasks. This setup uses three feature maps of different sizes to predict objects at various scales.

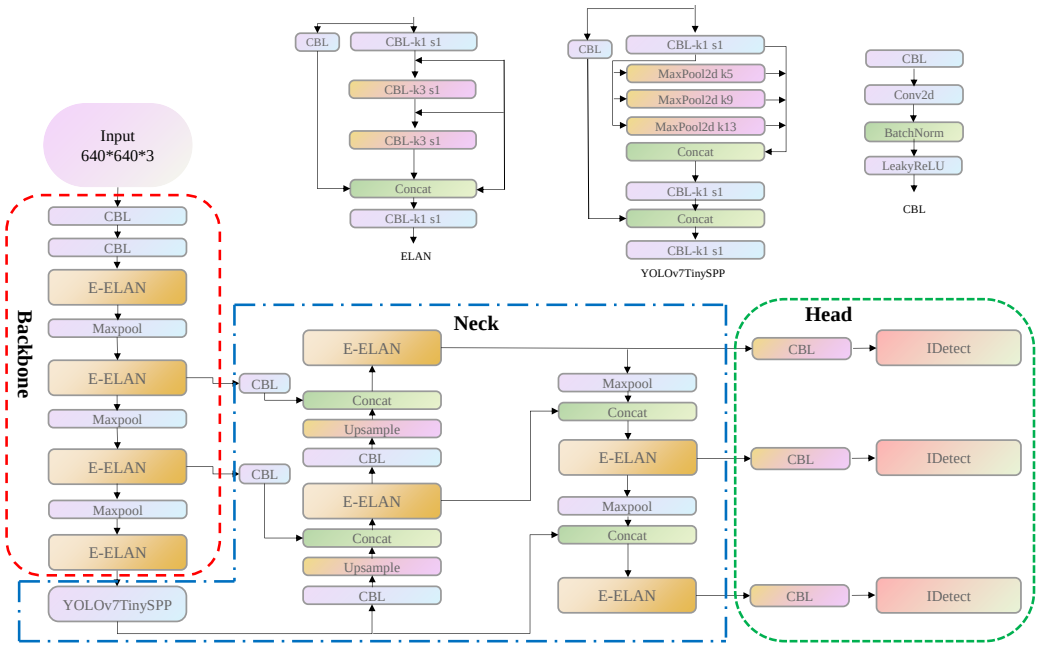


Figure 1. The network structure of YOLOv7-tiny.

3.2. Model Improvement

3.2.1. Re-Clustering Anchors with K-Means and Four-Scale Detection Head

To address the issue of missing small targets, we added shallower feature layer information to the existing feature fusion pathway, forming a four-scale detection head. Specifically, this involves adding the E-ELAN feature extraction module to extract upsampled feature information from the neck network and fusing it with the backbone information, providing features for the new detection head. The P2 layer is located in the shallow layer, which is able to capture more fine-grained edge and texture information compared to deeper features, giving it more accurate localization information when dealing with small objects, thus improving the overall detection performance. Moreover, small targets with smaller widths and larger heights (Pedestrians and Cyclists) are more frequent in this paper's dataset. The anchors are then re-clustered to match the distribution and number of detection heads in this dataset.

The YOLOv7 source code inherits the K-means clustering algorithm from YOLOv5, where the classification between different clusters relies on Euclidean distance. Given the considerable feature differences in various scales within this dataset, and the fact that Euclidean distance only considers the magnitude of feature points, specifically the

width and height of anchor boxes in the YOLO algorithm, the clustering may become insensitive to smaller dimensions when the width of two anchors differs greatly. Therefore, this study employs 1-IoU as the metric, with the new anchor clustering formula provided in Equation (1). This formula re-clusters the labeled boxes in the dataset to obtain new anchors, as shown in Table 2, which compares the original and improved anchors.

$$D = 1 - IoU(bboxs, anchors) \tag{1}$$

In the formula, D denotes the distance metric, which takes values ranging from 0 to 1, and is used to measure the difference between predicted bounding boxes (bboxes) and anchor boxes (anchors), IoU represents the Intersection over Union operation, bboxs refers to the ground truth boxes, and anchors are the prior boxes. The Intersection over Union operation measures the overlapping area between two boxes.

Table 2. Comparison of Anchors Before and After Improvement.

Original Anchors	Improved Anchors
	[7, 7, 6, 14, 12, 11]
[10, 13, 16, 30, 33, 23]	[9, 24, 19, 16, 27, 21]
[30, 61, 62, 45, 59, 119]	[17, 39, 48, 24, 38, 36]
[116, 90, 156, 198, 373, 326]	[62, 48, 91, 72, 151, 133]

3.2.2. Multi-Stage Feature Fusion Module

The main idea of the MSFF [39] module is to incorporate shallow backbone features into the feature fusion parts (P2, P3, and P4 layers) of the YOLOv7-tiny neck and extract information from the corresponding feature layers of the network. Considering that the P5 layer is a high-level semantic feature layer, it contains less specific information about small targets and has a large number of channels, making its computational cost very high. Therefore, the MSFF structure is only applied to the P2, P3, and P4 feature fusion layers. The specific implementation process of the Multi-Stage Feature Fusion module is shown in Figure 2.

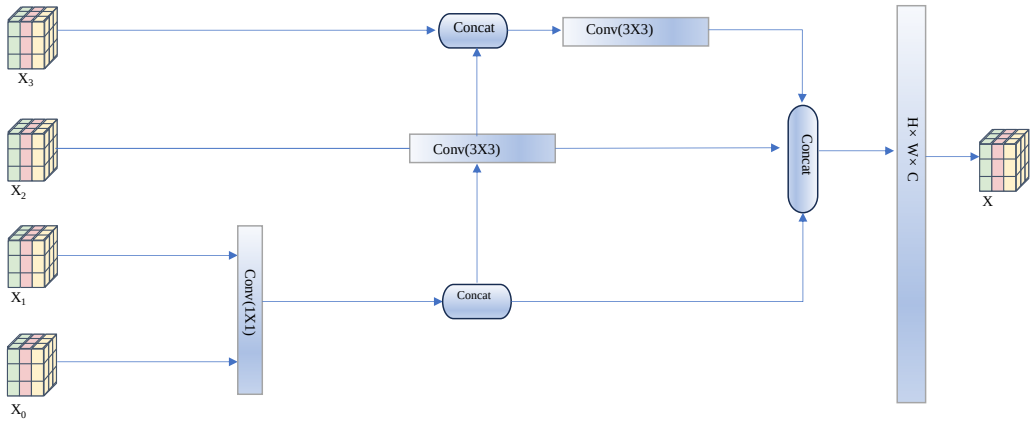


Figure 2. The network structure of MSFF.

The formulas used in the fusion process are represented by Equation (2).

$$\begin{aligned} W_0 &= \mathrm{Conv}_{1\times1}(X_0) \\ W_1 &= \mathrm{Conv}_{1\times1}(X_1) \\ W_2 &= \mathrm{Conv}_{3\times3}(W_1 + X_2) \\ W_3 &= \mathrm{Conv}_{3\times3}(W_2 + X_3) \end{aligned} \tag{2}$$

The MSFF module is divided into four branches, but it follows different fusion paths at various fusion layers (P2, P3, and P4). Specifically, at the P2 layer, the input comes

from shallow features of the backbone network, such as the first layer (X_0) and second layer (X_1) in Figure 2. The middle layer of the module fuses the features of the first and second layers, represented as W_1. Then, X_2 uses a 3×3 convolution with a stride of 1, and the extracted feature is represented as W_2. Subsequently, the 21st layer (X_3) from the backbone network is fused with the feature layer obtained from X2, and another 3×3 convolution is applied to extract the features, represented as W_3. Finally, W_1, W_2, and W_3 are concatenated, and a 1×1 convolution is applied to adjust the number of channels; the final fused feature map is denoted as W. At the P3 layer, the input comes from the third layer in Figure 2, and after adjusting the channels, it is denoted as W_1. The middle layer of the module fuses the features from the third and nineteenth layers, and after applying a 3×3 convolution with a stride of 1, the resulting feature is represented as W_2. The 26th layer from the neck network is then fused with W_2, and another 3×3 convolution is used to extract features, represented as W_3. Finally, W_1, W_2, and W_3 are concatenated, and the channels are adjusted to obtain the final W. The operation at the P4 layer is similar to that of the three branches in the P3 layer, except for the different feature layer positions. The main difference between feature fusion in P2 and that in P3 and P4 is that an additional shallow feature layer is added in the first branch of the P2 layer to better capture the positional information of small objects. The multi-scale feature fusion module (MSFF) improves multi-scale detection capability while maintaining low computational complexity, significantly speeding up model convergence and enhancing robustness. These formulas represent the 1×1 convolution and 3×3 convolution operations, respectively, while denoting the stacking operation. The Multi-Stage Feature Fusion module enhances small target detection by incorporating shallow backbone features into the neck network.

3.3. Model Compression

3.3.1. Model Pruning

In neural networks, each layer's parameter count can be controlled by setting different sparsity levels. The goal of model pruning is to achieve a balance between sparsity and model performance. However, determining the appropriate sparsity for each layer typically requires heuristic methods or extensive hyperparameter searches, which can waste substantial resources. Therefore, this paper employs the Layer-Adaptive Magnitude-based Pruning (LAMP) [40] algorithm for pruning. This algorithm, proposed by Jaeho Lee et al. in a 2021 paper, introduces an innovative scoring mechanism that re-scales and sorts the weights in convolutional or other layers and removes the less important weights to achieve adaptive layer sparsity. Equation (3) is the core formula of the LAMP algorithm, used to compute the score for a weight $W[\mu]$, where μ and v represent the indices of the weight vectors, and $W[\mu]$ and $W[v]$ denote the weight values at indices μ and v, respectively.

$$\text{score}(\mu; \text{W}) := \frac{(\text{W}[\mu])^2}{\sum_{v \geq \mu} (\text{W}[v])^2} \tag{3}$$

Assuming the LAMP algorithm is applied to convolutional layers, its specific process is as follows: (1) Flatten the four-dimensional tensor of the convolutional layer into a one-dimensional vector, where the length of this vector is as follows: kernel width $\times$ kernel width $\times$ number of input channels $\times$ number of output channels; (2) Insert each value in the vector into a list and calculate the LAMP score for each weight index according to the formula; (3) Globally prune the connections with the lowest LAMP scores until the global sparsity constraint is met.

The weight-based LAMP is an unstructured pruning algorithm. To apply it on a general platform, it needs to be converted into a structured pruning algorithm. In 2023, Gongfan Fang et al. [41] proposed a structured pruning library suitable for any model, named Torch-Pruning. This library restructures the LAMP algorithm, converting it into a channel-level structured pruning algorithm based on normalization. It uses the LAMP weight scoring method to sum the weights in each channel's convolution kernel, thereby determining the importance of the channels.

The implementation of structured pruning in the Torch-Pruning library mainly involves two steps: (1) Modeling the dependencies between different layers and within a single layer to obtain a dependency graph D. (2) Deriving all network layer relationship group matrices G based on D.

The process of modeling dependencies between different layers and within a single layer is as shown in the Equation (4). In the formula, $\vee$ and $\wedge$ represent logical "OR" and "AND" operations, f_i^+ and f_i^- Represented inputs and outputs, and 1 indicates a guiding function that returns "True" when the condition is met. The inputs and outputs of parameterized layers (such as convolution) or non-parameterized operations (such as residual addition) f_i are denoted as I_{f_i} and O_{f_i} , respectively. The first part of the formula checks the inter-layer dependencies caused by network connectivity, while the second part checks the intra-layer dependencies introduced by shared pruning schemes between the layer's input and output. The formula explanation follows:

$$D\left(f_i^-, f_j^+\right) = \underbrace{1\left[f_i^- f_j^+\right]}_{\text{Inter-layer Dep}} \vee \underbrace{\left[i = j \wedge sch\left(f_i^-\right) = sch\left(f_j^+\right)\right]}_{\text{Intra- -ayer Dep}} \quad (4)$$

3.3.2. Model Quantization

TensorRT is a software development kit (SDK) specifically designed to optimize pre-trained deep learning models for high-performance inference. Precision calibration is a critical feature of TensorRT used for model quantization. In deep learning, higher precision floating-point numbers (FloatPoint32) are typically used to store weights and activation values to train more accurate models. However, these consume significant memory and computational resources. To reduce computational costs and model size during deployment, quantization can convert floating-point representations to lower precision fixed-point formats (FloatPoint16 or Integer8). This conversion often leads to some precision loss. The data precision ranges and types are shown in Table 3.

Table 3. Data precision ranges and types.

Data Type	Precision Range
FP32	1.18×10^{-38}–3.40×10^{38}
FP16	6.10×10^{-5}–6.55×10^{4}
INT8	−128–127

4. Experimental Content and Analysis

4.1. Experimental Content

4.1.1. Experimental Environment and Model Training Parameters

The experimental environment for this study is shown in Table 4. To ensure the fairness of subsequent improvement experiments, pre-trained weights were not used during training. The training parameters were set as follows: the number of training epochs was set to 200, the batch size to 32, and the initial learning rate to 0.008. A warm-up strategy was adopted, with the number of warm-up epochs set to 3. The initial learning rate for the warm-up was set to 0.1, with a momentum of 0.8, gradually decreasing to the initial learning rate of 0.008 over the three warm-up epochs. Subsequently, the learning rate started to decrease gradually. The SGD optimizer was used with a cosine annealing learning rate decay and L2 regularization was added to prevent overfitting due to model complexity, with the regularization coefficient set to 0.0001.

Table 4. Experimental environment.

Name	Environmental Parameters
Operating System	Ubuntu 18.04
CPU	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40 GHz
GPU	NVIDIA RTX A5000 (24 GB)
Framework	Pytorch1.11.0, CUDA11.3
Programming Language	Python3.8

4.1.2. Dataset and Evaluation Metrics

SODA10M is a large-scale 2D autonomous driving dataset released in 2021 through a collaboration between Huawei Noah's Ark Lab and Sun Yat-sen University. The dataset contains approximately 10 million unlabeled images and 10,000 labeled images. The data are sourced from 32 cities in China and includes various scenarios such as urban areas and highways, as well as different weather conditions like cloudy, rainy, and overcast. This study uses 10,000 labeled images from SODA10M for model training and testing. The dataset annotates six main categories: Pedestrian, Cyclist, Car, Truck, Tram, and Tricycle [42]. Given the scarcity of Tricycle labels and their limited presence in real traffic scenarios, Tricycle and empty labels were removed. The remaining data were randomly divided into training, validation, and test sets in a ratio of 9:1:1, resulting in 8056 training images, 896 validation images, and 995 test images.

The evaluation metrics for object detection include Precision (P), Recall (R), Average Precision (AP), Precision-Recall (P-R) curve, mean Average Precision (mAP), model size (Size), number of parameters (Parameters), computational load (Floating Point Operations, FLOPs), and frames per second (FPS). These nine metrics assess the model's performance. Precision and Recall are often conflicting metrics, with Recall usually being prioritized in practical applications. The area under the P-R curve represents the AP value, and both AP and mAP provide further judgment on model accuracy, with higher values being better. Parameters, FLOPs, and Size measure the model's complexity, while FPS measures the inference speed per image, with higher FPS indicating faster inference, beneficial for edge deployment. The formulae for P, R, AP, mAP are Equations (5)–(8).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6}$$

$$\text{AP} = \int_0^1 \text{Precision}(\text{Recall}) dR \tag{7}$$

$$\text{mAP} = \frac{1}{n} \sum_{i=1}^{n} \text{AP}_i \tag{8}$$

4.2. Experimental Results and Analysis

4.2.1. Analysis of Model Improvement Experiment Results

The MSFF module proposed in this paper is based on the shallow information of the P2 detection layer, and there is a fusion path from the P2 layer in the MSFF module, which cannot be taken out of the MSFF module alone to do the ablation experiment. Therefore, the ablation experiments in this paper are composed of baseline model, baseline model + P2 detection layer and baseline model + P2 detection layer + MSFF module.

After adding the MSFF module, the experimental results are shown in Tables 5 and 6. Analysis of Table 5 shows that after sequentially adding P2 and MSFF, the model's recall rate in both the validation and test sets significantly improved by 7.7% and 7.6%, respectively. In practical applications, greater attention is given to the model's recall rate to ensure that all positive samples are fully recognized, reducing the risk of false negatives and improving

the overall effectiveness of the object detection task. Additionally, the average precision for the five categories increased, with the Car category improving by 5.7% and 5.9% in the validation and test sets, respectively, and the Cyclist and Pedestrian categories improving by at least 7.6% and 4.5%, respectively. This demonstrates that the addition of these two modules has further enhanced the model's learning ability.

Table 5. Precision (P), Recall (R), and Average Precision (AP) of the Model After Adding MSFF.

Algorithm Model	P	R	AP (Car)	AP (Cyclist)	AP (Truck)	AP (Tram)	AP (Pedestrian)
v7-tiny(val)	80.9%	56.2%	84.0%	62.6%	72.2%	67.6%	38.9%
v7-tiny+P2	76.4%	62.4%	88.3%	68.8%	74.7%	73.5%	46.9%
v7-tiny+P2+MSFF	79.8%	63.9%	89.7%	70.2%	77.7%	74.4%	47.5%
v7-tiny(test)	78.8%	56.7%	83.6%	62.9%	74.3%	67.1%	37.0%
v7-tiny+P2	80.7%	56.7%	88.5%	66.2%	77.2%	68.5%	43.6%
v7-tiny+P2+MSFF	75.1%	64.3%	89.5%	67.4%	79.5%	70.7%	46.4%

Table 6. Detection Accuracy under COCO Standards After Adding MSFF.

Algorithm Model	AP.50	AP.50:0.95	AP.50:0.95 (Small)	AP.50:0.95 (Medium)
v7-tiny(val)	65.0%	40.9%	8.8%	33.9%
v7-tiny+P2+MSFF	72.3%	46.0%	19.9%	41.0%
v7-tiny(test)	64.7%	40.4%	9.8%	33.7%
v7-tiny+P2+MSFF	70.3%	45.2%	19.0%	40.8%

To further verify the improved model's detection performance for small targets, the inference results in txt format were converted to json format, and the COCO toolkit was used to obtain the detection accuracy for small targets under COCO evaluation standards. Analysis of the data in Table 6 shows that the detection accuracy for small targets in the validation and test sets improved by 11.8% and 10.1%, respectively, and for medium targets, the accuracy improved by 7.9% and 7.0%, respectively. The obj-loss curve during training, shown in Figure 3, indicates that the initial curve gradually converges around 150 epochs, with significant fluctuations, suggesting insufficient feature expression by the network. After adding the P2 and MSFF modules, the model begins to show a convergence trend at 75 epochs, with minimal loss fluctuations, indicating that the improved model has learned sufficient image features. The mAP value change curve, shown in Figure 4, demonstrates that after adding P2 and MSFF, both mAP@0.5% and mAP@0.5:0.95% showed significant improvements, further validating the trends observed in the obj-loss curve.

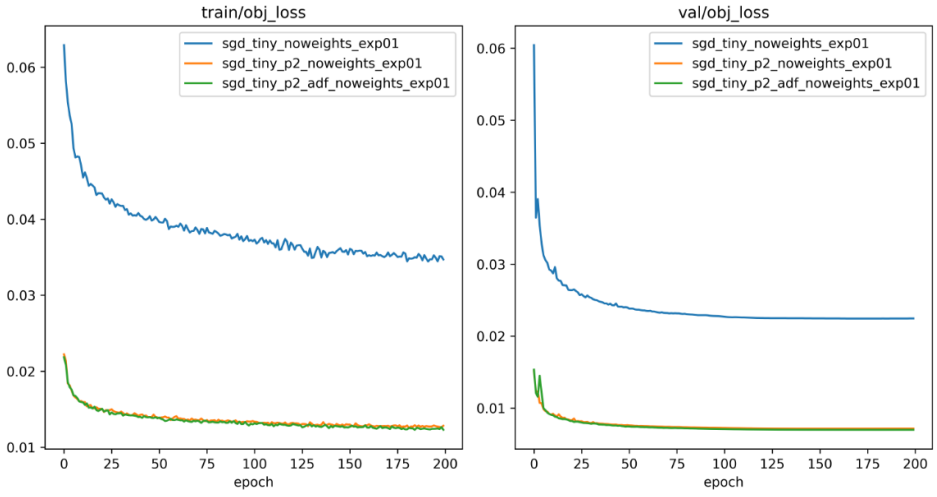


Figure 3. The obj-loss curve during training.

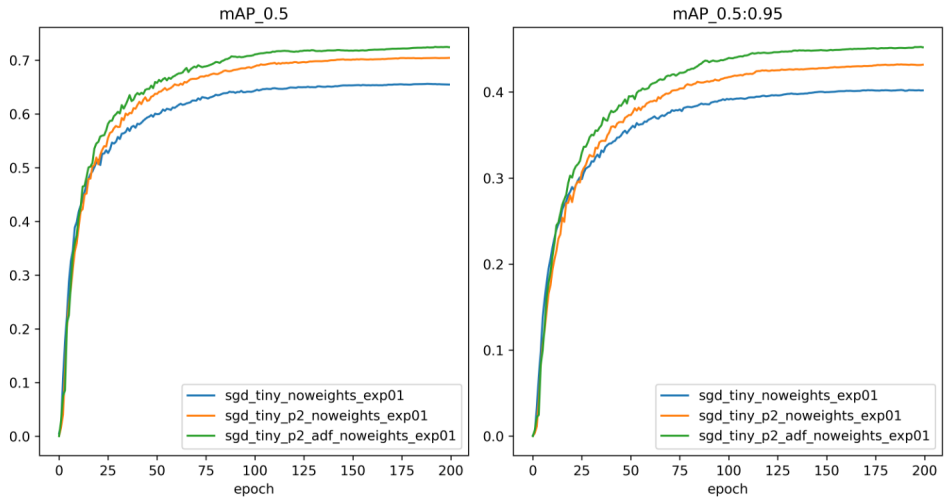


Figure 4. The mAP value change curve.

To fairly test the FPS of the improved scheme, the model trained on the server was transferred to an RTX 3060 Laptop GPU device for FPS testing. Additionally, the mAP@0.5%, mAP@0.5:0.95%, model parameters, computational load, model size, and network layers of the models obtained from the two different schemes were compared. The experimental data are shown in Table 7.

Analysis of Table 7 reveals that after sequentially adding P2 and MSFF, the model's have an mAP@0.5% and mAP@0.5:0.95% in the validation and test sets. Additionally, the network layers of the model were reduced by 362 layers, and the FPS increased by 43.7%. However, the computational load, model size, and parameter count all increased. Figure 5 shows the network structure of the improved model, with green arrows indicating the fusion paths.

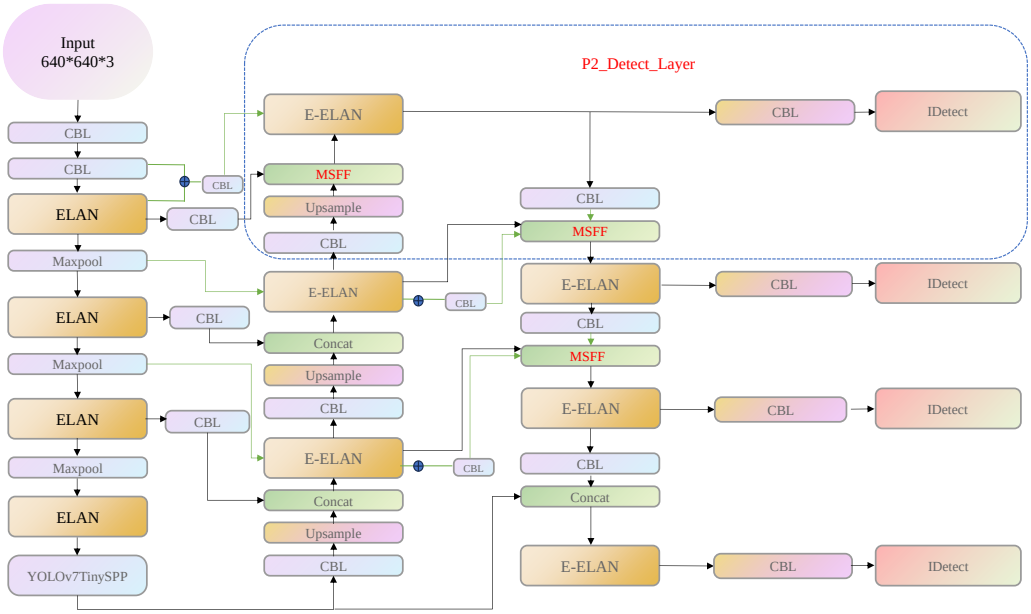


Figure 5. The network structure of the improved model.

Table 7. Performance Parameters of the Models Obtained from the Improvement Schemes.

Algorithm Model	mAP.5	mAP.5:0.95	Params/10^6	FLOPs/10^9	Size/M	FPS	Layer
v7-tiny(val)	65.1%	40.3%	6.01	13.1	9.7	35.1	172
v7-tiny+P2	70.4%	43.2%	6.11	15.4	12.1	26.0	258
v7-tiny+P2+MSFF	72.4%	45.2%	7.03	22.2	13.9	19.4	300
v7-tiny(test)	65.0%	39.9%	6.01	13.1	9.7	35.1	172
v7-tiny+P2	68.8%	42.0%	6.11	15.4	12.1	26.0	258
v7-tiny+P2+MSFF	70.7%	44.6%	7.03	22.2	13.9	19.4	300

4.2.2. Analysis of Model Pruning Experiment Results

The results of the pruning algorithm experiment for the second improvement scheme are shown in Table 8. The table indicates that after pruning with the LAMP algorithm, the model's FPS reached above 179. Additionally, the mAP values of the model at different pruning rates were higher than those achieved by the L1 [43] and Group-norm [41] pruning algorithms. The model's parameter count and size were also lower than those of the models pruned using the other two algorithms. Considering the application scenarios in this study, the v7-tiny+P2+MSFF model, pruned by 45% using the LAMP algorithm, was selected for edge deployment.

When the pruning rate is set to 45%, the comparison of the number of channels before and after pruning for the second improvement scheme is shown in Figure 6. From the figure, it can be seen that this network structure is simpler compared to Scheme One. Analyzing the data in Table 8, the model's parameter count is reduced to 20.0% of the original, and the computational load is reduced to 55.5% of the original, which also confirms the characteristics shown in the orange parts of Figure 6, where the values are higher and more numerous.

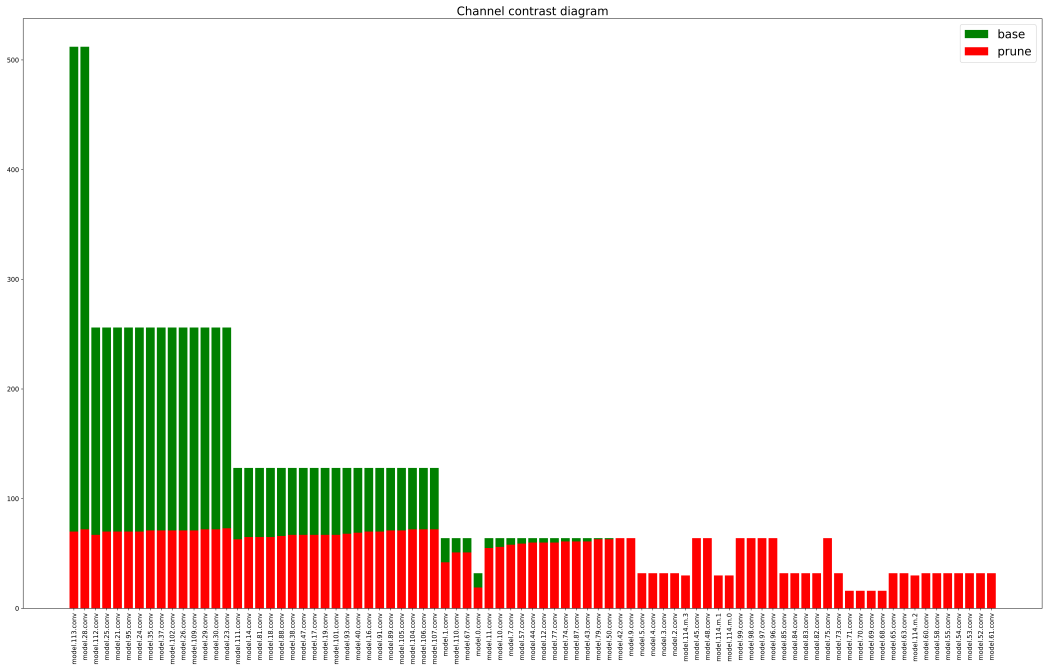


Figure 6. The comparison of the number of channels before and after pruning.

Table 8. Results of the Pruning Algorithm Experiment for the Second Improvement Scheme.

Algorithm Model	Pruning Rate	mAP.5	Params/10^6	FLOPs/10^9	Size/M	FPS
v7-tiny+P2+MSFF	–	72.4%	7.0	22.2	14.6	155
LAMP	50%	72.0%	1.2	10.9	2.9	200
	45%	73.0%	1.4	12.2	3.4	195
	37.5%	74.2%	1.9	14.7	4.5	179
L1	50%	60.6%	5.5	11.0	12.3	172
	45%	63.0%	5.8	12.1	12.8	167
	37.5%	69.0%	6.1	13.8	13.5	160
Group_norm	50%	48.8%	5.6	11.1	12.3	180
	45%	52.4%	5.9	12.3	12.9	169
	37.5%	65.8%	6.3	13.7	13.8	160

4.2.3. Comparative Analysis of Experimental Results After Model Improvement and Pruning

When comparing the pruned models with other single-stage object detection algorithms on the same validation set and the same device with a batch size of 1, the experimental results are shown in Table 9. Analysis of the table reveals that although the pruned V7-tiny-P2-AFPN-EMA model has reduced parameters, size, and computational load compared to the baseline model, and has significantly improved mAP values, its FPS has drastically decreased, falling by more than three times compared to the baseline model. On the other hand, the pruned V7-tiny-P2-MSFF model not only outperforms the baseline model in terms of parameters, size, and computational load, but also the mAP value is 13.5% and 17.6% higher than the advanced YOLOv8-tiny and YOLOv10-tiny [44] algorithms, respectively, YOLOv8 is an algorithm released by the Ultralytics company on 10 January 2023, and the FPS reaches 91, which meets the real-time detection requirements.

Table 9. Performance Comparison of Two Pruned Improvement Schemes with Other Single-Stage Object Detection Algorithms.

Algorithm Model	mAP.5	mAP.5:0.95	Params/10^6	FLOPs/10^9	Size/M	FPS
YOLOv7-tiny	60.7%	36.4%	6.02	13.2	9.7	100
YOLOv5s	62.4%	38.3%	7.02	15.8	14.3	103.1
YOLOX-s	64.2%	40.4%	8.97	13.4	102	32.4
YOLOX-tiny	63.8%	37.9%	5.03	7.57	71.9	35.0
YOLOv3-tiny	49.9%	28.1%	8.68	12.9	16.6	120.5
YOLOv6-tiny	44.7%	28.4%	9.67	12.3	106	41.8
YOLOv8-tiny	59.5%	39.0%	3.0	8.2	6.2	125
YOLOv10-n	55.4%	34.8%	2.6	8.2	5.7	84
P-V7_tiny-P2-MSFF	73.0%	44.7%	1.4	12.2	3.4	91

The detection accuracy of the pruned models under COCO standards for small and medium targets is shown in Table 10. Analysis of the data in the table indicates that the pruned model's small target detection accuracy on the validation set not only did not decline compared to before pruning but also showed slight improvements, increasing by 11.8% compared to the original model, resulting in excellent detection outcomes. On the test set, the small target accuracy remained almost unchanged, improving by 9.1% compared to the original model. The detection accuracy for medium targets also improved by 7.9% and 7.2% on the validation and test sets, respectively. From Tables 8 and 9, it can be concluded that the model achieves a balance between accuracy and speed, meeting the requirements for real-time detection while ensuring high accuracy.

The detection performance of the models before and after improvement under different lighting conditions in the test set is shown in Figures 7–10. As seen in Figure 8, under cloudy conditions, the improved model detects small pedestrian targets and vehicles in occlusion more accurately. Additionally, Figures 7, 9 and 10 demonstrate that the improved model also shows significant improvements in vehicle detection accuracy under low light (night), sunny, and rainy conditions.

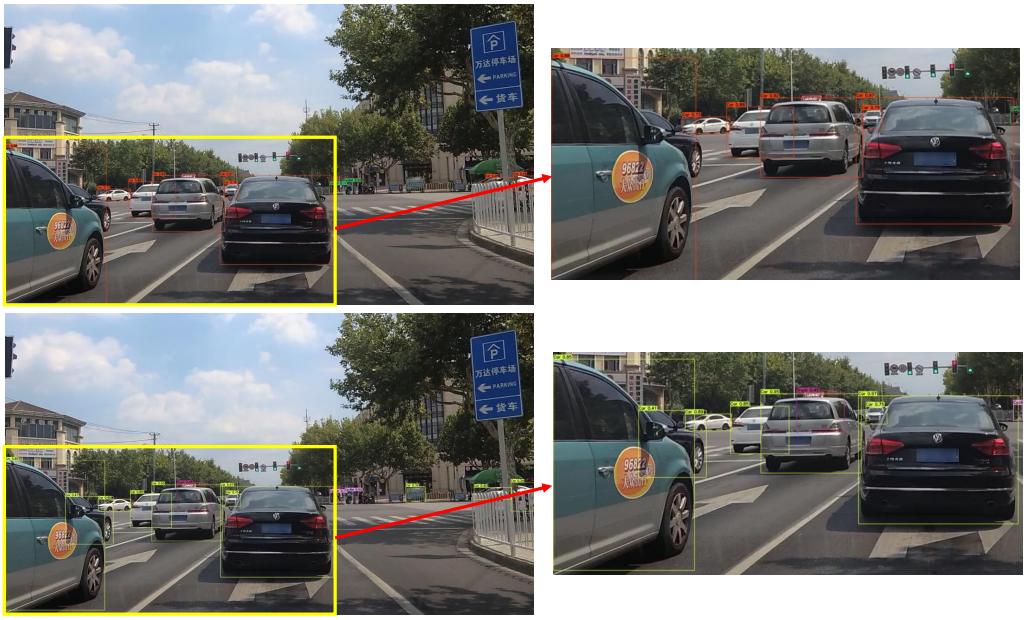


Figure 7. The detection performance of the models before and after improvement.



Figure 8. The detection performance of the models before and after improvement.

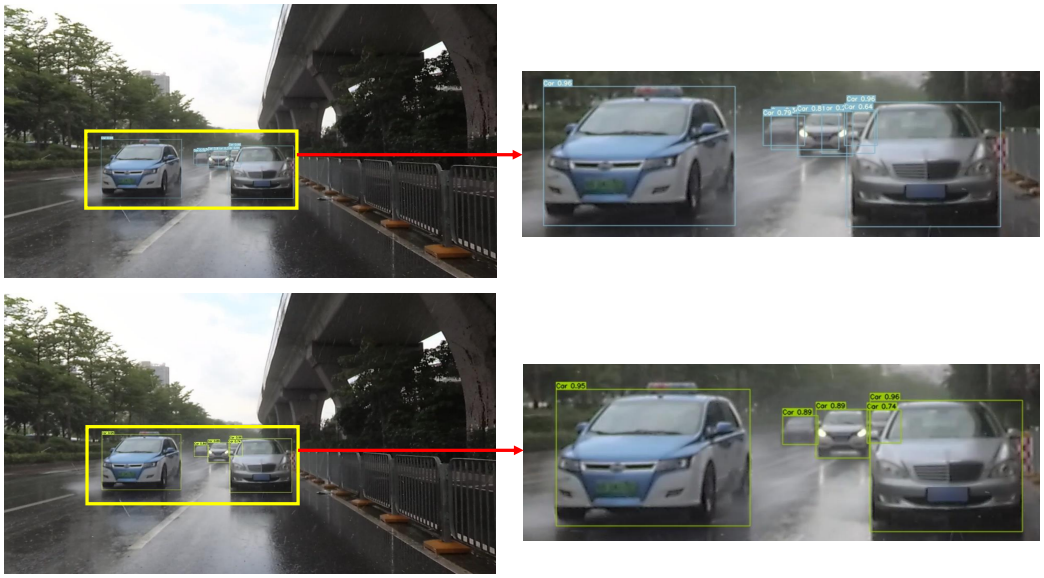


Figure 9. The detection performance of the models before and after improvement.



Figure 10. The detection performance of the models before and after improvement.

Table 10. The accuracy of the model after pruning in detecting small and medium targets.

Algorithm Model	AP.50	AP.50:0.95	AP.50:0.95 (Small)	AP.50:0.95 (Medium)
v7-tiny(val)	65.0%	40.9%	8.8%	33.9%
v7-tiny+P2+MSFF	72.3%	46.0%	19.9%	41.0%
Prune-v7-tiny	72.5%	46.3%	20.6%	41.8%
v7-tiny(test)	64.7%	40.4%	9.8%	33.7%
v7-tiny+P2+MSFF	70.3%	45.2%	19.0%	40.8%
Prune-v7-tiny	71.0%	45.2%	18.9%	40.9%

4.3. Model Edge Deployment

4.3.1. Introduction to Embedded Environment

In this study, the NVIDIA Jetson AGX Xavier is used as the deployment platform. Hardware-wise, the AGX Xavier is more powerful than other models in the Jetson series, such as the TX2 and Xavier NX, making it more suitable for traffic scenarios. The AGX Xavier has significantly more CPU and GPU cores compared to the Xavier NX and TX2, offering a maximum computational power of 11 TFLOPS (FP16) and 32 TFLOPS (FP16). Additionally, the Xavier platform features a deep learning acceleration engine, which allows for faster model inference to achieve real-time detection. The software environment on Xavier is shown in Table 11. The functions used during the C++ inference process are shown as illustrated in Figure 11.

Table 11. Embedded platform version.

Name	Version
Jetpack	4.6
TensorRT	8.0.1.6
CUDA	10.2.300
CuDNN	8.2.1.32
OpenCV	4.1.1

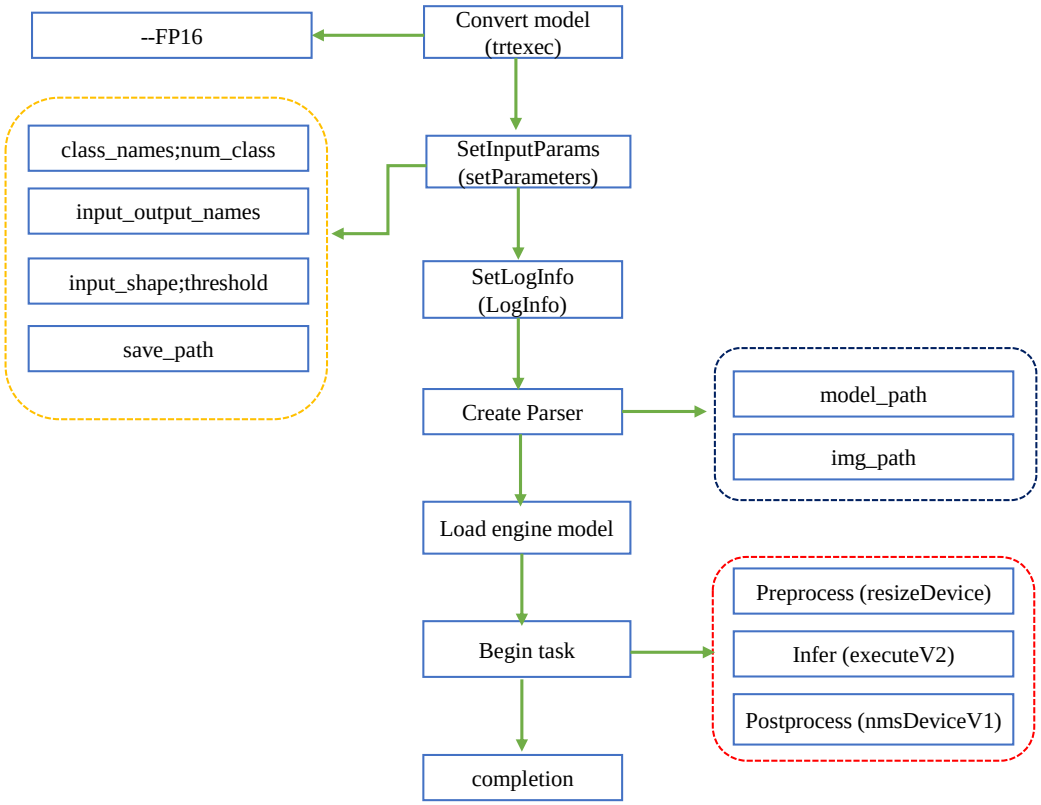


Figure 11. The C++ inference process.

After configuring the software and hardware environment, the best-performing model obtained in the previous section is used for the experiment. Upon completing the model conversion and code compilation, an executable file is generated. Running this executable file allows the image inference task to be carried out. The experimental platform setup is shown in Figure 12.



Figure 12. Experimental platform.

4.3.2. Introduction to Embedded Environment

From the validation and test sets, 10 images each were selected for inference experiments on the Xavier platform. To accurately measure the inference time, each model was given a warmup phase where the same image was inferred 20 times, and the average inference time of the last ten inferences was taken as the model's inference time (Infer-time). The comparative experiments were as follows: (1) Quantizing the improved model to FP16. (2) Using YOLOv5s-7.0 for a horizontal comparison. The experimental results are shown in Table 12.

Table 12. Embedded platform inference speed comparison.

Algorithms	Infer_time (ms)
Noprune_YOLOv7-tiny (FP32)	21.5
Pruned_I-v7-tiny (FP32)	15
YOLOv5s_7.0 (FP32)	16.5
Q-Noprune_v7-tiny (FP16)	10.5
Q-Pruned_I-v7-tiny (FP16)	9.5
Q-YOLOv5s_7.0 (FP16)	7.5

From Table 12, it can be seen that the inference speed of the Pruned-I-v7-tiny model, after pruning but not quantizing, improved by 30.23% compared to before pruning, with an FPS of 66.7. This is slightly faster than the YOLOv5s-7.0, which has an FPS of 60.6. After quantizing to FP16, the inference speeds of all models increased significantly. The pruned and quantized Pruned-I-v7-tiny model achieved an FPS of 95.2, meeting the real-time inference requirements. The inference performance on the Xavier platform of the unpruned but quantized Q-Noprune-v7-tiny, Q-YOLOv5s-7.0, and the pruned and quantized Pruned-I-v7-tiny models are shown in Figures 13 and 14.

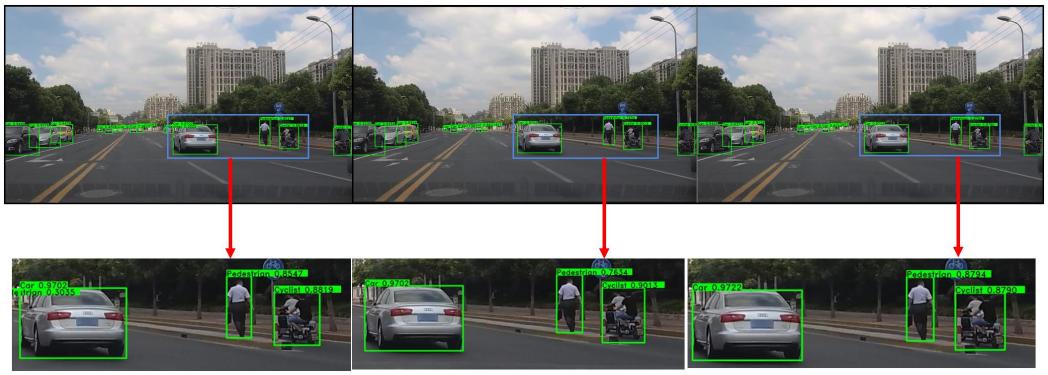


Figure 13. The inference performance on the Xavier platform.



Figure 14. The inference performance on the Xavier platform.

5. Conclusions

This paper proposes a lightweight YOLOv7-tiny network. First, an improvement scheme based on YOLOv7-tiny is introduced: adding a small target detection layer to the original network to form a four-scale detection head and using a redefined K-means clustering standard to cluster anchor boxes suitable for the dataset. The MSFF module is added to the neck to fully integrate shallow, medium, and deep feature information to extract more comprehensive small target information. Second, by combining the Layer-Adaptive Magnitude-based Pruning (LAMP) algorithm and the Torch-Pruning library, different pruning rates are applied to the improved model. Finally, considering both model accuracy and speed, the V7-tiny-P2-MSFF model, pruned by 45% using LAMP, is selected as the final deployment algorithm and deployed on the NVIDIA Jetson AGX Xavier embedded platform. Experimental validation shows that the improved algorithm meets the real-time detection requirements for vehicles and pedestrians in complex traffic scenarios, providing significant reference value for vehicle and pedestrian detection from a dashcam perspective.

While the deployment on the NVIDIA Jetson AGX Xavier demonstrated favorable performance, this platform benefits from relatively strong computational capabilities and dedicated hardware accelerators. This high level of performance may not be replicable on other embedded platforms, particularly those with limited computational resources (e.g., Raspberry Pi or older hardware). Future research directions are as follows: Further efforts should focus on model lightweighting and optimization tailored to different hardware architectures; exploring the combination of LiDAR, infrared sensors, or radar data with existing RGB images could enhance detection accuracy through multimodal data fusion, especially under low-light conditions or adverse weather scenarios.

Author Contributions: Conceptualization, R.M. and Z.L.; methodology, Z.L. and W.W.; software, Z.L. and W.W.; validation, H.Y. and J.W.; formal analysis, Z.L. and H.G.; investigation, W.W.; resources, Z.L.; data curation, W.W.; writing—original draft preparation, Z.L.; writing—review and editing, B.L.; visualization, J.W.; supervision, J.F.; project administration, R.M.; funding acquisition, R.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work is funded by the Project of Science and Technology Programme of Inner Mongolia Autonomous Region (2022YFSJ0040); Project of Basic Research Operating Costs of Colleges and Universities directly under the Inner Mongolia Autonomous Region (JY20220170).

Data Availability Statement: Restrictions apply to the availability of these data. Data were obtained from public datasets SODA10M.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ahmed, S.K.; Mohammed, M.G.; Abdulqadir, S.O.; El-Kader, R.G.A.; El-Shall, N.A.; Chandran, D.; Rehman, M.E.U.; Dhama, K. Road traffic accidental injuries and deaths: A neglected global health issue. *Health Sci. Rep.* **2023**, *6*, e1240. [CrossRef] [PubMed]
2. Hua, J.; Li, L.; Ning, P.; Schwebel, D.C.; He, J.; Rao, Z.; Cheng, P.; Li, R.; Fu, Y.; Li, J.; et al. Road traffic death coding quality in the WHO Mortality Database. *Bull. World Health Organ.* **2023**, *101*, 637. [CrossRef] [PubMed]
3. Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–26 June 2005; Volume 1, pp. 886–893.
4. Zhu, C.; Liang, J.; Zhou, F. Transfer learning-based YOLOv3 model for road dense object detection. *Information* **2023**, *14*, 560. [CrossRef]
5. Wang, N.; Shang, L.; Song, X. A Transformer-Optimized Deep Learning Network for Road Damage Detection and Tracking. *Sensors* **2023**, *23*, 7395. [CrossRef] [PubMed]
6. Nguyen, T.; Park, E.A.; Han, J.; Park, D.C.; Min, S.Y. Object detection using scale invariant feature transform. In *Proceedings of the Genetic and Evolutionary Computing: Proceedings of the Seventh International Conference on Genetic and Evolutionary Computing, ICGEC 2013, Prague, Czech Republic, 25–27 August 2013*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 65–72.
7. Sabry, E.S.; Elagooz, S.; El-Samie, F.E.A.; El-Bahnasawy, N.A.; El-Banby, G. Assessment of various feature extraction methods for object discrimination in different scenarios. *J. Opt.* **2024**, *53*, 49–69. [CrossRef]
8. Devi, N.B.; Kavida, A.C.; Murugan, R. Feature extraction and object detection using fast-convolutional neural network for remote sensing satellite image. *J. Indian Soc. Remote Sens.* **2022**, *50*, 961–973. [CrossRef]
9. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2117–2125.
10. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587.
11. Girshick, R. Fast r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1440–1448.
12. He, K.; Zhang, X.; Ren, S.; Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *37*, 1904–1916. [CrossRef]
13. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*. [CrossRef]
14. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part I 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
15. Howard, A.G. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
16. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 6848–6856.
17. Tan, M. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv* **2019**, arXiv:1905.11946.
18. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
19. Liu, G.; Li, Y.; Song, Y.; Liu, Y.; Xu, X.; Zhao, Z.; Zhang, R. A lightweight convolutional network based on pruning algorithm for YOLO. In Proceedings of the Fourteenth International Conference on Graphics and Image Processing (ICGIP 2022), Nanjing, China, 21–23 October 2022; Volume 12705, pp. 302–311.
20. Zhang, A.; Guo, R.; Liu, K. A model compression method based on YOLOv5s target detection. In Proceedings of the 2023 2nd International Conference on Cloud Computing, Big Data Application and Software Engineering (CBASE), Chengdu, China, 3–5 November 2023; pp. 1–5.

21. Guo, L.; Zhao, Y.; Gao, J. Compression of Vehicle and Pedestrian Detection Network Based on YOLOv3 Model. *IEICE Trans. Inf. Syst.* **2023**, *106*, 735–745. [CrossRef]
22. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
23. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. 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, 27–30 June 2016; pp. 779–788.
24. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
25. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
26. Zhang, Y.; Guo, Z.; Wu, J.; Tian, Y.; Tang, H.; Guo, X. Real-time vehicle detection based on improved yolo v5. *Sustainability* **2022**, *14*, 12274. [CrossRef]
27. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. 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, 17–24 June 2023; pp. 7464–7475.
28. Liang, L.; Ma, H.; Zhao, L.; Xie, X.; Hua, C.; Zhang, M.; Zhang, Y. Vehicle detection algorithms for autonomous driving: A review. *Sensors* **2024**, *24*, 3088. [CrossRef]
29. Berwo, M.A.; Khan, A.; Fang, Y.; Fahim, H.; Javaid, S.; Mahmood, J.; Abideen, Z.U.; MS, S. Deep learning techniques for vehicle detection and classification from images/videos: A survey. *Sensors* **2023**, *23*, 4832. [CrossRef]
30. Wang, Z.; Zhan, J.; Duan, C.; Guan, X.; Lu, P.; Yang, K. A review of vehicle detection techniques for intelligent vehicles. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 3811–3831. [CrossRef]
31. Abiyev, R.; Arslan, M. Vehicle detection systems for intelligent driving using deep convolutional neural networks. *Discov. Artif. Intell.* **2023**, *3*, 16. [CrossRef]
32. Wiseman, Y. Real-time monitoring of traffic congestions. In Proceedings of the 2017 IEEE International Conference on Electro Information Technology (EIT), Lincoln, NE, USA, 14–17 May 2017; pp. 501–505.
33. Zhang, Y.; Sun, Y.; Wang, Z.; Jiang, Y. YOLOv7-RAR for urban vehicle detection. *Sensors* **2023**, *23*, 1801. [CrossRef]
34. Ding, Q.; Li, W.; Xu, C.; Zhang, M.; Sheng, C.; He, M.; Shan, N. GMS-YOLO: An Algorithm for Multi-Scale Object Detection in Complex Environments in Confined Compartments. *Sensors* **2024**, *24*, 5789. [CrossRef] [PubMed]
35. Qian, X.; Wang, X.; Yang, S. A YOLO Algorithm with Lightweight Feature Fusion Network for Multi-Scale Defect Detection. *IEEE Access* **2022**, *10*, 130339–130349. https://doi. org/10.1109/ACCESS. [CrossRef]
36. Wang, S.; Hao, X. YOLO-SK: A lightweight multiscale object detection algorithm. *Heliyon* **2024**, *10*, e24143. [CrossRef] [PubMed]
37. Guo, Y.; Chen, S.; Zhan, R.; Wang, W. LMSD-YOLO: A lightweight YOLO algorithm for multi-scale SAR ship detection. *Remote Sens.* **2022**, *14*, 4801. https://doi. org/10.3390/rs14194801. [CrossRef]
38. Carrasco, D.P.; Rashwan, H.A.; García, M.Á.; Puig, D. T-YOLO: Tiny vehicle detection based on YOLO and multi-scale convolutional neural networks. *IEEE Access* **2021**, *11*, 22430–22440. [CrossRef]
39. Peng, X.; Zhou, X.; Zhu, H.; Ke, Z.; Pan, C. MSFF-Net: Multi-stream feature fusion network for surface electromyography gesture recognition. *PLoS ONE* **2022**, *17*, e0276436. [CrossRef]
40. Lee, J.; Park, S.; Mo, S.; Ahn, S.; Shin, J. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv* **2020**, arXiv:2010.07611.
41. Fang, G.; Ma, X.; Song, M.; Mi, M.B.; Wang, X. Depgraph: Towards any structural pruning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 18–22 June 2023; pp. 16091–16101.
42. Han, J.; Liang, X.; Xu, H.; Chen, K.; Hong, L.; Mao, J.; Ye, C.; Zhang, W.; Li, Z.; Liang, X.; et al. Soda10m: A large-scale 2d self/semi-supervised object detection dataset for autonomous driving. *arXiv* **2021**, arXiv:2106.11118.
43. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning filters for efficient convnets. *arXiv* **2016**, arXiv:1608.08710.
44. Wang, A.; Chen, H.; Liu, L.; Chen, K.; Lin, Z.; Han, J.; Ding, G. Yolov10: Real-time end-to-end object detection. *arXiv* **2024**, arXiv:2405.14458.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.