Middlesex University

CST3990 Undergraduate Individual Project

Title: Vehicle Detection and Tracking with Lightweight Object Detection Model Based on YOLOv11n

Student Name: ChingHsuan Tsai

Student ID: M01001570

Supervisor: Francois Chadebecq

7th April 2025

1. Declaration

I hereby confirm that the work presented here in this report and in all other associated material is wholly my own work.

Signature: ChingHsuan Tsai

Student ID: M01001570

Date: 24th March 2025

2. Acknowledgement

I would like to thank my supervisor Dr. Francois Chadebecq for his guidance and support throughout this project. I also want to mention that during the writing process, I use Google Translate to help me better express my ideas in English.

3. Abstract

Vehicle detection and tracking systems play a key role in modern intelligent transportation systems (ITS). By accurately tracking vehicle types and driving patterns. This technology can provide valuable data to help optimize traffic management strategies and reduce traffic congestion. In recent years, the use of object detection models combined with object tracking models has gradually become the mainstream of this technology. The object detection model can detect objects at high speed and accuracy; the object tracking model can know the movement of objects. Nevertheless, in the intelligent transportation system, if the vehicle is traveling too fast, it will cause miscalculation if the detection speed is not fast enough. Furthermore, if the system needs to be deployed in a resource-constrained environment, the resource usage of the two models combined is also a major issue. In this study, a lightweight object detection model RG-YOLOv11n is proposed. The model can detect objects faster and also takes up less resources due to its lightweight structure. This article introduces the object detection algorithm RG-YOLOv11n and proposes an experiment to evaluate multiple mainstream object detection algorithms with the object tracking model Deep SORT to find a model that can perform object detection with high speed and low resource usage under a certain accuracy.

Key words: Vehicle detection; Vehicle tracking; YOLOv11; Deep SORT; Lightweight

Content

4. Introduction

4.1 Problem Definition

With the increasing popularity of vehicles, problems such as traffic congestion and accidents have arisen. To solve these problems, it is necessary to track vehicles on the road in real time to determine traffic flow and adjust the transportation system in a timely manner (Bui et al., 2024). By accurately tracking vehicle types, technology is provided with valuable data to help optimize traffic management strategies, reduce traffic congestion, and improve overall transportation efficiency. In recent years, many people have begun to use object detection models to make vehicle detectors.

However, when dealing with fast-moving vehicles, the object detection model will have difficulty processing a large amount of calculation in a short time. The accuracy of the results will be reduced. In addition, internet connection in developing regions is unstable. This situation makes the model difficult to transmit continuously (Chauhan et al., 2019). In this environment, Deployment on edge devices is the best choice. The high computational burden also limits the processing speed of this case. Deployment of resource-constrained edge devices such as embedded systems or loT (Internet of things)-based traffic monitoring equipment is a challenge. These challenges are particularly acute in large-scale smart city deployments and in remote areas with limited computing resources and network connectivity. To ensure that the systems can track fast-moving vehicles in real time and accuracy. Improving the computational efficiency of the object detection model or reducing its computational complexity becomes an important issue.

4.2 Objective

The study developed a vehicle detection for the lightweight model. By reducing the

model parameters and computational complexity. The model can run efficiently on devices with limited resources and maintain high detection accuracy and real-time performance in high-speed operating environments.

Specifically, the study proposed a lightweight model based on YOLOv11n combined with RepGhostBottleneck. It achieves lightweight by reducing the computational complexity of the model. Therefore, this model can not only operate efficiently on resources-constrained devices but also achieve real-time vehicle detection in high-speed driving environment.

4.3 Article Structure

This article will be divided into five parts. First, the research objectives and problem definition will be introduced. The literature review will explore the literature related to this study, including the status of vehicle detection models, the introduction and comparison of various object tracking algorithms, the technical details of object detection algorithms, and lightweight models and way. The third part is the implementation details, describing the data set selection and pre-processing process, the experimental methods and experimental settings, the model training process and evaluation. Finally, the overall research will be summarized, and subsequent research directions will be proposed.

4.4 Contribution

The contribution of this study is to propose a lightweight vehicle detection model that can effectively improve the real-time performance and accuracy of the traffic management system. By combining YOLOv11n and RepGhostBottleneck, the model significantly improves the computing speed while maintaining high accuracy and is suitable for resource-constrained devices. This study also uses a variety of YOLO-

related models on the vehicle detection system to check the response speed of the vehicle detection system. The experiment will evaluate the performance and find the model that can achieve the fastest vehicle detection from the evaluation.

5. Literature Review

5.1 Background

Vehicle detection and tracking technology plays a vital role in modern traffic management (Kamkar and Safabakhsh, 2015). In the past, people often used background subtraction and optical flow methods. Although background subtraction is easy to implement and has low computational overhead, it is susceptible to sudden changes in illumination; while optical flow is accurate but computationally expensive (Deshmukh and Uke, 2015). Deep learning-based object detection models are increasingly replacing traditional methods. Because it demonstrates higher accuracy, speed and reliability in real-time target detection. This makes deep learning the mainstream choice for current vehicle detectors (Kumar et al., 2023).

5.2 Multiple Object Tracking

This chapter will introduce three core algorithms for multi-object tracking (MOT): SORT, DeepSORT and StrongSORT. Their principles, characteristics and application scenarios are compared and analyzed, and the advantages and limitations of these methods in vehicle detection and tracking are explained.

5.2.1 SORT

SORT (Simple Online and Realtime Tracking) is a detection-based multi-object tracking (MOT) algorithm proposed by (Bewley et al., 2016). It mainly uses a Kalman filter to predict the motion state of objects and employs the Hungarian algorithm to

perform data association based on the Intersection-over-Union (IoU) of bounding boxes. SORT is designed with an emphasis on speed. It has a tracking update rate of up to 260Hz. Suitable for application scenarios that require immediate processing. However, SORT does not use appearance features other than bounding box position and size for tracking, which makes its performance highly dependent on the quality of the detector. If the accuracy of detector is not high, tracking effect of SORT will be significantly affected. In addition, SORT has poor handling of occlusion issues and is prone to ID switching issues, resulting in unstable tracking. in other words, SORT is a simple and efficient tracking method, suitable for scenarios with high real-time requirements, but its performance is limited when dealing with complex scenarios.

5.2.2 DeepSORT

DeepSORT is an extended version of SORT that improves tracking performance by integrating appearance information (Wojke et al., 2017). It uses a convolutional neural network (CNN) pre-trained on a large-scale person re-identification dataset to extract the appearance features of the target. For data association, it combines motion information (using Mahalanobis distance) and appearance information (using cosine distance) to achieve more accurate object matching. DeepSORT introduces the Matching Cascade strategy to further improve tracking stability. It prioritizes targets that were most recently observed. This effectively reduces the number of ID switching times. Especially when the target is occluded for a long time.

5.2.3 StrongSORT

StrongSORT is based on DeepSORT and has made many improvements (Du et al., 2022). StrongSORT uses more powerful object detectors (such as YOLOX) and more discriminative appearance feature extraction models (such as BoT) to improve tracking

accuracy. Apart from this, StrongSORT adopts a variety of latest inference techniques, such as EMA feature update, ECC camera motion compensation and NSA Kalman filter. This further optimizes tracking performance. In the association stage, StrongSORT uses both appearance and motion costs for object matching and abandons the matching cascade strategy of DeepSORT in favor of a normal global linear assignment to simplify the matching process. To solve the problem of missing associations, StrongSORT proposed AFLink (Appearance-Free Link Model) and used GSI (Gaussian Smoothing Interpolation) to handle the case of missing detection.

5.2.4 Evaluation

The above three algorithms each have their pros and cons. Although SORT is fast, its accuracy is low. It is suitable for scenarios with extremely high real-time requirements but no requirements for accuracy. StrongSORT has greatly improved its accuracy after multiple improvements, but its computational cost is relatively high, and its running speed is very slow. It is not suitable for real-time detection tasks. This is why DeepSORT is still widely used in vehicle detection after many improvements. DeepSORT achieves a good balance between accuracy and real-time performance. It can not only effectively handle the problems of target occlusion and rapid movement but also meet the immediacy requirements while ensuring high accuracy. Based on the above evaluation, users can make choices according to their needs.

5.3 Object Detection

This chapter will explore in depth the evolution and technical architecture of the YOLO series of object detection algorithms. Since this project is a lightweight design based on the YOLO architecture, this chapter will focus on three important versions developed by the Ultralytics team: YOLOv5, YOLOv8, and YOLOv11.

5.3.1 YOLO

YOLO (You Only Look Once) is a widely used and very popular object detection algorithm (Jiang et al., 2022). As its name suggests, YOLO can directly input images and predict the bounding boxes and categories of all objects in the image at once through a single neural network. The reason for its continued popularity is that its modular design allows for easy customization. It can export trained models into various formats such as ONNX, CoreML and TFLite and facilitate deployment on different platforms.

One-stage detection means treating object detection as a regression problem and directly predicting the location and category probability of the bounding box. The previous two-stage detection approach first generates candidate regions and then classifies them. One-stage greatly simplifies the process and increases speed.

YOLO provides various sizes

YOLO provides variants of different sizes, namely n, s, m, l, and x, to adapt to different computational requirements.

- N (Nano) is the smallest model, designed for resource-constrained environments.
- S (Small) is suitable for scenarios that have certain requirements on speed and resource consumption.
- M (Medium) provides a moderate level of accuracy and computational requirements.
- L (Large) is designed for tasks that require high accuracy.
- X (Extra Large) is the largest variant providing the highest accuracy, but its computational cost is relatively high.

5.3.2 YOLOv5

YOLOv5 integrates several innovations including CSPNet backbone network (Cross stage partial networks), spatial pyramid pooling (SPP) model, PAN (Path Aggregation network) neck network, convolutional head, prediction based on anchor boxes and loss function (Hussain, 2024). These analyses help readers understand YOLOv5 and provide a theoretical basis for the subsequent improvements.

- CSPNet Backbone Network

The CSPNet backbone network is an improvement of ResNet (a classic deep learning model commonly used in image processing). Improve efficiency and reduce computational workload through CSP connection.

- PAN Neck Network

PAN Neck network can combine features from different levels and utilize upsampling (a technique for enlarging feature maps) to increase the resolution of feature maps.

- Convolutional Head

The head of YOLOv5 consists of multiple convolutional layers, which are responsible for predicting the bounding boxes and category labels of objects.

- Prediction Based on Anchor Boxes

YOLOv5 uses predefined anchor boxes with different configurations and sizes to link each bounding box to a set of predefined aiming boxes.

- Loss Function

The loss function of YOLOv5 consists of binary cross entropy loss and CIoU loss (Complete Intersection over Union Loss). Binary cross entropy loss helps the model determine whether the predicted class is correct and how confident the model is in the

prediction. CIoU loss is used to measure the overlap between the predicted bounding box and the actual bounding box to ensure the accuracy of positioning.

5.3.3 YOLOv8

YOLOv8 is an improvement of YOLOv5, including CSPDarknet backbone network, PAN neck network (improved version), anchor-free detection head, hyperparameter optimization and C2f building block (Hussain, 2024).

- CSPDarknet Backbone Network

YOLOv8 uses an enhanced version of the CSPDarknet backbone network, which is an improved Darknet architecture combined with CSP technology. The key to CSPDarknet is that it splits the feature map of each stage into two parts: one part is processed by a dense convolution block (a basic unit for processing images), and the other part is directly connected to the output of the dense block. Dense blocks mean that each convolutional layer is directly connected to all previous layers. Different from the ResNet structure, each layer is only connected to the previous and next layers. The design of CSPDarknet reduces the computational complexity and maintains high accuracy.

- PAN Neck Network (improved version)

YOLOv8 uses an improved version of Path Aggregation Network (PAN) in the PAN neck network. This approach allows the model to utilize both low-level and high-level features, thereby improving the model's ability to detect objects at different scales. Especially the detection effect of small objects is better. In addition, YOLOv8 improved post-processing after prediction, especially the Non-Maximum Suppression (NMS) algorithm. This algorithm solves the problem of generating multiple overlapping

bounding boxes for the same object. The improved NMS can more effectively select the most appropriate bounding box and remove other bounding boxes to avoid duplicate detection.

- Anchor-Free Detection Head

Previous YOLO models relied on pre-defined anchor boxes to predict the location of objects. However, YOLOv8's anchor-free method directly predicts bounding boxes. This design simplifies the model architecture, reduces the amount of computation, and makes reasoning faster.

- Hyperparameter Optimization

YOLOv8 includes automatic hyperparameter optimization, which allows the model to adjust its own hyperparameters to achieve optimal performance.

- C2f Building Blocks

YOLOv8 introduces the C2f (Cross Stage Partial with two fusion) building block, which is an improvement in feature extraction and fusion technology. The C2f block allows the model to better capture subtle details and complex patterns in images by more efficiently combining features from different levels. This design enhances the feature extraction capability and enables the model to perform better when dealing with complex scenes.

5.3.4 YOLOv11

YOLOv11 extends and enhances the foundation of YOLOv8 (Khanam et al., 2024). The main improvement of C2f block is called C3k2. The improved C2f block of YOLOv8 is called C3k2. In addition, there is a new C2PSA block. This section

introduces the three basic components: the backbone, the neck, and the head.

- The Backbone

The backbone network structure of YOLOv11 is similar to that of YOLOv8. The CBS block is also used to downsample the image and the C2f block is replaced by the C3k2 block.

The CBS (Convolution-BatchNorm-SiLU) block combines convolution, batch normalization (BatchNorm), and SiLU activation functions. The principle of batch normalization is to standardize the input data of each layer to make network training more stable. The SiLU activation function is an improved activation function. It combines the properties of Sigmoid and ReLU. This enables it to better capture complex nonlinear relationships and enhance feature expression capabilities. This combination makes the CBS block an efficient and powerful basic unit in YOLOv11, helping the model to better process and optimize feature maps.

Following is the mathematical formula of SiLU activation function where x is the input.

$$\sigma(x) = x \times \frac{1}{1 + e^{x}}$$

The C3k2 block is a more computationally efficient implementation of the Cross Stage Partial (CSP) Bottleneck. Its characteristic is that two smaller convolution kernels are used to replace a large convolution kernel. Moreover, the C3k2 block can be adjusted according to needs: when c3k = False, its structure is similar to the traditional C2f block; when c3k = True, it uses a deeper C3 module to enhance feature learning capabilities. This design not only reduces the amount of calculation but also improves the efficiency of feature extraction. This allows the model to better capture details and complex patterns in images while maintaining efficient computation.

- The Neck

The neck network of YOLOv11 includes the SPPF (Spatial Pyramid Pooling - Fast) module. It is a fast and efficient multi-scale features extraction technology that can better handle objects of different sizes. YOLOv11 also introduced the C2PSA (Convolutional block with Parallel Spatial Attention) module. This is a new attention mechanism that allows the model to focus more on key areas in the image. Especially when detecting small or partially occluded objects.

- The Head

The head network of YOLOv11 is responsible for generating the final object detection and classification prediction. The details are not mentioned here because its architecture is similar to YOLOv8.

5.3.5 Conclusion

This chapter reviews the evolution of the YOLO object detection algorithm from YOLOv5 to the latest YOLOv11. Besides that, the key improvements in each version are discussed. YOLOv5 introduced CSPNet and PAN neck network, laying the foundation for the modern YOLO architecture; YOLOv8 further optimized Backbone, Neck and anchor-free detection head to make the model more flexible and efficient; YOLOv11 built on this foundation and improved feature extraction and attention mechanism through C3k2 and C2PSA modules to make the detection effect more accurate.

5.4 Lightweight Models

This section will mention the lightweight model RepGhost used in this project and its previous versions. The core design of GhostNet and RepGhost will be discussed in

detail including the module architecture and the technologies used. Finally, the lightweight models mentioned in this chapter will be summarized and evaluated.

5.4.1 GhostNet

GhostNet is a lightweight convolutional neural network (CNN) designed to achieve efficient feature extraction at a lower computational cost (Han et al., 2020). Its core innovation is the introduction of the Ghost module. It can significantly reduce the number of parameters and computational complexity without sacrificing the quality of feature maps. There is redundancy between the feature maps output by many convolutional layers. Some feature maps are similar to each other. Therefore, there is no need to use a large number of convolution operations to generate these redundant feature maps one by one. As shown in Figure 1, this is a visualization of some feature maps, and similar maps are annotated with the same color. One of each pair of feature maps can be transformed into the other using a cheap operation. Cheap operations are represented by spanners.

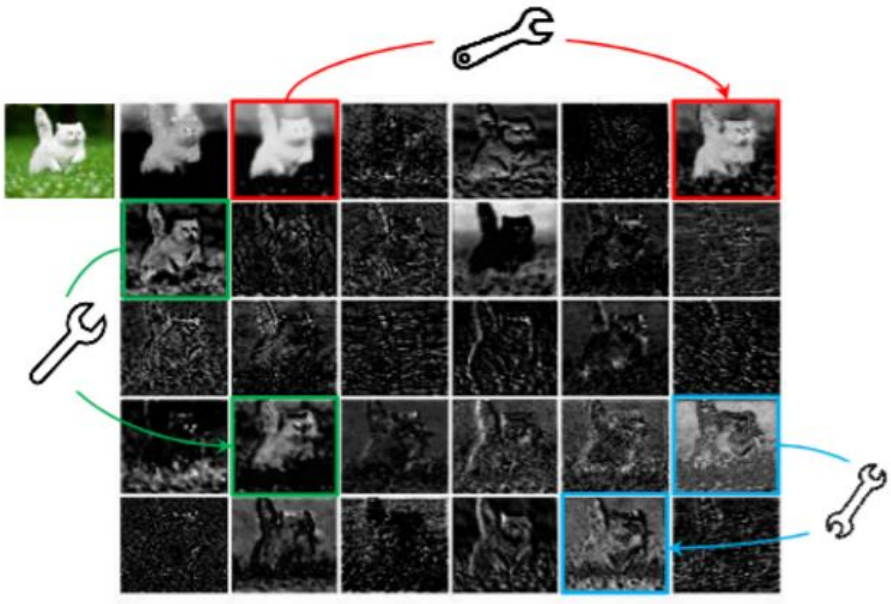

Figure 1. Visualization of the feature maps generated by the residual blocks in

ResNet-50 (Han et al., 2020).

- The Principle of Ghost Module

In the convolutional layer in Figure 2(a), the input will directly output a new set of feature maps after convolution. Figure 2(b) Ghost module splits the process into two steps. Generate some essential feature maps through fewer standard convolutional layers. After that, these feature maps go through a series of cheap linear operations (Φ_1, Φ_2, …, Φ_k). The feature map generated by linear operation combined with the essential feature map is called a "ghost" feature map.

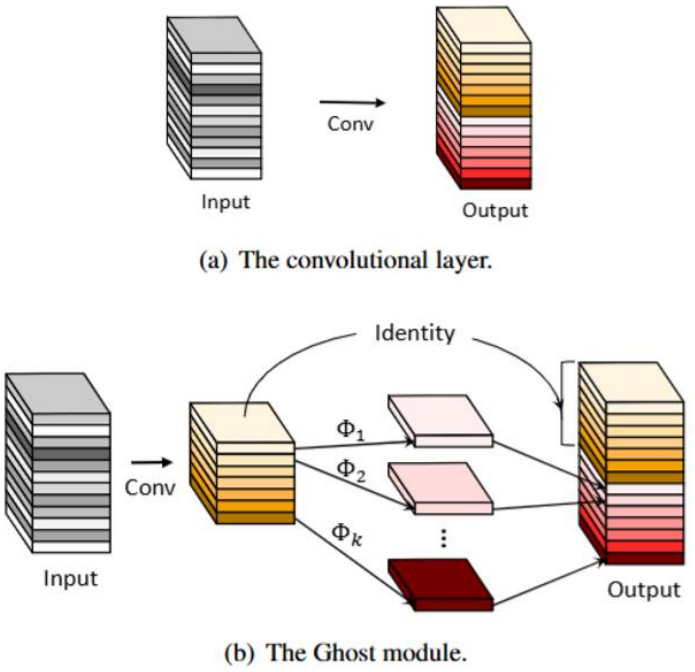

Figure 2. Operation of convolutional layer and Ghost module (Han et al., 2020).

The purpose of these ghost feature maps is to expand the diversity of features and fully reveal the information hidden in the essential features without significantly increasing the computational cost and the number of parameters. This design first uses ordinary convolution to generate representative essential features and then uses linear transformation to "copy" many "ghost" feature maps.

5.4.2 Improved Design of RepGhost

RepGhost is an improved Ghost module that aims to address the problem of high computational cost of concatenation on hardware devices in the original GhostNet (Chen et al., 2024). The evolution of RepGhost includes:

Replacing the concatenation operation with an addition operation is used to reduce the computational cost of the concatenation, as shown in Figure 3(b).

The ReLU activation function is shifted backward to satisfy the reparameterization rule (described in detail in the next paragraph). As shown in Figure 3(c).

At the same time, batch normalization (BN) is added during training so that multiple branches can be fused into one convolutional layer during inference, as shown in Figure 3(d).

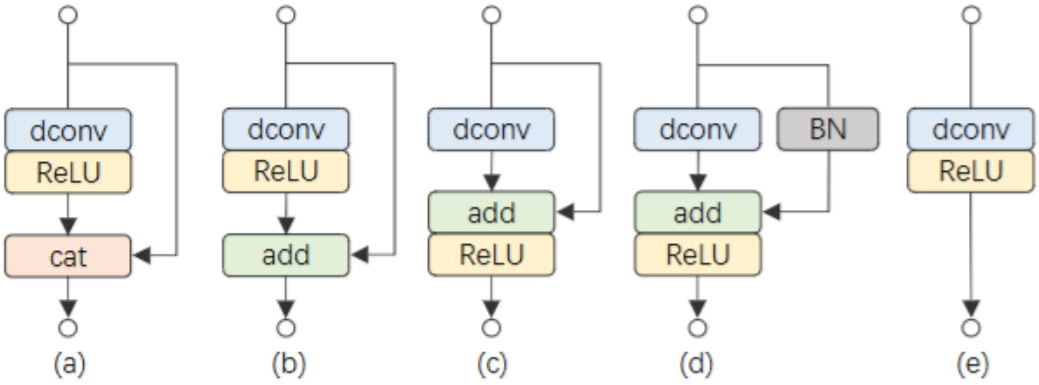

Figure 3. Ghost convolution to RepGhost module(Chen et al., 2024).

- Figure 3(a) shows the basic structure of the original Ghost module. It first uses a 1x1 convolutional layer to produce a small number of "essential feature maps". After that, it applies a series of cheap linear transformations to these essential feature maps to generate more "ghost feature maps". Finally, the essential feature maps and ghost feature maps are combined into the final output through

concatenation. The figure also shows that the ReLU activation function is usually connected after the deep convolution.

- Figure 3(b) depicts the first step of the evolution, replacing the concatenation in the Ghost module with element-wise addition. Although concatenation is free in terms of FLOPs and parameters, it is computationally expensive in hardware. Using addition can reduce this cost.
- Figure 3(c) moves the ReLU activation function after the addition. The purpose of this is to ensure that different branches of linear operations can be more easily fused at inference time.
- Figure 3(d) depicts the structure of the RepGhost module during the training phase. It contains an addition. The addition is followed by a depthwise convolution and a ReLU. Moreover, this architecture introduces a parallel identity mapping branch and adds batch normalization (BN) to this branch. The BN layer introduces nonlinearity during training and can be fused with the previous convolutional layer during inference. This is an important step in reparameterization.
- Figure 3(e) demonstrates the structure of the RepGhost module during the inference phase. Through the reparameterization process, multiple branches during training (including the main convolutional path and the identity mapping branch with BN) are fused into a single ordinary convolutional layer followed by a ReLU activation function. This simplified structure is why RepGhost is able to achieve hardware-efficient inference.

These designs make RepGhost more efficient in hardware, because concatenation takes about twice as long as addition. It implicitly achieves feature reuse through reparameterization instead of using explicit concatenation operations such as GhostNet.

5.4.3 Reparameterization

Reparameterization is a technique used to improve hardware efficiency in neural networks, first proposed by methods such as RepVGG (Chen et al., 2024). During training, the network may contain multiple parallel branches of linear operations (such as convolutional layers and batch normalization layers). These branches can be merged into a single convolutional layer at inference time through mathematically equivalent transformations. According to the authors of RepGhost, the focus of reparameterization is on the fusion properties of linear operations. For instance, the weights and biases of a convolutional layer can absorb the parameters from a parallel batch normalization layer to form a new, equivalent convolutional layer. Nevertheless, non-linear activation functions (such as ReLU) destroy the fusibility of linear operations. Thus, it is necessary to move ReLU to after all linear operations are merged to satisfy the reparameterization rules.

5.4.4 RepGhost Bottleneck Compared to Ghost Bottleneck

Figure 4. compares the structure of the RepGhost Bottleneck and the original Ghost Bottleneck. The original Ghost Bottleneck consists of two stacked Ghost modules, which may contain batch normalization and ReLU activation functions in the middle and are connected by shortcuts to form a residual block structure.

The first Ghost module acts as an expansion layer to increase the number of channels; the second Ghost module reduces the number of channels to match the shortcut path. The RepGhost Bottleneck replaces the two Ghost modules in the original Ghost Bottleneck with the RepGhost module and introduces a narrower middle channel and wider output channel. At the same time, each RepGhost module contains a parallel BN branch. These branches help the network learn richer features during training. The

RepGhost bottleneck may also include SE Layer (Squeeze-and-Excitation Layer) to further improve model performance.

During the inference phase, RepGhost Bottleneck simplifies the complex structure during training into a more efficient single convolutional layer through reparameterization technology. The parallel BN branches are fused into the weights of the convolutional layer to form an equivalent convolution operation. This design makes RepGhost Bottleneck more concise and computationally efficient during inference. Besides, it inherits the hardware efficiency advantages of the RepGhost module. Specifically, it has higher inference speed, lower hardware requirements and maintains high performance.

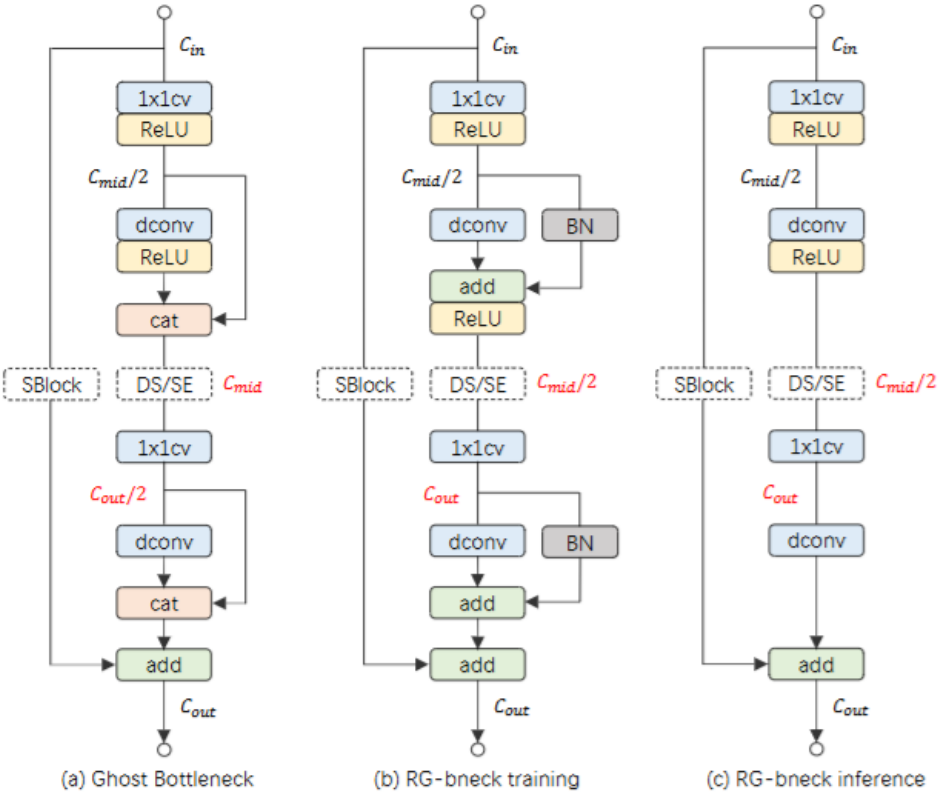

Figure 4. RepGhost Bottleneck compared to Ghost Bottleneck (Chen et al., 2024).

5.4.5 Conclusion

GhostNet and RepGhost represent two important stages in the design of lightweight convolutional neural networks. GhostNet achieves efficient feature extraction and reduced computational costs through the new design of the Ghost module. On the other hand, RepGhost further improves hardware efficiency and inference speed through reparameterization technology. Particularly suitable for resource-constrained environments. The success of RepGhost lies not only in its efficient reasoning structure, but also in its implicit implementation of feature reuse. This allows it to significantly reduce computing costs while maintaining high performance.

5.5 Lightweight Way

The lightweight approach of this study is inspired by the paper (Zhu et al., 2024). This paper replaces the bottleneck of YOLOv8 with RepGhost-Bottleneck-EMA (which is the improved model in the paper). RepGhost Bottleneck is used in the model to replace Bottleneck in the C2f module of YOLOv8. The goal was to achieve lightness without sacrificing precision. And the integration of the focus mechanism EMA (Exponential Moving Average) can effectively capture the environmental context.

5.6 Conclusion

The literature review mentioned the history of vehicle detection, the development of object detection and tracking technology including SORT, DeepSORT and StrongSORT, a detailed analysis of the evolution of the YOLO series algorithms from v5 to v11, the lightweight models GhostNet and RepGhost, and the lightweight way. These techniques provide important knowledge to help understand the current state-of-the-art concepts and give direction to the selection and implementation of the research model.

6. Implement

6.1 Overview

This chapter will introduce the implementation process of this study, including dataset selection, preprocessing, model setting and training process, and final evaluation. The dataset section will explain the reasons for selecting this dataset and the preprocessing. Next, the adjustment methods and model architecture before training are explained in detail. The experimental setup section will describe the hardware and software environment used in the training process, as well as the selected development framework and tools. During the training process, this study systematically tested and evaluated some of the current mainstream object detection algorithms. The test objects include five YOLO variants with different architectures, including RG-YOLOv11n, RG-YOLOv8n, standard version of YOLOv11n, YOLOv8n and YOLOv5n. These models will be trained on the same dataset for performance comparison. After completing the basic evaluation, this study examined the performance of these five different object detection algorithms combined with Deep SORT. Based on the above experimental results, this study finally chose to combine RG-YOLOv11n with Deep SORT to realize the vehicle detection and tracking system.

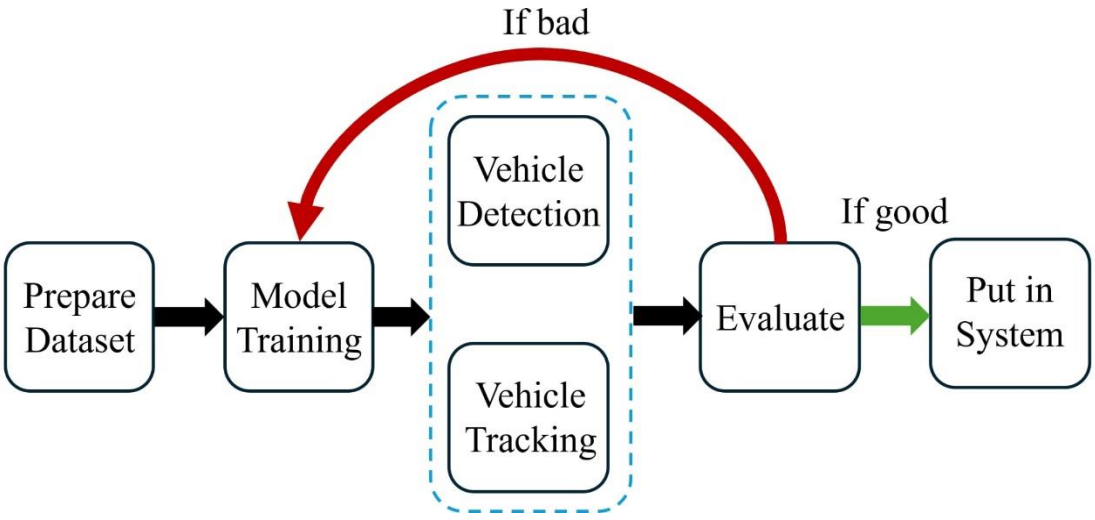

Figure 5. Implementation Flowchart

6.2 Dataset Selection and Preprocessing

In training the object detection model, the experiment used the "Vehicle Detection Image Dataset" from Kaggle (https://www.kaggle.com/datasets/pkdarabi/vehicle-detection-image-dataset/data), and the size of each initial image was 640×640. This study originally planned to use the coco dataset because of its rich and complete content. However, Coco is not specifically designed for vehicle tracking scenarios. This dataset covers a variety of objects in daily life and is not particularly focused on transportation or vehicles. It cannot provide the special annotations and scenarios required for vehicle detection. As a result, this study selected this dataset to simulate the scenarios mentioned in the literature review as much as possible. There are 144 images in total, and the proportions of training data, validation data, and test data are 70%, 20%, and 10%, respectively.

6.3 Methodology and Model Design

This study selected the YOLOv11 framework as the model. Because of the frequent use of Bottleneck in this framework, this study can be lightweight for this module. The reason for choosing YOLOv11n is that version n is the lightest in the YOLOv11 series. Hence, this study believes that this model is most suitable for the situation of this study. The lightweight method RepGhost is similar to that mentioned in the literature review. The change of the thesis to Bottleneck in YOLOv8 inspired the ideas of this study for changes to Bottleneck in YOLOv11.In Figure 6, this study draws a diagram of the improved model architecture.

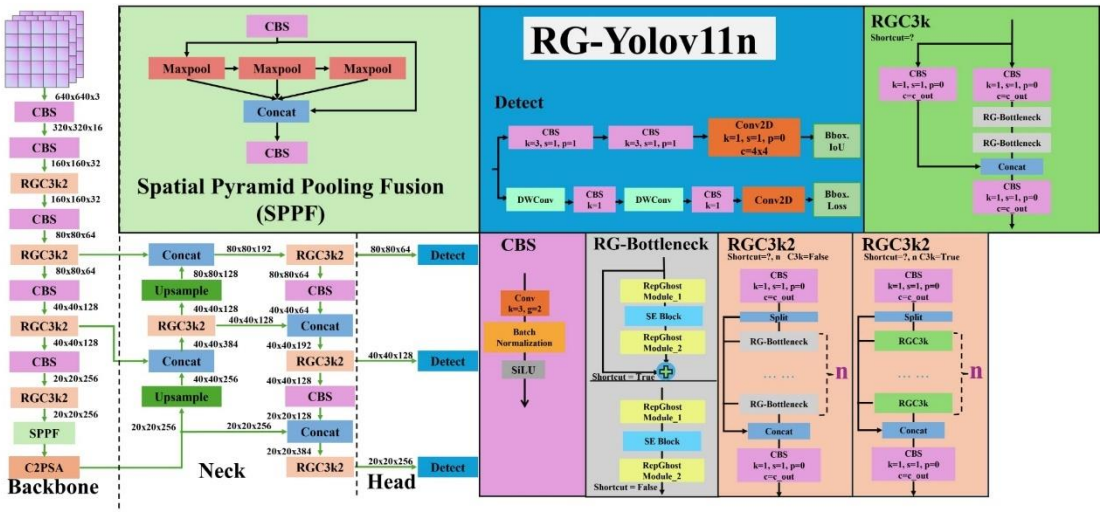

Figure 6. The diagram of RG-YOLOv11n architecture.

Since this study also introduced RG-YOLOv8n for testing, its architecture diagram was also drawn in Figure 7.

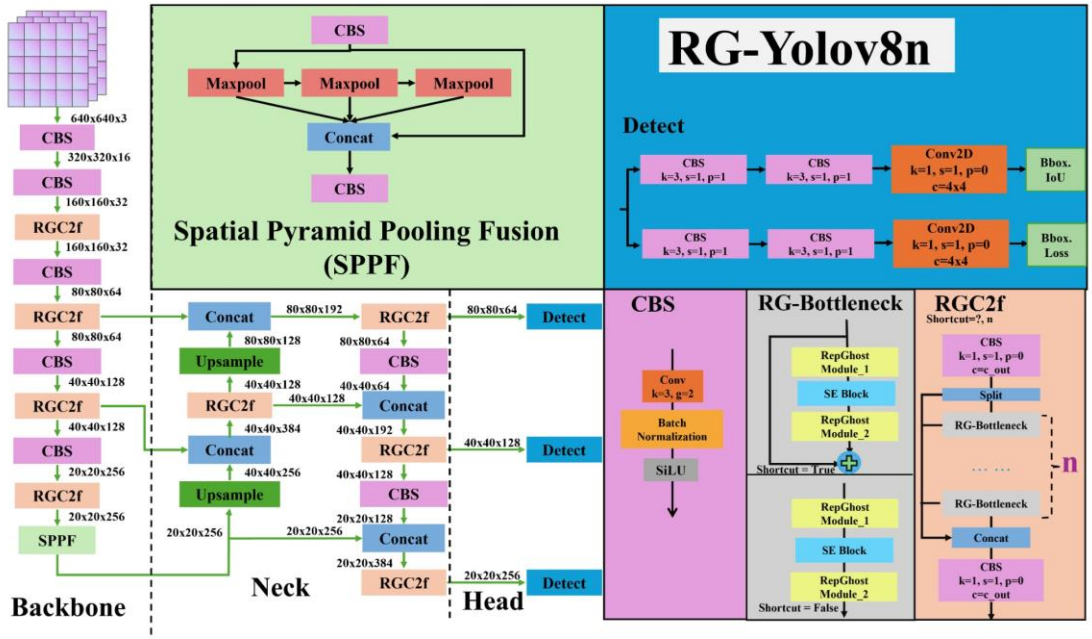

Figure 7. The diagram of RG-YOLOv8n architecture.

6.4 Experimental Setup and Environment Configuration

This study will first use GPU workstations to train the object detection model and combine it with Deep SORT to implement a vehicle detection system. Finally, this study

evaluates the performance of these systems. Table 1 shows the hardware configuration used in this study.

Table 1. Hardware Specification

Resource	Workstation
GPU	NVIDIA GeForce GTX 4070 Ti
CPU	Intel(R) Core (TM) i7-13700H CPU @ 3.60GHz
Memory	64GB
Storage	CT1000MX500SSD1 1TB

This experiment mainly uses the Ultralytics suite and the Deep_sort_realtime suite to implement the vehicle detection and tracking system. Since both above two main packages require the use of Pytorch, it is used as the deep learning framework in this study. Table 2 shows the software packages used in this study.

Table 2. Recipe for Packages

Software	Version
Python	3.9.21
Pytorch	1.9.1
Anaconda	23.7.2
Ultralytics	8.3.76
Deep_sort_realtime	1.3.2

6.5 Model Training

During training, the epoch was set to 300 and the batch size was set to 32 to ensure that each model was compared on the same basis. Finally, the input image size is set to 640*640 because the image size of the dataset is 640*640. After training, each object detection model is combined with Deep SORT. Finally, record the performance of the above combinations and find the most suitable method to solve the problem among these combinations.

6.6 Evaluation and Results

In Table 3, the first row shows the training time of each model, and the second row shows the model inference time. To summarize the results of this training, RG-YOLOv8 has the shortest inference time, and YOLOv5 has the longest inference time.

Table 3. Training and inference time of object detection models.

Model	YOLOv5n	YOLOv8n	RG-YOLOv8n	YOLOv11n	RG-YOLOv11n
Training (hr)	0.89	1.1	1.5	0.96	1.3
Inference (ms)	31.64	27.88	20.57	24.16	22.21

Table 4 presents the performance comparison of various object detection models. The parameters (2,276,735) of the model proposed in this study are the lowest among all models, and the FLOPs (6.1G) are also the lowest among all models. Although the model sacrificed some precision and mAP@50, the lightweight purpose of this study was quite successful.

Table 4. Performance comparison of different object detection models.

Model	Param	FLOPs	Precision/%	Recall/%	mAP@50/%
YOLOv5n	2,503,919	7.1G	71.3	68.9	74.6
YOLOv8n	3,006,623	8.1G	83.9	73.1	75.7
RG-YOLOv8n	2,230,335	6.1G	75.8	67.2	69.3
YOLOv11n	2,538,127	6.3G	86.0	73.1	78.8
RG-YOLOv11n(our)	2,276,735	6.2G	83.7	68.9	74.5

The FPS comparison of each object detection model combined with Deep SORT is presented in Table 5. The model proposed in this study slightly improves the FPS of the entire system (37.26), allowing the entire system to detect vehicles at a faster speed.

Table 5. FPS of object detection models with deep SORT.

Model	YOLOv5n	YOLOv8n	RG-YOLOv8n	YOLOv11n	RG-YOLOv11n(our)
FPS	24.26	30.86	40.13	34.15	37.26

This study uses the lightweight structure RepGhost Bottleneck to replace the Bottleneck in YOLOv11n to achieve the purpose of lightweight. The model in this study has significantly lower parameters than other models and has good performance in FPS

without losing much accuracy.

7. Conclusion

7.1 Limitations

The object detection algorithm proposed in this study can effectively improve the model detection speed and reduce resource usage. However, in order to increase FPS and reduce resource usage, a lot of accuracy has been sacrificed. During testing, we also found that the object tracking model is easily affected by occlusion. Any obstruction in the image will cause distortion of subsequent detection and tracking data.

7.2 Future work

Based on the above two limitations, I believe that attention mechanisms other than SE Layer can be introduced into the structure of the object detection model to improve prediction accuracy. I also hope to find an object tracking algorithm that is not affected by occlusion. Strong SORT is a good choice, but its resource usage and FPS do not meet the purpose of this study. Therefore, this study needs to find a more powerful object tracking model.

7.3 Summary

In general, this study successfully used the object detection model combined with Deep SORT to implement a vehicle detection and tracking system. In addition, to meet the requirements of high FPS and low resource usage, this study also proposed a lightweight object detection algorithm RG-YOLOv11n. Although the accuracy is reduced to make the model lightweight, the model still successfully meets the above requirements. In the experiments, this study evaluated many mainstream object detection models combined with Deep SORT to achieve vehicle detection and tracking tasks. With the information from this study, if there are more powerful object detection

models or object tracking models in the future, they will be able to reproduce the experiments in this study and get better results.

8. References

Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B., 2016, September. Simple online and realtime tracking. In 2016 IEEE international conference on image processing (ICIP) (pp. 3464-3468). Ieee.

Bui, T., Wang, G., Wei, G. and Zeng, Q., 2024. Vehicle multi-object detection and tracking algorithm based on improved you only look once 5s version and DeepSORT. *Applied Sciences*, *14*(7), p.2690.

Chauhan, M.S., Singh, A., Khemka, M., Prateek, A. and Sen, R., 2019, January. Embedded CNN based vehicle classification and counting in non-laned road traffic. In *Proceedings of the tenth international conference on information and communication technologies and development* (pp. 1-11).

Chen, C., Guo, Z., Zeng, H., Xiong, P. and Dong, J., 2022. Repghost: A hardware-efficient ghost module via re-parameterization. *arXiv preprint arXiv:2211.06088*.

Deshmukh, G. and Uke, N. J. (2015) 'A Systematic Review of Image Based Moving Vehicle Detection Technique', International Journal of Science and Engineering Research, 4(9), pp. 15938. Available at: https://www.ijser.in/archives/v4i9/IJSER15938.pdf (Accessed: 1 March 2025).

Du, Y., Zhao, Z., Song, Y., Zhao, Y., Su, F., Gong, T. and Meng, H., 2023. Strongsort: Make deepsort great again. *IEEE Transactions on Multimedia*, *25*, pp.8725-8737.

Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C. and Xu, C., 2020. Ghostnet: More features from cheap operations. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 1580-1589).

Hussain, M., 2024. Yolov5, yolov8 and yolov10: The go-to detectors for real-time vision. *arXiv preprint arXiv:2407.02988*.

Jiang, P., Ergu, D., Liu, F., Cai, Y. and Ma, B., 2022. A Review of Yolo algorithm developments. *Procedia computer science*, *199*, pp.1066-1073.

Kamkar, S. and Safabakhsh, R., 2016. Vehicle detection, counting and classification in various conditions. *IET Intelligent Transport Systems*, *10*(6), pp.406-413.

Khanam, R. and Hussain, M., 2024. Yolov11: An overview of the key architectural enhancements. *arXiv preprint arXiv:2410.17725*.

Kumar, S., Singh, S.K., Varshney, S., Singh, S., Kumar, P., Kim, B.G. and Ra, I.H., 2023. Fusion of deep sort and Yolov5 for effective vehicle detection and tracking scheme in real-time traffic management sustainable system. *Sustainability*, *15*(24), p.16869.

Wojke, N., Bewley, A. and Paulus, D., 2017, September. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)* (pp. 3645-3649). IEEE.

Zhu, J., Hu, T., Zheng, L., Zhou, N., Ge, H. and Hong, Z., 2024. YOLOv8-C2f-Faster-EMA: an improved underwater trash detection model based on YOLOv8. *Sensors*, *24*(8), p.2483.