Efficient Detection Model of Illegal Driving Behavior in Two-Wheeled Vehicles

Liuyu Zhu[1,2] , Zhiguang Wang[1,2] , Zhiqiang Liu[1,2] , Xiaoxue Li[1,2] and Shen Li[1,2]

[1] China University of Petroleum, Beijing, Beijing 102200, China
[2] Beijing Key Laboratory of Petroleum Data Mining. Beijing 102200, China
liuyuzhu9528@gmail.com , cwangzg@cup.edu.cn

Abstract. The intelligent detection of illicit driving behaviors exhibited by two-wheeled vehicles, encompassing electric two-wheeled vehicles, motorcycles, and bicycles, constitutes a pivotal facet in developing a contemporary intelligent traffic monitoring system. However, prevailing challenges confront intelligent detection in this domain, manifesting in two principal predicaments. The first challenge is the absence of pertinent open-source datasets, and the second challenge is the suboptimal accuracy and swiftness in discerning illicit driving behavior of two-wheelers within the prevailing object detection model. In response to the aforementioned challenges, we put forth two potential solutions. Firstly, we construct the two-wheeled vehicle illegal driving behavior detection (TIDBD) dataset coupled with annotating 10 driving states, and secondly, we proposed an efficacious detection model, YOLOv8_VanillaBlock, tailored for detecting illegal driving behavior in two-wheeled vehicles. We experimentally compared YOLOv8_VanillaBlock with the original YOLOv8 using the TIDBD dataset employing evaluation metrics such as floating point operations (FLOPs), mean average precision (mAP), and GPU inference time. The outcomes indicate that YOLOv8_VanillaBlock yields superior detection results.

Keywords: Illigal driving behavior detection, Dataset, VanillaBlock.

1 INTRODUCTION

The detection of illegal driving behavior of two-wheeled vehicles is to ascertain whether the driver, captured in an image, is engaged in unlawful activities, such as helmet-less driving, cell phone usage, or smoking. The intelligent detection of illegal driving behavior in two-wheelers facilitates prompt interventions by traffic management departments, mitigating the occurrence of traffic accidents. Consequently, it holds significant practical implications.

However, two issues persist in the current deep neural network-based detection of two-wheeler illegal driving behavior. The first issue is a lack of pertinent open-source datasets. In the field of two-wheeler illegal driving behavior detection, most of the open source datasets only label whether the driver is wearing a helmet or not. [1,2], etc. Consequently, a comprehensive open-source dataset encompassing a complete set of labeled classes for detecting illegal riding behavior in two-wheelers is lacking. The second issue is as follows: Mainstream object detection models still exhibit potential for improvement in two-wheeler illegal driving behavior detection.

Hence, we initially formulated the two-wheeled vehicle illegal driving behavior detection (TIDBD) dataset. After that we designed the VanillaBlock module based on the idea of VanillaNet [3], resulting in the YOLOv8_VanillaBlock model.

The main contributions of this study are as follows:

- We constructed a two-wheeler illegal driving behavior detection dataset, compris ing 3637 images capturing 10 distinct illegal driving behaviors.
- We designed the VanillaBlock module with a reduced layer count, replacing the C2f module in YOLOv8. This adaptation led to the development of the YOLOv8 VanillaBlock model.
- Experiments conducted on the TIDBD dataset show that our proposed YOLOv8 VanillaBlock model yielded a significant improvement in detection speed, with a marginal enhancement in detection accuracy compared to the original YOLOv8.

2 RELATED WORK

2.1 Object Detection Based On Deep Learning

Object detection in deep learning can be broadly categorized into two types based on the algorithmic approach: two-stage and one-stage object detection algorithms. Most popular approaches favor one-stage algorithms because they are less time-consuming than two-stage algorithms. Representative examples of one-stage object detection algorithms are found in the YOLO series. Since the advent of YOLOv1[4], YOLO has undergone nine iterations. In 2020, the Ultralytics team introduced YOLOv5 [5], marking another leap in the YOLO series. YOLOv5 optimized the gradient repetitions in the backbone network by introducing the CSPNet [6] based on YOLOv4 [7]. It compressed model volume, and replaced the spatial pyramid pooling (SSP) module [8] with the more efficient spatial pyramid pooling-fast (SPPF) module, substantially improving computation speed. Since then, the YOLO algorithm has undergone two distinct versions: YOLOv6[9] and YOLOv7[10]. Another well known version of YOLO is YOLOv8[11]. YOLOv8 employed the C2f module from the E-ELAN network [10] instead of the C3 module in YOLOv5, providing enhanced gradient flow information.

2.2 Lightweight Feature Extraction Network

Exploring lightweight feature extraction networks remains a prominent focus in research. GhostNet [12] introduced the concept of “generating redundant feature maps by intrinsic feature maps” to design a lightweight neural network. Nevertheless, the utilization of depthwise separable convolution by Ghostnet enables accelerated processing on ARMS, CPUs. However, this acceleration is not observed on GPU devices with sufficient parallel computing capabilities. Therefore, Huawei Noah’s Ark Lab introduced G-GhostNet [13]. Chen et al. proposed VanillaNet [3] to further achieve acceleration on GPU. VanillaNet seeks to construct the network with as few layers as possible. Simultaneously, it introduces the series of activation function and the deep training strategy to address the limited nonlinear capability of shallow networks.

3 TIDBD DATASET

A paucity of datasets pertaining to the detection of illegal driving behavior on two-wheelers currently exists. Consequently, we have constructed our own dataset TIDBD which is tailored for detecting illegal driving behavior on two-wheeled vehicles, surpassed existing resources by encompassing a broader spectrum of scenarios, collection methods and diversity of annotation classes. This section explains the data collection methods, data cleaning contents, and data labeling techniques.

3.1 Data Acquisition

This study employed two approaches for data collection. The first method used a horizontal camera to capture footage at fixed intersections on inner-city roads. Continuous shooting was conducted at two specific intersections in Wuxi City, spanning from 7:05 am to 10:44 pm on August 11, 2023. We then extracted 1915 images from the captured videos. The second method entailed capturing images on national, provincial, and county highways and inner-city roads in Guizhou province, using overhead cameras positioned on the road. The second method involved collecting a total of 2183 images.

3.2 Data Cleaning and Annotation

Data cleaning adhered to two fundamental principles, ensuring the utilization of pertinent and valuable data for the model. The first principle entailed eliminating images devoid of a moving two-wheeler in the entire image. The second principle involved cleaning images containing mutilated two-wheelers. After data cleaning, a total of 3637 images were retained. The distribution of the cleaned data is presented in Table 1.

Data annotation also adhered to several principles. The first one is the occlusion problem, heavily occluded objects were treated as background and remained unlabeled because the collected data consisted of individual frames without continuity. The second principle pertained to the annotation of drivers using mobile phones. Conventional annotation methods, focusing solely on annotating the cell phone, are logically flawed for inferring whether the driver is using a cell phone. Additionally, the small and inconspicuous shape of the cell phone complicates both annotation and model detection. We found noticeable bending movement of the arm of the driver when using the cell phone. Therefore, we also labeled the arm movement (see Fig. 1(b)).

Fig. 1 depicts some of the labeling methods used in this study. Panel (a) illustrates the conventional labeling method for “use phone”, while panel (b) showcases the modified labeling method adopted in this study. Panels (c) and (d) represent the labeling classes for “helmet” and “no helmet ,” respectively. Panels (e) and (f) portray the labeling classes for “smoking” and “canopy (retrofit canopy)”, respectively.

Table 1. The proportion of different data collection methods and Collection locations in the clean dataset

	National Highways	Provincial Highways	Prefectural Highways	Inner-city Roads	Sum
Horizontal Camera	0%	0%	0%	100%	1654
Overhead Camera	2.67%	8.47%	2.27%	86.59%	1983

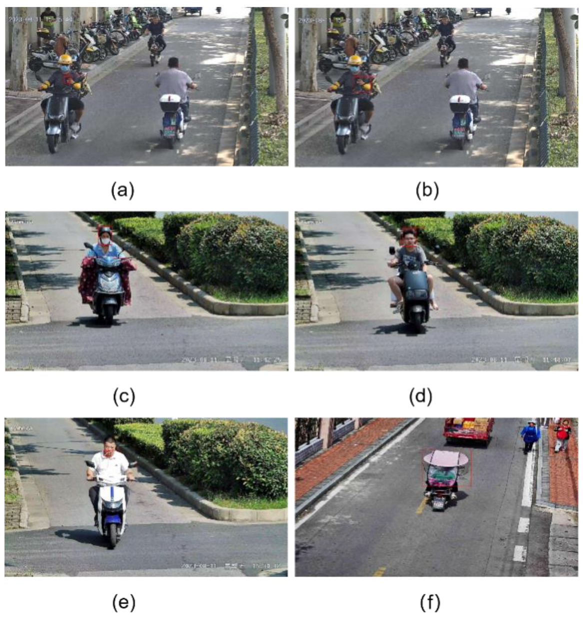

Fig.1. Displaying the certain label classes

4 METHOD

Despite the capability of the YOLOv8 detection model to produce results, it exhibited slow detection speed and unsatisfactory accuracy when applied to our dataset. Hence, we replaced the C2f module in the backbone of the YOLOv8 model with our improved VanillaBlock. This section introduces the key components of YOLOv8, including the CBS module, Bottleneck, and C2f module. Furthermore, it outlines the structure of the backbone network used by YOLOv8, followed by an in-depth explanation of the backbone network within our proposed YOLOv8_VanillaBlock. The mathematical proof of the acceleration ratio of the VanillaBlock relative to the C2f module is also discussed.

4.1 YOLOv8 Backbone

We introduce some key components before delving into the backbone network of YOLOv8, namely the CBS module, the Bottleneck, and the C2f module.

CBS Module. The main module for convolutional operations in YOLOv8 was the CBS module. It comprised a convolutional layer (Conv2d), a batch normalization layer (BN), and a SiLU activation function.
Bottleneck. The Bottleneck module adopted a distinctive residual structure incorporating multiple small convolution kernels instead of a large one. This structural modification enhanced network depth while concurrently reducing the overall number of parameters. YOLOv8 integrated two variants of the Bottleneck module: with or without a shortcut connection.
C2f Module. C2f essentially combines the ideas of C3 module and ELAN module [14]. C2f module improves the network structure while obtaining rich gradient streaming information, resulting in enhanced performance of the YOLOv8 model. However, the C2f module required feature fusion, which increased the computational complexity of the model, leading to extended training and inference times. Fig. 2 provides an overview of the comprehensive structure of the C2f module.

Based on the above components, we give the structure diagram of YOLOv8(Fig. 3.)

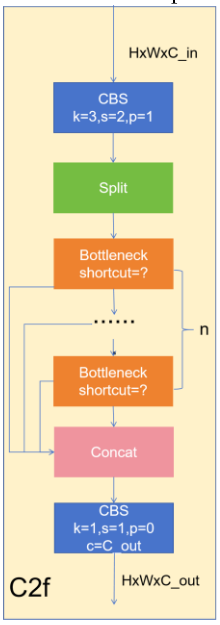

Fig. 2 C2f module structure diagram

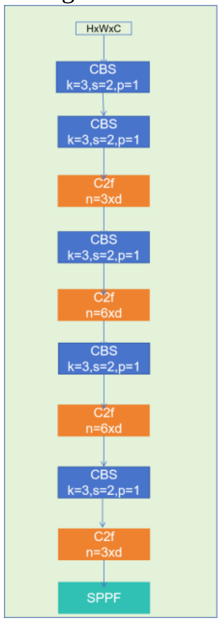

Fig. 3. YOLOv8 backbone diagram

4.2 VanillaBlock

Chen et al. [3] discovered that the computational speed of neural networks encountered a bottleneck in the number of layers rather than the number of parameters when using GPUs with sufficient parallel computing power. We built upon this concept to introduce VanillaBlock. Unlike the block used in Vanillanet, our VanillaBlock did not incorporate the deep training strategy. This deviation was intentional as the deep training strategy while reducing computational effort during the inference stage, presents challenges such as high training costs, and slow convergence speed. The impact of not employing the deep training strategy on the nonlinear loss was deemed insignificant since VanillBlock was not exclusively used to construct the backbone of YOLOv8_VanillaBlock but served as an integral part of the entire backbone network together with other modules. Fig. 4 illustrates the structure of our VanillaBlock, consisting of two convolutional layers and two pooling layers with an activation function in between. The final component of the VanillaBlock module was the series activation function consisting of n activation functions, as represented by equation (1).

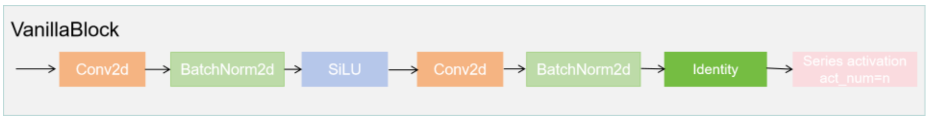

Fig. 4. VanillaBlock module structure diagram

$$A_s(x) = \sum_{i=1}^{n} a_i A(x + b_i) \tag{1}$$

Equation (1) defines $A(x)$ as a single activation function in the neural network with input x. The scaling factor and bias of each activation function, $a_i$and b_i respectively, are used to prevent the accumulation of identical activation functions.

We estimated the computational complexity of each component in both VanillaBlock and the C2f module to demonstrate that our proposed VanillaBlock requires less computation than the C2f module. The equations used for the estimation are provided below

$$O(CBS_i) = H \times W \times C_{in} \times C_{out} \times k_i \times k_i \tag{2}$$

$$O(Bottleneck) = H \times W \times C_{in} \times C_{out} \times 2k_B^2 \times num \times d \tag{3}$$

$$O\left(CB_j\right) = H \times W \times C_{in} \times C_{out} \times k_j \times k_j \tag{4}$$

$$O(SA) = H \times W \times C_{in} \times n^2 \tag{5}$$

We only considered the computational complexity of the convolution operation in the C2f module to simplify the equations because it required more computation than the batch normalization and activation layers. However, we considered the sum of the computational complexity of the convolutional layer and the activation layer in VanillaBlock. Our standards for computational complexity differed in the C2f module and

VanillaBlock. However, this difference better illustrated that VanillaBlock required less computational complexity. Additionally, we assumed that the input feature maps of each module had the same height (H), width (W), number of input channels (C_{in}), and number of output channels (C_{out}).

Equation (2) defines $O(CBS_i)$ as the computational complexity of the convolution operation in the i_{th} CBS structure in the C2f module, where $i = 1,2$, and k_i denotes the convolution kernel size in the i_{th} CBS structure. Equation (3) defines $O(Bottleneck)$ as the computational complexity of the Bottleneck structure in the C2f module, where, k_B corresponds to the size of the convolution kernel used in the Bottleneck structure, num represents the number of Bottlenecks required in the C2f module (which varied in different C2f modules at different locations in YOLOv8), and d denotes the scaling factor for different C2f module in different sizes of YOLOv8 models.Equation (4) defines $O(CB_j)$ as the computational complexity of the j_{th} CB structure (Conv+BatchNorm) in VanillaBlock, where $j = 1,2$, and k_{vj} denotes the size of the convolution kernel used in the j_{th}CB structure. Equation (5) defines $O(SA)$ as the computational complexity of the series activation structure in VanillaBlock, where n corresponds to the number of activation functions in the series activation function. The speedup ratio S of VanillaBlock with respect to C2f can be calculated using (5)-(6).

$$S = \frac{O(C2f)}{O(VanillaBlock)} = \frac{\sum_{i=1}^{2} O(CBS_i) + O(Bottleneck)}{\sum_{i=1}^{2} O(CB_j) + O(SA)} \tag{5}$$

$$S = \frac{C_{out} \times (k_1^2 + k_2^2 + 2k_B^2 \times num \times d)}{C_{out} \times (k_{v1}^2 + k_{v2}^2) + n^2} \tag{6}$$

For example, let us examine the last C2f module of the YOLOv8l backbone and the last VanillaBlock of the YOLOv8_VanillaBlock_l backbone. In this case, we have $k_1 = k_B = 3$, $num = 3$, $d = 1$,$k_2 = 1$, $k_{v1} = k_{v2} = 1$, and $n = 4$. We can ignore n because it is significantly smaller than C_{out}, leading us to conclude that $S = 32$.

4.3 YOLOv8_VanillaBlock

We developed a novel backbone network named YOLOv8_VanillaBlock to enhance the detection speed of YOLOv8 on GPUs, built upon the VanillaBlock. The overall structure of YOLOv8_VanillaBlock is elucidated in Fig. 5. YOLOv8_VanillaBlock comprised five CBS modules and four VanillaBlock modules, culminating with the SPPF module.

Fig. 2. indicates that each C2f module encompassed a substantial number of bottlenecks involving operations such as feature fusion and shortcut connections, increasing the complexity of the network. Additionally, each bottleneck module includes at least two ordinary convolutional modules and several activation functions, resulting in a high number of layers that are not suitable for GPU parallel computing. We replaced the C2f module—the most intricate module with the highest number of layers in the YOLOv8 backbone—with the straightforward and efficient VanillaBlock module introduced in the preceding section to alleviate the computational burden and streamline the complexity of the backbone network. We retained the CBS module to increase the nonlinearity of the network and the already efficient SPPF module.

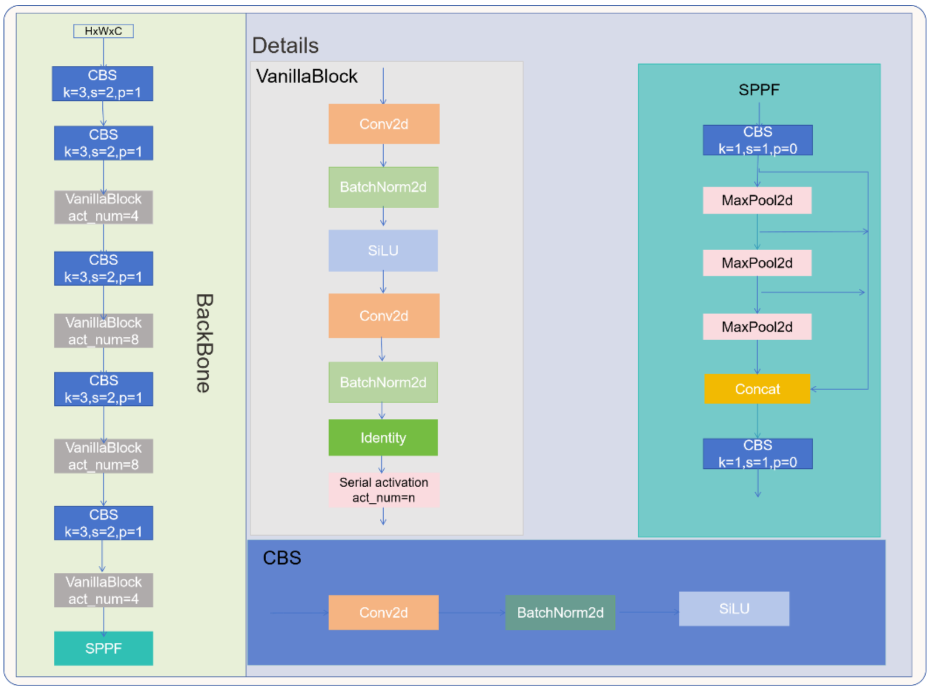

Fig. 5. YOLOv8_VanillaBlock backbone network structure diagram

5 EXPERIMENT

The TIDBD dataset, introduced in Chapter 3, was divided into three parts: 10% was allocated for the test set, 15% for the validation set, and the remaining part served as the training set. The NVIDIA RTX 3090 GPU was employed for training, while the NVIDIA RTX 3070ti GPU was used for testing.

5.1 Metrics

The metric employed for accuracy was mAP50, denoting the mean average precision (mAP) across each class at an IOU threshold of 0.5. Equation (7) shows the calculation formula.

$$mAP = \frac{\sum_{c \in C} AP_c}{|C|} \tag{7}$$

where C represents all classes to be detected and AP_c corresponds to the average precision for class c.

The detection speed was evaluated by considering the time required for the entire detection model to predict an image, including preprocessing time, inference time, and postprocessing time. We used the frames per second (FPS) evaluation metric to demonstrate the improved real-time performance of our method.

Furthermore, we tested the number of FLOPs required by each model. This metric offered a more intuitive explanation for the reduction in detection time achieved by our models on GPUs.

5.2 Evaluation Using TIDBD Dataset

Comparison of Detection Accuracy and FLOPs. We conducted experiments on the TIDBD dataset using larger-sized YOLOv8l and YOLOv8_VanillaBlock_l models to balance detection accuracy and model size. We compared the two models based on the average precision (AP50) evaluation criterion. The experimental results (Fig. 6) demonstrated that our proposed YOLOv8_VanillaBlock_l outperformed the original YOLOv8l in detecting “use phone”, “canopy” and other classes.

We conducted experiments on the following five models to illustrate the effectiveness of the proposed VanillaBlock in improving accuracy and reducing computation across all YOLOv8 sizes: n, s, m, l, and x. The results are summarized in Table 2, which clearly shows the significant advantages of our YOLOv8_VanillaBlock model across all sizes. Our proposed YOLOv8_VanillaBlock model with corresponding sizes reduced the FLOPs by 18.5%, 21.8%, 30.2%, 26.7%, and 37.5% compared to the original YOLOv8 model with sizes n, s, m, l, x, respectively. It improved mAP50 by 1.45%, 0.5%, 0.18%, 1.31%, and 2.2%, respectively. These results demonstrate that our YOLOv8_VanillaBlock model attained higher accuracy while reducing FLOPs than the original YOLOv8 model.

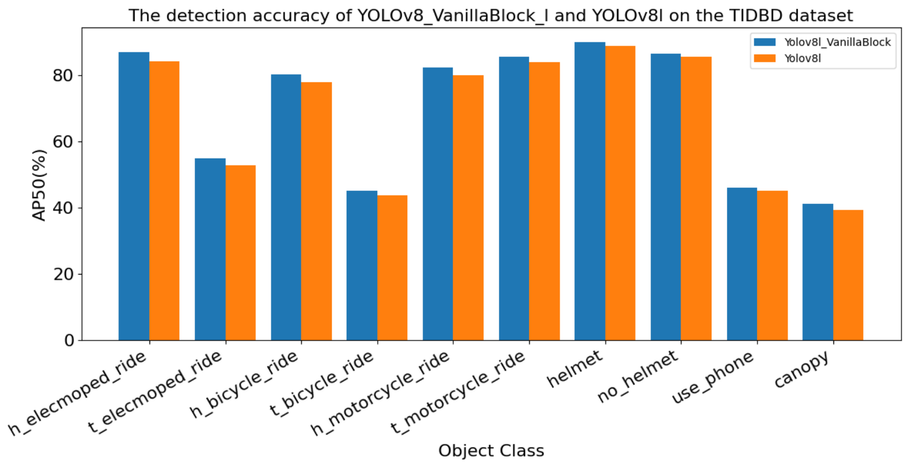

Fig. 6. Comparison of AP50 of different detection classes on the TIDBD dataset between YOLOv8_VanillaBlock_l and YOLOv8l

Table 2. Comparison of mAP50 and FLOPs between all sizes of YOLOv8_VanillaBlock and YOLOv8

Models	FLOPs(G)	mAP50	Image size (pixels)
YOLOv8n	8.1	49.10	1088
YOLOv8s	28.5	61.63	1088
YOLOv8m	78.8	64.15	1088
YOLOv8l	165.0	64.79	1088
YOLOv8x	257.6	66.73	1088
YOLOv8_VanillaBlock_n	6.6	50.55	1088
YOLOv8_VanillaBlock_s	22.3	62.13	1088
YOLOv8_VanillaBlock_m	55.0	64.33	1088
YOLOv8_VanillaBlock_l	120.9	66.10	1088
YOLOv8_VanillaBlock_x	161.1	68.93	1088

Comparison of Detection Speed and Real-time Performance. We conducted tests on detection time and FPS for input images scaled to 320, 640, and 1088 to underscore the superiority of the proposed model in terms of detection speed and real-time performance. The detection time test results are presented in Fig. 7, and the FPS test results are depicted in Table 3.

Fig. 7 shows the models of the same size by lines of the same color, with YOLOv8_VanillaBlock represented by a solid line and original YOLOv8 by a dashed line. From Fig. 7 we can see that the detection time of the YOLOv8_VanillaBlock model was consistently lower than that of the original YOLOv8 model of the corresponding size for the same input image size. The improvement in detection speed was notably pronounced when the input image size was 1088.

Table 3 illustrates that, with the input images of the same size, the YOLOv8_VanillaBlock model processed a higher number of frames per second than the original YOLOv8 model. When the input image was 1088, the FPS of YOLOv8_VanillaBlock was 14.2%, 3.9%, 13.7%, 16.1%, 30.2% higher than that of original YOLOv8 of the corresponding size. The outcome demonstrates that our proposed method can significantly enhance the detection speed of YOLOv8.

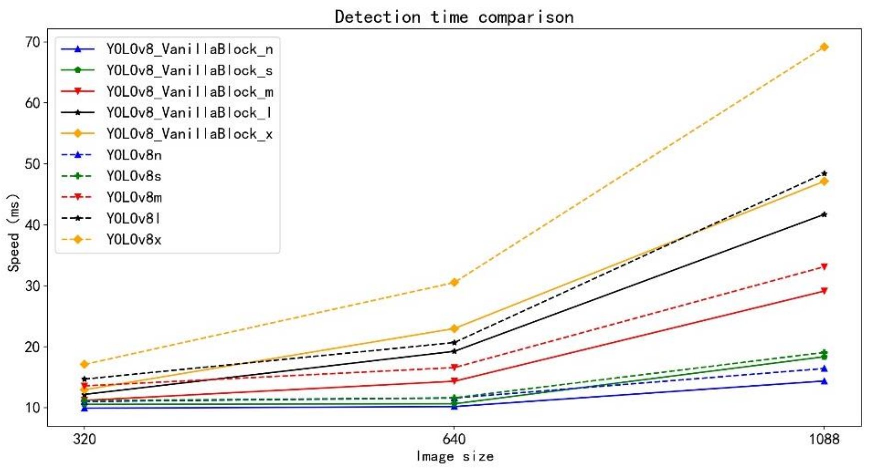

Fig. 7. Comparison of detection speed of YOLOv8_VanillaBlock and YOLOv8 on images of different scaled versions

Table 3. FPS of YOLOv8_VanillaBlock VS YOLOv8 with different sizes of input images

Image sizes(pixels) / Models	320	640	1088
YOLOv8n	90.87	86.43	60.94
YOLOv8s	88.51	86.21	52.49
YOLOv8m	73.74	60.38	30.20
YOLOv8l	68.17	48.37	20.65
YOLOv8x	58.56	32.76	14.46
YOLOv8_VanillaBlock_n	101.03	98.38	69.62
YOLOv8_VanillaBlock_s	94.90	94.17	54.52
YOLOv8_VanillaBlock_m	89.48	69.82	34.35
YOLOv8_VanillaBlock_l	82.22	53.30	23.98
YOLOv8_VanillaBlock_x	77.39	43.54	18.84

6 CONCLUSION

This study introduces the detection of illegal driving behavior in two-wheelers employing the YOLOv8_VanillaBlock model. In the initial phase, real-world data was collected, cleaned, labeled, and used to construct the TIDBD dataset. Subsequently, the YOLOv8_VanillaBlock model was proposed. Next, we mathematically proved that our proposed model has smaller computational complexity. Finally, we did a comparative experiment on TIDBD dataset. The experimental results on the TIDBD dataset showed a substantial increase in detection speed with a minor improvement in accuracy compared to YOLOv8. Future endeavors include accomplishing more complex tasks of detecting two-wheeler illegal driving behavior on our dataset while continually improving the performance of the model.

Acknowledgments. This study was funded by Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum, Beijing.

References

1. “TWHD:two wheeler helmet dataset.” https://gitee.com/bilibilee/TWHD, last accessed 2024/3/25
2. “Helmet Detection.” https://tianchi.aliyun.com/dataset/90136, last accessed 2024/3/25
3. H. Chen, Y. Wang, J. Guo, and D. Tao, “Vanillanet: the power of minimalism in deep learning,” arXiv preprint arXiv:2305.12972 (2023)
4. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” In: IEEE conference on computer vision and pattern recognition, pp. 779–788 IEEE Press Las Vegas, NV, USA (2016)
5. Ultralytics, “Model Structure.” https://docs.ultralytics.com/yolov5/tutorials/architecture description/#1-model-structure, last accessed 2024/3/25
6. C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, “Cspnet: A new backbone that can enhance learning capability of cnn,” In: IEEE conference on computer vision and pattern recognition workshops, pp. 390–391. IEEE Press Seattle, WA, USA (2020)
7. A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” arXiv preprint arXiv:2004.10934 (2020)
8. K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” IEEE transactions on pattern analysis and machine intelligence, 37 (9), pp. 1904–1916 (2015)
9. C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie et al., “Yolov6: A single-stage object detection framework for industrial applications,” arXiv preprint arXiv:2209.02976 (2022)
10. C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 7464–7475. IEEE Press Vancouver, BC, Canada (2023)
11. “YOLO by Ultralytics,” https://github.com/ultralytics/ultralytics, 2023,last accessed 2024/3/25
12. K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “Ghostnet: More features from cheap operations,” In: IEEE conference on computer vision and pattern recognition, pp. 1580–1589. IEEE Press Seattle, WA, USA (2020).
13. K. Han, Y. Wang, C. Xu, J. Guo, C. Xu, E. Wu, and Q. Tian, “Ghostnets on heterogeneous devices via cheap operations,” International Journal of Computer Vision, 130(4), pp. 1050–1069 (2022)
14. X. Zhang, H. Zeng, S. Guo, and L. Zhang, “Efficient long-range attention network for image super-resolution,” In: European Conference on Computer Vision, pp. 649–667. Springer, Tel Aviv, Israel (2022)