mathematics

MDPI

Article

Driver Distraction Detection Based on Cloud Computing Architecture and Lightweight Neural Network

Xueda Huang [1], Shaowen Wang [1], Guanqiu Qi [2,*], Zhiqin Zhu [1], Yuanyuan Li [1], Linhong Shuai [3], Bin Wen [4], Shiyao Chen [4] and Xin Huang [1]

[1] College of Automation, Chongqing University of Posts and Telecommunications, Chongqing 40065, China; huangxd@cqupt.edu.cn (X.H.); S220333022@stu.cqupt.edu.cn (S.W.); zhuzq@cqupt.edu.cn (Z.Z.); liyy@cqupt.edu.cn (Y.L.); huangxin@cqupt.edu.cn (X.H.)
[2] Computer Information Systems Department, State University of New York at Buffalo State, Buffalo, NY 14222, USA
[3] Intelligent Interaction R&D Department, Chongqing LiLong Zhongbao Intelligent Technology Co., Chongqing 40065, China; shuailinhong@li-hong.com.cn
[4] Chongqing Dima Industrial Co., Ltd., Chongqing 40065, China; wenbin@dima.cn (B.W.); chenshiyao@dima.cn (S.C.)
* Correspondence: qig@buffalostate.edu

Abstract: Distracted behavior detection is an important task in computer-assisted driving. Although deep learning has made significant progress in this area, it is still difficult to meet the requirements of the real-time analysis and processing of massive data by relying solely on local computing power. To overcome these problems, this paper proposes a driving distraction detection method based on cloud–fog computing architecture, which introduces scalable modules and a model-driven optimization based on greedy pruning. Specifically, the proposed method makes full use of cloud–fog computing to process complex driving scene data, solves the problem of local computing resource limitations, and achieves the goal of detecting distracted driving behavior in real time. In terms of feature extraction, scalable modules are used to adapt to different levels of feature extraction to effectively capture the diversity of driving behaviors. Additionally, in order to improve the performance of the model, a model-driven optimization method based on greedy pruning is introduced to optimize the model structure to obtain a lighter and more efficient model. Through verification experiments on multiple driving scene datasets such as LDDB and Statefarm, the effectiveness of the proposed driving distraction detection method is proved.

Keywords: driving distraction behavior detection; cloud–fog computing architecture; service computing; scalable networks; lightweighting

MSC: 68T07

Citation: Huang, X.; Wang, S.; Qi, G.; Zhu, Z.; Li, Y.; Shuai, L.; Wen, B.; Chen, S.; Huang, X. Driver Distraction Detection Based on Cloud Computing Architecture and Lightweight Neural Network. *Mathematics* **2023**, *11*, 4862. https://doi.org/10.3390/math11234862

Academic Editors: Ke-Lin Du and Jonathan Blackledge

Received: 19 October 2023
Revised: 24 November 2023
Accepted: 27 November 2023
Published: 4 December 2023

1. Introduction

Driving detection as a key task in Intelligent Transportation Systems (ITS) plays an indispensable role in the field of traffic safety. Among them, driver distraction detection [1,2] aims to analyze and identify driver behaviors to improve traffic safety and prevent traffic accidents. Due to the advantages and improvements of target detection algorithms based on deep learning in terms of feature learning capabilities, adaptability, and multi-modal data processing capabilities, they [3,4] are widely used in assisted driving, traffic monitoring and other fields, providing strong support for the realization of intelligence and automation.

Early driving detection algorithms relied on traditional computer vision technology and machine learning methods, usually using hand-designed feature extractors [5] to classify and identify driver behaviors, which to a certain extent limited the universal application capabilities and adaptability of the algorithm. With the development of deep learning,

especially the widespread application of convolutional neural networks (CNN), driving detection algorithms based on deep learning gradually replace traditional methods. Due to the widespread application of vehicle-mounted edge devices with limited computing resources, the demand for real-time edge computing in driving detection scenarios is very urgent. However, traditional deep convolutional neural networks, such as VGG [6] and ResNet [7], have a large number of parameters and calculations, which makes them difficult to deploy on resource-constrained edge devices, especially for terminals with weak computing capabilities such as mobile devices. From this, the subfield of lightweight networks was born, which focuses more on the trade-off between speed and accuracy, making the networks lightweight and small enough to be deployed on in-vehicle edge terminal devices. The two most famous lightweight networks mobilenet [8] and shufflenet [9] are examples.

Traditional convolutional neural networks tend to process fixed-size input images and may not perform well on target objects or scenes of different sizes and scales. Recently, more and more research has been conducted by using scalable structures [10,11], in order to maintain excellent detection performance on driving behavior images and be able to adapt to feature extraction at different scales. This is especially important for handling different scenarios and adapting to different levels of features.

Sufficient and effective driving scene data are prerequisites for using deep learning models to detect driver distraction behavior. With the popularization and development of Internet of Things (IoT) technology, more and more IoT devices such as sensors, cameras, and on-board computing are widely used in the automotive field, providing massive data sources for driving detection. Additionally, local devices have limited computing power and insufficient storage capacity, making it difficult to process these massive data quickly and accurately. Thus, cloud computing technology was introduced to solve this problem.

Cloud computing [12,13] as a new technology provides computing resources, storage and services through the network. It has powerful parallel processing capabilities and can efficiently process massive data in the cloud. Through device–cloud collaboration, that is, making full use of the efficient parallel data processing capabilities of cloud computing and the real-time computing capabilities of local terminals, massive data calculation tasks in the driving detection can be transferred to the cloud for processing, and real-time calculations and decisions can be performed on local terminals to achieve a more efficient and stable driving detection.

Edge computing technology refers to the methods of data processing, storage and computation in close proximity to the data source, and its main purpose is to improve the real-time and efficiency of data processing. By deploying lightweight networks and related detection algorithms on edge-end devices such as in-vehicle devices, driving behavior data can be processed instantly at the place where the data are generated, which reduces data transmission latency and improves the real-time performance. This design not only reduces the burden of cloud computing, but also adapts well to the complex and changing driving environment.

Cloud–fog computing architecture as a distributed computing architecture combines cloud computing and fog computing. It aims to push computing, storage and data processing capabilities to the edge of the network to meet the demand for real-time, low-latency, edge computing in the context of the rapid growth of large-scale data and IoT devices. A large amount of data generated from cameras, sensors and other monitoring devices at the end-user layer is initially processed at the edge device to extract key information about the driver's behaviors. This information is sent to the fog computing layer, constituting a complex data stream. At the fog computing layer, the data are processed in a real-time, continuous manner during transmission by employing streaming processing techniques. Device–cloud collaboration is an important part of the cloud–fog computing architecture. By coordinating the computing capabilities of cloud and edge devices, more efficient and real-time computing data processing and applications can be achieved. In practice, the cloud–fog computing architecture is formed by combining edge computing with cloud computing. The cloud–fog computing architecture establishes a close collaboration between

the edge-end devices and the high-performance computing resources in the cloud, realizing instant data transmission, real-time processing and distributed computing.

Service computing as a computing paradigm aims to meet the changing needs of users by providing computing resources and capabilities as scalable and flexible services. This emphasizes migrating computing from traditional local devices and applications to cloud computing platforms to achieve a more efficient, scalable and manageable computing environment. Service computing can play a key role in distracted driving detection based on cloud computing architecture. Inspired by research related to data complexity, the need for a deeper understanding and application of new data processing and analysis methods can be realized when processing and analyzing large-scale, complex data sets. Through service computing, driving data and monitoring equipment can be connected to the cloud to collect and analyze driver biometric data and driving behavior data in real time. These data can be processed in a service deployed in the cloud that leverages advanced deep learning algorithms to identify signs of distracted driving, such as taking your eyes off the road or taking your hands off the steering wheel. Once distracted driving is detected, the system can quickly issue a warning to remind the driver to refocus, thus improving road safety. In addition, service computing can support the storage, sharing and analysis of data, thereby continuously improving the performance and accuracy of distracted driving detection systems and providing drivers with a safer road experience.

This paper proposes a distracted behavior detection method based on cloud computing architecture, aiming to achieve a full utilization of multimodal data in driving scenarios and ensure the accurate real-time detection of distracted driving behavior. The proposed method contains scalable modules and a model-driven optimization strategy based on greedy pruning, which makes full use of cloud computing to process complex driving scene data and solves the problem of local computational resource limitation. For feature extraction, scalable modules are used to accommodate different levels of feature extraction to effectively capture the diversity of driving behaviors. In addition, to improve model performance, a model-driven optimization method based on greedy pruning is introduced to optimize the model structure and obtain a lighter and more efficient model. Through validation experiments on multiple driving scenario datasets such as LDDB and Statefarm, the effectiveness of the proposed distracting behavior detection method is verified. The main contributions of this paper are given as follows.

1. This paper uses the advantages of cloud computing architecture in big data processing and edge computing deployment to propose a driver distraction behavior detection method that supports cloud and fog computing. By training deep learning models in a cloud computing environment, and then deploying the trained models to edge devices with limited computing resources, the model is updated and optimized on edge devices through a two-level optimization path of data-driven and model-driven devices.
2. The Progressive Scalable Detection Network (PSDNet) is introduced, which fuses a multi-branch scalable perceptual backbone network and a lightweight progressive feature pyramid, aiming to decouple the training time and inference time of the model by employing structural reparameterization and simultaneously quantizing the scalable network to improve detection accuracy and efficiency.
3. A model-driven approach based on the performance-aware approximation integrating a sequential greedy channel pruning algorithm and a performance-aware prediction criterion is proposed. Experimental results show that the proposed model-driven approach can reduce FLOPs and parameters by more than 30% with small performance degradation and achieve 1.2× to 2.0× speedups on cloud and edge mobile platforms.

The remaining sections of this paper are organized as follows. Section 2 reviews related work on object detection methods and cloud computing. Section 3 discusses the proposed and progressively scalable detection network based on cloud computing architecture. Section 4 presents the experimental dataset, evaluation indicators and experimental results

for the driver's abnormal behavior detection tasks during driving. Section 5 concludes this paper and proposes future research directions.

2. Related Work

2.1. Driving Distraction Detection

In recent years, CNNs have become a popular choice for many computer vision tasks, especially in challenging tasks such as target detection. Currently, single-stage object detectors such as YOLOv5 have become the first choice for driving detection scenarios with high real-time requirements due to their excellent balance between speed and accuracy.

Recently, various lightweight network technologies have demonstrated superior performance and real-time computing capabilities in various computer vision tasks, including driving detection. Deploying a model with a large number of parameters and a bloated structure on an edge computing device often causes local real-time computing on the edge device to face problems of insufficient computing power and memory limitations. In order to solve the above problems and enable the model to perform real-time calculations on edge devices, the model can be enabled to perform real-time calculations through pruning [14,15], quantization [16,17] and other techniques to make the network more lightweight. For example, Mingxing Tan et al. [18] proposed an automated Mobile Neural Network Architecture Search method (MNAS) to achieve a model that balances accuracy and latency. Furthermore, Chen [19] proposed a new partial convolution (PConv) that can extract spatial features more efficiently by simultaneously reducing redundant computation and memory access.

In addition, scalable networks [20] have the advantage of adapting to different scenarios and characteristics, and can flexibly adjust the structure and parameters to achieve efficient computing and better performance. For example, Inception [21] is a deep convolutional neural network proposed by the Google team. Its main feature is the use of multi-scale convolution kernels and multiple parallel convolution layer branches, which effectively improves the expression ability and computational efficiency of the model. NASFPN [22] uses a neural architecture search algorithm to automatically search for the optimal connection structure. Diverse Branch Block [23] enriches the feature space of convolutional blocks with a multi-branch structure. A feature pyramid network proposed by Guoyu Yang et al. [24] supports the direct interaction of non-adjacent layers to avoid large semantic gaps between non-adjacent layers during the feature fusion process. Therefore, this paper proposes a scalable network model specifically designed for driving distraction behavior detection.

2.2. Cloud Computing in Driving Distraction Detection

The challenges in the field of driving behavior detection have increased with the continuous innovation of driving behavior detection techniques and the dramatic increase in driving behavior data. To cope with these challenges, researchers have gradually recognized the limitations of traditional computing resources that cannot meet the demand of processing large-scale driving behavior data. Against this background, cloud–fog computing architecture has become a highly sought-after solution. Cloud fog computing not only combines high-performance computing resources in the cloud with edge-side computing devices, but also provides powerful data storage and processing capabilities.

In the past few years, the application of deep learning technology in driving behavior detection has gained widespread attention and recognition. Deep learning models are able to automatically learn complex driving behavior patterns for the efficient data analysis and anomaly detection. However, with the increasing data size, the processing power of a single computing device can hardly meet the demand for the real-time processing of large-scale data. Therefore, integrating deep learning technology into cloud–fog computing through end-cloud collaboration has become a research hotspot.

Under the cloud computing architecture, the driving distraction detection algorithm can perform complex data analysis and feature extraction with the help of high-performance computing resources in the cloud. At the same time, edge-side devices can also collect

driving behavior data in real time and transmit the data to the cloud for further processing. This distributed computing model greatly improves the processing efficiency and real-time performance of driving behavior data. Moreover, in the cloud, researchers can use big data technology to dig deeper into the driving behavior data and discover the hidden laws and patterns, providing more insights for the improvement of driving distraction detection algorithms.

Overall, the integration of deep learning technology and cloud computing architecture brings unprecedented opportunities to the field of driving distraction detection. It not only improves the efficiency of data processing, but also provides more possibilities for learning and analyzing driving behavior patterns. This fusion of innovations will propel driving distraction detection technology to new heights, injecting a strong impetus for the development of intelligent driving systems.

Yan et al. [25] built a traffic cloud measurement collaboration platform to realize the collaboration and unification of various business resources and data on both sides of the vehicle cloud. Xun et al. [26] proposed a driving behavior assessment scheme based on vehicle edge cloud architecture. Inspired by edge collaboration, Khan et al. [27] proposed an automated framework based on embedded systems, edge computing and cloud computing modules for monitoring the driver behavior.

3. The Proposed Method

3.1. Progressive Scalable Detection Networks

This section provides an in-depth explanation of the design of the Progressive Scalable Detection Network, which covers the overall architecture, the multi-branch scalable sensing backbone, and the lightweight progressive feature pyramid. The overall architecture section highlights the overall framework and emphasizes the synergy among the components. The presentation of the multi-branch scalable perceptual backbone explains how this design enables efficient perception and cooperative work when dealing with multimodal data. The section on lightweight progressive feature pyramids highlights the flexibility of extracting features at different levels.

3.1.1. Overall Architecture

The overall architecture of the proposed PSDNet is shown in Figure 1. It consists of three parts, the backbone structure, the neck and the head. The backbone network is responsible for extracting features from the input image and is used to capture the low- and mid-level features of the image. The neck network further processes and integrates the features extracted by the backbone network to obtain higher-level semantic features. The head network performs the final prediction task of object detection.

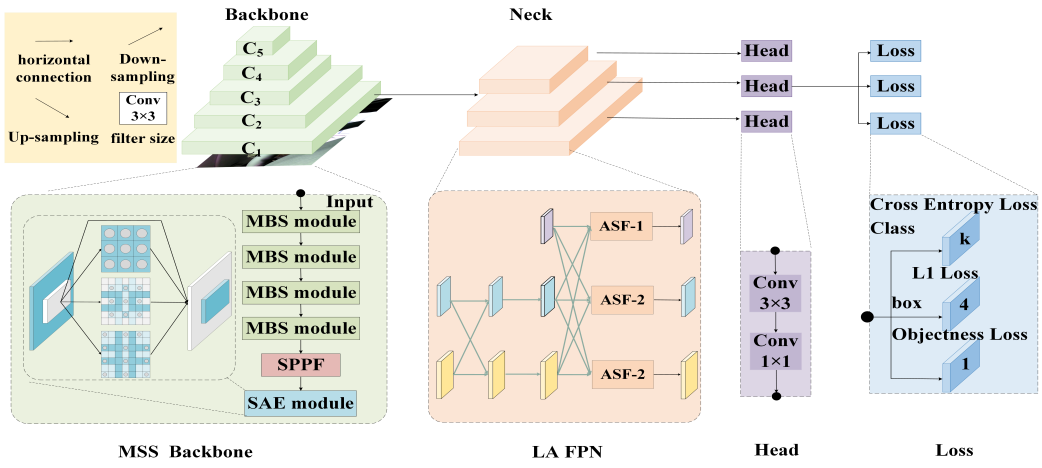

Figure 1. The Overall Architecture of The Proposed Progressive Scalable Detection Network.

The proposed new network architecture design mainly consists of a multi-branch scalable sensing backbone network and lightweight asymptotic fusion blocks. The multi-branch scalable sensing backbone network aims to decouple the model structure at the training and inference by employing structural reparameterization. The model is a complex multi-branch structure during training to improve performance, while the multi-branch structure is equivalently fused into a simple one-way structure during inference. Even with a simple one-way structure, the multi-branch structure still has the high performance gained from training, thereby reducing inference time loss. A scale-aware enhancement (SAE) module is connected to the final output top-level feature map to expand the perceptual field while fusing multi-scale features.

According to the idea of asymptotic architecture design, the lightweight asymptotic feature pyramid starts from the fusion of adjacent low-level features and gradually incorporates high- and low-level features into the fusion process to avoid large semantic gaps between non-adjacent layers. It also introduces lightweight offset convolutions in the feature fusion stage, aiming to achieve lower memory footprint and higher speed while maintaining the same output as the original convolutions.

3.1.2. Multi-Branch Scalable Sensing Backbone

The multi-branch scalable sensing backbone mainly consists of two parts, the multi-branch scaling module and the scale-aware module, as shown in Figure 2. The structure of the multi-branch scaling module during training contains four branches with different combinations of convolution and pooling operations, and has convolution branches of different scales, so that different sizes of sensory fields can be obtained. Therefore, it can enhance the representation ability of a single convolutional layer by combining multiple branches at different scales, thereby generating a richer feature space. In addition, the nonlinear capability of the batch normalization (BN) layer during training also benefits the performance of the multi-branch scaling module, so that each individual branch is followed by a BN layer.

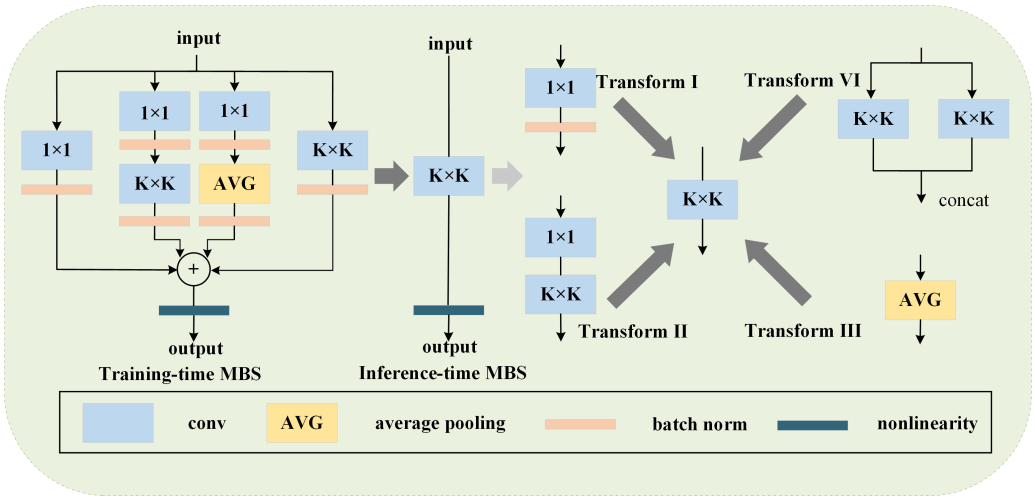

Figure 2. Four Transformation Paradigms of The Multi-branch Scaling Module.

The convolution operation is also essentially a linear operation, so in some cases convolution has some linear properties. (1) Additivity means that in the case of two convolution kernels with the same shape, the convolution result satisfies additivity, as shown in Equation (2).

$$\text{Input} \odot W_1 + \text{Input} \odot W_2 = \text{Input} \odot (W_1 + W_2) \quad (1)$$

where W_1 and W_2 denote two independent convolution operations, respectively. (2) Homogeneity is shown in Equation (2)

$$\text{Input} \odot (pW_1) = p(\text{ Input } \odot W_1). \tag{2}$$

The transformations of the multi-branch scaling module are operated based on the above two basic attributes. After the model training is completed, the parameters of the trained multi-branch structure are first calculated as a linear combination of the corresponding single-path structure parameters, then set into the deployment model, and finally retained and used only in the deployment model.

Four different branch transformations are summarized as shown in Figure 2 to implement a multi-branch scaling module through BN layers, deep cascading, average pooling and convolution sequences.

Conversion 1—Conv-BN Fusion: In CNNs, convolutional layers and BN layers usually appear in pairs, with the BN layer performing channel-by-channel regularization and linear scale deflation. Let j be the channel index, μ_j, σ_j be the cumulative channel-by-channel mean and standard deviation, respectively, and γ_j, β_j be the learned scale factor and bias term, respectively. The chi-square property of convolution allows incorporating BN operations into the previously mentioned conv for inference. In practice, it only needs to construct a convolution kernel F' and bias b', and assigns the converted values of the parameters of the original BN sequence to them.

Conversion 2—Sequence Convolutional Fusion: A sequence of 1×1 conv-BN-$k \times k$ conv can be merged into a $k \times k$ conv. Assume that the kernel shapes of 1×1 and $k \times k$ convolution layers are $D \times C \times 1 \times 1$ and $E \times D \times K \times K$, respectively, where D refers to any value. First, the two BN layers are fused inside the two convolutional layers, resulting in

$$F'_{j,:,:} \leftarrow \frac{\gamma_j}{\sigma_j} F_{j,:,:}, \tag{3}$$

where $F_{j,:,:}$ denotes a slice in the 3D tensor F, j denotes the index of the slice, which denotes the j-th channel in the tensor F, $F'_{j,:,:}$ denotes the corresponding slice in the updated tensor F, in which $\frac{\gamma_j}{\sigma_j}$ denotes the scaling of the slice, and γ_j and σ_j are scalars and used to scale the slice $F_{j,:,:}$

$$b' \leftarrow -\frac{\mu_j \gamma_j}{\sigma_j} + \beta_j, \tag{4}$$

$$F^{(1)} \in R^{D \times C \times 1 \times 1}, b^{(1)} \in R^D, F^{(2)} \in R^{E \times D \times K \times K}, b^{(2)} \in R^E. \tag{5}$$

where $F^{(1)}$ is a four-dimensional tensor representing the weights of the first convolutional layer, which has D output channels and C input channels, $b^{(1)}$ is a vector with D elements representing the bias terms of the first convolutional layer, $F^{(2)}$ is a four-dimensional tensor representing the weights of the second convolutional layer, which has E output channels, D input channels, and a convolution kernel size of $K \times K$, b is a vector with E elements representing the bias terms of the first convolutional layer, and $b^{(2)}$ is a vector with E elements representing the bias terms of the second convolutional layer.

The outputs are

$$O' = \left(I \otimes F^{(1)} + REP\left(b^{(1)}\right)\right) \otimes F^{(2)} + REP\left(b^{(2)}\right). \tag{6}$$

As expected, this is expressed in terms of the kernel and bias of a single convolution. Let F' and b' satisfy the following equation.

$$O' = I \otimes F' + REP\left(b'\right). \tag{7}$$

Applying the additivity of convolution, the following equation is obtained.

$$O' = I \otimes F^{(1)} \otimes F^{(2)} + REP\left(b^{(1)}\right) \otimes F^{(2)} + REP\left(b^{(2)}\right). \tag{8}$$

Conversion 3—Average Pooling Layer Conversion: The average pooling layer operates by sliding the feature map through a sliding window and averaging the elements within the window. Unlike a convolutional layer, a pooling layer is specific to the individual input channels, whereas a convolutional layer sums the results of all input channels. The average pooling layer can be equivalent to a fixed-weight convolutional layer. Assuming that the average pooling layer window size is 3×3, the 3×3 convolutional layer weight can be set to $1/9$.

Conversion 4—Multi-scale convolutional fusion: A $Kh \times Kw$ convolution can be converted into a $K \times K$ convolution by zero padding.

Since the above multi-branch complex structure brings different receptive fields, and different sizes of receptive fields mean different abilities to capture long-range dependencies, the SAE scale-aware enhancement module is accessed after the top-level features of the backbone network, which makes full use of the receptive fields in feature maps by using dilated convolutions. As shown in the backbone of the overall architecture in Figure 1, the SAE module uses three dilated convolution branches with different rates to capture multi-scale information and dependencies of different ranges, while adding residual connections to prevent gradient explosion and vanishing during training. All branches have shared weights, and the only difference is their unique sensory fields.

3.1.3. Lightweight Asymptotic Feature Pyramid Network

As shown in the Neck part of the overall architecture in Figure 1, different levels of low-, middle-, and top-level features are extracted in the bottom-up feature extraction process of the backbone network. For feature fusion, low-level features are first fed into the feature pyramid network, and then high-level features are gradually added. If feature fusion is performed directly, the fusion effect of non-adjacent level features will be poor, because the semantic gap between non-adjacent level features is larger than the semantic gap between adjacent level features, especially the bottom and top features. Therefore, an asymptotic architecture is adopted to first fuse two adjacent low-level features, and then gradually incorporates mid-level features and top-level features into the fusion process. In the process of multi-level feature fusion, ASF is used to assign different spatial weights to features at different levels, which enhances the importance of key levels and alleviates the impact of conflicting information on different targets.

Based on the design idea of lightweight networks, the quantization and shift-based convolution layer variant is introduced to the feature fusion module simultaneously. Lower memory usage and higher computational speed can be easily achieved by decomposing the traditional convolution kernel into two parts: the variable quantization kernel (VQK) and distribution shift. During the implementation process, only integer values are stored in the VQK while maintaining the same output as the original convolution by applying kernel- and channel-based distribution shifts.

3.2. Deployment Strategies Based on Cloud–Fog Computing Architecture

The proposed driver distraction detection method is developed based on a cloud–fog computing architecture. As shown in Figure 3, the cloud–fog computing architecture consists of three layers, the end-user layer, the fog computing layer, and the cloud computing layer.

First, the end-user layer as the bottom layer includes various cameras, sensors and other devices used to monitor driving behavior and collect driving information in real time. These devices continuously generate large amounts of data, such as driver posture, expressions, gestures, vehicle status, road conditions, etc. The collected data are used as input for driver behavior detection. Service computing can support multi-party data sharing and collaboration, which is particularly important in the field of driving detection. Data from different vehicles, drivers and roads can be collected, aggregated and analyzed to improve the effectiveness of the driving detection system.

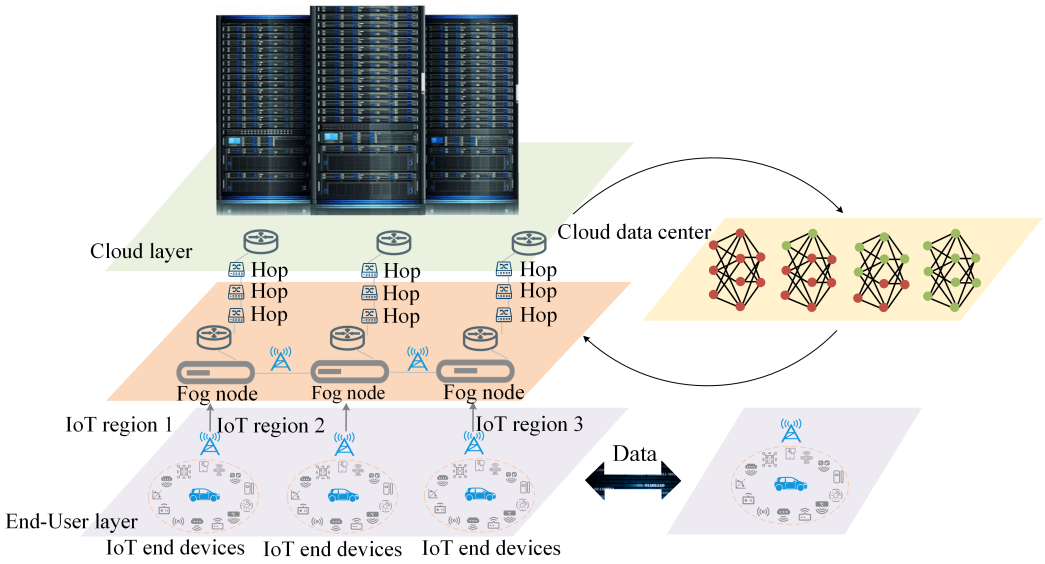

Figure 3. Driving Distraction Detection and Deployment Architecture Diagram Based on Cloud–fog Computing.

As the middle layer, the fog computing layer mainly performs edge computing and is used to perform part of the computional processing locally. Edge computing devices are deployed at the fog computing layer to perform the real-time data processing locally. These devices are responsible for basic data filtering, initial feature extraction, and immediate response to data. This helps to reduce the dependence on cloud resources and improve the real-time performance of the system. When driver distraction behavior information is detected, the information is transmitted to the fog computing layer. The edge devices in the fog computing layer are responsible for the real-time calculation and processing of these data. However, due to the limited computing power and memory resources of edge devices, they are unable to complete complex computing tasks. Therefore, the fog computing layer requires device–cloud collaboration to transfer the model training tasks to the cloud computing layer.

As the top layer, the cloud computing layer has large-scale computing resources and storage capabilities. Therefore, data can be processed in the cloud as a continuous stream by employing streaming processing techniques. When the edge device cannot meet the computing demand, the fog computing layer transfers the training tasks to the data center of the cloud computing layer. At the cloud computing layer, new massive amounts of data are preprocessed and used to train deep learning models. Additionally, the data processing center also needs to have strong computing power to process the data in real time and make timely decisions. The proposed solution considers the integration of cloud computing into existing in-vehicle systems.After training, the optimized model will be deployed on the terminal of the edge computing platform to achieve a real-time detection and response of edge devices. Using service computing, a real-time monitoring system can be deployed in the cloud to continuously monitor the driver's biometrics and driving behavior. Once the system detects signs of possible distracted driving, such as eyes closed for too long or the vehicle veering off the road, the system can issue an alert or reminder to prompt the driver to take action.

In the whole architecture, the bottom layer is the sensor data processing module, the middle layer is the feature extraction and selection module, and the top layer is the decision and response module. Each module is designed to operate independently and communicate with each other through flexible interfaces, thus making the whole system scalable. In order to improve the performance and adaptability of the system to cope with complex and changeable driver distraction behavior detection tasks, two optimization paths, data-driven and model-driven, are introduced. As shown in the right part of Figure 4, new data

are transmitted from the data layer to the fog computing layer and cloud computing layer along the data-driven optimization path, continuously updating the deep learning model. In the data center of the cloud computing layer, these new data are used to further train and optimize the model to improve the accuracy and performance of the model.

Figure 4. Ten categories of driving behaviors in the Statefarm dataset.

Driver behavior and environmental information is constantly changing, and new data may contain previously unobserved situations and patterns. By uploading new data to the cloud computing layer for training, the model can be continuously updated and optimized to adapt to new data, and the data-driven path ensures that the model always adapts to new data. Model updates ensure driver distraction detection systems remain accurate and efficient in changing real-world driving scenarios.

The model-driven optimization path is based on a greedy pruning algorithm with an performance-aware approximation, further reducing the model's computational load and parameters by intelligently adjusting the model structure, thereby achieving acceleration on cloud and edge mobile platforms with less performance degradation. The main goal of this optimization path is to maintain efficient computing and responsiveness using limited edge device resources. Edge devices have limited computing power and memory resources and cannot support complex deep learning models. Through the greedy pruning algorithm, the optimized model can run quickly on edge devices while maintaining a high detection accuracy. The system utilizes elastic compute resources from cloud service providers to ensure that the system automatically scales compute capacity in the face of fluctuations in real-time data. Therefore, this automated elastic computing strategy can provide sufficient resources when needed and effectively reduce computing resource usage when the load is light. In this way, the system can perform real-time driver distraction detection on edge devices, reducing the reliance on cloud computing, lowering latency and network burden, while improving the overall performance and efficiency of the system.

With cloud computing infrastructure, driving distraction detection systems can be more easily scaled to accommodate different regions and sizes. In addition, the system's algorithms and models can be updated in the cloud to continuously improve detection accuracy and efficiency.

Algorithm 1 illustrates the detailed process of updating and optimization using the proposed model-driven path in cloud computing. The algorithm is divided into the following steps: (1) Pruning layer selection: First, according to the reduction of FLOPs of each layer under different pruning rates, the pruning layers are sorted to select the layer that contributes more to model compression as a candidate pruning layer. (2) Sensitivity calculation: For each candidate pruning layer, the sensitivity of each channel is calculated to identify the tasks that are most sensitive to model performance. The sensitivity value indicates how much the channel affects model performance. (3) Task compression and update: In each pruning layer, the most sensitive task that has the greatest impact on model performance is selected based on the sensitivity value. The impact of pruning on the model is controllable by gradually compressing tasks. The compression ratio is updated based on the model calculation amount and the number of parameters after pruning to balance model compression and performance requirements. (4) Compression strategy and fine-tuning

optimization: Select some layers that contribute the most to compression after pruning, and retain those layers that have a greater impact on the model performance to achieve the best balance between model compression and performance. Reset the model and fine-tune the pruned model to restore model performance and maintain high detection accuracy.

Algorithm 1 The proposed framework.

Input: The pretrained model M_0, dataset D, reserved ratio Γ of FLOPs or parameters, performance drop threshold α, initial drop threshold d_1, masking ratio γ of filters, and filtering ratio P of pruning layers.

Output: The compressed model M_p satisfying the compression requirements Γ.

initialize $p \leftarrow 1$, $M_p \leftarrow M_0$;
while $p > \Gamma$ **do**
  reorder the pruning layer sequence according to the FLOPs reduction at a certain pruning ratio of each single layer $\leftarrow l_F$;
  compute λ;
  for i, l in enumerate(l_F) **do**
    for $j = 1 : K_i$) **do**
      compute S_i^j;
      mask γ filters with lowest value in S_l and find the most sensitive task t_l;
      prune filters with lowest value in S_l until the relative performance drop of Task t_l oversteps d_i;
      $R_l \leftarrow$ ratio of pruned filters layer l;
      $M_p \leftarrow$ ratio of pruned filters layer l;
      $d_{i+1} \leftarrow \lambda d_i$;
      sort the compression contribution C of each layer at the ratio of $\boldsymbol{R} = \{R_1, \ldots, R_L\}$ and select top P layers as the pruning layer set $\leftarrow L_n$;
      reset model $M_p \leftarrow M_0$;
    end for
  end for
  for l *in* L_F **do**
    if l *in* L_p **then**
      prune filters of M_p in S_l with the ratio of $R_l \rightarrow M_p$;
    end if
  end for
  $M_0 \leftarrow$ fine-tuned M_p;
  update p;
end while
return M_p

The channel importance s_d score measures the contribution of each output channel to the performance of the model and is calculated based on the Euclidean norm (L2 norm) of each channel in the pruned weight matrix $\mathbf{F}^{(1)}$. The higher channel's importance score means the greater channel's impact on model performance.

$$s_d = \frac{\left\| \mathbf{F}^{(1)}[:,;,;,d] \right\|_2}{\sqrt{D}}. \tag{9}$$

where $\mathbf{F}^{(1)}[:,:,:,d]$ denotes the subset of the weight matrix $\mathbf{F}^{(1)}$ corresponding to the d-th output channel after pruning. Here, $\| \cdot \|_2$ represents the L2 norm (Euclidean norm), and D is the total number of output channels.

Performance evaluation criteria $\left(P\left(\mathbf{L}^{(0)}, \mathbf{L}^{(1)}\right)\right)$ are employed to assess the model's performance both before and after the pruning process. This criterion is derived from the multi-task loss function of the model prior to pruning $\left(\mathbf{L}^{(0)}\right)$ and post-pruning $\left(\mathbf{L}^{(1)}\right)$. A

value closer to 1 indicates that the performance after pruning closely resembles that before pruning, suggesting a minimal performance loss.

$$P\left(\mathbf{L}^{(0)}, \mathbf{L}^{(1)}\right) = \frac{\mathbf{L}^{(0)}}{\mathbf{L}^{(1)}} \tag{10}$$

where $\left(\mathbf{L}^{(0)}\right)$ is the multitask loss of the original model and $\left(\mathbf{L}^{(1)}\right)$ is the multi-task loss of the pruning model. The use of these indicators helps to achieve a balance between performance and model compression during pruning.

Algorithm input parameters include M_0: the pre-trained model; D: the dataset used to train and evaluate the model; τ: the reserved FLOPs or parameter ratios used to control the degree of compression; α: the performance degradation threshold used to control the performance loss of compression; d_i: the initial performance degradation threshold used to guide the pruning process; γ: the filter masking ratio used to determine the filter that needs to be pruned; and P: the filtering ratio of the pruning layer, specifying which layer needs to be pruned. The output of the algorithm is a model that has been compressed to meet the compression requirements.

Algorithm 1 describes the model-driven optimization process based on greedy pruning on cloud computing architecture. First, during the initialization phase, variables p and M_0 are set to their initial values. Then, in each iteration, the order of the pruned layers is rearranged based on the FLOP reduction of each individual layer at a specific pruning rate. This ranking reflects the relative importance of each layer in the model computation. Subsequently, the parameter p is calculated to guide the subsequent pruning process. Next, for each index i and layer l in the set of pruned layers lF, conduct the following operations in a loop: iteratively calculate parameters in the range from 1 to K_i to determine the convolution kernel for pruning. Then, find the mask with the lowest value in the convolution kernel and identify the most sensitive task. The pruning operation continues until the relative performance of the task drops beyond a preset threshold. Next, the pruning rate in pruning layer l is calculated and the parameters are updated. Then, the pruned layers are ordered by applying ratio = $R_1, \ldots, R_L$ to the compression contribution C of each layer, and the top layer is selected as the set of pruned layers. After the set of pruning layers is determined, the model is reset and prepared for fine-tuning operations. Finally, for each layer l in the pruned layer, set l_F, and check whether it is in the pruned layer set l_p. If so, the layer's filter is pruned using the specified ratio. Task sensitivity is introduced at each pruning step. It uses the post pruning structure of the model from the previous step and a vector of the multitask loss stated from the previous step after pruning. This vector contains information about each term in the loss function, allowing the compression process of the current layer to be implemented on a task-sensitivity-aware basis, which thereby maintains control over the model performance. Then, perform fine-tuning operations to optimize the model performance. Finally, after a series of iterations and pruning operations, a compression model that meets the set compression ratio and performance requirements is returned. The algorithm obtains an approximation of the optimization objective by analyzing the problem to achieve an effective channel pruning without introducing a regular penalty term.

4. Experiments

4.1. Datasets

To demonstrate the effectiveness of Progressive Scalable Network (PSDNet) and cloud computing architecture, experiments were conducted on the non-public Lilong Distracted Driving Behavior (LDDB) dataset and a public competition dataset (Statefarm [28]). The obtained results are compared with those on ResNet-50 [7], DenseNet-40 [29], ShuffleNet-v2 [9], MobileNet-V2 [8] and MobileNet-V3 [30]. Various module ablation studies on PSDNet are reported to elucidate the impact of various design decisions.

The LDDB dataset was created through a series of road experiments. The dataset contains 14,808 videos collected from infrared cameras, covering 6 driving behaviors of

2468 participants, namely talking on the phone (c0), safe driving (c1), sleepy driving (c2), smoking (c3), turning head (c4) and yawning (c5). The videos were manually annotated at five frames per second, resulting in a total of 287,808 images. For LDDB dataset, 70%, 10% and 20% of such dataset are considered as training, validation and testing data.

As shown in Figure 4, the StateFarm dataset contains ten categories, namely safe driving (c0), texting—right (c1), talking on the phone—right (c2), texting—left (c3), talking on the phone—left (c4), operating the radio (c5), drinking (c6), reaching behind (c7), doing hair and makeup (c8) and talking to the passenger(s) (c9). The specified 22,424 images and 67,272 images processed with offline data augmentation (i.e., Gaussian blur, Gaussian noise, and CutMix [31]) were used for training. Since the 79,726 images officially provided for testing were not labeled, approximately 1000 images in each category of images used for network performance evaluation were labeled in this study.

4.2. Experiment Details

Loss function: The loss function of the proposed driving distraction detection method employs multiple components to perform object detection and bounding box regression tasks. Among them, the object detection loss uses cross-entropy loss, which is used to measure the difference between the predicted target category and the actual category, expressed as:

$$L_{\text{cls}} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) + (1-y_{i,c})\log(1-\hat{y}_{i,c}), \tag{11}$$

where N is the number of bounding boxes, C is the number of categories, y is the actual category label, and $\hat{y}$ is the category confidence predicted by the network. Additionally, the mean square error loss is used to measure the error between the predicted and actual positions of the bounding box, expressed as:

$$L_{\text{box}} = \sum_{i=1}^{N}\sum_{k\in\{x,y,w,h\}} (y_{i,k}-\hat{y}_{i,k})^2, \tag{12}$$

where y is the actual bounding box position and $\hat{y}$ is the bounding box position predicted by the network. Finally, the total loss function consists of the object detection loss and the bounding box regression loss, which is obtained by weighted summation:

$$L_{\text{total}} = L_{\text{cls}} + \lambda L_{\text{box}}, \tag{13}$$

where λ is a hyperparameter used to balance the two loss terms. By optimizing this loss function, efficient and accurate driving distraction detection tasks can be achieved.

Evaluation indicators: In order to evaluate the quality of the detection results, this paper uses commonly used evaluation indicators such as mAP@0.5, mAP@0.5:0.95, calculation amount, number of parameters, FPS, etc.

mAP@0.5: This is the mean average precision (mAP) value when IoU threshold is 0.5. It measures the model's detection accuracy at a 0.5 IoU threshold.

mAP@0.5:0.95: This is the mean average precision (mAP) value within the IoU threshold range of 0.5 to 0.95. It combines detection accuracy at different IoU thresholds to more comprehensively evaluate the performance of the model.

Calculation amount: This refers to the computing resources required for model inference or training, usually measured in FLOPs (Floating Point Operations) or GFLOPs (Gigafloating Point Operations).

Parameters: This is the total number of parameters to be learned in the model, including weights and biases. The number of parameters is usually in units of millions (Million, M) or billions (Billion, B).

Frames Per Second (FPS): This refers to the number of image frames processed by the model per second during the inference phase, also known as the inference speed or

processing speed. A higher FPS means the model is more computationally efficient in real-time applications.

Experimental environment: In order to simulate the cloud environment and verify the proposed driving behavior image processing method, this paper uses the cloud computing strategy to conduct experiments. High-performance private cloud servers with multiple GPUs are adopted. These servers are equipped with NVIDIA TITAN RTX and NVIDIA GeForce RTX 4090, respectively. The model was built, trained, and tested using the Pytorch framework. NVIDIA Jetson TX2 components are used as edge computing fog nodes to verify the effectiveness of the proposed architecture. NVIDIA Jetson TX2 is a high-performance embedded computing module launched by NVIDIA. It uses a Pascal architecture GPU, equipped with a 64-bit ARM Cortex-A57 central processor and a dual-core Denver 2.0 processor, and 8GB LPDDR4 memory.

Hyperparameter settings: The hyperparameters used are given as follows. The optimizer used was the stochastic gradient descent; To balance the training speed and model performance, 50 epochs were chosen for training to ensure that the model learns with enough iterations on the dataset; the batch size was set to 32; a linear decay learning rate scheduling strategy was implemented with an initial learning rate of 0.01 and a recurrent learning rate of the same value; the momentum and weight decay values were set to 0.937 and 0.0005, respectively. In addition, for the activation function of the proposed model, the SiLU (Sigmoid Linear Unit) activation function was used, which is a nonlinear activation function with good performance and convergence speed.

Dataset preprocessing: Data expansion includes random rotation with 0.5 probability, random resize cropping and horizontal flipping, and random erasure with 0.2 probability. Dropout rate (0.2) and label smoothing (0.1) were used to avoid overfitting.

Visualization techniques: GradCAM [32], GradCAM++ [33] and XGradCAM [34] were used, all of which are heatmap visualization techniques for visualizing the decision-making process of deep learning models. They generate heatmaps showing the most influential regions in the input image for model-specific predictions.

Among them, GradCAM is a technology for visualizing key areas in the decision-making process of deep learning models. It does this by using the gradient of the object category score with respect to the feature map of the last convolutional layer of the network. The gradient shows how sensitive the output is to changes in the feature map. By combining gradients with feature maps, GradCAM generates a heatmap showing the regions in the input image that contribute most to the final prediction.

GradCAM++ is an extension of the original GradCAM method. It is designed to improve the positioning accuracy of heatmaps generated by GradCAM. GradCAM++ solves this problem by taking into account both positive and negative gradients when calculating importance scores. This additional consideration helps to obtain clearer and more focused heatmaps, thereby improving the localization of important regions in the input image.

XGradCAM extends GradCAM by combining gradient information from multiple network layers to achieve more comprehensive image understanding. The gradients of different layers capture different features of the image, and by combining these gradients, XGradCAM generates richer, more detailed heatmaps. These technologies play an important role in deep learning model interpretation, helping researchers and practitioners understand the decision-making basis of the model, enhancing the interpretability of the model and providing support for the model debugging and optimization. They are also widely used for visual analysis, plausibility building and a deeper understanding of model behavior.

4.3. Comparative Experiments

To verify the performance of the proposed progressively scalable network for driving distraction detection, this paper compared it with several excellent convolutional networks, including ResNet-50 and DenseNet-40 based on traditional networks, ShuffleNet-V2 based on lightweight networks and two different-scale versions of MobileNetV3, GhostNet and FasterNet. To qualitatively analyze the experimental results, all experiments were

performed on a unified benchmark and evaluated using consistent metrics. The evaluation results of these different methods are shown in Table 1. The best performance values are highlighted in bold.

Table 1. Evaluation Indicators for LDDB Dataset Comparison Experiments.

Models	mAP	Param (M)	GFLOPs	Latency (PC)	Latency (TX2)
ResNet-50 [7]	0.935	19.35	18.58	15.2 ± 1.2	118.5 ± 2.2
ShuffleNet-V2 [29]	0.914	4.02	0.64	8.0 ± 0.9	47.1 ± 6.1
MobileNetV3-Large [30]	0.929	4.23	0.30	5.6 ± 0.7	38.4 ± 7.9
MobileNetV3-Small [30]	0.901	2.21	**0.09**	4.7 ± 0.8	33.5 ± 7.3
OLCMNet [35]	0.959	2.78	0.68	**4.7 ± 0.5**	**32.8 ± 4.6**
GhostNet [36]	0.948	4.75	7.6	9.3 ± 1.1	-
FasterNet [19]	0.946	5.55	11.2	13.5 ± 0.2	-
PSDNet	**0.964**	**1.33**	3.2	9.9 ± 0.5	52.5 ± 6.2

Figure 5 shows the corresponding results for a more accurate comparison. Figure 5a–c depicts the performance of various detection methods in terms of accuracy, weight and speed. In each case, the best performing method is marked with an asterisk on the corresponding column. The proposed scalable network PSDNet shows an excellent performance in terms of detection accuracy on the LDDB dataset, with an average accuracy of 96.4%. Furthermore, the number of parameters and computational complexity of this model are comparable to other lightweight networks with 1.33 M parameters and 3.2 GFLOPS computation.

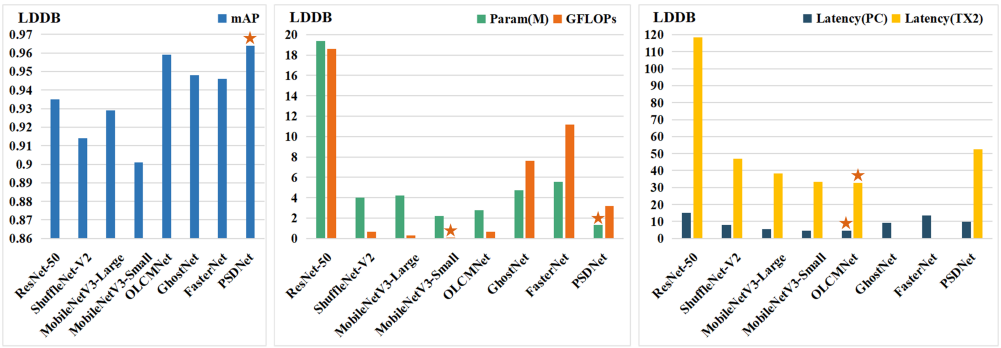

Figure 5. Indicator Comparison of LDDB Dataset Experiments.

As shown in the results in Table 1 and Figure 5, this method has a superior performance compared with other methods. On the LDDB benchmark, the method achieves a higher accuracy than a range of traditional lightweight networks, while model weights and times remain at the level of lightweight networks. This detection method shows a good balance between speed and accuracy. The method performs well across all categories of distracted driving behavior, showing significant improvements in detection accuracy and model optimization calculations.

Figure 6 shows the use of three different visualization techniques, GradCAMPlusPlus, GradCAM and XGradCAM, on the LDDB dataset to understand the decision-making process of the deep learning model in the image classification task, and to visually reveal the image regions on which the model is predicted for different categories.

First, GradCAM is applied, which analyzes gradient information to locate the most influential regions in the image. In this case, to identify phone calling behavior, the GradCAM heatmap clearly shows the important areas related to the mobile phone area, which provides key visual clues.

Then GradCAMPlusPlus as an extended technique is used, which takes into account both positive and negative values of gradients, resulting in more refined heatmaps. When identifying head-turning behavior, GradCAMPlusPlus highlights areas related to the di-

rection of head rotation, resulting in a more accurate representation of the model's focus of attention.

Figure 6. Heatmap Visualization Results on LDDB Dataset for Comparative Testing.

Not only that, XGradCAM further enhances interpretation capabilities. It combines gradient information from multiple network layers to capture visual features at different abstraction levels. To detect yawning behavior, XGradCAM heatmaps highlight regions related to oral morphology, revealing how the model makes decisions through multiple levels of feature extraction. By explicitly displaying the image areas of interest to the model, the model's interpretability is improved while also providing a powerful tool for the driving behavior analysis to ensure road safety and driving behavior monitoring.

Table 2 shows the comparison of the proposed PSDNet network with other networks on the StateFarm dataset. The optimal values have been presented in bold, where the MAP column is labeled with the highest accuracy. Notably, we have further emphasized the minimum number of parameters and computations, which highlights the degree of lightness of the model. Additionally, we have bolded the lowest latency, which is a direct reflection of the high speed of the network when it comes to actual inference. These bolded values not only highlight the model's superior performance in terms of accuracy, but also its outstanding properties in terms of lightweight and efficient reasoning. The results show that the proposed PSDNet exhibits the same advantages in terms of accuracy and computational complexity on the StateFarm dataset, and has an excellent detection speed on both cloud platforms and edge devices. The corresponding indicators of accuracy, model parameters with computation and detection speed are shown in Figure 7.

When exploring and understanding distracted driving behavior, an interpretable perspective is critical to gain insight into the basis of model decisions. As shown in Figure 8, by applying three visualization techniques, GradCAMPlusPlus, GradCAM and XGradCAM, we can highlight the interpretability of different distracted behaviors (e.g., safe driving, texting, talking on the phone, etc.) on the State Farm Driving Distraction Behavior Dataset.

Table 2. Evaluation Indicators for Statefarm Dataset Comparison Experiments.

Models	mAP	Param (M)	GFLOPs	Latency (PC)	Latency (TX2)
ResNet-50 [7]	0.895	19.35	18.58	15.2 ± 1.2	118.5 ± 2.2
ShuffleNet-V2 [29]	0.836	4.02	0.64	8.0 ± 0.9	47.1 ± 6.1
MobileNetV3-Large [30]	0.848	4.23	0.30	5.6 ± 0.7	38.4 ± 7.9
MobileNetV3-Small [30]	0.793	2.21	**0.09**	4.7 ± 0.8	33.5 ± 7.3
OLCMNet [35]	0.895	2.78	0.68	**4.7 ± 0.5**	**32.8 ± 4.6**
GhostNet [36]	0.847	4.75	7.6	9.3 ± 1.1	-
FasterNet [19]	0.899	5.55	11.2	13.5 ± 0.2	-
PSDNet	**0.917**	**1.33**	3.2	9.9 ± 0.5	52.5 ± 6.2

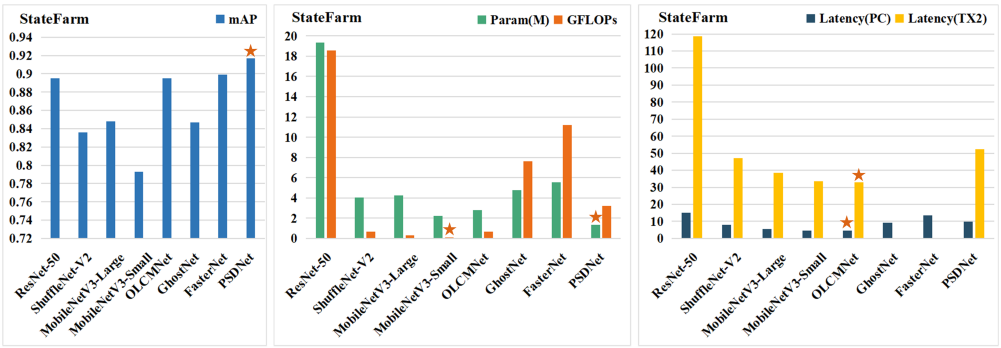

Figure 7. Indicator Comparison of Statefarm Dataset Experiments.

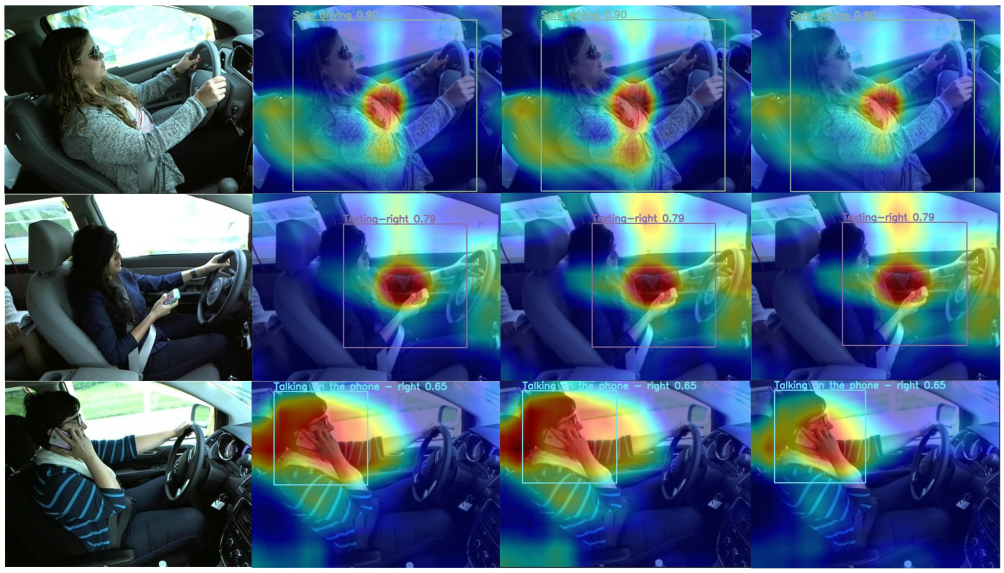

Figure 8. Heatmap Visualization Results on Statefarm Dataset for Comparative Testing.

First, GradCAM is applied, which allows the key features of each distracting behavior to be presented from a model perspective. For example, when the model identifies safe driving behaviors, GradCAM heatmaps highlight areas associated with normal driving operations, such as vehicle consoles, hand movements, facial features, and more. This visualization approach allows for a better understanding of how the model links these specific visual features to safe driving behavior.

Furthermore, using GradCAMPlusPlus, the approach is able to achieve more nuanced interpretability. By taking into account both positive and negative values of the gradient, GradCAMPlusPlus generates more accurate and detailed heatmaps. For example, when analyzing texting behavior, a GradCAMPlusPlus heatmap might highlight areas associated with hand and mobile phone interactions, revealing key details that the model focuses on during classification.

Finally, with XGradCAM, we are able to obtain more comprehensive informationfrom features at multiple levels of abstraction. By combining gradients from multiple network layers, XGradCAM heatmaps highlight areas associated with features at different levels. For example, for the behavior of talking on the phone, an XGradCAM heatmap might highlight mouth areas associated with talking and areas associated with the vehicle console, providing a more comprehensive visual interpretation.

4.4. Ablation Study

To further verify the effectiveness of each component of the proposed method module, such as the multi-branch scalable sensing backbone, lightweight asymptotic feature pyramid and model-driven optimization based on a greedy pruning algorithm, an ablation

study is performed on the LDDB dataset. This study aims to evaluate the impact of these components on the overall performance.

CSPDarknet53+PAFPN serves as the baseline model. Different components are gradually added to evaluate their effectiveness. The performance of the following four detection models is compared.

Baseline: CSPDarknet53+PAFPN is used as the baseline model.

Baseline+A (MSS Backbone): A multi-branch scalable perception backbone is used as the backbone network of the driver distraction behavior detector architecture to enhance the feature extraction capabilities of the backbone network with the smallest possible increase in complexity and number of parameters.

Baseline+A (MSS Backbone)+B (LaFPN): The lightweight asymptotic feature pyramid on top of the multi-branch scalable perceptual backbone is further enhanced to improve feature fusion while minimizing the increase in the number of model parameters and computational complexity.

Baseline+A (MSS Backbone)+B (LaFPN)+C (Model-Driven Optimization based on Greedy Pruning Algorithm): The model-driven optimization based on the greedy pruning algorithm on top of the above model in introduced to achieve a better trade-off between accuracy and speed on cloud platforms and edge devices.

The performance of Baseline and Baseline+A is shown in Table 3 to demonstrate the capabilities of multi-branch scalable backbone networks incorporated into the model. Comparing Baseline+A with Baseline+A+B confirms that the introduction of lightweight progressive feature pyramid brings significant performance gains and a slight increase in model weights. Subsequent comparisons show that model-driven optimization based on the greedy detection algorithm brings significant improvements in model calculation volume, computational complexity, and detection latency on cloud platforms and edge devices, with minimal performance loss, proving that it achieves an excellent balance between speed and accuracy.

Table 3. Evaluation Indicators for Ablation Experiments on LDDB dataset.

Models	AP50	AP	Param (M)	GFLOPs	Latency (ms)	FPS	Size (M)
Baseline	0.951	0.771	1.76	4.2	24.4	40.98	3.73
+ A	0.954	0.775	1.84	4.2	29.3	34.09	4.34
+ A + B	**0.964**	**0.794**	2.16	5.6	23.7	42.26	6.44
+ A + B + C	0.960	0.788	**1.33**	**3.2**	**9.9**	**84.98**	**2.87**

Table 3 shows the detection performance of various models. The optimal values have been presented in bold, where the MAP column is labeled with the highest accuracy. Notably, we have further emphasized the minimum number of parameters and computations, which highlights the degree of lightness of the model. Additionally, we have bolded the lowest latency, which is a direct reflection of the high speed of the network when it comes to actual inference. These bolded values not only highlight the model's superior performance in terms of accuracy, but also its outstanding properties in terms of lightweight and efficient reasoning. It is evident from the data in the table that each of the above components contributes to an improved detection. It is pointed out that individual modules in the scalable network aim to improve detection accuracy, and model-driven optimization aims to improve detection efficiency and speed. Figure 9a,b shows the performance of various ablation experiment steps in terms of accuracy, weight and speed. In each case, the best performing method is marked with an asterisk on the corresponding column. The proposed scalable network PSDNet ranks first in both accuracy indicators, with AP50 and AP of 0.964 and 0.794, respectively. After model-driven optimization of the model based on the greedy pruning algorithm, the number of model parameters is reduced by 38.42%, the computational complexity is reduced by 42.85% and the accuracy loss is only 0.004%. Additionally, the optimized model doubles in latency, FPS and model size, indicating that a better balance between model accuracy and speed is achieved.

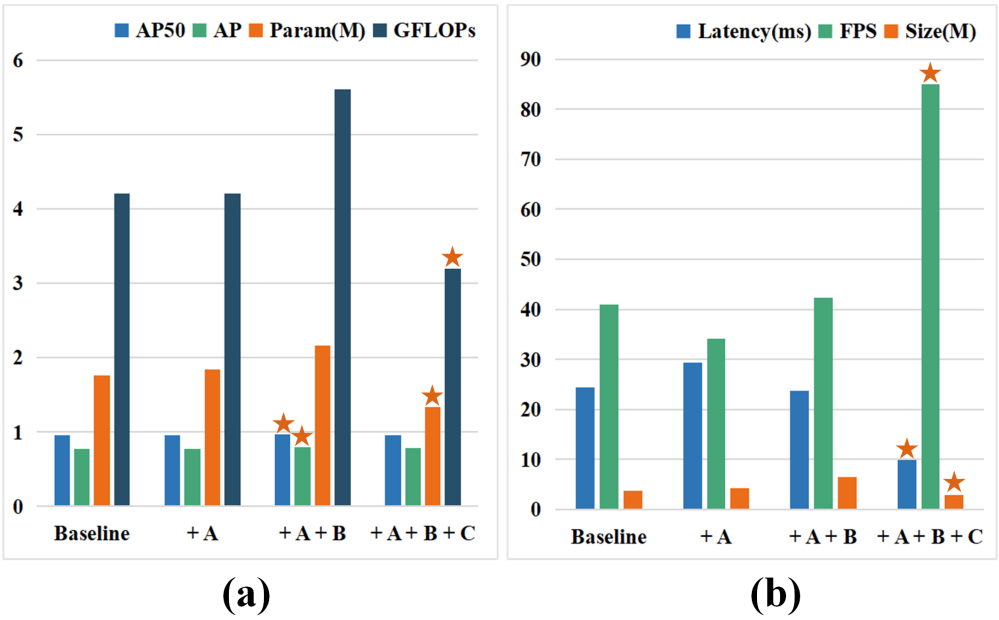

Figure 9. Indicator Comparison for Ablation Experiments on Statefarm Dataset

5. Conclusions

In this study, we propose a cloud computing-based approach for driving distraction behavior detection by taking advantage of cloud computing architecture, including a multi-branch scalable perceptual backbone network and a lightweight asymptotic feature pyramid. We employ a model-driven optimization path based on greedy pruning to accelerate the inference and computation process. Experimental results show that our approach achieves excellent performance on multiple driving distraction behavior image datasets, confirming the superiority of the scalable network component. Future research directions will focus on two main areas: lightweight model design and acceleration for real-world deployment. First, the exploration of further lightweight network architectures to meet the real-time performance requirements of embedded devices and mobile-based platforms continues. Second, we will work on applying our findings to real-world scenarios and further optimizing the performance in cloud and fog computing environments. We plan to conduct more field tests to verify the performance under different hardware devices and network conditions to provide a more reliable and efficient solution for real-time driving behavior detection.

Author Contributions: Conceptualization, S.W.; Methodology, X.H. (Xueda Huang); Validation, Z.Z.; Formal analysis, G.Q.; Resources, Y.L.; Data curation, L.S.; Visualization, B.W.; Supervision, S.C.; Project administration, X.H. (Xin Huang). All authors have read and agreed to the published version of the manuscript.

Funding: This research is jointed sponsored by National Natural Science Foundation of China (82205049, 62276037), Natural Science Foundation of Chongqing (cstc2020jcyj-msxmX0259, cstc2021jcyj-bshX0064), and Special key project of Chongqing technology innovation and application development: CSTB2022TIAD-KPX0039, Basic Research and Frontier Exploration Project of Yuzhong District, Chongqing, Grant/Award Number: 20210164.

Data Availability Statement: The data presented in this study are available in the article.

Conflicts of Interest: Authors Shiyao Chen and Xin Huang were employed by the company Chongqing Dima Industrial Co., Ltd., Chongqing, China. Author Linhong Shuai was employed by the company Chongqing LiLong Zhongbao Intelligent Technology Co., Chongqing, China. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Chhabra, R.; Verma, S.; Krishna, C.R. A survey on driver behavior detection techniques for intelligent transportation systems. In Proceedings of the 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence, Noida, India, 12–13 January 2017 ; pp. 36–41.
2. Kim, W.; Choi, H.K.; Jang, B.T.; Lim, J. Driver distraction detection using single convolutional neural network. In Proceedings of the 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 18–20 October 2017; pp. 1203–1205.
3. Qi, G.; Wang, H.; Haner, M.; Weng, C.; Chen, S.; Zhu, Z. Convolutional neural network based detection and judgement of environmental obstacle in vehicle operation. *CAAI Trans. Intell. Technol.* **2019**, *4*, 80–91. [CrossRef]
4. Zhu, Z.; Wang, S.; Gu, S.; Li, Y.; Li, J.; Shuai, L.; Qi, G. Driver distraction detection based on lightweight networks and tiny object detection. *Math. Biosci. Eng.* **2023**, *20*, 18248–18266. [CrossRef]
5. Peng, Y.; Wang, Y. Real-time forest smoke detection using hand-designed features and deep learning. *Comput. Electron. Agric.* **2019**, *167*, 105029. [CrossRef]
6. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**. arXiv:1409.1556
7. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
8. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
9. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 11–18 October 2018; pp. 116–131.
10. Zhang, Z.; Zhang, H.; Wang, Y.; Liu, T.; He, Y.; Tian, Y. Underwater Sea Cucumber Target Detection Based on Edge-Enhanced Scaling YOLOv4. *J. Beijing Inst. Technol.* **2023**, *32*, 328–340.
11. Li, Y.; Xu, P.; Zhu, Z.; Huang, X.; Qi, G. Real-time driver distraction detection using lightweight convolution neural network with cheap multi-scale features fusion block. In Proceedings of the 2021 Chinese Intelligent Systems Conference, Fuzhou, China, 16–17 October 2021; Springer: Berlin/Heidelberg, 2022; Volume II, pp. 232–240.
12. Jadeja, Y.; Modi, K. Cloud computing-concepts, architecture and challenges. In Proceedings of the 2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET), Nagercoil, India, 21–22 March 2012; pp. 877–880.
13. Murthy, C.V.B.; Shri, M.L.; Kadry, S.; Lim, S. Blockchain based cloud computing: Architecture and research challenges. *IEEE Access* **2020**, *8*, 205190–205205. [CrossRef]
14. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2736–2744.
15. Ye, H.; Zhang, B.; Chen, T.; Fan, J.; Wang, B. Performance-aware Approximation of Global Channel Pruning for Multitask CNNs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**. [CrossRef] [PubMed]
16. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713.
17. Nascimento, M.G.d.; Fawcett, R.; Prisacariu, V.A. Dsconv: Efficient convolution operator. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 5148–5157.
18. Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; Le, Q.V. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 2820–2828.
19. Chen, J.; Kao, S.h.; He, H.; Zhuo, W.; Wen, S.; Lee, C.H.; Chan, S.H.G. Run, Don't Walk: Chasing Higher FLOPS for Faster Neural Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 17–24 June 2023; pp. 12021–12031.
20. Qi, G.; Zhang, Y.; Wang, K.; Mazur, N.; Liu, Y.; Malaviya, D. Small object detection method based on adaptive spatial parallel convolution and fast multi-scale fusion. *Remote Sens.* **2022**, *14*, 420. [CrossRef]
21. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2818–2826.
22. Ghiasi, G.; Lin, T.Y.; Le, Q.V. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 7036–7045.
23. Ding, X.; Zhang, X.; Han, J.; Ding, G. Diverse branch block: Building a convolution as an inception-like unit. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA , 20–25 June 2021; pp. 10886–10895.
24. Yang, G.; Lei, J.; Zhu, Z.; Cheng, S.; Feng, Z.; Liang, R. AFPN: Asymptotic Feature Pyramid Network for Object Detection. *arXiv* **2023**. arXiv:2306.15988
25. Yan, C.; Sheng, S. Research on behavior element decision of cloud edge collaboration capability based on 5g automatic driving scene. In Proceedings of the 2021 IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Xi'an, China, 15–17 October 2021; Volume 5; pp. 1787–1790.

26. Xun, Y.; Qin, J.; Liu, J. Deep learning enhanced driving behavior evaluation based on vehicle-edge-cloud architecture. *IEEE Trans. Veh. Technol.* **2021**, *70*, 6172–6177. [CrossRef]
27. Khan, M.A.; Nawaz, T.; Khan, U.S.; Hamza, A.; Rashid, N. IoT-Based Non-Intrusive Automated Driver Drowsiness Monitoring Framework for Logistics and Public Transport Applications to Enhance Road Safety. *IEEE Access* **2023**, *11*, 14385–14397. [CrossRef]
28. Abouelnaga, Y.; Eraqi, H.M.; Moustafa, M.N. Real-time distracted driver posture classification. *arXiv* **2017**. arXiv:1706.09498
29. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
30. Howard, A.; Sandler, M.; Chu, G.; Chen, L.C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. Searching for mobilenetv3. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October 2019–2 November 2019; pp. 1314–1324.
31. Yun, S.; Han, D.; Oh, S.J.; Chun, S.; Choe, J.; Yoo, Y. Cutmix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October 2019–2 November 2019; pp. 6023–6032.
32. Selvaraju, R.R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 618–626.
33. Chattopadhay, A.; Sarkar, A.; Howlader, P.; Balasubramanian, V.N. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 12–15 March 2018; pp. 839–847.
34. Fu, R.; Hu, Q.; Dong, X.; Guo, Y.; Gao, Y.; Li, B. Axiom-based grad-cam: Towards accurate visualization and explanation of cnns. *arXiv* **2020**. arXiv:2008.02312
35. Li, P.; Yang, Y.; Grosu, R.; Wang, G.; Li, R.; Wu, Y.; Huang, Z. Driver distraction detection using octave-like convolutional neural network. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 8823–8833. [CrossRef]
36. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. Ghostnet: More features from cheap operations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1580–1589.