electronics

MDPI

Article

Boosting the Response of Object Detection and Steering Angle Prediction for Self-Driving Control

Bao Rong Chang [1], Hsiu-Fen Tsai [2,*] and Fu-Yang Chang [1]

[1] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 81148, Taiwan; brchang@nuk.edu.tw (B.R.C.); m1115509@mail.nuk.edu.tw (F.-Y.C.)
[2] Department of Fragrance and Cosmetic Science, Kaohsiung Medical University, Kaohsiung 80708, Taiwan
* Correspondence: sftsai@kmu.edu.tw

Abstract: Our previous work introduced the LW-YOLOv4-tiny and the LW-ResNet18 models by replacing traditional convolution with the Ghost Conv to achieve rapid object detection and steering angle prediction, respectively. However, the entire object detection and steering angle prediction process has encountered a speed limit problem. Therefore, this study aims to significantly speed up the object detection and the steering angle prediction simultaneously. This paper proposes the GhostBottleneck approach to speed the frame rate of feature extraction and add the SElayer method to maintain the existing precision of object detection, which constructs an enhanced object detection model abbreviated as LWGSE-YOLOv4-tiny. In addition, this paper also conducted depthwise separable convolution to simplify the Ghost Conv as depthwise separable and ghost convolution, which constructs an improved steering angle prediction model abbreviated as LWDSG-ResNet18 that can considerably speed up the prediction and slightly increase image recognition accuracy. Compared with our previous work, the proposed approach shows that the GhostBottleneck module can significantly boost the frame rate of feature extraction by 9.98%, and SElayer can upgrade the precision of object detection slightly by 0.41%. Moreover, depthwise separable and ghost convolution can considerably boost prediction speed by 20.55% and increase image recognition accuracy by 2.05%.

Keywords: ghostbottleneck; SElayer; object detection; LWGSE-YOLOv4-tiny; steering angle prediction; LWDSG-ResNet18

Citation: Chang, B.R.; Tsai, H.-F.; Chang, F.-Y. Boosting the Response of Object Detection and Steering Angle Prediction for Self-Driving Control. *Electronics* **2023**, *12*, 4281. https://doi.org/10.3390/electronics12204281

Academic Editors: Hyeonjoon Moon and Lien Minh Dang

Received: 9 September 2023
Revised: 11 October 2023
Accepted: 13 October 2023
Published: 16 October 2023

1. Introduction

In recent years, developing control systems for self-driving cars has become a critical research project in artificial intelligence (AI). Numerous cars moving on the roads simultaneously is a complex system for vehicles that is difficult to control. Several studies exploit vision sensing techniques for effective self-driving car control systems as emerging developments, for example, Tesla. Therefore, this study aims to significantly improve real-time object detection and image recognition using novel visual algorithms to boost the response of self-driving control in order to minimize the risk of wrong steering judgment as much as possible.

Many AI visual algorithms can rapidly fuse object detection and steering angle prediction to real-time response to the exact steering control for safe self-driving. A notable model, YOLOv4-tiny [1], derived from the original YOLOv4 [2], can achieve object detection properly due to its lightweight architecture. This visual algorithm can rapidly detect vehicles and traffic signs before a car. On the other hand, a remarkable model, ResNet18 [3], a variant of ResNet models [4], can predict the steering angle precisely at the lane intersection or multi-lanes of a road. If a self-driving system cannot respond to identified objects early enough to correctly maintain driving control, it could induce the risk of severe car accidents. Even though our previous work [5] proposed the lightweight version of models LW-YOLOv4-tiny and LW-ResNet18 implemented with rapid object detection and steering

angle prediction to respond to self-driving control quickly, the entire object detection and steering angle prediction process has encountered a speed limit problem. In other words, the method proposed in the last paper cannot achieve a response time as short as possible for self-driving control. The delayed steering angle prediction could cause the risk of severe car accidents. Thus, the speed is more important than precision in this case. Accordingly, we are looking for an improvement to speed up the execution speed and enhance the image recognition accuracy concurrently.

Therefore, this study aims to propose a further speed-up approach for improving LW-YOLOv4-tiny and LW-ResNet18 models where a modified GhostBottleneck [6] and SElayer [7] replace CSP_Block (called LWGSE-YOLOv4-tiny). This can reduce the computation loading and maintain high detection precision where CSP_Block makes the input and output of feature maps the same size so that the model can learn more features. A simplified depthwise separable [8] and ghost convolution is substituted for ghost convolution (LWDSG-ResNet18), which can shorten the computational burden and maintain high prediction accuracy. In such a way, the proposed approach can significantly reduce the model inference time to speed up the entire process of self-driving control, which is our main contribution to this study. According to our previous work [5], a model car called Nvidia JetRacer [9] goes around a planar road map to imitate a self-driving vehicle driving around an urban area autonomously. Meanwhile, this study also introduces several YOLVO-related object detection models for checking the response time of self-driving control. Similarly, this paper also conducted several RestNet-related steering angle models in the test. The experiment will deliver the performance evaluation among them and check which one can best boost the response of self-driving control.

2. Related Work

2.1. Literature Review

Techniques for developing advanced visual algorithms for rapid object detection are outlined in the following papers. According to a remote-controlled car, Karni et al. [10] constructed a small-sized self-driving vehicle using a vision-based CNN network to mimic self-driving control. Wei et al. [11] combined millimeter-wave radar and vision fusion to detect the obstacle precisely. There are three fuse methods: data-level, decision-level, and feature-level fusion. Rajaram et al. [12] developed an iterative region-of-interest pooling framework using a CNN model to predict increasingly tight object boxes and explained limitations. On the other hand, several methods of how to enhance the precision of object detection are mentioned in the following articles. According to images sized 1242 $\times$ 375, the proposed method achieves up to 6% improved accuracy in detection at 0.22 s per frame. Wu et al. [13] devoted to SqueezeDet for self-driving, a fully CNN network with instant inference speed for detecting objects, guaranteeing timely car control with a small model and running in an energy efficiency embedded platform. The SqueezeDet+ model can obtain a frame rate of 32.1 FPS and a precision mAP of 80.4% in the detection of objects. Y. Cai et al. [14] presented a framework, YOLOV4-5D, to increase detection precision based on a real-time operation of YOLOv4. This proposed algorithm can detect objects at 66 FPS where they experimented on an NVIDIA GTX 2080Ti with CUDA 10.0 and cuDNN v10.0. Nevertheless, we improved visual algorithms to increase the speed and precision of object detection simultaneously in our previous work [5]. Chang et al. [5] incorporated Ghost Conv into the YOLOv4-tiny for detecting objects rapidly, abbreviated as LW-YOLOv4-tiny, and the ResNet18 for predicting steering angles quickly, abbreviated as LW-ResNet18. As a result, the proposed combination can achieve 56.1 fps and 0.0683 mean square error. This work experimented on an NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores in Jetson Nano and NVIDIA GTX 1080Ti in a GPU workstation.

2.2. Model Car and Planar Road Map

Our previous work [5] shows a small model car, NVIDIA JetRacer, and uses it to realize self-driving scenario simulation. JetRacer is an autonomous AI racer using NVIDIA

Jetson Nano. Simulation examples and interactive programming can be accessed from a web browser using a JetRacer, which features optimized high frame rates for high-speed movement. A self-driving scenario was made using a JetRacer, with seven cameras around a planar road map for the simulation and self-driving test, as shown in Figure 1. In Figure 1, many small-sized speed limit signs, turn signs, and traffic lights are installed on the planar road map where the JetRacer can be operated to drive and follow traffic rules.

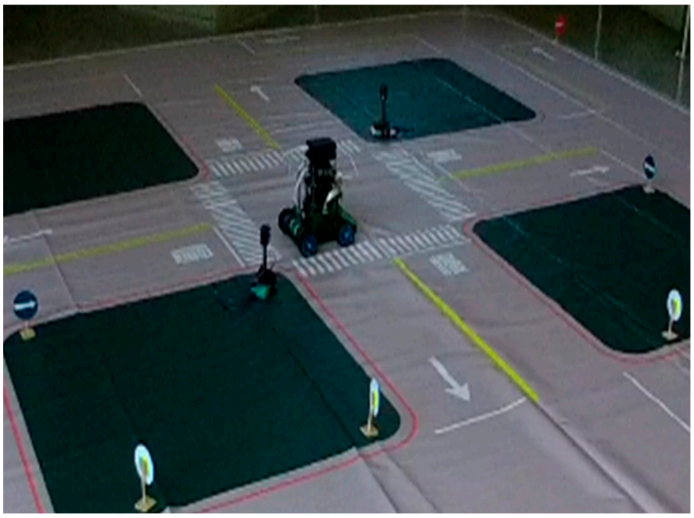


Figure 1. Model car and planar road map [5].

2.3. Rapid Response from LW-YOLOv4-Tiny and LW-ResNet18 Models

In our previous work [5], we improved YOLOv4-tiny and ResNet18 network architecture to speed up the response of detecting objects and predicting steering angles, which resulted in their corresponding lightweight networks, abbreviated as LW-YOLOv4-tiny and LW-ResNet18, as shown in Figures 2 and 3. This combination can shorten the response time of driving control to reduce the risk of severe car accidents. Regarding energy efficiency, both lightweight models can consume less power in detecting objects and predicting steering angles. Furthermore, the previous paper [5] also provided the execution flow of fusing the information and assisting the steering visually in detail. In object detection models, the backbone network extracts features, the neck extracts some more complex features, and the prediction or the head calculates the predicted output.

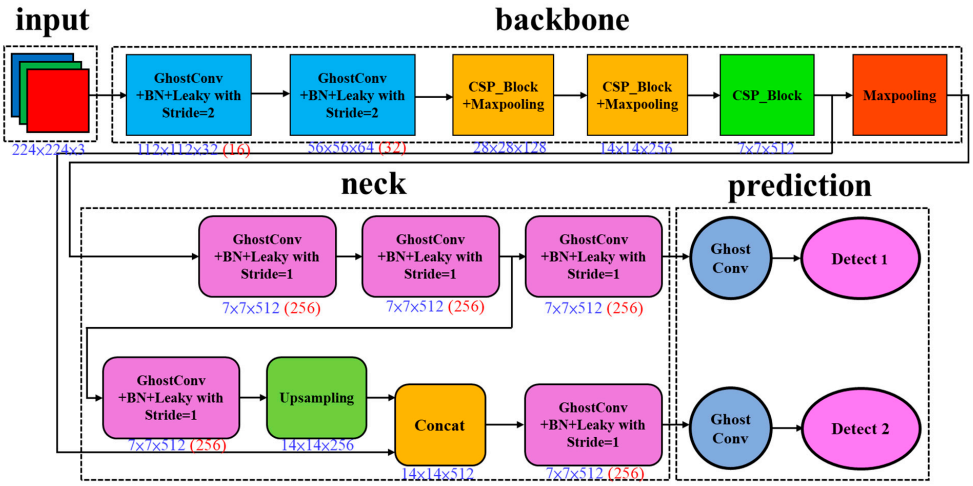


Figure 2. LW-YOLOv4-tiny architecture [5]. Numerical number in red indicates the number of filters in convolutions.

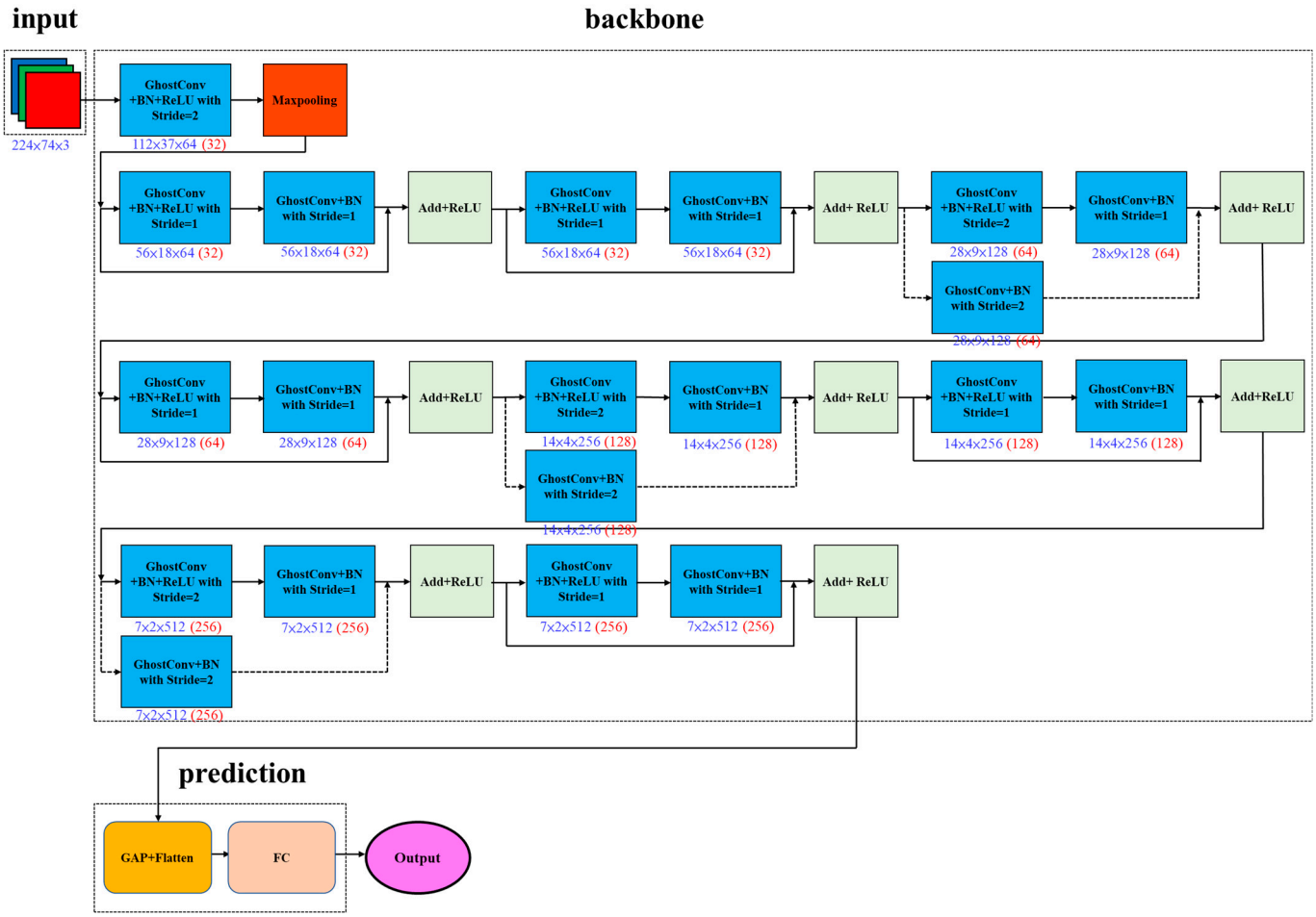


Figure 3. LW-ResNet18 architecture [5]. Numerical number in red indicates the number of filters in convolutions.

3. Methods

3.1. High-Speed Response from LWGSE-YOLOv4-Tiny and LWDSG-ResNet18 Models

Even though LW-LYOLOv4-tiny and LW-ResNet18 proposed in our previous work [5] can accelerate the response of self-driving promptly, this study continues to improve their network architecture to boost the execution speed further. This study intends to alleviate computation load to shorten the response time of the self-driving system and diminish the risk of wrong control judgment. Moreover, the modified architecture can reduce power consumption due to lower convolution computation, making it an energy-efficient entity. According to the previous paper [5], this study proposed a modified architecture of LW-YOLOv4-tiny to construct an enhanced object detection model abbreviated as LWGSE-YOLOv4-tiny, as shown in Figure 4. Moreover, this study introduced a simplified architecture of LW-ResNet18 to build an improved steering angle prediction model abbreviated as LWDSG-ResNet18, as shown in Figure 5. Combining LWGSE-YOLOv4-tiny and LWDSG-ResNet18 can increase the frame rate and speed up the response to self-driving.

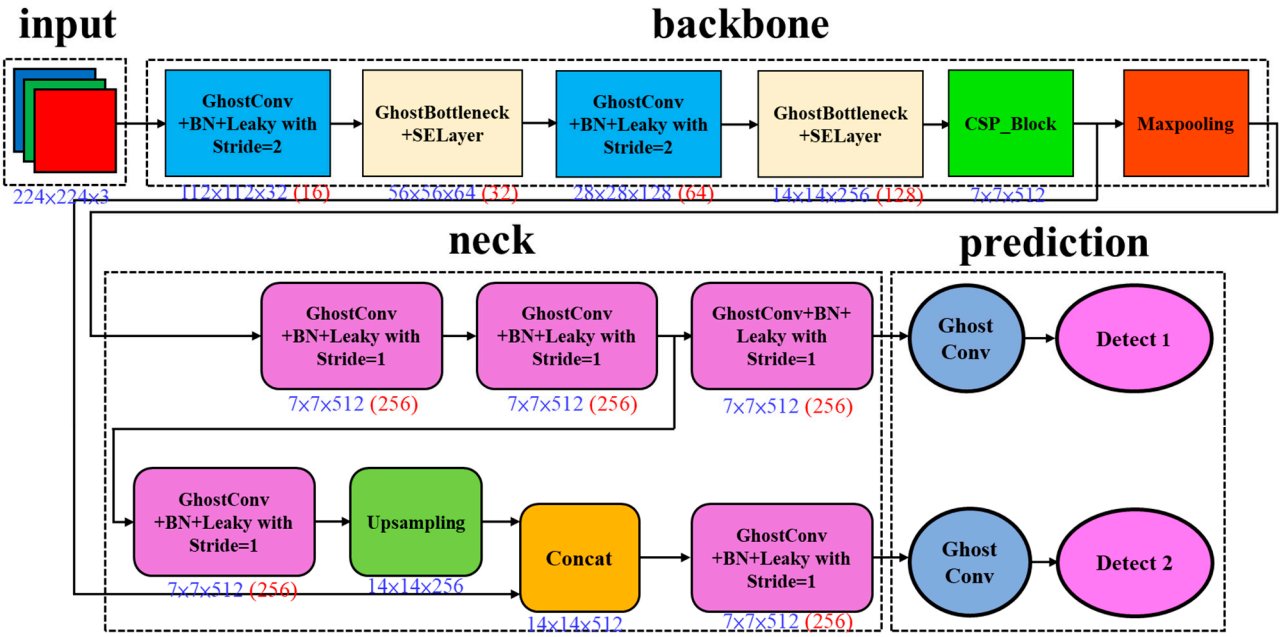


Figure 4. LWGSE-YOLOv4-tiny architecture where the "GhostBottleneck+SELayer" block replaces "CSP_Block+Maxpooling." Numerical number in red indicates the number of filters in convolutions.

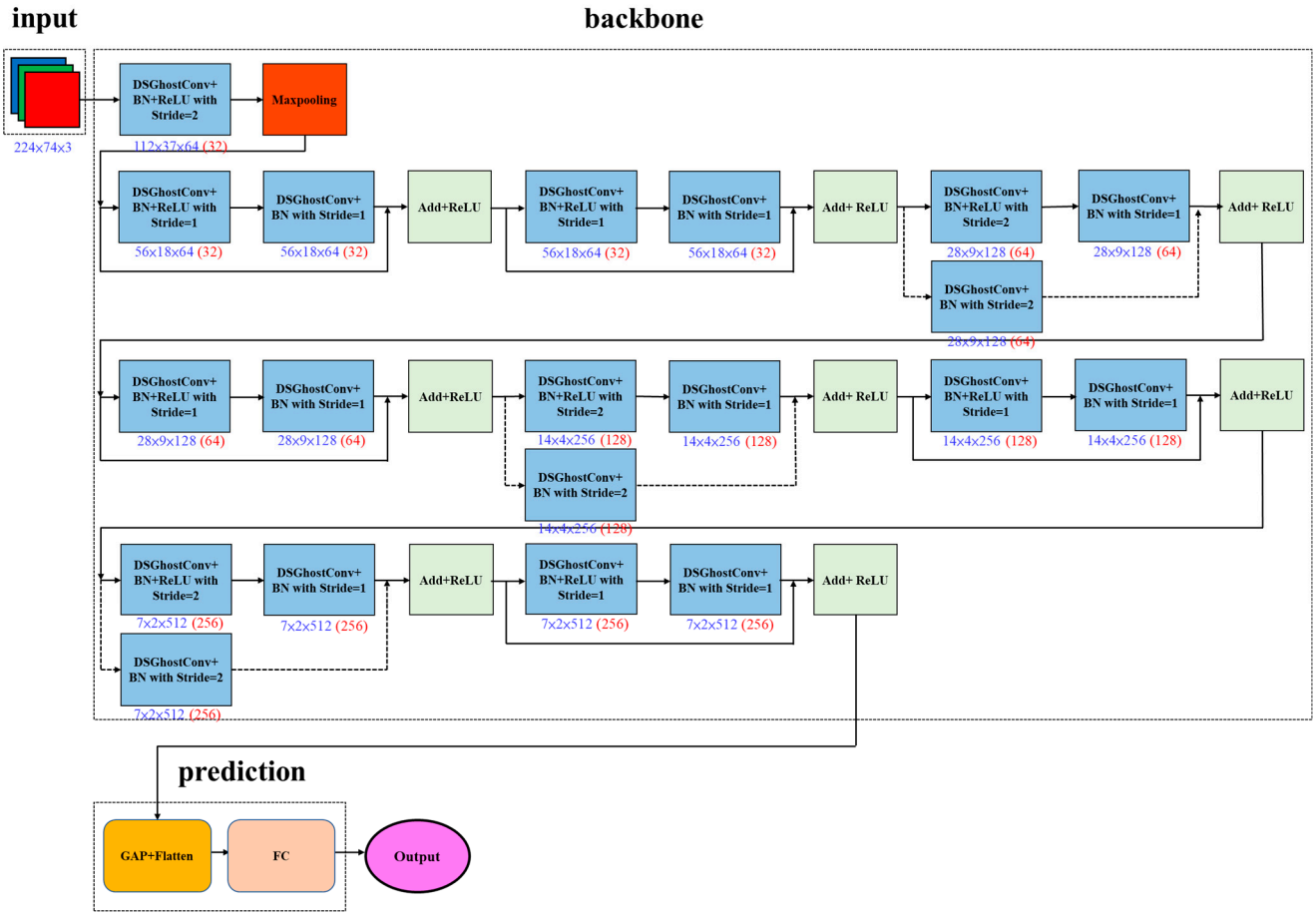


Figure 5. LWDSG-ResNet18 architecture where the: DSGhostConv" block replaces the "GhostConv". Numerical number in red indicates the number of filters in convolutions.

The proposed LWGSE-YOLOv4-tiny uses the GhostBottleneck module [6] to replace the CSP_Block module of the LW-YOLOv4-tiny model, as shown in Figure 4. In Figure 6, the GhostBottleneck uses the ghost convolution instead of the traditional convolution. Its improvement is to reduce the architecture complexity while maintaining precision in feature extraction. The GhostBottleneck module combines feature extraction containing multiple sub-layers, such as convolutional layers, batch normalization layers, and activation function layers. When performing feature transformation, the GhostBottleneck module uses fewer parameters than the CSP_Block module. Technically, GhostBottleneck first performs feature calculations in lower dimensions and then projects the results back to the original high-dimensional space. Such an approach helps reduce computational costs while preserving key features. In addition, Equation (1) computes Randomized Leaky ReLU (RReLU) [15], where p_{ji} represents input, q_{ji} stands for output, α_{ji} denotes the coefficient, N shows the normal distribution, μ is the mean, σ is the variance, and ε is the upper bound. Based on Equation (1), we replace the activation function ReLU in the GhostBottleneck module with the activation function RReLU to further facilitate the feature learning of the model and improve the gradient flow. The general ReLU function returns the positive value when the input is positive but returns zero when the input is negative. RReLU allows a small amount of negative output, which helps to alleviate the vanishing gradient problem and improve the convergence speed and performance of the model.

$$q_{ji} = \begin{cases} p_{ji} & if\ p_{ji} \geq 0 \\ \alpha_{ji} \cdot p_{ji} & if\ p_{ji} < 0 \end{cases}, \ where\ \alpha_{ji} \sim N(\mu, \sigma),\ \mu = 0,\ \sigma \in [0, \varepsilon) \tag{1}$$

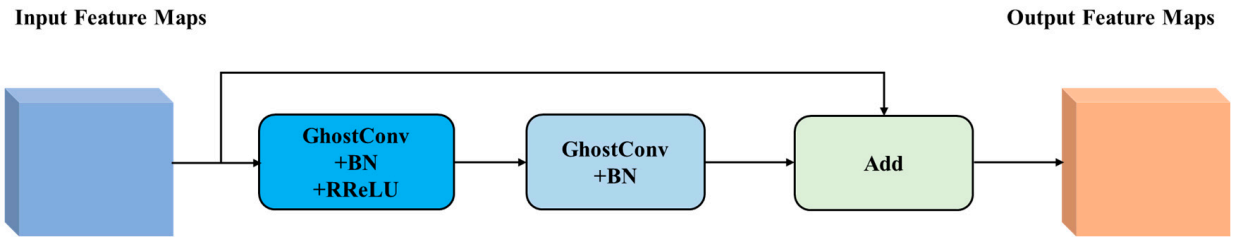


Figure 6. GhostBottleneck architecture.

In Figure 4, this study adds the Squeeze-and-Excitation Layer (SElayer) [7] to the GhostBottleneck module. It removes Maxpooling in the original CSP_Block module to enhance feature representation in convolutional neural networks. The primary purpose of the SElayer is to allow the model to automatically focus on critical features and suppress trivial features, thereby improving the model's overall performance, as shown in Figure 7. The SElayer performs two key stages: extracting key features (Squeeze) and enhancing importance (Excitation). In the Squeeze stage, the SElayer uses global average pooling to reduce the scale of the feature map to generate a global feature descriptor. This function helps to focus on essential regions in the feature map. In the Excitation stage, the SElayer exploits a small neural network called the Gating Network to selectively amplify or compress the features of different channels. This part enables the model to learn the importance of each channel to better capture key features. In other words, the SElayer can automatically adjust the weight of features so that the model can focus on critical features more precisely while reducing the interference of unnecessary information on the model. Moreover, we set the SElayer to improve the activation function from traditional ReLU to RReLU. This improvement allows the model to handle negative inputs better and helps solve the vanishing gradient problem, enhancing the model's feature learning and training efficiency. Finally, ghost convolution [6] performs feature extraction in the neck and prediction parts.

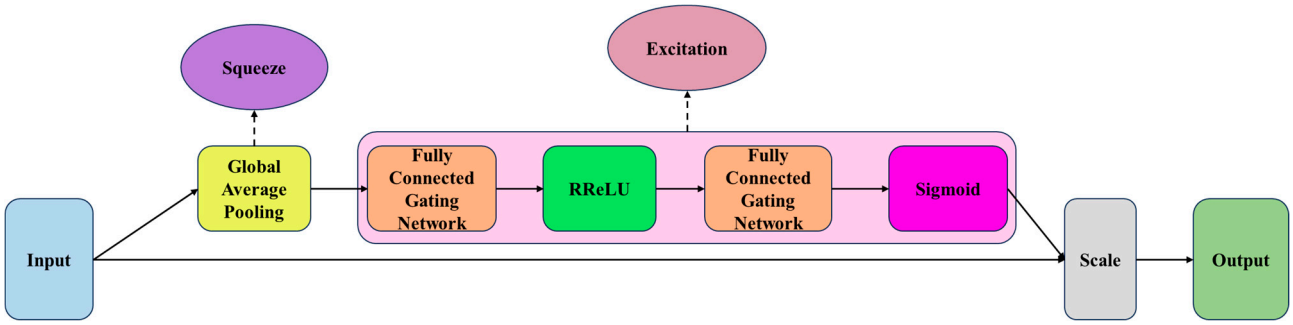


Figure 7. Squeeze-and-Excitation Layer (SElayer) architecture.

In Figure 5, the proposed LWDSG-ResNet18 model adopts a highly efficient convolution called depthwise separable convolution (DS Conv) [8] substituted for a traditional convolution to produce a set of intrinsic feature maps before entering ghost convolution. The primary purpose of DS Conv is to reduce the computational cost while maintaining the model's performance, as shown in Figure 8. In addition, we also propose RReLU activation to replace the ReLU activation function in the DS Conv to optimize the model's performance further. In Figure 9, DS Conv consists of depthwise convolution and pointwise convolution. In the depthwise convolution, each input channel applies a convolution filter independently, effectively capturing the spatial relationships in the input image. In other words, this convolution emphasizes feature interactions inside individual channels. Then, in the pointwise convolution, the output channels of the previous convolution are linearly combined to produce the final feature map of the output channels. It performs cross-channel combinations, enabling the model to obtain satisfactory feature representations with relatively few parameters. Regarding pointwise convolution, Figure 10 provides a more detailed description of how to produce a set of intrinsic feature maps as input feature maps in the subsequent ghost convolution. We can refer to the previous paper [5] to show the steepest gradient descent algorithm, which can update the weight matrix continuously $Wghost_{i,j}$ and the bias matrix $Bghost_{i,j}$, optimizing the matrixes [5] for ghost convolution during the training phase.

The overall process of DS Conv helps to reduce the computational cost because it uses fewer parameters in the convolution operation of these two stages, which can achieve computational efficiency while maintaining the performance of convolutions. Technically, this study has simplified the original Ghost Conv to be depthwise separable and ghost convolution (DSGhost Conv), which can construct the LWDSG-ResNet18 model, as shown in Figure 11. Algorithms 1 and 2 provide the execution flow of DSGhost Conv in detail. In Algorithm 1, DS Conv computes intrinsic feature maps. Then, in Algorithm 2, Ghost Conv takes a series of simple linear transformations of intrinsic feature maps, generating ghost feature maps.

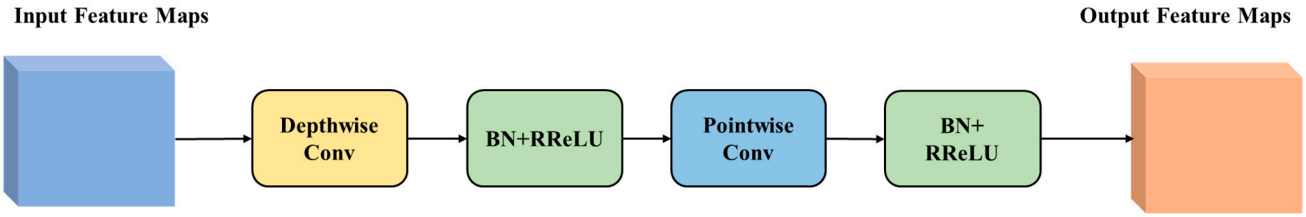


Figure 8. Depthwise separable convolution (DS Conv) architecture.

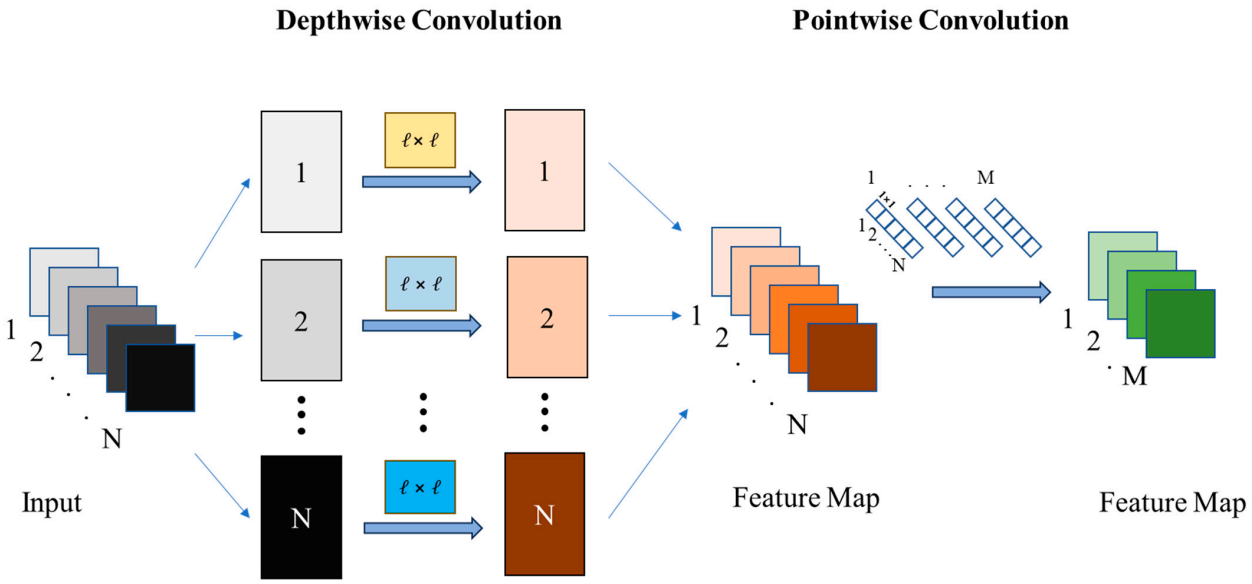


Figure 9. Execution flow of depthwise separable convolution (DS Conv).

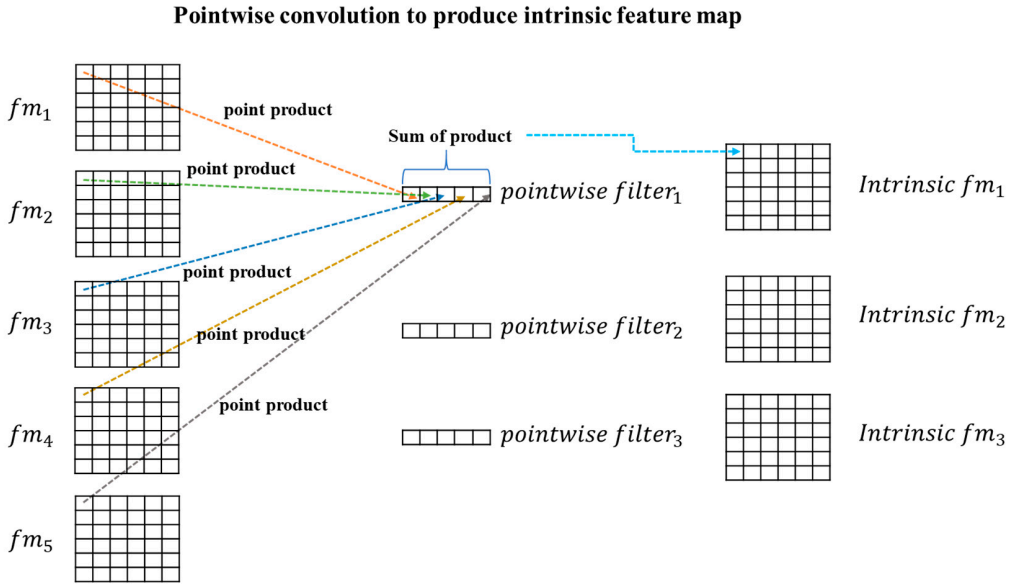


Figure 10. Pointwise convolution.

Algorithm 1 Depthwise Separable Convolution (DS Conv)

Input: Image X, pointwise convolution functions $\mu_{h,i}$
Output: Intrinsic feature map $Insfm_i$

1. Execute a one-time traditional convolution of each input image x_h, with its corresponding filter f_h by $l \times l$ independently to obtain the respective feature map fm_h and complete depthwise convolution where N is the number of input images, the input image $X = \{x_h, \forall\, h = 1, 2, \ldots, N\}$, and the depthwise filter $F = \{f_h, \forall\, h = 1, 2, \ldots, N\}$.
2. Compute the output of the pointwise convolution: $Insfm_i = \mu_{h,i}(fm_h) = fm_h \odot pf_i$ $\forall\, h = 1, 2, \ldots, N,\ i = 1, 2, \ldots, M$ where $\mu_{h,i}$ is a pointwise convolution to obtain intrinsic feature maps $Insfm_i$, $\odot$ shows a point-to-point product of the sum between the feature map fm_h and pointwise filter pf_i, N represents the number of input images, and M stands for the number of intrinsic feature maps.

Algorithm 2 Ghost Convolution (Ghost Conv) [5]

Input: Intrinsic feature map $Insfm_i$, ghost modules with linear transformation functions $\varphi_{i,j}$
Output: $Outfm$

1. Compute the output of the pointwise convolution: $Insfm_i = \mu_{h,i}(fm_h) = fm_h \odot pf_i$ $\forall\, h = 1, 2, \ldots, N,\ i = 1, 2, \ldots, M$ where $\mu_{h,i}$ is a pointwise convolution to obtain intrinsic feature maps $Insfm_i$, $\odot$ shows a point-to-point product of the sum between the feature map fm_h and pointwise filter pf_i, N represents the number of input images, and M stands for the number of intrinsic feature maps.
2. Compute the output of the ghost module:
   $Ghostfm_{i,j} = \varphi_{i,j}(Insfm_i) = Wghost_{i,j} \otimes Insfm_i \oplus Bghost_{i,j}$
   $\forall\, i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, m$ where $\varphi_{i,j}$ is a simplified convolution operation, $Wghost_{i,j}$ shows a weight matrix of the ghost module, $Bghost_{i,j}$ implies a bias matrix of the ghost module, n represents the number of intrinsic feature maps, m stands for the number of the ghost modules, the symbol $\otimes$ denotes the pixel-wise product of two matrixes, and the symbol $\oplus$ indicates the pixel-wise sum of two matrixes.
3. Transform intrinsic feature maps $Insfm_i$ into the ghost feature maps $Ghostfm_{i,j}$, with multiple ghost modules.
   $Outfm = \{Outfm_1, Outfm_2, \ldots, Outfm_n\} = \{Outfm_i,\ \forall\, i = 1, 2, \ldots, n\}$
   $Outfm_i = \{Insfm_i,\ Ghostfm_{i.1},\ Ghostfm_{i.2}, \ldots, Ghostfm_{i.m}\}$
   $= \left\{Insfm_i,\ Ghostfm_{i.j},\ \forall\, j = 1, 2, \ldots, m\right\}$ where $Outfm_i$ represents the output feature maps, including the corresponding intrinsic feature map and ghost feature maps, and $Outfm$ stands for the whole output feature maps in a convolution layer.

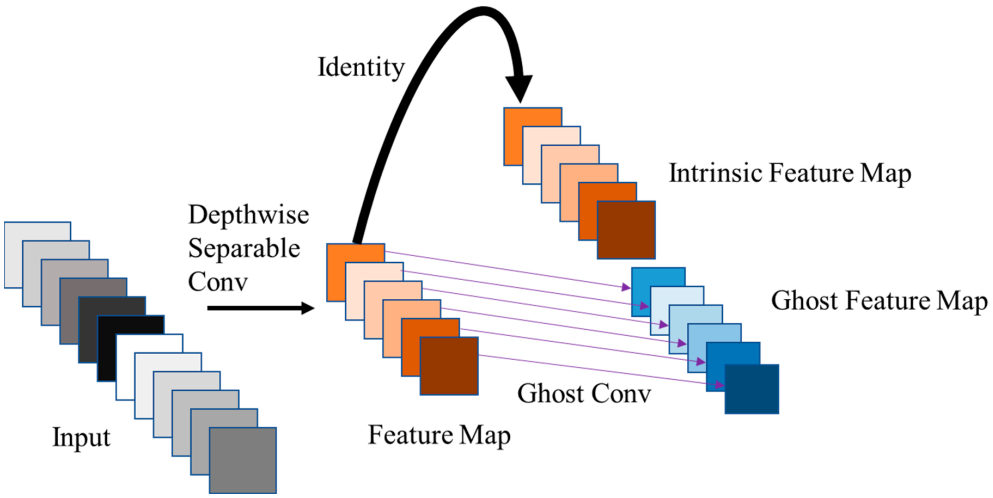


Figure 11. Algorithm 1 implements depthwise separable convolution, and Algorithm 2 performs ghost convolution [5] (abbreviated DSGhost Conv).

Technically, according to specific filters, our previous work [5] performed traditional convolution operations to obtain intrinsic feature maps and then took simple linear transformations to obtain ghost feature maps. In such a way, we can fulfill the overall feature maps and save time to compute traditional convolution operations. The time complexity of the traditional convolution, ghost convolution, and depthwise separable and ghost convolution are $O(\sum_{l=1}^{u} r_l^2 \cdot s_l^2 \cdot c_l \cdot h_l)$, $O(\sum_{l=1}^{u} (r_l^2 \cdot s_l^2 \cdot c_l \cdot v_l + r_l^2 \cdot (h_l - v_l)))$, and $O(\sum_{l=1}^{u} (r_l^2 \cdot s_l^2 \cdot c_l + t_l^2 \cdot u_l^2 \cdot c_l \cdot z_l + r_l^2 \cdot (h_l - z_l)))$, respectively. We defined the notation of time complexity as follows:

r : The side length of the output feature map
s : The side length of the filter
c : The number of channels of the input feature map
h : The number of channels of the output feature map

v : The number of channels of a set of filters of the output feature map through a traditional convolution operation

l: An index of the convolutional layer

u : The number of convolutional layers

t : The side length of the output feature map after depthwise convolution

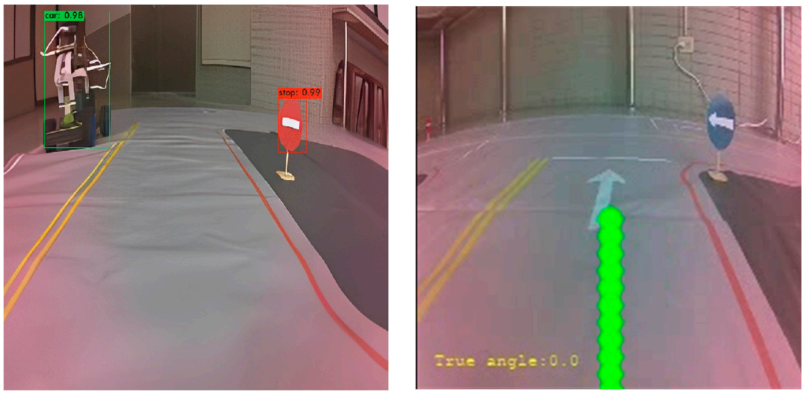
u : The side length of the pointwise filter

z : The number of channels of a set of filters of the output feature map through a pointwise convolution operation

The proposed depthwise separable and ghost convolution can better boost the convolution computation speed than pure ghost convolution. In addition, the proposed one can slightly increase the prediction accuracy because it eliminates redundant information about the convolution operations.

3.2. Scenario of Object Detection and Steering Angle Prediction

In Figure 12a, the front and rear panels of the installed JetRacer dual cameras capture instant video streaming, and LWGSE-YOLOv4-tiny detects the live object and recognizes the class of object. Our previous work [5] adopts visual odometry to measure the distance between the detected objects and the vehicle. In Figure 12b, LWDSG-ResNet18 predicts the steering angle timely along the route while the car is being operated. The expected values partially caused keen steering angle changes that resulted in the car shaking from side to side, making it swing dramatically during operation. Therefore, our previous work [5] added the PID controller to alleviate swing phenomena.



(**a**) (**b**)

Figure 12. Self-driving scenario. (**a**) LWGSE-YOLOv4-tiny instantly detecting objects and (**b**) LWDSG-ResNet18 timely predicting steering angle with a green visual indicator.

4. Experiment Results and Discussion

In the experiment, this study first tested remarkable object detection algorithms, including LWGSE-YOLOv4-tiny, LW-YOLOv4-tiny [5], YOLOv4-tiny [1], YOLOv5s [16], YOLOv5n [17], and YOLOv7-tiny [18]. Then, we tested notable steering angle prediction algorithms, such as Nvidia-CNN [19], traditional CNN [20], ResNet18 [3], LW-ResNet18 [5], and LWDSG-ResNet18. The NVIDIA Corporation has developed a specific convolutional neural network, Nvidia-CNN [19], for self-driving cars' image recognition and steering angle prediction tasks. Compared with traditional CNN, Nvidia-CNN is suitable for image classification, where it has a simple network architecture using convolutional and fully connected layers and adopts a smaller filter. Initially, this study trained the object detection model LWGSE-YOLOv4-tiny and the steering angle prediction model LWDSG-ResNet18. After that, we examined different combinations of six object detection models and five steering angle prediction models. Then, we combined LWGSE-YOLOv4-tiny and LWDSG-ResNet18 running in Jetson Nano, which realized the self-driving of the JetRacer.

4.1. Experiment Setting

With the exact hardware specifications as our previous work [5], this experiment employs the GPU workstation and embedded platform Jetson Nano to train and test the visual algorithms. Similarly, with the same software packages as the previous paper [5], this experiment adopts TensorFlow and TensorRT to accelerate the execution speed of the deep learning models. This experiment uses Anaconda 3 to build the executable environment of deep learning models and collect the training and test images. It also modifies two previous models, LW-YOLOv4-tiny and LW-ResNet18, mentioned in the last paper [5] to be new ones, LWGSE-YOLOv4-tiny and LWDSG-ResNet18, to make them more suitable for instantly implementing object detection and image recognition. Then, it deploys the improved models, LWGSE-YOLOv4-tiny and LWDSG-ResNet18, to the embedded platform Jetson Nano with an executable run-time environment. Finally, this study tests the previous work and this proposal to compare performance.

4.2. Model Training, Inference, and Capability

In training object detection models, the experiment collected 1476 images as a training data set and 366 images as a test data set, with each image at a size of 224 × 224. The ratios of the amount of data are 65%, 16%, and 16% for training, validation, and test data, respectively. The GPU workstation trained all models, and the epoch is 50. Equation (2) evaluates the inference time of detecting the objects in test images where EIT_i is the inference time of each test image, IT_i represents total inference time (IT), i stands for the ith model, I indicates the number of models, x denotes the xth image, and X means the number of images.

$$IT_i = \sum_{x=1}^{X} EIT_i, \ \ where \ \ i = 1, \ 2, \ \ldots, \ I, \quad x = 1, \ 2, \ldots, \ X \tag{2}$$

In Table 1, the first row shows the training time of each model and the second row indicates the inference time. To summarize the results of this training, LWGSE-YOLOv4-tiny bests the others.

Table 1. Training and inference time of object detection models (unit: s).

Method	LWGSE-YOLOv4-Tiny	LW-YOLOv4-Tiny	YOLOv4-Tiny	YOLOv5s	YOLOv5n	YOLOv7-Tiny
Training	258	274	296	497	436	1314
Inference	4.13	4.98	5.32	6.64	6.48	5.75

In steering angle prediction training, the experiment collected 14,710 images as a training data set and 1000 images as a test data set, with each image at a size of 224 × 74. The GPU workstation trained all models, and the epoch is 30. Likewise, Equation (2) evaluates the inference time of predicting the steering angle in test images, where i represents the ith model.

In Table 2, the first row indicates the training time of each model, and the second row shows the inference time. To summarize this training, training LWDSG-ResNet18 is much longer than CNN-related models, but its inference time is slightly better than the others.

Table 2. Training and inference time of steering angle prediction models (unit: s).

Method	LWDSG-ResNet18	LW-ResNet18	ResNet18	CNN	Nvidia-CNN
Training	2745	2832	2880	360	364
Inference	20.17	23.79	25.18	21.27	21.56

Speaking of model size, Table 3 gives the parameters of object detection models, and Table 4 mentions the parameters of steering angle prediction models. YOLOv5s has the

most parameters, and YOLOv5n the least. On the other hand, ResNet18 has the most parameters and CNN the least.

Table 3. Parameters of object detection models.

Method	LWGSE-YOLOv4-Tiny	LW-YOLOv4-Tiny	YOLOv4-Tiny	YOLOv5s	YOLOv5n	YOLOv7-Tiny
# of parameters	3,567,329	3,940,751	5,892,596	7,043,902	1,776,094	6,036,636

Table 4. Parameters of steering angle prediction models.

Method	LWDSG-ResNet18	LW-ResNet18	ResNet18	CNN	Nvidia-CNN
# of parameters	5,468,711	6,367,975	11,180,161	776,289	1,872,643

4.3. Losses in Training and Validation

The experiment uses a tool to visualize the training process and applies a callback function to save the parameters of the best-performed model. After 50 training epochs, Figure 13 shows six loss plots from the proposed object detection model. The results from the other models can refer to our previous work [5]. In Figure 13, the first row shows the training losses, and the second row displays the verification losses. The first column shows the positioning loss, the second column displays the confidence level loss, and the third column indicates the loss of the predicted frame matching the actual frame. In short, LWGSE-YOLOv4-tiny achieved the minimum loss.

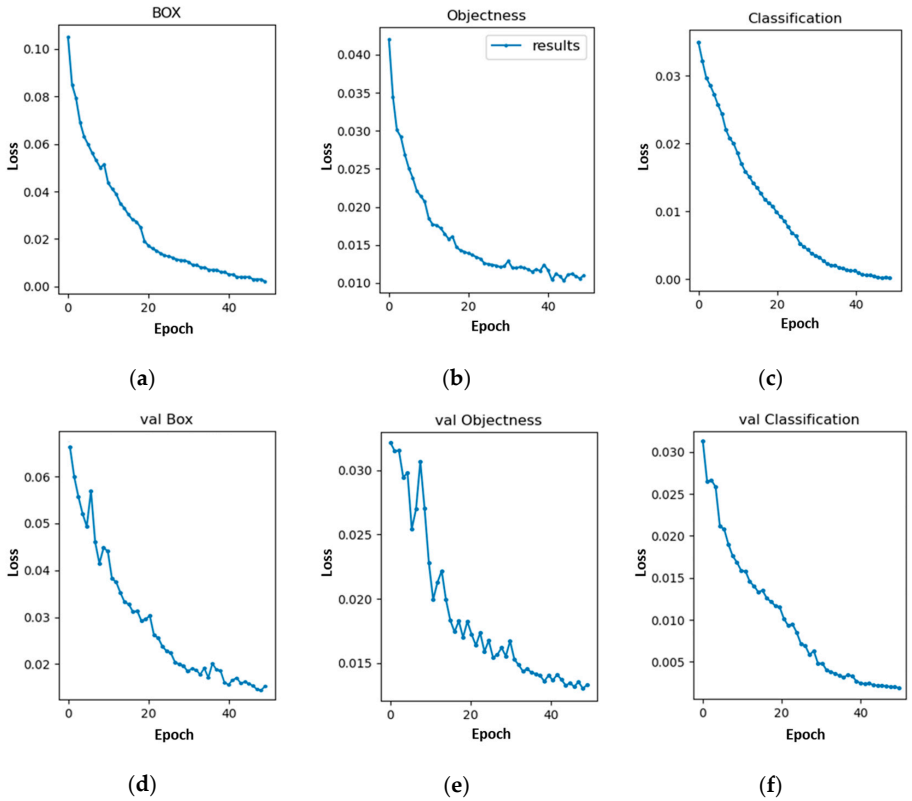


Figure 13. Losses in training and validation for LWGSE-YOLOv4-tiny. (**a**) Loss in box training. (**b**) Loss in objectness training. (**c**) Loss in classification training. (**d**) Loss in box validation. (**e**) Loss in objectness validation. (**f**) Loss in classification validation. Each plot indicates that the x-axis represents the error value and the y-axis stands for the number of epochs.

Figure 14 shows loss plots from LWDSG-ResNet18 resulting from 30 training epochs. Our previous work [5] has given the results of the other models, such as LW-YOLOv4-tiny, YOLOv4-tiny, YOLOv5s, YOLOv5n and YOLOv7-tiny. In Figure 14, the blue indicates the training loss, and the green is the verification loss. LWDSG-ResNet18 lowers the verification loss to 0.0313.

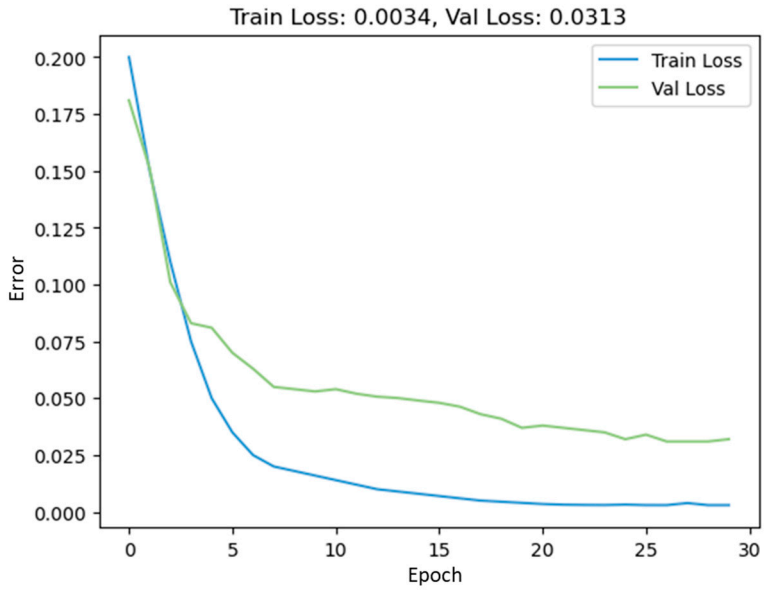


Figure 14. Losses in training and validation for LWDSG-ResNet18.

4.4. Model Testing

Equation (3) computes the speed of detecting objects by frames per second (FPS), where $IRAIT_j$ indicates the time of instantly detecting an object using the jth model, the FPS_j represents the FPS of the jth model, and J stands for the number of models.

$$FPS_j = \frac{1}{IRAIT_j}, \ \ where\ j = 1,\ 2, \ldots,\ J \tag{3}$$

The mean Average Precision (mAP) represents the precision of object detection for a given model, and we can obtain it from the mean of each category's average precision for all categories. Equation (4) computes the precision mAP_l of all object detection models, where AP_{k_l} denotes the precision of a specific category in the lth model, mAP_l stands for the mean Average Precision of the lth model, k_l means a specific category in the lth model, C_l indicates the number of identified categories in the lth model, and L represents the number of models.

$$mAP_l = \frac{\sum_{k_l=1}^{C_l} AP_{k_l}}{C_l}, \ \ where\ k_l = 1,\ 2,\ \ldots,\ C_l,\ l = 1, 2, \ldots, L \tag{4}$$

We can compute the execution speed and precision of each model by testing the object detection of 366 images. Figure 15 plotted the precision–recall curve, called the PR curve. In Figure 15, each point represents a particular recall and precision, where the recall is indicated on the x-axis and the precision is denoted on the y-axis. Our previous work [5] has given the results of the other models, such as LW-YOLOv4-tiny, YOLOv4-tiny, YOLOv5s, YOLOv5n and YOLOv7-tiny. In Table 5, Equation (3) computes FPS, and Equation (4) calculates mAP. To summarize this test, YOLOv5s obtained the best outcome and YOLOv7-tiny demonstrated the poorest outcome.

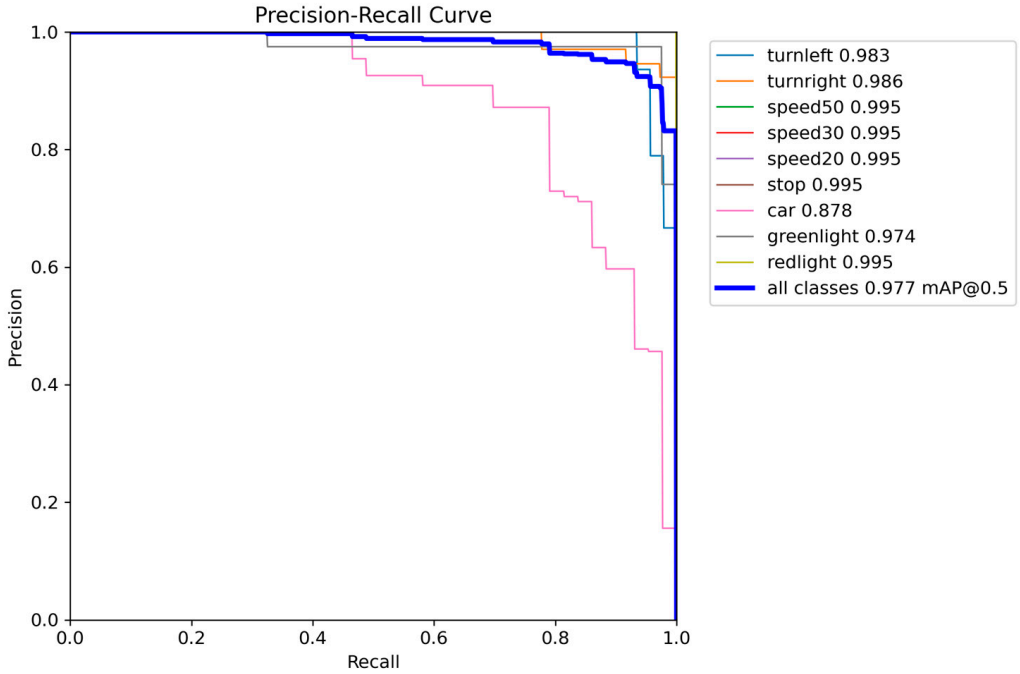


Figure 15. Precision–recall curve for LWGSE-YOLOv4-tiny.

Table 5. Speed and precision of object detection.

Metrics	LWGSE-YOLOv4-Tiny	LW-YOLOv4-Tiny	YOLOv4-Tiny	YOLOv5s	YOLOv5n	YOLOv7-Tiny
FPS	61.7	56.1	46.8	30.4	41.7	43.1
Precision (%)	97.7	97.3	97.2	99.1	97.7	97.0

Equation (3) computes the FPS of predicting steering angles by frames per second (FPS), where j is the jth model. Equation (5) calculates the mean square error (MSE) of predicting steering angles, where $\hat{y}_i$ is the predicted value, y_i means the actual value, N stands for the number of images, k indicates the kth image, and MSE represents the accuracy of a prediction model. The smaller the MSE, the higher the accuracy of predicting the steering angle.

$$MSE = \frac{\sum_{k=1}^{N} (y_k - \hat{y}_k)^2}{N}, \text{ where } k = 1, 2, \ldots, N \tag{5}$$

We can compute the execution speed and accuracy of each model by testing the steering angle prediction of 1000 images. Figure 16 plotted the predicted and actual values. In Figure 16, "−1" represents turning right, "1" stands for turning left, and "0" means going straight where the turning point has a limited scale between −1 and 1. The red indicates the actual steering angle, and the blue shows the predicted steering angle. The results from the other models can refer to our previous work [5]. In Table 6, Equation (3) computes FPS, and Equation (5) computes MSE. In short, this test shows that the LWDSG-ResNet18 achieves the smallest MSE, and the Nvidia-CNN performs the worst.

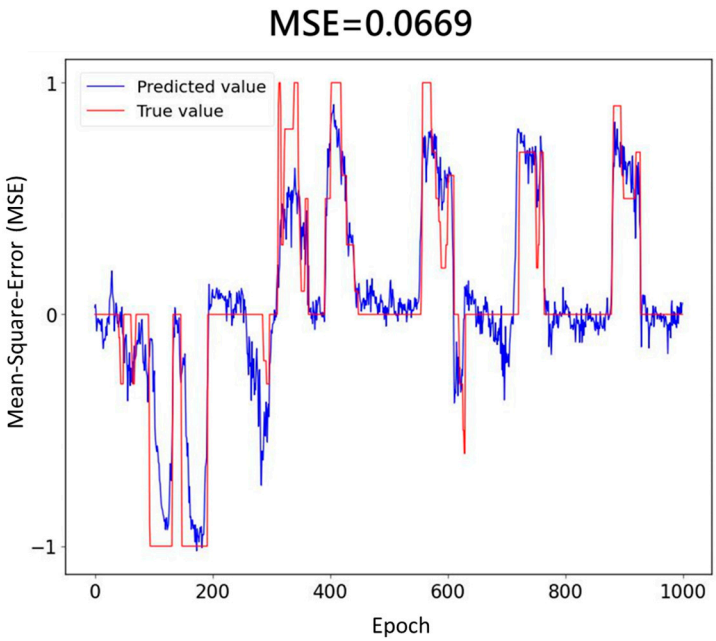


Figure 16. Predicted and actual steering angle using LWDSG-ResNet18.

Table 6. Speed and loss of predicting steering angle.

Metrics	LWDSG-ResNet18	LW-ResNet18	ResNet18	CNN	Nvidia-CNN
FPS	39.3	32.6	27.0	31.9	31.5
MSE	0.0669	0.0683	0.0712	0.0849	0.0953

4.5. Self-Driving System Assessment

The evaluation indicator uses frame rate and mean square error when Jetson Nano executes self-driving control. The self-driving control must instantly detect the object and predict the steering angle simultaneously while JetRacer drives around the road map. Longer to detect objects and insufficient time to complete steering angle prediction will endanger self-driving. Thus, frame rate by frames per second (FPS) is probably the most critical factor. Here, TensorRT can accelerate the FPS of detecting objects.

Equation (3) computes the frame rate of different combinations given a resolution of 224 × 224 per frame. In Table 7, LWGSE-YOLOv4-tiny obtains the best speed on average in various combinations. Combining LWGSE-YOLOv4-tiny and CNN obtains the best FPS, and combining YOLOv5s and ResNet18 is the least.

Table 7. FPS of model combination.

O. D. / St. A. P.	LWGSE-YOLOv4-Tiny	LW-YOLOv4-Tiny	YOLOv4-Tiny	YOLOv5s	YOLOv5n	YOLOv7-Tiny
Nvidia-CNN	25.0	20.4	19.5	15.6	18.1	19.0
CNN	25.3	20.6	19.6	15.8	18.2	19.1
ResNet18	22.2	18.9	18.4	14.5	17.1	17.8
LW-ResNet18	23.8	19.4	18.9	15.1	17.8	18.6
LWDSG-ResNet18	24.9	21.8	21.3	18.2	18.9	21.0
Average	24.2	20.2	19.5	15.8	18.0	19.1

O. D.—object detection; St. A. P.—steering angle prediction.

Then, each combination examines the precision of detecting objects and the accuracy of predicting steering angles given a resolution of 224 × 224 per frame. Equations (4) and (5) compute the precision of detecting objects and the accuracy of predicting steering angles,

respectively. In Table 8, YOLOv5s achieves the best precision and LWDSG-ResNet18 has the lowest MSE.

Table 8. Accuracy and precision of model combination.

MSE \ Precision (%)		O. D.					
		LWGSE-YOLOv4-Tiny	**LW-YOLOv4-Tiny**	**YOLOv4-Tiny**	**YOLOv5s**	**YOLOv5n**	**YOLOv7-Tiny**
St. A. P.	Nvidia-CNN	(0.0953, 97.7)	(0.0953, 97.3)	(0.0953, 97.2)	(0.0953, 99.1)	(0.0953, 97.7)	(0.0953, 97.0)
	CNN	(0.0849, 97.7)	(0.0849, 97.3)	(0.0849, 97.2)	(0.0849, 99.1)	(0.0849, 97.7)	(0.0849, 97.0)
	ResNet18	(0.0712, 97.7)	(0.0712, 97.3)	(0.0712, 97.2)	(0.0712, 99.1)	(0.0712, 97.7,)	(0.0712, 97.0)
	LW-ResNet18	(0.0683, 97.7)	(0.0683, 97.3)	(0.0683, 97.2)	(0.0683, 99.1)	(0.0683, 97.7)	(0.0683, 97.0)
	LWDSG-ResNet18	(0.0669, 97.7)	(0.0669, 97.3)	(0.0669, 97.2)	(0.0669, 99.1)	(0.0669, 97.7)	(0.0669, 97.0)

The first number within a parenthesis represents the steering angle prediction loss (MSE), and the second stands for the object detection precision (%). O. D.—object detection; St. A. P.—steering angle prediction.

4.6. Discussion

The method proposed by Wei et al. [11] must combine millimeter wave and visual detection, which may waste a lot of time when fusing information so that object detection will be slower. The architecture proposed by Rajaram et al. [12] achieves impressive gains in performance at a fast run-time speed of 0.22 s per frame, and such a speed of object detection cannot respond timely to the control of a self-driving car when driving at high speeds. The model SqueezeDet proposed by Wu et al. [13] is a fully convolutional neural network for instantly detecting objects obtaining a rate of 32.1 FPS and precision in mAP of 80.4%, where insufficient precision may cause a large misjudgment that may lower the performance of object detection.

A study by Y. Cai et al. [14] conducted its experiment on an NVIDIA GTX 2080Ti as the computing platform to implement object detection using 66 frames/s (fps). In contrast, the proposed approach adopted a GPU workstation with NVIDIA GTX 1080Ti to train the model and Jetson Nano with 128 NVIDIA CUDA® cores to realize the object detection task in the experiment. The graphic computing speed of Jetson Nano is much slower than that of NVIDIA GTX 1080Ti. Nevertheless, this study applied Jetson Nano with the proposed visual algorithm to significantly implement object detection using 61.7 frames/s (fps) in the self-driving system. In other words, the approach proposed in this study can simultaneously perform object detection and steering angle prediction rapidly and precisely.

The delayed steering angle prediction could cause the risk of severe car accidents. Therefore, speed is more important than precision in this case. This study improves the two previous models [5], LW-YOLOv4-tiny and LW-ResNet18, speeding up the execution speed and enhancing the image recognition accuracy concurrently by introducing two modified models, LWGSE-YOLOv4-tiny and LWDSG-ResNet18, respectively. As a result, compared with LW-YOLOv4-tiny, LWGSE-YOLOv4-tiny can boost the frame rate of feature extraction up to 9.98% and slightly upgrade the precision of object detection by 0.41%. Compared with LW-ResNet18, LWDSG-ResNet18 can boost the prediction speed to 20.55% and increase image recognition accuracy by 2.05%. Therefore, the proposed approach can accomplish the primary goal of this study.

However, some drawbacks occurred in this case. The embedded platform Jetson Nano encountered a problem with the hardware limitation. Jetson Nano cannot process instant higher-resolution video streaming while the car is driving. Thus, we can replace it with the Jetson AGX Xavier to capture and process higher-resolution video streaming. Unfortunately, JetRacer cannot supply enough battery power to the Jetson AGX Xavier. Therefore, there is a need for power-efficient model cars.

5. Conclusions

The main contribution of this study is the enhancement of the execution efficiency of both the frame rate of the object detection and the image recognition accuracy of steering

angle prediction results, which boosts the entire response time of self-driving control significantly. In other words, the proposed approach achieved the objective of this study to implement a highly efficient self-driving control by reducing the execution time considerably. Compared with the previous work, the proposed approach can significantly speed up the entire self-driving control by 1.28 times. The performance evaluation shows that the proposed method also outperforms the other alternatives.

We must devote ourselves to object detection and steering angle prediction for further improvements in future works. In addition, integrating ROS and high-performance visual algorithms can effectively implement instant self-driving control. Since the self-driving system performs advanced vision algorithms for an extended period while the car is driving and has a high degree of computing power consumption, we will also seek low-power and high-performance embedded platforms to execute advanced vision algorithms in the future. Ultimately, we will also look for digital maps to provide some path planning for self-driving control to facilitate automatic navigation. Thus, the above measures can hopefully deal with the discussed limitations.

Author Contributions: B.R.C. and F.-Y.C. conceived and designed the experiments; H.-F.T. collected the dataset and proofread the manuscript; and B.R.C. wrote the paper. All authors have read and agreed to the published version of the manuscript.

Funding: The Ministry of Science and Technology fully supports this work in Taiwan, Republic of China, under grant numbers NSTC 112-2622-E-390-001 and NSTC 112-2221-E-390-017.

Data Availability Statement: The Sample Programs for Sample Program.zip data used to support the findings of this study. https://drive.google.com/file/d/1rDxUP-VzPiA07YYQ12_4GAj0xfww1DIc/view?usp=sharing (accessed on 1 September 2023).

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

References

1. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. Scaled-Yolov4: Scaling Cross Stage Partial Network. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 13029–13038.
2. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal Speed and Accuracy of Object Detection. *arXiv* **2020**, arXiv:2004.10934.
3. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
4. Zhang, G.; Geng, L.; Chen, X. Sound Source Localization Method Based on Densely Connected Convolutional Neural Network. In Proceedings of the 2022 5th International Conference on Information Communication and Signal Processing (ICICSP), Shenzhen, China, 26–28 November 2022; pp. 743–747.
5. Chang, B.R.; Tsai, H.F.; Chou, H.L. Accelerating the Response of Self-Driving Control by Using Rapid Object Detection and Steering Angle Prediction. *Electronics* **2023**, *12*, 2161. [CrossRef]
6. 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, 13–20 June 2020; pp. 1580–1589.
7. Hu, J.; Shen, L.; Sun, G. Squeeze-and-excitation networks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 7132–7141.
8. Howard, A.G. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861. [CrossRef]
9. JetRacer AI Kit, Waveshare Wiki. Available online: https://www.waveshare.com/wiki/JetRacer_AI_Kit (accessed on 1 May 2023).
10. Karni, U.; Ramachandran, S.S.; Sivaraman, K.; Veeraraghavan, A.K. Development of Autonomous Downscaled Model Car Using Neural Networks and Machine Learning. In Proceedings of the 3rd IEEE International Conference on Computing Methodologies and Communication, Erode, India, 27–29 March 2019; pp. 1089–1094.
11. Wei, Z.; Zhang, F.; Chang, S.; Liu, Y.; Wu, H.; Feng, Z. Mmwave radar and vision fusion for object detection in autonomous driving: A review. *Sensors* **2022**, *22*, 2542. [CrossRef] [PubMed]
12. Rajaram, R.N.; Ohn-Bar, E.; Trivedi, M.M. Refinenet: Refining object detectors for autonomous driving. *IEEE Trans. Intell. Veh.* **2016**, *1*, 358–368. [CrossRef]
13. Wu, B.C.; Iandola, F.; Jin, P.H.; Keutzer, K. SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 129–137.

14. Cai, Y.; Luan, T.; Gao, H.; Wang, H.; Chen, L.; Li, Y.; Sotelo, M.A.; Li, Z. YOLOv4-5D: An effective and efficient object detector for autonomous driving. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 4503613. [CrossRef]
15. Fujii, S.; Hayashi, H. Comparison of Performance by Activation Functions on Deep Image Prior. In Proceedings of the 2019 International Conference on Artificial Intelligence in Information and Communication, Okinawa, Japan, 11–13 February 2019; pp. 255–258.
16. Jocher, G.; Stoken, A.; Borovec, J.; Changyu, L.; Hogan, A.; Diaconu, L.; Ingham, F.; Poznanski, J.; Fang, J.; Yu, L. Ultralytics/Yolov5: v3.1-Bug Fixes and Performance Improvements. 2020. Available online: https://zenodo.org/record/4154370#.ZFjQp3ZByUk (accessed on 1 September 2023).
17. Jocher, G.; Stoken, A.; Chaurasia, A.; Borovec, J.; Kwon, Y.; Michael, K.; Skalski, S.P. ultralytics/yolov5: v6.0-YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support. *Zenodo Technol. Rep.* **2021**. [CrossRef]
18. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-The-Art for Real-Time Object Detectors. *arXiv* **2022**, arXiv:2207.02696.
19. Bojarski, M.; Testa, D.W.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L.D.; Monfort, M.; Muller, U.; Zhang, J.; et al. End to End Learning for Self-Driving Cars. *arXiv* **2016**, arXiv:1604.07316.
20. LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.* **1989**, *1*, 541–551. [CrossRef]