Vehicle and Pedestrian Detection Algorithm Based on Lightweight YOLOv3-Promote and Semi-Precision Acceleration

He Xu, Mingtao Guo, Nadia Nedjah, Jindan Zhang, and Peng Li, Member, IEEE

Abstract—Aiming at the shortcomings of the current YOLOv3 model, such as large size, slow response speed, and difficulty in deploying to real devices, this paper reconstructs the target detection model YOLOv3, and proposes a new lightweight target detection network YOLOv3-promote: Firstly, the G-Module combined with the Depth-Wise convolution is used to construct the backbone network of the entire model, and the attention mechanism is introduced and added to perform weighting operations on each channel to get more key features and remove redundant features, thereby strengthening the identification ability of feature network model’s to distinguish target objects among background; Secondly, in order to delete some less important channels to achieve the effect of compressing the model size and improving the calculation speed, the size of the scaling factor gamma in the batch normalization layer is used; Finally, based on NVIDIA’s TensorRT framework model conversion and half-precision acceleration were carried out, and the accelerated model was successfully deployed on the embedded platform Jetson Nano. The performed KITTI experimental results show that the inference speed of our proposed method is about 5 times that of the original model, the parameter volume is reduced to one tenth, the mAP is increased from 86.1% of the original model to 93.1%, and the FPS reaches 25.5fps, realizing the requirements of real-time detection with high precision.

Index Terms—Target detection, lightweight model, model prune, model deployment, semi-precision acceleration.

Manuscript received 29 July 2021; revised 13 November 2021; accepted 14 December 2021. Date of publication 4 January 2022; date of current version 1 October 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2103003; in part by the National Natural Science Foundation of China under Grant 61872196, Grant 61872194, and Grant 61902196; in part by the Scientific and Technological Support Project of Jiangsu Province under Grant BE2017166 and Grant BE2019740; and in part by the Major Natural Science Research Projects in Colleges and Universities of Jiangsu Province under Grant 18KJA520008; in part by the Postgraduate Research and Practice Innovation Program of Jiangsu Province under Grant KYCX_0973; in part by the Six Talent Peaks Project of Jiangsu Province under Grant RFW-111; and in part by the 1311 Talent Plan of the Nanjing University of Posts and Telecommunications (NUPT). The Associate Editor for this article was B. B. Gupta. (Corresponding authors: Jindan Zhang; Peng Li.)

He Xu and Mingtao Guo are with the School of Computer Science, Nanjing University of Posts and Telecommunications (NUPT). The Associate Editor for this article was B. B. Gupta. (Corresponding authors: Jindan Zhang; Peng Li.)

Nadia Nedjah is with the Department of Electronics Engineering and Telecommunications, Rio de Janeiro State University, Rio de Janeiro 20550013, Brazil (e-mail: nadia@eng.uerj.br).

Jindan Zhang is with the Department of Electronics Engineering and Telecommunications, Nanjing University of Posts and Telecommunications, Nanjing 210023, China (e-mail: xuhe@njupt.edu.cn; 1219043832@njupt.edu.cn).

Peng Li is with the School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China (e-mail: lipeng@njupt.edu.cn).

Digital Object Identifier 10.1109/TITS.2021.3137253

NOWADAYS, the development of the Internet of vehicles [33], [34] is becoming more and more popular in the world. The Internet of vehicles integrates the Internet of things, intelligent transportation, and cloud computing [1]. At present, the most well-known and vigorously developed Internet of vehicles application is autonomous driving [35]–[37]. There is a driver assistance system in autonomous driving. The system uses cameras and Lidar to collect information about the scene around the car in real time and make warnings to remind the driver of abnormal conditions around the vehicle, so that the driver can find dangerous situations that he/she is unaware of, as soon as possible, which improves the driver’s driving safety. The rapid detection of targets such as vehicles and pedestrians is very important for the driving assistance systems.

In recent years, the object detection method has stood out among many detection algorithms, and has attracted the attentions of professionals and scholars in the industry and academia, and has gradually become the favor of the industry. With the development of deep learning technology, the target detection algorithm has also changed from the two-stage method that originally included R-CNN [2], SPP-Net [3], Fast R-CNN [4] and Faster R-CNN [5] to the more popular one-stage method. One-stage method is a kind of regression-based target detection, which aims to solve the problem of incompatibility between real-time and accuracy. It is mainly divided into Single Shot MultiBox Detector [6]–[10] (SSD) series methods and You Only Look Once [11]–[13] (YOLO) series of methods. YOLOv3 builds a deep residual network to extract image features by referring to the residual network structure of ResNet, and obtains the best detection speed and accuracy at present. Although YOLOv3 has the ability to detect small targets, it is not obvious for long-distance small targets in complex situations, and it is prone to missed, false and repeated detections, as shown in Figure 1. In addition, the amount of parameters and calculations of the YOLOv3 native model are very large, and the requirements for hardware in the operating environment are high. Therefore, it is not feasible to use the original YOLOv3 model to deploy in mobile devices. Real-time and on-vehicle detection cannot be realized, and it can only function as a driving recorder.

One of the major shortcomings of deep learning model is that the calculations are massive and the number of parameters is very large. YOLOv3 is an example [14]. This is also one of the reasons deep learning projects cannot be commercialized in
recent years, especially with some marginalized devices. Since these devices are not designed for computationally intensive tasks, deploying deep learning models directly on them would definitely cause problems such as delays and high-power consumption. In addition, for the current automotive system, real-time detection must be supported by a powerful remote server. When the parameters the detection model are too many, the estimation time will be long, the real-time performance would be greatly compromised. This would directly lead to higher power consumption and thus higher cost. Therefore, people in the industry are trying to mitigate this problem from various aspects. For example, with the rise of various neural network chips and high memory graphics cards, this can try to increase hardware’s computing power to speed up the network model. Another idea is to focus on the software. Since many parameters of the model only play a role in the training stage and are not used in the prediction stage, there is a lot of parameter redundancy, which makes the network calculation time-consuming. Therefore, in this paper, we use model reconstruction and model pruning to obtain a lightweight model, and the proposed model can be deployed on mobile terminal devices, and can achieve high-precision for target detection.

The main contributions of the proposed improved model, termed YOLOv3-promote, are three-fold:

1. YOLOv3-promote uses Depth-Wise convolutional combination of G-Module to construct the backbone network of the entire model, and adds an attention mechanism to the backbone network to strengthen the ability of feature network to distinguish target objects from background.

2. YOLOv3-promote uses the size of the scaling factor gamma in the Batch Normalization layer to delete some less important channels to achieve the compressed model, which can reduce the size of the proposed model, thus offering a new lightweight model.

3. YOLOv3-promote performs model conversion and half-precision acceleration based on NVIDIA’s TensorRT framework, and successfully deploys the accelerated model to the embedded platform Jeston Nano, realizing real-time detection of mobile terminals.

II. METHODOLOGY

Once trained, excellent neural network models produce feature maps that are rich with informations, which may even be redundant. For instance, in the YOLOv3 model, some features are shown in Figure 2. One can see that the highlighted with a red and blue markers present similar features. One would think that these redundant features increase the amount of calculation of the network, and thus try to eliminate them. However, these so-called redundant features are particularly important for target detection and recognition. It is because of their existence that the network has a comprehensive understanding of the input data. Therefore, by keeping these redundant features, we propose a new model that can generate more feature maps with only some calculation-G-Modules. In this paper, the most important part of the proposed model is the Depth-Wise convolution.

A. Depth-Wise Convolution

Depth-Wise convolution [15]–[17] can effectively reduce the complex of calculations and the number of parameters of the model, yet the network can still express image features well under the premise of greatly reducing model complexity. Through the following comparison of traditional convolution and Depth-Wise separable convolution, one can clearly understand the difference between the amount of calculations and the number of parameters.

Generally, the number of channels of the convolution kernel of traditional convolution coincide with that of channels of the input feature matrix, and the number of channels of the output feature matrix coincide with that of convolution kernels. Figure 3 illustrates a traditional convolution operation process.
The size of the input feature matrix in Figure 3 is \( DF \times DF \times M \). After the operation of \( N \) convolution kernels with a depth of \( DK \), the depth of the output feature matrix is equal to the number of convolution kernels, so an \( N \)-channel output feature matrix is obtained. The overall number of FLOPs of the traditional convolution of Figure 3 is \( DK \times DK \times M \times N \times DF \times DF \).

The Depth-Wise convolution is a combination of DW convolution and PW convolution. In general Depth-Wise convolution, the channel size of the convolution kernel is set as 1, and the number of channels of the input feature matrix is the number of convolution kernels, which is also equal to the number of the output feature matrix. The Point-Wise convolution is composed of a \( 1 \times 1 \) convolution kernel. Therefore, in order to compare with the calculation amount of traditional convolution, the size of the convolution operation \( \phi \) is set as \( 1 \), and the size of the convolution operation \( \phi \) is \( 1 \times 1 \times 1 \).

The Depth-Wise convolution process is shown in Figure 4. After the operation of \( \phi \), the channel size of the convolution kernel is set as \( 1 \), and the number of channels of the input feature matrix is equal to the number of convolution kernels, so an \( N \)-channel output feature matrix is obtained. The overall number of FLOPs of the Depth-Wise convolution of Figure 4 is \( DK \times DK \times M \times DF \times DF + M \times N \times DF \times DF \).

The ratio between the number of FLOPs of Depth-Wise convolution and that of the traditional convolution, is defined in (1):

\[
FLOPs = \frac{1}{N} + \frac{1}{DK^2} \tag{1}
\]

In this work, we use convolution kernels of size \( DK = 3 \), so the overall number of FLOPs is as in (2):

\[
FLOPs = \frac{1}{N} + \frac{1}{9} \tag{2}
\]

That is, theoretically, the calculation complexity in terms of FLOPs of the traditional convolution is 8 to 9 times that of Depth-Wise separable convolution.

B. G-Module

In this paper, with the application of DW convolution, G-Module is proposed. As the number of convolution kernels increases, the number of generated feature maps will also increase, and FLOPs will also increase significantly. Therefore, in order to reduce the amount of calculations, we must reduce the number of convolution kernels.

Because YOLOv3 produce the phenomenon of feature map redundancy [18], since there is no need to generate redundant feature maps, that is, redundant feature maps can be generated from a type of feature map through some operation. Then it can be assumed that \( M \) mother-feature maps are generated first, and then a simple linear transformation is performed through these maps to obtain the final required \( N \) feature maps. In theoretically, it simplifies the calculation and retains the richness of features. Therefore, we design a G-Module which is shown in Figure 5.

The G-Module is divided into the following two parts. The one part is the convolution operation where we use the reduced number of convolution kernels. The input feature matrix \( X \) is \( c \times h \times w \), and the size of the convolution operation \( \phi \) is \( m \times k \times k \). The convolution kernel has \( m \) feature maps, and the parameter overall number is \( H \times W \times m \times c \times k \times k = H \times W \times \frac{m}{d} \times c \times k \times k \).

The second part is the linear transformation \( \phi \). In this work, it is modeled as a series of Depth-Wise separable convolutions, which perform \( s \) linear transformations on \( m \) feature maps \( (n = m \times s) \), where the linear kernel size is \( d \times d \), the output characteristic matrix \( Y \) is \( n \times H \times W \), and the parameter number is \( H \times W \times m \times d \times d \times (s - 1) = H \times W \times \frac{m}{d} \times d \times d \times (s - 1) \).

Considering the traditional convolution operation method, the input is the characteristic matrix of \( c \times h \times w \) and the output is the characteristic matrix of \( n \times H \times W \), thus the
parameter overall number of the model is \( H \times W \times k \times k \times n \).

\[
R = \frac{H \times W \times c \times k \times k \times n}{H \times W \times c \times k \times k \times \frac{2}{3} + H \times W \times c \times d \times d \times (s-1)} \\
= \frac{c \times k \times k \times \frac{1}{3} + d \times d \times \frac{s-1}{s}}{s + c - 1} \\
\approx s
\]  

(3)

The ratio \( R \) between the parameter number of the traditional convolution operation and the G-Module operation proposed in this paper, can be obtained as in formula (3). It follows that the parameter number of the traditional convolution operation is \( s \) times of the G-module operation parameter number proposed in this paper. Therefore, for obtaining the same feature map, G-Module can significantly reduce the model complexity while retaining redundant features, which has a great impact on subsequent model deployment.

C. G-Bottleneck

Based on the ideas of ResNet [19] and MobileNet-v2, we combine the residual structure with G-Module, and propose the G-Bottleneck module. According to the stride size, it is divided into G1-Bottleneck and G2-Bottleneck. Figure 6 shows the structure of G-Bottleneck under stride = 1 and stride = 2 respectively. G1-Bottleneck first contains the input feature matrix, and then undergoes two G-Module operations. The difference is that the first G-Module has an ReLu activation function after the batch normalization operation, and the second G-Module is after the batch normalization operation. The resulting output feature matrix will be spliced with the initial position input feature matrix through shortcut to get the final result. G2-Bottleneck is the same with the only difference from G1-Bottleneck is that a layer of DW convolution is added between the two G-Modules. The main purpose of G2-Bottleneck is to reduce the size of the feature maps and prepare for the next processing round.

D. Attention Mechanism

The attention mechanism [20] has been widely used in the fields of natural language processing (NLP) and computer vision (CV). The visual attention mechanism is obtained through the response of the human brain. Humans obtain important target information by quickly viewing images. This important target information is the so-called attention point [21]. The attention mechanism in computer vision is similar to the attention mechanism of the human brain [22]–[24]. It also selects the most important information currently needed from many target information. Attention mechanisms are divided into two types, one is soft attention, and the other is hard attention. The soft attention mechanism can focus on channel and region information, and it is differentiable. Hence the attention weights of channels and regions can be assigned through the back propagation of the neural network so that the channels or regions corresponding to important targets in the image get more weights. The hard attention is not differentiable, and it is generally used in reinforcement learning.

In this work, an efficient attention mechanism is proposed. We present the advantages and disadvantages regarding the parameter number and accuracy among the Squeeze-and-Excitation Networks (SENet) attention mechanism [25], Convolutional Block Attention Module (CBAM) [26], and our proposed attention mechanism [27] to the backbone networks of ResNet50, ResNet101, and ResNet152. After many experiments, it is concluded that the attention mechanism proposed in this paper has the advantages of fewer parameters and higher accuracy, and the results are shown in Figure 7.

Although the SE Module uses two fully connected layers to weigh the channels, the dimensionality reduction operation of the first fully-connected layer reduces the correlation between the channels. Therefore, the attention mechanism used in this work abandons the dimension reduction and captures cross-channel channel interaction in an effective way, as shown in Figure 8, which is the attention mechanism used in our work.

The channel attention mechanism used in this paper exploits global pooling to aggregate the spatial characteristics of the feature map. Unlike the SE module, the attention module in the proposed model generates channel weights quickly by using \( K \) one-dimensional convolutions, and the value of \( K \) is adjusted adaptively by channel dimension mapping. The purpose of the attention mechanism is used to capture the local cross-channel interaction, where the key is to determine the interaction coverage. By analogy, the coverage ratio of the interaction should be proportional to the channel dimension \( C \), that is as
shown in (4):

\[ C = \varphi(K) \]  

(4)

The mapping \( \varphi \) is unknown. Generally, the simplest linear function can be expressed as \( C = y \times k + B \). However, from the above analysis, \( K \) and \( C \) are in nonlinear proportion, and channel \( C \) is generally the exponential power of 2. Therefore, the solution is that we can convert the linear function \( C \) into the exponential form of nonlinear function as formula (5):

\[ C = \varphi(K) = 2 \times (y + k + b) \]  

(5)

Then, given the size of channel dimension \( C \), the kernel size \( K \) can be solved by formula (6):

\[ K = \theta(C) = \frac{\log_2 c}{y} + \frac{b}{y_{\text{odd}}} \]  

(6)

The \( \text{odd} \) in (6) denotes the odd number nearest to each other. In this work, \( y \) and \( b \) are determined, \( K \) is proportional to \( c \). The larger the value of \( c \), the larger the value of \( K \).

Figure 9 shows the effect after adding the attention module proposed herein. After the feature map passes the previous convolutional layer, the target with higher confidence is selected and mapped to the original image. The red region in Figure 9 is the one with highest confidence. It can be seen that the proposed attention mechanism enables the image to focus more on the part of the target object and less attention to the background. This shows that the proposed attention mechanism effectively enhance the important features of the image, suppress redundant features, and improve the network’s ability to recognize foreground and background.

E. Overall Structure of the Model

In this paper, on the basis of YOLOv3, three YOLO output layers are retained, the backbone network of the entire model is reconstructed, and the G-Bottleneck module composed of G-Module is added to obtain a new lightweight model YOLOv3-promote. In addition, the G-Bottleneck module in YOLOv3-promote in the proposed model adds an attention mechanism, through a local cross-channel, non-dimensionality reduction channel interaction method, autonomously learns the weight of each channel, thereby eliminating redundant features and enhancing the Characteristics of key information.

The network structure of YOLOv3-promote based on model reconstruction and attention mechanism is shown in Figure 10. Therein, the backbone network before the 131 layer in the figure is the reconstructed backbone network, and the following is the detection network in the original YOLOv3. The purple part is attention Force mechanism module.

The backbone network of YOLOv3-promote is G-Net, which refers to the residual structure proposed by ResNet and MobileNet. Because batch normalization is used, it avoids the risk of gradient explosion caused by increasing the depth of the network. In addition, YOLOv3-promote uses convolution with stride of 2 to achieve downsampling, abandoning the pooling layer used in many networks originally. The purpose is to further reduce the negative effect of gradients brought by pooling and improve the accuracy. The Convolutional layer in Figure 10 is composed of three components, namely Conv2d + Batch Normalization + Leaky ReLu. For small
TABLE I
LAYER STRUCTURE OF YOLOv3-PROMOTE

| Input | Operation | Output | Attention | Stride |
|-------|-----------|--------|-----------|--------|
| $416 \times 3$ | Conv2d $3 \times 3$ | 16 | N | 2 |
| $208 \times 16$ | G1-bneck | 16 | N | 1 |
| $208 \times 16$ | G2-bneck | 24 | N | 2 |
| $104 \times 24$ | G1-bneck | 24 | N | 1 |
| $104 \times 24$ | G2-bneck | 40 | Y | 2 |
| $52 \times 40$ | G1-bneck | 40 | Y | 1 |
| $52 \times 40$ | G2-bneck | 80 | N | 2 |
| $26 \times 80$ | G1-bneck | 80 | N | 1 |
| $26 \times 80$ | G1-bneck | 80 | N | 1 |
| $26 \times 80$ | G1-bneck | 80 | N | 1 |
| $26 \times 80$ | G1-bneck | 112 | Y | 1 |
| $26 \times 112$ | G1-bneck | 112 | Y | 1 |
| $13 \times 160$ | G1-bneck | 160 | N | 1 |
| $13 \times 160$ | G1-bneck | 160 | N | 1 |
| $13 \times 160$ | G1-bneck | 160 | Y | 1 |
| $13 \times 160$ | G1-bneck | 160 | Y | 1 |
| $13 \times 160$ | Conv2d $1 \times 3$ | 960 | N | 1 |
| $13 \times 960$ | AvgPool $13 \times 13$ | - | - | - |

After that, the subsequent series of Conv2d operations which include $1 \times 1$ and $3 \times 3$ convolutions are used and then the output feature matrix is used for subsequent YOLO layer detection.

In the proposed model, we retain the three layers of YOLOv3. The G-Bottleneck module is composed of the proposed G-Module. The number of network layers is set according to FPN. Based on the results of some experiments, it is finally set as the depth in Figure 10. The size of the feature map coincides with the input image. During the experiments, the used images have a size of 416 × 416.

III. MODEL PRUNE

The concept of model pruning is based on a hypothesis, or the consensus of all current deep learning workers, that is, the over-parameterization of deep learning neural networks. Deep learning trains the network by calculating a large number of parameters, and finally obtains the predicted result. Like many machine learning models, deep neural networks can be divided into two parts: training and detection. The training stage mainly learns various parameters based on the data set. The detection stage delivers the new data to the trained model, and the result is obtained through calculation. Over-parameterization means that the original network needs enough parameters to fully find the optimal solution of the target. When the training is completed and the detection phase is entered, we can find the target just as long as we keep the optimal parameters. In short, model pruning can be understood as eliminating all the long-distance roads leading to the destination, leaving only the best shortcut. Based on this assumption, we can simplify the model before deployment. The simplified model has the following advantages: First, the amount of calculation is reduced, which leads to lower latency and lower power consumption; second, the memory usage is lower, and it can run on some low-end embedded devices; third, it is lightweight Model packages are conducive to release, update and maintenance. In this work, model pruning is used to compress the proposed reconstructed YOLOv3-promote, so that the parameters and delay of the model are reduced while keeping the accuracy almost unchanged. Model pruning is divided into fine-grained pruning and coarse-grained pruning [28]. Fine-grained pruning belongs to unstructured pruning, and the granularity of pruning is a single neuron. The coarse-grained pruning is to directly subtract the filter or channel. Due to the convenience and feasibility of operation and the need for special hardware support, channel pruning has been widely studied and used.

In this paper, we adopt the Batch Normalization (BN) layer, and the subsequent pruning strategy is performed by adding the L1 penalty function to the BN layer. The BN layer is actually not an optimization algorithm, but an adaptive reparameterization method, in order to solve the problem of the model’s difficulty in training caused by the deepening of the model layer. The entire normalization function is in (7) and (8):

$$\tau = \frac{\alpha_i - \mu}{\sigma_i}$$  \hspace{1cm} (7)

$$a_{i}^{\text{norm}} = y_i \times \tau + \beta_i$$  \hspace{1cm} (8)
where $a_i$ is the original activation value of a certain neuron, that is, the feature map of each channel $B \times H \times W$, $a_i^{\text{norm}}$ is the activation value that is normally distributed after normalization operation, and $\mu$ is the activation value contained in the neuron set $S$. The average number of activation values of $n$ neurons in $S$, $\sigma_i$ is the standard deviation of the activation value of each neuron and the mean $\mu$, and $\gamma_i$ and $\beta_i$ are the two adjustment factors corresponding to each channel feature map. In short, $\gamma$ can be regarded as the weight of each channel in the BN layer. If the weight corresponding to the current channel $C_1$ is $\gamma_1$ and $\gamma_1 = 0$ or $\gamma_1 \approx 0$, then there will be $\gamma_1 \cdot \tau = 0$, that is, the channel will have no effect in the following calculations. Therefore, the adjustment factor $\gamma$ can be an indicator to measure the importance of the channel. When $\gamma = 0$ or $\gamma$ approx 0, the channel where $\gamma$ is located is tailored. This method can achieve a slimming effect on the network, reduce unnecessary parameters and calculations and increase the forwarding speed of the network, making it easier to deploy to terminal devices. The flow of channel pruning is shown in Figure 11. In Figure 11, the true values of the gammas, beta and lambda, are given as examples. We can get the results of all gammas, beta and lambda values obtained by experiments. In our paper, we obtained the optimal values of beta, gamma, and lambda through multiple experiments using the proposed model.

Although through the above method, the $\gamma$ value is usually distributed normally, and the channel with $\gamma = 0$ or $\gamma$ approx 0 can be clipped, but in many cases, there will not be many $\gamma$ values equal to or close to 0. Therefore, we need to use the L1 regular expression to sparse the $\gamma$ value of each BN layer, and strive to achieve the effect of making a large number of parameter values 0. The L1 regularization formula is in (9).

$$L = \sum_{(i, y)} l (f (x, W), y) + \lambda \sum_{\gamma \in \Gamma} g (\gamma)$$  \hspace{1cm} (9)$$

The first term is used to calculate the loss produced by the model prediction, the second is used to constrain $\gamma$, and $\lambda$ is used to weigh the hyperparameters set by the two items. According to a large amount of experimental experience, the value of $\lambda$ is generally set as $1e^{-4}$ or $1e^{-5}$, is $g (s) = |s|$, which is the L1 paradigm, which plays a role of sparseness.

IV. TENSOR RT MODEL CONVERSION AND HALF-PRECISION ACCELERATION

NVIDIA TensorRT [29] includes a deep learning inference optimizer and runner. In the inference process, TensorRT-based applications execute 40 times faster than those based on CPU platforms alone. However, the performance of the GPU platform is definitely better than the Jetson Nano mobile terminal we used. The current mobile terminal without the GPU platform has the ability to deploy deep learning algorithms using our proposed model to implement the object detection. TensorRT can optimize the deployment of deep learning model to different applications, such as video streaming, speech recognition, recommendation systems and NLP. Reducing accuracy can significantly reduce application latency, which is a requirement for many real-time services, automation, and embedded applications [38], [39].

Before building the TensorRT platform, firstly, we need convert the proposed model into a form that TensorRT can read, such as the model structure of ONNX. Our model is implemented based on Pytorch’s deep learning framework. So, we convert the model to ONNX. Because Pytorch’s own interface torch.onnx contains the convolutional, pooling, activation, upsampling and detection layers used in the model, so only one-click conversion is required.

The obtained ONNX model is then loaded into TensorRT for model simplification and FP16 acceleration. The model simplification is mainly to merge the Conv layer, BN layer and Relu layer in the network into one layer, referred to as the CBR layer. As shown in Figure 12, taking the common Inception structure as an example, according to the model simplification principle in TensorRT, the inception structure in ① is converted to the network structure in ②; for the horizontal combination in the network, the input is the same tensor. It is merged with the layers of the same operation, as shown in the conversion from ② to ③. The specific process for transitions from ① to ②, and to ③ are the following steps: ① to ② requires that TensorRT merges the Conv layer, BN layer and ReLu layer in the network into one layer which is named as the CBR layer; ② to ③ is for the final stitching layer, wherein there is no need to perform concatenation separately and then input the calculation, but the input of concatenation is directly sent to the subsequent operation, which is equivalent to reduce the throughput of a system. With the support of the TensorRT platform, the FP16 acceleration reduces the data accuracy from 32 floating-point numbers to 16-bit floating-point numbers, greatly improving operation and calculation efficiency.
V. EXPERIMENT AND RESULT

The presented YOLOv3-promote method is made available via the public dataset KITTI. The model is implemented based on Pytorch’s deep learning framework. The hardware configurations are as follows: The processor is Intel(R) Core(TM) i9-9900K CPU with operation frequency of 3.60GHz; The memory size is 16.0GB; the graphics card is a single 2080Ti, and the video memory size is 11GB. The configuration environment of the software is: Ubuntu 18.04, CUDA 10.2, CUDNN 7.6.5, and the programming language is Python 3.7. The embedded platform deployed is the NVIDIA Jetson Nano suitable for industrial deployment, as shown in Figure 13.

Jetson Nano is an embedded high-performance development board newly launched by NVIDIA. It is equipped with quad-core CORTEX-A57 processor, 128-core MAXWELL GPU and 4GB LPDDR memory. The operating environment is JetPack 4.4.1. The batch size is increased in the experiments in order to heighten the usage of the graphics card memory, and the apex technology provided by NVIDIA to Pytorch is used for double-precision mixed training.

A. Data Set Description

The KITTI dataset [30], [31] is the largest computer vision algorithm evaluation data set in the automatic driving scene in the world. KITTI contains a variety of real scene image data, such as urban, rural, and highway areas. Therein, each image contains vehicles and pedestrians, shadows, different illumination, and so on, which can give an effective test for the robustness of the algorithm. The labels of the KITTI original dataset are divided into eight categories: Car, Van, Truck, Pedestrian, Pedestrian (sitting), Cyclist, Tram, Misc. However, since the primary goal of automatic driving in the application of Internet of vehicles is to detect vehicles and pedestrians, for the experiments performed in this work, we deletes the Misc, and changes the original eight categories of labels into the three categories: Car (including Van, Truck and Tram), Person (including Pedestrian and Pedestrian-sitting), Cyclist. So, the three categories are Car, Person, and Cyclist. We use 7481 images in the dataset as the experimental data and allocates one-tenth of the dataset to the verification set.

B. Execution Details

In the performed experiments, we do training and testing on images of the same size, and we compare the obtained results to those achieved by YOLOv3 as baseline. The input image resolution is increased to 608 × 608 pixels. Through the
darknet53 network, SPP, and attention modules, the information about the target vehicle and pedestrian in the image is extracted, and three feature maps with different scales are used to predict the target location and type. For anchor box selection, the K-means algorithm [32] is used to generate a total of 9 clusters’ centers for the labeled images in the KITTI dataset: (7, 66), (9, 23), (13, 34), (19, 54), (22, 161), (24, 36), (35, 65), (57, 107), (96, 196). Figure 14 shows the distribution of the 9 anchors in all real frames:

In this paper, the backbone network uses the model parameters of Darknet53.conv.74 in training process. YOLOv3-promote required a total of 2000 epochs. The batch size is 64, and the number of subdivisions is 16. The momentum parameter and weight decay regularization term are set to 0.9 and 0.0005 respectively, and the learning rate parameter is 0.001. When iterating to 7000 times and 10,000 times, the learning rate decreases to one-tenth of the previous. In addition, we use data enhancement to generate more training samples. By setting saturation parameter to 1.5, exposure amount to 1.5, hue to 0.1, with data jitter and horizontal flipping, the robustness is increased, the accuracy and the generalization of various real environments are improved.

C. Overall Process

The overall flow is shown as follows: (1) Use the G-Bottleneck combined with the G-Module and the attention mechanism to reconstruct the backbone network of the model, and the loss function is adjusted to GIoU loss function. (2) Use the K-means algorithm to re-cluster the dataset to obtain new anchor points. (3) Perform sparse training on the obtained new model YOLOv3-promote, so that the $\gamma$ factor in the BN layer is as close to 0 as possible. (4) After the sparse training is completed, prune the model to remove those channels with low weights. (5) Fine-tune the pruned model to obtain the final lightweight model. (6) Build the TensorRT platform, convert the model into ONNX format, read the model into the acceleration engine of TensorRT, and deploy it to the embedded device Jeston Nano. The overall process is shown in Figure 15.

D. Result of Detection

We use the precision ($P$), recall, mean average precision (mAP), FPS, parameter number and time delay as the evaluation criteria. Among these, the precision refers to the proportion of correctly detected targets in the total detected targets, and the recall refers to the proportion of correctly detected targets in all targets in the verification set. The $P$ and recall metrics are generally inversely proportional, that is, the greater the precision, the smaller the recall. In the field of target detection, the final measurement index is still mAP, which determines the detection effect. The value of mAP is generally derived from the area of the PR (precision-recall) curve.

Furthermore, we use normal and sparse training modes to train three models: YOLOv3, YOLOv3-tiny and YOLOv3-promote. The sparsity factor is set to 0.0001 for training. Table I shows the performance indicators of the three models after normal and sparse training. It can be seen that no matter which model, after sparse training, the general mAP is higher than after normal training due to the L1 in sparse training. Regularization reduces the over-fitting phenomenon in the process of model training. Due to the use of deep separable convolution to build the backbone network, the model YOLOv3-promote proposed in this paper is only about one-third of YOLOv3 in terms of parameters, but mAP is 6 percentage points higher than YOLOv3. The specific results are shown in Table II.

Figure 16 and Figure 17 show the sparsity level of the $\gamma$ factor of the proposed model YOLOv3-promote without and after 150 epochs sparsification training. It can be seen that after sparsity training, most of the $\gamma$ coefficients in all BN layers drop to a value close to 0, thus satisfying the preconditions for pruning, and thus pruning and fine-tuning are performed.

Table III shows the changes of each convolutional layer of model YOLOv3-promote after pruning. It can be seen that

![Fig. 15. The process of the experiment.](image-url)
TABLE II

| Methods      | Precision | Recall | mAP   | Parameters |
|--------------|-----------|--------|-------|------------|
| YOLOv3       | 82.4      | 83.2   | 88.5  | 88.7       | 86.1      | 86.3      | 61.9M |
| YOLOv3-tiny  | 52.7      | 53.5   | 58.4  | 58.2       | 57.6      | 59.7      | 8.9M  |
| YOLOv3-promote | 88.4     | 89.7   | 93.5  | 92.7       | 91.4      | 92.9      | 23.5M |

Fig. 16. Not trained in sparsity.

Fig. 17. Trained in sparsity.

TABLE III

| Convolution layer | Before prune | After prune |
|-------------------|--------------|-------------|
| Conv1             | 512          | 222         |
| Conv2             | 1024         | 378         |
| Conv3             | 512          | 184         |
| Conv4             | 1024         | 300         |
| Conv5             | 512          | 158         |
| Conv6             | 1024         | 394         |
| Conv7             | 256          | 137         |
| Conv8             | 512          | 236         |
| Conv9             | 256          | 109         |
| Conv10            | 512          | 193         |
| Conv11            | 256          | 88          |
| Conv12            | 512          | 296         |
| Conv13            | 128          | 58          |
| Conv14            | 256          | 120         |
| Conv15            | 128          | 60          |
| Conv16            | 256          | 108         |
| Conv17            | 128          | 54          |
| Conv18            | 256          | 131         |

so the result is no longer in exponential form. Table III also shows that, except for the three convolutional layers of Conv7, Conv12 and Conv18, less than one-half of the parameters have been subtracted, and the remaining convolutional layers have been subtracted by more than one-half or more of the parameters. This is the reason why the amount of calculation reduction is lower than the total amount of parameters.

Table IV shows the various indicators of the final model of YOLOv3-promote after pruning and fine-tuning (hereinafter collectively referred to as YOLOv3-promote-prune). In the experiment, because the pruning caused a sudden change in the model and the calculation was biased, the accuracy is reduced. However, after fine-tuning, the accuracy returned to normal, and the accuracy is slightly improved. Moreover, the detection speed compared with the original model, increased by 67.9%, and the total number of parameters reduced to a quarter of that of the original model. Although only 57.3% of the $\gamma$ coefficients are cut, the total number of model parameters still dropped significantly.

Finally, the proposed YOLOv3-promote-prune model is converted to ONNX format, and then loaded into the built TensorRT platform for half-precision acceleration (hereinafter referred to as the final model). Table V shows the final model after model reconstruction, model after pruning and half-precision acceleration, the performance indicators of the original YOLOv3 are changed. We can know that the detection speed of the final model is 5 times that of the original YOLOv3, and the parameters are only one-tenth of the former. Table VI present the running time for YOLOv3, YOLOv3-tiny and YOLOv3-promote proposed in our experimental environment. It can be seen that the presented YOLOv3-promote has similar time to the YOLOv3-tiny. This indicates that the presented model can be used for real-time detection.
The YOLOv3-promote-prune model is selected to compare with the optimal model of the initial YOLOv3. The comparison chart used is shown in Figure 18 and classified according to daytime, night, extreme weather, multiple targets and small targets.

As it can be observed in Figure 18, the difference between the effects of the two models is the smallest during the daytime, but the traditional YOLOv3 still misidentifies distant pedestrians as vehicles (misdetected or missed vehicles are marked with yellow arrows in Figure 18), whilst when using the proposed model all details are detected. Regarding the night images, the gap between the two algorithms is particularly obvious. The traditional YOLOv3 has difficulties to effectively detect places with weak light due to the lack of attention mechanism, whilst almost all the models detect all the details. Regarding extreme weather, the same conclusion is true, the traditional YOLOv3 did not detect small distant targets due to the rainy weather and blurred windows. In the case of multiple and small targets, the gap between YOLOv3 and the YOLOv3-promote-prune is reflected in the detection of small targets in the distance.

Figure 19 shows the deployment of the YOLOv3-promote-prune model into Jetson Nano.

VI. CONCLUSION

In this work, we carried out model reconstruction, model pruning and half-precision acceleration on the classic YOLOv3 model in target detection. Furthermore, the model parameters are reduced to tenths of what is required in the original model, the accuracy is significantly improved. We successfully deployed the final model on Jetson Nano, an embedded device with low computing power, and achieved a detection speed of 25.5 frames per second. We also successfully mitigate the effects of deep learning model regarding the required large size, slow response and difficult deployment in equipment’s of limited computational power. The experimental results show that the inference speed of the proposed method is about 5 times that of the original model, the parameter volume is reduced to one tenth, the mAP is increased from 86.1% of the original model to 93.1% on the KITTI data set.

In the future, we intend to implement a hardware to embed the presented model and perform more objects detection deployment. In addition, we are currently studying the recently proposed YOLOv4 [40], which has a higher performance in speed and accuracy for the detection of multiple objects of a single frame. Therefore, we will use our method to improve further the performance of YOLOv4.

REFERENCES

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.
[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Piscataway, NJ, USA, Jun. 2014, pp. 580–587.
[3] K. M. He et al., “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in Proc. Int. Conf. Pattern Anal. Mach. Intell., vol. 37, no. 9, Piscataway, NJ, USA, Nov. 2015, pp. 1904–1915.
[4] R. Girshick, “Fast R-CNN,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Piscataway, NJ, USA, Dec., 2015, pp. 1440–1448.
[5] S. Q. Ren et al., “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Proc. Int. Conf. Pattern Anal. Mach. Intell., Washington, DC, USA, 2017, vol. 39, no. 6, pp. 1137–1149.
[6] W. Liu et al., “SSD: Single shot MultiBox detector,” in Proc. Eur. Conf. Comput. Vis. (ECCV), vol. 9905. Cham, Switzerland: Springer, 2016, pp. 21–37.
[7] S. T. Liu, D. Huang, and Y. H. Wang, “Receptive field block net for accurate and fast object detection,” in Proc. Eur. Conf. Comput. Vis. (ECCV). Cham, Switzerland: Springer, 2018, pp. 385–400.
[8] Z. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, “Single-shot refinement neural network for object detection,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Piscataway, NJ, USA, Jun. 2018, pp. 4203–4212.
[9] J. Ren et al., “Accurate single stage detector using recurrent rolling convolution,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Washington, DC, USA, Jul. 2017, pp. 5420–5428.
[10] Z. Li and F. Zhou, “FSSD: Feature fusion single shot multibox detector,” 2017, arXiv:1712.00960.
[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Piscataway, NJ, USA, Jun. 2016, pp. 779–788.
[12] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Piscataway, NJ, USA, Jul. 2017, pp. 7263–7271.
