Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 7670724, 9 pages
https://doi.org/10.1155/2021/7670724

Research Article

An Edge Computing-Enabled Train Obstacle Detection Method Based on YOLOv3

Song Li,[1] **Hongli Zhao**,[2] **and Jinmin Ma**[3]

[1]*State Key Lab. of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing 100044, China*
[2]*National Engineering Research Center of Rail Transportation Operation and Control System, Beijing Jiaotong University, Beijing 100044, China*
[3]*Capital Normal University, Beijing 100044, China*

Correspondence should be addressed to Hongli Zhao; hlzhao@bjtu.edu.cn

Received 12 July 2021; Revised 21 August 2021; Accepted 30 August 2021; Published 8 October 2021

Academic Editor: Chi-Hua Chen

Rail transit is developing towards intelligence which takes lots of computation resource to perform deep learning tasks. Among these tasks, object detection is the most widely used, like track obstacle detection, catenary wear, and defect detection and looseness detection of train wheel bolts. But the limited computation capability of the train onboard equipment prevents running deep and complex detection networks. The limited computation capability of the train onboard equipment prevents conducting complex deep learning tasks. Cloud computing is widely utilized to make up for the insufficient onboard computation capability. However, the traditional cloud computing architecture will bring in uncertain heavy traffic load and cause high transmission delay, which makes it fail to complete real-time computing intensive tasks. As an extension of cloud computing, edge computing (EC) can reduce the pressure of cloud nodes by offloading workloads to edge nodes. In this paper, we propose an edge computing-based method. The onboard equipment on a fast-moving train is responsible for acquiring real-time images and completing a small part of the inference task. Edge computing is used to help execute the object detection algorithm on the trackside and carry most of the computing power. YOLOv3 is selected as the object detection model, since it can balance between the real-time and accurate performance on object detection compared with two-stage models. To save onboard equipment computation resources and realize the edge-train cooperative interface, we propose a model segmentation method based on the existing YOLOv3 model. We implement the cooperative inference scheme in real experiments and find that the proposed EC-based object detection method can accomplish real-time object detection tasks with little onboard computation resources.

1. Introduction

Over the years, the safety of railway transportation along the line is highly valued, but it is threatened by the failure of railway infrastructure, such as wrong signal light display, the physical environment changes caused by bad weather, and railway obstacles [1]. And the railway obstacles are especially of high frequency. As one of the research directions of intelligent rail transit, obstacle detection based on computer vision (CV) can help to detect pedestrians, vehicles, and other obstacles on the track and ensure safe operation of train systems. In case of any train control system failure, the obstacle detection system can assist manual driving and enhance the emergency treatment ability of the system.

There are many kinds of obstacle detection methods based on CV, but they are not fully applicable to rail transportation. In [1], He et al. detect obstacle with improved YOLOv4 on Jetson AgX with 256 CUDA cores and implemented 93.00% MAP (mean average precision) and 139 ms inference time. In [2], Cong and Li's method costs 670 ms in detecting pedestrians of one frame image with YOLOv2. However, these methods involve many model parameters, huge floating-point operations, and more memory. Because the calculation and storage resources of on-board devices

in rail transit are limited, GPU devices with high computing power, high power, and high performance will not be configured on the train, which fails to meet the need of real-time obstacle detection.

In order to reduce the computation burden of on-board devices, this paper proposes an EC-based method for rail transit obstacle detection, which uploads the on-board computing tasks to the edge computing server. The edge computing architecture is a distributed and layered structure, composed of cloud nodes, edge nodes, and edge devices [3]. The use of edge computing architecture has three advantages. First, the edge node can maintain data privacy. Second, the data related to obstacle detection, like model structure and model parameters, can be distributed to the edge device on the train from the cloud. At the same time, a large number of lines and trains can provide enough raw data to supplement data set in the cloud nodes. Third, edge computing can offload tasks in a regional network closer to the data source, and data generated by the edge device can be directly dealt on the edge node without being uploaded to the cloud node, thereby reducing network delays and communication costs.

At present, edge computing has many applications in the field of urban transportation and others, like [4, 5] for video analytics and [6] in urban transportation. And in rail transit field, Wang et al. proposed a collaborative cloud-edge system for real-time queries of large-scale surveillance video streams [7]. And Tong et al. designed a hierarchical tree hierarchy of geo-distributed edge cloud architecture to efficiently utilize the cloud resources to serve the peak loads from mobile devices. The ideas of the above methods are basically through model compression methods, such as kernel thinning [8], pruning [9], weight quantification [10], and network disintegration [11]. The model compression method mainly operates the parameters and structure of the model, which may lead to the risk of nonconvergence of the model. However, the core idea of our method is not task allocation, not resource allocation, but model allocation like Figure 1. By using the feature map as a link and retaining the complete structure of the model, it will not affect the convergence of network training and effect of inference.

To achieve collaborative inference with edge computing, this paper proposes a model segmentation method based on YOLOv3, which has the backbone of darknet53 (53 convolutional layers). And it uses k-means clustering to determine bounding box priors and uses binary cross-entropy loss for the class predictions [12]. According to the multiscale features, this paper segments the backbone network of YOLOv3 into three parts and rebuilds two submodels with them. These submodels can be allocated to edge node and on-board equipment for inference. Therefore, the partial computation burden is offloaded from on-board devices into edge nodes.

Combining YOLOv3 model and edge computing, the contributions of this paper are summarized as follows:

(1) We propose an edge computing architecture for distributed offload training and collaborative inference. In this structure, the train and ground server can cooperate to carry out inference

(2) We propose a method of segmenting model to split the computational loads and realize collaborative inference. Through this method, we can schedule the segmented model according to the switch strategy

(3) We evaluate training loss to verify the training potential of the submodel. And we compare the inference time on a single node with that on multinodes by the method of segmenting model

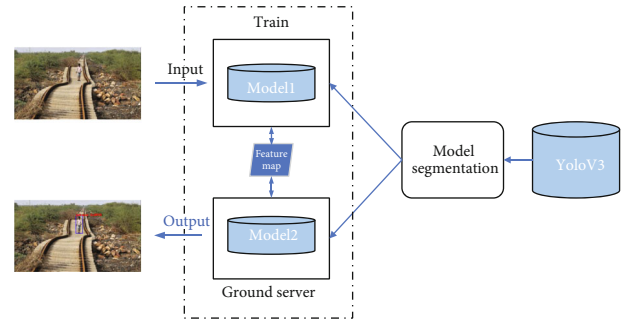

Figure 1: Our method selects the appropriate submodel through model segmentation to allocate the submodel for the train and ground server, so as to achieve the balance of resource utilization and detection time.

The rest of this paper is organized as follows. Section 2 reviews the hierarchical edge cloud architecture for obstacle detection. Section 3 provides a brief overview of the proposed model segmentation method. Section 4 describes the details of the training and inference process of submodel. Section 5 shows the inference time and MAP of the segmentation submodels.

2. Edge Computing Architecture Designed for Object Detection

Although cloud-based solutions can ensure high accuracy by using complex CNN models, they suffer from increasingly unaffordable bandwidth cost and severe query latency due to the transmission of explosively growing surveillance video data [7]. In the meanwhile, most on-board devices are resource-limited where only lightweight CNN models like mobile net can be deployed. Thus, we choose to move the most part of model from edge devices to edge nodes.

In the traditional edge computing structure proposed in [13], all edge nodes share one model, but this is hindered by delays in uploading data and downloading models. For edge computing, there exist two ways to offload computation. One is binary offloading, in which CNN models as a highly integrated task cannot be partitioned and have to be executed as a whole either locally at the edge device or offloaded to the edge node server by binary offloading. The other is partial offloading, and it allows the program to be partitioned into two parts with one being executed at the edge device and the other being offloaded for edge node execution

[14]. Based on these two methods of edge computing, we design a new computing architecture and use the segmentation method to allocate different model blocks to edge nodes and on-board devices.

In our edge computing architecture shown in Figure 2, ground servers near the base station act as edge nodes and are responsible for model training and collaborative inference. The cloud nodes are responsible for storing data sets, labeling, and updating the model data. In the meanwhile, the cloud nodes are also the brain of the whole architecture, and they monitor the running status of edge nodes and record corresponding logs. And the edge devices in the train mainly take on the task of acquiring the original images and sending the images and waiting for the detection results from the ground servers. To maximize resource utilization when the servers are idle, the ground servers of rail transit need to perform multiple tasks. Therefore, when the train moves into the range of a base station, the ground server assists the on-board devices to detect obstacles. After the train is away from the base station; the ground servers continue to perform other tasks.

The challenges that we have to face in our architecture are high transmission delay, appropriate object detection model, and model allocation method. Due to the development of the 5G communication technology and the support of millimeter wave (mm Wave), multiple-input multiple-output (MIMO), and multiaccess edge computing (MEC), the data peak rate can reach 1 GB/s, and the transmission delay is limited in the range of 1-5 ms [15]. And communication transmission time can also be compensated by less inference time of ground servers, so the influence of data transmission can be reduced. As for the object detection model, we reconstruct the YOLOv3 model, that is, a fast one-stage model.

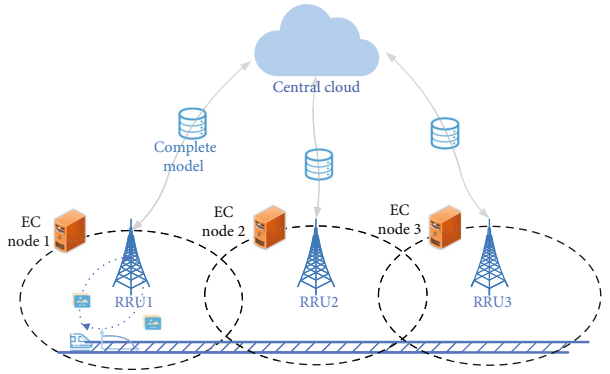

Figure 2: Edge computing architecture designed for object detection. The structure is composed of edge devices on the train and ground server as edge node and cloud node in control center. It is a distributed collaborative task execution architecture.

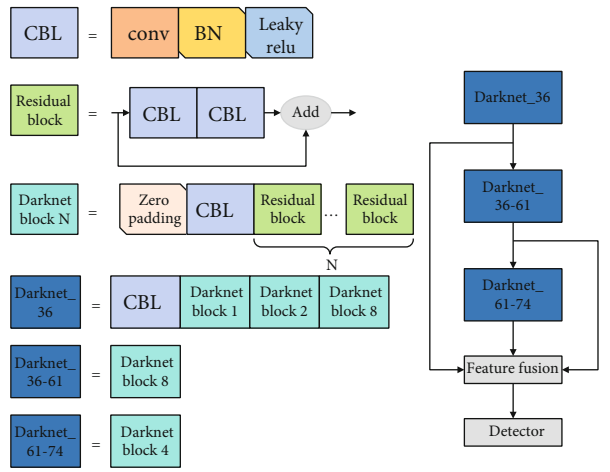

Figure 3: YOLOv3 network architecture. The left is three Darknet blocks that forms the backbone network. The network on the right is the whole backbone network of YOLOv3.

3. Model Construction

In this section, this paper adjusts the existing YOLOv3 network structure to adapt to the network segmentation method proposed in this paper and adds appropriate coupling structures to concatenate feature fusion module with detection module. In the meanwhile, different segmentation methods, including parallel and serial, are introduced in detail. Serial is single input single output, and parallel is multiple input single output.

3.1. The Basic YOLOv3 Structure. In order to meet the real-time requirements of train obstacle detection, we select a fast and accurate real-time object detection algorithm called YOLO (you only look once). YOLO is also one of the best one-stage target detection models. The structure of YOLOv3 is shown in Figure 3. In order to visualize the allocation process of sub models, we split the whole backbone network into several dark blocks, and each darknet block consists of one zero padding, one CBL (convolutional, batch normalization, leaky_relu layers) module and N residual blocks. The N equal to the Nth darknet block. The CBL module of each darknet block achieves the effect of down sampling by setting the step size of the convolution layer that reduces the feature map dimension. YOLOv3 also abandons the full connection layer, and thus, the whole YOLO network no longer limits the size of the input image in theory.

For intuitive display, we summarize the remaining components into feature fusion module and detector. In the feature fusion module, tensor concatenate and up sampling are included. Up sampling means feature map of different scales and sizes will be unified to the same shape. Then, YOLO concatenates the feature maps with the same shape and inputs it into the detector. In the detector, there include one CBL module, one layer of convolution, and NMS (non-maximum suppression).

The input of the complete YOLOv3 network is an image with standard size of (416, 416, 3); otherwise, the image will be extended or compressed into the size. After passing through the darknet network of YOLOv3, the input image is transformed into feature maps of three scales. Then, the feature fusion module absorbs semantic information of different scales and outputs with new three feature maps. Finally, after inputting these new feature maps, the detector returns the coordinates of bounding boxes and the classes of objects.

According to YOLOv3's unique network structure and different scale feature maps, it is possible to adapt to the edge computing architecture. We can connect the input and output of different darknet blocks through feature maps, and then, these feature maps can be input to the feature fusion module and be output through the detector.

3.2. Future Fusion and Detector. In this section, we elaborate the structure and function of the feature fusion module and detector in Figure 4. The function of the former is to mainly up sample the feature maps by a factor of 2 and concatenate it with others of low scales, and the function of the latter is to classify the object. The shapes of the detector's output are (1, 13, 13, 3, 85), (1, 26, 26, 3, 85), and (1, 52, 52, 3, 85). The value of first dimension is equal to the batch size, and the value of batch size defaults to 1. (13, 13), (26, 26), and (52, 52) represent the division of grid cells. 3 represents the sum of bounding boxes predicted for every grid cell. And 85 means the 80 classes of COCO data sets, the coordinate of bounding boxes (x, y, width, height) and mask indicating whether there is an object in the box. The combination use of the feature fusion module and detector will output the final detection results.

The final output of the model depends on the effect of model training. And the process of model training needs to be measured by loss function. The loss of YOLOv3 consists of the loss of coordinate, loss of IOU, and the loss of classification.

$$\text{Loss} = \text{Loss}_{xy} + \text{Loss}_{wh} - \text{Loss}_{\text{obj}} - \text{Loss}_{\text{class}}. \tag{1}$$

In this loss function, (x, y, w, h) presents the predicted coordinate of bounding boxes, and $(\hat{x}, \hat{y}, \hat{w}, \hat{h})$ denotes the labeled coordinate, means ground truth.

$$\begin{aligned}\text{Loss}_{xy} &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{obj}} (2 - w_i \times h_i), \\ &\left[(x_i - x\wedge_i)^2 + (y_i - y\wedge_i)^2\right], \\ \text{Loss}_{wh} &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{obj}} (2 - w_i \times h_i), \\ &\left[(w_i - w\wedge_i)^2 + (h_i - h\wedge_i)^2\right].\end{aligned} \tag{2}$$

λ_{coord} and λ_{noobj} are constants which are used to increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that do not contain objects. I_{ij}^{obj} denotes if object appears in the j th bounding box predictor of grid cell i [16].

$$\begin{aligned}\text{Loss}_{\text{obj}} &= \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{obj}} \left[\hat{C}_i \log (C_i) + \left(1 - \hat{C}_i\right) \log (1 - C_i)\right] \\ &+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{noobj}} \left[\hat{C}_i \log (C_i) + \left(1 - \hat{C}_i\right) \log (1 - C_i)\right].\end{aligned} \tag{3}$$

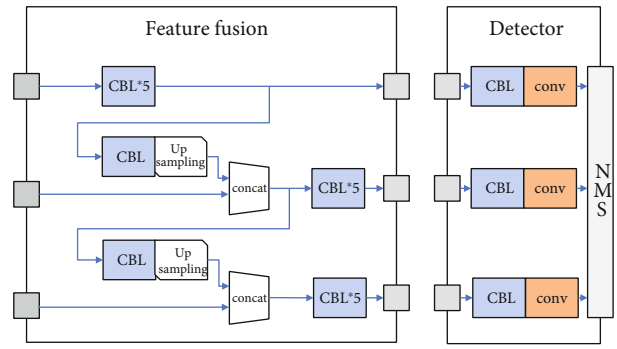

Figure 4: Feature fusion and detector. The grey squares are the input or output interface of the two module functions. And NMS is nonmaximum suppression.

The loss of object is to reflect confidence (C) error. The confidence level announces whether there is an object, and all the features of the whole object are included in the framed box. In the code, we uses binary cross entropy (BCE) instead of mean square error (MSE).

$$\begin{aligned}\text{Loss}_{\text{class}} = \sum_{i=0}^{S^2} \sum_{c \in \text{classes}} I_{ij}^{\text{obj}} [&\hat{p}_i(c) \log (p_i(c)) \\ &- (1 - \hat{p}_i(c)) \log (1 - p_i(c))].\end{aligned} \tag{4}$$

The loss of class is to reflect classification error. $p_i(c)$ denotes the probability of the class in grid cell i.

We also use the loss function to measure the prediction ability and robustness of the complete model and the submodels. The comparison results are shown in Section 5. At the same time, in order to solve the size matching problem between input and output of submodels by our model segmentation method, we add corresponding interfaces and reorganize the feature fusion and detector module like Figure 5.

3.3. Segmentation Model for Serial Inference. In this section, we will introduce the first segmentation method for edge computing. For achieving the real-time detection of a high-speed train, train and ground server cooperate to inference which offloads the computation burden of on-board device. Therefore, current obstacle detection task of train can be interrupted and transferred to the ground server. As shown in Figure 6, we call this method serial inference, and there are two nodes participating in the serial cooperative inference.

In order to properly allocate the computation burden of train and ground server, we need data to reflect the amount of computation. Thus, we use the sum of FLOPs (floating-point of operations) and parameters to quantify the computation of each darknet block, and the computation amount and parameter amount of each darknet block are counted in Table 1 of Section 5. After being assigned an appropriate submodel, the train with insufficient resources can load model parameters successfully with less computation burden. At the same time, the segmentation method can also handle the temporary shortage of train resources. Without

```
def feture_fusion_and_detector(masks=yolo_ancher_masks,classes=80,cut_point=-1):
    if cut_point == 0:
        x1 = input = Input([None, None, 1024])
         x1 = YoloConv(512, name='yolo_conv_0')(x1)
        output = YoloOutput(512, len(masks[0]), classes, name='yolo_output_0')(x1)
        return = Model(input1, (x1, outoput), name='detector0')

    if cut_point == 1:
        x1 = input1 = Input([None, None, 512])
        x2 = input2 = Input([None, None, 512])
        x1 = YoloConv(256, name='yolo_conv_1')((x1, x1))
        output = YoloOutput(256, len(masks[1]), classes, name='yolo_output_0')(x1)

    if cut_point == 2:
        x1 = input1 = Input([None, None, 256])
        x2 = input2 = Input([None, None, 256])
        x1 = YoloConv(128, name='yolo_conv_2')((x1, x1))
        output = YoloOutput(128, len(masks[2]), classes, name='yolo_output_2')(x1)
    return = Model((input1, input2), (x1, outoput), name='detector0')+str(cut_point))
```

Figure 5: Reorganized feature fusion and detector. To deal with different inputs differently, the first is to adapt to the number and shape of the input and choose convolutional layers with filters of different size.

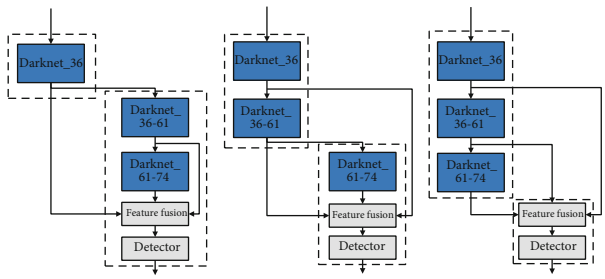

Figure 6: Segmentation model for serial inference.

Table 1: Parameters and FLOPs of different darknet blocks.

Darknet_block, module	36	36-61	36-74	FD	Total
Parameters(float)$*10^4$	320	1168	2570	2136	6195
FLOPs$*10^4$	642	2338	5144	4276	12400
Proportion	0.0518	0.1886	0.4148	0.3448	1

damaging the structure of the model, this method can still maintain the accuracy of the complete model.

The specific implementation process is as follows: the on-board device monitors its own status of GPU, memory occupancy, and communication delay with the ground server. Affected by these factors, high-speed train can choose the proper strategy to offload the obstacle detection task to the ground server. After receiving the feature map transferred from train, the ground server will match the shape of received feature map and chooses suitable layer for input. Of course, model segmentation method will update when the resource state changes dramatically.

In our method, the extra cost is the communication transmission delay of feature map. It is related to the size of feature maps and the bandwidth. We follow this formula for the specific calculation of transmission delay:

$$T_c = \frac{\text{Size}_{\text{file}}}{\text{Bd}}. \tag{5}$$

T_c represents the communication delay between train and ground server. $\text{Size}_{\text{file}}$ represents the byte size of feature map shown in Table 2. Bd represents the bandwidth. Then, we can calculate the total inference time cost of segmentation model method for serial inference. The time cost of this method is

$$T_i = T_{i1} + T_c + T_{i2} + T_{fd}. \tag{6}$$

T_i represents total inference time cost. T_{i1} represents the inference time of the submodel in the first node. T_c denotes the total transmission delay between train and ground server. T_{i2} represents the inference time of the submodel in the second node. T_{fd} represents the time cost of dealing the feature maps in feature fusion and detector modules.

During the transmission of feature maps, we choose pickle to serialize and covert the feature map into a file. Pickle is a Python tool module and implements binary protocols for serializing and de-serializing an object structure. Then, we use file transfer protocol (FTP) to send the file to the ground server. The byte size and the transmission delay of the file are shown in Table 2 of Section 5. And the experiment proved that this transmission method is effective.

Through the formula of time cost, we find out that time cost is cumulative. This also means that the ground server must wait for the train inference process to complete with the serial method. This greatly increases the inference time of serial segmentation method. Therefore, we also designed a segment-model method for parallel inference to let the ground server compete with the train for output.

3.4. Segmentation Model for Parallel Inference. In this section, we will introduce another segmentation method inspired by the idea of parallel rules. Although model segmentation method for serial inference can deal with the problems of emergency events and resource shortage, it is influenced by the limitation of linear execution. In the meanwhile, the ground server must wait for the feature map from train; otherwise, it do not output. In order to minimize the influence of communication transmission delay and maximize inference efficiency, we propose a model-segmented method for parallel inference.

As shown in Figure 7, we try to reduce the time cost of communication transmissions. And we input the image to be detected to every node and record the time stamp at the same time, so it is a competition between ground server and train, and the result of competition is the final inference time. The inference time of each one will be added to the inference time of darknet block in Table 2 and sorted. These sorted times become the indicators for switching strategy. If the node with the shortest inference time is ground server, it will be reassigned to a submodel with more computation. That is to say, the first task of this segmented method is to minimize both computation burden of on-board devices and the total inference time.

Similar to part C, we measure the inference time by recording the time when packets arrive at the feature fusion

Table 2: The byte size of output feature map and cost inference time.

Cut point num	Bytes	MB	Transfer time	Mean inference time
Cutpoint2	2769060	2.64	62 ms	26 ms
Cutpoint1	1384612	1.32	31 ms	38 ms
Cutpoint0	692388	0.66	16 ms	44 ms
Feature fusion and detector	/	/	/	31 ms
Complete model	/	/	/	75 ms

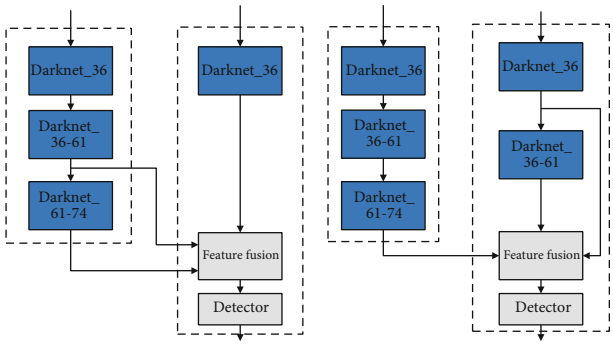

Figure 7: Segmentation model for parallel inference. The so-called parallelism means that multiple edge nodes receive the same input and infer at the same time. And the node where the feature fusion and detector modules are located is responsible for the output result.

module. The time cost of this segmented method is

$$T_i = \max\left(T_{i1} + T_c, T_{i2}\right) + T_{fd}. \tag{7}$$

Compared with segmentation model method for serial inference, the method for parallel inference includes more darknet blocks and occupy more resources. However, the inference time will be less than that of serial inference due to the competition between on-board devices and ground servers, and this is the advantage of parallel inference.

4. Model Training and Inference Process

In this section, we train the YOLOv3 network after being adjusted by our model segmentation methods and use the network for inference. We will introduce some tips and precautions about the training model and the process of collaborative inference by multiple devices.

4.1. Model Training Process. In this section, this paper elaborates on the training process of the complete model and three submodels. The first submodel consists of darknet block 36. The second consists of darknet block 36 and 36-61. And the third consists of all three kinds of darknet blocks.

And YOLOv3 uses feature pyramid networks (FPN) to integrate multiscale feature information and achieve the detection of different size objects [12]. The construction of pyramid we use involves a top-down pathway. The top-down pathway hallucinates higher resolution features by up sampling spatially coarser, but semantically stronger, feature maps from higher pyramid levels [17]. Thus, we can detect obstacles of different sizes, and the unique pyramid feature structure makes it possible for model segmentation and distributed training three submodels.

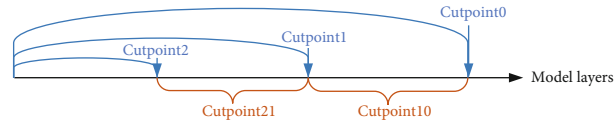

Figure 8: The numerical meaning of cut point.

However, only through individual training submodels, we cannot obtain and update the weights of the convolution layers in feature fusion and detector modules. Feature maps of different scales need to pass a layer of convolution before concatenating. When evaluating loss, three feature maps need to pass through two convolution layers (the components of detector). When the submodels are trained separately, the coupling structure among the submodels, such as the feature fusion module, contains part of the weight parameters. Therefore, the outputs of different scales are coupled or related, and the weight parameters of convolution layers before concatenating and the classification detector should be trained in advance. Thus, we freeze the weights of feature fusion modules and detector after training the whole YOLO model. Next, we try to train three submodels, which are composed of different darknet blocks and the loss of training data will be given in Section 5.

As for the allocation of training tasks, there are two solutions in our edge computing architecture. One is that the cloud nodes are responsible for training the whole model, and each edge node loads its own submodel weights through the weight of complete model. The other is that different submodels share the parameters of a complete model, which is trained locally and collected in the cloud, like federal learning. Of course, in order to offload cloud pressure and reduce the time of model training, we can train three submodels in parallel with solution 2. But their weights are not shared, and the parameters of feature fusion module between submodels will not be updated; we do not recommend the solution 2. At last, we select the solution 1 to train the model.

4.2. Model Inference Process. In this section, we will introduce the process of inference in detail and the strategy of allocating submodels. For better describing the segmentation method of the model, three cut points are inserted before the output of each darknet block. The cut-points are numbered by 2, 1, 0, and -1 as shown in Figure 8. The number of each cut point represents the following meaning:

(i) -1 means outputting all feature maps

(ii) 0 means outputting small-scale feature map, which is the third cut point

(iii) 1 means outputting medium-scale feature map, which is the second cut point

Input: $S_k = (S_{train}, S_{server})$,
Input_size = (416, 416, 3)
featuremap_size= [(13, 13, 1024),(26, 26, 512),(52, 52, 256)]
Output: T_k, k=0,1,2,3...
k=0
model ⟵ switch_sub_model(S_k);[1]
load_init_weights(*model*, full_model_weights);
while node received images **do**
  Record the time of receiving image;
  if shape(images) = Input_size **then**
    featuremap ⟵ inference(*model*, images)[2];
    use FTP to send featuremap to next node;
  else
    if shape(images) in featuremap_size **then**
      featuremap ⟵ images;
    end
  end
  if length(*featuremap*)=3 **then**
    feature_fusion_and_detector(*featuremap*);[3]
    Record the time of outputing detection result;
    T_k=t_2-t_1
    k=k+1
  end
end

Algorithm 1: The inference process of on-board device and ground server.

(iv) 2 means outputting big-scale feature map, which is the first cut point

We also use 21, 10, or 20 to intercept the middle layers of the complete model. And the inference process is divided into 4 parts. First, appropriate initial strategy S_0 is needed to select the cut point of the complete model. The influencing factors of the strategies include max performance of server, resource occupation, and communication delay. The initial strategy is distributed from cloud nodes to edge nodes and on-board devices, and we call it cloud edge collaboration. Second, on-board device and ground server load initial model according to initial strategy, and ground server waits for the feature map transferred from train after loading. The whole inference process is triggered when the train collect the images. Third, if on-board device uses resources excessively during inference, it will select the nearest cut point to exit the current inference process and send the feature map inheriting task progress to ground server. Of course, if all goes well, the equipment and train will continue to implement the established strategy. Last, after detecting an image, the inference time will be collected by the cloud. The calculation of inference time is based on the formulas in Section 3. And the data will have an effect on updating the next strategy S_2.

5. Results and Analysis

We carried out the experiment on GTX 1660ti and used GPU acceleration. The basic program environment includes tensorflow-gpu-2.0, cuda-10.0, and cudnn-10.0. The focuses of our experiment are the influence of various segmentation methods and whether collaborative inference can be realized without sacrificing precision.

Table 1 shows that different blocks of backbone net have different FLOPS and parameters. With the data, the idle resources can be reasonably matched with the corresponding parameters to maximize the efficiency of obstacle detection. In this table, darknet block 36-74 takes up the most resources, and the next is future fusion and detector modules. Thus, our strategy allocates these two blocks to the ground servers as much as possible.

In Table 2, we measured the size of different scale feature map, their transfer time by the network bandwidth of 500 Mbps, and their corresponding average inference time. It is found that the inference time of two same GPU devices is only 13% more than that of a single device. If the performance of one device is better, the time gap will be narrowed. The conclusion shows that it is feasible to offload the calculation pressure of on-board devices by increasing part of inference time.

During the training process, we evaluated the prediction ability of the complete model according to the loss function of part B in Section 2. As shown in Figure 9, there are three different outputs of different scales, and we found that the complete model we trained has pretty good prediction ability and robustness.

At the same time, we compare the inference time and resource occupancy of train and server shown in Figure 10. We find that parallel model segmentation schemes cost less inference time than that of parallel model segmentation schemes. But the parallel scheme takes up more train resources than the serial scheme. In general, model

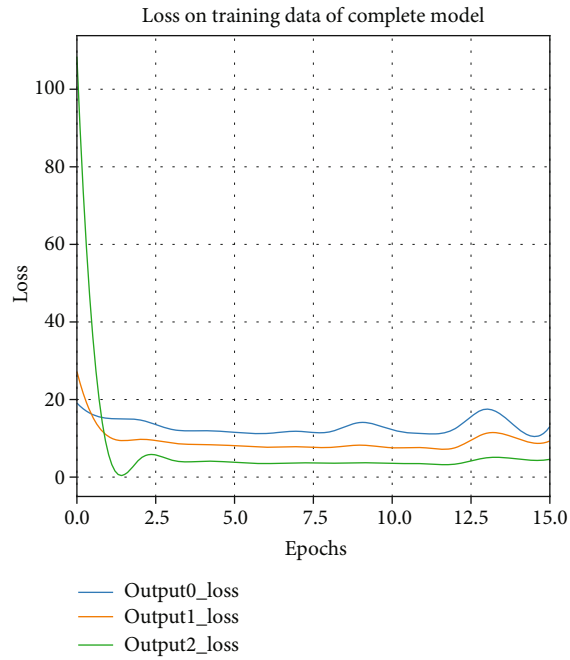

Figure 9: Loss of training complete model. Output0_loss is the loss of small-scale inference results, output1_loss is the loss of middle-scale inference results, and output2_loss is the loss of big-scale inference results which are also the final output results.

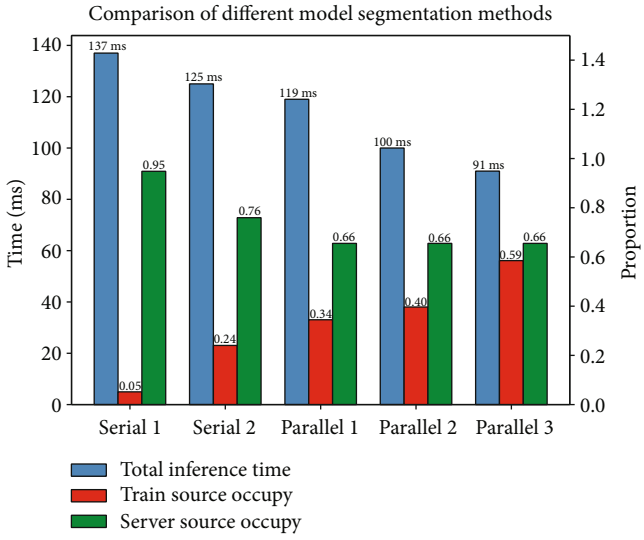

Figure 10: Comparison of different model segmentation methods.

segmentation method can effectively offload the computation burden of on-board devices at the cost of increasing inference time. If the GPU performance of the ground server is better, the total inference time can be reduced.

At last, we tested some trackside scenes with the third model segmentation method for parallel inference shown in Figure 11. The image in the lower right corner is taken from one frame of the video stream, and others are ordinary images. The results show that the segmentation model can detect small-scale, medium-scale, and multiple objects well.

Figure 11: Object detection results. We used the open-source coco data set with 80 classes to train the model, and the images to be detected only include trains and people two classes.

6. Conclusion

Research shows that the performance of the complete model in on-board devices is lower than that of collaborative inference. In the edge computing architecture, we have designed different model segmentation method models for collaborative inference and proved the validity of these methods.

We have offer two kinds of viable model-cutting methods to combine edge computing with object detection. One is for serial inference, and it takes up the same computing power as the original model. Thus, we can get the inference result with the least resource cost. And the other is for parallel, and it allows on-board devices and ground servers to do competitive inference to minimize the inference time.

The possible future work is to combine reinforcement learning or other similar methods with the model segmentation method of this paper and determine the execution of the segmentation strategy based on the performance of the detection effect. While maximizing the use of train and ground server resources, it also improves detection efficiency.

Data Availability

Image data sets for model training and testing reported in this manuscript have been deposited with the Microsoft COCO and PASCAL VOC. Copies of these data can be obtained free of charge from https://pjreddie.com/projects/pascal-voc-dataset-mirror/ and https://cocodataset.org/#home. And the specific versions of utilized data sets are PASCAL VOC 2012 and COCO 2017.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This paper was supported by the Fundamental Research Funds for the Central Universities (2021QY004).

References

[1] D. He, Z. Zou, Y. Chen, B. Liu, X. Yao, and S. Shan, "Obstacle detection of rail transit based on deep learning," *Measurement*, vol. 176, article 109241, 2021.

[2] Z. Cong and X. Li, "Track obstacle detection algorithm based on YOLOv3," in *2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 12–17, Chengdu, China, 2020.

[3] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, San Francisco, CA, USA, 2016.

[4] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: a mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018- IEEE Conference on Computer Communications*, pp. 1421–1429, Honolulu, HI, USA, 2018.

[5] S. Wan, S. Ding, and C. Chen, "Edge computing enabled video segmentation for real-time traffic monitoring in Internet of vehicles," *Pattern Recognition*, vol. 121, p. 108146, 2021.

[6] C. Chen, B. Liu, S. Wan, P. Qiao, and Q. Pei, "An edge traffic flow detection scheme based on deep learning in an intelligent transportation system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1840–1852, 2021.

[7] R. Dwivedi and L. Mackey, *Kernel Thinning*, 2021.

[8] P. Zhang, Y. Zhong, and X. Li, "Slimyolov 3: narrower, faster and better for real-time UAV applications," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Seoul, Korea, 2019.

[9] R. Krishnamoorthi, *Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper*, 2018.

[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *Computer Science*, vol. 14, no. 7, pp. 38-39, 2015.

[11] J. Redmon and A. Farhadi, "YOLOv 3: an incremental improvement," 2018, https://arxiv.org/abs/1804.02767.

[12] S. Wang, S. Yang, and C. Zhao, "Surveiledge: real-time video query based on collaborative cloud-edge deep learning," in *IEEE INFOCOM 2020- IEEE Conference on Computer Communications*, pp. 2519–2528, Toronto, ON, Canada, 2020, https://arxiv.org/abs/2001.01043.

[13] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and efficient object detection in edge computing: challenges and solutions," *IEEE Network*, vol. 32, no. 6, pp. 137–143, 2018.

[14] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: the communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[15] J. Zhao, J. Liu, L. Yang, B. Ai, and S. Ni, "Future 5G-oriented system for urban rail transit: opportunities and challenges," *China Communications*, vol. 18, no. 2, pp. 1–12, 2021.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, June 2016.

[17] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, Hawaii, July 2017.