Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 8659679, 11 pages
https://doi.org/10.1155/2023/8659679

Research Article

Edge Intelligence-Based OCS Fault Detection in Rail Transit Systems

Xuetao Shi,[1] **Hongli Zhao**,[1] **Hailin Jiang**,[1] **Huijun Zuo**,[2] **and Qiang Zhang**[3]

[1]*Electronic Information Engineering, Beijing Jiaotong University, Beijing 100044, China*
[2]*Unit 96901 of PLA, Beijing 100094, China*
[3]*Unit 61741 of PLA, Beijing 100094, China*

Correspondence should be addressed to Hailin Jiang; lhjiang@bjtu.edu.cn

Received 17 August 2022; Accepted 28 September 2022; Published 11 April 2023

Academic Editor: Lei Liu

The Overhead Contact System (OCS) is critical infrastructure for train power supply in rail transit systems. OCS state monitoring and fault detection are indispensable to guarantee the safety of railway operations. The existing human-based OCS state monitoring and fault diagnosing method has some inherent drawbacks, such as poor real-time capability, low detecting precision, and waste of human resources. Edge Intelligence (EI) can perform complex computing tasks offloaded from trains within a little delay, and it is believed to help empower the OCS. In this paper, we propose an EI-based OCS state monitoring and fault detecting system. The latest Computer Vision (CV) model YOLOv5s is used to detect the OCS faults using the collected images. Edge Computing (EC) is used to perform the CV model inference. The EC system receives the OCS images taken by the train cameras and calculates the real-time fault detection results. The consistency and scalability of running jobs on edge devices are also addressed in our approach. Extensive experimental results demonstrate that the proposed EI-based system can detect OCS faults in real-time. The adopted YOLOv5s achieves a high fault detection rate, outperforming other models.

1. Introduction

During the train operation, the Overhead Contact System (OCS) is applied to obtain power along the railway. However, OCS is mostly erected in the open air but lacks shielding protection facilities that easily hung foreign objects, which result in train delay or other safety accidents. Therefore, it is desirable to take effectual maintenance measures to identify OCS faults accordingly.

Concerning the development of OCS faults diagnosing, existing solutions can be categorized into two types: artificial cognition and image processing. At present, it mainly relies on on-site manual inspection, which is of great workload and low efficiency with long distribution, in bad weather or other abnormal factors. Besides, manual vision is difficult to capture imperceptible OCS elements, whereas it needs to build an overhead shelf contact inspecting. Thus, it is natural to shift the manual method to the Computer Vision (CV) method.

The CV algorithm witnessed the rapid development of deep learning, so a great deal of image processing is proposed to identify objects, including but not limited to Faster R-CNN built by Ren et al. [1], SVM (support vector machines) raised by Cortes and Vapnik [2], and YOLO (You Only Look Once) advanced by Redmon et al. [3]. There are kinds of research on fault diagnosis by object detecting, but the latest CV model YOLOv5s (small) is adopted in this paper to detect OCS faults, with high detection accuracy and fast speed (up to 140 frames per second). In the past few years, the remarkable advancement of emerging technologies in low-cost cameras able to acquire high-definition images has become an infrastructure to tackle these problems. In view of the convenience of image acquisition, it is feasible to use target detection for fault diagnosis.

Recent developments in the field of Edge Intelligence (EI) have led to increasing interest in researchers from both academia and industry, e.g., International Electrotechnical Commission described EI as a process when the data is

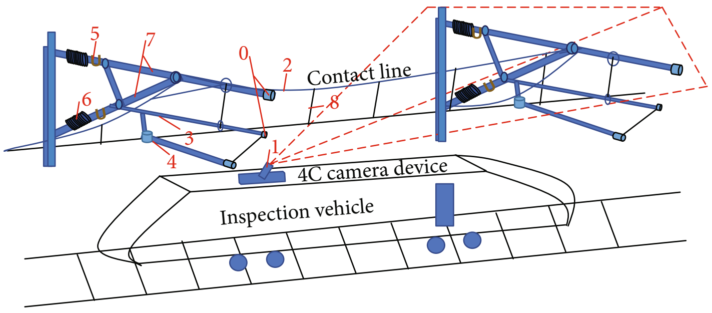


Figure 1: Nine features of OCS (0: Guanmao; 1: Xuanzhuanshuanger; 2: Chenglisuozuo; 3: Dingweixianja; 4: Dingweizhizuo; 5: Ubaogu; 6: Jueyuanzi; 7: Wanbidizuo 8: Diaoxuan).

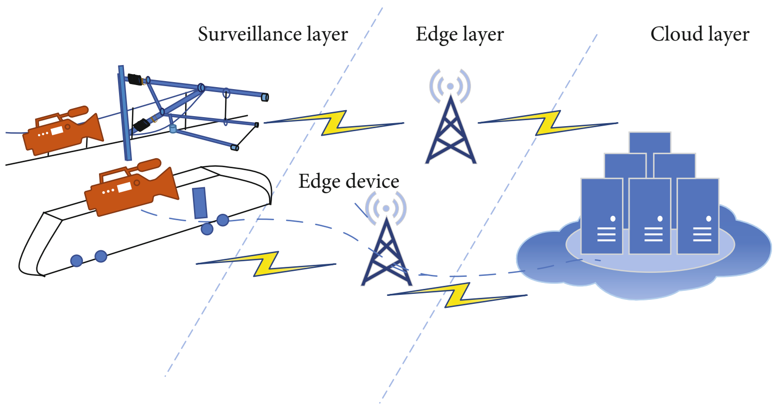


Figure 2: The OCS structure with three functional layers.

Table 1: The main assignment of the three components.

	Component names	Component action
Cloud layer	CloudCore	Send events and commands of the cloud to the edge and receive status and event reported by the edge terminal.
Edge layer	EdgeCore	Receive events and instructions issued by the cloud, execute them, and report the status and event at the edge to the cloud.
Container operation	Docker	Currently, KubeEdge supports Docker by default.

acquired, stored, and processed with machine learning at the network edge [4]. However, traditional cloud computing depends on network latency and limited throughput. Besides, the comes and goes of data transmission affect the safety of railway. Recent evidence suggests that Edge Computing (EC) can perform complex computing tasks offloaded from trains within a little delay and achieve the OCS detection with full railroad coverage but not upload to the cloud center. To implement the real-time monitoring OCS, we propose a solution to deploy OCS to EI with this inspiration.

In this paper, the adopted EC system is used to perform the CV model inference, capturing the OCS images taken by surveillance devices outside the train and YOLOv5s algorithm calculating the fault detection results in real time. The main contributions of this work are as follows:

(i) We deploy a consistent, easy-to-deploy, and low-cost EI platform for OCS fault diagnosis. It greatly helps to minimize the computational time and effort for sending the collected data from cameras to remote cloud servers

(ii) We propose a novel dataset composed of substantial amounts of high-definition images. It contains labeled corresponding OCS information, gathered from the real rail transit. To the best of our knowledge, it is the first to contain 9 frequently faulty OCS components, as seen in Figure 1

(iii) We design a CV-based real-time monitoring system using YOLOv5s for OCS recognition. YOLOv5s makes it possible to capture the essential details

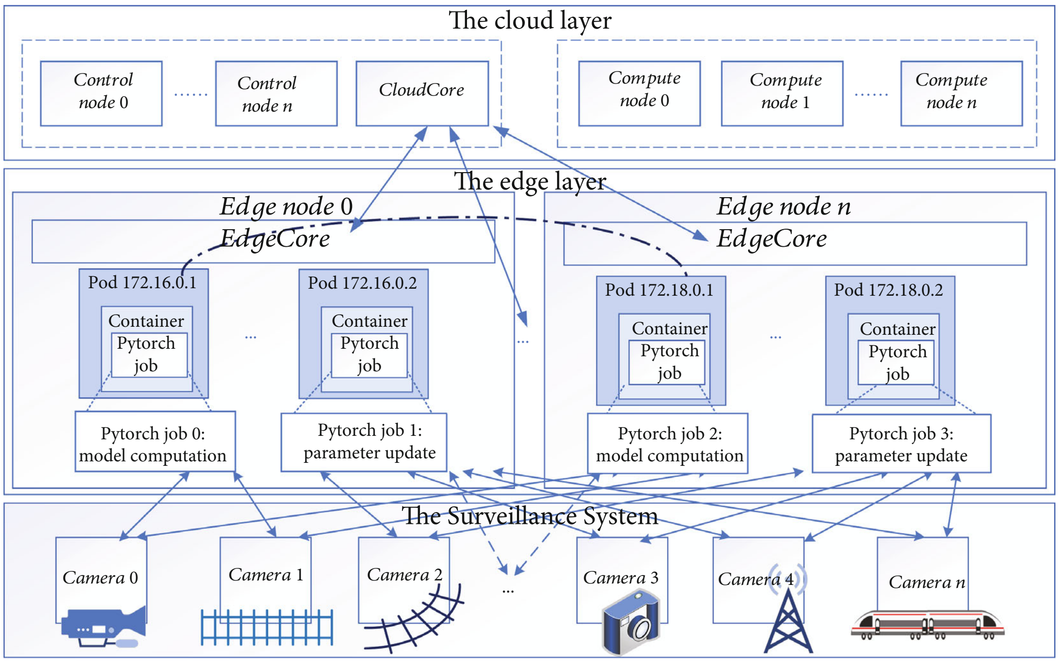
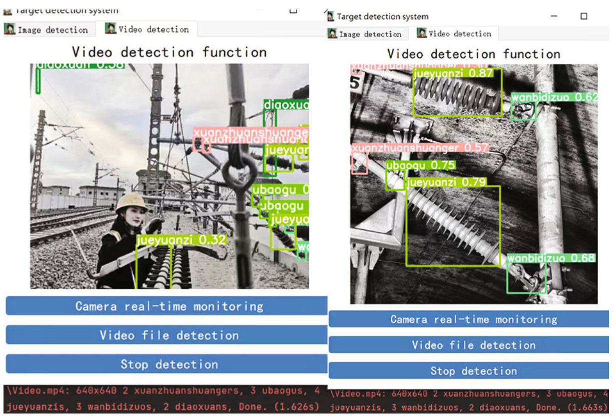


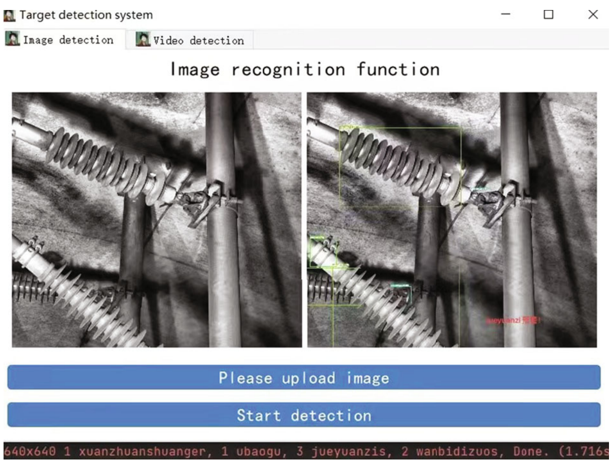
FIGURE 3: Logical structure for EC: cloud and edge collaboration; edge and end collaboration; cloud, edge, and end collaboration.



```
\Video.mp4: 640x640 2 xuanzhuanshuangers, 3 ubaogus, 4
jueyuanzis, 3 wanbidizuos, 2 diaoxuans, Done. (1.626s)
```

```
\Video.mp4: 640x640 2 xuanzhuanshuangers, 3 ubaogus, 4
jueyuanzis, 3 wanbidizuos, 2 diaoxuans, Done. (1.663s)
```

FIGURE 4: Real-time monitoring identification using webcam.



```
640x640 1 xuanzhuanshuanger, 1 ubaogu, 3 jueyuanzis, 2 wanbidizuos, Done. (1.716s
```

FIGURE 5: Object detection results in OCS images.

```
Require: Images, videos, or live video in camera detection
Ensure: The detected OCS components and the total time
1: def model_load: Model initialization
2: def initUI(self): Interface initialization
3: if $source == "fileName"$ then
4:     $Upload$ the $image$ and $then$ $test$ $it$
5: else
6:     $resize\_scale = self.output\_size/im0.shape[0]$
7:     $Save\ inference\ images$
8: end if
9: while $path, im, im0s, vid\_cap, s$ in $dataset$ do
10:     if $len(im.shape) == 3$ then
11:         $expand$ for $batch$ dim
12:         $Second\ stage\ classifier(optional)$
13:     else if $Rescale\ boxes\ from\ img\_size$ to $im0\ size$ then
14:         $print\ string$ and $normalization\ gain$
15:         $Print$ and $write\ results$
16:     end if
17: end while
18: def closeEvent(self, event): $Interface\ closing\ events$
19: def close_vid(self): $Video\ Reset\ Event$
20: Set tmp file to put the intermediate processing results
21: sys.exit(app.exec())
```

ALGORITHM 1: Real-time monitoring interface.

for OCS faults, and the precision outperforms the existing work

The rest of the paper is organized as follows. In Section 2, we review the related work. Section 3 introduces the implementation of EC and the real-time OCS monitoring system. The proposed OCS dataset and the CV-based YOLOv5 for OCS detection are described in Section 4.

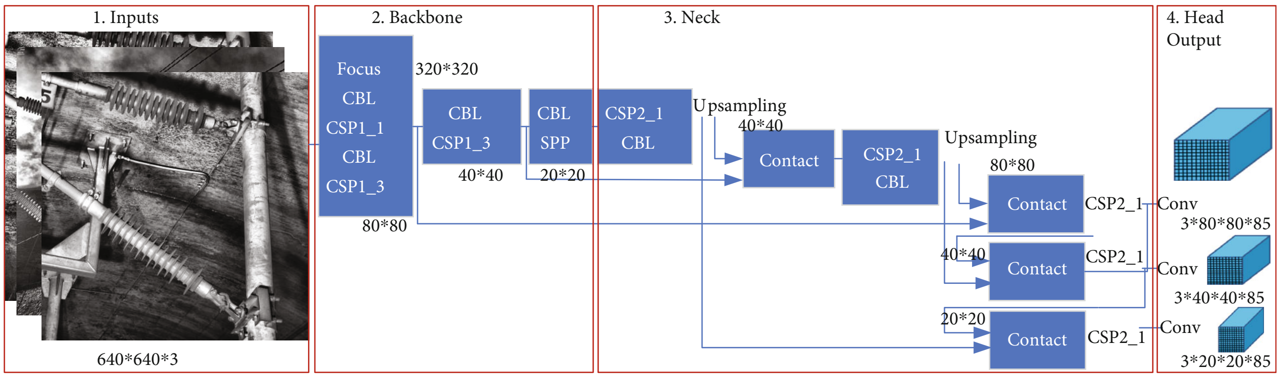


FIGURE 6: The Detailed Structure of The YOLOv5s for OCS.

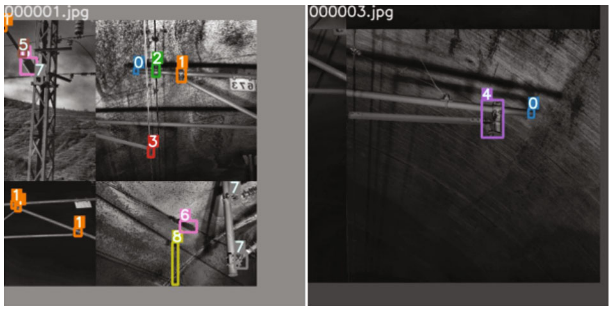


FIGURE 7: Adopt IOU_nms to filter BBoxes.

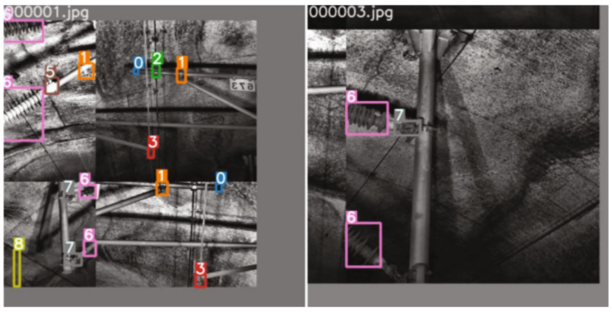


FIGURE 8: Adopt DIOU_nms to filter BBoxes.

TABLE 2: The statistics of handcrafted dataset.

	Images	Labels
Training	894	894
Validation	281	281
Testing	200	\
Total	1375	1175

FIGURE 9: The raw data samples.

Experiments and results are reported in Section 5. In Section 6, we conclude the paper.

2. Related Work

Generally, OCS images are obtained by 4C acquisition devices installed on the top of inspection vehicles, as shown in Figure 1. It's said that more than 30 cameras are installed on the train roofs. Each time a pole is passed, it triggers front and rear support devices. Each camera shoots a different OCS position along the railway line.

With emerging high-definition cameras and CV algorithms, image progressing becomes one of the most crucial instruments to realize automatic OCS fault diagnosis. Cheng proposed an improved Faster R-CNN into the abnormal diagnosis among 5 features, which proved the validity of the model in rail transit [5]. The well-modified model can show good performance, but cannot be fully employed in OCS-specific scenarios and needs further improvement. And the clevis was precisely located based on the same model, and missing fault detection of split pins was achieved by multiple linear SVM classifiers [6]. In [7], the researchers used YOLOv3 to classify the pantograph faults, with the training of 20000 images and 3 high-definition videos captured by noncontact cameras. A detection model using YOLOv3 is also created in [8] to detect the insulator faults. As we can see, the feature extraction in previous research is usually manually controlled, which is difficult for models to obtain satisfactory robustness. The focus of such studies remains narrow, dealing only with a certain component or a class in OCS, which have great limitations. Different with above all, the adopted CV algorithm can not only track the target accurately and ensure real-time performance, but also identify nine OCS components at the same time [9].

EC has been envisioned as a likely emerging distributed computing paradigm, which intends to transfer the computing effort from far-end cloud servers to the network edge, to satisfy the requirements of real-time OCS detection and location-aware data processing [10]. The EC system receives OCS images captured by train cameras and calculates real-time fault detection results, which are then compared with the sample images through a series of processing of the feature quantities in the edge device [11]. EC is very suitable for applications in ultrareliable and low-latency communication

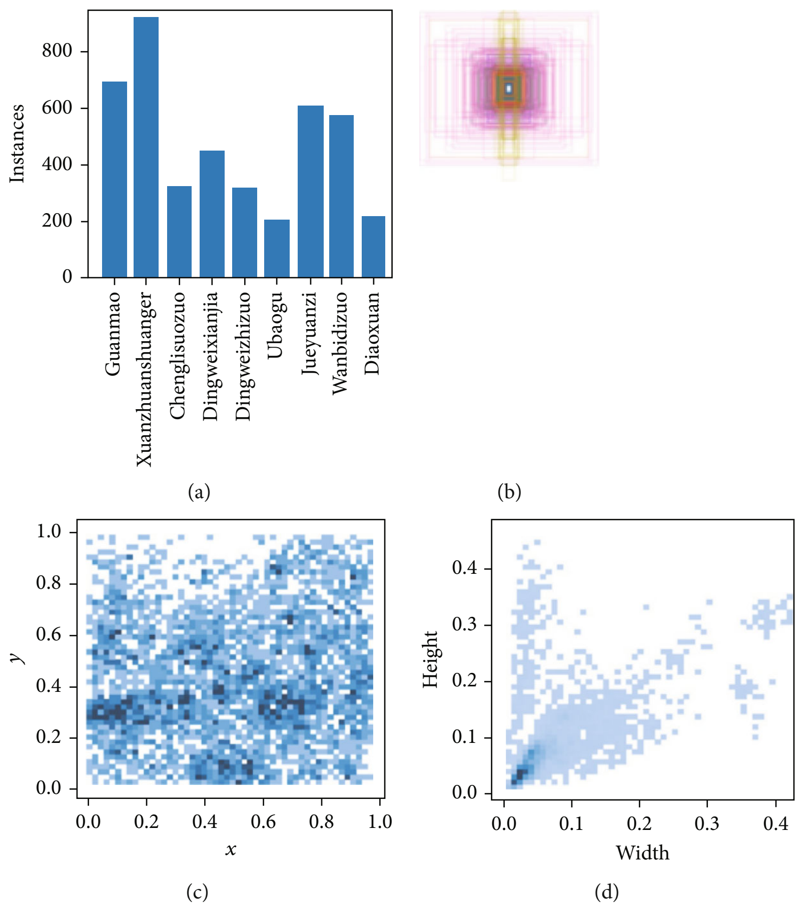


FIGURE 10: The labels of handcrafted dataset.

TABLE 3: Deploy EC system host configuration.

	Cloud node	EC node
Operation system	Ubuntu 21.04 64-bit	Ubuntu 21.04 64-bit
IP	172.20.0.3	172.20.0.1
CPU/RAM	2vCPU\8GiB	4vCPU\16GiB
Disk	60GiB	20GiB
Workload	K8S, Docker, CloudCore	Docker, EdgeCore
Version	K8S v1.16.2	KubeEdge v1.22

scenarios in smart railways [12]. More and more intelligent railway operations require instant processing, transmission, and computation of big data. With a long distributed distance, a wide range, and a real-time performance to recognize and process a particularly large number of images, it is desirable to identify the OCS faults and build EI architectures.

3. The EI-Based OCS Fault Detection

In this section, we construct a detailed introduction to our edge-based OCS faults checks from a system perspective. Our system consists of three functional layers, surveillance layer, edge layer, and cloud layer (Figure 2). This working pattern is illustrated as follows. First, the cameras collect

TABLE 4: Confusion matrix.

	Actual positive class	Actual negative class
Predicted positive class	*tp* (true positive)	*fn* (false negative)
Predicted negative class	*fp* (false positive)	*tn* (true negative)

the desired data from the 4C cameras deployed on the inspection vehicle or mounted along the railway with a panoramic view. Second, once the data is gathered, the built YOLOv5s model will load from the trackside edge servers, and instantly preprocess, to realize real-time automatic inspection. The cloud is used to store images and analyze results [13]. This technique removes redundant information of data so that some or all data analysis can be migrated to the edge. Thereby, it can reduce the calculation, storage, and network bandwidth requirements of the cloud center and improve the speed and efficiency of data analysis. Using available surveillance in OCS, we mainly focus on the interaction between the endpoints and the cloud servers.

3.1. The Cloud Layer. In this work, we formulate two nodes to the deployment of the EC system, i.e., choose Kubernetes (K8S) for the cloud component and KubeEdge for the edge

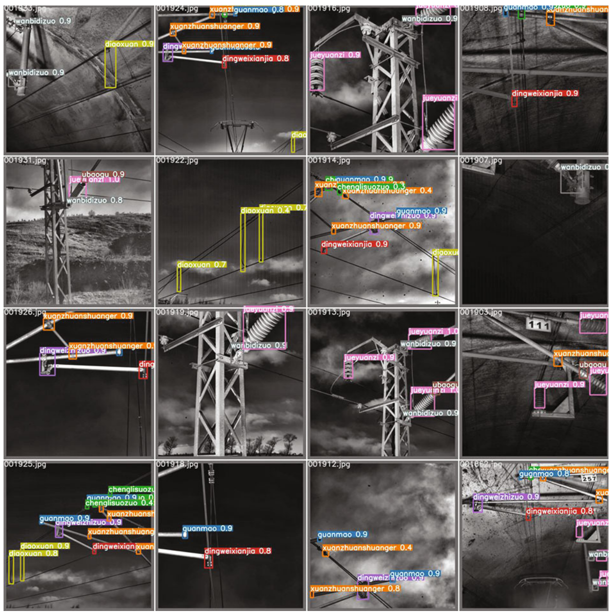


Figure 11: Detecting results.

component, each of which runs as illustrated in Table 1. Users can arrange pods with KubeEdge, manage devices, and monitor pods and device status on edge nodes just as they are in a traditional K8S cluster. There are two main things to perform in the cloud layer. One task is to manage K8S nodes, and the other is to train CNN models. K8S runs containers as pods when going through Docker containers. The logical architecture highlights the interaction and collaboration among the cloud, edge, and end parts of the EC system, as seen in Figure 3. K8S control node follows the original data model in the cloud to keep the original control and data flow unchanged. K8S can manage nodes running on KubeEdge just like normal nodes. Moreover, the reason why KubeEdge can run on edge nodes with limited resources and uncontrollable network quality is that it realizes the sinking of the orchestration and containerization of the cloud APPs through CloudCore and EdgeCore based on the K8S control nodes [14]. The CloudCore is responsible for monitoring the instructions and events of the Kubernetes control node and sending them to the EdgeCore. At the same time, it submits the status information reported by the EdgeCore to the control node in Kubernetes. The EdgeCore is responsible for receiving the commands and event information from the CloudCore in the cloud part and executing the Kubernetes orchestration containerized application.

3.2. The Edge Layer. KubeEdge is an open platform built on Kubernetes, which can extend the ability of K8S to orchestrate containerized APPs to edge nodes and provide basic support for network, APPs deployment, and data synchronization between cloud and edge. The data, composed of images or videos, is sent to the edge layer for further processing. The images are usually subjected to preprocessing such as classification and prediction using bounding box (BBox) and confidence score. After the images features are extracted, the processing results are sent to the cloud layer, and the cloud collaboratively manages the edge together and stores the data sent from the edge layer. Same as K8S, KubeEdge is also deployed by containers. The implemented KubeEdge nodes management module employs various load balancing strategies to perform the dynamic job allocation and achieve better resource utilization.

To cope with a large number of inputs and the convolutional neural network (CNN) parameters, Pytorch jobs are put in different containers for detecting, as shown in Figure 3, which is called data parallelism. The data parallelism to train CNN in parallel is to replicate data on each device and run each training step simultaneously on all replicas, each time using a different batch of data. Then the gradients computed by each replica are averaged, and the results are used to update the model parameters. Namely, it refers to distributing the data across multiple cores. To make Pytorch jobs operation smoothly on the Kubernetes (K8S) nodes, one of the most important parts is how to run computationally complex tasks with relatively limited resources. It needs a huge space for the storage of collected OCS images, in order to fully load the CNN model and all data samples into the local memory. PyTorch provides simple functions that enable easy but efficient parallel GPU computing with only a few lines of codes: *nn.parallel.data_parallel(module,inputs,device_ids=None, dim=0,output_device=None,module_kwargs=None) torch.nn.DataParallel(-module,device_ids=None,dim=0, output_device=None)*

We complete the visual interface and integrate it to the edge side, which is designed by Pyqt5. We have three functions: image recognition, video identification, and real-time monitoring identification, in Figures 4 and 5. In the black box, image recognition function identifies 7 parts in 1.7 s, and the video detection function identifies 15 parts in 1.6 s. With the model well trained, we obtain such good results in the interface, to facilitate client monitoring. The functions are shown as in Algorithm 1.

4. The YOLOv5-Based OCS Fault Detect Model

Researchers choose to use image types for analysis, interpreted through physical probability models, and finally, a CV algorithm is formed, which has proved that CV is effective and efficient for processing images [15]. In this section, our approach is presented in detail. In order to explore the attributes of the proposed dataset, we adopt a CV algorithm based on YOLOv5 deployed in the edge servers, as shown in Figure 6. The feature information of the inputs is extracted by CNN, and the image is divided into N × N grids. And the model was deployed in the edge layer. When the center of an object falls on a grid, the grid is responsible for predicting what it is. In the field of object detection, YOLOv5, a typical algorithm in deep learning, includes three components: backbone, neck, and head parts. The deeper network, the deeper backbone, the more complicated calculation will be. In this paper, YOLOv5s is selected with the minimum pretraining network structure and speed while meeting the accuracy requirements [16].

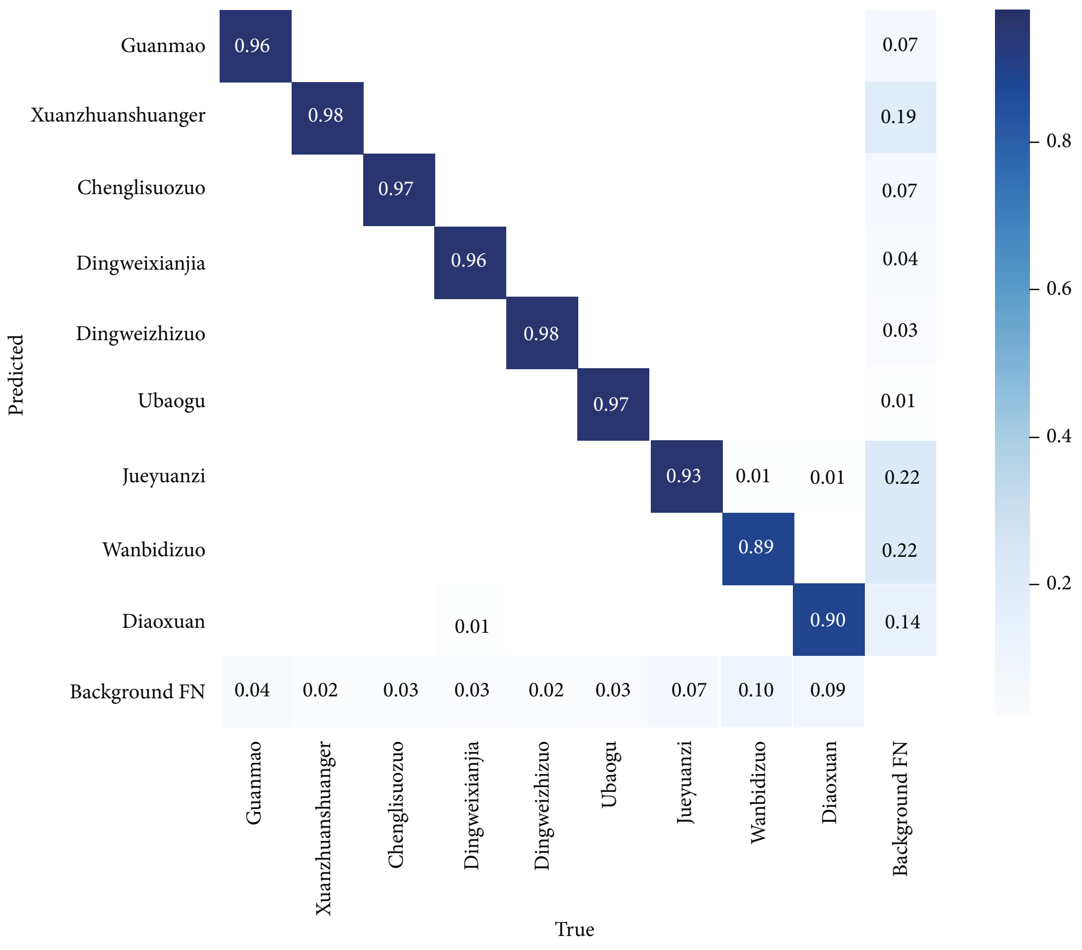


FIGURE 12: The confusion matrix for OCS classification.

TABLE 5: The results of 9 elements based on YOLOv5s.

Class	Labels	*P*	*R*	mAP@.5	@.5:.95
All	1317	92.6%	93.7%	94.5%	0.548
Guanmao	200	96%	95.3%	95.4%	0.513
Xuanzhuanshuanger	289	91.8%	94.8%	95%	0.463
Chenglisuozuo	97	94.9%	95.7%	97.4%	0.646
Dingweixianja	147	95.5%	95.9%	94.4%	0.409
Dingweizhizuo	97	97%	96.9%	98.5%	0.7
Ubaogu	58	97.5%	96.6%	97.2%	0.669
Jueyuanzi	185	87.9%	91.9%	93%	0.751
Wanbidizuo	163	82.6%	87.1%	87.2%	0.308
Diaoxuan	81	90%	88.9%	92%	0.472

Backbone is a CNN that aggregates different images at a fine granularity to form image features. Neck is a series of network layers for blending and combining image features and passes them to the prediction layer. The head part head makes predictions on image features, generates bounding boxes (BBoxes), and predicts categories. In this work, an OCS classification model based on YOLOv5 object detection network named OCS-YOLOv5 is designed.

For data augmentation in OCS images, we enabled mosaic enhancement and used random crops with 4 images between 0.25 * img_size, 0.75 * img_size pixels, scale factor between 0.9 and 1, and shear between 1 and 10, image translation (± fraction) between 0.9 and 1, image perspective (± fraction), range 0-0.001, image flip up – down (probability) = 1.0, image flip left-right (probability) between 0 and 1.0, and image mixup (probability) = 1.0. Note that these parameter values were determined based on experiments performed on the validation set. We used Mosaic [17], which is a well-known Python library for image augmentation, to apply these transformations: stretching image to modify size, using gamma transform, randomly select the contrast and brightness of the images, adaptive histogram equalization for the contrast-constrained case, motion blur algorithm to legend images, median filtering, normalizing, and so on.

The new images are scaled and stitched to form new images with rich backgrounds and different shapes from the original OCS. Therefore, the transformation makes the OCS dataset more diverse, increases the robustness of the model, makes the model more generalizable, and finally enables the OCS system to cope with more complex real-world situations.

In the postprocessing process of target detection, the screening of many target boxes usually requires non-

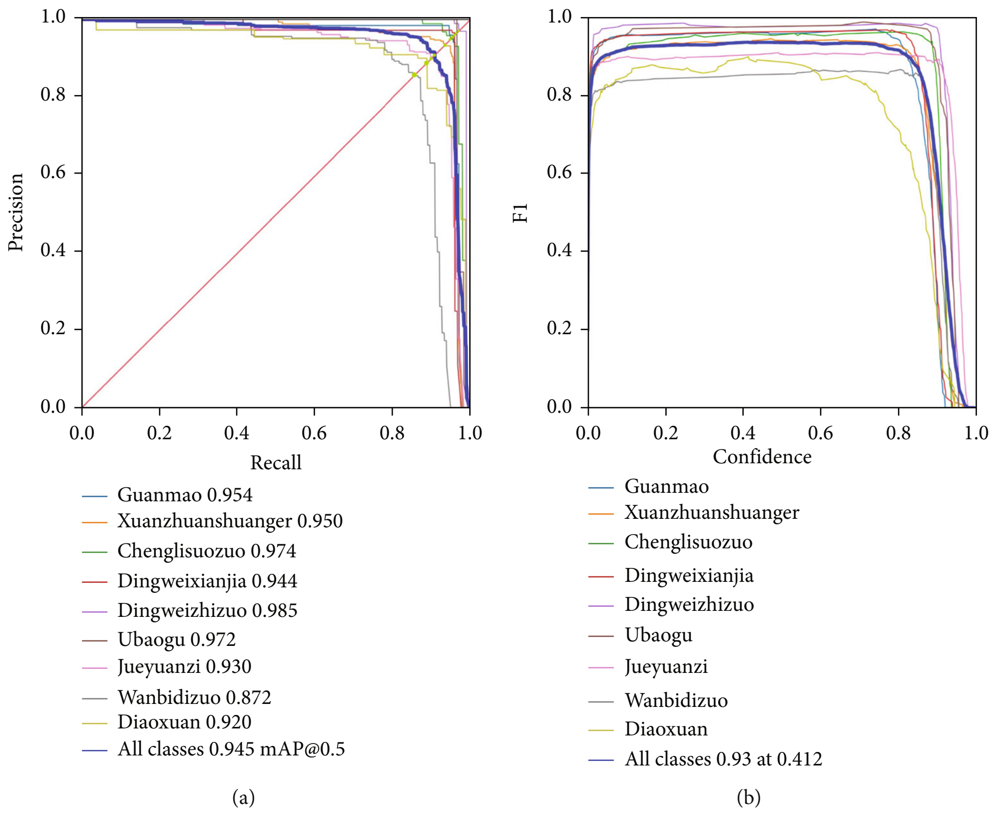


FIGURE 13: YOLOv5s: (a) PR_curve, (b) F1_curve.

maxima suppression (NMS) operation, which is used to filter those BBoxes that heavily predict the same object, and keep only candidate BBoxes with high response. In the project, DIOU_nms is also adopted to modify the IOU in nms to DIOU_nms under the same parameters. For some occlusion overlapping targets, there are indeed some improvements. As seen in Figures 7 and 8, although the effects are similar in most states, it is fine to have a slight improvement without increasing the computational cost.

The OCS fault detection based on YOLOv5s proposed in this paper mainly includes two parts: target tracking and target detection. Target tracking completes the tracking and detection of nine components. When the tracking target is lost, the target detection will complete the retrieval of the target by detecting the current frame.

5. Experiments

In this section, the conducted experiments and results are presented, comparing YOLOv5 CNNs. Our experiment includes edge nodes and a cloud server. K8S v1.16.2 is used to the cloud layer; KubeEdge v1.22 and Docker v20.10 are applied to manage the edge; and Pytorch v1.10 is utilized at both the training and inference phase of the CNN model.

5.1. The Handcrafted Dataset. In our testbed, we provide a handcrafted label for each image by using labeling, corresponding to a distinct OCS image. The dataset has some volunteers from Qingyang Railway capturing images, and each of them performs at a different time, so we need to further handle images due to the vehicle motion, scene illumination changes, image noise, etc. First, the total labeled data were split into a training set (75% included), a validation set (the other 25%), and extra 200 images for testing as detailed in Table 2. Second, the data are not quite adequate to train a full-fledged CNN model. To address this issue, we adopted a repetition strategy, which is widely used for model training on scarce data samples [18]. We repeated the training set 300 epochs, each time randomly selecting 50 samples as input. Repeated training could effectively solve this problem, while batch training could improve the convergence speed and memory usage. Third, we grayscaled and normalized the images, described by a quantized grey level, without color information, as seen in Figure 9. Figure 10 shows the normalized dataset through statistics, which can visualize the dataset and get some useful information, (e.g., (a) shows the instances of label statistics, (b), (c), and (d) can see the spatial distribution of targets in the annotation file).

5.2. Experimental Setup. Table 3 shows the details of the two hosts for deploying the EC system. The YOLOv5s CNN model was trained using the Pytorch framework, and our experiments were performed using Python 3.8, OpenCV 4.1, logging with TensorBoard 2.4, plotting with seaborn 0.11, plus with pycocotools 2.0, etc. Note that all networks were trained using the optimizer with epochs = 300, batch_size = 16, and img_size = 640 × 640.

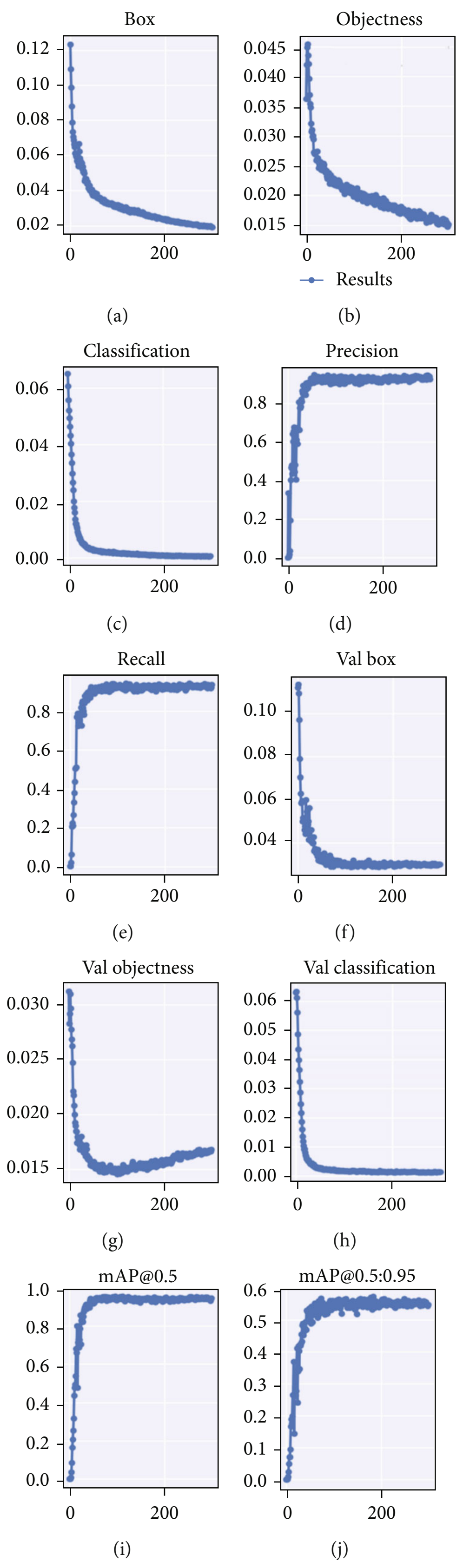


Figure 14: The training results of OCS dataset.

Table 6: Comparison of results of the three algorithms.

	YOLOv5s	YOLOv5m	YOLOv5l
Model size	28 MB	84 MB	346 MB
mAP@0.5	95%	92%	91%
300 epochs TrainingTime	11.220 h	14.590 h	14.990 h
291 images DetectTime	154.825 s	149.068 s	153.242 s
The resulting WeightModel	14.4 MB	42.5 MB	93.8 MB

5.3. Evaluation Metrics. For multiple categorization problems, the discrimination evaluation of the best solution during the classification training can be defined based on the confusion matrix as in Table 4. The table row represents the predicted class, while the column represents the actual class. The quantitative criteria we used to assess the performance are P (Precision) and R (Recall), as defined.

$$P = \frac{tp}{tp + fp}, \quad R = \frac{tp}{tp + fn}. \tag{1}$$

P and *R* are following performance criteria that are frequently in conflict. The smaller the *R*, the better the *P*. Therefore, we consider the *F*-score ∈[0, 1], which represents the harmonic mean of *P* and *R*, as shown

$$F = \frac{2}{1/P + 1/R}. \tag{2}$$

The area enclosed by the PR_curve is AP, and mAP (mean average precision) is the average AP of all categories to measure the average quality, defined as Equation (3); *Q* means the total numbers. The measurements of OCS multiple targets detecting are mAP@0.5 and mAP@0.5 : 0.95. mAP@0.5 is the average precision when IoU is 0.5. mAP@0.5 : 0.95 has a lower precision level than mAP@0.5 because the prediction threshold is increased by 0.05 from 0.5 to 0.95 at different IoU thresholds.

$$\text{mAP} = \frac{\sum_{q=1}^{Q} AP(q)}{Q}. \tag{3}$$

5.4. Results. It is found that after training, the model can not only detect the marked elements but also predict more unlabeled but available ones. The detection screen of the algorithm proposed in this paper for OCS is shown in Figure 11. In the complex background of various wire crossings, the algorithm can accurately identify the nine components of OCS.

Figure 12 is OCS confusion_matrix of the predicted results. This matrix makes it easy to see if the machine is confusing different classes. The matrix indicates the precision of 9 features, e.g., the precision of guanmao is 96%, and 4% of guanmao in images is a false negative, same as others. According to the evaluation metrics, the results of the 9 elements in target detection on the test set are as Table 5.

The more PR_curve is to the upper right, the better the overall performance of the model. As can be analyzed in Figure 13(a), the P rate of positioning supporters is above 98.5%, and the average (mAP@0.5) is up to 94.5%. We can compare the area size under the curve, but the balance point F1 is more commonly used. When $P = R$ (red slope), the larger the F1 value, the better the performance. Figure 13(b) calculates F1 scores for each category to distinguish them and finally get the arithmetic mean of F1 with all classes 0.93 at 0.412.

The results displayed are shown in Figure 14 in the form of prediction boxes, confidence values, and object classes. From the training results, it can be seen that the level of minimized return is achieved when the graphs start to form elbows in about 60 epochs at *P*, *R*, mAP@0.5, and mAP@0.5 : 0.95. (a) YOLOv5s uses GIoU as the loss of bbox. The box is the mean value of GIoU loss function. The more precision accurate, the smaller the box. As the training epochs increase, the box gradually drops below 0.02, the more epochs, the less decreasing trend. (b) Objectness is the average target detection loss, the smaller the target detection, the more accurate it is. When training to 300 times, the loss is reduced to less than 1.5%. (c) Classification is the mean of classification loss, there is essentially no difference while training to 100 epochs. (d) The highest score for *P* is 97.7% at 60 epochs. After that, the change is very small in the range of 0.05. (e) The highest value for *R* is 94.3% at 43 epochs. Thus, training for more than 45 epochs has experienced diminishing returns. BBox loss, target detection loss means, and Classification loss means of the validation set are as shown in (f) val BOX, (g) val Objectness, and (h)val classification. (i) The OCS on mAP@0.5 has a high level of precision, which is up to 93.5% in 60 epochs. (j) The highest value of mAP@0.5 : 0.95 reaches 0.573 and then tends to decline and stabilize at a range of 0.567. The m*AP* value indicates the average precision value is far above the threshold of 0.7 so that the resulting model is feasible to use.

Using the CNN model in this paper to compare with YOLOv5s, YOLOv5m, YOLOv5l, respectively, listed in Table 6, our YOLOv5s model achieved high accuracy, nearly 95%, and outperformed all other models and settings. We also tested our model on the EI system designed in Section 3 for the real-time monitoring OCS. Note that the same OCS dataset and epochs were used in the same EI testbed.

6. Conclusions

In this paper, we proposed an EI-based architecture for OCS fault detection by using a CV-based YOLOv5 algorithm, in order to detect the status of the OCS in the running of the train and perform real-time tracking detection of surveillance videos or images. Our design includes the surveillance layer, the edge layer, and the cloud layer. In the implementation, we used two Ubuntu operating systems simulating the edge layer and the cloud layer and K8S to manage nodes running on KubeEdge. We also evaluated three structures of YOLOv5 to find a trade-off between efficiency and performance. Extensive tests indicate that our YOLOv5s model built on EI achieves obvious advantages in speed and reliable accuracy, which can guarantee real-time detection of OCS equipment. In the future, we would like to incorporate more OCS elements to detect. Besides, we can further study the performance of our design with OCS Geometric forms.

Data Availability

The data used to support the findings of this study are available from the authors upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This paper was supported by the Natural Science Foundation of China (under grant 61973026), Beijing Natural Science Foundation (L201002 and L211002), Beijing Municipal Education Commission Funding (I20H100010 and I19H100010), in part by the Beijing Jiaotong University Project (under grant RCS2021ZZ005), and the Fundamental Research Funds for the Central Universities (2021CZ107 and 2021QY004).

References

[1] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, 2017.

[2] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[3] 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*, pp. 779–788, Las Vegas, NV, USA, 2016.

[4] “Edge intelligence (white paper),” 2017, http://storage-iecwebsite-prd-iec-ch.s3.eu-west-1.amazonaws.com/2019-09/content/media/files/iec.

[5] Z. Cheng, “Deep learning methods for fastener identification and location of high speed railway catenary support devices,” *Chinese Journal of Engineering Mathematics*, vol. 37, no. 3, pp. 261–268, 2020.

[6] K. Gaoqiang, G. Shibin, Y. Long, and C. Jianxiong, “Fault detection of missing split pins in swivel with clevis in high-speed railway catenary based on deep learning,” *Journal of the China Railway Society*, vol. 42, no. 10, pp. 45–51, 2020.

[7] Y. Chen, S. Lili, N. Huimin, L. Ting, W. Tao, and G. Xiugang, “Detection of pantograph faults based on yolov3-tiny detection and kcf tracking algorithm,” *Intelligent Computer and Applications*, vol. 5, 2020.

[8] C. Zhiyu and M. Feng, “Detection method of catenary insulator based on improved yolo v3,” *Journal of Wuhan Institute of Technology*, vol. 4, 2020.

[9] J. Feng, L. Liu, Q. Pei, and K. Li, “Min-max cost optimization for efficient hierarchical federated learning in wireless edge networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2687–2700, 2022.

[10] S. Zhang, W. Li, Y. Wu, P. Watson, and A. Zomaya, “Enabling edge intelligence for activity recognition in smart homes,” in *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 228–236, Chengdu, China, 2018.

[11] L. Zhu, Y. Li, F. R. Yu, B. Ning, and X. Wang, “Cross-layer defense methods for jamming-resistant CBTC systems,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 7266–7278, 2020.

[12] L. Yi, D. Gencai, L. Wei, J. Bo, and C. Jinchuan, “Research on application of edge computing technology in railway 5g mobile communication,” *Chinese Railways*, vol. 11, pp. 23–30, 2020.

[13] Y. Li, L. Zhu, H. Wang, F. R. Yu, and S. Liu, “A cross-layer defense scheme for edge intelligence-enabled CBTC systems against MiTM attacks,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2286–2298, 2020.

[14] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, “Extend cloud to edge with kubeedge,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 373–377, Seattle, WA, USA, 2018.

[15] X. Zhang and S. Xu, “Research on image processing technology of computer vision algorithm,” in *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, pp. 122–124, Chongqing, China, 2020.

[16] L. Liu, J. Feng, Q. Pei et al., “Block chain-enabled secure data sharing scheme in mobile-edge computing: an asynchronous advantage actor–critic learning approach,” *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2342–2353, 2020.

[17] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: optimal speed and accuracy of object detection,” 2020, https://arxiv.org/abs/2004.10934.

[18] L. Zhu, H. Liang, H. Wang, B. Ning, and T. Tang, “Joint security and train control design in blockchain empowered CBTC system,” *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 8119–8129, 2021.