IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 20 October 2023, accepted 17 November 2023, date of publication 1 December 2023,
date of current version 5 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3334973

RESEARCH ARTICLE

The Accelerated Inference of a Novel Optimized YOLOv5-LITE on Low-Power Devices for Railway Track Damage Detection

CHAO DANG, ZAIXING WANG, YONGHUAN HE, LINCHANG WANG, YI CAI, HUIJI SHI, AND JIACHI JIANG

School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730073, China

Corresponding authors: Zaixing Wang (55232793@qq.com) and Chao Dang (dc224017@gmail.com)

ABSTRACT Railway track malfunctions can lead to severe consequences such as train derailments and collisions. Traditional manual inspection methods suffer from inaccuracies and low efficiency. Contemporary deep learning-based detection techniques have challenges in model accuracy, inference speed, and are often associated with expensive computational costs and high power consumption when deployed on devices. We propose an optimized lightweight network based on YOLOv5-lite. which employs an enhanced Fused Mobile Inverted Bottleneck Convolution (BF_MBConv) to reduce the number of parameters and floating-point operations (FLOP) during backbone feature extraction. The Squeeze-and-Excitation (SE) mechanism is adopted, emphasizing more critical track features by assigning different weights from a channel-wise perspective. Utilizing DropBlock with holistic dropping as a substitute for Dropout with random dropping offers a more efficient means of discarding redundant features. In the neck section, Shuffle convolution replaces the conventional one, significantly reducing the parameter count while better integrating feature information post-group convolution. Lastly, the incorporation of Focal-EIoU Loss augments regression, and with the application of incremental dataset processing techniques, it addresses accuracy and sample imbalance issues. The refined algorithm achieves a mean Average Precision (mAP)@0.5 of 94.4%, marking an 8.13% improvement over the original YOLOv5-lite. Moreover, by leveraging the embedded platform integrated with the Intel® Movidius™ Neural Compute Stick cluster as the portable device for model deployment, Achieved a frame rate of 18.7 FPS. Our findings indicate that this approach can efficiently and accurately detect railway track damages. Additionally, it addresses the previously overlooked issues of performance-cost trade-offs, countering the past trend of prioritizing high performance at the expense of elevated power consumption and costs, proposing a harmonized approach that prioritizes efficiency and affordability.

INDEX TERMS YOLOv5-lite, lightweight network, data enhancement, neural compute stick cluster2, energy-efficient consumption.

I. INTRODUCTION

A. BACKGROUND

Rail transportation represents one of the primary modes of transit worldwide.While it offers unparalleled convenience, the safety of railway systems has been increasingly challenged. With the surge in usage years of railway tracks, the quality of the tracks, especially under various extreme weather conditions, has become a pivotal factor in train operation safety. Continuous rail traffic leads to wear and tear, causing fractures, gaps, and other damages to the tracks. Consequently, these damages come with associated risks of potential accidents. Therefore, precise detection of surface defects in the rail becomes imperative [1].To date, several mainstream methods have been devised to detect rail defects. These include the use of lasers to measure track geometries

The associate editor coordinating the review of this manuscript and approving it for publication was Rajeeb Dey.

and profiles [2], Ground Penetrating Radar (GPR) to assess ballast contamination [3], [4], ultrasonics or eddy current techniques to identify internal track defects [5], [6], and LiDAR for track fouling detection [7]. These automated detection methods have significantly enhanced track safety while reducing the need for manual labor hours.

Rail transport demands efficient, low-power, and portable defect detection. The YOLO(You only look one) network, having proven its mettle in areas like facial recognition [8] and autonomous driving [9], has been explored for rail surface damages [10], showing proficiency in detecting under challenging conditions such as low-light or irregular crack growth. YOLO's inherent strength lies in its ability to recognize multiple objects in one glance, bypassing traditional methods and making it favorable for real-time applications. However, the diversity in rail damage types and external environmental factors can affect its performance. To address YOLO's intricacies,Li et al. [11] proposed YOLOv3-Lite, leveraging depthwise separable convolutions and feature pyramids for enhanced defect detection. Mandal et al. [12] further improved detection precision using YOLOv2 for road surface crack.detection.Acknowledging the computational constraints of some devices, YOLO versions have been developed for diverse requirements. For instance, YOLO-Lite is designed for resource-constrained devices, while YOLOv5, although more complex, offers varied sizes to cater to different resource needs and has been further optimized using decision tree pruning. Such pruning not only reduces computational demands but also addresses overfitting [13]. Image preprocessing techniques, incorporating incremental augmentation operations, have further refined the detection process, enhancing model robustness and data quality [14].

In prior work [15], an algorithm utilizing a Raspberry Pi with ultrasonic-equipped cameras for high-speed vehicle detection was proposed. However, constraints due to its embedded nature hampered real-time accuracy. This solution boasts low energy consumption, cost-efficiency, and compactness, providing an equilibrium between precision and power consumption. Conventionally, deep learning frameworks are centralized in cloud environments. This poses challenges like substantial network latency and heightened energy and financial implications [16]. To mitigate these, the ''Edge AI'' paradigm is suggested. Edge AI decentralizes AI processing, shifting from central servers to device edges, such as smartphones and embedded systems, facilitating real-time analytics closer to data sources. For high-speed trains, this implies enhanced energy efficiency and optimal spatial integration. With AI capabilities, the system can process video streams instantly on-device, reducing latency and preemptively identifying rail discrepancies. Given its cost-effectiveness, a broader implementation on high-speed trains appears viable.

The main contributions of this work are summarized below: 1) For real-time detection of railway damages, this study initially proposes a portable real-time monitoring methodology based on the optimized YOLOV5-lite and embedded devices. Data augmentation techniques were employed to increase dataset volume and simulate detection efficacy under various environments. This methodology is capable of accurately identifying a multitude of railway damage types across varied actual driving environments. 2) To enhance detection precision and inference speed, we adopted several lightweight models, including Group Conv, DWConv, MBCONV, and Shuffle Conv. We also introduced attention mechanisms and residual structures, constructing an improved B-Fused-MBCONV model. By utilizing MB_Shuffle_Block in place of the original feature extraction backbone network, we further elevated the accuracy of feature extraction. Simultaneously, DWConv and Shuffle Conv modules supplanted the original convolution modules in the Neck network, and ultimately, the original loss function was replaced with Focal-EIoU loss function, alleviating computational burden and memory overhead while preserving model accuracy. 3) We engineered a portable Edge AI detection apparatus based on Raspberry Pi and NSC2 cluster, which exhibits superior performance and harbors practical application potential.

II. RELATED WORKS

Addressing the challenge of real-time detection of rail damage with a high-speed train as a moving carrier poses significant difficulties. As mentioned in [17], ''Edge AI technology is an enabling technology that allows computation at the edge.'' Edge AI is renowned for its small size, low power consumption, and excellent performance in terms of recognition speed and accuracy. If the deep learning framework is mounted on a local server, workstation, or cloud server, the accompanying issue is that the consumption, whether human or device, is hard to accept in terms of resource allocation and budgeting. Pursuing high-output precision for algorithms while maintaining portability and low power consumption has also become a trending research direction. In the domain of Edge AI, companies like Google, Intel, and NVIDIA have made significant contributions. Google's Coral SBC is the first development board equipped with Google Edge TPU. This AI accelerator combines the NXP i.MX 8M quad-core Arm Cortex-A53 processor with 1GB of memory, offering a fully functional AI edge computing platform. However, it comes at a steep price of 160 USD. In contrast, Intel's Movidius Neural Compute Stick (NCS) is the most affordable device for algorithms with high computational demands using multi-layer CNNs [16]. NCS and NCS2 are USB 3.0-based sticks that encompass Myriad 2 and Myriad X Vision Processing Units (VPUs), respectively. They can be plugged into any device based on Windows, Linux, MacOS, or Raspbian [18]. Myriad 2 contains 12 SHAVE (Streaming Hybrid Architecture Vector Engine) processors, designed specifically for parallel processing of visual data. Their low power consumption makes them highly suitable for battery-powered devices like drones, security cameras, and AR/VR devices. They also support various deep learning frameworks, including TensorFlow and Caffe. Myriad X is a successor to Myriad 2.

TABLE 1. In this project research, the price reference is based on the most common prices in China. since the NVIDIA Jetson TK1 is out of production and has lost its cost-effectiveness, for the step of deployment on trains, it is necessary to consider the price, space occu-pancy, and performance simultaneously materials and methods.

	Raspberry Pi 4B	Raspberry Pi 2B	Nvidia Jetson TK1	Nvidia Jetson TX2	Neural Compute Stick	Neural Compute Stick2
Price	$35	$10	NA	$300	$40	$68
Power	9w	5w	11.2w	16.1w	1w	1.4w
Size	55.88 mm x86.36 mm	88.9 mm x 63.5 mm	127 mm x 127 mm	50 mm x 87 mm	72.5 mm x 27 mm x 14 mm	72.5 mm x 27 mm x 14 mm
Weight	46g	40g	105g	85g	30g	30g
Processor	BCM2711	BCM2836	ARM Cortex-A15	ARM Cortex-A57	Myriad 2 VPU	Myriad X VPU

It comprises 16 SHAVE processors and introduces a specialized hardware block known as the Neural Compute Engine, a hardware accelerator specifically designed for deep learning inference [19].

Nvidia's Jetson TK1 module, based on the NVIDIA Maxwell™ architecture, is equipped with 256 NVIDIA CUDA® cores and a 64-bit CPU, and it employs an energy-efficient design strategy. Moreover, the module integrates the latest advancements in deep learning, computer vision, GPU computing, and graphics, making it highly suitable for embedded AI computations. On the other hand, the Jetson TX2 module is built upon the NVIDIA Pascal™ architecture. Impressively, while maintaining a compact form factor, it delivers robust performance and energy efficiency, rendering it ideal for smart edge devices such as robots, drones, intelligent cameras, and portable medical equipment. The TX2 module supports all functionalities offered by the TK1 module and further provides capabilities for constructing larger and more complex deep neural networks [20]. Both these modules encompass a GPU, each with 256 CUDA cores, and are compatible with NVIDIA's CUDA and cuDNN libraries. This indicates that these libraries can be leveraged to hasten deep learning and other parallel computational tasks. In [21], a proposition was made to employ the NVIDIA Jetson Nano for training and testing a novel deep learning model aimed at automatically classifying electrocardiogram (ECG) signals into seven distinct types of ECG beats. Meanwhile, [22] showcased the implementation of a rapid monocular depth estimation model's primary functionalities on the Nvidia Jetson single-board computer. Project [23] adopted the Nvidia Jetson TX2 as its primary hardware platform and highlighted the advantages in energy and cost efficiencies across several domains such as the Internet of Things (IoT), robotics, autonomous driving, and drone surveillance.

We employed the Raspberry Pi as the hardware platform, hosting both the first and second generations of the Intel Movidius Neural Computing Stick (NCS) as well as the Nvidia Jetson TK1 and Nvidia Jetson TX2. These setups were implemented on high-speed trains for a comparative study of their performance in real-time rail damage detection.

Edge AI small model networks, akin to YOLOv5-Lite, include yolov3-tiny. YOLOv5-Lite stands out due to its compact model size and reduced floating-point operations (FLOPs), enabling its deployment on ships for detection tasks without sacrificing precision. Remarkably, it achieved a model size of 2.38 M [24]. An enhanced version of yolov3-tiny was utilized for a rapid tomato detection method in a picking robot operating in complex scenarios [25], evidencing commendable inference speeds and frames per second (FPS). Moreover, owing to its compact nature, YOLOv5-Lite has found extensive applications in edge AI. For instance, when YOLOv5-Lite was deployed for ship detection, it was integrated onto the Nvidia Jetson TX2 [24]. Conversely, the applications of yolov3-tiny are even more diverse. In [26], yolov3-tiny was integrated into the UltraScale XCKU040 FPGA (Field Programmable Gate Array) to expedite convolutional neural networks, enhancing both its FPS and inference speed.

The dataset for this study primarily comes from the "Railway Track Fault Detection" dataset available on Kaggle [27], consisting of 384 rail images. Additionally, with permissions obtained, we incorporated 500 images taken from local railway shoots. Keeping data diversity in mind, the railway tracks were photographed under varying light conditions (including day and night), tracks with significant foreign object obstructions, and during different weather conditions. This approach was adopted to ensure the system's robust performance when mounted on trains operating under varying conditions. Inference on edge devices is a pivotal component of edge AI. It's paramount to recognize that model training is a process heavily reliant on computational power and resources. Deploying this training process on edge AI devices would be ill-advised. Therefore, our training processes were stationed on workstations, and subsequently, models equipped with the obtained weights were deployed on target hardware for execution.

FIGURE 1. Environmental simulation of rail damage samples.

We collected information on six hardware devices available in the market, as shown in Table 1. Among them, Nvidia Jetson TX2 stands out as the most expensive and power-consuming, while in terms of volume, Nvidia Jetson TK1 is the most bulky. Within the Raspberry Pi series we selected, the 4B and 2B models differ by $20 in price, 4W in power consumption, with negligible differences in volume and weight. In the acceleration module, the Neural Compute Stick series 1st and 2nd generations differ by $28 in price, with other aspects being virtually identical. Notably, the TK1 model is now obsolete, rendering its price without reference value.

In this experiment, we will delve into the optimization of network models and the effects on hardware detection. For clarity, we will segregate the system into two phases: the model preparation phase and the deployment phase on edge AI devices.

III. MATERIALS AND METHODS

A. DATA AUGMENTATION

In this experiment, our dataset originates from Kaggle's ''Railway Track Fault Detection'' repository [27], supplemented with over 500 personally captured images, culminating in a total of 884 images encompassing several common railway track defects. During our data collection process, certain weather conditions were not encountered, hence we also employed data augmentation techniques to simulate these scenarios, as depicted in Figure 1. Common simulated scenario augmentation methods include: a) simulating snowy conditions in the case of missing fastener, b) simulating bright light scenarios in the case of Track fracture, c) simulating rainy conditions under Fine crack. The aim is to enhance the model's generalization capability, while simultaneously considering the challenges posed by various complex environments under real driving states for detection.

The crucial point to note is that all of our data augmentation schemes involve random processing of the original samples before integrating them back into the primary dataset.

- **i.** Crop pictures at will: Crop within a range of [0.1,1] in the original images. The scaled cropping will eventually be adjusted to the original size.

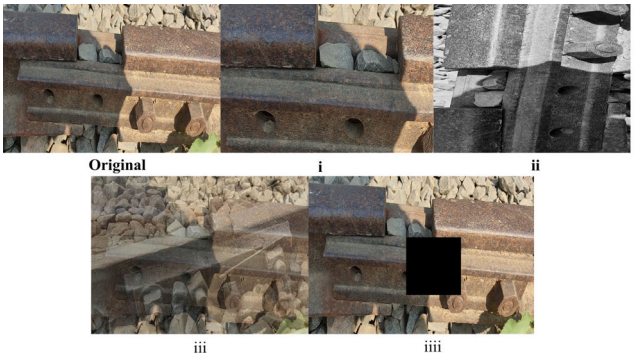

FIGURE 2. Demonstration of the effects for four different data enhancement strategies.

- **ii.** RandAugment is randomly applied twice, with an amplitude of 9 and a standard deviation of 0.5, as proposed in [28].
- **iii.** D. Lopez-Paz introduced a novel deep learning training methodology named Mixup [29]. This approach specifically addresses the potential overfitting issues and sensitivities to adversarial samples that deep neural networks might exhibit during training. The fundamental idea behind Mixup is to generate new training samples during the training process by creating a convex combination of training samples and their corresponding labels. Such a tactic encourages the neural network to exhibit simpler linear behaviors between training samples, subsequently enhancing its generalization capability.
- **iii.** Shielding: Randomly select an area within the range of [0.02,0.2] in the original images to be completely covered in black.

Figure 2 illustrates the effects brought about by the aforementioned four distinct data augmentation strategies proposed by us.

Following our data augmentation, a total of 4345 data items were obtained according to above, which were categorized into three data folders including Training, Validation, and Testing, with specific classification information as presented in Table 2. Among them, the circumstance of missing

TABLE 2. Class distribution of the railway damage data set used.

Set	Normal	Missing fastener	Track fracture	Fine Crack
Training	1211	1029	961	485
Validation	241	129	81	57
Testing	231	93	59	43

fastener occurs most frequently, indicating that the absence of fasteners is a common yet easily overlooked phenomenon.

B. RAIL DAMAGE DETECTION MODEL BASED ON IMPROVED YOLOV5-LITE

The significance of railway track damage detection is self-evident, hence the detection model is required not only to have faster detection speed but also to maintain high detection accuracy, and it needs to perform well when deployed on energy-efficient edge AI devices. The YOLOv5 algorithm is an efficient object detection algorithm, employing a lightweight and efficient model structure to ensure high detection accuracy and rapid detection speed. Therefore, we believe that utilizing the YOLOv5 model for railway track damage detection can yield better results. Reference [30] introduced a lighter, faster, and more deployable network: YOLOv5 LITE. It is mentioned that YOLOv5 LITE, by incorporating channel shuffling in the YOLOv5 neural network model and performing channel pruning on the YOLOv5 head, can achieve an inference speed of 10+ FPS on Raspberry Pi 4B with an input size of 320. Moreover, compared to YOLOv5, the feature extraction network of YOLOv5-Lite consists of Shuffle_Block, and has removed the Focus layer present in YOLOv5, the model quantization accuracy decline is within an acceptable range, making it more deployable. Due to the numerous parallel operations in the SPP structure, following the third principle in ShuffleNet V2, YOLOv5-Lite has cut off the SPP with more parallel operations and a C3 structure following it, not only enhancing the inference speed but also ensuring accuracy. Through the enhancement of the YOLOv5 LITE feature extraction backbone, neck, and other operations, we have obtained our BF_MB-YOLOv5 model, which has achieved improvements in accuracy and inference speed, well-suited for our task of detecting railway damages, and is highly compatible with our portable edge AI devices.

1) GROUP CONV

In Figure 3(a), the standard convolution operation is depicted. The input feature map dimension is $H^*W^*C_1$, the convolution kernel size is $h_1^*w_1^*C_1$ with a quantity of C_2, and the output feature map dimension is $H^*W^*C_2$. The equation for the parameter count of standard convolution is denoted as equation (1). In Figure 3(b), the grouped convolution operation is illustrated, where the input feature map is initially divided into g groups according to the channel count, hence each group's input feature map dimension is $H^*W^*(C_1/g)$, convolution size is $h_1^*w_1^*(C_1/g)$, and the output feature map dimension is $H^*W^*(C_2/g)$. Concatenating the results from g groups yields the final dimension of $H^*W^*C_2$ for the output. The parameter count for grouped convolution is denoted as equation (2). Comparing equation (1) with equation (2), it is discernible that the parameter count of grouped convolution is (1/g) of the standard convolution. It is evident from this comparison that grouped convolution indeed can reduce the parameter count of convolution, thereby enhancing efficiency.

$$par\ ams_{normal\ Conv} = c_1 \times h_1 \times w_1 \times c_2 \tag{1}$$

$$par\ ams_{Group\ conv} = g \times \frac{c_1}{g} \times h_1 \times w_1 \times \frac{c_2}{g} = \frac{params_{normalConv}}{g} \tag{2}$$

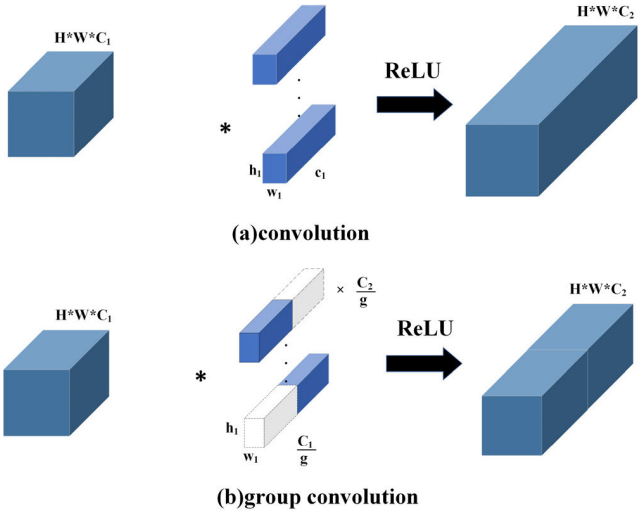

FIGURE 3. Group convolution operation.

DWConv (Depthwise Separable Convolution) represents a special case of GROUP [31]. The group count g equals the channels of the input feature map, with each channel of the input image being convolved by a singular convolution kernel. Post the channel-wise convolution operation, there is no inter-fusion of information among the channels. Subsequent to the point-wise convolution operation, a thorough fusion of features across different channels is achieved, thereby further obtaining the output feature map. To address the issue of excessive parameter count in the original model, we substituted the standard 3×3 convolution in the YOLOv5 model with DWConv.

In Figure 4(a), the input feature map of dimension $H^*W^*C_1$, post undergoing a 3×3 convolution layer, yields an output feature map of dimension $H^*W^*C_2$, where the parameter count for the convolution process is represented by equation (3). Conversely, in Figure 4(b), the applied DWConv dissects the standard convolution into a depth convolution of kernel size 3×3 and a point-wise convolution of kernel size 1×1. The depth convolution operates on each input channel,

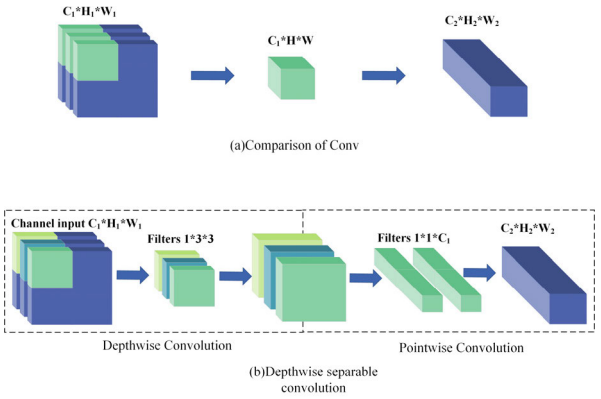

FIGURE 4. Depthwise separable convolution.

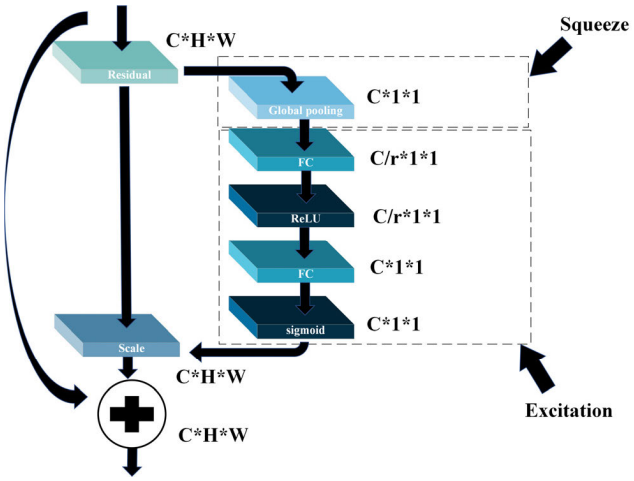

FIGURE 5. Structure of SE block.

followed by the point-wise convolution which performs a weighted sum and combination on the generated feature layers in the depth direction, thereby creating a new feature layer. The parameter count for this convolution process is denoted by equation (4). It can be discerned that the parameter count of DWConv is considerably lesser than that of the standard convolution. This delineates that this method can effectively diminish the parameter count of the network, thereby substantially elevating the detection efficiency.

$$par\ ams_{normal\ conv} = 3 \times 3 \times C_1 \times C_2 \tag{3}$$

$$\begin{aligned} par\ ams_{DWconv} &= 3 \times 3 \times C_1 + 1 \times 1 \times C_1 \times C_2 \\ &= par\ ams_{normal\ conv} \times (\frac{1}{C_2} + \frac{1}{9}) \end{aligned} \tag{4}$$

2) MBCONV BLOCK

In 2019, the report proposed a lightweight network known as EfficientNet [32], which is constituted by seven MBConv modules. An optimization upgrade is carried out within the MBConv modules by incorporating the Squeeze and Excitation (SE) operation, enabling shallow networks to extract image features and describe images through a global receptive field as well.

The SE attention mechanism [33] is utilized for capturing channel-wise information, with the objective of learning a set of weight values to represent the significance of each feature channel. Based on the magnitude of the weight values, a reorganization of the feature channels is conducted, thereby accentuating the useful feature channels and attenuating the less useful ones. The specific structure is illustrated in Figure 5. The squeeze module compresses the feature map through the spatial dimension, simplifying each H×W sized two-dimensional feature channel into a single real number Z. These real numbers are amalgamated into a 1 × 1 × c feature vector. This step, to some extent, is capable of capturing the spatial information of dimension W×H. The excitation module processes the 1×1×c vector obtained from the squeeze module to ascertain the significance weights of each feature channel. The processing sequence encompasses: dimension reduction to c/r via a fully connected layer, feature mapping through ReLU, restoration to c from c/r via another fully connected layer, and determining weights through a sigmoid activation. These two modules, operating in concert, enable the network to attribute varying significance to each feature channel during forward propagation, consequently enhancing the model performance.

As depicted in Figure 6(a), MBConv consists of standard convolution, depthwise convolution (accompanied by BN and Swish activation), SE module, and Dropout layer. Depthwise convolution is a channel-wise convolution, while the 1 × 1 standard convolution (also referred to as pointwise convolution) is employed to perform weighted operations on the channel dimension of the output from the depthwise convolution. This combination effectively reduces the computation and parameter count of the model. The report mentions that the MBConv employs the Depthwise Conv structure [32], which possesses few parameters compared to standard convolution, alongside high computational capacity. However, under general circumstances, Depthwise Conv cannot fully leverage the performance of some existing hardware acceleration devices, resulting in subpar inference performance of the MBConv structure. To address this, we propose a more adaptable structure for portable hardware acceleration devices known as B-fused-MBconv as illustrated in Figure 6(b). This new structure utilizes a convolution layer in tandem with a BN layer to improve the MBConv structure. During computation, the operations of the BN layer are fused into the convolution layer, thus indirectly omitting the computational load of the BN layer, accelerating training. Specifically, the Conv1×1 and Depthwise Conv3×3 in the main branch of the MBConv structure are replaced with a standard Conv3 × 3.

Moreover, we discovered that the characteristic of random drop by dropout, although highly effective for fully connected layers, is less impactful for convolution layers due to the spatial correlations in feature space. That is, even with dropout, the information regarding the input can still be conveyed to the subsequent layer, leading to network overfitting. Therefore, we introduced DropBlock, a structured form of dropout, which aggregates and drops units within adjacent areas of

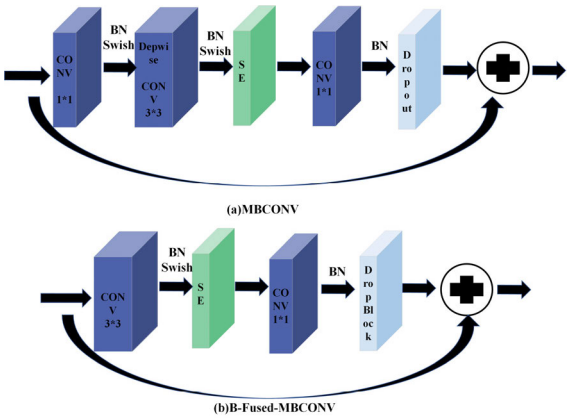

FIGURE 6. MBConv and the improved B-Fusion-MBCONV.

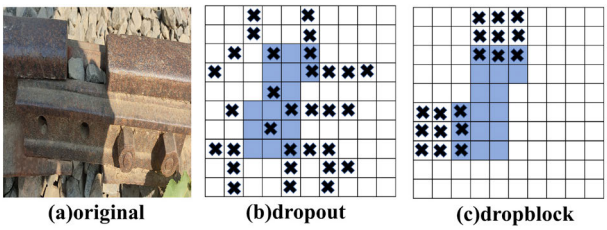

FIGURE 7. Contrast dropout with dropblock.

the feature map together. As illustrated in Figure 7, (a) represents the original image. In Figure 7 (b), the blue portion denotes the activated feature units, while (b) exhibits the random dropout of activated units. However, post-dropout, the network still learns similar information from the vicinity of the dropped activated units. In Figure 7 (c), the blue portion represents the activated feature units, while (c) displays the employed DropBlock in our work, through dropping a section of adjacent areas altogether. In (b), a portion of the rail along with a missing nut is filtered, prompting the network to focus on learning the features of another missing nut to achieve accurate classification, thus demonstrating enhanced generalization.

3) SHUFFLENET OF BACKBONE

In YOLOv5 LITE, 85% of the backbone portion is comprised of Shuffle block modules with strides of 1 and 2. The Shuffle block module originates from the ShuffleNet V2 network model. As depicted in Figure 8(a), channel splitting is performed following the feature channels. The left branch undergoes no operations, while the right branch undergoes three convolution operations, where two 1×1 convolutions have been replaced by regular convolutions from grouped convolutions in ShuffleNet v1. Subsequently, data from these two branch channels are merged with Concat+ channel shuffle operation, not only equating the number of input and output channels of this basic module but also avoiding the Add operation.

In Figure 8(b), the number of output channels is double the number of input channels, and the processes in the left and

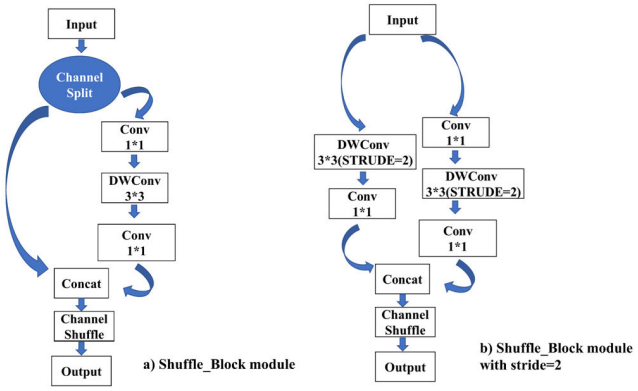

FIGURE 8. Shuffle Block module with stride of 1 and 2.

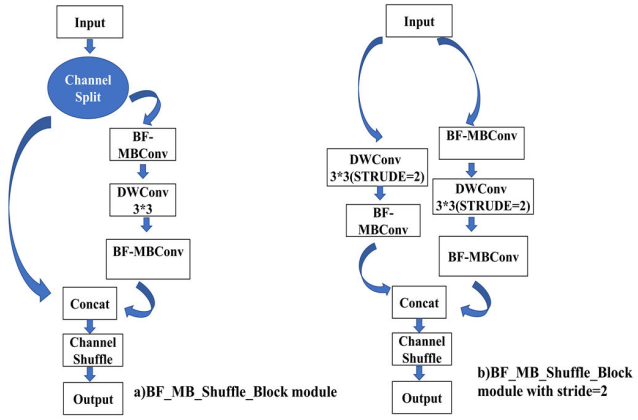

FIGURE 9. BF_MB_Shuffle_Block module.

right branches are essentially the same as in (a). The backbone of the YOLOv5-lite network structure primarily replaces the DarkNet53 backbone with ShuffleNet V2. To avoid the multiple usages of C3 layer and high-channel C3 layer, the 1024 conv from the shufflenetv2 backbone has been removed. The structure of the Shuffle block is illustrated in Figure 8.

The report mentions that due to the redundancy present in the Shuffle block structure [34], the feature maps affecting precision are obtained from convolution operations and are input into the next convolution layer for computation. This process encompasses a substantial number of network parameters. Concurrently, these network parameters consume a significant amount of computational resources and do not exhibit particularly outstanding performance in terms of feature extraction precision. Hence, we employ our enhanced module, BF_MB_Shuffle block, to replace the existing Shuffle block module. Specifically, the 1*1 convolution in the original Shuffle block module is substituted with our improved BF_MB convolution as depicted in Figure 9. We observed a notable increase in the number of layers in the feature extraction network post-replacement. With the increment in convolution layers, the feature extraction network's performance can be effectively enhanced, facilitating better analysis of high-level semantic information. On one hand, the attention mechanism used in the BF_MBConv block

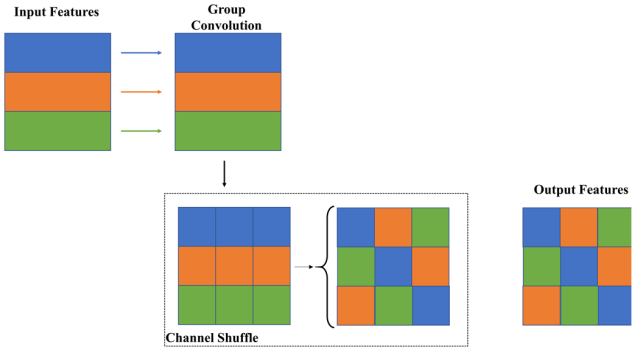

FIGURE 10. Channel shuffle operation after group convolution.

can efficiently reduce the network's parameter computation load while adding convolution layers; on the other hand, by introducing the concept of attention mechanism, the network during training can allocate greater weight to the channels where important information resides. However, with the increase in convolution layers, the model may encounter the issue of gradient vanishing during training, a problem which the residual structure introduced in the BF_MBConv block can effectively resolve. Moreover, our BF_MBConv improved structure effectively addresses the issues of the standard MBConv's inability to fully utilize the computational performance of hardware devices and the network's propensity for overfitting.

4) SHUFFLE CONV OF NECK

Based on the theoretical analysis of Group Conv and Channel Shuffle in Shuffle block as mentioned in the preceding sections, we amalgamated the two to construct our Shuffle Conv module to replace the 1*1 channel separation convolution block in the neck part of YOLOv5 LITE. Our formula (2) has indicated that compared to standard convolution, the parameter quantity in Group Conv would significantly diminish, meanwhile channel shuffling would better amalgamate the feature information post-group convolution. As illustrated in Figure 10, within the Channel Shuffle structure, the feature maps post-convolution operation are divided according to the number of groups, and channels from different groups are then shuffled together. This ensures that the output feature maps have inputs originating from different groups, allowing for the feature information to circulate among different groups, and further thoroughly amalgamating the feature information.

The Shuffle Conv module we constructed is depicted in Figure 11, replacing the conventional convolution structure by amalgamating 1×1 group convolution and the Channel Shuffle module. Post the group convolution operation, the feature maps are fed into the Channel Shuffle module and divided according to the number of groups, followed by a reshuffling of channels across different groups. This approach ensures that subsequent feature maps obtain inputs from different groups across channels, thereby effectuating efficient feature fusion. This module is employed within the network's

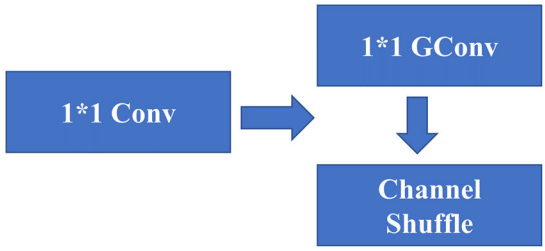

FIGURE 11. Shuffle Conv module.

feature enhancement architecture, achieving a superior fusion of information across three effective feature layers through the channel shuffling method, and to an extent, reducing the computational load of the model.

5) THE LOSS FUNCTION EIOU LOSS IS INTRODUCED

YOLOv5 by default employs CIoU Loss as the loss function for bounding boxes, while utilizing Logits and Binary Cross Entropy (BCE) for handling losses of target scores and class probabilities, respectively. Although CIoU Loss comprehensively takes into account the overlapping area of bounding boxes, the distance between centers, and aspect ratios, its treatment of aspect ratio discrepancies is not sufficiently accurate, which at times may constrain the optimization performance of the model. To address this issue, this paper proposes the adoption of Focal-EIoU Loss in lieu of CIoU Loss. This modification dissects the aspect ratio loss into the differences between predicted width and height and the width and height of the actual minimum enclosing box, thereby accelerating model convergence and enhancing regression accuracy.

Focal-EIoU Loss not only decomposes the aspect ratio loss term into the differences in predicted width and height versus the width and height of the actual minimum enclosing box, thereby accelerating the model's convergence and enhancing regression accuracy, but also incorporates Focal Loss to address the issue of class imbalance. Specifically, it diminishes the contribution of a large number of anchor boxes with lesser overlap with the target box in bounding box regression optimization, allowing the model to focus more on high-quality anchor boxes. This effectively resolves the class imbalance issue, improving the model's performance in bounding box regression tasks.The L_{EIoU} loss function formulae are represented as in equations (5), (6), and (7) respectively.

$$L_{\mathrm{EIoU}} = L_{\mathrm{IoU}} + L_{dis} + L_{\mathrm{asp}} \tag{5}$$

$$L_{\mathrm{EIoU}} = 1 - IoU + \frac{\rho^2\left(b, b^{gt}\right)}{(w^c)^2 + (h^c)^2} + \frac{\rho^2\left(w, w^{gt}\right)}{(w^c)^2} + \frac{\rho^2(h, h^{gt})}{(h^c)^2} \tag{6}$$

$$L_{\mathrm{IoU}} = 1 - \frac{|A \cap B|}{|\mathrm{A} \cup \mathrm{B}|} \tag{7}$$

In Focal-EIoU Loss, L_{IoU} denotes the IoU (Intersection over Union) loss, L_{EIoU} represents the center point distance

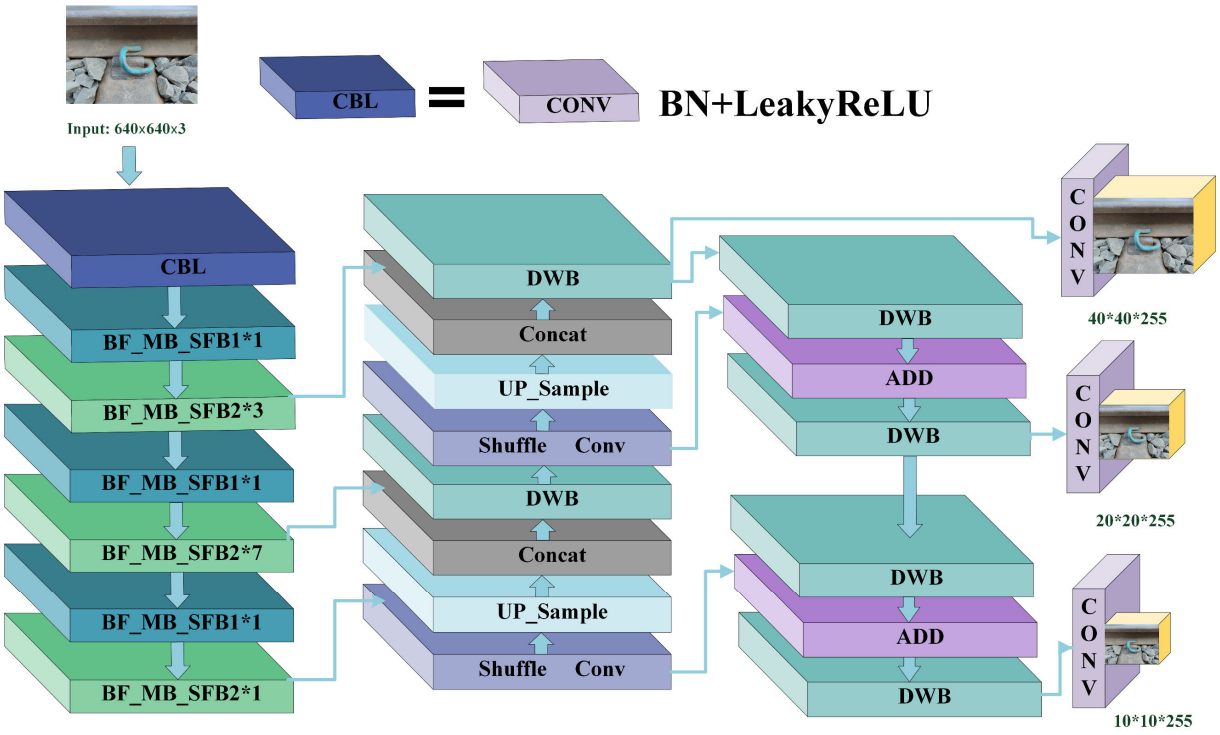

FIGURE 12. Improved BF_MB-YOLOv5 network structure diagram.

loss, and L_{asp} signifies the aspect ratio loss. Here, A and B respectively denote the predicted box and the true box. w^c and h^c correspond to the width and height of the Minimum Bounding Rectangle (MBR) encompassing both boxes, while b and b^{gt} respectively denote the center points of the predicted box and the target box.In essence, Focal-EIoU Loss is a comprehensive loss function, which not only attends to IoU and center point distance, but also particularly considers the aspect ratio loss. By decomposing the aspect ratio loss term into the difference in predicted width and height versus the width and height of the actual minimum bounding rectangle, and integrating Focal Loss, this loss function effectively accelerates model convergence, enhances regression accuracy, and addresses the issue of sample imbalance. This results in a model with increased precision and robustness in bounding box regression tasks.

In our BF_MB-YOLOv5 network as shown in Figure 12, the Backbone primarily consists of BF_MB_Shuffle_Block1 and BF_MB _Shuffle_Block2, and employs depthwise separable convolution in lieu of the original standard convolution. The 1 × 1 channel separation convolution block in the neck part of the original YOLOv5 LITE has been replaced with a lighter Shuffle Conv block. Lastly, the more superior Focal-EIoU loss function is utilized for bounding box regression tasks. This significantly reduces the model parameters, enhances the algorithm detection speed, and also results in a notable improvement in model detection accuracy.

TABLE 3. Model quantization performance comparison.

Model	Model Size(M)	mAP(%)
FP32 quantification	12.1	93.41
FP16 quantification	9.3	91.52
INT8 quantification	7.4	89.17

C. MODEL PROCESSING

1) MODEL PRUNING AND QUANTIZATION TECHNIQUES

In the OpenVINO toolkit, there is an in-built model optimizer that allows for user-driven model quantization. It's noteworthy that as the data type level of the model is reduced, the inference speed increases, typically at the cost of a decline in accuracy. The OpenVINO toolkit offers three model data types: int8, float32, and float16, as illustrated in Table 3. We conducted experiments on these three types of quantization, and it can be observed that among the three quantizations, the FP23 quantification has the largest Model Size and also the highest mAP.

2) OPENVINO ACCELERATED MODEL DEPLOYMENT

OpenVINO [19], introduced by Intel in 2018, is a framework designed for computer vision development. The primary utility of this framework is to expedite on-device model inference. OpenVINO facilitates inference acceleration

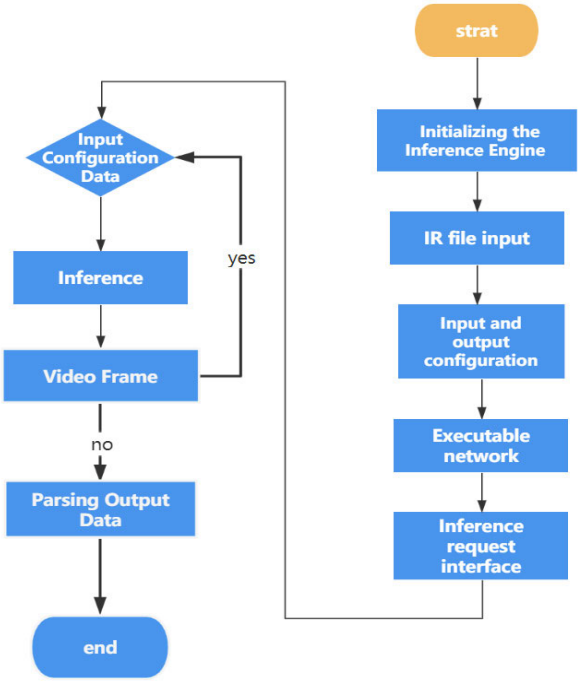

FIGURE 13. Openvino development flow chart.

specific to Intel CPUs and Intel Neural Compute Sticks by providing dedicated plugins. Deep learning engineers can easily integrate model inference and deployment on embedded systems by leveraging the API interface provided by OpenVINO. Key components of this framework are the Model Optimizer and Inference Engine. The Inference Engine acts as an API interface, tasked with loading trained models, and importantly, it supports asynchronous multi-threaded inference acceleration, crucial for our subsequent work. Meanwhile, the Model Optimizer refines models, such as removing redundant layers and enabling fusion acceleration for convolution layers combined with Batch Normalization (BN) and ReLU activation functions. A noteworthy feature is its integration with the OpenCV library, allowing direct raw image processing through API calls.

As discussed in [38] and depicted in FIGURE 13. the inference development process is as follows:

I. Initialization of the Inference Engine: The OpenVINO Inference Engine is initialized to ensure hardware availability and readiness for model inputs. For this study, Intel's second-generation Neural Compute Stick was used.

II. Loading the Model IR File: Subsequently, the IR (Intermediate Representation) file of the model is loaded, which consists of the network structure parameter file (.xml) and the weight parameter file (.bin) is loaded. Preliminarily, due to OpenVINO's lack of direct support for PyTorch model reading ofr OpenVINO's, conversion to the ONNX format is necessary via the Python interface is necessary, followed by the transformation of this ONNX format into the IR.

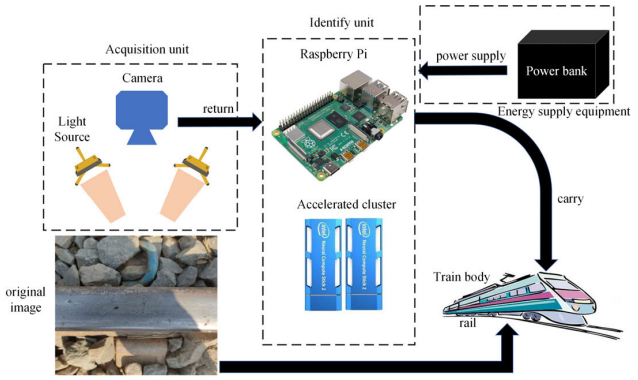

FIGURE 14. Detection system design.

III. Configuring Input and Output Parameters: Post model-loading, input and output parameters are set. Models have specific criteria for input, such as image size, format, normalization procedures, and the format for output, whether it be image classification, object detection, or semantic segmentation.

IV. Constructing an Executable Network: An executable network is constructed using the IR files, converting the IR into a format executable on the inference engine.

V. Creating an Inference Object: An inference object is instantiated, set to perform the inference operation.

VI. Feeding Data: Actual input data is fed to the inference object, which may range from individual and batches of images, to video frames.

VII. Executing Inference: Inference is executed, processing input data to yield output results.

VIII. Post-Inference Operations: Once inference concludes, output data is parsed, followed by requisite image post-processing. In the case of video data, settings can dictate intervals between frame captures, with inference carried out upon data retrieval, until the entirety of frames is processed.

D. APPLICATION

The hardware framework for the experimental design is illustrated in Figure 14. Initially, our pre-configured model was integrated into the main host. Subsequently, the camera collects information on the samples to be detected, it is noteworthy that to adapt to nighttime conditions, two illumination devices were mounted around the camera. A connection was then established between the camera and the Raspberry Pi development board. Considering the portability requirements of the device application, the Raspberry Pi was equipped with a portable power supply capable of displaying energy parameters. This not only ensured a convenient power supply but also provided energy to the device. Ultimately, the entire system is envisioned to be deployed on train. Regarding system architecture, the edge AI device on the train directly interfaces with a base station, ensuring timely transmission of critical alerts when significant issues are detected.

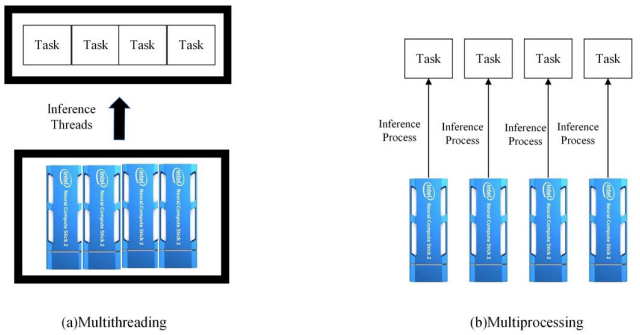

FIGURE 15. Multithread model and multiprocessing model.

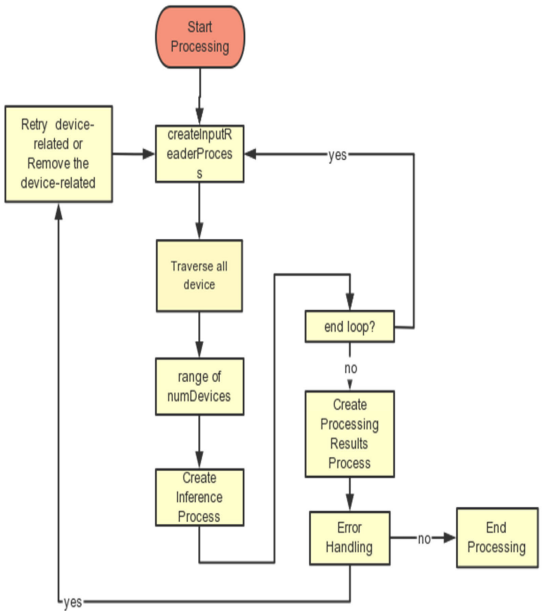

FIGURE 16. Multiprocessing method.

Lastly, to accelerate the recognition device and enhance detection efficiency, we proposed a cluster composed of two Intel Neural Compute Stick 2 units to support the inference phase. When integrated with high-speed trains, this system excels in terms of energy efficiency and cost-effectiveness. Whether considering large-scale implementation on railways or the potential impact on train operations, the foresight attributes demonstrated by this system appear to be promising.

1) MULTIPROCESSING

In the method described [39] and shown in Figure 15(a), a multithreaded approach is introduced. Throughout the inference process, each inference task is assigned to a separate thread for execution. Subsequently, the generated images are queued for processing by multiple NCS2s in a single procedure; each image may be handled by multiple NCS2 systems. In layman's terms, a single acceleration cluster handles one task. However, Python, aimed at simplifying memory management and ensuring thread safety, incorporates a Global Interpreter Lock (GIL) [40]. This implies that at any specific moment, only one thread can execute Python bytecode. Even on multicore hardware, due to the presence of GIL, Python's multithreading cannot achieve true parallel computing as only one thread operates at any time. This necessitates an alternative technique, namely Multiprocessing as depicted in Figure 15(b), where each NCS2 corresponds to a separate process. In layman's terms, a single acceleration entity only handles one task, deviating from the traditional operation of sharing a single process across multiple devices.

The Multiprocessing strategy also circumvents the obstacle of Python being unable to achieve true parallelism due to GIL. However, this enhanced processing performance results in stricter resource demands, implying the requirement for more processing devices. From a parallel computing perspective, considering that each NCS2 device (each process) handles different workload, this exploits data parallelism to boost computational performance. A caveat of this method is that since each process independently manages its memory and state, there might be an increased demand on system resources such as memory to maintain multiple autonomous processes.

Algorithm 1 Error Handling System Pseudocode

```
try:
        thread = createInferenceThread(i)
        launchThread(thread)
    except Exception as e:
        print(f"Error: Failed to create or launch thread for
device {i} : {e}")
        print("Attempting to restart the device...")
        try:
            restartDevice(i)
        except Exception as e:
            print(f"Error: Failed to restart device {i} : {e} ")
            print("Banning the device...")
            banDevice(i)
    end try
end for
```

For system stability, the integration of exception handling logic can be considered. For instance, if there is a failure in thread or process creation, a restart mechanism can be established. Considering the scenario of installation on high-speed trains, issues related to device lifespan and malfunctions may disrupt the queue handled by a single NCS2 device. To alleviate this situation, we have designed a system workflow with an error detection mechanism as shown in Figure 16, prioritizing system restart in the event of device failure. If persistent errors are detected post-restart, the faulty device will be disabled to ensure the integrity of the entire system. Employing the try/except structure aids in capturing and addressing potential exceptions. Upon error detection, notifications will be sent to the terminal before proceeding with the restart or disable program, followed by alerting railway personnel for device maintenance. Algorithm 1 elucidates this point with pseudocode.

IV. RESULTS AND EXPERIMENTS

A. COMPARISON AND EVALUATION OF NETWORK MODELS

The experimental platform is based on NVIDIA RTX 2060 Super 8G GPU, i7-5950HQ, and Windows 10 operating system. The software includes CUDA11.5, cuDNN8.2, Python 3.8, and the Pytorch 1.0.0 deep learning framework, The following hyperparameters were utilized: a learning rate of 0.01, the optimizer employed was Adam, and the batch size was set to 16.

In the domain of model performance evaluation, classification accuracy is often a conventional metric. However, in our work, it is not the optimal evaluation criterion. This bias is attributed to the typical skewed distribution of images of different classes in the training dataset. To comprehensively understand the capabilities of the model, multiple metrics were evaluated. Specifically, mAP@0.5 denotes the mean average precision with a set Intersection over Union (IoU) threshold of 0.5. IoU, ranging between 0 and 1, quantifies the overlap between the predicted bounding boxes and the ground truth bounding boxes, representing perfect consistency.

Accuracy is the ratio of correct predictions in all predictions as shown in equation (8). Precision elucidates the ratio of correctly predicted positive instances to the total instances predicted as positive, measuring the accuracy of the classifier in positive predictions as illustrated in equation (9). On the other hand, Recall depicts the ratio of correctly predicted positive instances to the actual positive instances, reflecting the proficiency of the classifier in identifying positive samples as illustrated in equation (10). The F1 score is the harmonic mean of precision and recall, encapsulating the overall accuracy and recognition ability of the classifier as shown in equation (11).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

$$F1 = 2^{*}\frac{Precision^{*}Recall}{Precision + Recal} \tag{11}$$

Within this context, True Positives (TP) refer to the count of positive class samples correctly predicted as the positive class by the model. False Positives (FP) denote the count of negative class samples incorrectly predicted as the positive class by the model. False Negatives (FN) are the count of positive class samples incorrectly predicted as the negative class by the model. True Negatives (TN) refer to the count of negative class samples correctly predicted as the negative class by the model. (Number) N represents the number of object classes. As depicted in the confusion matrix of FIGURE 17, the prediction accuracy is highest for the Normal category, while it is the lowest for the Fine Crack category. This may likely be due to a certain level of similarity between the Fine Crack and Track Fracture categories, posing a certain challenge for classification.

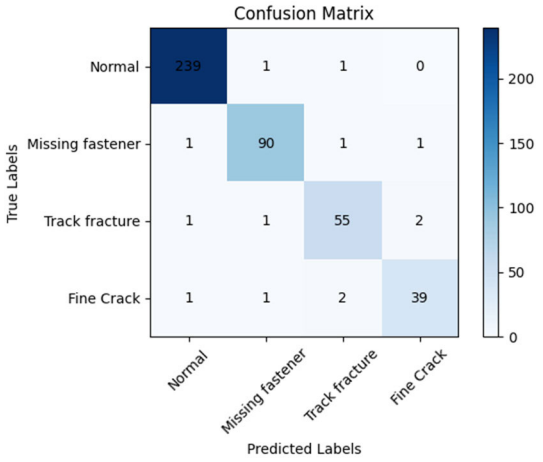

FIGURE 17. The confusion matrix of our improved method.

To evaluate the performance of different components within the model, this study conducted ablation experiments. The BF_MB _Shuffle, F_MB-ShuffleBlock (without dropblock) backbone, Shuffle Conv neck, and Focal-EIoU loss function are the distinct parts of the ablation study. As can be inferred from TABLE 4, employing the dropblock's overall drop method in the task of rail defect detection, which demands a higher level of feature integrity, indeed outperforms the random drop of dropout. This also validates that the improved algorithm has elevated the mAP@0.5 value of YOLOv5-lite (unimproved) by 8.13 percent, increased the recall rate by 10.37 percent, F1 score increased by 5.03% while augmenting the detection speed by 28 frames per second.

In the previous section, we discussed four dataset augmentation and enhancement techniques. Table 5 displays the accuracy of different classification schemes after applying data stacking and enhancement. The overlay from the fourth layer exhibited the best performance, while the accuracy of the third layer overlay declined. This decline can be attributed to the fact that during the stitching process, blocks containing rail damage (such as missing fasteners and cracks) were concatenated into normal blocks. The repeated stitching of multiple Fine Crack instances also led to decreases in accuracy for the Normal, Missing fastener, and Track fracture categories by 2.385%, 1.95%, and 3.35%, respectively. Overall, our image enhancements indeed played a positive role in improving the system's accuracy and robustness.

To assess the generalization capabilities of our model across varied scenarios, we conducted tests on both the improved and the original models over four distinct natural datasets. Figure 18 illustrates the classification accuracies of our model and the baseline model on these datasets, which include CIFAR10/100 [35], Describable Textures Dataset (DTD) [36], and Oxford 102 Flowers [37]. The outcomes indicate that our model continues to excel in fine-grained

TABLE 4. Ablation experiment results.

F_MB-Shuffle Block(without dropblock)	BF_MB-Shuffle Block	Shuffle conv	Focal-EIoU	mAP@0.5/%	Recall/%	F1%	FPS
-	-	-	-	0.8528	0.8522	0.8523	34
√	-	-	-	0.8885	0.8955	0.8810	50
-	√	-	-	0.8925	0.9058	0.8855	51
-	√	√	-	0.9154	0.9241	0.8911	67
-	√	√	√	0.9341	0.9559	0.9026	62

TABLE 5. Comparison of test accuracy of the original YOLOv5-Lite with our method at incremental enhancement levels.

Augmentation	Net	Normal	Missing fastener	Track fracture	Fine Crack
Nothing	YOLOv5-Lite	0.8743	0.8523	0.8332	0.8194
	Ours	0.9438	0.9023	0.9112	0.8913
i	YOLOv5-Lite	08836	0.8527	0.8429	0.8247
	Ours	0.9457	0.9112	0.9134	0.8923
i+ii	YOLOv5-Lite	0.8891	0.8472	0.8427	0.8328
	Ours	0.9621	0.9127	0.9258	0.9074
i+ii+iii	YOLOv5-Lite	0.8632	0.8354	0.8427	0.8429
	Ours	0.9383	0.8932	0.8923	0.9134
i+ii+iii+iiii	YOLOv5-Lite	0.8943	0.8532	0.8422	0.8359
	Ours	0.9829	0.9473	0.9344	0.9130

recognition tasks on the DTD. It also demonstrates commendable performance on the complex feature tasks of Flowers102. Moreover, our model outperforms the baseline model across all tasks, underscoring that our enhanced model can be generalized to a broad spectrum of classification detection challenges.

It's important to note that for the sake of fairness, we employed the same parameters and conducted tests on the CIFAR10 dataset. CIFAR10, organized by Hinton's students Alex Krizhevsky and Ilya Sutskever, is a compact dataset for generic object recognition. It comprises 10 categories of RGB color images: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks, totaling 50,000 training images and 10,000 testing images. The variety in size and shape of the points of interest within this dataset makes it a commendable choice for testing model robustness and recognition effectiveness.We evaluated algorithms such as GGS-Pf-YOLOv5, YOLOv8n, and YOLOv7-Tiny on the aforementioned CIFAR10 dataset. As evidenced by Table 6, YOLOv8n, which excels in terms of parameters, possesses only 6.2M, a reduction of 0.08M compared to ours. However, it compromised 2.1% in mAP@0.5% to achieve this

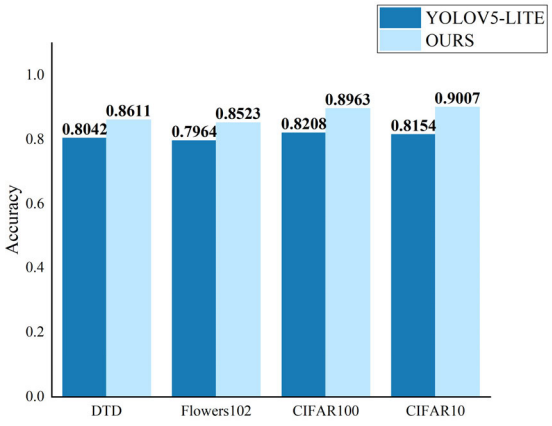

FIGURE 18. Comparison with our method and the original model on four classical natural datasets.

decrease in parameter size. This suggests that, in terms of balancing, YOLOv8n significantly lags behind our model. Our mAP@0.5 metric surpasses all comparative algorithms, with YOLOX-Tiny's 90.11% and YOLOv8n's 90.17% being the closest to our methodology. This highlights the superior

TABLE 6. Performance results for different networks on CIFAR10.

Network	Parameters(10^6)	Precision%	Recall%	mAP@0.5 %
YOLOv3-Tiny[41]	7.7	0.7550	0.7467	0.8180
YOLOv4-Mish	7.2	0.8174	0.8277	0.8117
YOLOv5s FMG[42]	5.76	0.8651	0.8773	0.8855
YOLOv7-Tiny[43]	7.1	0.8685	0.8521	0.8949
YOLOv8n[44]	**6.2**	0.8860	0.8158	0.9017
YOLOv8s	11.2	0.8678	0.8620	0.8831
GGS-Pf-YOLOv5[45]	7.06	0.8132	0.8364	0.8950
YOLOX-Tiny[46]	6.6	0.7940	0.8850	0.9011
Ours	6.12	**0.8873**	**0.8859**	**0.9041**

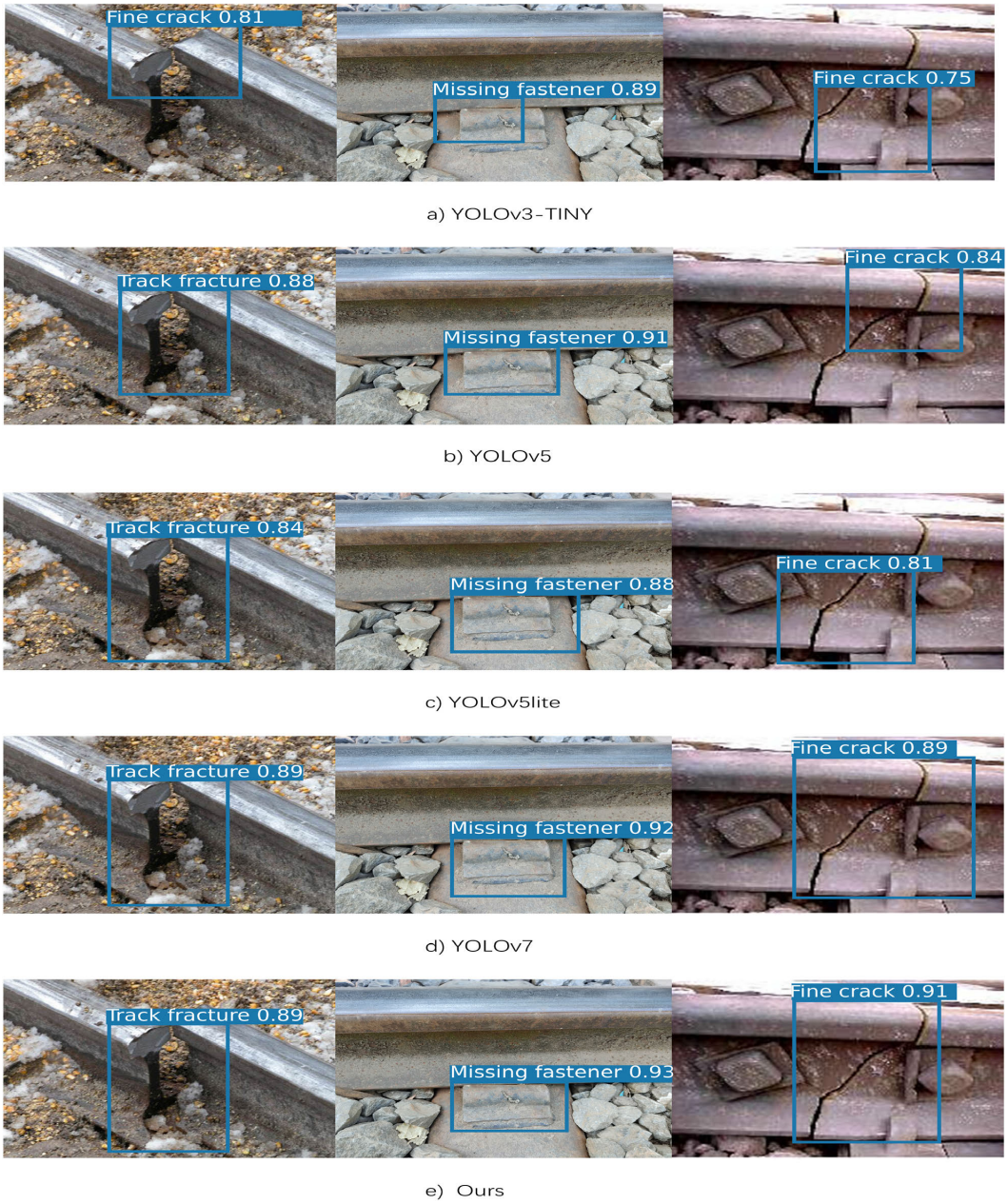

FIGURE 19. Regions of interest obtained from different algorithms.

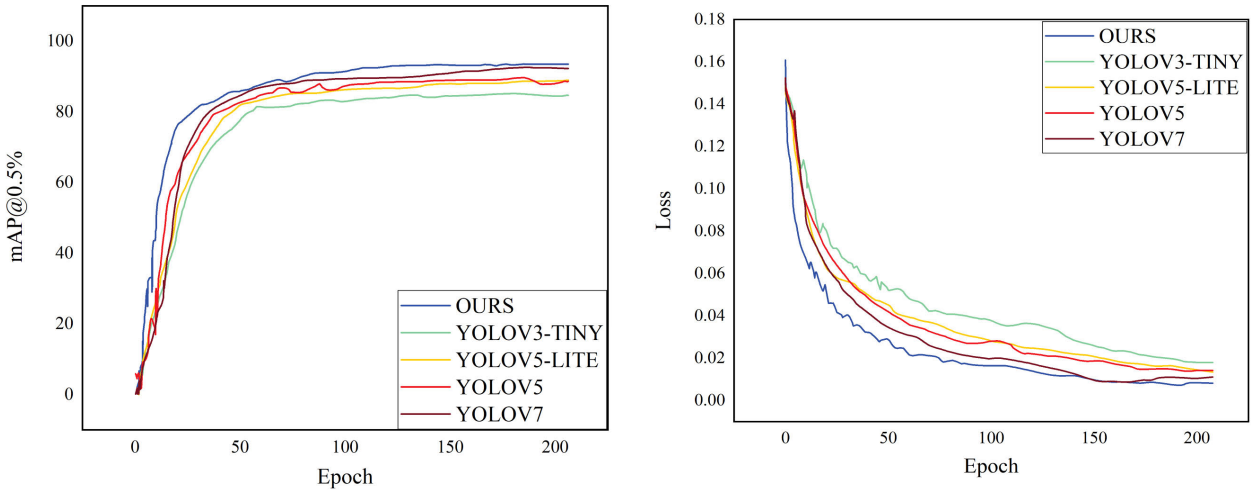

FIGURE 20. AP@0.5% and loss curves for 5 models on our dataset.

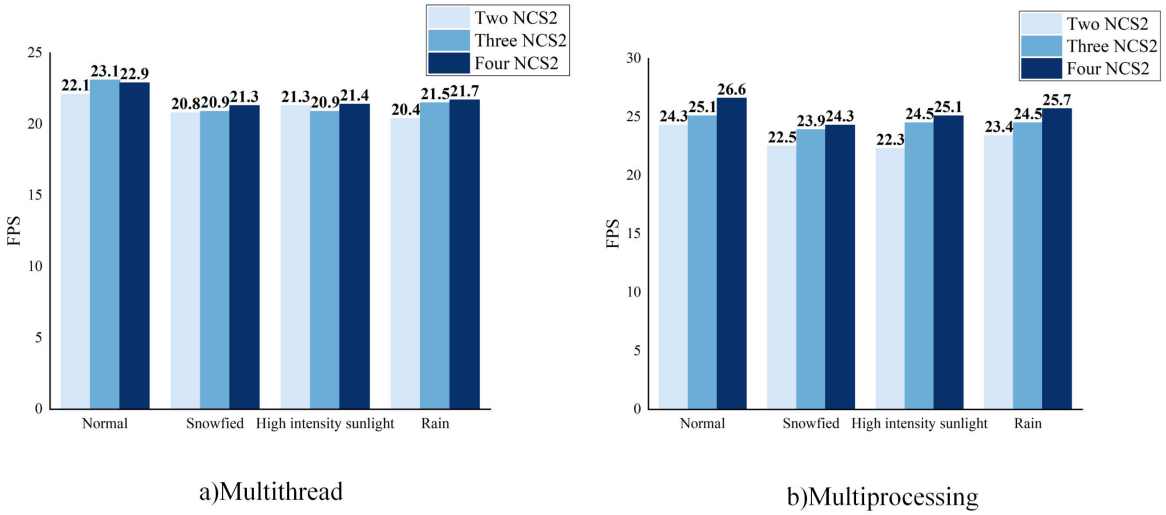

FIGURE 21. The performance demonstrated by the Multiprocessing and multithread models when installing different amounts of NCS2 under various simulated weather conditions.

detection performance of our model. The results emphasize our primary intention to strike an equilibrium between accuracy and speed, thus our model delivers a more harmonized performance. This infers that we can attain more accurate object detection outcomes at the cost of computational overhead, thereby meeting the real-time detection prerequisites of lightweight, low-energy portable systems. It also underscores the model's robustness across diverse datasets.

Subsequently, we evaluated several mainstream models on our dataset, including YOLOv3, YOLOv5, YOLOv5-lite, and YOLOv7, to assess their detection performance, with a particular emphasis on certain aspects of detection, as illustrated in Figure 19. The findings suggest that the approach offers high probability and accuracy in depicting the overall contours of damages. As evident from figures a) and c), YOLOv3-TINY and YOLOv5-lite demonstrated detection errors when addressing damages such as Track Fracture and Fine Crack, with notable similarity and relatively low confidence scores. In figures b) and d), while YOLOv5 and YOLOv7 did not exhibit missed detections, they showed instances of incomplete feature extraction during the detection process, and when compared to our model, also recorded lower confidence scores. Figure e) indicates that our model outperforms on our railway damage dataset. A specific focus was given to the mAP@0.5 training curve and the loss training curve, as depicted in Figure 20(a). The data suggests that, in terms of mAP@0.5, our approach significantly surpasses the original models YOLOv3-TINY, YOLOv5-lite, and YOLOv5, and also exhibits slightly superior performance when compared to YOLOv7. The loss curve in Figure 20(b) reveals that our method and YOLOv7 have comparable loss curves around Epoch 150, but by Epoch 200, our data demonstrates the best performance among the other four models.

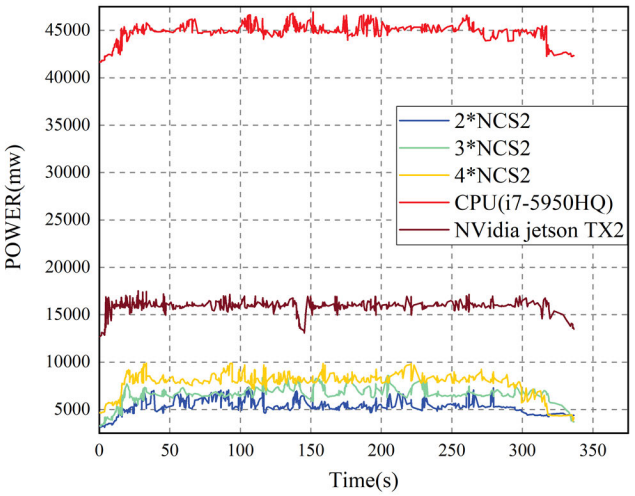

FIGURE 22. Power consumption and time relationship of different devices.

B. EXPERIMENTS WITH EDGE AI DEVICES

1) MULTIPROCESSING AND MULTITHREAD MODELS

In the experiment depicted in Figure 20. we evaluated the configurations of different quantities of NCS2 devices along with the FPS performance under multithread and Multiprocessing in four distinct weather conditions: snowy conditions, high-intensity sunlight, rainy weather, and regular weather. These tests validated the concerns posited in the previous section regarding the impact of the Global Interpreter Lock (GIL) on multithreading.

As observable in Figure 21(a). upon augmenting the quantity of NCS2 devices, there wasn't a notable enhancement in FPS, and, contrarily, more NCS2 led to a performance decrement in the testing environments of Normal and High-intensity sunlight. On the other hand, Figure 21(b).under the Multiprocessing mode, the increment of NCS2 indeed contributed to a performance improvement, amply demonstrating that adding more NCS2 devices does not significantly enhance performance in a multithreaded environment. Simultaneously, it elucidated that despite a slight performance reduction in our system under extreme driving conditions, as illustrated in Figure 21(b). where the worst performance with 2 NCS2 under strong sunlight conditions showed only a reduction of 1 in FPS across the four conditions, such minute alterations do not exert a consequential impact on our system. This robustly reflects the system's resilience.It should be noted that our inference experiments were conducted on an i7-5950HQ processor.

2) EDGE AI DEVICE PERFORMANCE COMPARISON

In Figure 22. to assess the deployability and energy efficiency ratio of the system, we examined the energy consumption relationship of our method with four different platforms. It can be observed that our method, when implemented using 2*NCS2 devices, exhibited the lowest energy consumption, with an average power usage of merely 5568mw. The power consumption of a single NCS2 stick is approximately 1.2 watts. In comparison, although devices like the CPU may exhibit superior performance, their energy consumption, around 45 watts, is significantly higher than that of edge AI devices.

In TABLE 7, we evaluated the relationship of low energy consumption and performance for our method compared with other four different platforms. From the perspective of selecting the quantity of NCS2 sticks in our system, the performance improvement brought about by the hardware accumulation of more than two NCS2 sticks is not sufficient to compensate for the increase in power consumption it incurs. Moreover, we can observe that although the CPU provides quite satisfactory detection results, it accompanies high energy consumption. In contrast, the device setup with two NCS2 sticks that we chose demonstrates the most excellent performance under the circumstances where both power consumption and performance are taken into account.

Lastly, we conducted an intriguing experiment to compare the price ratio per frame per second, as demonstrated in Table 8. If our objective is to strike a balance between performance and cost, this appears to be the optimal choice, with its Average FPS/Price reaching 0.1093. It's evident that the CPU does indeed offer superior performance, but it requires accompanying hardware and comes at a steeper price. With a base of 50.3 FPS, its Average FPS/Price is a mere 0.1043, not as favorable as our device. Moreover, considering the long-term prospects of deployment on trains, the combination of Multi (2*NCS2) + Raspberry Pi 4B is truly a commendable option.

In TABLE 7, we evaluated the relationship of low energy consumption and performance for our method compared with other four different platforms. From the perspective of selecting the quantity of NCS2 sticks in our system, the performance improvement brought about by the hardware accumulation of more than two NCS2 sticks is not sufficient to compensate for the increase in power consumption it incurs. Moreover, we can observe that although the CPU provides quite satisfactory detection results, it accompanies high energy consumption. In contrast, the device setup with two NCS2 sticks that we chose demonstrates the most excellent performance under the circumstances where both power consumption and performance are taken into account.

Lastly, we conducted an intriguing experiment to compare the price ratio per frame per second, as demonstrated in Table 8. If our objective is to strike a balance between performance and cost, this appears to be the optimal choice, with its Average FPS/Price reaching 0.1093. It's evident that the CPU does indeed offer superior performance, but it requires accompanying hardware and comes at a steeper price. With a base of 50.3 FPS, its Average FPS/Price is a mere 0.1043, not as favorable as our device. Moreover, considering the long-term prospects of deployment on trains, the combination of Multi (2*NCS2) + Raspberry Pi 4B is truly a commendable option.

TABLE 7. Performance ratio analysis of different platforms.

Platform	Average Power	Average FPS	Average Power /Average FPS
Nvidia Jetson TX2	17526	27.2	644.33
CPU(i7-5950HQ) and supporting equipment	44567	50.3	886.02
NCS2+Raspberry Pi 4B	5568	9.4	592.34
(2*NCS2)+Raspberry Pi 4B	6762	18.7	**361.60**
(3*NCS2)+Raspberry Pi 4B	8053	19.1	421.62
(4*NCS2)+Raspberry Pi 4B	9365	19.9	470.60

TABLE 8. Price/performance analysis of the different embedded platforms.

Platform	Price $	Average FPS	Average FPS/Price
Multi(2*NCS2)+Raspberry Pi 4B	171	18.7	**0.1093**
Nvidia Jetson TX2	300	27.2	0.0906
CPU (i7-5950HQ)and supporting equipment	482	50.3	0.1043

V. DISCUSSION

In our study, concerning the direction of the network model, we adopted three strategies to enhance our approach. By modifying the lightweight model of the original YOLOV5s version, namely YOLOv5-lite, we derived our custom BF_MB-YOLOv5 network. In the first strategy, we initially intended to replace the standard convolution in the backbone part with MBConv, which possesses high detection accuracy and low complexity. However, we realized that the Depthwise Conv in MBConv could not fully utilize hardware acceleration performance, which is fatal for our aspiration to deploy portable edge AI acceleration devices. Consequently, we adopted the fused-MBconv structure, which replaces Depthwise Conv with standard convolution. Ultimately, considering that our recognition target features are quite integrated, the characteristic of random dropping in dropout is far inferior to our DROPBLOCK in terms of feature extraction. In the second strategy, we employed Group Conv structure to replace the 1*1 channel separation convolution in the neck to derive a more lightweight model. In the third strategy, we used the Focal-EIoU Loss, which has superior regression accuracy to address the existing sample imbalance issue. It can be seen from TABLE 3 that each strategy brought positive results to our network in both detection speed and detection accuracy. Compared to the original model, our mAP value overall increased by 8.13%, recall rate improved by 10.37%, while the detection speed per second elevated by 28 frames

In the aspect of edge AI acceleration device deployment, we investigated the enhancement of detection speed in our deployed detection network by the NCS2 acceleration stick. In TABLE 7 and TABLE 8, we delved into the issues of detection speed, power consumption, and cost. It is observable that our Average Power/Average FPS is the most outstanding among the platforms compared, as low as 361.60, indicating that our device performs quite well in the domain of low power consumption and high performance. In terms of Average FPS/Price, we also excelled with a high value of 0.1093.In TABLE 9, we conducted an in-depth analysis of several related literature, encompassing edge artificial intelligence (Edge AI) devices, network models, and their cost-effectiveness analysis.

We meticulously compared the network models in different literature, including their key indices such as mAP, FPS, and COST/FPS. Through contrasting these indices, we can better comprehend the discrepancies in performance and cost-effectiveness among different models and devices. It is discernible that our model exhibits the most splendid performance in mAP, while in the aspect of FPS, the NVIDIA Jetson AGX Xavier with Yolov4-tiny in [49] is the most remarkable. However, it is not difficult to notice that the price of NVIDIA Jetson AGX Xavier is exceedingly expensive, almost five times as much as our method, and its Cost/FPS is also as high as 40.8, far exceeding our 9.14. The lowest cost is the first-generation NCS with Raspberry Pi 3B method, with a

TABLE 9. Performance of devices and network models in different studies.

References	Edge AI device	Net	Cost($)	FPS	mAP	Cost/FPS
Mazzia at al [16]	Nvidia Jetson Nano	YOLOv3-tiny	99	8	0.8364	12.375
Adami at al [47]	NCS+Raspberry Pi 3B	YOLOv3	75	5	0.8250	15.5
Liu at al[48]	NVIDIA TX2	Their own YOLO	300	10	0.8977	30.0
Lin at al[49]	NVIDIA Jetson AGX Xavier	YOLOv4-tiny	857	21	0.8734	40.8
Atitallah at al [50]	FPGA:ZCU 102	Improved YOLOV5	356	18.5	0.8342	19.24
Ours	2*NCS2+Raspberry Pi 4B	BF_MB-YOLOv5	171	18.7	0.9441	9.14

cost of merely 75 dollars, yet its FPS is only 5, leading to a Cost/FPS as high as 12.357. It is evident that in balancing cost and performance, our method indeed performs excellently.

Currently, most research in the direction of industrial detection problems overly emphasizes performance, unrestrainedly utilizing high power consumption, high-cost devices. Our research has made optimizations based on this foundation, which is a crucial aspect in industrial issues.

VI. CONCLUSION

This work proposes a system termed as BF_MB-YOLOv5, laden on Raspberry Pi 4B coupled with two NCS2 acceleration sticks, for the deployment of a railway damage detection network. This is an enhancement of YOLOv5-lite, designed to real-time detection of railway damages. The system is capable of real-time railway condition detection at low cost when mounted on mobile devices aboard trains. Improvements were made on YOLOv5-lite to maintain a balance between model complexity and model performance. In terms of the dataset, we utilized four effective image augmentation and enhancement techniques to mitigate the issue of dataset diversity deficiency. On the network model front, an improved MBConv replaced the conventional convolution structure in the model backbone structure of ShuffleNet V2, wherein BF_MBconv introduced the SE attention mechanism and employed the lighter and more portable-device adapted Fused-MBConv structure, ultimately incorporating a more precise dropblock structure. The overall modifications not only enhanced model performance but also addressed the issue of MBconv being unable to fully utilize hardware performance. Improvements were made on the convolution operations in the two Conv modules in Neck, replaced with Shuffle Conv modules combining Group Conv modules and Channel Shuffle modules, constructing a lighter model. Furthermore, Focal-EIoU Loss was utilized to replace the original loss function, rendering the model more precise and robust in bounding box regression tasks. Lastly, we explored an edge AI device that, under reasonable economic cost and power consumption circumstances, can still maintain real-time detection efficiency. Experimental results demonstrate that our method can enhance the detection accuracy of multi-class railway defects, possessing certain advantages compared to other algorithms. It effectively resolves the issues of low recognition efficiency, low recognition accuracy, lack of energy-saving in onboard devices, and high deployment cost in real-time railway damage detection, providing a good reference for addressing safety issues in railway transportation.

REFERENCES

[1] M. Wang, K. Li, X. Zhu, and Y. Zhao, "Detection of surface defects on railway tracks based on deep learning," *IEEE Access*, vol. 10, pp. 126451–126465, 2022.

[2] C. Liu, N. Li, H. Wu, and X. Meng, "Detection of high-speed railway subsidence and geometry irregularity using terrestrial laser scanning," *J. Surveying Eng.*, vol. 140, no. 3, Aug. 2014, Art. no. 04014009.

[3] I. L. Al-Qadi, W. Xie, and R. Roberts, "Scattering analysis of ground-penetrating radar data to quantify railroad ballast contamination," *NDT E Int.*, vol. 41, no. 6, pp. 441–447, Sep. 2008.

[4] Z. Leng and I. L. Al-Qadi, "Railroad ballast evaluation using ground-penetrating radar: Laboratory investigation and field validation," *Transp. Res. Rec., J. Transp. Res. Board*, vol. 2159, no. 1, pp. 110–117, Jan. 2010.

[5] P. Rizzo, M. Cammarata, I. Bartoli, F. L. D. Scalea, S. Salamone, S. Coccia, and R. Phillips, "Ultrasonic guided waves-based monitoring of rail head: Laboratory and field tests," *Adv. Civil Eng.*, vol. 2010, pp. 1–13, Dec. 2010.

[6] B. Yang and L. Fang, "Automated extraction of 3-D railway tracks from mobile laser scanning point clouds," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 12, pp. 4750–4761, Dec. 2014.

[7] S. S. Artagan, L. Bianchini Ciampoli, F. D'Amico, A. Calvi, and F. Tosti, "Non-destructive assessment and health monitoring of railway infrastructures," *Surveys Geophysics*, vol. 41, no. 3, pp. 447–483, May 2020.

[8] H.-J. Mun and M.-H. Lee, "Design for visitor authentication based on face recognition technology using CCTV," *IEEE Access*, vol. 10, pp. 124604–124618, 2022.
[9] G. Li, Z. Ji, X. Qu, R. Zhou, and D. Cao, "Cross-domain object detection for autonomous driving: A stepwise domain adaptative YOLO approach," *IEEE Trans. Intell. Vehicles*, vol. 7, no. 3, pp. 603–615, Sep. 2022.
[10] H. Qi, T. Xu, G. Wang, Y. Cheng, and C. Chen, "MYOLOv3-tiny: A new convolutional neural network architecture for real-time detection of track fasteners," *Comput. Ind.*, vol. 123, Dec. 2020, Art. no. 103303.
[11] Y. Li, Z. Han, H. Xu, L. Liu, X. Li, and K. Zhang, "YOLOv3-lite: A lightweight crack detection network for aircraft structure based on depth-wise separable convolutions," *Appl. Sci.*, vol. 9, no. 18, p. 3781, Sep. 2019.
[12] Q. Qiu and D. Lau, "Real-time detection of cracks in tiled sidewalks using YOLO-based method applied to unmanned aerial vehicle (UAV) images," *Autom. Construct.*, vol. 147, Mar. 2023, Art. no. 104745.
[13] J. Jeon, J. Kim, J.-K. Kang, S. Moon, and Y. Kim, "Target capacity filter pruning method for optimized inference time based on YOLOv5 in embedded systems," *IEEE Access*, vol. 10, pp. 70840–70849, 2022.
[14] Y. Xu, H.-K. Lam, G. Jia, J. Jiang, J. Liao, and X. Bao, "Improving COVID-19 CT classification of CNNs by learning parameter-efficient representation," *Comput. Biol. Med.*, vol. 152, Jan. 2023, Art. no. 106417.
[15] M. Anandhalli and V. P. Baligar, "A novel approach in real-time vehicle detection and tracking using raspberry pi," *Alexandria Eng. J.*, vol. 57, no. 3, pp. 1597–1607, Sep. 2018.
[16] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, "Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application," *IEEE Access*, vol. 8, pp. 9102–9114, 2020.
[17] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[18] *Introducing the Intel Neural Compute Stick 2—Intel's Next Generation AI Inference Development Kit*, Intel, Santa Clara, CA, USA, 2019. [Online]. Available: https://www.intel.cn/content
[19] *NCS_Product-Briefv4*, Intel, Santa Clara, CA, USA, 2023. [Online]. Available: https://www.intel.com/content/dam/support/cn/zh/documents/boardsandkits/neural-compute-sticks/NCS2_Product-Brief-English.pdf
[20] *Nvidia Jetson AGX Xavier Brief Introduction*, Nvidia, Santa Clara, CA, USA, 2023. [Online]. Available: https://www.nvidia.cn/autonomous-machines/embedded-systems-dev-kits-modules/
[21] Mohebbanaaz, Y. P. Sai, and L. V. R. Kumari, "Cognitive assistant DeepNet model for detection of cardiac arrhythmia," *Biomed. Signal Process. Control*, vol. 71, Jan. 2022, Art. no. 103221.
[22] T.-T. Dao, Q.-V. Pham, and W.-J. Hwang, "FastMDE: A fast CNN architecture for monocular depth estimation at high resolution," *IEEE Access*, vol. 10, pp. 16111–16122, 2022.
[23] S. Mittal, "A survey on optimized implementation of deep learning models on the NVIDIA Jetson platform," *J. Syst. Archit.*, vol. 97, pp. 428–442, Aug. 2019.
[24] X. Xu, X. Zhang, and T. Zhang, "Lite-YOLOv5: A lightweight deep learning detector for on-board ship detection in large-scene Sentinel-1 SAR images," *Remote Sens.*, vol. 14, no. 4, p. 1018, Feb. 2022.
[25] Z.-F. Xu, R.-S. Jia, Y.-B. Liu, C.-Y. Zhao, and H.-M. Sun, "Fast method of detecting tomatoes in a complex scene for picking robots," *IEEE Access*, vol. 8, pp. 55289–55299, 2020.
[26] D. Pestana, "A full featured configurable accelerator for object detection with YOLO," *IEEE Access*, vol. 9, pp. 75864–75877, 2021.
[27] *Railway Track Fault Detection Kaggle*. [Online]. Available: https://www.kaggle.com/datasets/salmaneunus/railway-track-fault-detection
[28] A. Xiao, B. Shen, J. Tian, and Z. Hu, "Differentiable RandAugment: Learning selecting weights and magnitude distributions of image transformations," *IEEE Trans. Image Process.*, vol. 32, pp. 2413–2427, 2023.
[29] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," 2017, *arXiv:1710.09412*.
[30] B. Cheng, Y. Su, and Y. Cai, "Research on real-time human fall detection method based on YOLOv5-lite," in *Proc. IEEE 6th Int. Conf. Electron. Inf. Commun. Technol. (ICEICT)*, Qingdao, China, Jul. 2023, pp. 218–221.
[31] D. B. Payne and J. R. Stern, "Wavelength-switched passively coupled single-mode optical network," in *Proc. IOOC-ECOC*, Boston, MA, USA, 1985, pp. 585–590.
[32] M. Tan and Q. V. Le, "EfficientNetV2: Smaller models and faster training," 2021, *arXiv:2104.00298*.
[33] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 2011–2023, Aug. 2020.
[34] S. Wu, S. Zhang, and X. Jing, "An industrial meter detection method based on lightweight YOLOX-CAlite," *IEEE Access*, vol. 11, pp. 3573–3583, 2023, doi: 10.1109/ACCESS.2022.3229874.
[35] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.
[36] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 3606–3613.
[37] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Proc. 6th Indian Conf. Comput. Vis., Graph. Image Process.*, Dec. 2008, pp. 722–729.
[38] X. S. Chen, "Research and application of road disease detection and recognition algorithm based on raspberry PI," Jianghan Univ., Wuhan, China, Tech. Rep., 2022, doi: 10.27800/D.CNKI.GJHDX.2022.000180.
[39] J. Mas, T. Panadero, G. Botella, A. A. Del Barrio, and C. García, "CNN inference acceleration using low-power devices for human monitoring and security scenarios," *Comput. Electr. Eng.*, vol. 88, Dec. 2020, Art. no. 106859.
[40] Python Documentation. *Thread State and the Global Interpreter Lock*. [Online]. Available: https://docs.python.org/3/c-api/init.html#thread-state-an
[41] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[42] Y. Zheng, Y. Zhan, X. Huang, and G. Ji, "YOLOv5s FMG: An improved small target detection algorithm based on YOLOv5 in low visibility," *IEEE Access*, vol. 11, pp. 75782–75793, 2023.
[43] C.-Y. Wang, A. Bochkovskiy, and H.-Y. Mark Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022, *arXiv:2207.02696*.
[44] E. Casas, L. Ramos, E. Bendek, and F. Rivas-Echeverría, "Assessing the effectiveness of YOLO architectures for smoke and wildfire detection," *IEEE Access*, vol. 11, pp. 96554–96583, 2023.
[45] Q. Xiao, Q. Li, and L. Zhao, "Lightweight sea cucumber recognition network using improved YOLOv5," *IEEE Access*, vol. 11, pp. 44787–44797, 2023.
[46] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," 2021, *arXiv:2107.08430*.
[47] D. Adami, M. O. Ojo, and S. Giordano, "Design, development and evaluation of an intelligent animal repelling system for crop protection based on embedded edge-AI," *IEEE Access*, vol. 9, pp. 132125–132139, 2021.
[48] Z. Liu and S. Wang, "Broken corn detection based on an adjusted YOLO with focal loss," *IEEE Access*, vol. 7, pp. 68281–68289, 2019.
[49] C.-J. Lin and J.-Y. Jhang, "Intelligent traffic-monitoring system based on YOLO and convolutional fuzzy neural networks," *IEEE Access*, vol. 10, pp. 14120–14133, 2022.
[50] A. B. Atitallah, Y. Said, M. A. B. Atitallah, M. Albekairi, K. Kaaniche, T. M. Alanazi, S. Boubaker, and M. Atri, "Embedded implementation of an obstacle detection system for blind and visually impaired persons' assistance navigation," *Comput. Electr. Eng.*, vol. 108, no. 1, Jan. 2023, Art. no. 108714.

CHAO DANG is currently pursuing the master's degree in engineering with Lanzhou Jiaotong University. His research interests include computer vision, image processing, FPGA, embedded systems, and machine learning.

ZAIXING WANG received the Ph.D. degree from Lanzhou University, Lanzhou, China, in 2008. He is currently an Associate Professor with Lanzhou Jiaotong University, Lanzhou. His research interests include novel device simulation and model, computer software and application, radio electronics, and automation technology.

YONGHUAN HE is currently pursuing the master's degree in engineering with Lanzhou Jiaotong University. His primary research interests include computer vision, image processing, and FPGA.

LINCHANG WANG is currently pursuing the master's degree in engineering with Lanzhou Jiaotong University. His primary research interests include semiconductor power devices and fundamentals of semiconductor devices.

YI CAI is currently pursuing the master's degree in engineering with Lanzhou Jiaotong University. Her primary research interests include semiconductor power devices and fundamentals of semiconductor devices.

HUIJI SHI is currently pursuing the master's degree in engineering with Lanzhou Jiaotong University. His primary research interests include computer vision, image processing, FPGA, embedded systems, and machine learning.

JIACHI JIANG is currently pursuing the master's degree in engineering with Lanzhou Jiaotong University. Her primary research interests include semiconductor power devices, fundamentals of semiconductor devices, MPS semiconductor devices, computer vision, image processing, FPGA, and embedded systems.