GEORGE BROWN PRESS
JOURNALS, BOOKS & PROCEEDINGS

Journal of Computer, Signal, and System Research

Article

Road Defect Detection System Based on Deep Learning

Shijie Lin [1,*]

[1] School of Computer Science and Big Data, Minjiang College, Fuzhou, 350000, China
* Correspondence: Shijie Lin, School of Computer Science and Big Data, Minjiang College, Fuzhou, 350000, China

Abstract: Road defect detection plays a vital role in the development of smart cities, enhancing the efficiency of road maintenance and providing reliable perception information for automated repair and inspection technologies. This article presents a deep learning-based road defect detection model, built upon advanced frameworks such as YOLOv8 and YOLOv5. Trained on a large-scale dataset of road images, the model is capable of accurately identifying common defect types, including cracks, potholes, and pavement deterioration. In addition, we developed a comprehensive road defect detection system featuring a user-friendly graphical interface, which supports real-time detection and visualization of road defects. Implemented using Python and PyQt5, the system enables intuitive display of detection results and provides detailed information for road maintenance planning. The proposed approach demonstrates promising performance in both static image recognition and continuous video monitoring, offering potential applications in intelligent transportation systems, urban road management, and automated infrastructure maintenance.

Keywords: YOLOv8; deep learning; road defect; real-time detection; PyQt5

1. Introduction

Road defect detection plays a crucial role in ensuring traffic safety, prolonging the service life of roads, and supporting the sustainable development of modern smart cities. With the rapid expansion of urban infrastructure, roads are subjected to heavy traffic loads and complex environmental conditions, which often lead to cracks, potholes, and other types of surface damage [1]. If these defects are not identified and repaired in a timely manner, they may not only accelerate the deterioration of road surfaces but also pose serious threats to drivers, pedestrians, and overall traffic flow. Therefore, the ability to detect road defects accurately and efficiently has become a fundamental requirement for urban management and transportation safety.

In the context of smart city construction, the integration of intelligent road defect detection systems has greatly enhanced the capacity of city administrators to monitor and maintain infrastructure. Compared with traditional manual inspection methods, which are often labor-intensive, time-consuming, and prone to human error, automated defect detection provides significant advantages. It enables real-time monitoring, reduces the need for large-scale manual inspections, and offers data-driven support for decision-making in maintenance planning. This contributes not only to lowering long-term maintenance costs but also to improving the reliability of transportation networks and the quality of urban life [2].

Automated road defect detection systems have been increasingly applied in various domains, including municipal road maintenance, traffic safety supervision, and infrastructure asset management. By deploying advanced computer vision techniques, city management departments can receive immediate feedback regarding road surface conditions and respond promptly with targeted repair measures. Such intelligent systems not

Received: 25 July 2025
Revised: 04 August2025
Accepted: 14 August 2025
Published: 21 August 2025

only improve the timeliness of maintenance but also optimize resource allocation by prioritizing the most urgent repair tasks. As a result, the efficiency and effectiveness of road management are substantially enhanced.

To address these needs, this study collected a dataset consisting of images and video materials of different road defects and developed a road defect detection system with an intuitive and user-friendly interface. The system was implemented using Python and PyQt5, and it integrates object detection models such as YOLOv8 and YOLOv5. It supports multiple input modes, including static images, recorded videos, and real-time camera feeds, and is capable of saving detection results for further analysis. This flexibility provides users with a comprehensive and practical tool for defect detection in diverse scenarios [3].

As illustrated in Figure 1, YOLOv8 demonstrates significant accuracy improvements over YOLOv5. However, these improvements are accompanied by substantial increases in model parameters and floating-point operations (FLOPs), which consequently lead to slower inference speeds in most cases. Although numerous enhancements in the YOLO series have achieved remarkable performance gains on large-scale benchmark datasets such as COCO, their generalization ability on custom, domain-specific datasets has not been fully validated. In fact, YOLOv5 is still widely recognized for its relatively strong generalization performance and computational efficiency. Taking these trade-offs into account, the system developed in this study adopts both YOLOv8 and YOLOv5 for computer vision tasks, aiming to leverage the complementary strengths of the two models in practical applications.

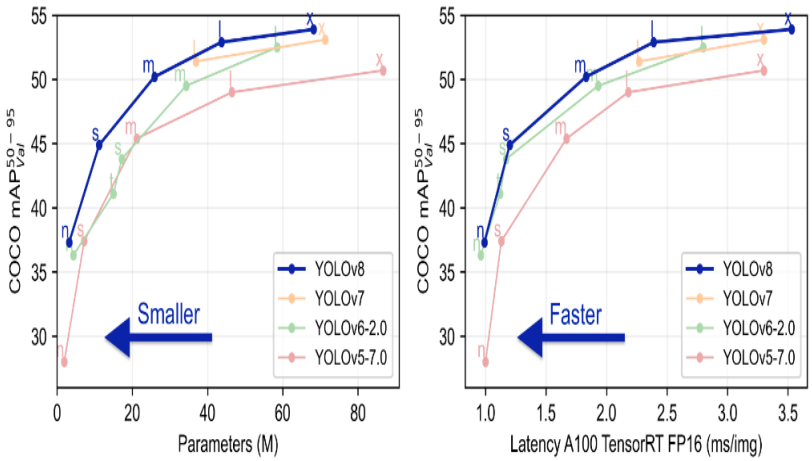

Figure 1. Official mAP, parameter count, and FLOPs results tested on the COCO Val 2017 dataset.

2. Overall System Design

2.1. Overall Design Concept

The overall architecture of the proposed road defect detection system is illustrated in Figure 2, which outlines the sequential workflow from initialization to result output. The process begins with program startup, during which the main window is initialized and core functional modules are loaded. At this stage, essential configuration files and model parameters are also read into memory to ensure consistent operation. The YOLOv8 model is then initialized as the core detection engine, while the system simultaneously establishes signal–slot connections in PyQt5 to bind user actions with backend functions. This event-driven design allows the interface to respond dynamically to user commands, such as selecting an input source or triggering the detection task, thereby ensuring smooth interaction between the graphical user interface and computational modules [4].

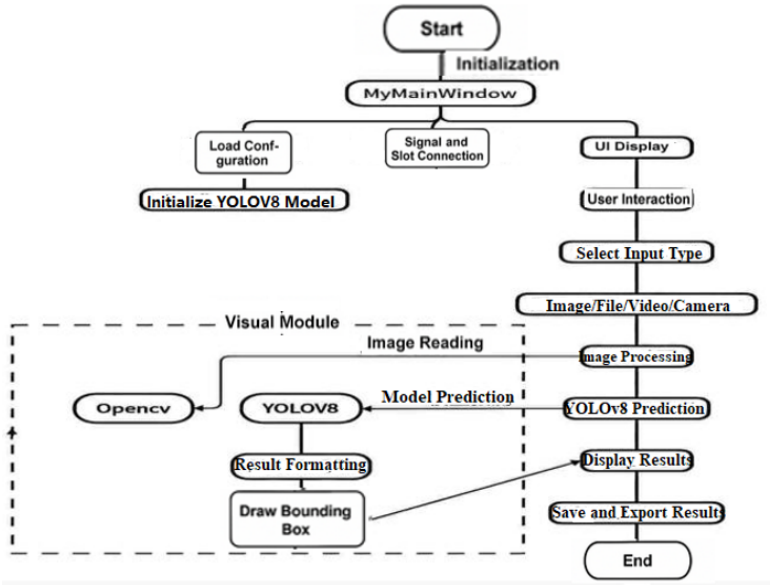

Figure 2. System flow chart.

Once initialization and logic binding are completed, the user interface is launched, providing an intuitive platform through which users can operate the system. Multiple input sources are supported, including static images, video files, and live camera streams, all of which are processed by OpenCV before being passed to the detection pipeline. The YOLOv8 model performs object detection and classification on the input data, generating bounding boxes, confidence scores, and labels that are superimposed on the original frame. These annotated results are displayed in real time on the interface, giving users an immediate and intuitive understanding of road conditions. For convenience and practical application, the system also allows prediction results to be saved locally. Furthermore, the PyQt5-based interface was designed with usability in mind, combining modular functionality with a user-friendly layout to make the detection process accessible to both technical and non-technical users.

2.2. YOLOV8 Model Introduction

As shown in Figure 3, the YOLOv8 network architecture is composed of three primary components: the Backbone, the Neck, and the Head. The Backbone is responsible for feature extraction and employs a combination of convolutional and deconvolutional layers, residual connections, and bottleneck structures to effectively reduce model size while maintaining accuracy. At the core of this design is the C2f module, which replaces the C3 module used in YOLOv5. The C2f module introduces a more compact structure with fewer parameters while preserving strong feature extraction capabilities, thereby reducing redundancy and improving computational efficiency. In addition, the Backbone integrates enhancements such as depthwise separable convolution and dilated convolution, both of which contribute to richer feature representation across multiple receptive fields. These optimizations collectively enable YOLOv8 to extract higher-quality features compared with its predecessors [5].

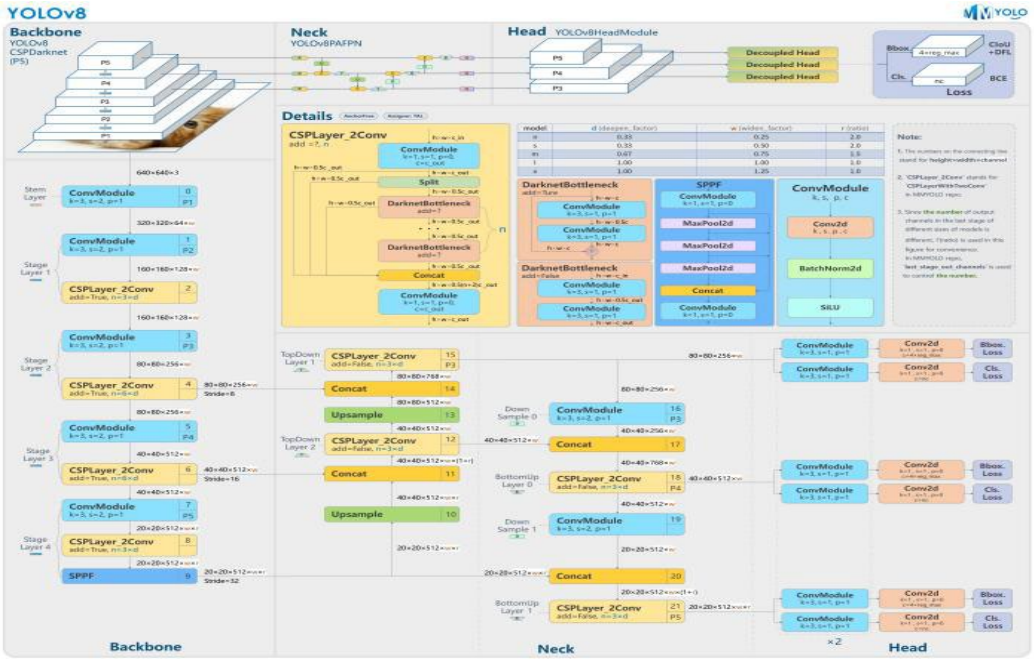

Figure 3. YOLOv8 network structure diagram.

The Neck component is dedicated to multi-scale feature fusion, ensuring that information captured at different stages of the Backbone is effectively integrated [6]. It incorporates several key modules to achieve this goal [7]. The SPPF (Spatial Pyramid Pooling Fast) module aggregates feature maps at multiple scales through pooling operations, improving the detection of objects of varying sizes. The PAA (Probabilistic Anchor Assignment) module provides adaptive anchor box assignment, which improves the balance between positive and negative samples during training and ultimately enhances model robustness. Furthermore, the Path Aggregation Network (PAN) module is used twice within the Neck to reinforce bottom-up and top-down information flow, thereby enhancing the representational power of fused features. Together, these modules significantly strengthen YOLOv8's ability to detect road defects of different shapes and scales with improved precision and efficiency.

2.3. Model Training Part

As shown in Figure 4, the model training process begins by specifying two separate configuration file paths along with the dataset configuration path, which are essential for initializing the YOLO network. In this study, both the baseline YOLOv8s model and an enhanced version integrated with the SE (Squeeze-and-Excitation) attention mechanism were trained to explore potential improvements in feature representation and detection performance. The model.train function was employed to set the training parameters, where the number of epochs was fixed at 200 to ensure sufficient iterations for convergence, and the batch size was set to 2 to accommodate the computational limitations of the available hardware. During training, the network iteratively updates its weights through backpropagation, minimizing a composite loss function that includes bounding box regression, objectness, and classification losses. Standard data augmentation techniques, such as random scaling, flipping, and color jittering, were applied to enhance model generalization and robustness [6].

```python
import os
from ultralytics import YOLO

if __name__ == '__main__':
    # 原yolov8s
    yaml_yolov8s = 'ultralytics/cfg/models/v8/yolov8s.yaml'
    # SE 注意力机制
    yaml_yolov8_SE = 'ultralytics/cfg/models/v8/det_self/yolov8s-attention-SE.yaml'

    # 替换一下变量名即可
    model_yaml = yaml_yolov8s
    # 模型加载
    model = YOLO(model_yaml)
    # 数据集路径的yaml文件
    data_path = r'config\traindata.yaml'
    # 以yaml文件的名字进行命名
    name = os.path.basename(model_yaml).split('.')[0]
    # 文档中对参数有详细的说明
    model.train(data=data_path,              # 数据集
                imgsz=640,                   # 训练图片大小
                epochs=200,                  # 训练的轮次
                batch=2,                     # 训练batch
                workers=0,                   # 加载数据线程数
                device='0',                  # 使用显卡
                optimizer='SGD',             # 优化器
                project='runs/train',        # 模型保存路径
                name=name,                   # 模型保存命名
                )
```

Figure 4. Main parts of model training.

Upon completion of the training process, the system outputs a weight file containing the optimized parameters of the trained network. This weight file can then be loaded into the main.py script, allowing the trained model to be directly applied to road defect detection tasks. By utilizing the enhanced YOLOv8 model with SE attention, the system achieves improved detection accuracy, better localization of road defects, and increased robustness under varying lighting and background conditions. Overall, this training procedure ensures that the system can perform reliable and efficient road defect recognition, providing a solid foundation for practical applications in urban road management and intelligent transportation systems.

2.4. Model Evaluation Part

After model evaluation using the val.py script, loss and accuracy curves were generated, as illustrated in Figure 5. The top row of Figure 5 represents the evaluation results on the training set, while the bottom row shows the evaluation on the validation set. The evaluation metrics include val/box_loss, val/cls_loss, and val/dfl_loss. Specifically, val/box_loss represents the bounding box regression loss on the validation dataset. Its downward trend, consistent with that observed during training, indicates that the model is gradually fitting the validation data and improving localization accuracy. val/cls_loss corresponds to the classification loss on the validation dataset, and the significant decrease demonstrates an enhancement in the model's ability to correctly classify road defect categories. val/dfl_loss, or distribution focus loss, reflects the model's object location regression performance, and its continuous decline further confirms that the network accurately predicts bounding box positions.

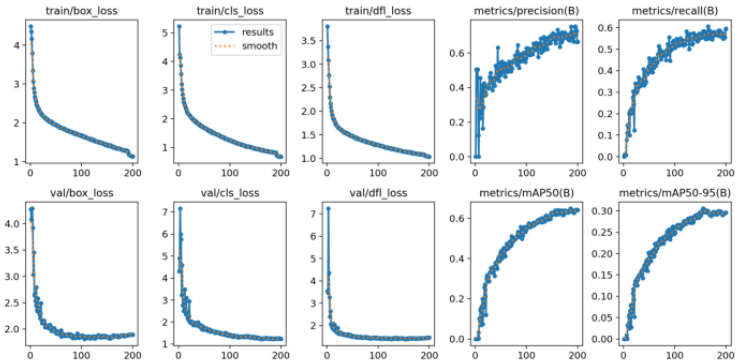

Figure 5. Changes in model loss and accuracy.

In addition to the loss metrics, detection performance was assessed using mean Average Precision (mAP) metrics. metrics/mAP50(B) indicates the mean Average Precision at an IoU threshold of 50%, and its steadily rising curve suggests consistent improvement in detection performance on the validation set. Furthermore, metrics/mAP50-95(B) represents the average mAP over multiple IoU thresholds from 0.5 to 0.95. The gradual increase in this metric demonstrates that the model maintains robust detection capabilities even under stricter localization requirements. Overall, these evaluation results indicate that the trained YOLO model exhibits both accurate classification and precise localization on unseen data, confirming its effectiveness and generalization ability for practical road defect detection tasks.

2.5. Model Detect Detection Part

As illustrated in Figure 6, the main function integrates the YOLO model to perform object prediction on incoming images. The workflow begins by invoking the trained YOLO model and feeding the captured image into the detection pipeline. Once the prediction process is executed, the system retrieves the recognition results, including the predicted object category and the corresponding confidence score. To ensure clear visualization, a bounding box is automatically drawn around each detected object, providing users with intuitive spatial information regarding defect location [7]. In scenarios where the model does not return any recognition results, the program is designed to assign default values for subsequent operations, thereby avoiding interruptions in the processing pipeline and maintaining overall system stability. This design ensures that the detection process remains robust, even in the absence of valid predictions, and lays the foundation for subsequent visualization and analysis. Overall, the implementation shown in Figure 6 highlights the role of the main function as the core entry point of the detection workflow, bridging the trained YOLO model with real-time input images and enabling seamless transition from prediction to result visualization.

```
def predict_img(self, img):
    # 初始化结果信息字典
    result_info = {}
    # 记录开始时间以计算处理时间
    t1 = time.time()
    # 设置结果图像的路径
    self.result_img_name = os.path.join(self.result_img_path, self.img_name)
    # 模型识别
    self.results = yolo.predict(img, imgsz=imgsz, conf=conf_thres, device=device, classes=classes)
```

Figure 6. Model prediction image.

Figure 7 further illustrates the implementation details of the frame function, which refines the raw detection results and generates the annotated image output. Specifically, the function iterates through each detection instance to extract critical attributes, such as the defect category name, the prediction confidence score, and the bounding box coordinates. Using these parameters, the function employs the cv2.rectangle method to draw precise bounding boxes around the identified objects. To improve readability, particularly in complex backgrounds, the category label and confidence score are overlaid on a red rectangular background, ensuring that the annotations remain clear under different lighting or pavement conditions. After completing the annotation process, the function outputs the fully processed frame, which can be directly displayed in the user interface, used for real-time video playback, or further analyzed in downstream computational tasks. Taken together, the operations in Figure 7 demonstrate how raw detection results are transformed into informative and user-friendly visual outputs, thereby enhancing the interpretability, accessibility, and overall practicality of the road defect detection system.

```
def draw_info(frame, results):
    # 遍历每一个检测结果
    for i, bbox in enumerate(results):
        cls_name = bbox[0]  # 类别名称 (如: 'pothole', 'crack')
        conf = bbox[1]      # 置信度得分 (如: 0.85)
        box = bbox[2]       # 边界框坐标 [x_min, y_min, x_max, y_max]
        # 根据索引 i 生成颜色，确保不同目标的颜色不同
        color = compute_color_for_labels(i)
        # 绘制边界框
        # cv2.rectangle() 参数:
        # - frame: 要绘制的图像
        # - (int(box[0]), int(box[1])): 左上角坐标 (x_min, y_min)
        # - (int(box[2]), int(box[3])): 右下角坐标 (x_max, y_max)
        # - color: 边框颜色
        # - 2: 边框线条的宽度
        cv2.rectangle(frame, (int(box[0]), int(box[1])), (int(box[2]), int(box[3])), color, 2)
        # 绘制类别名称和置信度，并使用红色背景提高可读性
        # draw_text_with_red_background() 参数:
        # - frame: 输入图像
        # - 文本：类别名称和置信度字符串组合，例如 'pothole 0.85'
        # - (int(box[0]), int(box[1])): 文本位置 (左上角坐标)
        # - font: 字体类型
        # - font_scale: 字体大小比例
        # - thickness: 字体线条粗细
        # - color: 文本颜色 (与边框颜色一致)
        frame = draw_text_with_red_background(frame,
                                              str(cls_name) + ' ' + str(conf),
                                              (int(box[0]), int(box[1])),
                                              font=cv2.FONT_HERSHEY_SIMPLEX,
                                              font_scale=0.5,
                                              thickness=1,
                                              color=color)
    return frame
```

Figure 7. Frame function draw info.

3. Experimental Performance Test

3.1. Image Recognition

As shown in Figure 8, in the image recognition experiment, the system is tasked with detecting road defects from static images. The YOLO model demonstrates strong capability in accurately identifying common defect types, such as cracks, potholes, and surface deterioration, by marking them with bounding boxes and displaying both the corresponding class labels and confidence scores. To evaluate the robustness of the model, a variety of static road images were tested under different environmental conditions, including changes in illumination (e.g., daytime versus nighttime scenarios), pavement textures (e.g., asphalt, cement, and mixed surfaces), and background interference such as shadows, lane markings, or small debris. The results show that the system consistently distinguishes road defects from normal pavement surfaces, with detection accuracy maintained across these challenging conditions. In addition, the graphical interface provides intuitive visual feedback, allowing users to clearly interpret the detection results and identify the location and severity of road defects. These experimental findings demonstrate that the system achieves reliable performance in single-image recognition tasks, confirming its value for practical applications in road condition assessment, maintenance planning, and early warning systems for infrastructure safety.

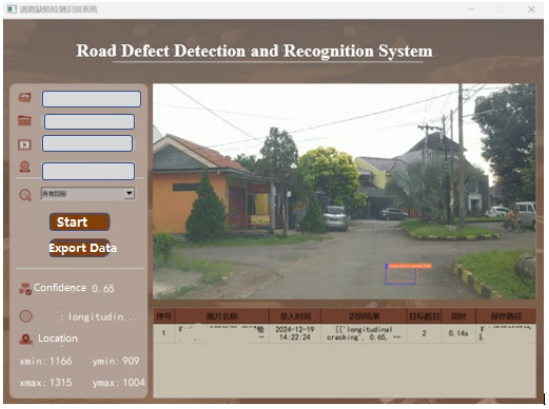

Figure 8. Identifying road defects in images.

3.2. Videosidentification

As shown in Figure 9, in the video recognition experiment, the system extends its functionality to real-time detection on continuous video frames, thereby addressing the dynamic characteristics of real-world road monitoring. The YOLO model captures and recognizes road defects frame by frame, while simultaneously overlaying bounding boxes and class labels on each frame during video playback. Unlike static image detection, video

input introduces challenges such as motion blur, camera jitter, and rapid changes in illumination. To ensure robustness, each frame is processed individually while temporal consistency of the annotations is maintained across consecutive frames. The results demonstrate that the system not only retains the accuracy observed in static images but also achieves stable and continuous monitoring in dynamic video streams. This enables the system to track defects as vehicles move, detect newly appearing defects in real time, and maintain reliable detection performance even under fluctuating environmental conditions.

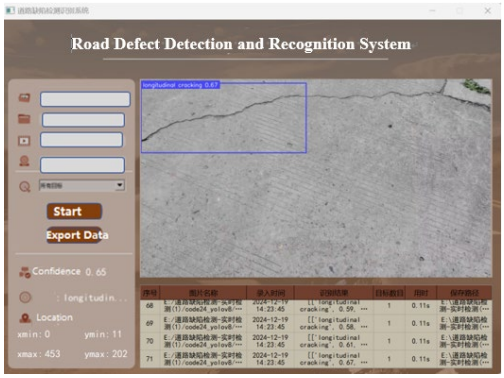

Figure 9. Identifying road defects in a video.

Overall, the experimental results confirm that the proposed road defect detection system, powered by YOLO-based models, performs effectively across both static and dynamic input scenarios. Its ability to provide accurate, real-time, and stable detection highlights its robustness and generalization capability. These strengths suggest wide applicability in urban road management, traffic safety monitoring, and intelligent transportation systems. Moreover, the system's adaptability to video input indicates promising potential for integration with autonomous vehicles, drone-based inspection platforms, and large-scale smart city infrastructure management. By combining image-level precision with real-time video processing, the system represents a step forward toward intelligent, automated, and data-driven road maintenance strategies.

4. Conclusion

The "Road Defect Detection and Identification System," developed using PyQt and YOLOv8, provides an efficient and convenient solution for road maintenance and inspection. The system not only detects and identifies road defects in real time, but also generates detailed reports that offer valuable insights for road repair, maintenance planning, and infrastructure management. With its powerful detection capabilities, stable performance under varying conditions, and flexible interface design, the system demonstrates significant potential for applications in multiple fields, including urban road management, traffic safety, and intelligent transportation systems. In practical terms, the ability to process images, videos, and real-time camera streams makes the system adaptable to different usage scenarios, from research analysis to deployment in road monitoring vehicles.

Looking ahead, the system can be further optimized and extended in several directions. For instance, integrating additional deep learning models and advanced detection algorithms could improve accuracy, while model optimization techniques such as pruning or quantization could enhance efficiency for real-time edge deployment. Moreover, combining the system with complementary technologies, such as drones or autonomous inspection vehicles, may enable more comprehensive monitoring and predictive maintenance. These advancements would contribute to more efficient resource allocation, reduced inspection costs, and improved road safety. Ultimately, by continuously improving

both the technical foundation and the practical integration of the system, it has the potential to play a significant role in supporting smart city development and ensuring the long-term sustainability of transportation infrastructure.

Reference

1. L. Zhang, F. Yang, Y. D. Zhang, Y. J. Zhu, "Road crack detection using deep convolutional neural network," *2016 IEEE Int. Conf. Image Process. (ICIP)*, IEEE, 2016, doi: 10.1109/ICIP.2016.7533052.
2. H. Maeda, et al., " Road damage detection and classification using deep learning," *Mach. Intell. Cybern.*, vol. 4, no. 2, pp. 1–8, 2018, doi: 10.1111/mice.12387.
3. A. Alfarrarjeh, D. Trivedi, S. H. Kim, C. Shahabi, "A deep learning approach for road damage detection from smartphone images," *2018 IEEE Int. Conf. Big Data (Big Data)*, IEEE, 2018, doi: 10.1109/BigData.2018.8621899.
4. K. Gopalakrishnan, et al., "Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection," *Constr. Build. Mater.*, vol. 157, pp. 322-330, 2017, doi: 10.1016/j.conbuildmat.2017.09.110.
5. M. Eisenbach, et al., "How to get pavement distress detection ready for deep learning? A systematic approach," *2017 Int. Joint Conf. Neural Netw. (IJCNN)*, IEEE, 2017, doi: 10.1109/IJCNN.2017.7966101.
6. E. Zalama, J. Gómez-García-Bermejo, R. Medina, and J. Llamas, "Road crack detection using visual features extracted by Gabor filters," *Comput.-Aided Civ. Infrastruct. Eng.*, vol. 29, no. 5, pp. 342–358, 2014, doi: 10.1111/mice.12042.
7. S. Yang, "The Impact of Continuous Integration and Continuous Delivery on Software Development Efficiency", *J. Comput. Signal Syst. Res.*, vol. 2, no. 3, pp. 59–68, Apr. 2025, doi: 10.71222/pzvfqm21.