Transport and Communications Science Journal

CRACK DETECTION ON CONCRETE SURFACES USING THE YOLOv8 QUANTIZATION MODEL

Ngo Thanh Binh[1*], Ngo Van Minh[1], Vu Ngoc Linh[1], Pham Tuan Dung[2]

[1]University of Transport and Communications, No. 3 Cau Giay Street, Hanoi, Vietnam

[2]School of Engineering, University of Aberdeen, King's College, Aberdeen AB24 3UE, United Kingdom

ARTICLE INFO

TYPE: Research Article
Received: 19/03/2025
Revised: 24/04/2025
Accepted: 10/05/2025
Published online: 15/05/2025
https://doi.org/10.47869/tcsj.76.4.4

[*] *Corresponding author*
Email: ngobinh74@utc.edu.vn; Tel: 0947699777

Abstract. The condition of concrete structures' surfaces, most importantly the condition of cracks on the surface and their development over time, is an important and common criterion used to diagnose health conditions and determine their service life. Rapid collection, identification, and monitoring of concrete structures' surfaces to assess the condition of bridge structures requires a fast-acting system that can meet the actual speed of the inspection process. This article introduces a YOLOv8 quantization model for rapid crack detection in concrete structures, leveraging GPU acceleration suitable for real-time video analysis during bridge inspections. The method incorporates histogram equalization and image enhancement to mitigate lighting issues and improve crack visibility. INT8 quantization reduces model size and accelerates processing while maintaining accuracy through dataset calibration. Converted to TensorRT and integrated into the inference pipeline for optimized GPU and memory management, the YOLOv8 quantization model achieves at least 30 FPS using full HD video footage. Field tests with NVIDIA GPUs demonstrated a 5x reduction in processing time, a 5x FPS increase, and a 6x improvement in GPU utilization, all while maintaining similar RAM usage. The quantum YOLOv8 model is optimized for NVIDIA GPUs, achieving a balance between accuracy and processing speed, allowing workers to analyze full HD videos in real-time at a rate of at least 30 FPS during field inspections.

Keywords: Quantization, YOLOv8, Image-processing, Crack detection, Concrete structures.

1. INTRODUCTION

Reinforced concrete (RC) and prestressed concrete (PSC) bridges account for a large proportion of the total number of bridge works in Vietnam due to low construction costs, simple and easy-to-master construction technology, and suitability for many complex terrains in Vietnam. Damages appearing on the surface of concrete structures are cracks, peeling, pitting, etc. Cracks are the most frequent damage and have a great and direct impact on the overall load-bearing capacity of road and bridge structures [1, 2]. Cracks on the surface of traffic works are very dangerous for the structure of the bridge [3]. Every year, bridge management and maintenance agencies must invest a large amount of resources, including time, money, and human resources, in surveying cracks and damage in general on existing bridge structures [4]. The survey results help the management unit to make a preliminary assessment of the extent of damage and predict the development of damage and the cause of its occurrence. Therefore, to minimize the cost and time for surveying and evaluating cracks on the surface of concrete structures, one of the current optimal solutions is to use modern techniques to conduct the survey and identify cracks automatically.

Rapid advances in computer vision and Artificial Intelligence (AI), especially machine learning and deep learning, have enabled automated surveys of box girder bridges or concrete surfaces using devices such as digital cameras and AI-enabled smartphones. Along with the development of structural assessment tools based on damage databases, especially for cracks, a key challenge remains that current crack detection models have difficulty processing high-resolution images and videos in real-time. This paper proposes the use of a quantum YOLOv8 model optimized for NVIDIA GPUs to achieve a balance between accuracy and processing speed, allowing workers to analyze full HD videos in real-time at a rate of at least 30 FPS during field inspections

2. RELATED WORKS

Machine learning, deep learning, and digital image processing are widely used for concrete crack recognition and detection. A key challenge in applying digital image processing is the presence of noise, which obscures cracked pixels. Early approaches addressed this via digital image processing techniques [1–9]. For example, Zou *et al*. [1] used the CrackTree method to reduce image blur and enhance crack visibility. Nishikawa *et al*. [5] used multiple sequential image filtering. Salman *et al*. [6] employed Gabor filters to mitigate the impact of uneven surface texture. Gabor filters are effective for crack detection on rough surfaces due to their ability to analyze texture and identify features with consistent frequency and orientation. Fujita and Hamatomo [7] developed a crack model that incorporates noise removal by applying a median filter for preprocessing, followed by a Hessian matrix filter to highlight cracks. Adaptive thresholding was then used to improve crack detection accuracy.

Although digital image processing techniques are easy to apply at low cost, they still have many disadvantages and often fail to achieve the desired accuracy [8]. Machine learning and deep learning are used as the basis for automatic crack pattern recognition with high accuracy. Some computational algorithms, such as Support Vector Machine (SVM), Decision Tree, Random Forest, and K-means Clustering, are used to detect and classify [9,10] and determine the location of cracks in the image [11,12]. Then, image processing techniques are applied to optimize the ability to recognize cracks. Although the accuracy has been improved, they still depend on the parameters chosen initially. Artificial Neural Networks (ANN) update important automatic ways to solve the existence of computational algorithms, from which many structured

ANNs are used for crack detection. Convolutional Neural Network (CNN) has emerged and is widely used in crack recognition tools. Ni *et al*. have automated the task of crack recognition (detection) and segmentation by merging features (feature maps) and classifying pixels [13]. From there, a CNN architecture named GoogleNet CNN is used for crack recognition. To segment cracks, the authors use the FPN (Feature Pyramid Network) structure, which contains merging layers and successive tick layers for crack segmentation.

The CNN-based CrackNet model was introduced by Zhang *et al*. [14] to detect cracks on the surface of the pavement. Unlike the traditional CNN model, CrackNet eliminated the pooling layers, which helps the model achieve pixel-level accuracy while the length and width of the image remain unchanged through all layers. Yang *et al*. [15] also used a variant of CNN, named a fully convolutional network (FCN), to segment cracks at the pixel level. Compared with CrackNet, this method outperforms pixel-level crack segmentation and reduces training time, but the performance is lower in accuracy. Cui *et al*. [16] optimized the YOLOv3 model to detect damage on the surface of concrete structures of bridges due to the abrasion effect of wind loads. Long *et al*. [17] developed a "CNN model for crack detection" based on a modified Viraja model [18] by adding additional modules to the original model. The first module improves the data processing speed by replacing the C3 module in the original model with a more compact neural network. The second module is replaced with his simpler convolutional structure, while the third module is used to improve the accuracy of damage detection by eliminating upsampling. Similarly, Ye *et al*. [19] modified the YOLOv7 model by combining three self-developed modules, named YOLOv7-AMF, to detect cracks on the surface of concrete structures under noisy image conditions. These models handle the crack detection requirement, but the processing speed needs to be improved to meet the real-time sampling of the video.

The YOLOv8 model has many outstanding advantages, such as real-time object recognition, high processing performance with only one neural network, the ability to accurately recognize objects in multi-object cases, and flexibility. YOLOv8 is compatible and supported on both CPU and GPU, taking advantage of advanced technologies such as NVIDIA's Tensor RT and Intel's OpenVino. In this study, the YOLOv8 model was developed by the research team to train crack recognition, and run quantization INT8 correction to provide optimal parameters of the model, helping the system to maintain the accuracy closest to the original model while increasing the processing speed of the model to meet the real-time response of the full HD camera as well as processing 30 FPS video recorded at the actual moving speed of the field inspection equipment.

3. PROPOSED SOLUTION

3.1. Data collection and sample synthesis

Crack detection models often utilize open datasets containing images both with and without cracks. This study employs a dataset compiled from Kaggle, SDNET2018, and real surface crack images collected and augmented by the research team. To optimize computational efficiency and training, images were preprocessed and labelled with a single "crack" class using YOLOv8 format in Roboflow. The final training dataset comprises 4672 labelled concrete crack images. Data augmentation techniques, such as random cropping, resizing, rotation, and noise injection, were applied to enhance the detection of small cracks by introducing variations in scale, viewpoint, and occlusion, thereby improving the model's robustness and generalization capabilities [20, 21].

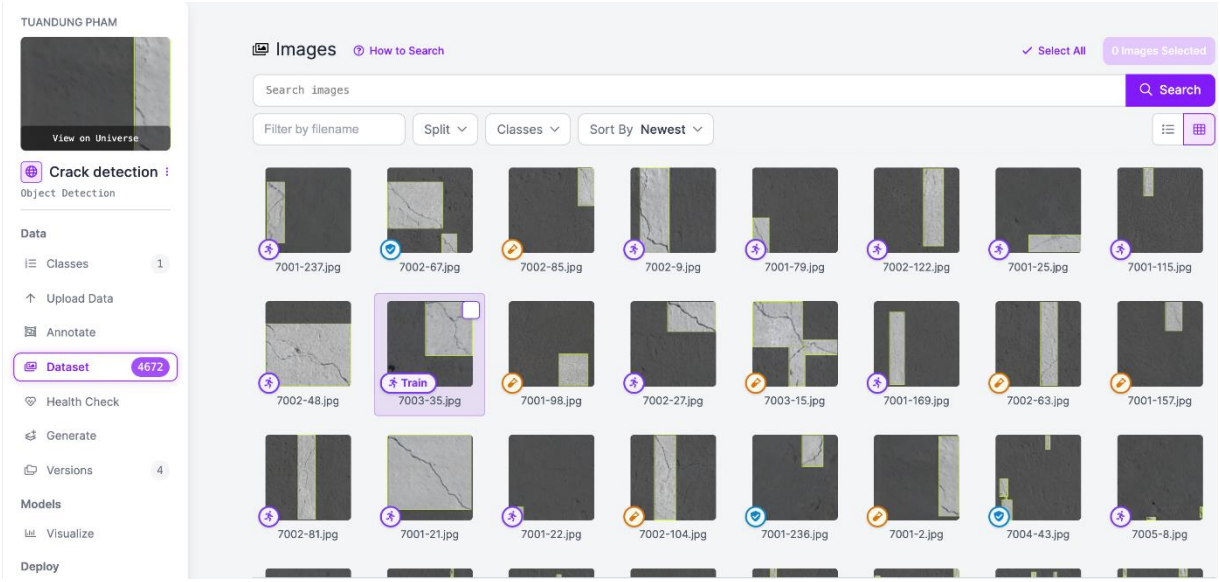


Figure 1. Crack image data labelled on the research team's Roboflow platform.

In this study, two main techniques are applied: histogram equalization and thresholding, in which the histogram equalization technique aims to enhance the image sample applied for sample addition, while thresholding mainly serves the observation, for experts to see the crack separately when evaluating. The histogram technique is a graph of the grey level distribution on the image, in the range [0, L-1], and the histogram is a function of the form (1):

$$P\left(r_k\right) = n_k / n \tag{1}$$

Where, r_k is the grey level value of the k^{th} pixel, n is the total number of pixels, and n_k is the total number of pixels with grey level r_k. The $P(r_k)$ value estimates the grey level distribution in the image. A histogram is used to represent image information about the ability to enhance the contrast of the image. If the histogram is in the form of a narrow band, concentrated on one side of the left (or right), the corresponding image is dark (or bright). If the histogram is concentrated on the middle, the image has a low contrast level, and if the histogram is evenly distributed, the image will have high contrast. Therefore, the histogram equalization technique is applied here to redistribute the grey levels in the image, helping the histogram to be distributed in the range [0, L-1] so that the image has high contrast (Figure 2). Histogram equalization is performed by using a transformation of the form $s = T\left(r\right)$, distributing the probability density of the r pixels to create an image with grey levels with uniform density.

The graph in Figure 2 illustrates the changes in the images before and after histogram equalization by the research team. The original image (top left) has uneven brightness, some areas may be too dark or too bright. The image after histogram equalization (bottom left) becomes clearer, and the contrast increases, allowing better observation of details. The histogram on the right clearly shows the improvement in the enhanced image sample. In the original image histogram (top right), the pixel values are concentrated in certain intensity ranges, indicating that the brightness and contrast are limited, and the image may be too dark or too bright in some areas. With the post-equalization image histogram (bottom right), the pixel values are more evenly spread over the entire intensity range (0-255), indicating that the image has better contrast, and the image becomes clearer with improved details.

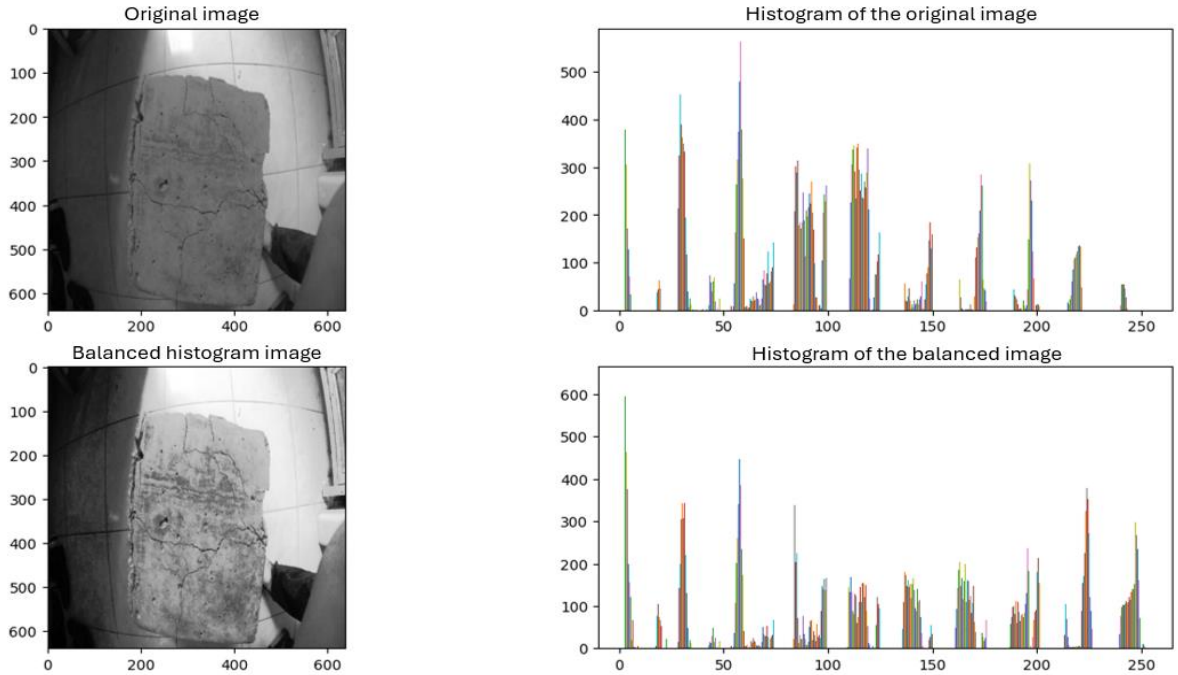


Figure 2. Applying histogram equalization of concrete crack images on real samples of the research team to enhance the image sample and support observation.

The histogram equalization enhances image contrast by redistributing light intensity, making the image easier to observe, especially in uneven lighting conditions, which is especially necessary in the working environment of surveying cracks in concrete bridge girder boxes.

3.2. Quantization YOLOv8

The structure of YOLOv8 includes the main parts: Backbone and Head. In the YOLOv8 model types, the YOLOv8-n, YOLOv8-s, YOLOv8-m, YOLOv8-l, and YOLOv8-x models have different model sizes, which are considered for research and deployment on computers with suitable configurations. In this study, the research team used the Ultralytics YOLOv8 version [22]. The model size will correspond to the mean average precision value (mAP) and be inversely proportional to the calculation and inference time of the model (Figure 3). This means that models with larger sizes have larger mAP values, but the inference time is also slower, and vice versa. Figure 4 shows the performance description of YOLOv8 models by size and compares the YOLOv8 model with other models. In this study, the research team chose the YOLOv8-l model [23] to optimize training time and computer resources, then applied quantization to speed up the system since this model has the highest accuracy. When optimizing the model, the achieved results will be highlighted with the best detection ability.

For high-performance environments, particularly on NVIDIA GPUs, exporting Ultralytics YOLOv8 models to TensorRT optimizes them for fast and efficient inference. NVIDIA's TensorRT SDK accelerates deep learning inference through optimizations like layer merging, precision calibration (INT8), dynamic tensor memory management, and kernel auto-tuning. Converting models to TensorRT unlocks the full potential of NVIDIA GPUs. TensorRT supports various formats, including TensorFlow, PyTorch, and ONNX, providing flexibility for integrating and deploying models across diverse environments.

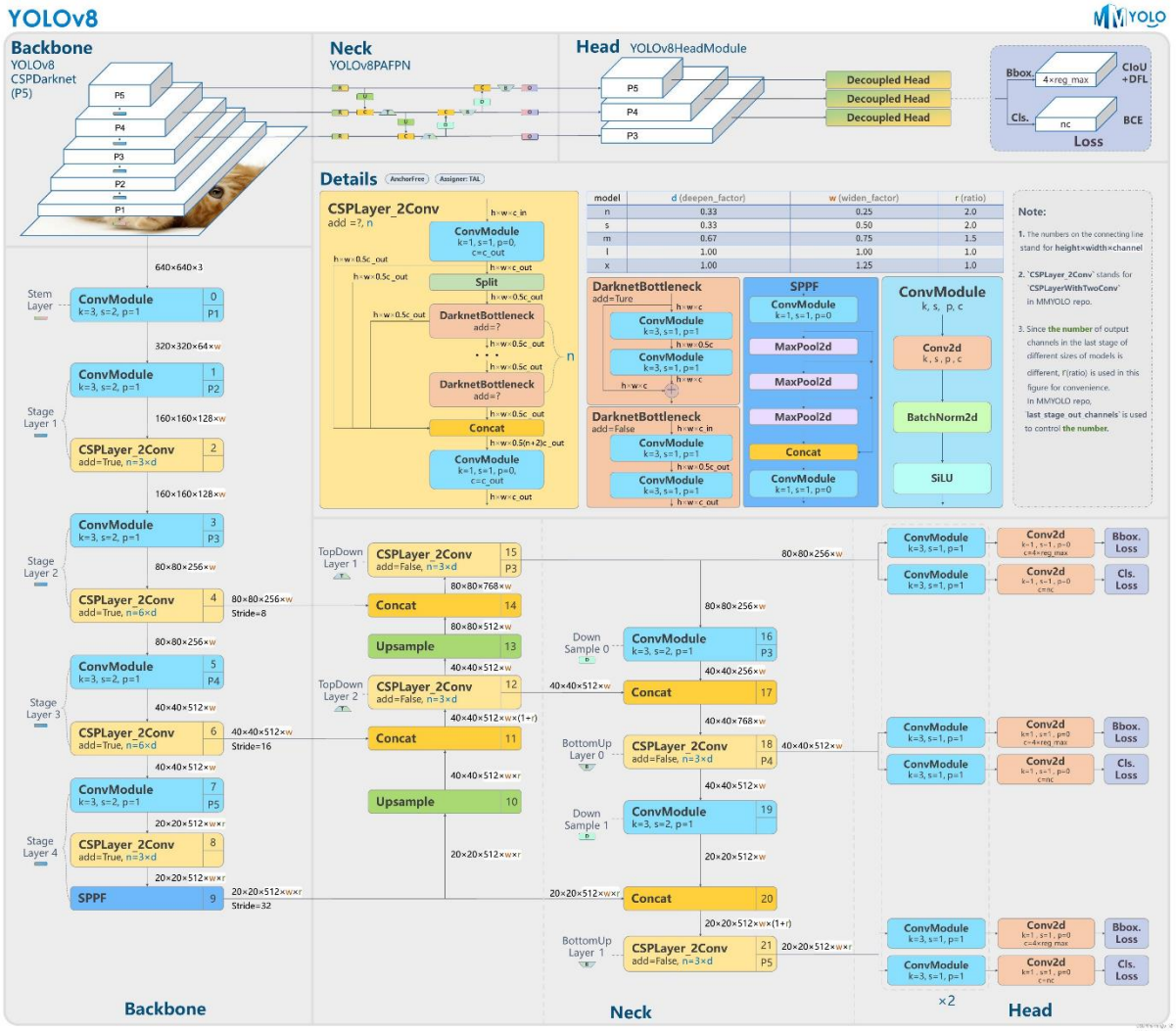


Figure 3. Model Architecture of YOLOv8 [23].

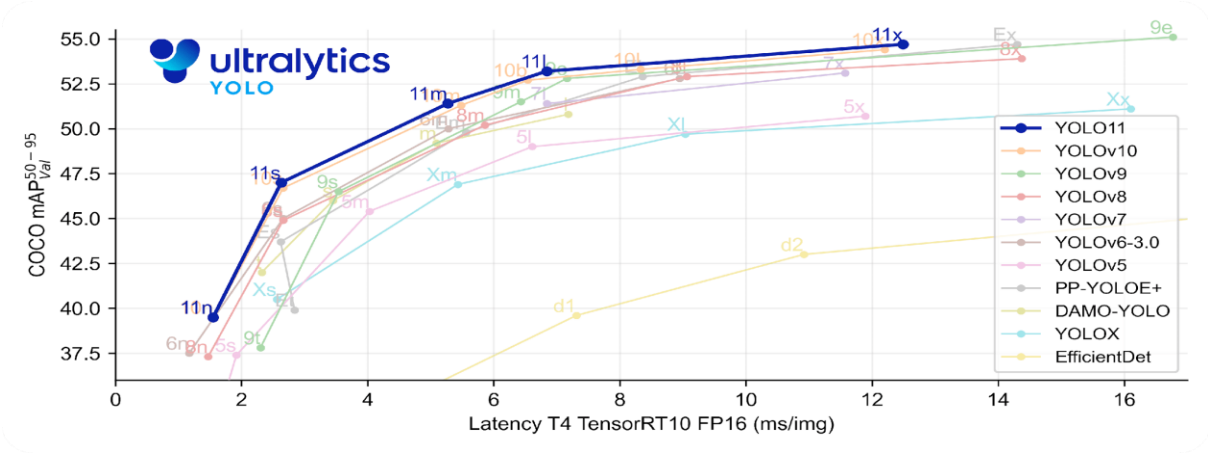


Figure 4. Comparison of accuracy and performance of some YOLO family models [22].

The research team uses quantization to optimize the performance of the model on NVIDIA graphics card-supported hardware devices, which requires making good use of the GPU and limiting the memory used. In addition to changing the structure to suit the environment and hardware deployed, an important factor in optimizing the model is the use of data types to help the model calculate. In the AI model, there are some popular formats such as FP32, FP16, and INT8 used as data standards for the AI model [17]. Model optimization not only brings the floating point representation format but also brings it to the integer format. Model quantization applies the commonly used format in the AI field, INT8, but not all AI models can run optimally with INT8. The quantization used in this paper by the research team is INT8. This format is being widely applied in Deep Learning workloads instead of FP32. Since lower precision weights in neural networks do not seem to matter and do not affect the performance of the model, the floating-point precision of FP format in general, and FP32, in particular, can be traded for speed.

Step 1: Convert PyTorch to ONNX format

ONNX (Open Neural Network Exchange) is an intermediate format that makes PyTorch models compatible with TensorRT. PyTorch models are usually in *.pth or *.pt format. To export a PyTorch model. We first need to convert it to an ONNX model. This process is done using "Algorithm 1: Convert PyTorch model to ONNX model", as follows:

Algorithm 1: Convert PyTorch model to ONNX model

Function: Convert model

INPUT: PyTorch model

OUTPUT: ONNX model

1: Name the output
2: If: Set dynamic model input
3: Then: Set input size according to batch index, width, height, and set personally
4: If: offset version left blank
5: Then: set default offset
6: Export model
7: Export model parameters
8: Write the model file.
9: If: enable simplification of the ONNX model
10: Then: remove redundant parts of the model
11: Save the model to the hard drive.

Important information to note is that the version of ONNX is ported so that it can be deployed in other environments if that environment requires a specific version. Next are the input name (input_name) and output name (output_name), these are 2 important elements to put the image into the model via input and get the detection result at the output.

Step 2: TensorRT model with quantization

TensorRT uses ONNX as input and applies optimizations to create a TensorRT model, done using "Algorithm 2: Convert ONNX model to TensorRT model with quantization", as follows:

Algorithm 2: Convert ONNX model to TensorRT model with quantization

Function: Convert model

INPUT: ONNX model

OUTPUT: TensorRT model

1: Set workspace size
2: If: version greater than 10
3: Then: set memory limit according to workspace
4: Otherwise: set memory according to the maximum workspace value
5: Set input and output
6: If: Set dynamic input
7: Then: set input, output according to auto-tuning parameter
8: If: optimize int8
9: Then: calibrate to get the calibration value for the configuration according to the sample set.
10: Free CUDA memory
11: Export model in *.engine format according to optimized configuration

After performing the conversion, the model with the extension *.engine will be obtained, and the crack detection system will be deployed with this model. The performance on a batch size of INT8 is superior to FP32 if the Batch size is more, but if the value is 2 batch Size, the two are equivalent in performance, with a batch size equal to 4, the performance of INT8 is double that of FP32, and with a value of 128, the performance is increased by 3 times. In terms of efficiency, the memory of INT8 is less than FP32, which is natural because FP32 uses 32 bits to represent, while INT8 only uses 8 bits, so the memory usage is also less. With a small batch size value, the memory difference is not large, but at a large scale like 128, FP32 takes up about 2.5 times more than INT8. In terms of accuracy, the two formats are equivalent in top1 (about 70%) and top5 (about 90%). Although floating-point bits are not used, NVIDIA's test results in similar accuracy. The process of converting the model to TensorRT format and quantizing the model was performed by the research team in 2 steps: Converting the PyTorch model to ONNX format and converting ONNX to TensorRT.

4. EXPERIMENTAL RESULT, MODEL EVALUATION, AND DISCUSSION

The YOLOv8 quantization model was customized, trained with 200 epochs, a batch size of 32, and tasked with detecting only one class (class) of cracks. The test results of the model are illustrated in Figure 5. It can be seen that the model detects most of the cracks appearing on the surface of real concrete structures. To evaluate this model, the research team clarified the main parameters to measure the model's suitability to the requirements of the crack detection problem. The evaluation parameters used are Accuracy, Precision, Recall, and mAP.

Our model was evaluated on a PC with a Ryzen 75000 series CPU, NVIDIA GeForce RTX 3060 6GB GPU, and 23GB of RAM. The evaluation environment comprised Python 3.9, the PyCharm IDE, PyTorch 1.13, and cuDNN 11.7. PyTorch, a deep learning framework, leverages CUDA for GPU-accelerated computation, enabling faster model processing.

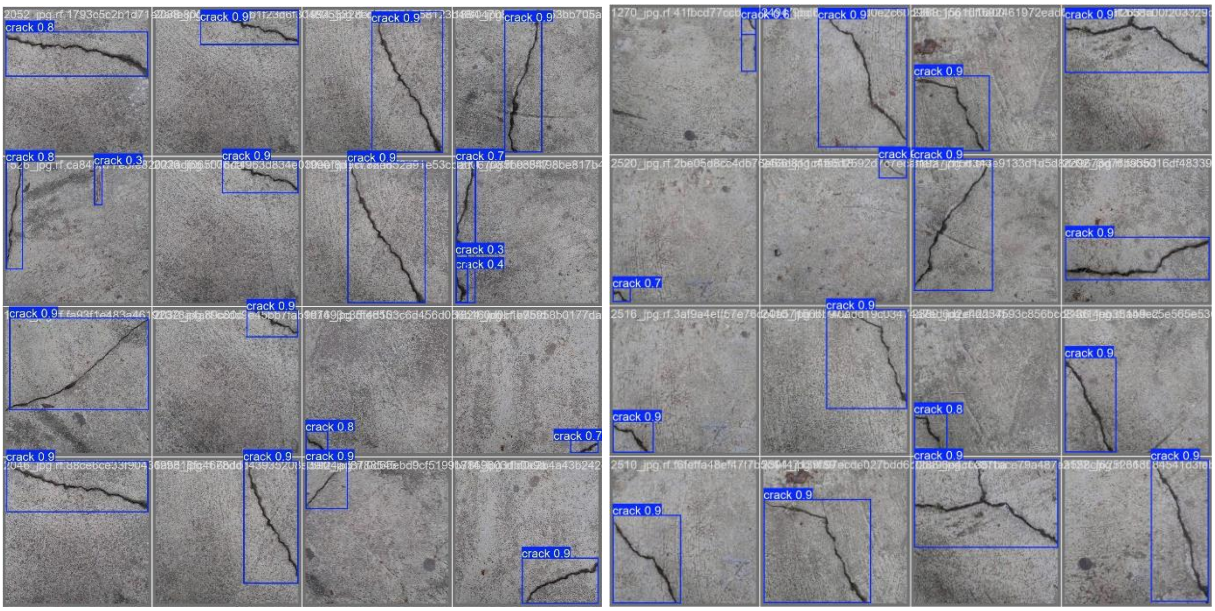


Figure 5. Results of testing surface crack image samples using the YOLOv8 quantization model of the research team.

Figure 6 demonstrates the model's effective learning, as evidenced by the steadily decreasing loss functions and gradually increasing evaluation metrics (Precision, Recall, mAP). Minimal overfitting is observed, indicated by the comparable loss values between the training and test sets. The program automatically compares the newly trained model's parameters with the previous best, updating the best model if improved performance is achieved. This automated process ensures the best-performing model is used for quantization. The graphs in Figure 6 show the loss values during training (top half) and testing (bottom half), as shown below:

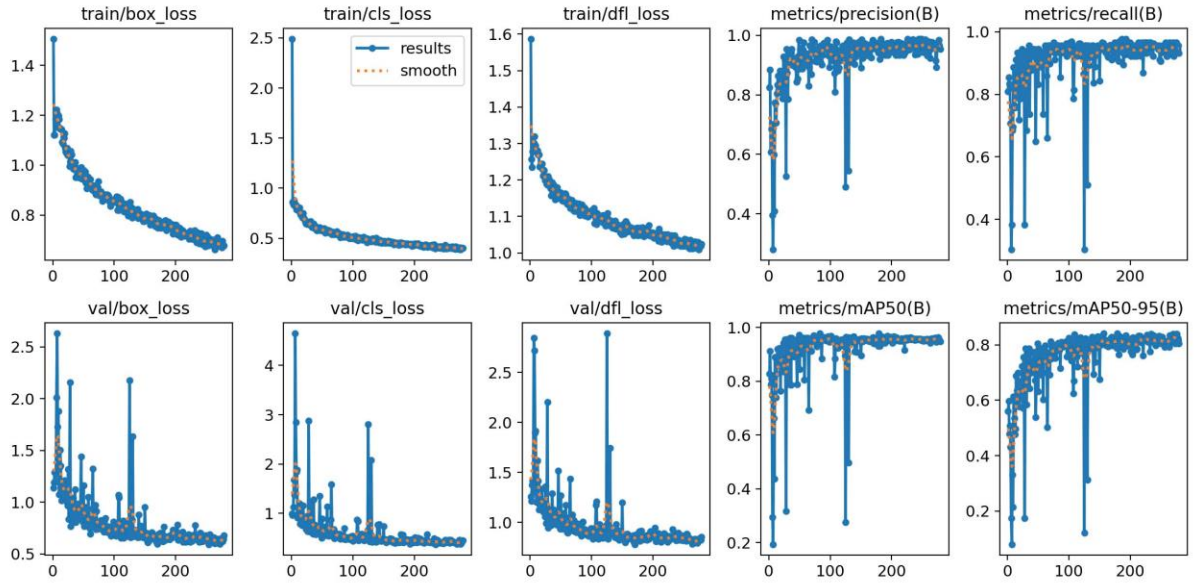
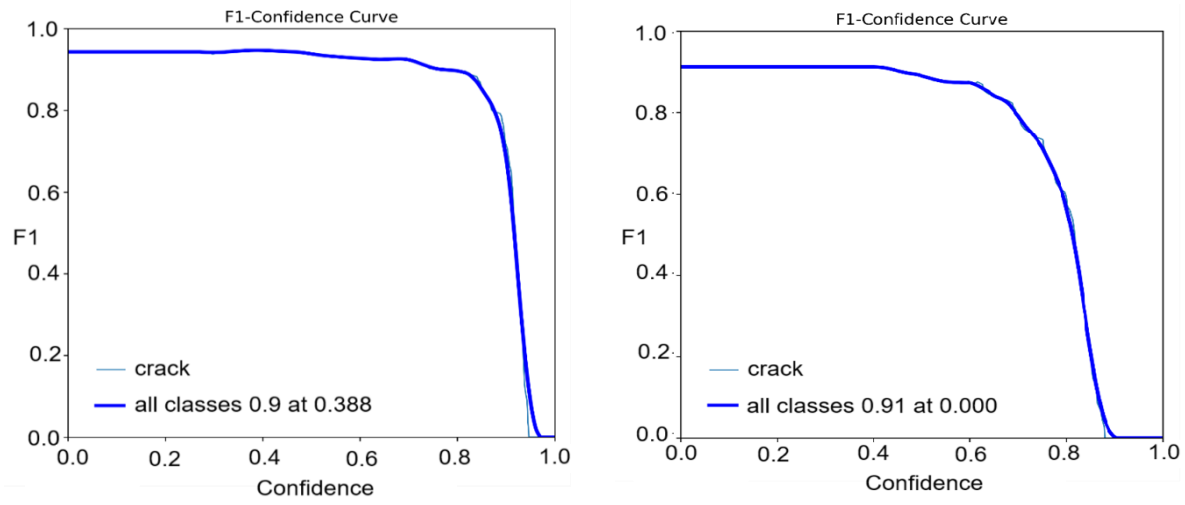


Figure 6. Model training graph after 280 epochs.

The graphs in Figure 6 show training (top) and testing (bottom) loss and metric values. During training, the train/box_loss decreases over epochs, from ~2.5 to ~0.386, indicating

improved object localization. The train/cls_loss, related to object label prediction, drops sharply initially and then stabilizes, suggesting better object recognition. The train/dfl_loss (Distribution Focal Loss), a YOLO component enhancing detection, also gradually decreases, reflecting better distance prediction. The metrics/precision(B) plot shows accuracy increasing to approximately ~0.98, meaning 98% of predictions are correct, with significant improvement early on, followed by slight fluctuations. The metrics/recall(B) plot, representing sensitivity, increases from approximately ~0.3 to nearly ~0.97, indicating the model misses fewer objects. On the test set, the val/box_loss also decreases, though it's slightly higher than during training, suggesting room for improvement. The val/cls_loss decreases significantly with some fluctuations. The val/dfl_loss mirrors the train/dfl_loss on the test set, decreasing gradually and demonstrating improved distance prediction. The metrics/mAP50(B) increases from 0.19 to approximately ~0.95, indicating improved accuracy at IoU ≥ 50%. The more stringent metrics/mAP50-95(B), calculated across IoUs from 50 to 95%, also increase gradually from approximately ~0.07 to nearly ~0.84.

Figure 7 plots the F1-score against the confidence threshold (ranging from 0 to 1). This threshold dictates the model's required confidence level for accepting predictions; higher thresholds lead to fewer accepted predictions. The F1-score, representing a balance between precision and recall, peaks when both are high and balanced. The F1 curve illustrates the relationship between F1-score and confidence threshold. Typically, F1 increases with the threshold initially, peaks, and then declines. The curve generally maintains a high F1 at low confidence levels but drops sharply above a threshold, such as 0.8. In Figure 7a, F1 reaches 0.95 at a confidence threshold of 0.388, maintaining a high precision level before a sharper decline, and exhibiting minor fluctuations with good stability. In contrast, Figure 7b achieves an F1 of 0.91 at a 0.000 threshold, but its F1 degrades sooner and sharply despite appearing smoother. The original model, exhibiting a higher peak F1, may indicate superior classification. Tensor models tend to degrade faster due to information loss from quantization. While the original model excels at maintaining F1 across a broad confidence spectrum, a tensor model might be preferable for optimization within a specific range.

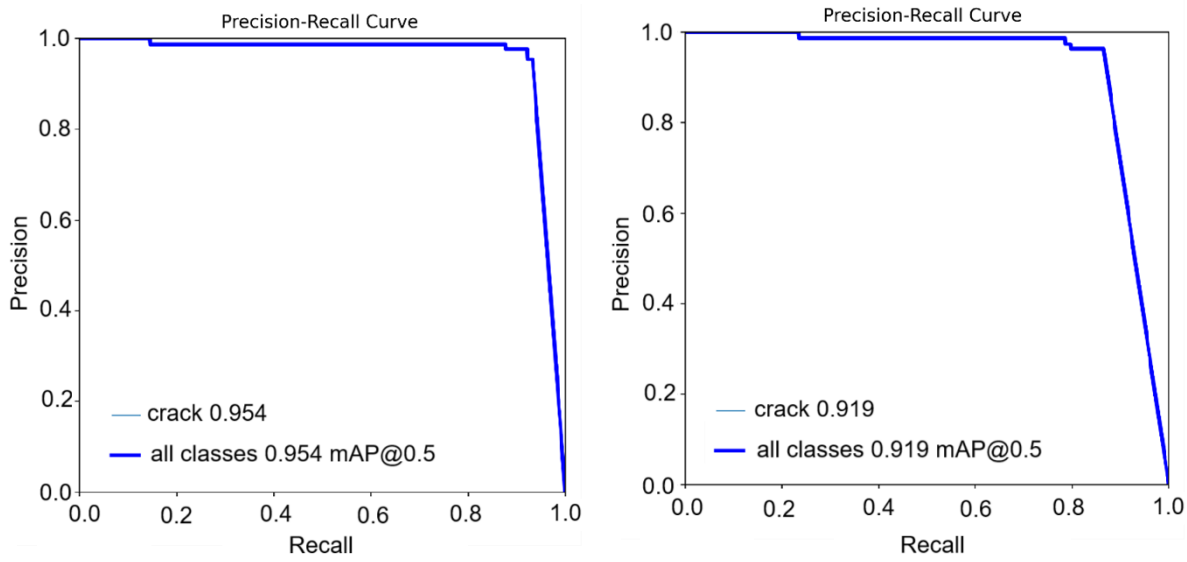


(a) Original mode

(b) Tensor mode

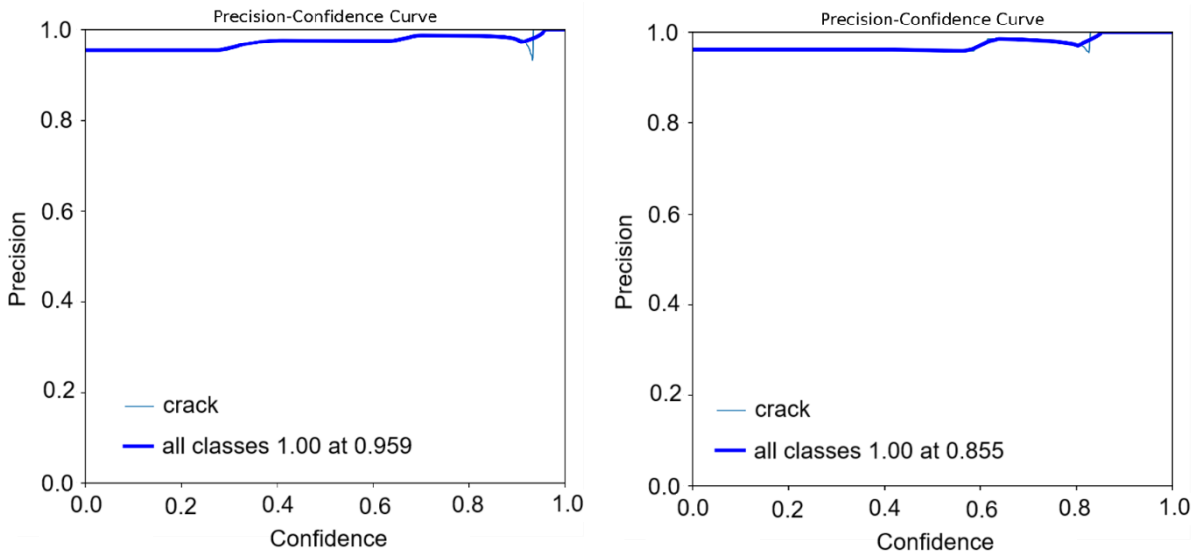
Figure 7. F1-Confidence Curve.

Figure 8's Precision-Recall curve illustrates the trade-off between precision and recall for the binary classification model across varying expectation thresholds.



(a) Original mode (b) Tensor mode

Figure 8. Precision-Recall Curve.



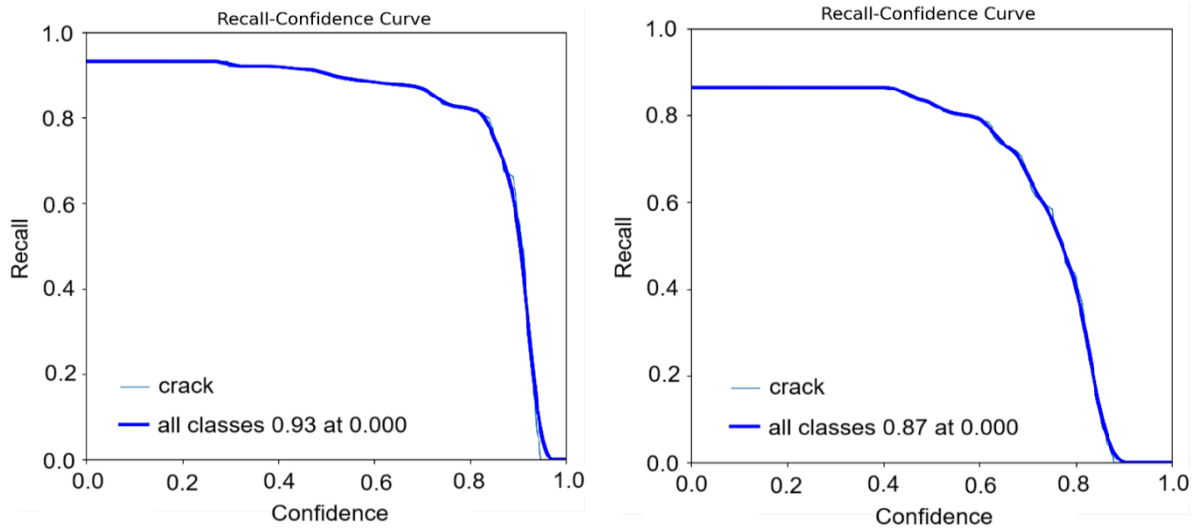
(a) Original mode (b) Tensor mode

Figure 9. Precision-Confidence Curve.

The Precision-Recall (PR) curve illustrates the trade-off between precision and recall at varying thresholds, particularly crucial for imbalanced datasets. Recall, plotted on the horizontal axis, represents the model's ability to detect actual objects (higher recall signifies fewer missed objects). Precision, on the vertical axis, indicates the accuracy of positive predictions (higher precision means fewer false positives). Figure 8 demonstrates that graph (a) has superior performance compared to graph (b), exhibiting both higher precision (0.954 vs. 0.919), reflecting a better correct prediction rate, and a higher mAP@0.5 (0.954 vs. 0.919), indicating better overall performance.

Figure 9 shows the Precision-Confidence Curve. The horizontal axis (Confidence) is the model's confidence threshold, ranging from 0 to 1. This threshold determines the level of certainty that the model needs to have in predicting an object. As the confidence threshold increases, the model will only accept predictions with higher confidence, eliminating less confident predictions. The vertical axis (Precision) shows the model's accuracy at each confidence level. Precision is the ratio of the number of correct predictions (True Positives) to the total number of predictions that the model considers correct (including True Positives and False Positives). A high Precision value indicates that the model rarely makes incorrect predictions.

The Precision curve illustrates the relationship between Precision and the confidence threshold. Lower confidence thresholds often lead to decreased Precision due to more incorrect predictions. As confidence increases, Precision improves as the model accepts fewer, more reliable predictions. Figure 9 shows that both models achieve perfect Precision (1.00) at specific confidence thresholds. However, model (a) reaches this level at a higher confidence threshold (0.959) than model (b) (0.855), suggesting model (a) requires greater confidence to achieve perfect Precision.



(a) Original mode

(b) Tensor mode

Figure 10. Recall-Confidence Curve.

Figure 10 shows the Recall-Confidence Curve. The horizontal axis represents the confidence threshold; as this threshold increases, the model only accepts predictions with higher confidence, discarding those with lower confidence. The vertical axis represents Recall, indicating the proportion of actual objects detected. Higher Recall signifies that the model detects a larger proportion of objects, thus missing fewer. In addition, Figure 10 demonstrates an inverse relationship between confidence threshold and Recall. Lower thresholds (0.0-0.2) result in higher Recall (0.91 for all classes), while higher thresholds decrease Recall by retaining only high-confidence predictions. At a 0.000 threshold, graph (a) shows a higher Recall (0.93) than graph (b) (0.87), suggesting graph (a)'s model is more effective at identifying positive cases even with low confidence. Evaluating both graphs at a 0.000 threshold provides a performance assessment without confidence filtering.

Table 1. Comparison table of the running speed of the YOLOv8 quantization model developed by the research team.

	Original Model	Model TensorRT
Avg Time (s)	0.09677740030510482	0.018645330917003544
FPS	10.33299093432304	53.63273006262675
GPU Usage (%)	10.026041666666668	63.411458333333336
RAM Usage (%)	59.6	60.0

The model after quantization runs faster, but the reliability of object prediction will decrease. FPS to process a frame with the TensorRT model is 5 times faster on the same machine configuration. The amount of RAM used is not very different but the operation on the GPU is higher because the TensorRT model is implemented on NVIDIA GPUs. These parameters meet the processing of crack image recognition from videos using a 30 FPS full HD camera filmed by workers at the actual moving speed during field inspection.

5. CONCLUSION

The study has outlined the basic methods of applying artificial intelligence, machine learning models, deep learning, and digital image processing techniques in the field of computer vision to detect cracks on the surface of concrete structures. From there, the research team proposed to use the YOLOv8 quantization model, a new model with many outstanding advantages compared to the previous version, taking advantage of GPU acceleration to optimize processing speed and accuracy to detect objects quickly and accurately. Digital image processing techniques such as histogram equalization are also used to process images to enhance the sample, making it easier to detect cracks, as well as supporting expert observers.

This study successfully quantized YOLOv8, leveraging GPU capabilities to optimize processing speed and accuracy. Using well-trained models, INT8 quantization reduced model size and increased speed, while maintaining accuracy close to the original FP32 model through dataset calibration. Converting the calibration dataset to TensorRT enabled efficient GPU memory management and integration into the inference pipeline. Real-world deployment on an NVIDIA GPU-based PC demonstrated a 5x reduction in processing time, a 5x increase in FPS, and a 6x increase in GPU utilization with comparable RAM usage. Optimized for NVIDIA GPUs, the YOLOv8 quantization model delivers real-time full HD video analysis at 30+ FPS for field inspections, balancing accuracy and speed.

ACKNOWLEDGEMENT

This research is funded by the Ministry of Education and Training (MoET), Vietnam under the grant number B2024-GHA-12.

REFERENCES

[1]. Q. Zou, Y. Cao, Q. Li, Q. Mao, S. Wang, CrackTree: Automatic crack detection from pavement images, Pattern Recognition Letters, 33 (2012) 227-238. https://doi.org/10.1016/j.patrec.2011.11.004

[2]. M. Gavilán, D. Balcones, O. Marcos, D.F. Llorca, M.A. Sotelo, I. Parra, M. Ocaña, P. Aliseda, P. Yarza, A. k;mAmírola, Adaptive Road Crack Detection System by Pavement Classification, Sensors, 11 (2011) 9628-9657. https://doi.org/10.3390/s111009628

[3]. Ngo Van Minh, Influence of shear stress to the formation of inclined cracks in webs of post-tensioned concrete box girder bridge constructed by the free cantilever method, Transport and Communications Science Journal, 70 (2019) 21-31. https://doi.org/10.25073/tcsj.70.1.40

[4]. Chul Min Yeum, Shirley J. Dyke, Vision-based automated crack detection for bridge inspection, Computer-Aided Civil and Infrastructure Engineering, 30 (2015) 759-770. https://doi.org/10.1111/mice.12141

[5]. T. Nishikawa, J. Yoshida, T. Sugiyama, Y. Fujino, Concrete crack detection by multiple sequential image filtering, Computer-Aided Civil and Infrastructure Engineering, 27 (2012) 29-47. https://doi.org/10.1111/j.1467-8667.2011.00716.x

[6]. M. Salman, S. Mathavan, K. Kamal, M. Rahman, Pavement crack detection using the Gabor filter, 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), The Hague, Netherlands, (2013) 2039-2044. https://doi.org/10.1109/ITSC.2013.6728529

[7]. Y. Fujita, Y. Hamamoto, A robust automatic crack detection method from noisy concrete surfaces, Mach Vis Appl, 22 (2011) 245–254. https://doi.org/10.1007/s00138-009-0244-5

[8]. R. G. Lins, S. N. Givigi, Automatic Crack Detection and Measurement Based on Image Analysis, IEEE Transactions on Instrumentation and Measurement, 65 (2016) 583-590. https://doi.org/10.1109/TIM.2015.2509278

[9]. L. Ying, E. Salari, Beamlet transform based technique for pavement image processing and classification, 2009 IEEE International Conference on Electro/Information Technology, Windsor, ON, Canada, (2009) 141-145. https://doi.org/10.1109/EIT.2009.5189598

[10]. A. Cubero-Fernandez, Fco. J. Rodriguez-Lozano, Rafael Villatoro, Joaquin Olivares, Jose M. Palomares, Efficient pavement crack detection and classification, J Image Video Proc., 39 (2017). https://doi.org/10.1186/s13640-017-0187-0

[11]. H. Oliveira, P. L. Correia, Automatic Road Crack Detection and Characterization, IEEE Transactions on Intelligent Transportation Systems, 14 (2013) 155-168. https://doi.org/10.1109/TITS.2012.2208630

[12]. Y. Shi, L. Cui, Z. Qi, F. Meng, Z. Chen, Automatic Road Crack Detection Using Random Structured Forests, IEEE Transactions on Intelligent Transportation Systems, 17 (2016) 3434-3445. https://doi.org/10.1109/TITS.2016.2552248

[13]. Fu Tao Ni, Jian Zhang, ZhiQiang Chen, Pixel-level crack delineation in images with convolute ional feature fusion, Structural Control Health Monitering, (2019) 26:e2286. https://doi.org/10.1002/stc.2286

[14]. L. Zhang, F. Yang, Y. Daniel Zhang, Y. J. Zhu, Road crack detection using deep convolutional neural network, 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, (2016) 3708-3712. https://doi.org/10.1109/ICIP.2016.7533052

[15]. Xincong Yang, Heng Li, Yantao Yu, Xiaochun Luo, Ting Huang, Xu Yang, Automatic pixel-level crack detection and measurement using fully convolutional network. Computer-

Aided Civil and Infrastructure Engineering, 33 (2018) 1090-1109. https://doi.org/10.1111/mice.12412

[16]. Xiaoning Cui, Qicai Wang, Jinpeng Dai, Rongling Zhang, Sheng Li, Intelligent recognition of erosion damage to concrete based on improved YOLO-v3, Materials Letters, 302 (2021) 130363. https://doi.org/10.1016/j.matlet.2021.130363

[17]. Long Ngo, Chieu Luong Xuan, Hoang Minh Luong, Binh Ngo Thanh, Dung Bui Ngoc, Designing image processing tools for testing concrete bridges by a drone based on deep learning, Journal of Information and Telecommunication, 7 (2023) 227–240. https://doi.org/10.1080/24751839.2023.2186624

[18]. Viraja1 crack detection: https://github.com/viraja1/crack_detection, Accessed March 28, 2025.

[19]. Guanting Ye, Jinsheng Qu, Jintai Tao, Wei Dai, Yifei Mao, Qiang Jin, Autonomous surface crack identification of concrete structures based on the YOLOv7 algorithm, Journal of Building Engineering, 73 (2023) 106688. https://doi.org/10.1016/j.jobe.2023.106688

[20]. Yanjun Li (2024), Mathematical Modeling Methods and Their Application in the Analysis of Complex Signal System, Advances Mathematical Physics, Image Processing based on Partial Differential Equations, Special Issues (2022). https://doi.org/10.1155/2022/1816814

[21]. P. Kaur, B. S. Khehra, E. B. S. Mavi, Data Augmentation for Object Detection: A Review, 2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), Lansing, MI, USA, (2021) 537-543. https://doi.org/10.1109/MWSCAS47672.2021.9531849

[22]. Ultralytics, "Ultralytics YOLOv8," GitHub repository, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics, accessed March 28, 2025.

[23]. YOLOv8 Architecture: https://wandb.ai/mukilan/wildlife-yolov8/reports/Object-detection-and-tracking-with-YOLOv8--Vmlldzo0MDU5NDA2, accessed March 28, 2025.