remote sensing

MDPI

Article

A Dual-Stage Deep Learning Framework for Fine-Crack Segmentation in High-Resolution UAV Images for Bridge Inspection

Anbin Yu [1], Wensheng Mei [1] and Dapeng Zhu [2,*]

[1] School of Geodesy and Geomatics, Wuhan University, No. 129, Luoyu Road, Wuhan 430079, China; abyu@sgg.whu.edu.cn (A.Y.); wshmei@sgg.whu.edu.cn (W.M.)
[2] GNSS Research Center, Wuhan University, No. 129, Luoyu Road, Wuhan 430079, China
* Correspondence: dpzhu@whu.edu.cn

Highlights

What are the main findings?

1. The dual-stage framework improved fine-crack segmentation relative to the evaluated baseline methods.
2. Selective segmentation demonstrated favorable computational scalability as input image resolution increased.

What are the implications of the main findings?

3. Combining neighboring context with graph reasoning offers a promising balance between crack recovery and computational cost.
4. A public dataset of crack-containing and crack-free bridge images supports reproducible research on crack screening and segmentation.

Abstract

Fine-crack segmentation in high-resolution UAV images of bridges is hindered by weak crack signals, complex backgrounds, and substantial computational demands. This study proposes a dual-stage framework combining lightweight grid-wise screening with graph-enhanced segmentation. CrackClsNet identifies candidate regions, and CrackSegNet processes contextual patches comprising each candidate and its eight neighbors. Multiscale and deformable convolutions, attention, and cross-patch graph reasoning support feature extraction, while overlapping predictions are fused during reconstruction. In experiments using 1000 pixel-annotated crack-containing images collected from 13 highway bridges and 500 crack-free images, neighborhood retention increased candidate region crack-pixel coverage from 91.97% to 95.65% on the classification test set. The end-to-end framework achieved an F1-score of 0.5826, Crack IoU of 0.4110, and mIoU of 0.7052, exceeding six baseline methods in these metrics. The results of ablation experiments further support the benefits of neighboring context and cross-patch feature exchange, with 3 × 3 neighborhoods balancing segmentation quality and computational cost. With approximately 4.41 million parameters, the framework achieved 4.10 FPS with a reported memory consumption of 6.94 GB at 4096 × 5632 pixels, providing the highest throughput and lowest reported memory consumption among the evaluated methods at this resolution. These results indicate a promising balance between segmentation performance and computational cost on the evaluated bridge image dataset.

Academic Editors: Jesús Balado Frías, Lorenzo Teppati Losè and Zhouyan Qiu

Received: 31 August 2026
Revised: 26 October 2026
Accepted: 26 October 2026
Published: 28 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

Keywords: fine-crack segmentation; bridge inspection; crack detection; high-resolution imagery; deep learning

1. Introduction

Many bridges worldwide are deteriorating after decades of service, increasing the demand for efficient inspection and maintenance strategies. Early identification and accurate localization of fine surface cracks are important, as these defects can indicate ongoing deterioration and inform maintenance planning. Early-stage cracks may be approximately 0.1–2 mm wide. If deterioration continues, environmental exposure, cyclic loading, and material aging can promote crack growth, reinforcement corrosion, and concrete spalling, thereby reducing durability and, in severe cases, degrading structural performance. Timely detection provides a technical basis for prioritizing follow-up inspection and repair [1].

Conventional crack detection techniques have inherent limitations. Visual inspection by trained personnel [2] provides direct observations but is affected by subjectivity, limited accessibility, safety risks at height, and inconsistent documentation. Non-destructive testing (NDT) methods, such as acoustic emission [3], ultrasonic testing [4], and ground-penetrating radar [5], can provide subsurface information but are often constrained by point-by-point scanning, surface-coupling requirements, or insufficient resolution for fine surface cracks. To overcome these limitations, image-based inspection—particularly with unmanned aerial vehicles (UAVs) equipped with high-resolution cameras—has rapidly gained attention in the field [6,7]. UAV-based image inspection improves inspector safety, enables rapid coverage of large structures, reduces the need for specialized NDT equipment or scaffolding, and generates comprehensive georeferenced visual records for longitudinal analysis and objective documentation.

Early image-based crack detection methods mainly relied on traditional computer vision algorithms applied to captured images. Typical pipelines included image preprocessing for denoising or contrast enhancement, segmentation by thresholding [8] or region growing [9], edge detection using operators such as Canny [10], and feature extraction based on texture descriptors such as Gabor filters or local binary patterns [11]. Although these approaches are computationally tractable and interpretable, their performance is highly sensitive to real-world conditions such as variable illumination, strong shadows, surface roughness, and background clutter. Manually designing features that remain robust across diverse bridge environments remains difficult [12]. As a result, traditional methods often lack the accuracy, robustness, and generalization needed to segment subtle low-contrast cracks in operational inspection settings.

Deep learning approaches, especially convolutional neural networks (CNNs), have substantially advanced image-based crack detection by enabling automatic feature extraction from raw pixels. Unlike traditional methods based on handcrafted thresholds, edge operators, or morphological filters, deep CNNs learn hierarchical representations directly from annotated data. This end-to-end learning framework allows models to capture subtle texture changes, low-contrast boundaries, branching patterns, and irregular crack morphologies, which are often obscured by noise, stains, or non-uniform illumination.

Encoder–decoder architectures have become a dominant paradigm for deep-learning-based crack segmentation because they learn discriminative features from raw images while progressively recovering spatial resolution for pixel-wise prediction. Representative frameworks include Fully Convolutional Networks (FCNs) [13], U-Net [14], and the DeepLab series [15–17]. These encoder–decoder models have served as the basis for many recent crack segmentation methods.

FCN replaces fully connected layers with convolutional layers to support end-to-end pixel-wise prediction on arbitrarily sized inputs [13]. Its skip architecture fuses coarse semantic information from deep layers with fine appearance details from shallow layers. However, FCN has limited sensitivity to fine linear structures and often produces discontinuous crack boundaries. Attention mechanisms have been incorporated into FCN-based models to address these limitations. For example, Zhang et al. proposed an FCN-attention model with a ResNet-50 encoder and channel-spatial attention, achieving 0.9079 global accuracy and 0.6970 IoU on asphalt pavement datasets [18]. Islam and Kim developed a fully convolutional encoder–decoder network for concrete crack segmentation [19]. FCN-based segmentation has also been used for fine linear fatigue crack detection in aero-engine components [20] and for large-scale concrete inspection using photogrammetric 3D texture mapping [21].

U-Net, originally developed for biomedical segmentation, has been widely adopted for crack segmentation because its symmetric encoder–decoder structure and skip connections preserve high-resolution spatial information needed to delineate thin, elongated cracks. Its compact design is also attractive when annotated data are limited. Nevertheless, standard U-Net models still struggle with multi-scale crack variability, background noise, and extreme class imbalance. Therefore, numerous variants have been proposed. STEH-UNet [22] incorporates Swin Transformer blocks, Haar wavelet downsampling, and coordinate-attention-based feature fusion. EFE-UNet [23] uses a lightweight EfficientNet-based backbone, Bi-Level Routing Attention, and depthwise separable convolutions to reduce complexity while maintaining accuracy under seam-like interference. DRA-UNet [24] combines residual learning, dual attention, and atrous spatial pyramid pooling for UAV-based mining surface crack detection. Residual Sharp U-Net [25] focuses on preserving sharp crack boundaries and mitigating class imbalance in the context of pavement crack severity assessment.

DeepLab uses atrous convolution to enlarge the receptive field without increasing parameters or sacrificing spatial resolution, and it employs atrous spatial pyramid pooling (ASPP) to capture multi-scale patterns. DeepLabV3+ further adds a decoder module to refine segmentation boundaries, making it effective for pixel-level crack analysis. Recent domain-specific variants include EDeepLab [26], which replaces the backbone with EfficientNetV2 and integrates attention mechanisms for tunnel lining cracks; Parallel-TDNet [27], which combines self-attention and squeeze-and-excitation blocks for pavement sealed-crack detection; and EKD-DeepLab, which uses dual-teacher knowledge distillation to train a compact student network for aircraft surface crack segmentation [28].

Beyond these canonical architectures, Transformer-based models have been investigated to capture longer-range dependencies. Examples include BridgeFusionNet, which iteratively fuses convolutional and self-attention features [29]; MorFormer, which adapts attention receptive fields to crack morphology [30]; and a crack-location segmentation Transformer that incorporates positional coordinates for efficient instance segmentation [31]. Generative models have also been used to augment limited training data, including SinGAN-based crack synthesis [32] and CrackGauGAN image–mask generation [33].

Selective processing has also been investigated through two-stage crack detection and segmentation frameworks. Liu et al. developed a two-step method in which a modified YOLOv3 detector first identifies pavement crack regions, followed by a modified U-Net that segments the detected regions at the pixel level [34]. More recently, Yue et al. proposed TSPCS-net, which uses a classification network to exclude crack-free images before an encoder–decoder network segments the images classified as containing cracks [35]. These studies establish relevant precedents for selective segmentation. For high-resolution bridge imagery, a related design consideration is how to retain surrounding evidence

when selecting candidate regions, particularly where narrow cracks extend across patch boundaries.

Graph-based contextual modeling provides another relevant methodological direction. Wang et al. proposed GGMNet, which combines pyramid pooling and graph reasoning to aggregate multi-scale features and model relationships between spatially distant regions in pavement images [36]. Liu et al. developed CGV-Net for tunnel lining crack segmentation, integrating convolutional features, graph neural networks, and a Vision Transformer to facilitate information exchange among encoded local regions [37]. These approaches provide a basis for investigating relational modeling alongside convolutional feature extraction. In a selectively processed patch-based framework, this motivates examining how contextual information can be exchanged among retained regions while controlling the computational cost of segmentation. Contextual modeling in other visual tasks provides an additional methodological perspective. Tian et al. proposed a knowledge-guided multimodal network for video summarization, combining a knowledge-guided encoder with neighbor-aware attention to enrich semantic representations and incorporate information from neighboring frames [38]. Their work illustrates the relevance of contextual relationships when local visual evidence is insufficient for interpretation.

Despite these advances, fine-crack delineation in high-resolution images remains challenging. When a crack is represented by only a few pixels, three related issues become important. First, repeated downsampling can attenuate sparse, low-contrast signals, and skip connections cannot fully recover information that has already been removed. Second, foreground–background imbalance is extreme because crack pixels occupy only a minute fraction of the image. Third, direct full-resolution segmentation requires substantial memory and computation, whereas aggressive resizing or nonoverlapping tiling can weaken fine detail, introduce boundary artifacts, or remove context needed to distinguish cracks from scratches, joints, shadows, and surface texture. Many global self-attention formulations also scale quadratically with the number of spatial tokens, which limits direct application to very large images.

Building on these research directions, this study investigates a dual-stage framework that combines selective segmentation with neighboring-context retention and graph-based feature interaction for fine-crack segmentation in high-resolution bridge imagery, as shown in Figure 1. First, a lightweight binary classification network screens small patches arranged on a regular image grid. The positively classified patches and their immediate spatial neighbors are then used to construct contextual inputs for a compact segmentation network. Within this network, spatially aligned encoder features are aggregated and processed through graph-based reasoning, after which the refined features are mapped back to the patch feature maps for decoding. Overlapping predictions are subsequently fused to reconstruct the full-resolution segmentation map. This combination is designed to concentrate segmentation computation on candidate regions while retaining contextual information relevant to cracks extending across patch boundaries.

The main contributions of this study can be summarized as follows:

1. An efficient dual-stage framework is proposed that combines lightweight grid-wise crack region screening with graph-enhanced pixel-level segmentation, designed to retain fine-crack details and spatial context in high-resolution bridge images. By restricting computationally intensive segmentation to candidate regions and neighboring patches, the proposed method balances segmentation accuracy, inference speed, and memory efficiency.
2. We introduce and publicly release a high-resolution bridge image dataset comprising 1500 images, including 1000 UAV-acquired RGB crack images collected from 13 highway bridges and 500 crack-free background images. The crack-containing images are accompanied by pixel-level annotations and predominantly depict fine cracks with

estimated widths of approximately 0.1–2 mm, based on image-pixel widths and nominal *GSD*. The crack-free images provide negative samples for training and evaluating candidate region screening. Together, these images support the reproducible development and evaluation of crack classification and fine-crack segmentation methods. The dataset is openly available on Figshare at https://doi.org/10.6084/m9.figshare.33000635.

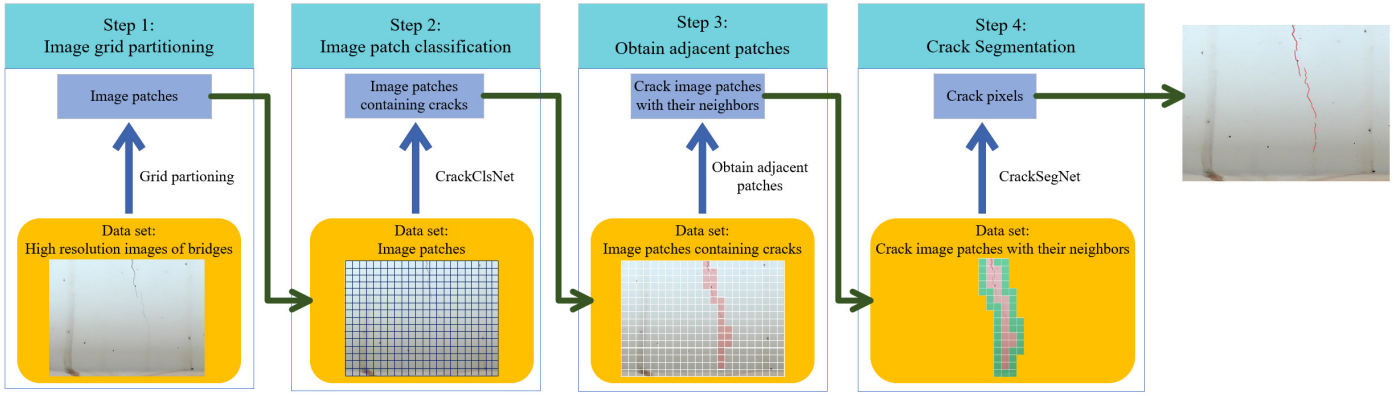


Figure 1. Overview of the proposed dual-stage framework for fine-crack segmentation. Pink grid cells denote patches classified as crack-containing, and green cells denote retained neighboring patches. Blue arrows indicate processing within each stage, while dark-green arrows indicate transfers between stages. The red trace in the final image indicates the predicted crack pixels.

The remainder of this study is organized as outlined below.

Our proposed dual-stage framework for pixel-level fine-crack segmentation in high-resolution bridge imagery is detailed in Section 2. The experimental results are presented and discussed in Section 3. Finally, Section 4 concludes this study.

2. Materials and Methods

2.1. Grid-Wise Crack Detection with CrackClsNet

In typical crack images, crack pixels account for only a very small proportion of the total image area, while most pixels belong to the background. This redundancy makes direct pixel-level processing of the original high-resolution image computationally expensive. To reduce the memory burden, the proposed framework first detects image regions that are likely to contain cracks. The framework retains patches predicted as crack-containing for pixel-level segmentation, while the remaining patches are not forwarded to the segmentation network and are treated as background during reconstruction.

Many object detection frameworks, including the YOLO series [39–41] and R-CNN series [42–44], have been widely used for visual defect localization. However, most mainstream detectors cannot efficiently process very high-resolution inputs without downsampling them to a manageable size. This is problematic for crack detection, as cracks are often thin linear structures represented by only a few pixels. Therefore, downsampling can remove critical fine-scale details and make cracks difficult to detect or localize.

To preserve fine-crack details while controlling computational cost, the proposed method operates directly on the original high-resolution image. The image is first partitioned into a regular grid, and each patch is evaluated by a lightweight neural network, CrackClsNet (approximately 0.49M parameters), to determine whether it contains crack features.

As shown in Figure 2, CrackClsNet adopts a cascaded downsampling architecture comprising six sequential ConvBlocks and a lightweight classification head. Each ConvBlock extracts multi-scale representations through four parallel convolutional branches:

a 1 × 1 convolution, a standard 3 × 3 convolution, and two 3 × 3 dilated convolutions with dilation rates of 2 and 3. Each branch produces one quarter of the block's output channels, and the resulting feature maps are concatenated along the channel dimension. The concatenated features are then refined by a Convolutional Block Attention Module (CBAM) [45], which applies channel attention through a shared multilayer perceptron over pooled spatial descriptors and spatial attention through convolution over concatenated mean and max activations. A 2 × 2 max-pooling layer with stride 2 halves the spatial resolution after attention recalibration. Starting from a 64 × 64 RGB patch, CrackClsNet increases the feature depth from 16 to 32, 64, 96, 128, and finally 256, while reducing the spatial resolution to 1 × 1 after the sixth block. The classifier compresses the 256-dimensional feature vector to 32 channels with a 1 × 1 convolution and ReLU activation, and a final 1 × 1 convolution projects it to a single logit. After sigmoid activation, this scalar represents the probability that the corresponding patch contains crack pixels.

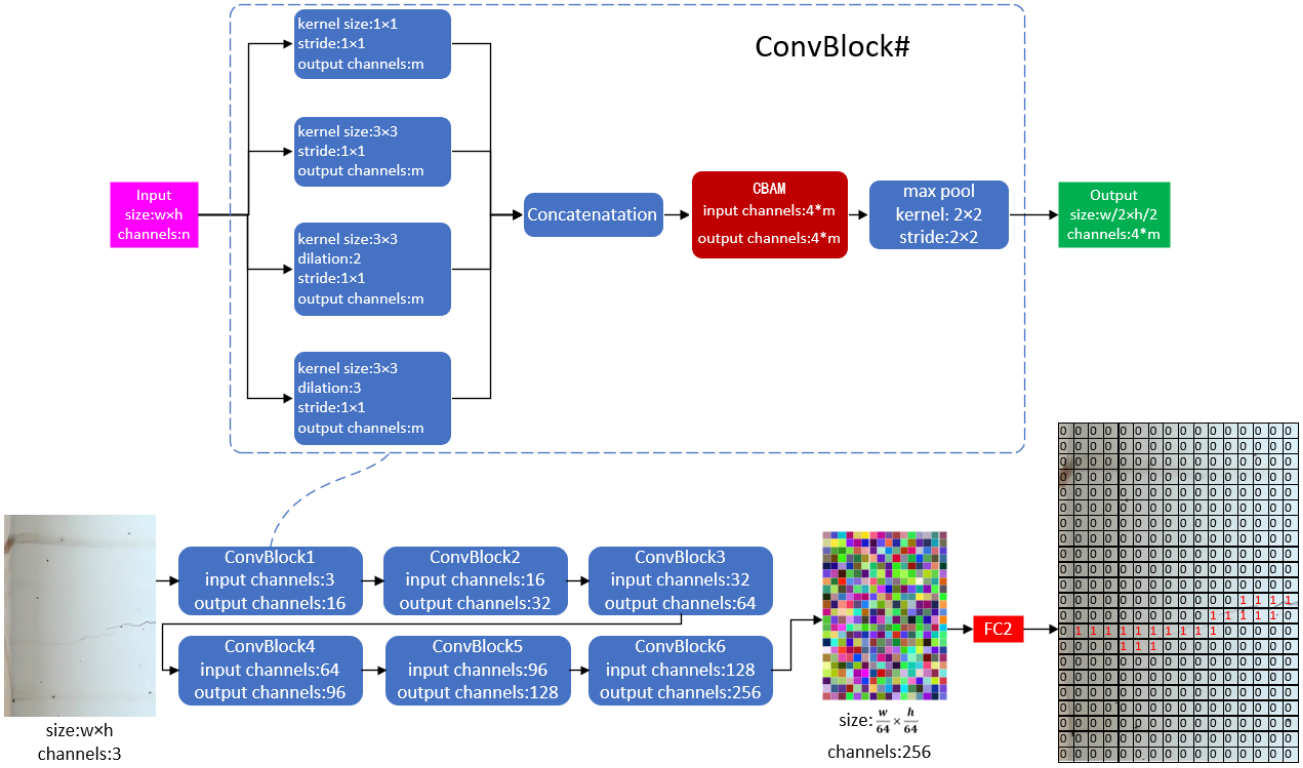


Figure 2. Architecture of CrackClsNet. Magenta and green boxes indicate input and output tensors, respectively. Blue boxes indicate convolution, concatenation, or pooling operations, and red boxes indicate attention or prediction operations. The symbol # denotes the ConvBlock index, and * denotes multiplication in channel counts. Red 1s and black 0s in the output grid indicate crack-containing and crack-free patch predictions, respectively.

CrackClsNet is designed to detect subtle cracks that vary greatly in width, orientation, contrast, and surrounding texture. Cracks often appear as thin elongated structures and can be confused with noise, shadows, or surface irregularities if only local edge information is used. Therefore, each ConvBlock integrates multi-scale dilated convolutions: the 1 × 1 kernel captures point-wise intensity cues, the standard 3 × 3 kernel captures fine local textures, and the dilated 3 × 3 kernels expand the receptive field without increasing the parameter count. Concatenating these representations provides features at multiple receptive-field scales, while CBAM recalibrates their channel and spatial responses. These operations are intended to support discrimination between crack-containing and background patches. The six-stage downsampling process gradually expands the receptive

field and builds a hierarchy from low-level edges to high-level semantic patterns. By the final stage, the spatial resolution is reduced to 1 × 1, allowing the classifier to make a patch-level classification decision. During training, a weighted binary cross-entropy (BCE) loss is used to address the severe imbalance between crack and non-crack grids:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[\alpha \cdot y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)] \quad (1)$$

where N is the number of patches, $y_i \in \{0,1\}$ is the ground-truth label for patch i (1 for crack and 0 for non-crack), $\hat{y}_i$ is the predicted probability for patch i, and α is the positive-class weight, which is set to 4.0. The weighted BCE objective compensates for the low prior probability of cracks and encourages the optimizer to learn discriminative crack features. The loss is computed for each grid from the scalar sigmoid output, prioritizing recall for cracked regions while maintaining a reasonable false-positive rate.

2.2. Pixel-Level Crack Segmentation with CrackSegNet

After CrackClsNet classification, the image is divided into crack-containing patches (S-patches) and crack-free patches (T-patches). To reduce the memory footprint and computational cost of pixel-level segmentation, only the S-patches and their immediate neighboring patches are retained; the remaining T-patches are discarded. On this basis, we design CrackSegNet to accurately segment crack pixels from the retained regions.

2.2.1. Neighbor-Aware Extraction of Candidate Crack Patches

To reduce the computational load of the segmentation stage, crack pixels are segmented only from S-patches and their immediate neighbors. The retained set is not restricted to S-patches alone, as illustrated in Figure 3. Including the surrounding patches slightly increases computation, but it provides two advantages. First, it mitigates boundary effects by supplying contextual information for cracks located near patch borders. Second, it compensates for errors made by CrackClsNet. Some truly cracked patches may be misclassified as crack-free T-patches, as shown in Figure 4, and these missed patches often lie adjacent to correctly detected S-patches. Therefore, processing the neighborhood around each detected S-patch reduces false negatives caused by upstream classification errors.

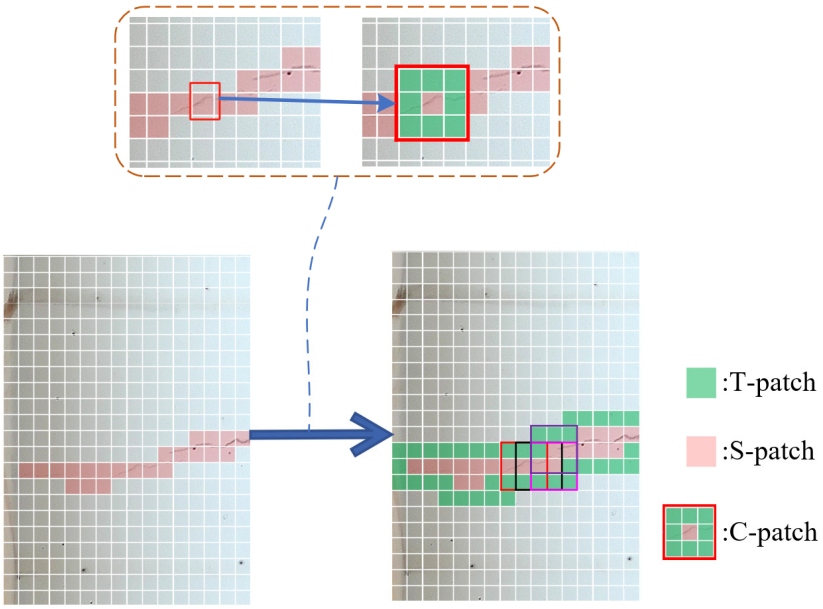


Figure 3. Extraction of 3 × 3 contextual patches for pixel-level crack segmentation. Pink cells denote S-patches, and green cells denote retained neighboring T-patches. Red, black, and purple outlines

distinguish overlapping C-patches centered on different S-patches. The dashed orange frame provides an enlarged illustration of neighborhood extraction.

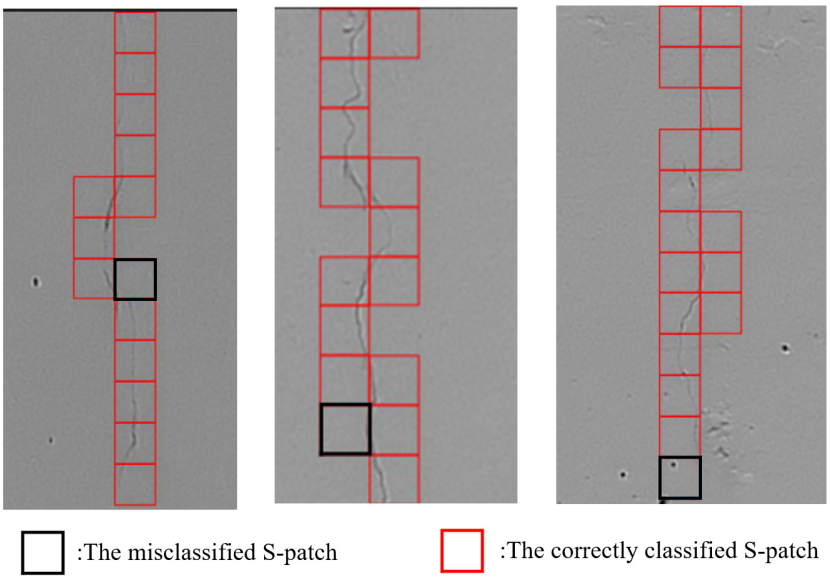


Figure 4. Examples of crack-containing patches missed by CrackClsNet.

For each selected S-patch, the patch itself and its eight immediate neighbors are merged into a 3 × 3 contextual patch, referred to as a C-patch, with a spatial size of 192 × 192 pixels. The center coordinate of the C-patch is (C_H, C_W). All C-patches extracted from an image are then used in the pixel-wise segmentation stage. Adjacent C-patches may overlap. Although this overlap introduces additional computation, it is intended to reduce boundary artifacts associated with rigid grid partitioning and to provide contextual information for interpreting crack segments that extend across patch borders.

2.2.2. Graph-Based U-Net for Crack Segmentation

U-Net is effective for image segmentation because its symmetric encoder–decoder design and skip connections combine multi-scale context with fine spatial detail. However, vanilla U-Net is suboptimal for fine-crack segmentation, where foreground pixels are extremely sparse, fine structures are easily weakened or lost during standard downsampling, and local convolutions have limited ability to capture long-range crack continuity. In addition, the retained crack-related regions are spatially distributed across the original image rather than forming a single continuous field, so their relationships must be modeled explicitly. To address these challenges, we build on the U-Net paradigm and design CrackSegNet, a graph-enhanced segmentation network for accurate pixel-level crack segmentation.

As shown in Figure 5, the input to CrackSegNet consists of all C-patches extracted from a single image and stacked into a tensor of shape N × 192 × 192 × 3. Here, N denotes the number of C-patches, 192 × 192 denotes the spatial size of each patch, and 3 denotes the RGB color channels. CrackSegNet adopts a U-shaped encoder–decoder architecture augmented with graph-based contextual reasoning. An initial convolutional stem first extracts low-level features, after which five Down modules progressively reduce the spatial resolution using pixel-unshuffle [46] operations while increasing the channel dimensionality. Each Down module employs parallel 1 × 1 and 3 × 3 convolutional branches with dilation rates of 1, 2, and 3 to capture crack patterns at multiple receptive-field scales. Standard convolutions are used in the shallow encoder stages, whereas deformable convolutions [47] and Convolutional Block Attention Modules are introduced in the deeper

stages to adapt the sampling locations to irregular crack geometries and emphasize informative channel-spatial responses. At the bottleneck, the deepest CNN features are flattened and aggregated according to predefined spatial indices to construct graph node representations. These nodes are processed by a graph neural network (GNN) with graph attention layers [48], allowing contextual information to be exchanged among neighboring graph nodes before patch-wise decoding. The refined node embeddings are then remapped to the spatial grid by a feature-gather module, concatenated with the deepest encoder features, and fused through a lightweight convolution. The decoder comprises five Up modules that combine pixel-shuffle upsampling, encoder–decoder skip connections, parallel pointwise and deformable convolutions, and channel-spatial attention to progressively recover fine spatial details and improve crack-boundary localization. Finally, two pointwise convolutional layers transform the decoded features into a single-channel pixel-wise crack logit map, to which a sigmoid function can be applied during inference to obtain crack probabilities.

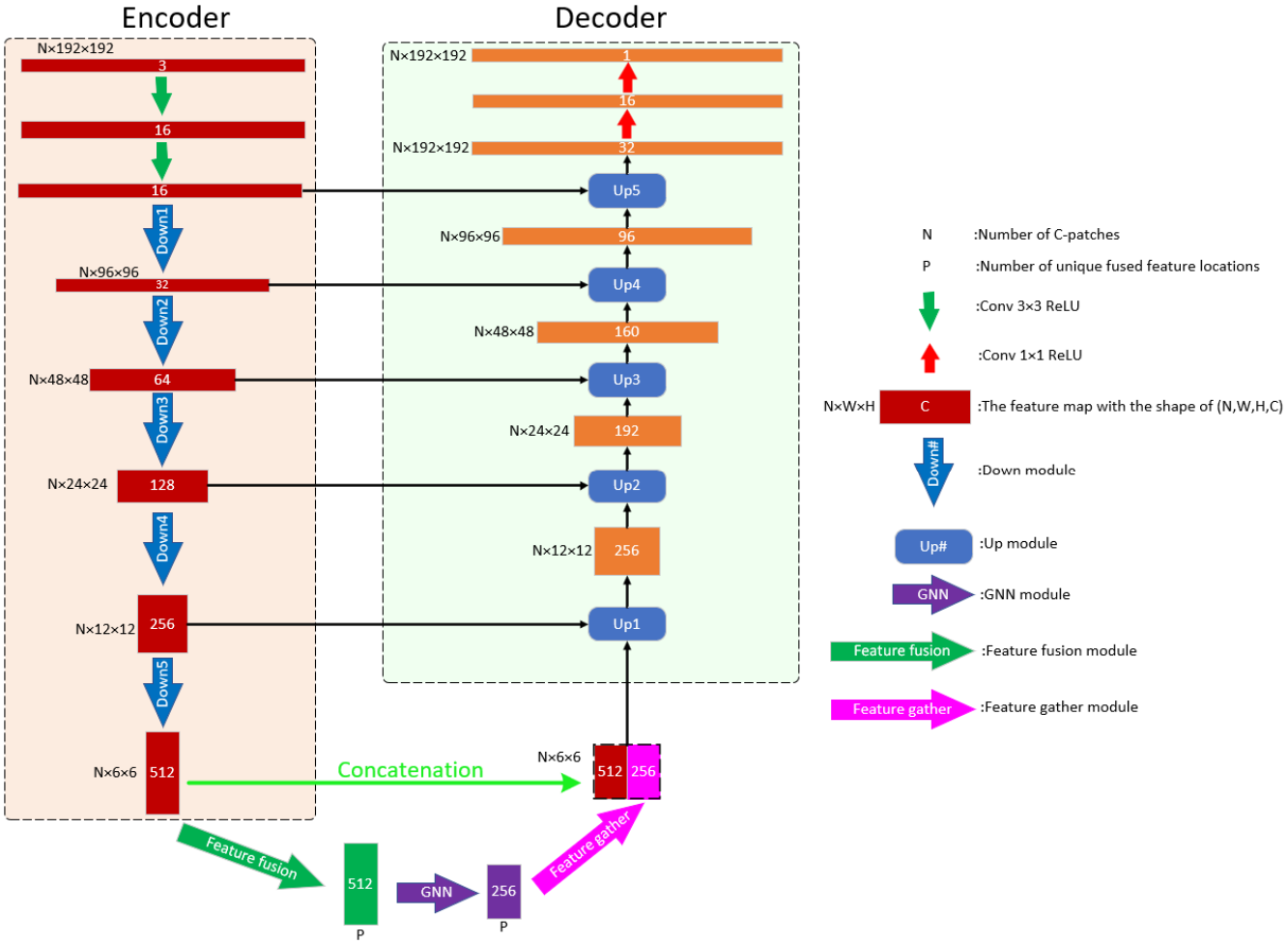


Figure 5. Architecture of CrackSegNet.

(1) Down module

The Down module, illustrated in Figure 6, is the primary encoding unit. It progressively reduces spatial resolution while aggregating multi-scale crack features from the input. The module processes a feature map through four parallel branches: a pointwise 1 × 1 convolution for channel-wise mixing and three deformable convolutional blocks with 3 × 3 kernels and dilation rates of 1, 2, and 3. These dilated deformable convolutions adapt their sampling locations to the elongated and irregular geometry of cracks while covering

different receptive-field scales. The branch outputs are concatenated along the channel dimension and passed through a CBAM, which emphasizes crack-relevant channels and spatial positions. A subsequent 1 × 1 convolution reduces the channel depth by half, and a pixel-unshuffle operation with a downscale factor of 2 folds spatial detail into the channel dimension. Pixel unshuffle reduces the spatial dimensions by rearranging activations into the channel dimension. The rearrangement itself does not discard activations.

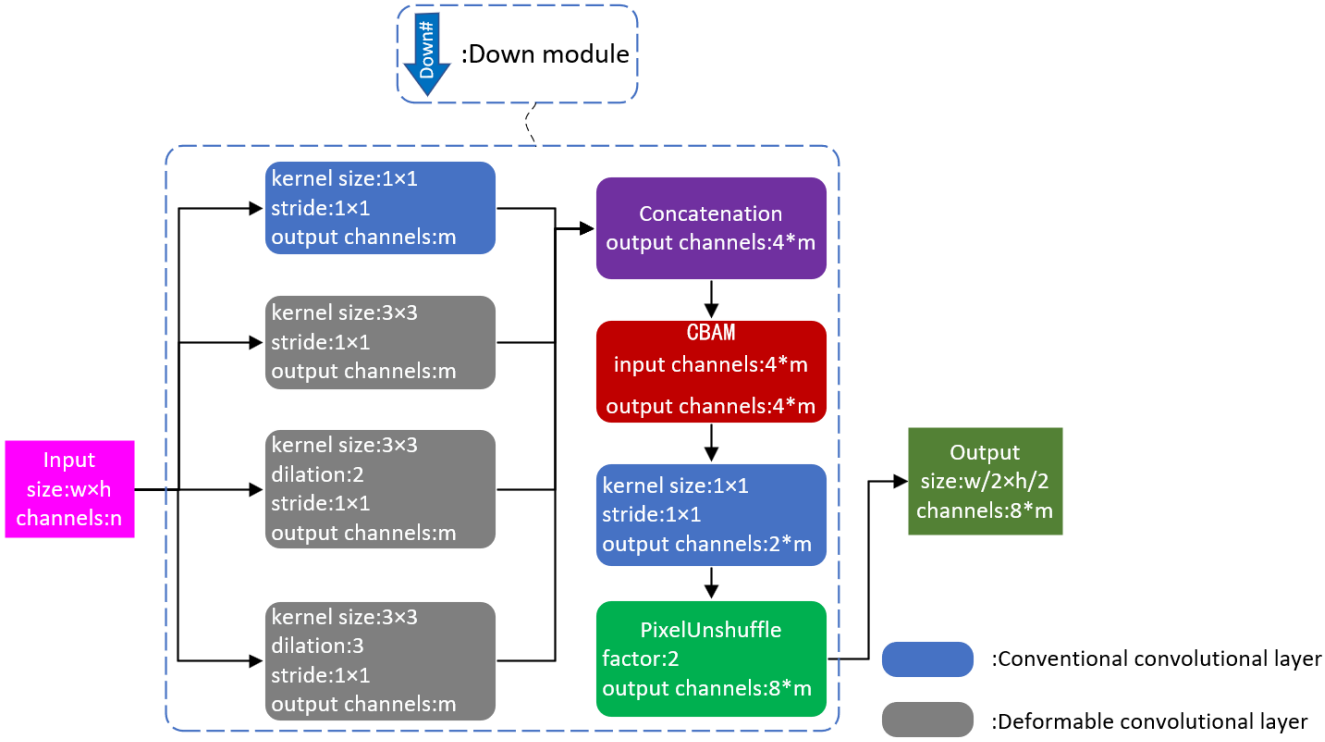


Figure 6. Architecture of the Down module in CrackSegNet. Blue and gray blocks denote conventional and deformable convolutional layers, respectively. The other operations are explicitly labeled in the diagram. The symbol # denotes the Down-module index, and * denotes multiplication in channel counts.

The Down module is designed around information-preserving spatial rearrangement, multi-scale crack perception, and symmetry with the decoder. In contrast to pooling or strided convolution, the pixel-unshuffle operation transfers spatial information into channel depth and helps preserve faint low-contrast crack signals. The multi-branch deformable convolutions capture cracks with different widths, orientations, and morphologies, while CBAM further suppresses irrelevant texture responses. The final 1 × 1 convolution compacts the attended features before the pixel-unshuffle operation, producing an information-dense representation. This design is particularly useful in an overlapping patch-based pipeline, where subtle crack signatures must remain visible through deeper encoding stages.

(2) Feature fusion module

Because adjacent C-patches overlap, their encoder feature maps also contain redundant regions. To obtain a unique feature representation at each spatial location for the subsequent GNN, we introduce the feature-fusion module shown in Figure 7. Consider a C-patch of 192 × 192 pixels whose center coordinates in the original image are (C_H, C_W). After encoding, the patch yields a 6 × 6 feature map with 512 channels. For a feature pixel with local coordinates (f_h, f_w), its corresponding global location index i in the original image is calculated as follows:

$$f_H = (f_h - 3) \times \frac{64}{2} + \frac{64}{4} + C_H \tag{2}$$

$$f_W = (f_w - 3) \times \frac{64}{2} + \frac{64}{4} + C_W \tag{3}$$

$$i = f_H \times W_0 + f_W \tag{4}$$

where W_0 denotes the number of columns in the original image, and (f_H, f_W) denotes the center coordinates of the feature pixel in the original image.

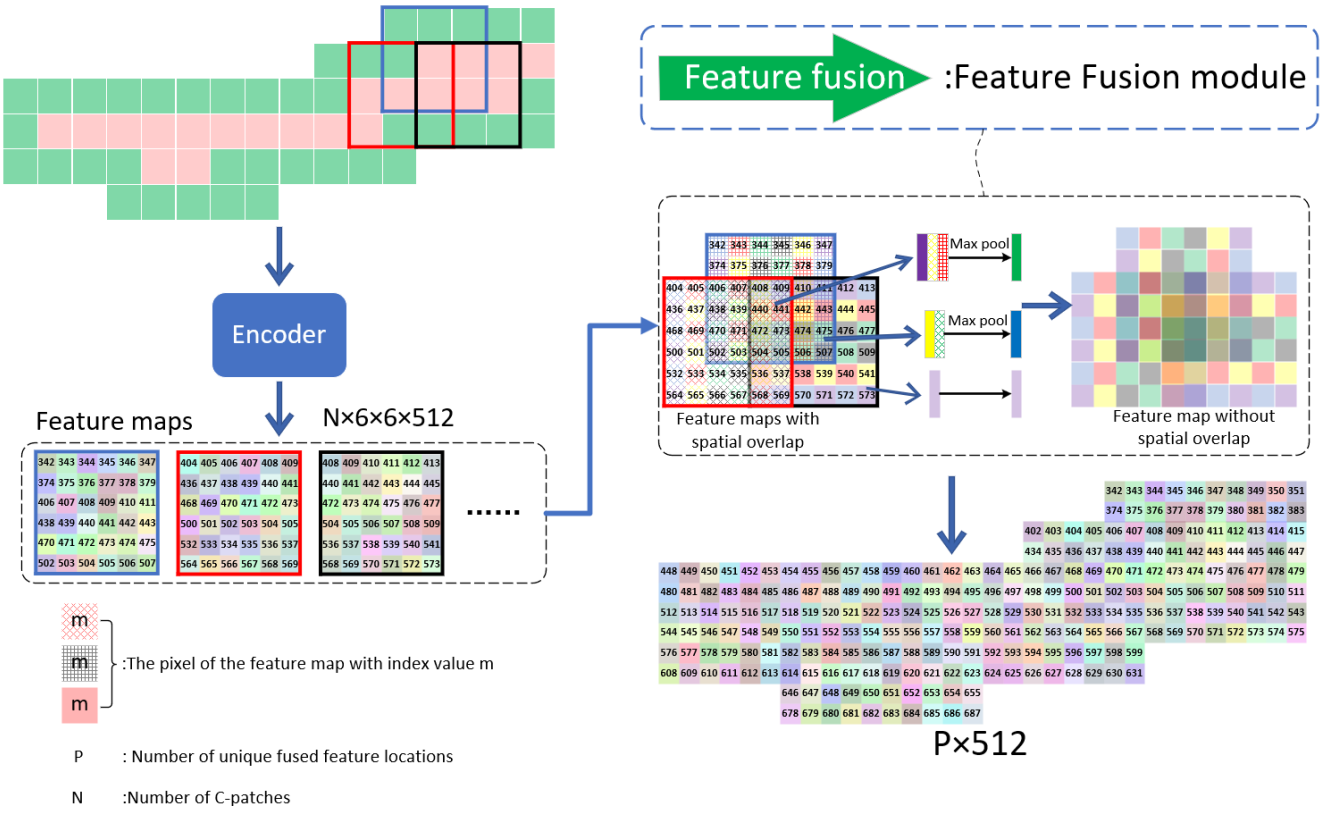


Figure 7. Aggregation of overlapping encoder features in the feature fusion module. Blue, red, and black outlines identify different C-patches and their corresponding encoder feature maps. Matching numerical indices identify features at the same global spatial location. Cell colors are schematic, and arrows indicate feature flow and element-wise max-pooling aggregation.

Let G_i denote the index of a feature pixel G. We collect all feature vectors from C-patches that correspond to the same index G_i, forming a set $\boldsymbol{C} = \{C_1, C_2, \ldots, C_m\}$, where m is the number of C-patches covering that global location. The set is fused by element-wise max pooling to produce a single fused vector $C_G = MaxPool(C_1, C_2, \ldots, C_m)$. Repeating this operation for all covered locations yields a global feature map of size P × 512, where P denotes the number of unique fused feature locations obtained from the retained patches. This fusion resolves overlap-induced redundancy and provides a consistent per-location feature set for GNN-based spatial analysis.

(3) GNN module

As each patch covers only a limited area, features extracted from a single patch may lack sufficient contextual information for reliable crack identification. The GNN module expands the effective receptive field by connecting feature pixels in a graph and applying graph neural network layers to integrate contextual information across the retained regions. For the fused feature map of size P × 512, the feature graph is constructed as follows:

5. Graph construction: Spatial relationships between feature pixels are represented by an undirected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$.
6. Nodes ($\boldsymbol{V}$): Each feature pixel is treated as a node $v_i \in \boldsymbol{V}$.
7. Edges ($\boldsymbol{E}$): Nodes v_i and v_j are connected by an undirected edge if they are adjacent on the feature map space. Groups are obtained by the transitive closure of these links. Each group is processed independently. An example graph is shown in Figure 8.
8. Node attribute (h_i): The 512-dimensional feature vector of each feature pixel obtained from the feature fusion module.
9. Edge attribute (m_{ij}): A vector encoding the coordinates and relative displacement of two connected nodes:

$$e_{\mathrm{ij}} = [\, f_H^j - f_H^i, f_W^j - f_W^i, \alpha, \beta] \tag{5}$$

$$\alpha = \sqrt{((f_H^j - f_H^i)^2 + (f_W^j - f_W^i)^2)} \tag{6}$$

$$\beta = \mathrm{atan2}(f_W^j - f_W^i,\ f_H^j - f_H^i) \tag{7}$$

where (f_H, f_W) denotes the center coordinate of a feature pixel. A two-layer Graph Attention Network (GAT) [48] is then applied. For the first layer, node features are first projected as

$$z_i(l) = W(l) \times h_i(l-1) \tag{8}$$

$$z_j(l) = W(l) \times h_j(l-1) \tag{9}$$

$$r_{ij} = W_e \times e_{\mathrm{ij}} \tag{10}$$

Here, h_i and h_j are the input features of node *i* and node *j*, respectively. $W(l)$ and W_e are learnable node and edge feature projections, respectively, and the attention vectors are also learned. For an incoming edge from node *i* to node *j*, the attention computation is

$$S_{ij} = \mathrm{LeakyReLU}\left(\alpha_{ij}{}^T \cdot Concat(z_i, z_j, r_{ij})\right) \tag{11}$$

$$\alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{k \in N(i)} \exp(s_{ik})} \tag{12}$$

Here, $N(i)$ is the set of neighbors of node *i*, which may include self-loops. The edge feature projection maps the four-dimensional descriptor to the output dimension of the corresponding layer.

The node *i* update is

$$h_i' = \mathrm{ReLU}\left(\sum_{j \in N(i)} \alpha_{ij} W h_j\right) \tag{13}$$

Thus, edge attributes influence the attention coefficients, while the messages being aggregated consist of projected node features. The first and second layers produce 128- and 128-dimensional node features, respectively. Their outputs are concatenated:

$$g_i = Concat(h_i(1), h_i(2)) \tag{14}$$

$$Dimension\ of\ g_i = 128 + 128 = 256 \tag{15}$$

Both layers use one attention head, zero attention dropout, a LeakyReLU negative slope of 0.2, and a learnable output bias. Each GAT layer is followed by LayerNorm and ReLU. No additional graph feature dropout or residual addition is applied. Self-loops are

added by the GAT operator. Their edge attributes are generated by averaging the incoming non-self-edge attributes. For a node without incoming spatial edges, the self-loop attributes are zero, and the node is updated using its own projected features rather than being discarded.

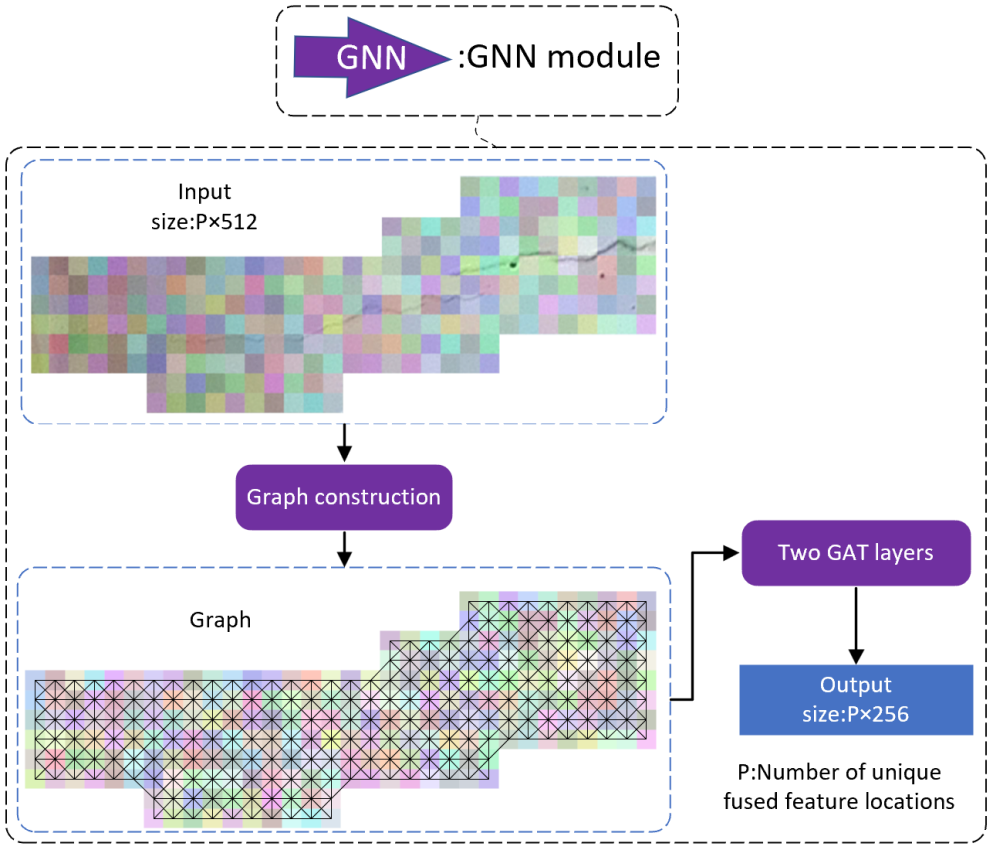


Figure 8. Graph construction and attention-based feature propagation in the GNN module. Colored cells schematically represent feature locations, and black line segments indicate graph connections between spatially adjacent locations. The colors do not encode attention weights or segmentation classes.

At image boundaries, contextual extraction windows are shifted inward to remain within the image, and duplicate crop locations are removed. Graph edges are constructed only between retained feature locations within the same group. Missing neighbors are omitted, without wrapping across image boundaries or inserting connections across spatial gaps. Attention is normalized over the available neighbors and self-loop. Therefore, the two-layer GAT propagates information over at most two graph hops and does not exchange information between disconnected components. GAT learns normalized attention weights over neighboring nodes, providing data-dependent aggregation of their feature representations.

(4) Feature-gather module

The feature-gather module maps the graph-enhanced node features back to the spatial feature grid, thereby connecting global relational reasoning with local convolutional decoding. As shown in Figure 9, it takes the enriched node feature matrix from the GNN module (size P × 256) and the inverse index mapping derived during feature fusion. Through index-based selection, the module gathers the corresponding node representation for each spatial position and reshapes the resulting sequence into a feature map of size N × 6 × 6 × 256. This operation restores the correspondence between graph nodes and

patch-grid locations, allowing the decoder to combine graph-derived contextual features with encoder skip features.

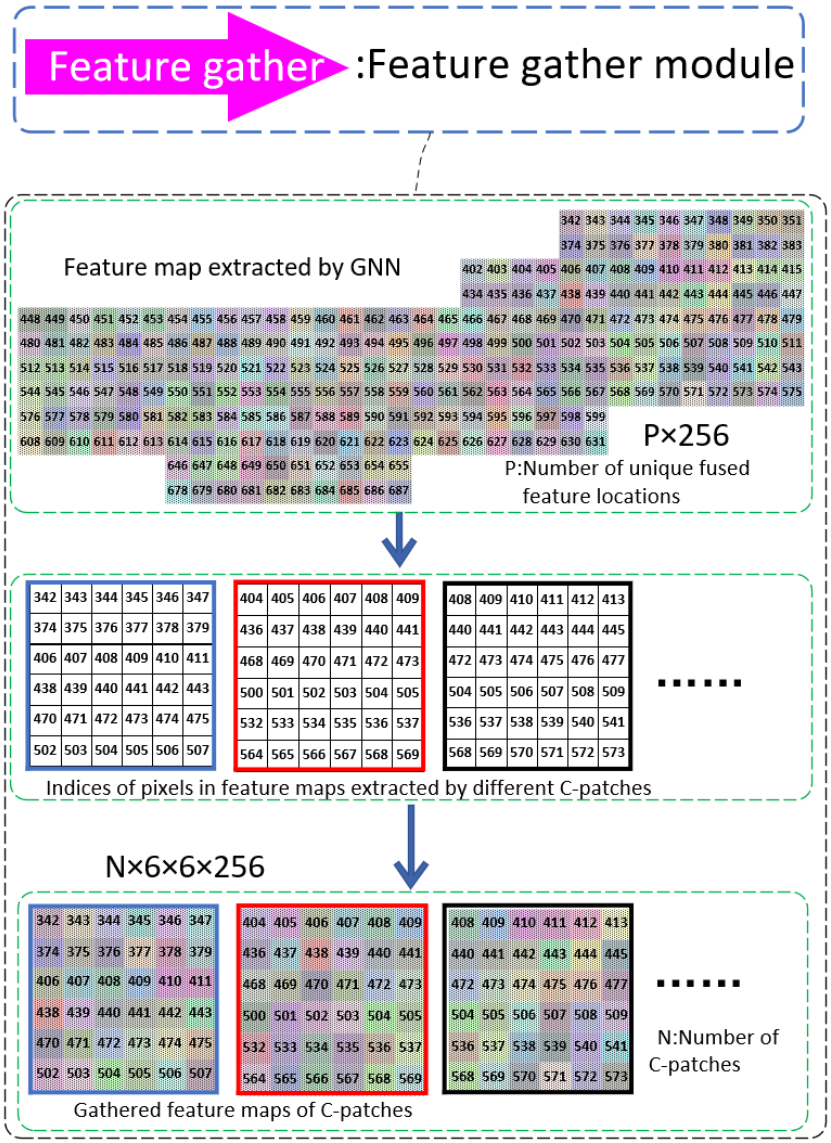


Figure 9. Mapping graph features back to patch feature maps using the feature-gather module. Blue, red, and black outlines identify different C-patches and their corresponding gathered feature maps. Matching numerical indices indicate correspondence to the same global feature location. Cell colors are schematic, and blue arrows indicate the feature-gathering sequence.

(5) Up module

The Up module is the primary decoding unit that progressively restores spatial resolution while incorporating fine-grained details from the encoder. As shown in Figure 10, the module receives a high-level feature map from the preceding decoder stage and a skip-connected feature map from the corresponding encoder stage. It first applies a pixel-shuffle operation with an upscale factor of 2. Unlike transposed convolution or interpolation-based upsampling, the pixel-shuffle operation rearranges channel information into spatial dimensions and can reduce checkerboard artifacts associated with some transposed-convolution configurations. The upsampled feature map is concatenated with the skip-connected encoder feature map, combining semantic information with spatial detail. The resulting feature volume is refined by an UpConv block with two branches: a 1 × 1 convolution for channel-wise interactions and a 3 × 3 deformable convolution for adaptive

spatial sampling. The branch outputs are concatenated, recalibrated by CBAM, and fused by a final 1 × 1 convolution to produce a refined feature map.

The Up module follows three principles: spatial resolution recovery through pixel rearrangement, deformation-adaptive detail enhancement, and symmetric information flow across the encoder–decoder pathway. It reverses the Down module using a pixel-shuffle operation to redistribute channel activations back into the spatial domain. This helps retain thin and faint crack signals that can be weakened by conventional pooling or strided operations. Deformable convolution within UpConv adapts the fused features to local crack geometry, improving sensitivity to tortuous, branched, and very thin cracks while suppressing irrelevant background structures. This design supports high-fidelity reconstruction of crack segmentation maps, especially near patch boundaries.

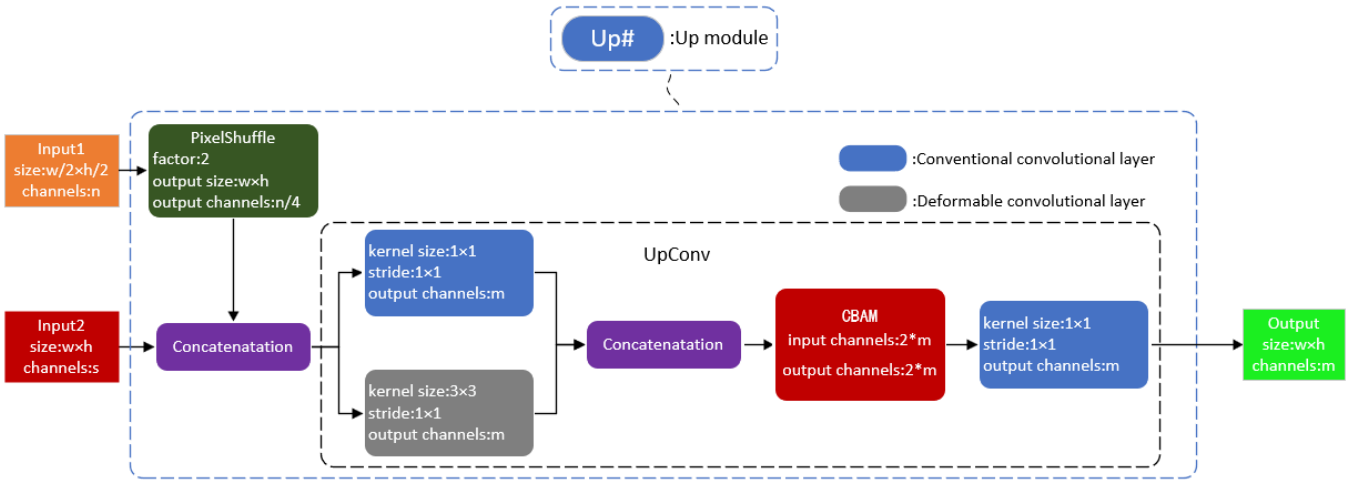


Figure 10. Architecture of the Up module in CrackSegNet. The symbol # denotes the Up-module index, and * denotes multiplication in channel counts.

2.2.3. Fusion of Overlapping Patch Predictions

To resolve ambiguities in overlapping regions, a fusion strategy is used when reconstructing the full-resolution probability map from C-patch predictions. Because adjacent C-patches share pixels, a given pixel may receive several predicted crack probabilities. Let $M = \{m_1, m_2, \ldots, m_n\}$ be the set of predicted probabilities for a single pixel, where n is the number of C-patches covering that pixel. The final probability is obtained by averaging all values in M. In implementation, predicted probabilities and a coverage counter are accumulated over the full image canvas. After all patches are processed, the accumulated probability sum is divided by the corresponding count to obtain the fused probability map. The map is then binarized with a fixed threshold of 0.5 to produce the final pixel-level crack segmentation.

2.2.4. Loss Function

The final decoder produces a crack logit for each pixel. To address severe foreground–background imbalance and optimize region overlap, the loss combines two complementary terms: weighted BCE and Tversky loss [49].

Weighted BCE increases the penalty assigned to crack pixels. For predicted probabilities $\hat{y}_i$ and binary labels y_i over N spatial locations, the loss is defined as

$$L_{\mathrm{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}[w y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \quad (16)$$

where $w = 8$ is the positive-class weight used in the reported experiments.

Tversky loss directly optimizes foreground overlap and permits different penalties for false positives and false negatives. With smoothing constant ε, it is defined as

$$L_{\mathrm{T}} = 1 - \frac{\sum_i \hat{y}_i y_i + \varepsilon}{\sum_i \hat{y}_i y_i + m\sum_i \hat{y}_i (1 - y_i) + n\sum_i (1 - \hat{y}_i) y_i + \varepsilon} \tag{17}$$

where m = 0.7 and n = 0.3 are the coefficients applied to the false-positive and false-negative terms, respectively.

The segmentation objective is the weighted sum of the BCE and Tversky terms:

$$L_{\mathrm{seg}} = \lambda_{\mathrm{BCE}} L_{\mathrm{BCE}} + \lambda_{\mathrm{T}} L_{\mathrm{T}} \tag{18}$$

where $\lambda_{\mathrm{BCE}} = 1.0$ and $\lambda_{\mathrm{T}} = 1.0$ in the reported experiments. Thus, the two objectives contribute equally before accounting for their numerical scales.

3. Results and Discussion

3.1. Dataset and Experimental Setup

To evaluate the proposed method, image data were collected from 13 representative highway bridges using a DJI (Shenzhen, China) Matrice 350 RTK UAV equipped with a DJI H30 multi-light zoom camera. The camera sensor measures 7.2 mm × 5.4 mm, with a pixel pitch of 0.98 μm, a maximum focal length of 172 mm, a maximum image resolution of 7328 × 5496 pixels, and a total of 40 megapixels. During image acquisition, the UAV-to-bridge-surface distance was maintained within 5 m, and the maximum zoom was used. For lens selection, the ground sampling distance (*GSD*) was calculated using the photogrammetric formula:

$$GSD = a \cdot \frac{h}{f}, \tag{19}$$

where a is the pixel pitch, h is the camera-to-crack-surface distance, and f is the focal length. At a distance of 5 m with maximum zoom,

$$GSD = \frac{0.98\ \mu\mathrm{m} \times 5000\ \mathrm{mm}}{172\ \mathrm{mm}} = 0.028\ \mathrm{mm}. \tag{20}$$

At the stated distance of 5 m and focal length of 172 mm, the nominal *GSD* is approximately 0.028 mm/pixel. Under these acquisition assumptions, a 0.1 mm-wide feature would span approximately 3.51 native image pixels. In our dataset, many images have crack widths of no more than 3 pixels. This calculation provides a geometric basis for interpreting crack scale, but does not independently establish a reliable minimum detectable width. Actual visibility also depends on contrast, focus, motion blur, viewing geometry, and noise. The approximate crack-width range of 0.1–2 mm is therefore based on image-pixel widths and nominal *GSD*. It should not be interpreted as a set of calibrated field measurements or as evidence of a validated minimum detectable crack width.

A representative dataset of 1000 crack-containing images was selected from images of 13 bridges, ensuring each bridge is represented. In many of these images, cracks are only 2 pixels wide, corresponding to an actual crack width of no more than 0.1 mm. Furthermore, the dataset includes cracks on both smooth and rough surfaces, as well as crack images affected by interference from scratches, stains, and shadows. We cropped these samples into crack-containing and crack-free images while preserving the original resolution, each at 1024 × 1408 pixels to facilitate network training. Pixel-level crack annotations for the 1000 images were generated using Labelme by 20 professionals with extensive experience in manual crack image annotation. After annotation, five professional bridge maintenance personnel conducted a quality inspection of the annotated cracks, provided feedback on mislabeling and omissions, and guided the annotators in making corrections. Ultimately, a high-quality annotated crack dataset was obtained. Representative image–label pairs are shown in Figure 11.

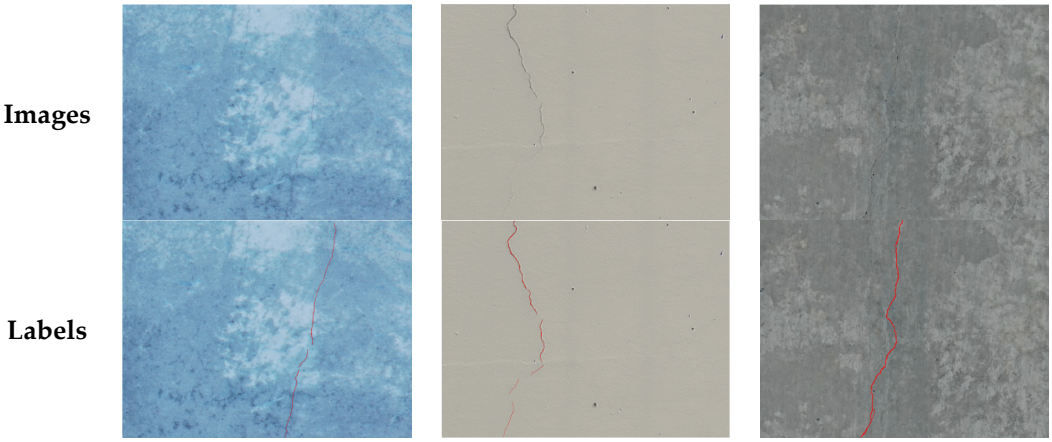


Figure 11. Representative bridge images and their corresponding pixel-level crack annotations.

To quantify inter-annotator variability in crack identification and boundary delineation, a subset of 50 crack-containing images was independently reannotated by three experienced annotators following a common annotation guideline. The annotators were blinded to the existing masks, model predictions, and one another's annotations. Pairwise foreground Dice and IoU were calculated at the native image resolution. For each image, the scores from the three annotator pairs were averaged, and the mean and standard deviation were then calculated across the 50 images. The resulting Dice and IoU were [0.85 ± 0.07] and [0.75 ± 0.08], respectively. These statistics characterize agreement among the three annotators on the sampled images and do not quantify agreement among all original annotators across the complete dataset. Uncertainty in annotation thickness and boundary placement may influence overlap-based evaluation metrics for cracks only a few pixels wide.

In addition, 500 representative crack-free images were selected from the images of these 13 bridges to serve as background images. The 500 crack-free background images contained crack-like but non-crack features, such as scratches, cold joints on concrete surfaces, and flow marks, shown in Figure 12.



Figure 12. Representative crack-free images containing crack-like background features.

Ultimately, the dataset used for network training and testing in this study comprises 1000 manually annotated crack-containing images and 500 crack-free background images (https://doi.org/10.6084/m9.figshare.33000635). The 1500 images were randomly shuffled and partitioned into Dataset A (670 images containing cracks and 330 images without cracks), Dataset B (160 images containing cracks and 70 images without cracks), and Dataset C (170 images containing cracks and 100 images without cracks). The random

partitioning was performed at the level of the prepared 1024 × 1408-pixel images. Bridge identity was not used as a grouping criterion, and independence at the flight-session or imaged-surface-area level was not established. The stage-specific training, validation, and test assignments are detailed in Sections 3.2 and 3.3. The final end-to-end test set, Dataset F, comprises 100 crack-containing images drawn from Dataset C. These images are excluded from the training and validation sets of both CrackClsNet and CrackSegNet. This arrangement provides separation at the dataset-image level.

CrackClsNet and CrackSegNet were implemented in PyTorch 2.5.1 and trained on an NVIDIA (Santa Clara, CA, USA) GeForce RTX 4070 GPU.

3.2. Training of CrackClsNet

To train CrackClsNet, each input image was divided into non-overlapping patches using a 16 × 22 grid, producing 64 × 64-pixel patches after preprocessing to a 1024 × 1408 training resolution. The label of each patch was generated automatically from the pixel-level annotation: a patch was assigned a positive label (1) if it contained at least one crack pixel and a negative label (0) otherwise. For CrackClsNet, Datasets A, B, and C were used as the training, validation, and test sets, respectively. CrackClsNet was trained from scratch for 150 epochs using the AdamW optimizer [50]. The model checkpoint was selected according to the highest validation F1-score at a fixed classification threshold of 0.5, with AP used to resolve ties. As shown in Figure 13a, the training F1-score continued to improve, whereas the validation F1-score became relatively stable during the later training stages. The resulting gap suggests some overfitting and supports selecting the checkpoint using validation performance. The best validation F1-score was 0.8968 at epoch 114, and this checkpoint was used for the final evaluation.

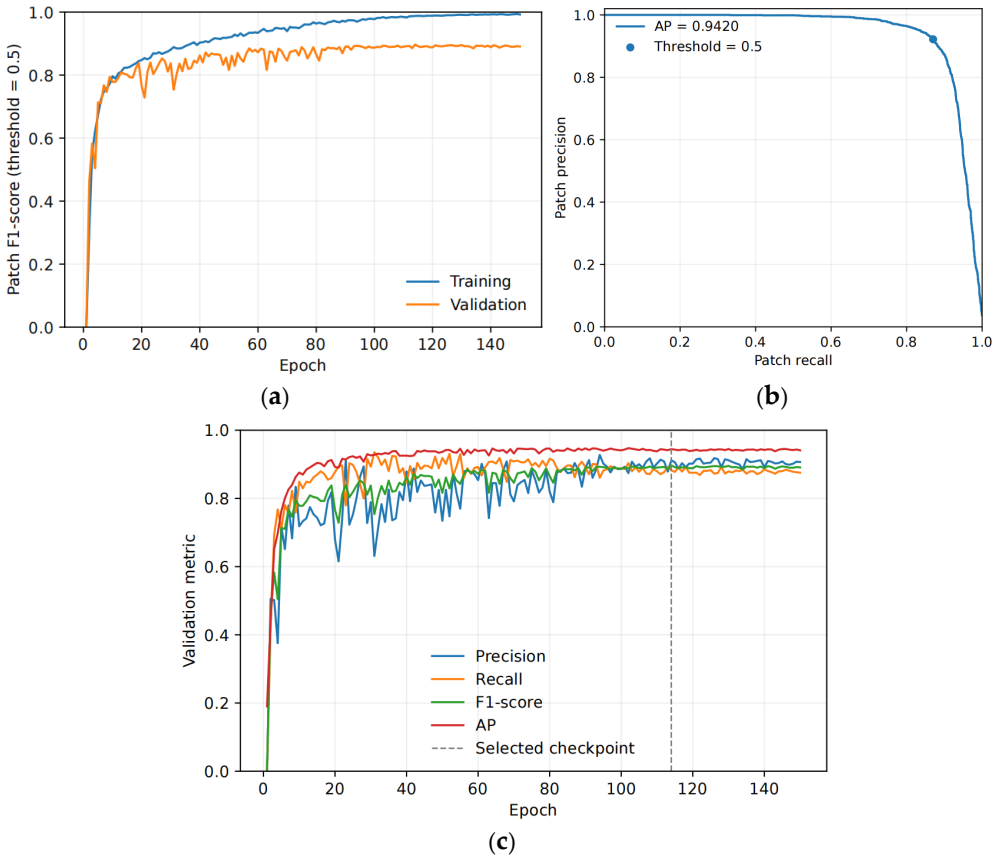


Figure 13. Training and evaluation results of CrackClsNet: (**a**) training and validation F1-scores; (**b**) test-set precision–recall curve; and (**c**) validation precision, recall, F1-score, and average precision.

The operating point at a classification threshold of 0.5 is marked in (**b**), and the selected checkpoint is indicated in (**c**).

Five metrics were used to evaluate patch-level classification: accuracy, precision, recall, *F*1-score, and average precision (AP). Accuracy, precision, recall, and *F*1-score were calculated using Equations (21)–(24), where *TP*, *TN*, *FP*, and *FN* denote the numbers of true-positive, true-negative, false-positive, and false-negative patches, respectively. AP summarizes the precision–recall relationship across classification thresholds by weighting precision according to increases in recall. Unlike the metrics reported at the fixed threshold of 0.5, AP evaluates the ranking of predicted patch probabilities across different operating points.

Accuracy is the ratio of correctly predicted instances to all instances:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{21}$$

Precision is the proportion of correctly predicted positive instances among all instances predicted as positive:

$$Precision = \frac{TP}{TP + FP} \tag{22}$$

Recall, also known as sensitivity, is the proportion of correctly predicted positive instances among all actual positive instances:

$$Recall = \frac{TP}{TP + FN} \tag{23}$$

The *F*1-score is the harmonic mean of precision and recall, which is useful for imbalanced datasets:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{24}$$

The metric values of CrackClsNet on the test set are summarized in Table 1.

Table 1. Patch-level classification performance of CrackClsNet on the test set.

Accuracy	Precision	Recall	F1-Score	AP
0.9926	0.9219	0.8701	0.8952	0.9420

As summarized in Table 1, CrackClsNet achieved an accuracy, precision, recall, F1-score, and AP of 0.9926, 0.9219, 0.8701, 0.8952, and 0.9420 on Dataset C, respectively. Because background patches dominate the test set, accuracy alone provides limited information about crack-screening performance. The precision indicates that most positively classified patches contained annotated crack pixels, whereas the recall shows that some crack-containing patches remained undetected. AP provides a complementary assessment of the precision–recall relationship across classification thresholds.

Figure 14 presents representative patch-screening results of CrackClsNet under different crack appearances and background conditions. The red rectangles indicate patches classified as crack-containing. In Figure 14a,b, the selected patches generally follow the visible trajectories of narrow cracks, including relatively faint segments and changes in direction. Figure 14c,d further illustrate candidate region identification on surfaces containing pores and irregular textures, while Figure 14e shows the retention of spatially separated fine-crack segments. In Figure 14f, patches along the diagonal crack are selected, whereas the crack-like background feature highlighted by the green rectangle is excluded. This example suggests that the classifier can distinguish some background patterns from

actual cracks under the illustrated conditions. Overall, these qualitative results illustrate useful localization of candidate crack regions while leaving much of the background unselected. Such behavior is consistent with the intended role of CrackClsNet as a screening stage that directs subsequent pixel-level segmentation toward potentially informative regions. However, the selected boxes indicate patch-level predictions and do not establish complete coverage of the underlying crack pixels, which is evaluated separately in Table 2.

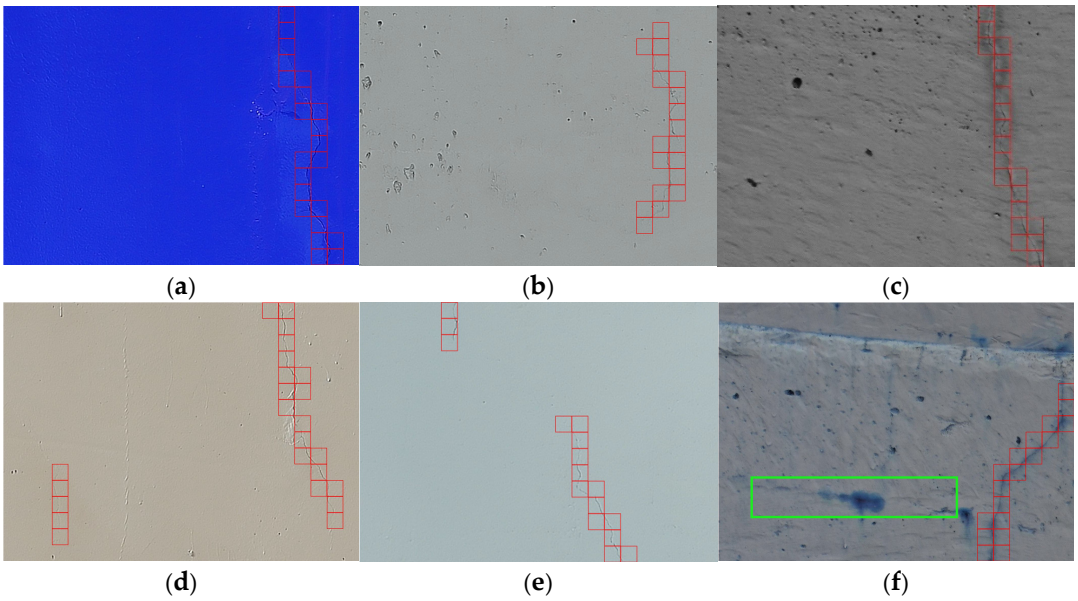


Figure 14. Representative patch-screening results of CrackClsNet: (**a**) a fine crack with a changing trajectory; (**b**) a narrow irregular crack; (**c**) a crack on a porous surface; (**d**) multiple fine-crack segments; (**e**) spatially separated fine-crack segments; (**f**) a diagonal crack near a crack-like background feature. Red rectangles indicate patches classified as crack-containing. The green rectangle in (**f**) highlights a non-crack background feature that was not selected by the classifier.

Table 2. Crack-pixel coverage of candidate regions selected by CrackClsNet with and without neighborhood expansion.

Patch Selection Strategy	Pixel Coverage Rate
Predicted crack patches only	0.9197
Predicted crack patches with a 3 × 3 neighborhood	0.9565

The observed classification performance is broadly consistent with the design motivation of the multiscale branches and CBAM. Each ConvBlock uses four parallel branches—a 1 × 1 convolution and three 3 × 3 convolutions with dilation rates of 1, 2, and 3—to extract features at different receptive field scales. This design may help the classifier identify cracks with varying widths and irregular geometries. The concatenated features are subsequently refined by CBAM, which recalibrates channel and spatial responses and may help reduce interference from background textures. The green-boxed non-crack region in Figure 14f provides a qualitative example of successful background rejection, although this result cannot be attributed specifically to the attention mechanism. Finally, the six-stage hierarchical downsampling reduces the spatial resolution by a factor of 64 and progressively aggregates local features into patch-level representations, potentially supporting the recognition of subtle crack patterns.

False-positive and false-negative predictions were examined at the fixed classification threshold of 0.5 to further characterize screening errors. Across Dataset C, CrackClsNet produced 254 false-positive patches among 91,591 ground-truth negative patches and

missed 448 of the 3449 ground-truth positive patches. These counts correspond to a patch-level false-positive and false-negative rates of 0.2773% and 12.9893%, respectively. The precision–recall curve in Figure 13b, calculated from all patches in the mixed test set, provides a complementary assessment across classification thresholds, with an AP of 0.9420.

A separate analysis considered the 100 crack-free images in Dataset C. An image-level screening false alarm was defined as the presence of at least one positively classified patch. These images contained 35,200 background patches, of which 67 were incorrectly classified as crack-containing, yielding a patch-level false-positive rate of 0.1903%. Eleven images contained at least one false-positive patch, corresponding to an image-level screening false-positive rate of 11%. Selected false-positive examples are presented in Figure 15. The predictions illustrate remaining responses near surface boundaries and textural irregularities. These examples complement the successful background rejection example in Figure 14f and provide a more balanced illustration of screening behavior.



Figure 15. Selected false-positive screening examples from the crack-free subset of Dataset C. Red rectangles indicate crack-free patches incorrectly classified as crack-containing by CrackClsNet

To evaluate how neighborhood retention affects the crack information available for subsequent segmentation, an additional pixel coverage experiment was conducted on Dataset C. Two settings were compared using the same trained CrackClsNet and a classification threshold of 0.5. The first retained only the 64 × 64-pixel patches predicted to contain cracks, referred to as S-patches. The second retained each S-patch together with its eight immediate neighbors, forming a 3 × 3 neighborhood. The classifier predictions were identical in both settings, allowing the effect of neighborhood retention to be assessed independently of changes to the classifier.

The pixel coverage rate was defined as the number of ground-truth crack pixels within the union of the retained regions divided by the total number of ground-truth crack pixels across the evaluated images. Pixel counts were accumulated over all images before calculating the ratio, and pixels within overlapping neighborhoods were counted only once. As shown in Table 2, retaining only the predicted crack patches achieved a pixel coverage rate of 0.9197. Including their 3 × 3 neighborhoods increased coverage to 0.9565, an improvement of approximately 3.7 percentage points. Specifically, the number of retained crack pixels increased from 276,272 to 287,345 out of 300,407 annotated crack pixels. Neighborhood expansion therefore recovered 11,073 additional crack pixels, representing

approximately 45.9% of those excluded by the initial screening. The proportion of crack pixels outside the retained regions decreased from 8.03% to 4.35%. These results show that neighboring regions contain crack pixels that would otherwise be excluded during initial screening. For spatially continuous cracks, a segment missed by the classifier can still be retained when it lies adjacent to a positively classified patch. The findings support the neighborhood retention strategy introduced in Section 2.2.1 as a means of reducing crack-information loss before segmentation.

3.3. Training of CrackSegNet

The crack-containing images from Dataset A, together with 130 crack-containing images from Dataset B, were combined to form the training set, Dataset D. The remaining 30 crack-containing images from Dataset B and 70 crack-containing images from Dataset C were used as the validation set, Dataset E, comprising 100 images in total. The remaining 100 crack-containing images from Dataset C were used as the test set, Dataset F. CrackSegNet was trained on Dataset D, validated and tuned on Dataset E during training, and finally evaluated on Dataset F. During CrackSegNet training, contextual patches centered on crack-containing regions were extracted from the training images. Randomly selected crack-free patches were also included so that the segmentation network could learn to suppress background responses. This negative sampling is intended to reduce false-positive masks when CrackClsNet forwards a non-crack patch to the second stage.

CrackSegNet was trained for 200 epochs using the AdamW optimizer with an initial learning rate of 0.0001. Because CrackSegNet performs pixel-level binary segmentation, accuracy, precision, recall, F1-score, crack-class Intersection over Union (Crack IoU), and mean Intersection over Union (mIoU) were used to evaluate its performance. The quantitative results on the test set are summarized in Table 3.

Table 3. Pixel-level segmentation performance of CrackSegNet on the test set.

Accuracy	Precision	Recall	F1-Score	Crack IoU	mIoU
0.9941	0.5264	0.6573	0.5846	0.4130	0.7036

The IoU is calculated as follows:

$$IoU_{crack} = \frac{TP}{TP + FP + FN} \tag{25}$$

$$IoU_{background} = \frac{TN}{TN + FP + FN} \tag{26}$$

$$\text{mIoU} = \frac{IoU_{crack} + IoU_{background}}{2} \tag{27}$$

As shown in Table 3, CrackSegNet achieves an overall pixel accuracy of 0.9941. Because crack pixels occupy only a small proportion of the evaluated regions, this value is strongly influenced by correct background predictions and provides limited information about foreground segmentation quality. Therefore, precision, recall, F1-score, and Crack IoU are emphasized in assessing the identification and delineation of fine cracks, while retaining mIoU as a complementary class-averaged measure.

CrackSegNet achieves a precision of 0.5264 and a recall of 0.6573, corresponding to an F1-score of 0.5846. These results suggest useful foreground recovery, with the model identifying a substantial proportion of the annotated crack pixels. Such recovery is relevant to bridge inspection, where retaining potential crack regions is important for subsequent assessment. The Crack IoU of 0.4130 indicates spatial correspondence between the predicted crack regions and the ground-truth annotations. These results are encouraging

given that many cracks in the dataset are represented by only one to a few pixels. For such narrow structures, small boundary displacements or changes in predicted width can noticeably affect foreground overlap.

The representative results in Figure 16 further illustrate the model's ability to recover crack structures. The predicted masks generally follow the principal trajectories of straight, curved, and irregular cracks, including several long and tortuous paths. Spatially coherent predictions are also observed across adjoining patch regions in the illustrated examples. Some predicted masks appear wider than the annotations, particularly in regions with weak or ambiguous boundaries, which may contribute to false-positive predictions. Nevertheless, the examples suggest that the model can retain the main morphology of fine cracks under the conditions represented in the figure.

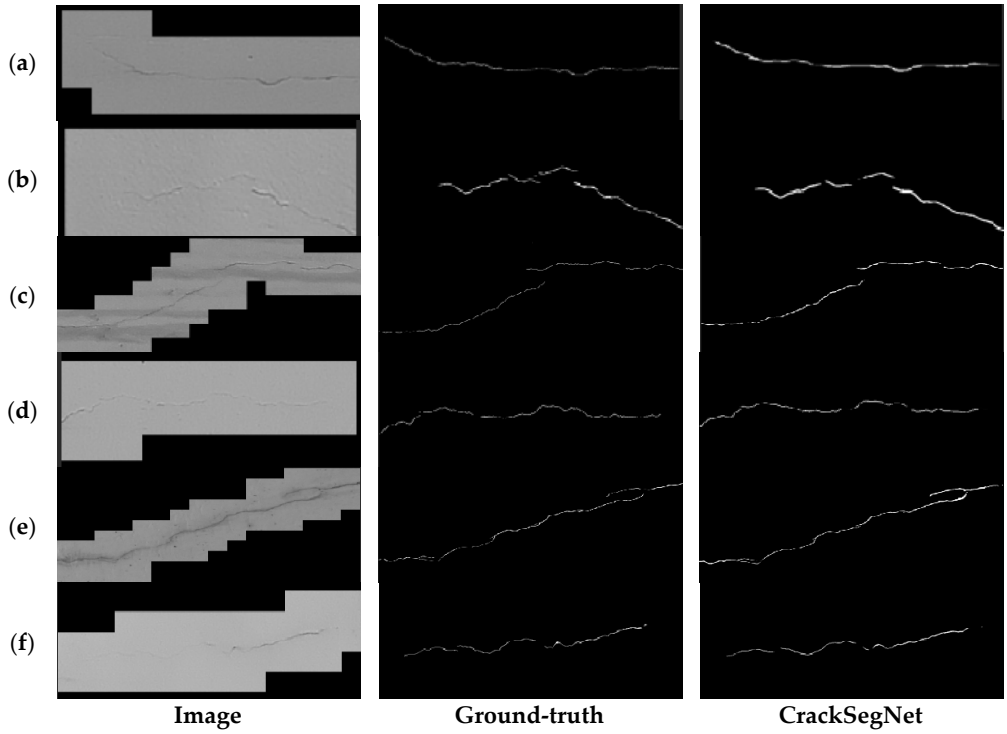


Figure 16. Representative segmentation results of CrackSegNet: (**a**) a narrow-curved crack; (**b**) a fine irregular crack; (**c**) an irregular crack; (**d**) a low-contrast, approximately vertical crack; (**e**) a tortuous crack with local branching; (**f**) a faint narrow crack. For each sample, the columns show the selected input region, ground-truth mask, and CrackSegNet prediction, respectively.

Several architectural choices may contribute to these results. The multiscale convolutional branches capture features at different receptive-field scales, while deformable convolutions provide flexible sampling that may accommodate variations in crack geometry. CBAM recalibrates channel and spatial responses and is intended to emphasize informative crack features. Graph-based reasoning enables contextual information exchange among adjacent regions, potentially assisting the interpretation of crack structures across patch boundaries. The pixel-shuffle-based decoder progressively restores spatial resolution, while encoder–decoder skip connections provide intermediate spatial features during reconstruction. The contributions of these components are examined further in the ablation experiments in Section 3.4.

The quantitative results and representative visual examples suggest that CrackSegNet provides useful foreground recovery and spatially coherent fine-crack predictions on the evaluated dataset. These findings support its suitability as the second-stage segmentation network in the proposed framework.

3.4. Ablation Experiments on CrackSegNet Module Structure

To examine the contributions of the main architectural components, six ablation configurations of CrackSegNet were evaluated, as summarized in Table 4. Deformable convolutions were replaced with conventional convolutions while preserving the dilation rates, or dilation was disabled while retaining deformable sampling. CBAM and the GNN branch were separately removed in two further configurations. Two additional variants examined neighboring context and cross-patch feature interaction. Without neighboring patches, CrackSegNet processed independent 64 × 64 patches, retained graph reasoning within each patch, and reconstructed predictions directly without overlap fusion. This configuration evaluates the neighboring-patch processing scheme as a whole. Without feature gathering, the 192 × 192 contextual inputs and overlapping prediction fusion were retained, but encoder features from different contextual patches were no longer aggregated into shared global nodes or gathered back from a shared graph representation. Instead, graph reasoning was performed independently within each contextual patch.

Table 4. Ablation study of the main architectural components of CrackSegNet.

Configuration	Accuracy	Precision	Recall	F1-Score	Crack IoU	mIoU
Without adjacent patches	**0.9990**	0.5084	0.6235	0.5601	0.3890	0.6940
Without deformable convolution	0.9938	0.4991	0.6404	0.5610	0.3898	0.6918
Without dilatation	0.9940	0.5151	0.6247	0.5646	0.3934	0.6937
Without CBAM	0.9941	0.5156	0.6381	0.5703	0.3989	0.6965
Without GNN	0.9942	0.5230	0.6367	0.5743	0.4028	0.6985
Without feature gather	**0.9990**	0.5184	0.6465	0.5754	0.4039	0.7015
Full CrackSegNet	0.9941	**0.5264**	**0.6573**	**0.5846**	**0.4130**	**0.7036**

Note: Bold values indicate the highest value in each metric column; tied highest values are also shown in bold.

The complete CrackSegNet achieved the highest precision, recall, F1-score, Crack IoU, and mIoU among the configurations in Table 4. Its F1-score, Crack IoU, and mIoU reached 0.5846, 0.4130, and 0.7036, respectively. Removing adjacent patches produced the largest decreases in F1-score and Crack IoU, which fell to 0.5601 and 0.3890, while recall decreased from 0.6573 to 0.6235. These results support retaining neighboring regions and their associated overlap processing before reconstructing the segmentation mask. Some ablated configurations achieved higher overall pixel accuracy, but their lower foreground segmentation scores illustrate the need to interpret accuracy alongside F1-score and Crack IoU when crack pixels are sparse.

Replacing deformable convolutions with conventional dilated convolutions reduced the F1-score to 0.5610 and Crack IoU to 0.3898. Disabling dilation while retaining deformable convolutions yielded an F1-score of 0.5646 and Crack IoU of 0.3934, with recall decreasing to 0.6247. Both configurations performed below the complete model, supporting the combined use of multiscale receptive fields and adaptive sampling. Dilation broadens the spatial context available for feature extraction, whereas deformable sampling allows the convolutional receptive field to adapt to irregular crack geometries.

Removing CBAM reduced the F1-score and Crack IoU to 0.5703 and 0.3989, respectively. Removing the GNN yielded corresponding values of 0.5743 and 0.4028. These decreases are consistent with the intended roles of attention-based feature selection and graph-based contextual reasoning. The configuration without feature gathering achieved an F1-score and Crack IoU of 0.5754 and 0.4039, respectively, remaining below the complete model by 0.0092 and 0.0091. Because this configuration retained neighboring image content, within-patch GNN processing, and output overlap fusion, the comparison

supports the value of exchanging encoder features across patches in addition to combining their final predictions.

Table 5 further examines the effect of neighborhood size. The configurations without adjacent patches and with 3 × 3, 5 × 5, and 7 × 7 neighborhoods correspond to input patches of 64 × 64, 192 × 192, 320 × 320, and 448 × 448 pixels, respectively; each neighborhood includes the central patch. Memory consumption and inference speed were evaluated alongside segmentation quality. Introducing a 3 × 3 neighborhood improved foreground segmentation relative to the configuration without adjacent patches, but increased memory consumption from 0.15 to 2.36 GB and reduced inference speed from 23.48 to 10.74 FPS. Neighborhood retention therefore entails a measurable computational cost.

Table 5. Effect of neighborhood size on the segmentation performance and computational efficiency of CrackSegNet.

Neighborhood Size	Accuracy	Precision	Recall	F1-Score	Crack IoU	mIoU	Memory (GB)	FPS
1 × 1 (without adjacent patches)	**0.9990**	0.5084	0.6235	0.5601	0.3890	0.6940	**0.15**	**23.48**
3 × 3 (proposed)	0.9941	0.5264	0.6573	0.5846	0.4130	0.7036	2.36	10.74
5 × 5	0.9942	0.5264	**0.6586**	0.5851	0.4135	0.7038	4.92	5.12
7 × 7	0.9941	**0.5272**	0.6579	**0.5853**	**0.4138**	**0.7039**	7.94	3.19

Note: Bold values indicate the highest segmentation scores and FPS, or the lowest memory consumption, in the corresponding columns.

Further enlargement provided only small numerical improvements. Expanding the neighborhood from 3 × 3 to 5 × 5 increased the F1-score and Crack IoU from 0.5846 to 0.5851 and 0.4130 to 0.4135, respectively, while memory consumption increased to 4.92 GB and inference speed decreased to 5.12 FPS. The 7 × 7 configuration achieved an F1-score and Crack IoU of 0.5853 and 0.4138, respectively, but required 7.94 GB and operated at 3.19 FPS. Recall reached its highest value of 0.6586 with the 5 × 5 neighborhood and decreased slightly to 0.6579 with 7 × 7, indicating that enlarging the neighborhood did not improve every metric monotonically.

Under the evaluated conditions, the 3 × 3 neighborhood preserved most of the segmentation quality obtained with larger neighborhoods while requiring substantially less memory and enabling faster inference. These results support its use as the default configuration of CrackSegNet, balancing the benefit of neighboring crack information against the additional cost of processing larger contextual regions.

3.5. End-to-End Performance Comparison and Visualization

The end-to-end pipeline first applies CrackClsNet to identify candidate regions, extracts the corresponding contextual patches, processes them with CrackSegNet, and fuses the overlapping predictions into a full-resolution mask. The testing was performed on Dataset F and Table 6 reports the resulting end-to-end performance.

Table 6. End-to-end segmentation performance of the proposed framework and baseline methods on Dataset F.

Method	Accuracy	Precision	Recall	F1-Score	Crack IoU	mIoU
FCN	0.9986	0.3895	0.5933	0.4702	0.3074	0.6530
U-Net	0.9990	0.5474	0.4400	0.4878	0.3226	0.6608
DeepLabV3+	0.9990	0.5187	0.5793	0.5474	0.3768	0.6879
CrackSegFormer	0.9986	0.3748	0.5369	0.4413	0.2832	0.6408
DeepCrack	0.9990	0.4927	0.5966	0.5397	0.3696	0.6843
CrackSeg-GWD	0.9987	0.4064	0.4865	0.4429	0.2844	0.6416
Proposed method	**0.9994**	**0.5540**	**0.6143**	**0.5826**	**0.4110**	**0.7052**

Note: Bold values indicate the highest value in each evaluation metric column.

Six segmentation networks were used as baselines: FCN-8s, U-Net, DeepLabV3+ with a ResNet-50 backbone, DeepCrack [51], CrackSegFormer [52], and CrackSeg-GWD [53]. FCN, U-Net, and DeepLabV3+ represent established semantic segmentation paradigms, whereas DeepCrack, CrackSegFormer, and CrackSeg-GWD represent crack-specific convolutional, Transformer-based, and lightweight approaches, respectively.

All six baseline networks were trained for 200 epochs using Dataset D, validated on Dataset E, and evaluated on Dataset F. The input images had a resolution of 1024 × 1408 pixels and were used without further resizing or additional data augmentation. Each baseline was initialized without pretrained weights and trained using the AdamW optimizer. The segmentation objective was identical to that described in Section 2.2.4, combining weighted BCE and Tversky loss with the same loss coefficients. For each baseline, the checkpoint achieving the highest Crack IoU on Dataset E was selected for final evaluation. A fixed threshold of 0.5 was used to obtain binary segmentation masks, and Dataset F was not used for checkpoint or threshold selection.

The baseline networks performed full-image segmentation without overlapping tiled inference or prediction fusion. The proposed framework instead applied CrackClsNet screening, contextual patch extraction, CrackSegNet segmentation, and overlapping prediction fusion, as described in Section 2. These differences form part of the methods being compared. Accuracy, precision, recall, F1-score, Crack IoU, and mIoU were calculated on the same test set. The quantitative results are summarized in Table 6, and representative predictions are presented in Figure 17, where yellow and red circles indicate false-positive and false-negative regions, respectively.

As shown in Table 6, all evaluated methods achieve high pixel-level accuracy, ranging from 0.9986 to 0.9994. Because background pixels account for most of the image area, accuracy alone does not adequately distinguish their ability to segment sparse cracks. The comparison therefore places greater emphasis on foreground precision, recall, F1-score, and Crack IoU. Although mIoU summarizes performance across both classes, its relatively high value partly reflects the substantially higher background IoU. Consequently, it is reported as a complementary measure rather than the principal basis for assessing fine-crack segmentation quality.

The proposed end-to-end framework achieved accuracy, precision, recall, F1-score, Crack IoU, and mIoU of 0.9994, 0.5540, 0.6143, 0.5826, 0.4110, and 0.7052, respectively, obtaining the highest values for all six metrics among the evaluated methods. Relative to DeepLabV3+, the strongest baseline in terms of F1-score, Crack IoU, and mIoU, the proposed framework improved recall, F1-score, and Crack IoU by 3.50, 3.52, and 3.42 percentage points. These improvements indicate a better balance between identifying crack pixels and maintaining agreement with the annotated crack regions.

Among the baseline methods, U-Net achieved the highest precision of 0.5474 but a recall of 0.4400, indicating a comparatively conservative prediction pattern. The proposed framework achieved both higher precision (0.5540) and higher recall (0.6143), resulting in an F1-score of 0.5826, compared with 0.4878 for U-Net. FCN and DeepCrack achieved recalls of 0.5933 and 0.5966, respectively, demonstrating their ability to recover a substantial proportion of crack pixels. However, their lower precisions of 0.3895 and 0.4927 indicate greater difficulty in separating crack responses from background interference.

DeepLabV3+ provided the strongest overall baseline performance, achieving an F1-score, Crack IoU, and mIoU of 0.5474, 0.3768, and 0.6879, respectively. CrackSegFormer and CrackSeg-GWD achieved F1-scores of 0.4413 and 0.4429, respectively, with corresponding Crack IoU values of 0.2832 and 0.2844. Their results show that the Transformer-based and lightweight architectures evaluated here remain challenged by the fine crack structures and complex surface textures in this dataset. The proposed method achieved

higher foreground overlap while retaining the highest recall among the compared methods.

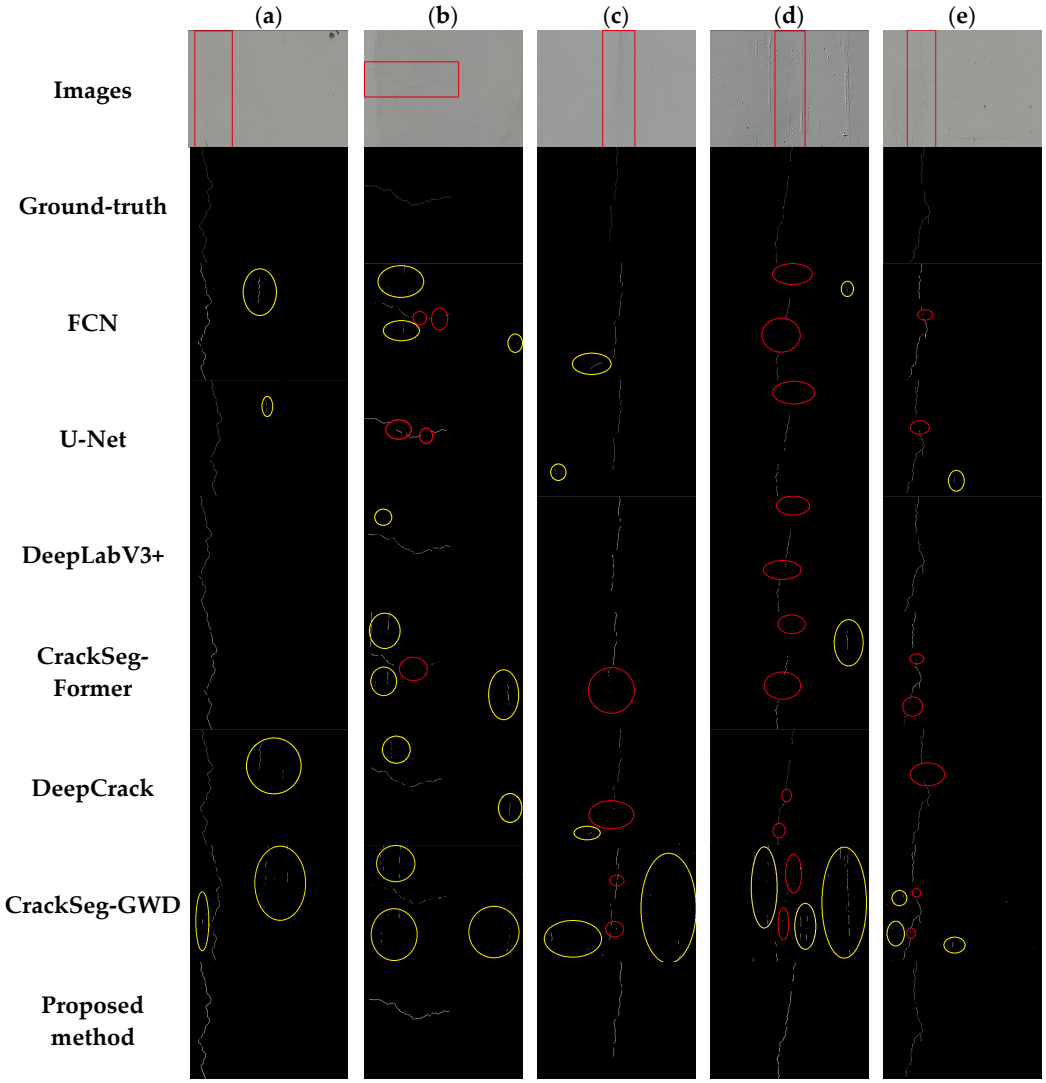


Figure 17. Qualitative comparison of the proposed framework with baseline segmentation methods: (**a**) a predominantly vertical curved crack; (**b**) a curved horizontal crack; (**c**) a fine vertical crack; (**d**) a vertical crack on a strongly textured surface; (**e**) an irregular fine crack. Rows show the input images, ground-truth masks, and predictions from FCN, U-Net, DeepLabV3+, CrackSegFormer, DeepCrack, CrackSeg-GWD, and the proposed method. Red rectangles in the input images highlight the crack regions of interest. Yellow and red circles indicate selected false-positive and false-negative regions, respectively.

The qualitative comparisons in Figure 17 are consistent with these quantitative differences. FCN detects the principal crack trajectories but also produces responses to crack-like background structures, particularly in samples (a) and (b). U-Net generates relatively clean masks, although portions of the faint cracks remain incompletely recovered. DeepLabV3+ captures the main crack structures effectively, but discontinuities remain in the challenging textured region of sample (d). CrackSegFormer and DeepCrack also show missed crack segments or additional background responses in several examples, whereas CrackSeg-GWD produces more scattered responses around surface textures.

In comparison, the proposed method generally follows the annotated crack trajectories while suppressing several background responses visible in the baseline predictions. In sample (b), it preserves the curved horizontal crack with fewer extraneous predictions

around the target structure. In sample (d), where surface texture interferes with crack identification, it recovers the main vertical crack with less background interference than several competing methods. The remaining examples similarly show effective recovery of long, narrow crack structures. These representative results support the quantitative advantage in F1-score and foreground overlap, although some extremely thin segments remain challenging.

The observed performance is consistent with the design of the proposed framework. CrackClsNet identifies candidate regions, allowing the segmentation stage to concentrate on selected areas. Retaining neighboring patches supplies context around grid boundaries and can preserve crack regions missed during initial screening. Within CrackSegNet, multiscale convolutions, deformable convolutions, attention mechanisms, graph-based contextual reasoning, and pixel rearrangement operations combine adaptive feature extraction with spatial context modeling. Overlapping patch predictions are fused to reconstruct the full-image segmentation map. These results indicate improved foreground segmentation relative to the evaluated baselines on the present dataset.

The present results were obtained from a dataset collected from 13 bridges, and caution is required when extending the findings beyond the conditions represented in the evaluation data. Feng et al. [54], in comparing machine learning methods with a physical model for shallow-landslide risk modeling, discussed how data accuracy, parameter uncertainty, and model limitations complicate model selection for other regions. Although their study addresses a different application, this perspective is relevant to interpreting the generalization of the proposed framework. In bridge crack segmentation, changes in structural materials, surface textures, illumination, imaging distance, or camera settings may alter the appearance of cracks and crack-like backgrounds. These changes could affect both candidate screening by CrackClsNet and pixel-level segmentation by CrackSegNet.

3.6. Efficiency Testing and Comparison

To evaluate how computational requirements change with input image dimensions, an efficiency benchmark was conducted at five resolutions: 256 × 352, 512 × 704, 1024 × 1408, 2048 × 2816, and 4096 × 5632 pixels. For the proposed method, Dataset F images were resized to these dimensions. The same six baseline networks used in Section 3.5 were included, and each method retained the same trained weights across resolutions without additional training. Testing was performed using PyTorch 2.5.1, with the networks in evaluation mode and gradient computation disabled. After preliminary inference runs, repeated executions were performed at each resolution.

Image preprocessing was completed before timing and was excluded from the reported processing time. Timing began when the input tensor was prepared and ready to be fed into the network, and ended after network inference and output reconstruction were completed. For the proposed framework, the measured interval therefore included network inference and the fusion and reconstruction of patch predictions into the full-resolution segmentation map. FPS was calculated by dividing the number of processed images by the elapsed wall-clock time. GPU memory was measured using PyTorch's maximum allocated-memory statistic, torch.cuda.max_memory_allocated(), which records the peak allocation for GPU tensors up to the measurement point rather than reserved allocator memory or total device memory consumption.

This benchmark assesses computational efficiency and memory requirements only; segmentation accuracy at the resized resolutions was not evaluated. Image enlargement increased the input dimensions without introducing additional scene detail. The results are presented in Tables 7 and 8 and Figure 18.

Table 7. Parameter counts and inference speed of the compared methods at different input resolutions.

Method	Parameters	FPS				
		256 × 352	512 × 704	1024 × 1408	2048 × 2816	4096 × 5632
FCN	134.26M	85.62	38.89	13.28	3.77	0.12
U-Net	31.39M	105.92	25.72	6.89	0.90	<0.01
DeepLabV3+	42.39M	223.37	89.28	20.59	4.96	1.46
CrackSegFormer	3.71M	205.09	**150.55**	19.25	0.84	<0.01
DeepCrack	30.91M	32.43	8.19	0.39	<0.01	<0.01
CrackSeg-GWD	**0.41M**	**346.51**	135.71	**29.98**	6.15	2.06
Proposed method	0.49M + 3.92M	46.35	18.42	10.74	**6.46**	**4.10**

Note: Bold values indicate the lowest parameter count or the highest FPS in the corresponding columns.

Table 8. Parameter counts and memory consumption of the compared methods at different input resolutions.

Method	Parameters	Memory Consumption (GB)				
		256 × 352	512 × 704	1024 × 1408	2048 × 2816	4096 × 5632
FCN	134.26M	0.89	0.92	1.44	3.75	12.52
U-Net	31.39M	0.37	1.11	4.06	15.87	27.49
DeepLabV3+	42.39M	0.20	0.28	**0.64**	**2.05**	7.72
CrackSegFormer	3.71M	0.08	0.24	1.01	15.30	>120 GB
DeepCrack	30.91M	0.92	3.33	12.96	29.44	46.35
CrackSeg-GWD	0.41M	**0.05**	**0.18**	0.70	2.79	11.17
Proposed method	0.49M + 3.92M	0.56	1.12	2.36	4.17	**6.94**

Note: Bold values indicate the lowest memory consumption at each input resolution.

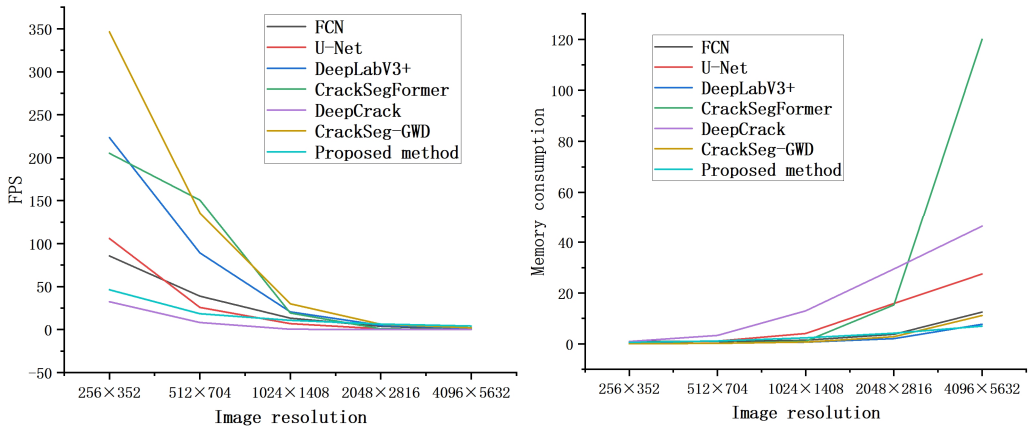


Figure 18. Computational efficiency of the compared methods at different input resolutions: (**a**) inference speed (FPS); (**b**) memory consumption (GB).

As shown in Table 7, the proposed framework consists of the 0.49M-parameter CrackClsNet and the 3.92M-parameter CrackSegNet, resulting in approximately 4.41M parameters in total. This parameter count is substantially lower than those of FCN, U-Net, DeepLabV3+, and DeepCrack, which contain 134.26M, 31.39M, 42.39M, and 30.91M parameters, respectively. Although CrackSegFormer and CrackSeg-GWD contain fewer parameters, the proposed method achieves higher processing speeds at the two largest input resolutions. These results indicate that parameter count alone does not determine computational efficiency when processing high-resolution images.

At relatively low resolutions, several baseline networks achieve higher processing speeds than the proposed framework. The proposed method processes 46.35 FPS at 256 × 352 pixels and 18.42 FPS at 512 × 704 pixels, whereas CrackSeg-GWD achieves 346.51 and 135.71 FPS, respectively. At 1024 × 1408 pixels, the proposed method achieves 10.74 FPS, exceeding the 6.89 FPS of U-Net and 0.39 FPS of DeepCrack, but remaining slower than FCN, DeepLabV3+, CrackSegFormer, and CrackSeg-GWD. Thus, its efficiency advantage is not uniform across all input resolutions.

When the resolution increases to 2048 × 2816 pixels, the proposed method achieves the highest processing speed of 6.46 FPS, approximately 5.0% higher than the 6.15 FPS of the second-fastest method, CrackSeg-GWD. At 4096 × 5632 pixels, the proposed method maintains 4.10 FPS, approximately twice the 2.06 FPS achieved by CrackSeg-GWD. DeepLabV3+ and FCN achieve 1.46 and 0.12 FPS, respectively, whereas U-Net, CrackSegFormer, and DeepCrack each operate below 0.01 FPS. The increasing relative advantage at these resolutions demonstrates the favorable throughput scaling of the proposed framework within the evaluated range.

The memory consumption results in Table 8 show a similar dependence on input resolution. At lower resolutions, several baseline networks require less memory than the proposed framework. At 2048 × 2816 pixels, the proposed method requires 4.17 GB, which remains higher than DeepLabV3+ (2.05 GB), CrackSeg-GWD (2.79 GB), and FCN (3.75 GB). At 4096 × 5632 pixels, however, the proposed method has the lowest reported memory consumption of 6.94 GB. Compared with DeepLabV3+, the most memory-efficient baseline at this resolution, its memory requirement is reduced by approximately 10.1%. The reduction reaches approximately 74.8% relative to U-Net, which requires 27.49 GB. DeepCrack requires 46.35 GB, while the reported requirement of CrackSegFormer exceeds 120 GB.

These efficiency trends are consistent with the dual-stage design of the proposed framework. CrackClsNet identifies candidate regions, and CrackSegNet performs detailed segmentation on the retained contextual patches. This strategy concentrates the more computationally demanding segmentation operations on selected regions rather than constructing dense segmentation features over the entire image. Processing overlapping patches and performing graph-based reasoning also introduce computational costs, which may partly explain the less favorable results at smaller input resolutions. As image resolution increases, the benefit of restricting detailed segmentation to selected regions becomes more apparent under the reported benchmark conditions.

The trends in Figure 18 further illustrate this difference in scalability. When the resolution increases from 1024 × 1408 to 4096 × 5632 pixels, the number of input pixels increases 16-fold, whereas the reported memory consumption of the proposed method increases from 2.36 to 6.94 GB, or approximately 2.94 times. Its processing speed decreases from 10.74 to 4.10 FPS, while the baseline methods exhibit larger proportional reductions over the same range. These observations show that the proposed framework handles the increase in input size more efficiently within the tested settings.

Under the reported benchmark conditions, the proposed method recorded 4.10 FPS and a peak allocated GPU tensor-memory reading of 6.94 GB at 4096 × 5632 pixels. These measurements are intended as reference values for the evaluated implementations. Actual processing speeds and memory requirements may vary with hardware configurations, software environments, implementation settings, and input-dependent workloads, including the number of retained candidate patches. Nevertheless, the reported results provide a preliminary indication of the computational requirements and relative efficiency of the compared methods under the present experimental conditions. They do not establish segmentation accuracy or scale robustness at the resized resolutions.

4. Conclusions

This study proposed an efficient dual-stage deep learning framework for fine-crack segmentation in high-resolution bridge images. The framework first partitions the original image into regular grids and employs the lightweight CrackClsNet to identify candidate crack-containing regions. Only the positively classified patches and their immediate neighbors are subsequently processed by CrackSegNet for pixel-level segmentation. This design concentrates computational resources on informative regions while retaining surrounding context to reduce the influence of patch boundaries and upstream classification errors. Within CrackSegNet, multi-scale convolutions, deformable convolutions, attention mechanisms, graph-based contextual reasoning, and pixel-unshuffle and pixel-shuffle operations are integrated to model thin, irregular cracks and their spatial context. The rearrangement operations are reversible permutations of tensor elements; this property does not imply lossless retention of input or task-relevant information by the complete learned network, which also includes channel compression, convolutions, and nonlinear transformations.

CrackClsNet achieved an accuracy, precision, recall, F1-score, and AP of 0.9926, 0.9219, 0.8701, 0.8952, and 0.9420, respectively. Retaining the 3 × 3 neighborhoods of positively classified patches increased ground-truth crack-pixel coverage from 0.9197 to 0.9565, supporting the use of neighboring regions to reduce crack information loss before segmentation. In the end-to-end comparison, the proposed framework achieved an accuracy, precision, recall, F1-score, Crack IoU, and mIoU of 0.9994, 0.5540, 0.6143, 0.5826, 0.4110, and 0.7052, respectively, obtaining the highest values for all six metrics among the evaluated methods. Relative to DeepLabV3+, the absolute improvements in recall, F1-score, Crack IoU, and mIoU were 0.0350, 0.0352, 0.0342, and 0.0173, respectively. The representative examples also illustrated recovery of several thin and low-contrast crack trajectories.

The proposed framework also demonstrated favorable computational scalability. CrackClsNet and CrackSegNet contain approximately 0.49M and 3.92M parameters, respectively, resulting in a total parameter count of approximately 4.41M. Although several baseline networks remained faster or required less memory at lower resolutions, the advantages of the proposed method became more evident as the input resolution increased. At 2048 × 2816 pixels, it achieved the highest processing speed of 6.46 FPS with a memory consumption of 4.17 GB. At the highest tested resolution of 4096 × 5632 pixels, it maintained 4.10 FPS with a reported memory consumption of 6.94 GB, providing the highest processing speed and lowest memory requirement among the evaluated methods at this resolution. Its processing speed was approximately twice that of CrackSeg-GWD, while its memory consumption was approximately 10.1% lower than that of DeepLabV3+, the most memory-efficient baseline at this resolution.

The quantitative and qualitative results indicate a favorable balance between foreground recovery, segmentation quality, and computational cost on the evaluated dataset. The ablation experiments support the contributions of neighboring context, deformable sampling, dilation, attention, and graph-based feature interaction under the tested conditions. Enlarging the neighborhood beyond 3 × 3 provided only small additional improvements while increasing memory consumption and reducing processing speed. These findings support the selected configuration and provide a basis for further evaluation of the framework for bridge inspection.

The present evaluation used random image-level partitions. Although the final end-to-end test images were excluded from the training and validation sets of both network stages, the partitioning did not enforce separation by bridge, and independence at the flight-session or imaged-surface-area level was not established. Visual similarity and spatial correlation across subsets may therefore influence the reported performance. Bridge-

disjoint evaluation, per-bridge performance summaries, and statistical uncertainty estimates were not included, so generalization to previously unseen bridges remains to be established. Future work will prioritize leave-one-bridge-out or grouped cross-validation, with both network stages trained and selected without access to the held-out bridges, together with per-bridge results and uncertainty estimates. External dataset and field evaluations will further examine performance under different acquisition conditions. In addition, the multi-resolution benchmark assessed computational efficiency and memory requirements without evaluating segmentation accuracy across resolutions.

Author Contributions: Conceptualization, A.Y. and D.Z.; methodology, A.Y. and W.M.; software, A.Y.; data curation, D.Z.; writing—original draft preparation, A.Y.; writing—review and editing, D.Z. and W.M.; project administration, W.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Our dataset is openly available on Figshare at https://doi.org/10.6084/m9.figshare.33000635.

Acknowledgments: During the preparation of this manuscript, the authors used ChatGPT (version 5.5) for the purposes of language polishing and editing. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Abdallah Abdelrahman, M.; Atadero Rebecca, A.; Ozbek Mehmet, E. A State-of-the-Art Review of Bridge Inspection Planning: Current Situation and Future Needs. *J. Bridge Eng.* **2022**, *27*, 03121001.
2. Phares Brent, M.; Washer Glenn, A.; Rolander Dennis, D.; Graybeal, B.A. Routine Highway Bridge Inspection Condition Documentation Accuracy and Reliability. *J. Bridge Eng.* **2004**, *9*, 403–413.
3. Nair, A.; Cai, C.S. Acoustic emission monitoring of bridges: Review and case studies. *Eng. Struct.* **2010**, *32*, 1704–1714.
4. McCann, D.M.; Forde, M.C. Review of NDT methods in the assessment of concrete and masonry structures. *NDT E Int.* **2001**, *34*, 71–84.
5. Hugenschmidt, J. Concrete bridge inspection with a mobile GPR system. *Constr. Build. Mater.* **2002**, *16*, 147–154.
6. Ellenberg, A.; Kontsos, A.; Moon, F.; Bartoli, I. Bridge related damage quantification using unmanned aerial vehicle imagery. *Struct. Control Health Monit.* **2016**, *23*, 1168–1179.
7. Seo, J.; Duque, L.; Wacker, J. Drone-enabled bridge inspection methodology and application. *Autom. Constr.* **2018**, *94*, 112–126.
8. Otsu, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Trans. Syst. Man. Cybern.* **1979**, *9*, 62–66.
9. Adams, R.; Bischof, L. Seeded region growing. *IEEE Trans. Pattern Anal. Mach. Intell.* **1994**, *16*, 641–647.
10. Canny, J. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **1986**, *PAMI-8*, 679-698.
11. Ojala, T.; Pietikainen, M.; Maenpaa, T. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.* **2002**, *24*, 971–987.
12. Jahanshahi, M.R.; Kelly, J.S.; Masri, S.F.; Sukhatme, G.S. A survey and evaluation of promising approaches for automatic image-based defect detection of bridge structures. *Struct. Infrastruct. Eng.* **2009**, *5*, 455–486.
13. Shelhamer, E.; Long, J.; Darrell, T. Fully Convolutional Networks for Semantic Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 640–651.
14. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015*. Springer International Publishing: Cham, Switzerland, 2015.
15. Chen, L.C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A.L. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *40*, 834–848.
16. Chen, L.-C.; Papandreou, G.; Schroff, F.; Adam, H. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv* **2017**, arXiv:1706.05587. https://doi.org/10.48550/arXiv.1706.05587.

17. Chen, L.-C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In *Computer Vision—ECCV 2018*; Springer International Publishing: Cham, Switzerland, 2018.
18. Zhang, H.; Liu, J.; Hu, G. FCN attention enhancing asphalt pavement crack detection through attention mechanisms and fully convolutional networks. *Sci. Rep.* **2025**, *15*, 25257.
19. Islam, M.M.M.; Kim, J.-M. Vision-Based Autonomous Crack Detection of Concrete Structures Using a Fully Convolutional Encoder–Decoder Network. *Sensors* **2019**, *19*, 4251. https://doi.org/10.3390/s19194251.
20. Si, J.; Lu, J.; Zhang, Y. An FCN-based segmentation network for fine linear crack detection and measurement in metals. *Int. J. Struct. Integr.* **2025**, *16*, 1117–1137.
21. Chaiyasarn, K.; Buatik, A.; Mohamad, H.; Zhou, M.; Kongsilp, S.; Poovarodom, N. Integrated pixel-level CNN-FCN crack detection via photogrammetric 3D texture mapping of concrete structures. *Autom. Constr.* **2022**, *140*, 104388.
22. Hou, L.; Zhang, Z.; Zhang, X.; Zhu, X.; Yang, S.; Huang, C.; Wang, W.; Li, X.; Hu, J.; Zhao, L. A novel enhanced pavement crack semantic segmentation method based on STEH-UNet model. *Adv. Space Res.* **2026**, *77*, 2881–2905.
23. Guo, F.; Cui, Q.; Wang, Y.; Chen, J. EFE-UNet: Efficient detection of tunnel cracks under seam linear interference and multiscale crack differences. *Signal Image Video Process.* **2025**, *19*, 197.
24. Wang, W.; Du, W.; Song, X.; Chen, S.; Zhou, H.; Zhang, H.; Zou, Y.; Zhu, J.; Cheng, C. DRA-UNet for Coal Mining Ground Surface Crack Delineation with UAV High-Resolution Images. *Sensors* **2024**, *24*, 5760. https://doi.org/10.3390/s24175760.
25. Ali, L.; AlJassmi, H.; Swavaf, M.; Khan, W.; Alnajjar, F. Rs-net: Residual Sharp U-Net architecture for pavement crack segmentation and severity assessment. *J. Big Data* **2024**, *11*, 116.
26. Pan, Z.; Zhang, X.; Jiang, Y.; Li, B.; Golsanami, N.; Su, H.; Cai, Y. High-precision segmentation and quantification of tunnel lining crack using an improved DeepLabV3+. *Undergr. Space* **2025**, *22*, 96–109.
27. Allen, A.Z.; Xu, X.; Ding, Y.; Qian, Y.; Dong, Z.; Zhang, H.; He, A. Intelligent Detection of Sealed Crack with 2D Asphalt Pavement Images. *J. Transp. Eng. Part B Pavements* **2025**, *151*, 04024054.
28. Wang, D.; Wang, Z. Dual-teacher knowledge distilled DeepLabV3+ with enhanced edge features for efficient airplane surface crack segmentation. *Signal Image Video Process.* **2025**, *19*, 1395.
29. Zheng, J.; Tang, C.; Sun, Y. BridgeFusionNet: A hybrid convolutional-transformer architecture for road surface crack. *Eng. Appl. Artif. Intell.* **2025**, *160*, 112085.
30. Guo, X.; Tang, W.; Wang, H.; Wang, J.; Wang, S.; Qu, X.; Lin, X. MorFormer: Morphology-Aware Transformer for Generalized Pavement Crack Segmentation. *IEEE Trans. Intell. Transp. Syst.* **2025**, *26*, 8219–8232.
31. Zhao, Y.; Li, W.; Ding, J.; Wang, Y.; Pei, L.; Tian, A. Crack instance segmentation using splittable transformer and position coordinates. *Autom. Constr.* **2024**, *168*, 105838.
32. Mahmood, F.; Inglezou, M.; Chatzakos, P.; Porichis, A. Enhancing Crack Segmentation With Limited Data: SinGAN-Based Synthesis and Blending of Textures and Cracks. *IEEE Access* **2025**, *13*, 104555–104564.
33. Deng, L. CrackGauGAN: Semantic Layout-based Crack Image Synthesis for Automated Crack Segmentation. In *Proceedings of the 41st International Symposium on Automation and Robotics in Construction*; International Association for Automation and Robotics in Construction (IAARC): Odense, Denmark, 2024; pp. 1–8.
34. Liu, J.; Yang, X.; Lau, S.; Wang, X.; Luo, S.; Lee, V.C.-S.; Ding, L. Automated pavement crack detection and segmentation based on two-step convolutional neural network. *Comput.-Aided Civ. Infrastruct. Eng.* **2020**, *35*, 1291–1305.
35. Yue, B.; Dang, J.; Sun, Q.; Wang, Y.; Min, Y.; Wang, F. TSPCS-net: Two-stage pavement crack segmentation network based on encoder-decoder architecture. *Eng. Appl. Artif. Intell.* **2025**, *141*, 109840.
36. Wang, Y.; He, Z.; Zeng, X.; Zeng, J.; Cen, Z.; Qiu, L.; Xu, X.; Zhuo, Q. GGMNet: Pavement-Crack. Detection Based on Global Context Awareness and Multi-Scale Fusion. *Remote Sens.* **2024**, *16*, 1797. https://doi.org/10.3390/rs16101797.
37. Liu, K.; Ren, T.; Lan, Z.; Yang, Y.; Liu, R.; Xu, Y. CGV-Net: Tunnel Lining Crack Segmentation Method Based on Graph Convolution Guided Transformer. *Buildings* **2025**, *15*, 197. https://doi.org/10.3390/buildings15020197.
38. Tian, X.; Jin, Y.; Zhang, Z.; Liu, P.; Ni, F. A knowledge-guided multimodal network for video summarization. *Expert. Syst. Appl.* **2026**, *324*, 132566.
39. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.
40. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. YOLOv7, Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023.

41. Wang, C.-Y.; Yeh, I.H.; Mark Liao, H.-Y. YOLOv9, Learning What You Want to Learn Using Programmable Gradient Information. In *Computer Vision—ECCV 2024*; Springer Nature Switzerland: Cham, Switzerland, 2025.
42. Girshick, R. Fast R-CNN. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015.
43. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014.
44. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 1137–1149.
45. Woo, S.; Park, J.; Lee, J.-Y.; Kweon, I.S. CBAM: Convolutional Block Attention Module. In *Computer Vision—ECCV 2018*; Springer International Publishing: Cham, Switzerland, 2018.
46. Shi, W.; Caballero, J.; Huszár, F.; Totz, J.; Aitken, A.P.; Bishop, R.; Rueckert, D.; Wang, Z. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016.
47. Dai, J.; Qi, H.; Xiong, Y.; Li, Y.; Zhang, G.; Hu, H.; Wei, Y. Deformable Convolutional Networks. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017.
48. Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. *arXiv* **2017**, arXiv:1710.10903. https://doi.org/10.48550/arXiv.1710.10903.
49. Salehi, S.S.M.; Erdogmus, D.; Gholipour, A. Tversky Loss Function for Image Segmentation Using 3D Fully Convolutional Deep Networks. In *Machine Learning in Medical Imaging 2017*; Springer International Publishing: Cham, Switzerland, 2017.
50. Loshchilov, I.; Hutter, F. Decoupled Weight Decay Regularization. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
51. Zou, Q.; Zhang, Z.; Li, Q.; Qi, X.; Wang, Q.; Wang, S. DeepCrack: Learning Hierarchical Convolutional Features for Crack Detection. *IEEE Trans. Image Process.* **2019**, *28*, 1498–1512.
52. Li, H.; Zhang, H.; Zhu, H.; Gao, K.; Liang, H.; Yang, J. Automatic crack detection on concrete and asphalt surfaces using semantic segmentation network with hierarchical Transformer. *Eng. Struct.* **2024**, *307*, 117903.
53. Mohammed, M.A.; Zhou, H.; Zhang, J.; Xu, S.Automated lightweight networks for multi-material bridge crack segmentation. *Autom. Constr.* **2026**, *183*, 106808.
54. Feng, L.; Guo, M.; Wang, W.; Chen, Y.; Shi, Q.; Guo, W.; Lou, Y.; Kang, H.; Chen, Z.; Zhu, Y. Comparative Analysis of Machine Learning Methods and a Physical Model for Shallow Landslide Risk Modeling. *Sustainability* **2023**, *15*, 6. https://doi.org/10.3390/su15010006.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.