Efficient attention-based deep encoder and decoder for automatic crack segmentation

Dong H Kang and Young-Jin Cha

Abstract
Recently, crack segmentation studies have been investigated using deep convolutional neural networks. However, significant deficiencies remain in the preparation of ground truth data, consideration of complex scenes, development of an object-specific network for crack segmentation, and use of an evaluation method, among other issues. In this paper, a novel semantic transformer representation network (STRNet) is developed for crack segmentation at the pixel level in complex scenes in a real-time manner. STRNet is composed of a squeeze and excitation attention-based encoder, a multi head attention-based decoder, coarse upsampling, a focal-Tversky loss function, and a learnable swish activation function to design the network concisely by keeping its fast-processing speed. A method for evaluating the level of complexity of image scenes was also proposed. The proposed network is trained with 1203 images with further extensive synthesis-based augmentation, and it is investigated with 545 testing images (1280 x 720, 1024 x 512); it achieves 91.7%, 92.7%, 92.2%, and 92.6% in terms of precision, recall, F1 score, and mIoU (mean intersection over union), respectively. Its performance is compared with those of recently developed advanced networks (Attention U-net, CrackSegNet, Deeplab V3+, FPHBN, and Unet++), with STRNet showing the best performance in the evaluation metrics—it achieves the fastest processing at 49.2 frames per second.

Keywords
Image segmentation, image analysis, concrete crack segmentation, image synthesis, pixel-level classification, real-time processing, computer vision, damage detection, deep learning, semantic segmentation

Introduction
Deep learning–based approaches were introduced to overcome the limitations of the traditional image processing–based damage detection approaches in recent years. Cha et al.1 proposed the detection of structural damage using deep a convolutional neural network (CNN). They designed a unique CNN, and it was trained and tested to detect concrete cracks in the various image conditions that have real and uncontrolled lighting conditions including blurry and shadowed. For practical applications, the network has been examined using the images coming from an unmanned aerial vehicle (UAV) for concrete crack detection.2 The network adopted a sliding window technique to localize the detected cracks, but this technique requires heavy computational cost, and defining the proper size of the sliding window is another issue by considering camera and lens properties, camera and object distance, and size of cracks. Instead of the sliding window approach, faster region-based convolutional neural network (Faster R-CNN)3 was applied for damage detection and localization.4,5 This Faster R-CNN proposes various sizes of bounding boxes to detect and localize different sizes of damage. The network uses the same base network for detection and localization; therefore, it is faster than the other types of localization methods (e.g., sliding window technique) and became the mainstream in the deep learning–based multiple types of damage detection problems.5–9

Localization of structural damage with bounding boxes is not enough for damage quantification. Specifically, it is too coarse to use bounding boxes or sliding window to measure the thickness and length of detected concrete cracks. U-net10 was applied for pixel-level crack segmentation.11 However, this method was only applied to pure asphalt surfaces without any complex objects or background scenes. There are numerous similar studies in this crack segmentation problem. From extensive literature reviews, we recognized four major limitations or

University of Manitoba, Winnipeg, MB, Canada

Corresponding author:
Young-Jin Cha, University of Manitoba The Department of Civil Engineering, E1-430, EITC, 15 Gillson Street, Winnipeg, MB, R3T 5V6, Canada.
Email: young.cha@umanitoba.ca
disadvantages of existing studies that should be overcome or improved:

(1) Although monitoring pavements without considering complex scenes may not constitute a serious problem, detecting structural damage such as concrete cracks is a major limitation if the network cannot detect only cracks in the complex scenes since many structures are located within various different visual scenes. Many researchers worldwide have conducted pixel-level detection of cracks and reported results as shown in Table 1. Only SDDNet,12 HB FasterRCNN,13 and Resnet15014 considered cracks in the complex scenes.

(2) Another limitation is that most existing studies did not use proper evaluation metrics. Rather, most used accuracy, precision, recall, and F1 score as presented in Table 1. Only SDDNet,12 HB FasterRCNN,13 and Resnet15014 used mIoU as an evaluation metric to efficiently train their networks. However, for crack segmentation, only seven studies12–14,17,21,26,27 used IoU as an evaluation metric. However, most of the claimed IoU performances should be improved.

(3) Most of the existing studies used heavy networks or existing traditional networks that were originally developed for the segmentation of many objects; therefore, these networks need inherently unnecessarily heavy computational cost due to their excessive learnable parameters. Therefore, it is impossible for real-time processing with relatively large input images or video frames (e.g., 1000 × 500) that have 30 frames per second (FPS). Fast processing is an important aspect of civil infrastructure monitoring due to its large scale and is required to process many images to inspect large-scale structures. It does not necessarily process in a real-time manner, but it reduces overall monitoring costs and provides fast updates of the structural states. For example, as presented in Table 1, DeepCrack used VGG16 as the backbone network. Liu et al.20 used U-net10 architecture for concrete crack detection, Dung and Anh.16 used fully convolution network (FCN),31 König et al.19 used Attention network,32 Bang et al.14 used Resnet,33 Mei et al.23 used DenseNet,34 and Ji et al.17 used

| Authors                    | Backbone network | Train | Val | Test | F1 score | mIoU   | Test input size | FPS |
|----------------------------|------------------|-------|-----|------|----------|--------|----------------|-----|
| Bang et al., 201914        | Yes              | Resnet150 | 427  | -    | 100      | 89.0   | 59.7           | 0.22|
| Benz et al., 201915        | No               | Crack NaasNet | 1303 | 487  | 115      | 82.9   | 512×512        | -   |
| Choi and cha, 201912       | Yes              | SDDNet | 160  | -    | 40       | 84.6   | 1024×512       | 36  |
| Dung and anh, 201916       | No               | FCN    | 400  | 100  | 100      | 89.3   | —              | 13.8|
| Ji et al., 202017          | No               | Deeplabv3+ | 300  | 50   | 50       | 73.3   | 512×512        | -   |
| Jiang and zhang, 202018    | No               | SSDLiteMobileNetV2 | 1030 | -    | 300      | -      | 640×480        | 24  |
| Kang et al., 202013        | Yes              | HB FasterRCNN | 400  | -    | 100      | -      | 83             | 0.25|
| König et al., 201919       | No               | AttentionUnet | 95   | -    | 60       | 92.8   | 48×48          | -   |
| Liu et al., 2019a20        | No               | U-net   | 38   | 19   | 27       | 90.0   | 512×512        | 8   |
| Liu et al., 2019b21        | No               | DeepCrack | 300  | -    | 237      | 86.5   | 85.9           | 10  |
| Liu et al., 202022         | No               | U-net, ResNet-34 | 770  | 257  | 257      | 95.75  | 800×600        | 4   |
| Mei et al., 202023         | No               | DenseNet | 700  | 100  | 200      | 75.4   | 256×256        | 0.25|
| Nayyeri et al., 201924     | No               | RTV     | 352  | -    | 352      | 75.0   | 400×500        | -   |
| Ni et al., 201925          | No               | GoogLeNet | 65K  | -    | 32K      | -      | 224×224        | -   |
| Ren et al., 202026         | No               | CrackSegNet | 307  | -    | 102      | 74.6   | 59.1           | 11  |
| Tong et al., 202027        | No               | CNN, SVM | 5292 | -    | 1764     | -      | 75.8           | 6.1 |
| Yang et al., 201928        | No               | FPHBN   | 200 (1896) | 50 | 908 (1124) | -     | 480×320        | -   |
| Zhang et al., 201711       | No               | CrackNet | 1800 | -    | 200      | 88.8   | 1024×512       | 0.34|
| Zhang et al., 2019a29      | No               | SegNet  | -    | 155  | 79.4     | 256×256 | 1.42           |    |
| Zhang et al., 2019b30      | No               | CrackNet-R | 300  | -    | 500      | 91.84  | 1024×512       | 1.4 |
DeepLabV3+,35 and Ren et al.26 applied SegNet.36 Among all these networks, only SDDNet.37 could do real-time processing with 36 FPS for 1024 × 512 RGB images.

(4) Some studies used a too small number of training and testing data sets with small sizes of input images. This results in the high possibility of over-fitting for the specific types of cracks with specific image conditions. For example, Ji et al.17 used a total of 84 images of relatively small sizes (i.e., 512 × 512), and SDDNet also used only 40 images for testing with relatively large input image (1024 × 512). Further, most of the studies used very small testing input image sizes which are all below 1000 × 500 except those conducted by references12,14,30. Testing input image of small sizes also has the possibility of over-fitting to specific types of cracks. It is also not efficient to monitor large-scale civil infrastructure, and it is also very limited in terms of detecting thin cracks in a relatively long distance of camera and object.

Based on our extensive literature review described above, we propose a new deep encoder and decoder based network with an improved/increased data set and performance to resolve the four issues mentioned above in this pixel-level crack detection problem in complex scenes. In order to realize this, we propose the use of semantic trainable representation network (STRNet) to improve performance in terms of mIoU by keeping the real-time network processing speed for a relatively large size of testing input image frame (1024 × 512) from Tesla V100 GPU. Also, we establish a large ground truth dataset (i.e., 1748 RGB images with sizes of 1024 × 512, 1280 × 720) for training and testing purposes to consider complex background scenes for robust detection by avoiding overfitting to specific types of cracks and background scenes. We used some publicly available datasets—deep crack21 and concrete crack segmentation datasets37—by fixing severe errors. Some ground-truth images of these datasets were coarsely labeled. This can cause poor training results, even when an advanced network is designed and used. Therefore, the images of these existing datasets were re-annotated to reduce annotation errors. To improve the network’s performance, we also used focal-Tversky loss function38 and adopted image synthesis techniques to augment the prepared ground truth training data to negate and detect crack-like features on complex scenes.

Proposed STRNet

An architecture named STRNet of deep convolutional neural network is proposed to segment concrete cracks on complex scenes in pixel-level in a real-time manner (i.e., at least 30 FPS) with a testing input size of 1024 × 512 RGB images/videos. The STRNet is composed of a new STR module-based encoder, an Attention decoder with coarse upsampling block, a traditional convolutional (Conv) operator, a learnable Swish nonlinear activation function,99 and batch normalization (BN) to segment only cracks in complex scenes with real-time manner. The schematic view of the STRNet is shown in Figure 1. In order to develop this high-performance network with low computational cost, many advanced networks were investigated to figure out their strengths and weaknesses.

STRNet processes an input image by 16 Conv filters with a size of 3 × 3 × 3 with stride (S) 1, BN30 and Hswish41 activation function with a skipped connection. The result of these processes in the first block of Figure 1 is inputted to a newly designed STR module and final “Concatenation block” as shown in Figure 1. The STR module is repeated 11 times, and afterward, the feature map is fed into Max pooling operator and is then forwarded to the newly designed Attention decoder and Upsampling module. The result of Max pooling goes through the Attention decoder two times, and the output is fed into Upsampling and Coarse upsampling modules. The outputs of the final upsampling and coarse upsampling modules are concatenated with the output of the first Conv block as shown in Figure 1. The concatenated features are processed by pointwise convolution (PW) to match the output to the input image size for final pixel-level segmentation. The details of the developed modules and their roles are described in the following subsections.

STR module

The STR module is newly developed in this paper to improve the segmentation accuracy by reducing the computational cost for real-time processing on the complex scenes. The STR module is composed of PW, depthwise convolution (DW), BN, Swish activation function, squeeze, and extension-based attention module as shown in Figure 2. STR module has three different configurations (i.e., STR_config1, STR_config2, and STR_config3) as shown in Figure 2. STR_config1 has simple processes of 3rd block, 4th block, and 10th block with PW, DW, BN, and rectified linear unit (ReLU) activation function,43 illustrated as the dark greenish block shown in Figure 2. STR_config2 is combined with STR_config1 and squeeze and excitation-based attention (SEA) module44 with ReLU illustrated as the yellowish block shown in Figure 2. STR_config3 is the entire network of the STR module with blocks from 1st to 11th. STR module is repeated 11 times, and different configuration is operated in each repeat as presented in Table 2.

All these arrays of configurations are new and unique with different DW convolution sizes, different stride sizes (S1, S2), with/without SEA module, ReLU/Swish activation
function, and skipped connection. The Con in Table 5-2 indicates the skipped connection with the red arrow line as shown in Figure 5-2, which only happens with US and AD which stand for upsampling and attention decoder, respectively. Therefore, the Connector is only used in the repeats in 3 and 11 to keep multilevel features. Publicly available segmentation networks usually apply stride 16 or 32 to the feature map in an encoder module, which means that the extracted feature map size is reduced to 16 or 32 times smaller than the original input image size. However, these large spatial contractions of the extracted feature maps compared to the input size may cause the loss of important features. This issue is found throughout our extensive experimental studies to develop this unique network, although it might be only applicable to this unique crack segmentation problem. Due to the nature of cracks with very long and thin shapes, a network may need a slightly larger feature map. Therefore, we applied stride 8 (i.e., $S^1$), since we have three “2” in Table 2, but this small stride causes the high computational cost through deep hidden layers of the proposed network.

To resolve this issue, and to maintain important features and real-time processing, we use different STR configuration (i.e., configs 1–3) in each repeat as presented in Table 2. Through the STR_configs1 and 2, we extracted features by keeping its relatively large feature map, but these large feature maps require large computational costs compared to small feature maps through the deep layers of the network. Therefore, to reduce its feature map by keeping the important features, we used STR_config3 with squeeze and excitation based attention operation. The role of squeeze and excitation operation is to extract important features. In order to squeeze the extracted feature map, global average pooling at the fifth block is applied in STR_configs2 and 3. The global average pooling performs the average pooling operation entire W (input width) and H (input height) size in each feature channel, so the output feature map becomes $1 \times 1 \times \alpha D$ at the sixth block. The physical meaning of this global average pooling is the extraction of important (i.e., mean) features from the extracted features. Here, $\alpha$ is given in Table 2, and D is 16 since we conducted traditional Conv 16 times, as shown in Figure 1. This process is called squeeze process, and it extracts important features while compressing information.
This feature is fed into two linear functions (LinearT)\textsuperscript{45} with ReLU and $H = \text{Sigmoid}$\textsuperscript{42,45}

$$H - \text{Sigmoid} = \frac{\text{ReLU}6(x + 3)}{6} \quad (1)$$

where ReLU6 is an embedded activation function in Pytorch.\textsuperscript{46} ReLU6 has a unique shape with a maximum output value six for all inputs greater than or equal to 6. The excitation process recovers the squeezed feature map to the original size by reproduction of the squeezed feature map ($1 \times 1 \times \alpha D$). The $H - \text{Sigmoid}$ expressed in equation (1) provides the bi-linearity activation function. The output of DW from fourth block is multiplied ($\bigodot$) by the output of excitation at eighth block.

The role of STR\textsubscript{config}3 restores an important feature of the STR module use ReLU. The result of PW convolution and one DW convolution as expressed in equation (7).

$$\begin{align*}
\text{STR\textsubscript{config}3} & = (\text{PW}_2 + \text{DW}_2 + \text{PW}_2) + (\text{PW}_2 + \text{DW}_3 + \text{PW}_3), \\
& = C_2 H_2 W_2 (k^2 + 2O_2) + C_1 H_1 W_1 (k_2 + 2O_3), \\
& = \frac{C_2 H_2 W_2 (k^2 + 2O_2)}{6} + \frac{C_1 H_1 W_1 (k^2 + 2O_3)}{4} \quad (8)
\end{align*}$$

where $C_2 = 6C_2 = C_3$, $H_1 = H_2 = 2H_3$, $W_1 = W_2 = 2W_3$, and $O_1 = O_2 = O_3$. The discrepancy between equation (7) and equation (8) is expressed as equation (9). The equation (9) is clearly a positive value, which means that the number of calculation of the STR\textsubscript{config}3 is smaller than that of the STR\textsubscript{config}1, which contributes the real-time processing of the STRNet.

$$\text{STR\textsubscript{config}1} - \text{STR\textsubscript{config}3} = \frac{(7k^2 + 14O_1 + 2)}{12} \quad (9)$$

Another technical contribution of this STR module is the implementation of a non-linear activation function. Most recently, proposed networks in this area typically only use ReLU because of its simplicity in differential calculation for backpropagation and to reduce computational cost and automatic hibernation of unnecessary learnable parameters in the network. However, our objective is to develop a concise and efficient network by using a smaller number of hidden layers, meaning most of the assigned learnable parameters in each filter in each layer should be fully used to extract multiple levels of features for high performance of the pixel-level segmentation.

Therefore, using ReLU is no longer a viable option for this concise and light objective specific network. We only used this ReLU for the first three STR module repetitions for the stable training process as presented in Table 2. After that, we used a learnable Swish nonlinear activation function\textsuperscript{39} to resolve this issue in the STR module.

$$\text{Swish}(x) = x \cdot \text{sigmoid}(\theta x) \quad (10)$$

where $\theta$ is a learnable parameter of the Swish activation function. The major benefit of this learnable Swish activation function is that it can be converted from scaled linear ReLU to a nonlinear function by changing the $\theta$ from $0$ to $\infty$. Due to the dynamic shape of the activation function, this network is able to extract features more efficiently and precisely. However, it also may cause an unstable training process; therefore, as we described, the first three repetitions of the STR module use ReLU. The result of PW convolution in the 10th block in Figure 2 is upsampled to the input
feature size (first block) of W and H. The input of the STR module and the upsampled result are densely concatenated to keep the different multi-level levels of features in the 11th block. This process recovers the loss of features from the two strides (S2) of DW convolution in the second block. After, the densely piled features are processed by PW convolution to restore the D channel value, which serves to facilitate the repetition of the STR module.

**Attention decoder**

The role of traditional decoders in this pixel-level segmentation problem is to recover the size of the extracted feature map from well-designed encoders. However, the performance of the encoders is not usually high enough to achieve a very high level of segmentation as we previously discussed in the Introduction section. Therefore, in this paper, we developed a unique attention-based decoder to support the role of the STR encoder to screen wrongly extracted features in the encoding process. Initially, we used existing attention decoders, but due to their heavy computational cost, real-time processing was impossible. Therefore, we designed a unique decoder by configuration of Attention decoder, Upsampling and Coarse upsampling by using the attention operation minimally to reduce the heavy computational cost to keep its real-time processing performance as shown in Figure 3.

The role of “Attention decoder” shown in Figure 3 is to screen wrongly extracted features from the STR encoder and to recover the reduced feature spatial size from STR module by keeping its unique features from the original input size. Usually, an attention decoder is repeated more than 4 times in publicly available networks. However, we only repeated it two times to reduce computational cost, and we used Upsampling and Coarse upsampling operators to supplement this reduced number of attention decoder repeat as shown in Figure 1.

In Figure 3, the first input size ([W', H', D'] = [64, 32, 96]) is the final output of the encoder with the result of 2 × 2 max pooling. This input is applied to 3 × 3 convolution and BN. This result is processed by PW with/without BN and produces Query \([W', W', H']\), Key \([W', W', H']\), and Value \([W', W', H']\). These maps are then reshaped using embedded function View of Pytorch from 3D to 2D and resulted in \([W' \times H', D']\), \([W', W', H']\), and \([W' \times H', D']\), respectively. The Query and Key are multiplied (symbolized as “•”) and result in M1 attention map. The M1 attention map is filtered by (11) and output M2. The reshaped Value is multiplied with the M2 attention map which is attention process.

\[
M2 = \text{softmax}(\frac{M1}{\sqrt{D'}}) \tag{11}
\]

The object context produced by attention process and the output of first Conv operation from the first block of the overall architecture of the STRNet as shown in Figure 1 are concatenated as shown in Figure 3. PW_Conv condenses this concatenated feature map, and dropout is applied to prevent overfitting. Finally, the transposed convolution restores the semantic mask.

**Upsampling and coarse upsampling**

The Upsampling layer is intended to double the dimensions of input, and it is commonly used in any segmentation network. The input feature passes the bilinear upsampling. Bilinear upsampling increases width \(_W\) and height \(_H\) two times. After that, the 3 × 3 convolution, BN, and ReLU activation function are performed to reduce the depth of the map. The size of upsampling output follows the size of original input image.

**Concatenation block**

Skip connection or simple bilinear upsampling has been widely used for encoder and decoder-based networks to keep multi-level features. We also use the multiple skip connections to obtain better segmentation as shown in Figure 1. As shown in Figure 4, we concatenate the results of the traditional Conv block, Attention decoder with...
Coarse upsampling, and Upsampling. The $\hat{W}$, $\hat{H}$, and $\hat{D}$ are 1024, 512, and 516, respectively. The concatenated feature map is processed by PW convolution to have the same depth size compared to a binary ground truth.

Established data bank

To train the developed STRNet for crack segmentation on various complex scenes, we prepared ground truth data from various sources. A total of 1784 images sized 1024 × 512 and 1280 × 720 were prepared. Some (612) of them came from existing available datasets. The raw images of these existing datasets were re-annotated to reduce annotation errors. Some (300) of them came from our previous studies, and new datasets (836) from various structures and locations was established. The detailed information of the developed datasets is presented in Table 3. To minimize the time and effort to prepare training image data, we took advantage of using our previous SDDNet. The raw images were initially processed by this network and the output errors such as false positives and false negatives were fixed manually. The raw images that should be ground-truth datasets were initially processed by the second author’s previously developed SDDNet to speed up the tedious dataset labeling task. Since his previous SDDNet provides satisfactory performance of segmentation (i.e., pixel level labeling) of cracks, we used the results of the SDDNet, and some of the incorrectly segmented (i.e., labeled) pixels were fixed to be used as ground-truth data.

Data augmentation

The prepared ground truth data presented in Table 3 is not enough to achieve high performance segmentation which can negate the detection of any cracklike features on the complex scenes. Therefore, traditional data augmentation skills such as random rotation and random cropping were conducted. Moreover, synthesis techniques of ground truth images to generate cracks on complex scenes were also applied by inserting an object of interest into another non-target image with complex scenes that would allow us to achieve a robust classifier. Figure 5 shows two approaches to generating the procedure and synthetic images.

The first approach is that the image with cracks is set as a background image, and a non-target image having complex scenes but without cracks is inserted in the background image as shown in Figure 5. The second approach is vice versa. After, the synthesized images are further processed with random flipping, rotation, and brightness operations, and they are resized to 1024 × 512. The complex non-target images without cracks are collected from Open Image Dataset v4. We used 1203 images from 99,999 images. In order to crop the area having crack pixels in ground truth images, the CropNonEmptyMaskIfExists function from Alns, and they are resized to 1024 × 512. The cropped crack area was patched to a non-target complex background image as shown in Figure 5. The cropped crack image size is randomly selected from 300 × 204 to 400 × 512, and the location of insertion is also randomly selected. Therefore, the eventual total number of augmented images for training is 12,030 as presented in Table 3.

Complexity of the proposed dataset

Considering complex background scenes of structural damage in the real world is critical, as mentioned in the Introduction section. However, the evaluation of these
scenes’ complexity levels can be subjective if no quantitative evaluation method is used. For this reason, we put forward an algorithm for evaluating the complexity of an image dataset. The fundamental concept of the complexity check evaluation algorithm is to count the number of objects in a scene. The higher the object number, the greater the complexity level.

To count the number of objects in an image, we used Felzenszwalb’s graph segmentation method. Felzenszwalb’s algorithm verifies the relationship between pixels and edges in an image. For example, if an edge appears between two pixels, these two pixels are assumed to be located in different clusters (i.e., objects), $C_m$, as expressed in equation (12). If no edge appears between pixels, then they are assumed to be located in the same object.

**Table 4.** Comparison of complexity scores.

| Authors     | Complexity score | # Of images |
|-------------|------------------|-------------|
| Crack500    | 57               | 27          | 494         |
| CrackForest | 58               | 3.57        | 118         |
| CrackSegNet | 26               | 6.69        | 813         |
| DeepCrack   | 21               | 11.4        | 527         |
| FCN         | 16               | 7.3         | 776         |
| Ours        | -                | 41.23       | 1748        |
\[
\text{GraphSeg}(\pi_i, \pi_j) = \begin{cases} 
\text{if} & (\pi_i \leq e_n \leq \pi_j) \\
\text{else} & (\pi_i, \pi_j) C_n 
\end{cases}
\]  \tag{12}

where \(\pi_i\) and \(\pi_j\) are the input pixels, and \(e_n\) is the pre-defined edge value, which should be defined by a user. In this study, we defined this value as constituted by color and intensity. \(C_n\) is the object cluster, that is, the cluster of pixels, as expressed in equation (13).

\[
C_n = (p_0, \ldots, p_n) \tag{13}
\]

\(C_n\) is too fine a level of object segmentation without consideration of image noise. Therefore, a smooth function was adopted, as expressed in equation (14).

\[
\text{Felzenszwalb}(C_i, C_j) = \begin{cases} 
\text{True} & (\text{Min}(C_i - C_j) < \text{Max}(C_i(p_i) - C_i(p_j))) \\
\text{False} & \text{otherwise} 
\end{cases}
\]  \tag{14}

where \(\text{Min}(C_i - C_j)\) expresses the minimum pixel value difference between two clusters via pixel-to-pixel comparisons of cases. \(\text{Max}(C_i(p_i) - C_i(p_j))\) denotes the maximum pixel value differences within the same cluster that should be tuned. If the calculated minimum value is lower than the calculated maximum value, the two clusters are merged. On the basis of this rule, the algorithm assigns all pixels and generates an object segmentation map. As shown in Figure 6, Felzenszwalb’s algorithm produces an object segmentation map and identifies the boundary of objects in images obtained from our dataset and existing crack datasets (i.e., Crack 500, \(^{57}\) CrackForest, \(^{58}\) CrackSegNet, \(^{26}\) DeepCrack, \(^{21}\) and FCN\(^{16}\)).

Using the results of Felzenszwalb’s algorithm, we assigned an object number to each object within an image. To calculate the complexity score (i.e., the level of complexity), the number of unique numbers for each image was determined. To evaluate the level of complexity of the available crack datasets, all the images were evaluated, and the average was calculated for each dataset. The results are presented in Table 4. Crack 500 had a slightly higher complexity score than those generated by the other datasets (except ours), even though it comprised only pure pavement surface images. This result is attributed to the fact that most
of the images were asphalt surface images with high asperity due to the ingredients of coarse pavement material. Nevertheless, our dataset showed the highest complexity score.

Training details

This section describes the details of the training process and hardware. Python programming language with Pytorch 1.6 deep learning library was used to code the STRNet. The STRNet was trained in a graphic processing unit (GPU) equipped workstation. The workstation specifications are Intel Core i76850K CPU, Titan XP GPU, and 128 GB RAM. To train our models, we set up the four Titan XP GPU using Nvidia Apex distributed data parallel (DDP) training library. The input image size is 1024 × 512, which is randomly cropped if the image size is bigger than the input size. The use of proper loss function is crucial; therefore, we investigated several recently developed functions such as cross entropy loss, dice cross entropy loss, and mIoU. Eventually, focal-Tversky loss function was used for training. The focal-Tversky loss was used as a combination of the loss function as follows

\[ TL = \frac{TP + S}{TP + FP \cdot \alpha + FN \cdot \beta + S} \]  

\[ Focal - TverskyLoss = (1 - TL)^\gamma \]  

where TL is Tversky loss. TP, FP, and FN are true positive, false positive, and false negative, respectively. α, β, γ, and S are all hyperparameters. Based on trial and error, α, β, γ, and S are defined as 0.5, 0.5, 1.3, and 1.0, respectively. Abraham et al. investigated the performance of this focal-Tversky loss function in the segmentation problem and showed that it outperformed to get balance between precision (FP) and recall (FN) compared to the dice loss function.

In order to do backpropagation for the learnable parameter updating, the Adam optimizer was employed. The hyperparameters such as first moment, second moment, and dropout rate were defined as 0.9, 0.999, and 0.2, respectively. The initial learning rate was 0.005, and dropped by 20% when the number of epochs were 30, 70, and 120, to keep a stable training process. To reduce the training time, a DDP with batch size eight was also used for four GPUs.

The progress of the focal-Tversky loss through training epoch iteration is plotted in Figure 7. As shown in the figure, we conducted two types of training and validation processes: hold-out validation and train-valid-test split validation. For the hold-out validation, we divided the total dataset into training and testing sets, as tabulated in Table 3, and conducted training and testing as the validation, which is plotted in Figure 7(a). For the train-valid-test split validation, a 10% validation dataset (170 images) of the total images (1748 images) was randomly selected from the training dataset (1203 images), and training and validation losses and scores were plotted during the training iterations, as shown in Figure 7(b). In these two validation processes, there was only a small discrepancy between the training score (93.8) and validation score (91.0), see Figure 7(b).
This means that the training set is not developed/determined to achieve a high performance from the specific testing and validation datasets, because the training score (93.8) is slightly higher than the validation score (91.0) and the claimed testing score (92.6).

**Case studies**

The developed STRNet was extensively experimentally investigated. In *case studies*, some parametric studies were carried out to find effective image synthesis technique, loss function, activation function, and effective decoder. In *Parametric studies of STRNet*, the eventual STRNet based on the parametric studies was tested on many complex scenes to segment concrete cracks. In *Comparative studies*, extensive comparative studies were conducted in the same training and testing datasets with the same conditions of loss function for fair evaluation.

**Parametric studies of STRNet**

We conducted parametric studies to find the most effective parameters and architecture of STRNet. In order to train and test the developed network, the training and testing data presented in Table 3 were used. All data augmentation techniques described in *Established data bank* were also applied. The used evaluation metrics are

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{17}
\]

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{18}
\]

\[
F1\,\text{-}\,\text{score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{19}
\]

\[
mIoU = \frac{TP}{TP + FP + FN} \tag{20}
\]

The first study was for the method of image synthesis to overcome the limitation of prepared ground truth datasets. We compared two different image synthesis techniques described in *Data augmentation* and the second image synthesis method showed better performances as presented in Table 5. This resulted in a 1.6% improvement. Two different loss functions for effective training of the STRNet were tested. The general IoU loss function, which is the most popular loss function in this field, and the focal-Tversky loss function were compared. The focal-Tversky loss function showed better performance, with a 6.7% improvement of mIoU.

At this experimental test, the image synthesis was applied for both cases. We used the coarse upsampling technique in STRNet and tested the effectiveness. The coarse upsampling method improved the mIoU by approximately 1%. Another unique technique in this STRNet was the attention decoder. The effectiveness of the attention decoder was also investigated, which showed that it improved the mIoU by approximately 2.4%. With these parametric studies, we decided the eventual network of the STRNet with training methods such as image augmentation and loss function.

In order to check any possibility of overfitting and underfitting problems, k-fold random validations for the fully trained network were conducted. In each validation, the 10% (170) of the validation sets were randomly selected from training, testing, and total dataset, respectively, and conducted experiments to calculate mIoU using the fully trained network. All the mIoU from the total 30 number of validation datasets from the training, testing, and total datasets are expressed as "Train," "Test," and "Total," respectively, in Table 6. The average mIoUs from the three different datasets are 93.93%, 92.59%, and 93.15%, respectively. Our claimed maximum

| Model (%) | Precision | Recall | F1 score | mIoU | FPS | # Of param, M |
|-----------|-----------|--------|----------|------|-----|--------------|
| AttentionU-net | 85.63 | 91.22 | 88.33 | 89.1 | 17 | 34 |
| CrackSegNet | 86.33 | 84.89 | 85.61 | 87.1 | 21.4 | 12.4 |
| DeeplabV3+ | 77.37 | 83.6 | 80.36 | 83.24 | 30.2 | 59 |
| FPHBN | 86.35 | 87.2 | 86.78 | 88.28 | 34.0 | 5.9 |
| UNet + | 84.25 | 77.78 | 80.08 | 85.23 | 30.6 | 26.9 |
| STRNet | 91.7 | 92.7 | 92.2 | 92.6 | 49.2 | 2 |

This resulted in a 1.6% improvement. Two different loss functions for effective training of the STRNet were tested. The general IoU loss function, which is the most popular loss function in this field, and the focal-Tversky loss function were compared. The focal-Tversky loss function showed better performance, with a 6.7% improvement of mIoU.

At this experimental test, the image synthesis was applied for both cases. We used the coarse upsampling technique in STRNet and tested the effectiveness. The coarse upsampling method improved the mIoU by approximately 1%. Another unique technique in this STRNet was the attention decoder. The effectiveness of the attention decoder was also investigated, which showed that it improved the mIoU by approximately 2.4%. With these parametric studies, we decided the eventual network of the STRNet with training methods such as image augmentation and loss function.

In order to check any possibility of overfitting and underfitting problems, k-fold random validations for the fully trained network were conducted. In each validation, the 10% (170) of the validation sets were randomly selected from training, testing, and total dataset, respectively, and conducted experiments to calculate mIoU using the fully trained network. All the mIoU from the total 30 number of validation datasets from the training, testing, and total datasets are expressed as "Train," "Test," and "Total," respectively, in Table 6. The average mIoUs from the three different datasets are 93.93%, 92.59%, and 93.15%, respectively. Our claimed maximum
performance of the STRNet was 92.6% as shown in Table 5. Therefore, the obtained validation results are quite close to the final performance. Through these total 30-fold validation processes with total 5100 images, we assume that our trained STRNet is not underfitted and overfitted.

**Experimental testing of STRNet**

In this section, the eventual parameters and module from the experimental studies in Section V-A was selected as the final STRNet. This STRNet showed a maximum 92.6% mIoU on 545 images having complex scenes with 49.2 FPS using single V100 GPU for 1024 × 512 input images. This is much faster than required speed (i.e., 30 FPS) for real-time processing. It provides very stable performance without unbalance among false positives and false negatives based on 91.7% precision and 92.7% recall evaluation metrics including 92.2% F1 score. The reported mIoU 92.6% is considered to be a very high level of accuracy since all the ground truth (GT) data has a minimum level of annotation error because there are many unclear cases that a pixel is included in a crack or intact concrete surface. Therefore, it

| Image | (a) | (b) | (c) | (d) | (e) |
|-------|-----|-----|-----|-----|-----|
| GT    |     |     |     |     |     |
| AT. U-net |     |     |     |     |     |
| CrackSegNet |     |     |     |     |     |
| DeeplabV3+  |     |     |     |     |     |
| FPHEN      |     |     |     |     |     |
| Unet++     |     |     |     |     |     |
| STRNet     |     |     |     |     |     |
Seems that a maximum of 5% error is unavoidable in ground truth data. Some example results of the STRNet on complex scenes are illustrated in Figure 8. The Case 1 is related to the image having shadow, so cracks in the image are unclear to the naked eye, but the STRNet segmented cracks are very accurately based on the ground truth. Case 2 depicts a very thin crack with a blurry image, Case 3’s image has water stains, Case 4 portrays crack-like features on concrete wall, and Cases 5 & 6 are images with complex scenes. In each of these cases, the STRNet showed satisfactory results.

**Comparative studies.** Extensive comparative studies were conducted to show the superior performances of the proposed STRNet compared to the traditional networks. The selected networks are attention Unet,19 DeepLabv3+,17 Unet++,61 FPHBN,28 and CrackSegNet.26 All these advanced networks are recently developed and showed state of the art performances in this segmentation area and applied them to the crack segmentation problem.

Each of these six selected networks were trained using the same training dataset, data augmentation techniques, and hyperparameters, including s function for fair comparison. All of these well-trained networks were also tested by the same 545 testing images presented in Table 3. The experimental results are tabulated in Table 7. It showed that the proposed STRNet still demonstrated the best performances in terms of precision, recall, F1 score, and mIoU with the fastest processing with 49.2 FPS using single V100 GPU. in Table 7. The number of learnable parameters of the STRNet is smallest, which has the advantage of being operable in embedded microcomputing devices. This is beneficial for real structural applications and commercialization. The attention Unet, DeepLabV3+, and Unet++ showed unbalanced precision and recall scores, which means that these networks involve problems with false positive or false negative detections. In order to compare the performances, the complex scene images in different locations and structures with different lighting conditions are selected and processed by six networks as shown in Figure 9. The proposed STRNet showed superior performance in the selected images. DeepLabV3+ showed the worst performance, with an approximately 9% lower mIoU than that of STRNet. DeepLab V3+ also showed very weak performance in negating dark areas to be detected as cracks. Attention Unet (i.e., AT U-net) and Unet++ have problems negating shadowed areas. FPHBN achieved balanced false positive and false negative detections but still has issues with false positive and false negative detections, as shown in Figure 9 (a) and (d).

**Conclusion**

In this paper, a novel STRNet, which is a deep convolutional neural network, is developed for concrete crack segmentation in pixel level. The developed network was trained using large training data set and tested on 545 images. The performances of the proposed network in terms of precision, recall, F1 score, and mIoU are 91.7%, 92.7%, 92.2%, and 92.6%, respectively, with 49.2 FPS using V100 GPU which is able to process relatively large input images (1280 × 720, 1024 × 512) with real-time manner. From the extensive comparative studies, this demonstrated the best performance in terms of the upper four evaluation criteria. New technical contributions of this paper are:

1. A new deep convolutional neural network was designed to be able to do real-time processing using relatively large input images (1280 × 720, 1024 × 512) with 49.2 FPS.
2. The proposed network showed state of the art performance in segmentation of cracks with 92.6% mIoU.
3. The STRNet has the lightest network size among the compared networks with a 2m memory size, which offers the great benefit of being applicable to real world problems using a microcomputer.
4. The network was able to segment cracks on highly complex scenes including different area, structures, and lighting conditions.
5. The evaluation method of image complexity evaluation method was proposed, and our training and testing datasets showed the highest level of complexity among available the examined datasets.
6. The new encoder named as the STR module was developed to extract multi-level features effectively.
7. The new decoder with the attention module was developed to support the STR encoder by screening wrongly extracted features from the encoder to improve the segmentation accuracy (i.e., 2.4% mIoU).
8. Coarse upsampling was adopted for this crack segmentation problem. It improved the 1% mIoU.
9. The new loss function (Focal-Tversky loss function) was adopted to train the newly designed network to improve the crack segmentation performance (i.e., 6.7% mIoU).
10. Many training and testing data with large image sizes were established to conduct extensive evaluations (see Table 3).
11. The prepared ground truth data were drastically reduced in annotation errors compared to the publicly available crack segmentation data.
12. A new image synthesis technique was adopted to augment the ground truth training data to improve the network performance (i.e., 1% mIoU).
13. A learnable swish activation was adopted to improve the segmentation performance by keeping a concise network which enables faster than real-time processing. This may give us the possibility to increase the testing input size.

The performance of the STRNet was outstanding on the given testing and training datasets, but a larger dataset will be required to monitor the many varying types of structures together using a single trained network. However, this problem can be resolved by grouping the structures, such as bridges, buildings, and dams. Then, depending on the specific group, the user can collect data and train the network. The trained network can be installed beneath a reinforced concrete bridge deck or girders with a vision sensor and microcomputer as an
example of a real structure application. The mixed precision training strategy must test for faster speed.

**Declaration of conflicting interests**

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

**Funding**

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The research was supported by the NSERC Discovery Grant (RGPIN-2016–05923) and the CFI JELF grant, (3739,4).

**ORCID iD**

Young-Jin Cha  https://orcid.org/0000-0002-0738-5615

**References**

1. Cha Y-J, Choi W and Büyüköztürk O. Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil Infrastructure Eng* 2017; 32(5): 361–378.
2. Kang D and Cha Y-J. Autonomous uavs for structural health monitoring using deep learning and an ultrasonic beacon system with geo-tagging. *Computer-Aided Civil Infrastructure Eng* 2018; 33(10): 885–902.
3. Ren S, He K, Girshick R, et al. Faster r-cnn: towards real-time object detection with region proposal networks. *arXiv Preprint arXiv:1506.01497* 2015.
4. Cha Y-J, Choi W, Suh G, et al. Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types. *Computer-Aided Civil Infrastructure Eng* 2018; 33(9): 731–747.
5. Xue Y and Li Y. A fast detection method via region-based fully convolutional neural networks for shield tunnel lining defects. *Computer-Aided Civil Infrastructure Eng* 2018; 33(8): 638–654.
6. Maeda H, Sekimoto Y, Seto T, et al. Road damage detection and classification using deep neural networks with smartphone images. *Computer-Aided Civil Infrastructure Eng* 2018; 33(12): 1127–1141.
7. Beckman GH, Polyzois D and Cha Y-J. Deep learning-based automatic volumetric damage quantification using depth camera. *Automation in Construction* 2019; 99: 114–124.
8. Mei Q and Gül M. Multi-level feature fusion in densely connected deep-learning architecture and depth-first search for crack segmentation on images collected with smartphones. *Struct Health Monit* 2020; 19(6): 1726–1744.
9. Zhang L, Shen J and Zhu B. A research on an improved unet-based concrete crack detection algorithm. *Struct Health Monit* 2020; 20: 1864–1879, 1475921720940068.
10. Ronneberger O, Fischer P and Brox T. U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany, October 5-9. Springer, 2015, pp. 234–241.
11. Zhang A, Wang KCP, Li B, et al. Automated pixel-level pavement crack detection on 3d asphalt surfaces using a deep-learning network. *Computer-Aided Civil Infrastructure Eng* 2017; 32(10): 805–819.
12. Choi W and Cha Y-J. Sddnet: real-time crack segmentation. *IEEE Trans Ind Electro* 2019; 67(9): 8016–8025.
13. Kang D, Benipal SS, Gopal DL, et al. Hybrid pixel-level concrete crack segmentation and quantification across complex backgrounds using deep learning. *Automation in Construction* 2020; 118: 103291.
14. Bang S, Park S, Kim H, et al. Encoder-decoder network for pixel-level road crack detection in black-box images. *Computer-Aided Civil Infrastructure Eng* 2019; 34(8): 713–727.
15. Benz C, Debus P, Khanh Ha H, et al. Crack segmentation on uas-based imagery using transfer learning. In: 2019 International Conference on Image and Vision Computing New Zealand (IVCNZ), New Zealand, 2-4 December. IEEE, 2019, pp. 1–6.
16. Dung CV and Anh LD. Autonomous concrete crack detection using deep fully convolutional neural network. *Automation in Construction* 2019; 99: 52–58.
17. Ji A, Xue X, Wang Y, et al. An integrated approach to automatic pixel-level crack detection and quantification of asphalt pavement. *Automation in Construction* 2020; 114: 103176.
18. Jiang S and Zhang J. Real-time crack assessment using deep neural networks with wall-climbing unmanned aerial system. *Computer-Aided Civil Infrastructure Eng* 2020; 35(6): 549–564.
19. König J, Jenkins MD, Barrie P, et al. A convolutional neural network for pavement surface crack segmentation using residual connections and attention gating. In: 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, September 22-25, IEEE, 2019, pp. 1460–1464.
20. Liu Z, Cao Y, Wang Y, et al. Computer vision-based concrete crack detection using u-net fully convolutional networks. *Automation in Construction* 2019; 104: 129–139.
21. Liu Y, Yao J, Lu X, et al. Deepcrack: a deep hierarchical feature learning architecture for crack segmentation. *Neurocomputing* 2019; 338: 139–153.
22. Liu J, Yang X, Lau S, et al. Automated pavement crack detection and segmentation based on two-step convolutional neural network. *Computer-Aided Civil Infrastructure Eng* 2020; 35(11): 1291–1305.
23. Mei Q, Gül M and Azim MR. Densely connected deep neural network considering connectivity of pixels for automatic crack detection. *Automation in Construction* 2020; 110: 103018.
24. Nayyeri F, Hou L, Zhou J, et al. Foreground-background separation technique for crack detection. Computer-Aided Civil Infrastructure Eng 2019; 34(6): 457–470.

25. Ni F, Zhang J and Chen Z. Zernike-moment measurement of thin-crack width in images enabled by dual-scale deep learning. Computer-Aided Civil Infrastructure Eng 2019; 34(5): 367–384.

26. Ren Y, Huang J, Hong Z, et al. Image-based concrete crack detection in tunnels using deep fully convolutional networks. Construction Building Mater 2020; 234: 117367.

27. Tong Z, Yuan D, Gao J, et al. Pavement defect detection with fully convolutional network and an uncertainty framework. Computer-Aided Civil Infrastructure Eng 2020; 35(8): 832–849.

28. Yang F, Zhang L, Yu S, et al. Feature pyramid and hierarchical boosting network for pavement crack detection. IEEE Trans Intell Transportation Syst 2019; 21(4): 1525–1535.

29. Zhang X, Rajan D and Story B. Concrete crack detection using context-aware deep semantic segmentation network. Computer-Aided Civil Infrastructure Eng 2019; 34(11): 951–971.

30. Zhang A, Wang KCP, Fei Y, et al. Automated pixel-level pavement crack detection on 3d asphalt surfaces with a recurrent neural network. Computer-Aided Civil Infrastructure Eng 2019; 34(3): 213–229.

31. Long J, Shelhamer E and Darrell T. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Boston, USA, June 7-12, 3431–3440, 2015.

32. Oktay O, Schlemper J, Folgoc LL, et al. Attention u-net: Learning where to look for the pancreas. arXiv preprint arXiv: 1804.03999, 2018.

33. He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Las Vegas, USA, June 26-July 1, 770–778, 2016.

34. Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Honolulu, USA, July 21-26, 4700–4708, 2017.

35. Chen LC, Zhu Y, Papandreou G, et al. Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Proceedings of the European conference on computer vision (ECCV), Munich, Germany, September 8-14, 801–818, 2018.

36. Badrinarayanan V, Kendall A and Cipolla R. Segnet: a deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions Pattern Analysis Machine Intelligence 2017; 39(12): 2481–2495.

37. Ozgener FC. Concrete crack segmentation dataset. Mendeley Data 2019; 1: DOI: 10.17632/jwsv7fbrp.1.

38. Abraham N and Khan NM. A Novel Focal Tversky Loss Function with Improved Attention U-Net for Lesion Segmentation. In: 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019), Venice, Italy, April 8-11, 2019, pp. 683–687, IEEE.

39. Ramachandran P, Zoph B and Le QV. Searching for activation functions. arXiv Preprint arXiv:1710.05941, 2017.

40. Ioffe S and Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning, Lille, France, July 6-11, 448–456, 2015.

41. Avenash R and Viswanath P. Semantic segmentation of satellite images using a modified cnn with hard-swish activation function. In: VISIGRAPP (4: VISAPP), Prague, Czech Republic, February 25-27, 413–420, 2019.

42. Howard A, Sandler M, Chu G, et al. Searching for mobilnetv3. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, South Korea, October 27-November 2, 1314–1324, 2019.

43. Nair V and Hinton GE. Rectified linear units improve restricted boltzmann machines. icml 2010.

44. Hu J, Shen L and Sun G. Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Salt Lake City, USA, June 18-22, 7132–7141, 2018.

45. Courbariaux M, Bengio Y and David J-P. Binaryconnect: Training deep neural networks with binary weights during propagations. arXiv Preprint arXiv:1511.00363, 2015.

46. Paszke A, Gross S, Chintala S, et al. Automatic differentiation in pytorch. 2017.

47. Anderson A, Vasudevan A, Keane C, et al. Low-memory gemm-based convolution algorithms for deep neural networks. arXiv Preprint arXiv:1709.03395, 2017.

48. Guo J, Li Y, Lin W, et al. decoupling: from regular to depthwise separable convolutions. arXiv preprint arXiv:1808.05517, 2018.

49. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. arXiv Preprint arXiv:1706.03762, 2017.

50. Yuan Y and Wang J. Ocnet: object context network for scene parsing. arXiv preprint arXiv:1809.00916, 2018.

51. Zeng Z, Xie W, Zhang Y, et al. Ric-unet: an improved neural network based on net for nuclei segmentation in histology images. Ieee Access 2019; 7: 21420–21428.

52. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting. Journal Machine Learning Research 2014; 15(1): 1929–1958.

53. Dumoulin V and Visin F. A guide to convolution arithmetic for deep learning. arXiv Preprint arXiv:1603.07285, 2016.

54. Kuznetsova A, Rom H, Alldrin N, et al. The open images dataset v4. International Journal of Computer Vision; 128(7): 1956–1981, 2020.

55. Buslaev A, Iglovikov VI, Khvedchenya E, et al. Albu-mention: fast and flexible image augmentations. Information 2020; 11(2): 125.

56. Felzenszwalb PF and Huttenlocher DP. Efficient graph-based image segmentation. Int Journal Computer Vision 2004; 59(2): 167–181.

57. Zhang L, Yang F, Daniel Zhang Y, et al. Road crack detection using deep convolutional neural network. In: 2016 IEEE
international conference on image processing (ICIP), Phoenix, USA, September 25-28, IEEE, 2016, pp. 3708–3712.

58. Shi Y, Cui L, Qi Z, et al. Automatic road crack detection using random structured forests. *IEEE Trans Intell Transportation Syst* 2016; 17(12): 3434–3445.

59. Van Rossum G, et al. *Python*, 1991.

60. Diederik Kingma P and Ba J. Adam: a method for stochastic optimization. arXiv Preprint 2014, arXiv:1412.6980.

61. Zhou Z, Siddiquee MMR, Tajbakhsh N, et al. Unet++: re-designing skip connections to exploit multiscale features in image segmentation. *IEEE Transactions Medical Imaging* 2019; 39(6): 1856–1867.