sustainability

MDPI

Article

Deep Learning-Based Defect Detection Framework for Ultra High Resolution Images of Tunnels

Kisu Lee [1], Sanghyo Lee [2,*] and Ha Young Kim [1,*]

[1] Graduate School of Information, Yonsei University, 50 Yonsei-ro, Seodaemun-gu, Seoul 03722, Republic of Korea
[2] Division of Smart Convergence Engineering, Hanyang University ERICA, 55 Hanyangdaehak-ro, Sangnok-gu, Ansan 15588, Republic of Korea
* Correspondence: mir0903@hanyang.ac.kr (S.L.); hayoung.kim@yonsei.ac.kr (H.Y.K.); Tel.: +82-31-400-5965 (S.L.); +82-2-2123-4194 (H.Y.K.)

Abstract: This study proposes a defect detection framework to improve the performance of deep learning-based detection models for ultra-high resolution (UHR) images generated by tunnel inspection systems. Most of the scanning technologies used in tunnel inspection systems generate UHR images. Defects in real-world images, on the other hand, are noticeably smaller than the image. These characteristics make simple preprocessing applications, such as downscaling, difficult due to information loss. Additionally, when a deep learning model is trained by the UHR images under the limited computational resource for training, problems may occur, including a reduction in object detection rate, unstable training, etc. To address these problems, we propose a framework that includes preprocessing and postprocessing of UHR images related to image patches rather than focusing on deep learning models. Furthermore, it includes a method for supplementing problems according to the format of the data annotation in the preprocessing process. When the proposed framework was applied to the UHR images of a tunnel, the performance of the deep learning-based defect detection model was improved by approximately 77.19 percentage points (pp). Because the proposed framework is for general UHR images, it can effectively recognize damage to general structures other than tunnels. Thus, it is necessary to verify the applicability of the defect detection framework under various conditions in future works.

Keywords: tunnel inspection system; ultra-high resolution; defect detection; preprocessing; postprocessing; deep learning

Citation: Lee, K.; Lee, S.; Kim, H.Y. Deep Learning-Based Defect Detection Framework for Ultra High Resolution Images of Tunnels. *Sustainability* **2023**, *15*, 1292. https://doi.org/10.3390/su15021292

Academic Editors: Weiwei Lin and Jun He

Received: 2 November 2022
Revised: 5 January 2023
Accepted: 6 January 2023
Published: 10 January 2023

1. Introduction

With the rapid development of expressways and increasing highway tunnel construction, tunnel defects have become an important problem in tunnel engineering, and tunnel lining cracks, which are common defects, have adverse effects on the stress, waterproofing, and appearance of tunnel structures [1]. If various tunnel defects are not treated on time, hazards may occur, including concrete corrosion, deformation of surrounding rock, and reduced lining bearing capacity, which can adversely affect the safety of the whole tunnel during construction and operation. Thus, regular inspection is important in ensuring the safety of tunnels [2].

Tunnel inspection is traditionally performed by professional workers who manually carry instruments and are lifted close to the ceiling of the tunnel. Such traditional tunnel inspection is very time-consuming, inefficient, and highly dependent on workers' knowledge and experience. This method also disturbs normal travel and presents potential safety issues for the workers [3]. Accordingly, the modernization of the inspection, assessment, and maintenance methods of tunnels is now urgently required due to a shortage of labor and the aging of maintenance workers. To overcome the disadvantages of manual inspection, automated tunnel inspection systems using photographic equipment, photogrammetric

techniques, and computer vision techniques, such as laser beams, silt cameras, line sensor cameras, CCD (charge-coupled device) cameras, etc., as well as radar technology, have been proposed [4–6].

Efficient automatic extraction of lining cracks is a crucial step for presenting a further assessment of the tunnel lining status and performing reasonable maintenance. Therefore, there is an increasing interest in image processing techniques for the automatic extraction method [7]. In particular, owing to the recent rapid development of artificial intelligence, deep learning methods based on convolutional neural networks (CNNs) present new solutions for image processing and defect detection [8]. Studies have also been actively conducted on using CNN models for recognizing various defects that occur in tunnels [2,3,9–11]. Most of them focus on the improvement of the performance of CNN models for accurate detection of various and unclear defects that occur in structures.

In our study, because of the resolution of our input images, we focused on the framework, unlike previous studies, for better training the CNN-based detection models to improve the detection performance. Most of the scanning technologies used in tunnel inspection systems generate UHR images, and there are noticeably smaller defects than those. If deep learning-based detection models are trained with UHR images as inputs, the computational cost [12] increases dramatically with size. The higher computational cost can lead to a need for higher-capability graphics processing units (GPUs). When the models are trained on the same GPU, the huge inputs restrict the batch size used as the units for updating the model parameters, and it can cause unstable training of the models. Furthermore, when the same CNN model is used, the huge inputs lead to huge feature maps, and these can lead to the need for more parameters in the prediction heads that extract the final outputs from the feature maps. For example, let us assume that there is a prediction head that extracts one value from the feature maps. Then, the head maps the feature map to one value for the final output. The number of parameters that occurred in the mapping process is changed by the feature map size because the output size is fixed. In other words, the head with a larger feature map as an input has more parameters. This phenomenon promotes faster overfitting during model training when the number of data is the same, and it can be the reason for the decline in detection performance. In other words, if the model is trained with limited model training resources (the number of images, GPU capacity, etc.), the UHR images can disturb the training of the deep learning-based models. Thus, most deep learning-based studies for computer vision use resizing methods, such as downscaling, as preprocessing to reduce the input size of the models. However, simple downscaling [13] has a problem in that it can cause information loss when detecting the objects. Especially when the objects are much smaller than the images that contain them, information loss is more remarkable, and this causes the detection performance to decline [14].

To address these problems, unlike previous studies, we propose a defect detection framework based on "patchification" to improve the performance of a deep learning-based detection model for the UHR images. The patchification means splitting original images into smaller patch images. Our framework consists of three steps, namely preprocessing, detection, and postprocessing. The patchification is included in the preprocessing, and it can resize the UHR images to the proper size while avoiding the above problems. However, another problem arises when patchifying the bounding boxes in which the defect exists loosely. As a result, we address this with an additional step in the preprocessing. Additionally, predictions on the patchified images make it hard to see the locations of predictions on the original UHR images. The postprocessing step transforms the predictions on the patchified images into the predictions on the UHR images. Further details related to this are described in Section 3.

Finally, to verify the effect of the proposed framework, we compare the results of training the same deep learning-based detection model using the framework with those of training without it. The training of the detection model without our framework (the latter) uses a traditional downscaling method to preprocess the UHR images. For all classes in our

dataset, on average, the former's detection rate is 86.71%, outperforming the latter's result of 9.52%. These results show the effectiveness of our framework to detect small defects in UHR images. Section 4 delves deeper into the findings.

The first contribution of our framework is that it can automatically detect defects in tunnel lining. Additionally, even though the tunnel images are UHR images, the general deep learning-based detection model can be trained with limited model training resources. Furthermore, this framework has the advantage of being applicable to general UHR images rather than just tunnel images. Thus, our framework can also be used for the inspection systems for the other huge structures.

Section 2 examines the prior literature for traditional inspection methods, automated inspection methods using deep learning models, and so on. In Section 3, we propose our framework to efficiently detect the defects in the UHR images. In Section 4, we present the experimental setting and the comparison of the detection performance. In Section 5, we discuss the results of the traditional methods and the proposed methods.

2. Literature Review

Traditional non-destructive (ND) inspection methods for tunnel structures can be divided into visual, strength-based, sonic, ultrasonic, magnetic, electrical, thermography, radar, radiography, and endoscopy methods. However, tunnel inspection is performed through scheduled, periodic, and tunnel-wide visual observations by inspectors who identify structural defects and rate these defects. This process is slow, labor-intensive, and subjective, and the working environment is unpleasant due to dust, an absence of natural light, uncomfortable conditions, and even toxic substances, such as lead and asbestos [15].

For these reasons, research on automated tunnel inspection systems has received significant attention in recent years to facilitate the process of visual inspection. One of the main functions of these automated tunnel inspection systems is tunnel interior visualization. Visualization, in this context, is a means of organizing large image datasets to create a layout plan of the tunnel lining, thereby aiding inspection. Technical condition evaluation can be conducted offline and analyzed further using digital processes by the constructed models, reducing the presence of personnel in the tunnels while providing a more objective observation [4].

Studies related to tunnel interior visualization have been actively conducted. Ukai [16] developed an inspection system for tunnel walls that is capable of efficiently extracting deformations, such as cracks, from images and automatically drawing deformation charts. Attard et al. [17] proposed a novel approach for position offset correction of images taken from a moving robotic platform in tunnel environments using image mosaicing. Zhang et al. [18] presented a crack detection and classification approach for subway tunnels based on the application of CMOS (complementary metal–oxide-semiconductor) line scan cameras. Lee et al. [19] provided a simple and effective method for rectifying distorted images of a tunnel wall and established a novel image-mosaic technology to produce the layout of tunnel surfaces. Stent et al. [20] introduced a low-cost robotic system designed to enable safe, objective, and efficient visual inspection of tunnels. The system captures high-resolution images and processes them to produce maps of the tunnel linings suitable for detailed inspection. These studies aim to effectively implement tunnel-interior visualization based on large image datasets. In tunnel-interior visualization, various image datasets are used to generate UHR images, based on which the digital inspection process of the tunnel can be initiated.

For objective and efficient digital inspection of tunnels, technology to automatically detect defects from the image data acquired through the aforementioned visualization is required. To this end, studies have proposed defect detection methods for tunnels based on deep learning technology. Zhou et al. [2] proposed a new defect detection model named YOLOv4-ED. This model can implement real-time, automatic, cost-effective, and high-precision detection of tunnel lining defects. Zhao et al. [10] developed a deep learning-based approach that extends the PANet model by adding a semantic branch,

thereby refining the process of crack evaluation to reduce inaccuracies associated with crack discontinuities and image skeletonization. Liao et al. [3] presented a new mobile tunnel inspection system for capturing tunnel lining images and effectively detecting cracks. A new imaging module composed of high-resolution CCD cameras, a laser scanner, and a lighting array was designed for fast tunnel lining imaging. A lightweight CNN was developed for effective and efficient tunnel crack detection, with novel spatial constraints imposed during training to maintain the continuity of local cracks. Huang et al. [21] presented a deep learning method for the instance segmentation of cracks from shield tunnel lining images using a mask region-based convolutional neural network (Mask R-CNN) incorporated with a morphological closing operation. These studies focused on deep learning models to improve defect detection performance for tunnel interior images. However, the performance evaluation mainly included deep learning models for small images; thus, the studies have limitations in reflecting the characteristics of the UHR images generated by the tunnel interior visualization technology mentioned above.

According to the findings of studies related to deep learning models, detecting small objects, such as cracks, in UHR images may degrade the efficiency and performance of deep learning models. Růžička et al. [13] proposed an attention pipeline method that uses a two-stage evaluation of each image or video frame under rough and refined resolution to limit the total number of necessary evaluations. They highlighted that the downscaling of UHR images degraded the detection performance and adopted a method of dividing images to address this problem. Liu et al. [14] proposed the high-resolution detection network (HRDNet) to effectively detect small objects in high-resolution images. Zhang et al. [12] proposed a new module called Pre-Locate Net, which is a plug-and-play structure that can be combined with the most popular detectors. They highlighted that the computational cost increased with the resolution of images.

To this end, this study proposes a defect detection framework for tunnels composed of a UHR image preprocessing phase, a defect detection phase, and a postprocessing phase to improve the defect detection performance of deep learning models.

3. Research Framework

The dataset used in this study to detect the defects in the tunnel lining consisted of 1120 UHR images with annotations of the defects included in each image. We determined the classes of defects in the entire dataset to be "crack", "crazing", and "delamination", which are the most frequent types of defects in real-world datasets. As shown in the pie chart in Figure 1, there is a data imbalance for each class. For this problem, we applied the BoxAug method [22] when we trained a deep learning-based model, and the details are described in Section 4.

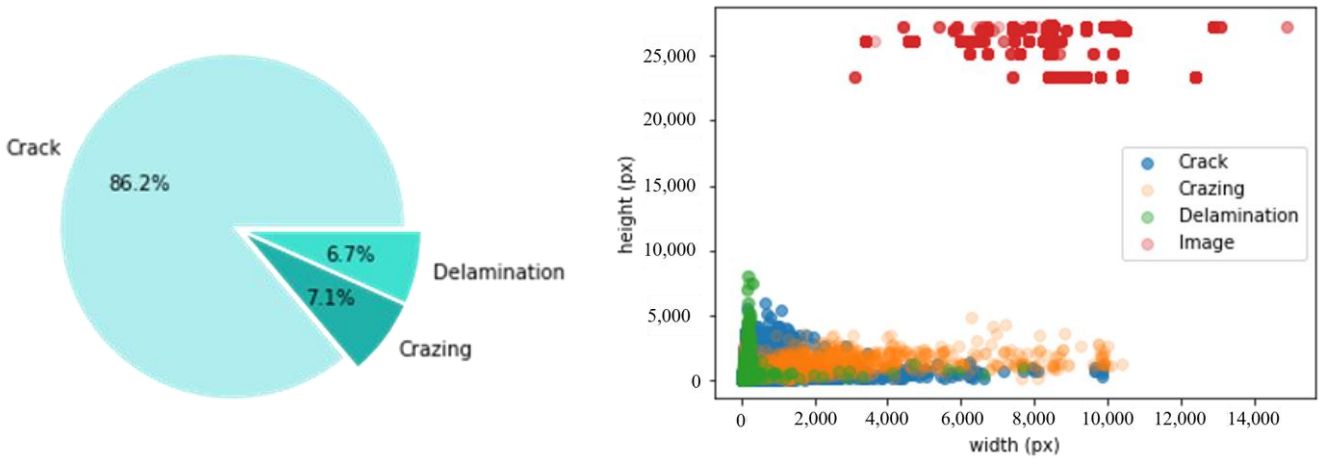

Figure 1. Distribution of the number of defects (**left**). Distributions of the UHR images and defect size (**right**).

The scatter plot in Figure 1 describes the distribution of width and height for the original UHR images and for the bounding boxes of the defects in each class. The average width and height of the UHR images are approximately 9085 and 25,555 pixels, respectively. Given that ImageNet [23], a representative image dataset for deep learning-based models, has a size of 224 × 224 pixels, it can be confirmed that the resolution of the original images is too high. Accordingly, it is necessary to transform the deep learning-based detection model into a size suitable for learning through preprocessing of the original images.

However, from the size distribution diagram in Figure 1, it can be observed that the size of each class is noticeably smaller than that of the original image. Numerically, the average size of the bounding boxes for each class of defect is 361 × 661 (crack), 2210 × 1296 (crazing), and 198 × 765 (delamination) pixels. The difference in size between the object and the image containing the object makes the object detection task more challenging because information is lost during the preprocessing of the images. Additionally, as shown in Figure 2, there is the characteristic that defects, such as cracks and crazing, exist linearly in the bounding boxes. Crazing, in particular, has larger bounding boxes than other classes, but it has a much thinner line than a crack. This characteristic makes it more likely to lose information about the object when downscaling methods, such as image resizing, are used.

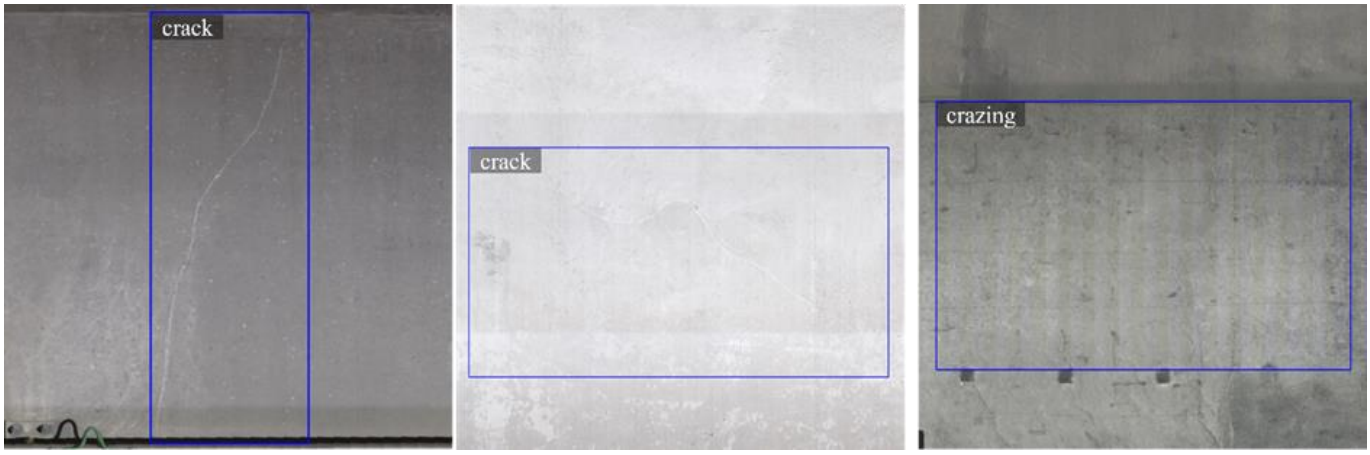

Figure 2. Crack and crazing samples.

As mentioned above and in Section 1, to avoid the problems related to the size of input images, we propose our framework, based on patchification, consisting of preprocessing, detection, and postprocessing, as shown in Figure 3. Our framework operates differently in the training and testing phases for each step. The training phase is for effectively training a deep learning-based model in the detection step with patchified images created by the preprocessing. As such, in this phase, postprocessing is not used. The testing phase is for predicting the location and class of defects in the UHR images. In the testing phase, the model to predict those is the same as the trained model with the patchified images in the training phase. Thus, in the testing phase, all three steps are used. Details about each step in our framework are described in detail in the next subsections.

3.1. Preprocessing

There are several preprocessing methods for using UHR images as input for DCNN (n convolutional neural Network)-based complex detection models, including resizing the image itself [13,14,24]. In this study, a method of splitting UHR images into patches of appropriate size was used to minimize information loss. The preprocessing method was determined according to two typical features of the data. One characteristic is that the difference between the resolution of the input images and the size of defects was conspicuous, and another is that the types of defects to be detected included cracks and crazing. Objects, such as cracks and crazing, exist in a form similar to lines in the area, even if an annotation is given as a bounding box. If such objects use resize-based image

preprocessing, it is highly likely that their shape will be lost in an image within the bounding box area. That is, information that is important for detecting the object will be lost. In contrast, patchification can adjust the size of the input image of the detection model while retaining the information about the object in the original images to be detected.

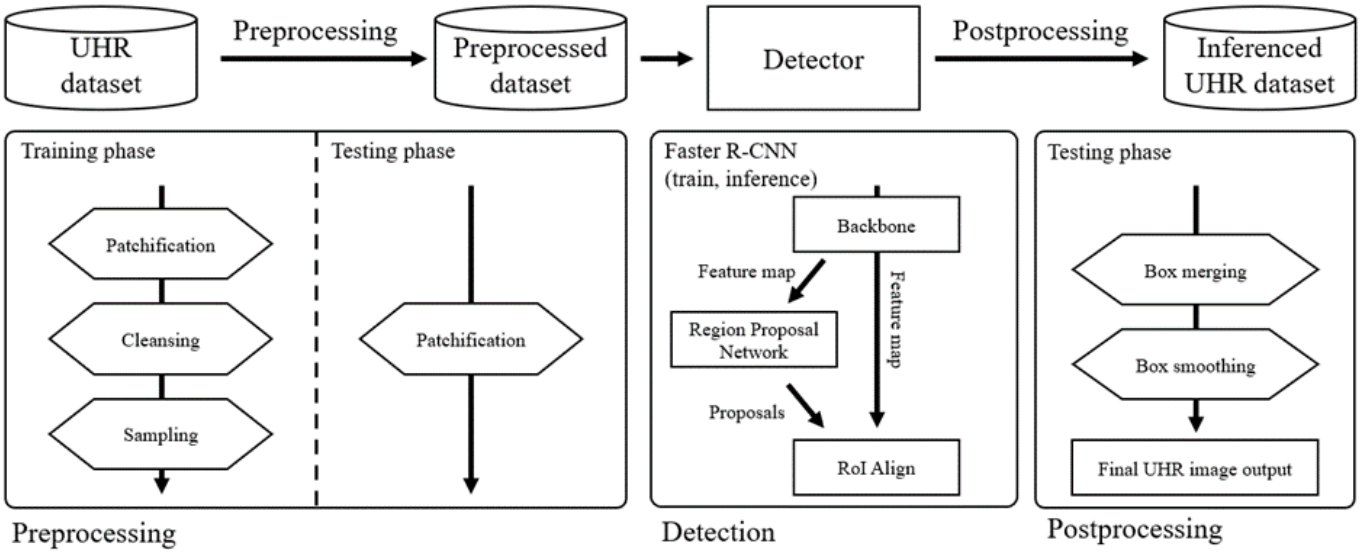

Figure 3. Research framework.

After the patchifying process, bounding boxes given as annotations for objects are also divided into patches. As can be seen in Figure 4, in this case, if an object, such as a crack, exists loosely in the bounding box, a defect in the divided bounding box may not exist. We called this bounding box an error sample. Using all annotations, including these error samples, for training detection models can cause confusion in the training process because of the error samples. Therefore, an additional data cleansing process is necessary to exclude error samples, such as the yellow boxes in Figure 4. At this time, because conducting the data cleansing process manually is cumbersome and time-consuming, it is efficient to exclude error samples from training according to cleansing rules.

The most common cleansing rules can be set according to two rules. In the first rule, error samples share the edge of each patch image. The second rule is the filtering of error samples by using thresholds for their width and height.

Most error samples occur when the existing bounding box is split by the patch. This implies that error samples share the edge with the patch image they contain; therefore, the first rule is self-evident. To verify the first rule, some random sampling was performed on a 2000 $\times$ 2000 patch dataset as an example, and then error samples were selected within the partial dataset. Of the 2729 samples in the partial dataset, the error samples accounted for 343 of the samples, and all error samples satisfied the first cleansing rule.

Then, we identified the occurrence location and size of the error samples to determine the validity of the second rule. The error samples can exist in a total of eight cases within each patch image. Figure 5 describes the size distribution of the error samples at each location. As can be seen in Figure 5, the error samples have a very small height or width. In addition, when the error samples share only one edge with the patch images, the characteristics are remarkable. As a result, a cleansing rule can be established for each case by adjusting the width and height thresholds. In the left- and right-center cases in Figure 5, we can use just the width threshold to filter. Similarly, in the top- and bottom-center cases in Figure 5, just the height threshold can filter the error samples. Thus, we classify sample cases into those where the minimum x-coordinate of the box is 0 or the maximum x-coordinate of the box is patch size, and those where the minimum y-coordinate is 0 or the maximum y-coordinate of the box is patch size. Then, we filter the first two cases using the

width threshold and the other cases using the height threshold. To verify the second rule, we filter the samples based on the thresholds for height and width in the same environment as the verification environment for the first rule. In the 2000 × 2000 patch dataset case with 343 error samples, if the width and height thresholds are set to 200 and 150, respectively, 340 error samples (99.12%) can be filtered. This means that the second rule of cleansing with proper thresholds can filter out almost error samples in dataset.

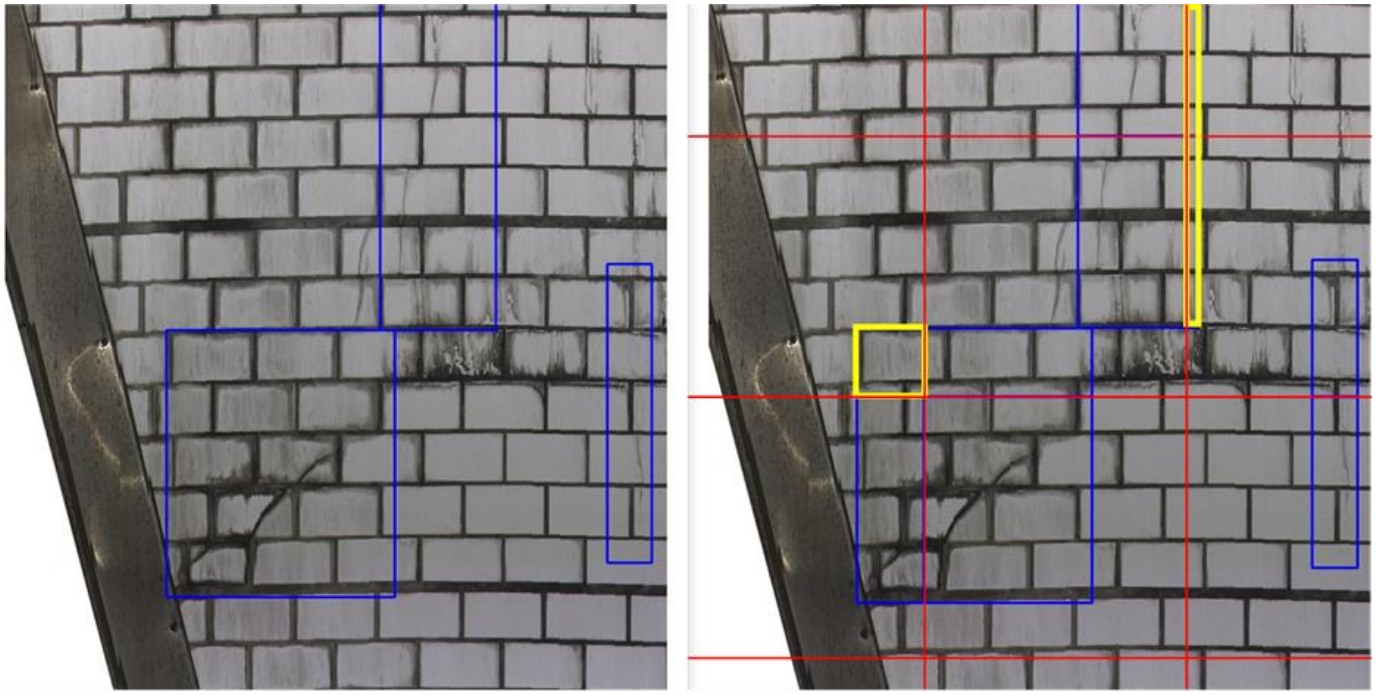

Figure 4. Patch examples for UHR images (the **left** image is the part of the original UHR image with blue ground-truth boxes. The yellow boxes in the **right** image are the error samples caused by the patchification for bounding box annotations. The red lines in the right image are the guideline for the patchification).

However, it should be noted that samples, not error samples, are also filtered by thresholding to preserve the data required for model learning. For example, if the width threshold is set to 200 for the left-center case (36 samples) in Figure 5, the error samples are all excluded from the training. However, if the threshold is applied to non-error samples (2386 samples), approximately 50% (1199 samples) of the training samples are discarded. Similarly, if the width threshold is set to 100, approximately 80% (1885 samples) of the non-error samples can be preserved, excluding approximately 78% (20 samples) of the error samples. Because such a trade-off occurs, it is important to set an appropriate threshold. Additionally, the appropriate threshold value may vary depending on the size of the patch. However, by checking the width and height distributions of error samples in 2000 × 2000 patches, a range of approximate threshold values can be set, so even patches of other sizes can determine appropriate threshold values through experiments based on this threshold; in this study, this threshold was determined through experiments.

The patchification of preprocessing is used in both the training and testing phases. However, because the cleansing process assists in detection model training, it is not used in the testing phase. In addition, only patches with defects are used as the training dataset in the training phase. The testing phase, on the other hand, uses all patches from the original images because it only advances the inference. In other words, our preprocessing consists of patchification, the cleansing process, and sampling for training the detection model at the training phase. Furthermore, at the testing phase, our preprocessing is only patchification.

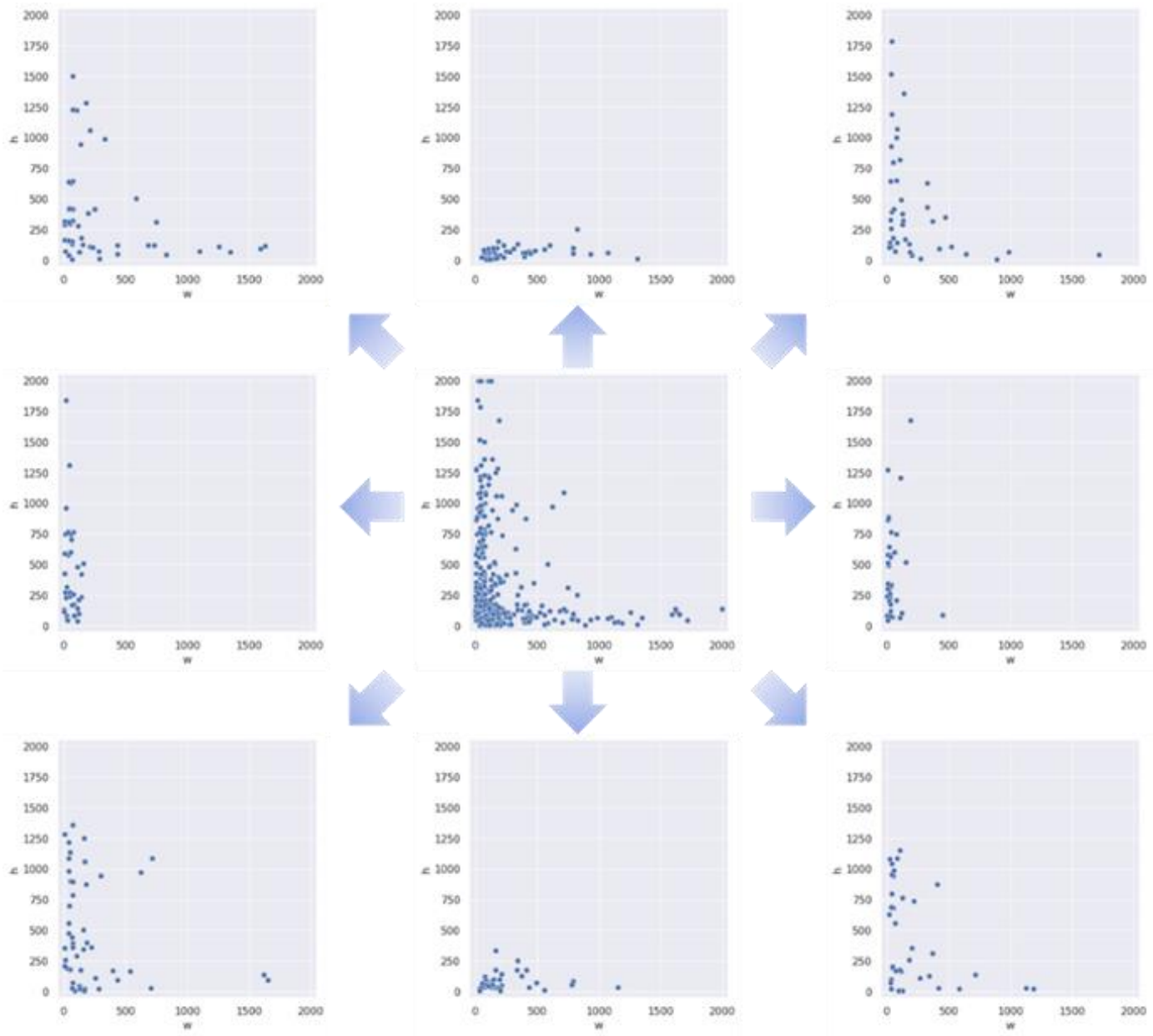

Figure 5. Size distribution of error samples at each shared edge location within each patch image (center—size distribution of all error samples; x-axis—width; and y-axis—height).

3.2. Detection

The detection step is applied to the detector's training and inference processes to detect defects in the input patches of the training dataset and the test dataset, respectively. The faster region-based convolutional neural network (Faster R-CNN) [25], the detector used in this study, is a representative two-stage detector and has been applied to detect objects in images in various fields depending on their learning stability and detection performance.

The Faster R-CNN consists of the backbone network, the region proposal network (RPN), and the region of interest (RoI) head. The backbone network is used to extract features from the input images. There are several architectures for the backbone network, such as ResNet [26], VGG [27], MobileNet [28], etc. We selected ResNet50, which is the representative feature extractor, for the backbone network. Extracted features by the backbone network are used for inputs to the RPN and the RoI head.

The RPN, which is the biggest characteristic of Faster R-CNN, is the small network for predictions of the location and objectness of objects from feature maps extracted by the backbone network. The RPN uses the spatial window of the feature map as an input to the small network and slides the window to extract the lower-dimensional features from each location. In this case, in each window, multiple region proposals are simultaneously extracted using anchor boxes according to the set size and ratio. The extracted proposals calculate the regression and classification losses and the corresponding location and objectness, respectively.

After aligning the RoI (RoI align), we calculate the regression and classification losses by the final outputs as in the RPN. The classification loss in the RoI head differs from the loss in the RPN in the classes to classify. Note that the classification loss in the RPN classifies the presence or absence of an object in the predicted location, while the classification loss in the RoI head classifies its type. These four losses in the RPN and the RoI head train Faster R-CNN. This process is briefly described in the detector part of Figure 3.

When training a detector using UHR images, the number of parameters increases significantly in the detector head for final prediction compared to otherwise. In order to supplement the learning problem according to the increase in the number of parameters, traditional preprocessing, such as downscaling, has been conducted in previous studies. As mentioned above, our framework has proceeded with different preprocessing for different problems. For better detection performance, the detector uses only patches with defects after the preprocessing in the training phase. As with training, the inference is performed on patches rather than original UHR images. This implies that the final detection performance in the UHR image cannot be confirmed only by the output of the detector's inference. Therefore, in this study, the inference result for the final UHR images is derived through postprocessing using the inference for patches.

3.3. Postprocessing

As the detection model predicts the location and class of the objects for the patch images, the detection performance in the original UHR cannot be simply calculated. Thus, we propose postprocessing in this study, which is a process to transform the results on the patch images to the results on the original UHR images. The first process is merging patch images and bounding box coordinates that have been inferred by the detection model into the same form as the original UHR images. However, the merged bounding box predictions can indicate one ground-truth box rather than multiple ground-truth boxes, because patchification in preprocessing can split not only the original images but also the bounding boxes with detection targets. Measuring the final performance by conducting the inference in a patch-split state is not the same as measuring the detection performance in the original UHR images, making the performance comparison with the existing methodology unfair. Depending on these factors, the "box smoothing" method is applied as the final step of the postprocessing to measure the final performance of the detection model.

As shown in Figure 6, the process of the box smoothing method is very simple. After merging the predictions, if the pair of merged predictions satisfy our smoothing condition, the pair is smoothed to one prediction. The first condition for smoothing is that the pair of predictions belong to the same class. The next condition is that the distance between the bounding boxes does not reach a certain level (threshold). Herein, the distance between bounding boxes is set to the minimum Euclidean distance between the vertex coordinates of the bounding boxes. To calculate the distance, we extract the vertex coordinates of the pair of predicted bounding boxes from the merged image. Subsequently, we construct the pairs of vertex coordinates for each bounding box and examine the distances for each pair of coordinates. Then, the minimum of the distance of the vertex coordinates is set to the distance of the bounding boxes. Finally, if the two bounding boxes satisfy the smoothing conditions, they are considered to represent the same object in the original image and are made into one bounding box (smoothed box). Making the pair of boxes satisfying the smoothing conditions into one bounding box is very simple. Commonly, the location of a

bounding box is represented by minimum coordinates (x1, y1) and maximum coordinates (x2, y2). When a pair of bounding boxes is represented by (x1a, y1a, x2a, y2a), (x1b, y1b, x2b, y2b), the smoothed box is defined by (min(x1a, x1b), min(y1a, y1b), max(x2a, x2b), max(y2a, y2b)).

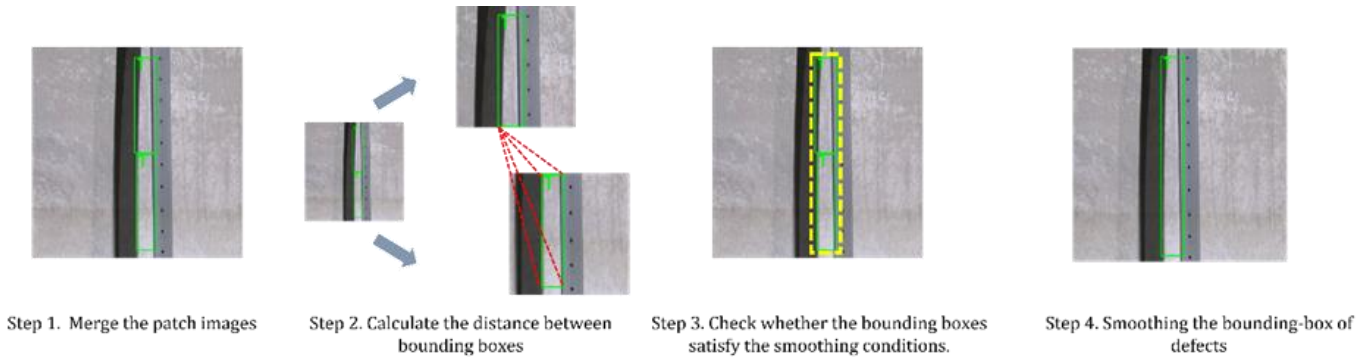

Figure 6. Steps of the box smoothing method.

This process is repeated until the bounding boxes can no longer be combined (until there are no boxes that satisfy the smoothing condition). For a fair comparison of the detection performance, this process is applied to the patchified ground-truth boxes as well as the predicted boxes, and the same threshold is used in the box smoothing method. Additionally, as mentioned above, this postprocessing is used to measure the final detection performance on the UHR images and is, thus, only used in the testing phase.

4. Experiments and Results

Our goal is to detect the defects in the huge tunnel lining using our framework for the UHR tunnel dataset. In this section, we present an analysis of the results of detecting defects in the UHR images of the tunnels using our framework. The entire data distribution for each dataset used to train the detector is described in Table 1. As mentioned in Section 3, the "Objects" row in Table 1 represents the sum of the rows below, since the types of objects to detect present in the tunnel image are set to "Crack", "Crazing", and "Delamination". First, the "Original" column in Table 1 presents the distribution of the number of images and defects by type in our original UHR images dataset. There was a deep learning-based detection model (the detector) and several hyper-parameters to set for the application of our framework, as mentioned in Section 3. As mentioned in Section 3.2, we selected the Faster R-CNN [25] for the detector. In addition, based on the performance of the detector in experiments of various sizes, we determined that the patch size for the patchification step in preprocessing was 500. When we patchify the UHR images to the target resolution (500×500), smaller patch images than the target resolution can be generated because the width and height of the UHR images are not exact multiples of 500. For these patch images, we extended the images to the target resolution with zero (known as "zero padding"). Because the patchification step is for both the training and testing phases, we patchified all of the images in our UHR dataset and then split it into training, validation, and test datasets for training and validating the detector. Note that our goal is to detect the defects in the UHR images. This means that the training and test datasets should be divided not only in terms of the patch images but also in terms of the UHR images, which include the patch images. As a result, we divided the original patch dataset into training, validation, and test datasets, with a 7:1:2 ratio for each dataset in consideration of the above. As shown in the "Training" column in Table 1, the dataset is unbalanced according to the class of the object. Because the data imbalance can cause biased training of the detector, we used the BoxAug [22] method to counteract this problem. In other words, our final training dataset is described as the "BoxAug" column in Table 1. Note that BoxAug should not be applied to the validation and test datasets because BoxAug is included in the data

augmentation techniques. Because the validation and test datasets are intended to measure the generalization performance of the detection model in the real dataset, they should not be manipulated by the data augmentation techniques.

Table 1. Data distribution.

The Number of	Dataset after the Patchification and Cleansing					Original Dataset
	Training	BoxAug	Validation	Test	Total	
Images	30,045	60,090	4344	8636	43,025	1120
Objects (Total)	31,733	70,369	4630	9053	45,416	15,831
Crack	17,773	23,637	2716	5456	25,945	13,643
Crazing	12,502	38,587	1716	3198	17,416	1126
Delamination	1458	8145	198	399	2055	1062

Model training was conducted using one GeForce RTX 2080ti, and the batch size was set to 8 according to the GPU's capacity (11,019 MiB). An optimal model for defect detection was established using the early stopping technique [29]. We conducted an inference for patches existing in the test dataset using a detector that has been trained. After the inference in each patch image, postprocessing was performed, as described in Section 3.3. Figure 7 is an example of a box smoothing result when the threshold is set to 50 during the postprocessing of the ground-truth and the prediction boxes.

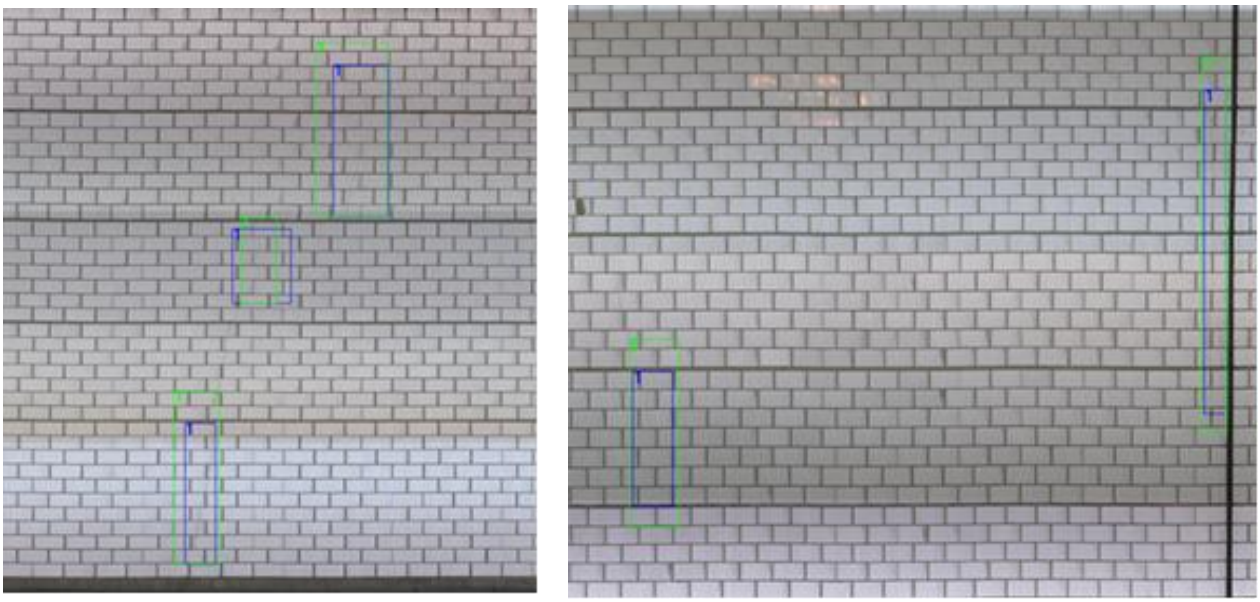

Figure 7. Parts of the final detection output (green boxes are smoothed prediction boxes; blue boxes are smoothed ground-truth boxes; and number is the class id of the defects in the box).

After the above setup, we compared the detection performance of the traditional detector training method and our detector training method (using our framework) on the same UHR image dataset. The methods used the same detector (the Faster R-CNN) to test the effectiveness of our framework. We selected the downscaling method over the traditional method to train the detector. The outputs of the detector trained by the traditional method are represented in the UHR images, unlike the detector in our framework. Thus, our detection performance is derived by postprocessing in our framework. Then, we calculated the detection rate with predictions on the UHR images for each method, and the comparison of the final detection performances is presented in Table 2. The "w/framework" row in Table 2 described the detection performance for the outputs generated by our framework.

In Table 2, the "w/o framework" row described the detection performance for the outputs produced by the detector trained using the traditional method. The "Objects" column in Table 2 indicated the detection performance for the objectness regardless of the class of the defects detected by the detector.

Table 2. Comparison of final detection performance results with and without our framework.

Category	Detection Rate				
	Objects	**Crack**	**Crazing**	**Delamination**	**Average**
w/framework *	81.82%	80.53%	96.73%	82.86%	86.71%
w/o framework **	9.32%	8.88%	16.09%	3.60%	9.52%

* w/framework—the detection performance of the detector trained with our framework; ** w/o framework—the detection performance of the detector trained with the traditional preprocessing method (downscaling).

As shown in Table 2, the detection rate when using the proposed framework has a noticeable improvement compared to the detection rate when not using the framework. The difference in the two results is caused by just how to address the UHR images. In other words, the low detection performance in detecting the defects in the UHR images with the traditional method implies that there is a critical loss of information while addressing the UHR images with the traditional method. Unlike the traditional method, our framework allows the detector to be learned by enlarging a small portion of the original UHR image without lowering the resolution. The high detection performance of "w/framework" implies that patchification-based preprocessing can prevent information loss.

The delamination occupies the smallest area in the UHR images, as mentioned in Section 3. This means there is the highest probability of information loss for the defects when the downscaling is applied to preprocess the UHR images. Our framework's ability to prevent information loss can be seen as an improvement in performance in detecting delamination distributed in the smallest size among the types of defects. The detection performance improved on not only the delamination analysis but also the analysis of the other defects.

5. Discussion

As mentioned in Section 2, there are previous studies [12–14] with similar goals to our framework. The main difference is the way to detect small objects in the image dataset used as the input dataset for the deep learning-based models. They tried to detect small objects in the image by transforming the model itself into a pyramid structure. They used 4K or 8K video [14] and 1920 × 1080 image datasets [12]. First of all, the resolution of each dataset differs greatly from ours. Therefore, in these studies [12,14], preprocessing, such as downscaling, which was eventually used, was included. Therefore, it is difficult to apply our dataset to the studies according to the contents mentioned in Section 3. The method proposed by Růžička et al. [13] uses crop-based processing similar to our method. However, unlike us, the object exists compactly in the bounding box. These differences cause the same problems as those presented in Section 3.1 when applying our dataset to their pipelines. To prevent this, we added a cleaning step to the preprocessing stage and, when applied, the detection performance at each patch increased by about 10 pp compared to when it was not. According to these points, we can confirm the contribution of our framework to UHR images.

However, the number of annotations for cracks is significantly larger compared to that of defects in other classes, but the performance improvement is less than that of the other classes. It is believed that this phenomenon occurs because the class from which the most error samples are derived in the patchification process is the crack. Defects in classes other than cracks occupy a sufficient area within the bounding box, but cracks do not. Although we worked hard to prevent the error samples that cause confusion for the detector during the preprocessing step, we were unable to completely eliminate the error

samples. Furthermore, the cleansing step has other constraints that remove even non-error samples. Because these limitations reduce the detection performance of the defects in the UHR images, the related steps must be improved in future works.

Additionally, although UHR images for tunnel images were used in this study, there is an advantage in that this framework can be used sufficiently for defect detection in other domains, since our framework is for general UHR images. However, these limitations are caused by the format of the annotations for the cracks, and so these must be considered for the UHR images of other domains, not just for tunnel images.

6. Conclusions

This study proposes a defect detection framework to improve the detection performance of deep learning models for UHR images generated by tunnel inspection systems. The proposed framework comprehensively includes preprocessing and postprocessing of UHR images, considering their characteristics, rather than focusing on deep learning models. For tunnels, mobile inspection systems have been used. Such systems, which are equipped with a number of cameras and scanners, capture images according to the tunnel geometry and create UHR images by matching them. They also convert UHR images into digital information for objective and efficient defect detection in tunnels. When deep learning models are applied to UHR images, it is necessary to rescale the images because the computational cost sharply increases. However, with the traditional methods, such as downscaling, detection performance decreases because the information about defects included in UHR images can be lost. Considering these problems, this study adopted the method of dividing UHR images into patches of an appropriate size rather than downscaling them in the preprocessing phase. Based on this, our framework adjusts the input image of the deep learning model used as a detector while retaining the object information in UHR images. In the detection phase, the Faster R-CNN model, a representative deep learning model, was applied to the divided patch images. Finally, in the postprocessing phase, the divided patch images were restored to the original UHR images using the "box smoothing" method. We compared the detection performance of the detection model trained by our framework with the performance of the detection model trained by the traditional method (downscaling). It was confirmed that this process improved the performance of the deep learning-based defect detection model by 77.19 pp, from 9.52% to 86.71%, while maintaining the information about the defects in the UHR images.

The proposed framework may also be significant from a practical perspective. In general, studies that applied deep learning models to facility inspection systems focused on detection; however, studies that apply them to damage levels, such as crack width, are gradually increasing. Generally, the damage level of a structure due to defects is automated by calculating the number of pixels that correspond to the damage. Thus, it is necessary to increase the image resolution, because sufficient pixels must be allocated to relatively tiny damage. Moreover, in the case of unmanned aerial vehicles (UAVs) that have been used for facility safety inspections, a certain separation distance from the target structure is maintained by the UAV, considering flight stability and imaging efficiency rather than capturing images of local areas. Thus, high-resolution images are generated for relatively wide structures. Therefore, the proposed framework can also be applied to identify the damage levels of various structures from a practical perspective.

However, for structures, such as bridges and buildings, there are various environmental conditions, including weather, shadows, and distance. Such conditions may affect the division and matching of images and the detection performance of deep learning models. Therefore, it is necessary to verify the applicability of the proposed defect detection framework under various environmental conditions.

Author Contributions: K.L. developed the concept and drafted the manuscript; S.L. and H.Y.K. supervised the overall work. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by a grant (21CTAP-C163951-01) from the Technology Advancement Research Program (TARP) funded by the Ministry of Land, Infrastructure and Transport of Korean government.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the Ministry of Land, Infrastructure and Transport of the Korean government for funding this research project.

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

References

1. Zhang, N.; Zhu, X.; Ren, Y. Analysis and study on crack characteristics of highway tunnel lining. *Civ. Eng. J.* **2019**, *5*, 1119–1123. [CrossRef]
2. Zhou, Z.; Zhang, J.; Gong, C. Automatic detection method of tunnel lining multi-defects via an enhanced You Only Look Once network. *Comput.-Aided Civ. Infrastruct. Eng.* **2022**, *37*, 762–780. [CrossRef]
3. Liao, J.; Yue, Y.; Zhang, D.; Tu, W.; Cao, R.; Zou, Q.; Li, Q. Automatic tunnel crack inspection using an efficient mobile imaging module and a lightweight CNN. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 15190–15203. [CrossRef]
4. Attard, L.; Debono, C.J.; Valentino, G.; Castro, M.D. Tunnel inspection using photogrammetric techniques and image processing: A review. *ISPRS J. Photogramm. Remote Sens.* **2018**, *144*, 180–188. [CrossRef]
5. Davis, A.G.; Lim, M.K.; Petersen, C.G. Rapid and economical evaluation of concrete tunnel linings with impulse response and impulse radar non-destructive methods. *NDT E Int.* **2005**, *38*, 181–186. [CrossRef]
6. Yoon, J.S.; Sagong, M.; Lee, J.S.; Lee, K.S. Feature extraction of a concrete tunnel liner from 3D laser scanning data. *NDT E Int.* **2009**, *42*, 97–105. [CrossRef]
7. Jiang, Y.; Zhang, X.; Taniguchi, T. Quantitative condition inspection and assessment of tunnel lining. *Autom. Constr.* **2019**, *102*, 258–269. [CrossRef]
8. Yang, S.; Wang, Z.; Wang, J.; Cohn, A.G.; Zhang, J.; Peng, J.; Nie, L.; Sui, Q. Defect segmentation: Mapping tunnel lining internal defects with ground penetrating radar data using a convolutional neural network. *Constr. Build. Mater.* **2022**, *319*, 125658. [CrossRef]
9. Xue, Y.; Li, Y. A fast detection method via region-based fully convolutional neural networks for shield tunnel lining defects. *Comput.-Aided Civ. Infrastruct. Eng.* **2018**, *33*, 638–654. [CrossRef]
10. Zhao, S.; Zhang, D.; Xue, Y.; Zhou, M.; Huang, H. A deep learning-based approach for refined crack evaluation from shield tunnel lining images. *Autom. Constr.* **2021**, *132*, 103934. [CrossRef]
11. Xue, Y.; Jia, F.; Cai, X.; Shadabfar, M.; Huang, H. An optimization strategy to improve the deep learning-based recognition model of leakage in shield tunnels. *Comput.-Aided Civ. Infrastruct. Eng.* **2021**, *37*, 386–402. [CrossRef]
12. Zhang, Y.; Xu, T.B.; Wei, Z. Pre-locate net for object detection in high-resolution images. *Chin. J. Aeronaut.* **2022**, *35*, 313–325. [CrossRef]
13. Růžička, V.; Franchetti, F. Fast and accurate object detection in high resolution 4K and 8K video using GPUs. *arXiv* **2018**, arXiv:1810.10551.
14. Liu, Z.; Gao, G.; Sun, L.; Fang, Z. HRDNet: High-resolution detection network for small objects. *arXiv* **2020**, arXiv:2006.07607.
15. Montero, R.; Victores, J.G.; Martinez, S.; Jardon, A.; Balaguer, C. Past, present and future of robotic tunnel inspection. *Autom. Constr.* **2015**, *59*, 99–112. [CrossRef]
16. Ukai, M. Advanced inspection system of tunnel wall deformation using image processing. *Q. Rep. RTRI* **2007**, *48*, 94–98. [CrossRef]
17. Attard, L.; Debono, C.J.; Valentino, G.; Castro, M.D. Image mosaicing of tunnel wall images using high level features. In Proceedings of the 10th International Symposium on Image and Signal Processing and Analysis, Ljubljana, Slovenia, 18–20 September 2017; pp. 141–146. [CrossRef]
18. Zhang, W.; Zhang, Z.; Qi, D.; Liu, Y. Automatic crack detection and classification method for subway tunnel safety monitoring. *Sensors* **2014**, *14*, 19307–19328. [CrossRef] [PubMed]
19. Lee, C.H.; Chiu, Y.C.; Wang, T.T.; Huang, T.H. Application and validation of simple image-mosaic technology for interpreting cracks on tunnel lining. *Tunn. Undergr. Space Technol.* **2013**, *34*, 61–72. [CrossRef]
20. Stent, S.; Girerd, C.; Long, P.; Cipolla, R. A low-cost robotic system for the efficient visual inspection of tunnels. In Proceedings of the 32nd International Symposium on Automation and Robotics in Construction, ISARC, Oulu, Finland, 15–18 June 2015; pp. 1–8. [CrossRef]
21. Huang, H.; Zhao, S.; Zhang, D.; Chen, J. Deep learning-based instance segmentation of cracks from shield tunnel lining images. *Struct. Infrastruct. Eng.* **2022**, *18*, 183–196. [CrossRef]

22. Lee, K.; Lee, S.; Kim, H.Y. Bounding-box object augmentation with random transformations for automated defect detection in residential building façades. *Autom. Constr.* **2022**, *135*, 104138. [CrossRef]
23. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255. [CrossRef]
24. Ahlswede, S.; Asam, S.; Röder, A. Hedgerow object detection in very high-resolution satellite images using convolutional neural networks. *J. Appl. Remote Sens.* **2021**, *15*, 018501. [CrossRef]
25. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv* **2015**, arXiv:1506.01497. [CrossRef] [PubMed]
26. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
27. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
28. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
29. Caruana, R.; Lawrence, S.; Giles, C.L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems*; The MIT Press: Cambridge, MA, USA, 2001; pp. 402–408.