Hindawi
Advances in Multimedia
Volume 2022, Article ID 3712289, 12 pages
https://doi.org/10.1155/2022/3712289

Research Article

Downstream Semantic Segmentation Model for Low-Level Surface Crack Detection

Thitirat Siriborvornratanakul

Graduate School of Applied Statistics, National Institute of Development Administration, 148 Serithai Road, Klong-Chan, Bangkapi, Bangkok 10240, Thailand

Correspondence should be addressed to Thitirat Siriborvornratanakul; thitirat@as.nida.ac.th

Received 30 January 2022; Revised 26 April 2022; Accepted 5 May 2022; Published 29 May 2022

Academic Editor: Deepu Rajan

As surface crack detection is essential for roads and other building structures in most countries, this has been a very popular topic in computer vision for automating structural health monitoring. Recently, many deep learning engineers have attempted to find solutions to the problem. However, to the best of our knowledge, most previous methods were about designing and experimenting with a deep learning model from scratch, which is highly technical and very time-consuming. This study proposes a new approach of using downstream models to accelerate the development of deep learning models for pixel-level crack detection. An off-the-shelf semantic segmentation model named DeepLabV3-ResNet101 is used as a base model and then experimented with different loss functions and training strategies. Our experimental results have revealed that the downstream models trained by the classic cross-entropy loss function cannot provide reasonable results in pixel-level crack detection. The most successful downstream model we found is trained by the focal loss function without using the pretrained weights that are accompanied by the base model. Our selected downstream model is generalized well across different test datasets and yields the optimal dataset scale F-measures of 84.49% on CrackTree260, 80.29% on CRKWH100, 72.55% on CrackLS315, and 75.72% on Stone331.

1. Introduction

Crack is an early form of surface damage that can be easily detected or recognized by visual inspection, both manually by human experts and automatically by computer vision algorithms. For low-level surface analysis tasks, it is non-trivial to get pixel-perfect detection results regarding very fine and thin cracks because unpredictable grains and textures of the surface can be extremely confused as being crack pixels. When looking for cracks in an input image, we can consider the problem from either high-level or low-level perspectives. Although both perspectives result in localizing cracks (if any) in an input image, the high-level approaches tend to perform the high-level scene understanding and localize cracks in an approximate bounding-box manner, whereas the low-level approaches prefer localizing cracks in an accurate pixel-level manner. Because analysis landscapes are different in the two types of approaches, it is necessary to choose the right level of problem consideration. If an input image contains not only road surface but also other high-level objects like building, sky, tree, traffic cone, car, and so forth, the high-level approaches should be applied first to localize cracks in this complicated input image. After that, if a more accurate pixel-level localization is required, the low-level approach should be applied.

Before [1] first applied deep learning to road crack detection in 2016, most works in vision-based low-level crack detection relied on handcrafted feature methods and were not able to reach a pure 2D vision-based solution that generalized well to most road surfaces. This is because actual crack pixels have very similar visual characteristics compared to other unpredictable noisy pixels like road textures/grains and shadows. Since AlexNet [2] successfully debuted deep learning to computer vision communities in 2012, there have been many works in vision-based crack detection [1, 3, 4] which experimentally confirmed the superiority of deep learning solutions over traditional vision-based solutions as well as traditional machine learning solutions. Accordingly, this paper will not focus on classical handcrafted-feature solutions and traditional machine learning solutions. For the rest of this paper, Section 2 explores the overall landscape of deep-learning-based crack detection that includes all available deep learning solutions for crack detection, both high and low levels. Section 3 explains our proposed method of using a pretrained model with a special loss function for severely imbalanced data in order to yield a good low-level vision-based crack detector. Experimental results are shown and discussed in Section 4 before we conclude this paper in Section 5. Note that this paper is different from our previous works [5, 6] regarding vision-based road damage detection. While this paper's focus is on deep-learning-based solutions, our previous work in [5] focused on vision-based pothole detection using handcrafted features. As for our previous work in [6], it proposed a deep learning model for pixel-level crack detection like this paper. However, the previously proposed model was designed and trained from scratch with no experimental concern regarding downstream tasks.

Apart from our detailed literature review of deep-learning-based crack detection in Section 2, the contributions of this paper include the following:

(1) We propose a new study regarding how to properly use an existing off-the-shelf high-level semantic segmentation model architecture (as officially available in a main deep learning framework like PyTorch) for a problem with very different and unique object characteristics of low-level crack segmentation.

(2) We experimentally confirm that building a downstream model upon the state-of-the-art semantic segmentation model named DeepLabV3-ResNet101 is completely failed for low-level crack detection when the classic cross-entropy loss function is used.

(3) We discover that a downstream model is successfully trained upon DeepLabV3-ResNet101 if the focal loss function [7, 8] is used.

(4) Because of unique visual characteristics of cracks, we discover that training DeepLabV3-ResNet101 from initially random weights shows more consistent performances across different test sets than training it from existing weights (as pretrained on a subset of COCO train2017).

Despite the strength of being easy to apply and follow, the limitation of our downstream model is the segmentation correctness which is still lower than a deep learning model designed and proposed specifically for crack segmentation (but not officially available in any major deep learning framework). This limitation can be further researched in the future by investigating new deep learning architectures or alternative training/optimization strategies. Nevertheless, the insight provided by this paper should help accelerate future works in vision-based low-level crack detection in a way where deep learning researchers and practitioners can extend an off-the-shelf architecture like DeepLabV3-ResNet101 and get reasonable crack segmentation results upon their custom crack dataset without designing and experimenting everything from scratch.

2. Related Works

This section explains our literature review regarding how previous researchers applied deep learning techniques to both high-level and low-level crack detection problems.

2.1. Sliding Window Approach. In the early years of applying deep learning to low-level crack detection, the most popular approach was perhaps the one that combines a traditional solution of sliding window and a deep learning solution of Convolutional Neural Network (CNN) image classifier. In this approach, a big input image is cropped into several small patches using one or more fixed-size sliding windows. Then, based on a patch input, the CNN outputs classification results of crack, noncrack, or other predefined classes. In the latter part of CNN classification, there are two main classification strategies for crack detection—patch-level and pixel-level classification.

The patch-level classification is when one input patch results in one classification label. For example, in the work of [9], one input image was twice scanned by a sliding window, producing lot of 256×256 pixel patches. Then, a small CNN was trained to perform patch-level binary classification (crack or noncrack). Although this work yielded 97% classification accuracy on their test set, performing the window sliding twice per one input image hurt the overall speed (4.5 seconds per image). Similarly, Mitsubishi Electric Research Laboratories [10] proposed a system combining a 520×520 pixel sliding window and four patch-level CNN binary classifiers (for crack, deposit, water leakage, and any). In the test set evaluation, they got a precision of 79.5% and a recall of 65.0%. Apart from the patch-based classification, this work mentioned using two different weights for defect and nondefect training patches in order to deal with the problem of imbalanced dataset where the number of defect patches was much less than the number of nondefect patches. In addition, they used the strategy of Active Learning in order not to put too much effort into the initial phase of data labeling and annotation.

Speaking of the pixel-level classification, it refers to the same sliding window technique but the CNN is now designed for classifying some specific pixels in the input patch. This strategy usually provides an output image that contains per-pixel classification results. For example, [11] used one CNN as a pixel-level binary classifier. Input to the CNN was a 18×18 pixel patch with its center pixel as the target pixel. The output of the CNN was a class label (crack or noncrack) for the center pixel of the input patch. In [4], the authors used a CNN for multilabel classification where the 27×27 pixel input patch resulted in 25 prediction values belonging to the 5×5 structure centered in the input patch. The authors said that, by forcing CNN to generate many outputs at a time, it prevented CNN from misunderstanding the classification problem or being trapped by one specific feature.

2.2. Object Detection Approach. Solving high-level crack detection problems often relies on recent state-of-the-art deep-learning-based object detectors. For example, [12] employed a famous one-stage object detector named Single-Shot MultiBox Detector (SSD) [13] to detect upright rectangular bounding boxes regarding many types of road damage including cracks. This detection was done despite unpredictable presence of many nonroad objects in their city view images. Likewise, [14] proposed using Faster R-CNN [15], the most popular baseline for two-stage object detectors, to localize and detect five types of surface damage including concrete crack, steel corrosion (medium and high damages), bolt corrosion, and steel delamination. Also, in the recent work of [16], a self-reconfigurable robot was proposed using deep learning for pavement segmentation as well as pavement defects (crack and other damages) and garbage detection. In the part of pavement defects and garbage detection, the authors used a TensorFlow implementation of YOLOv2 [17], another popular deep-learning-based one-stage object detector.

Compared to the sliding window approach described in Section 2.1, deep-learning-powered object detectors are superior in two aspects. First, the sliding window approach significantly depends on one or more fixed-size sliding windows, whereas the object detector approach allows more optimized bounding boxes whose sizes are automatically adjusted to fit each crack. This allows the latter approach to deal with images of different sizes and scales. Second, as mentioned in [18] that convolutional layers are costly, the sliding window approach that repeatedly applies CNN over each patch is expensive compared to recent deep-learning-based object detectors that rely on the concept of one CNN as a shared feature extractor for all patches.

Although these deep-learning-powered object detectors are good at high-level scene understanding and able to bound variable-sized cracks on road surfaces, they all share the same limitation. Because of the nature of upright rectangular bounding box outputs, these detectors are not able to precisely localize crack pixels in each of their detected bounding boxes. Consequently, these detectors alone are not enough for low-level analysis tasks like measuring crack width, understanding crack orientation and its distribution pattern, and so forth. Also, for a case when a long thin crack appears diagonally regarding its bounding box, the detector has to output an unnecessarily big box in order to bound the crack. In very unfortunate cases, the big bounding box may cover the majority area of an input image in a way that makes this box meaningless for crack detection. Our suggestion is that we should use these deep-learning-based object detectors to first locate all possible boxes of cracks out of other unrelated objects or complicated scenes (if any). Then, we should move to other approaches for pixel-level fine-grained crack analysis.

2.3. Segmentation Approach. Although the sliding window approach (Section 2.1) is costly at the first glance, it provides more precise crack localization compared to bounding box outputs of the object detection approach (Section 2.2) that can be too big or too small. Therefore, there are crack detection research works that choose the sliding window over the object detector. For example, in the NB-CNN framework [19], the authors explicitly mentioned that they did not prefer the object detector but the sliding window with patch-level classification for crack detection on a video of metallic nuclear power plant specimens. This selection was in order to cover their problem including very thin and long-spanning cracks.

By the way, using the sliding window approach with patch-level classification often ends up overestimating the crack width (low precision) during aggregating predictions among neighborhood pixels. But if the patch size is decreased for the sake of higher precision, it will result in a higher classification error and a lower detection accuracy as the information contained in one tiny patch is not enough to distinguish a crack. To get pixel-perfect crack localization, the sliding window approach with pixel-level classification (explained in Section 2.1) is one alternative solution as it provides an output image with per-pixel dense classification results. Nevertheless, this sliding window approach is slow in execution due to area redundancy between patches and the number of times to execute CNN repeatedly for each patch.

At this point, we are looking for other deep learning solutions that are capable of providing pixel-level classification (low-level crack detection) without redundant computation. Speaking from the perspective of computer vision researchers, we believe that an end-to-end vision-based segmentation model is the appropriate solution for this problem. Examples of deep-learning-based segmentation models for crack detection are CrackNets. The original CrackNet (AKA CrackNet I) [20] and CrackNet II [21] utilized the same concept of using convolutional layers to detect all crack pixels in an input image. Unlike common CNNs, CrackNet I and CrackNet II did not use any pooling layers, so all layers (including the input and output layers) produced outputs of the same 2D spatial size. CrackNet I and CrackNet II resulted in precision of 90.13% and 90.20%, recall of 87.63% and 89.06%, and F-measure of 88.86% and 89.62%, respectively. Although their experimental results looked promising, both CrackNet models were not able to perfectly detect the continuity of cracks, meaning that they might detect several discontinued cracks instead of one connected crack. At this point, another model named CrackNet-R [22] was proposed to solve this problem of crack continuity. CrackNet-R did not use convolutional layers but it considered one crack pixel candidate as one input and fed it to a recurrent neural network (RNN). The RNN then learned whether or not to connect the series of input candidates as one continuous crack. While it was a good idea to use RNN to learn how pixels should relate to one another in a continuous manner, CrackNet-R introduced many processes with handcraft features and fixed algorithms that could not learn to adjust themselves to fit different inputs.

Although a standard CNN can be simply adapted to maintain the same 2D spatial size from the input image all along to the output image as in CrackNet I and CrackNet II, doing this with fully connected layers inside CNN is very costly and is not efficient in time and memory consumption.

Therefore, many researchers prefer using proper segmentation models borrowing from the field of computer vision as these segmentation models are carefully designed and optimized for segmentation tasks. For example, the works of [6, 23–26] performed pixel-level crack detection using the popular U-Net architecture [27] that was firstly introduced for end-to-end semantic segmentation in medical images. Among these research works, some used the U-Net architecture as is, whereas some made major modifications in U-Net to fine-tune it specifically for the context of crack detection. Another state-of-the-art work named DeepCrack [28] proposed a new deep learning architecture that looked similar to U-Net but with a more sophisticated feature map combination and loss fusion. DeepCrack resulted in the precise pixel-level crack detection with the maximum F-measure value of 91%.

3. Proposed Method

3.1. Model Architecture. According to Section 2, previous research works on pixel-level crack detection often designed and trained their deep learning models from scratch; they did not consider much about accelerating their research and development process with the concept of downstream models. Because using downstream models reduces complications in designing and training deep learning models, downstream models have been applied for various tasks in many real-life applications. In the context of pixel-level crack detection, the goal of this paper is to observe downstream models built upon the state-of-the-art segmentation model named DeepLabV3 [29]. At the time of writing this paper, there are pretrained versions of DeepLabV3 available in the official PyTorch. Also, DeepLabV3 with ResNet-101 backbone shows the best performances in semantic segmentation as reported on the official website of PyTorch (https://pytorch.org/vision/stable/models.html#semantic-segmentation accessed on September 20, 2021).

Recently, there have been many state-of-the-art deep learning models proposed for semantic segmentation, both CNN-based models and the rising alternatives of Transformer-based models. Table 1 shows some of them together with mIOU (mean intersection over union) comparison. Nevertheless, this paper will not use these models as our purpose is to use an off-the-shelf deep learning model officially available in a major deep learning framework like PyTorch; this is in order to observe how far one can achieve from downstream models based on an off-the-shelf architecture in the context of low-level crack detection. According to Table 1, Transformer-based models seem to be superior to CNN-based models, particularly in the ADE20K dataset. However, most Transformer-based models for image segmentation are quite new and were originally proposed for high-level semantic segmentation. Hence, it is difficult to directly compare them with our work that focuses on low-level semantic segmentation and crack datasets due to differences in datasets, segmentation targets, and evaluation metrics. In addition, for a general-purpose computer vision backbone like Swin Transformer to perform semantic segmentation, a base model like UperNet is needed, adding another different factor to conduct a fair comparison. As for other CNN-based models listed in the table, they are not available as off-the-shelf models and DeepLabV3 is comparable to them in terms of performance. So we will stick with DeepLabV3 from this point onward.

Speaking of the state-of-the-art DeepLab series, although it is trained on natural images regarding the COCO dataset that is significantly different from most low-level crack image datasets, there are some works that reported performances of DeepLab on crack image datasets. For example, [37] integrated a dense upsampling convolution (DUC) module into DeepLabv3 for dam crack semantic segmentation and got the maximum mIOU of 57%. Also, the work of [38] introduced the densely connected atrous spatial pyramid pooling module into DeepLabv3+ and got an average intersection ratio of 82.37%. Unlike these previous works, our work presented in this paper focuses on the off-the-shelf DeepLabV3-ResNet101 architecture as available in PyTorch 1.7.1 and torchvision 0.8.2 with no major architecture modification other than the final classification layer.

In this work, our model's fine-tuning includes trying four alternative loss functions (i.e., one cross-entropy loss and three binary focal losses), four initial learning rates (i.e., 0.1, 0.01, 0.001, and 0.0001), two alternative numbers of output classes (i.e., one or two segmentation resultant maps), two ways of pixel normalization regarding an input image (i.e., with or without pixel normalization), two ways of using pretrained weights (i.e., using or not using the pretrained weights from COCO as provided by PyTorch), and two ways of freezing layers (i.e., train all layers in the model or train only the classifier). The best results we discovered so far are reported from this point onward.

For one three-channel (RGB) 2D input image, each of our experimental models is set to output a one-channel 2D segmentation map whose pixel value represents a probability of the pixel being crack; this setting is done by changing the classifier of the loaded DeepLabV3-ResNet101 model to DeepLabHead(2048,1); the first parameter of 2048 is the size of feature vector outputted by the ResNet-101 backbone, whereas the second parameter of 1 is the number of desired output classes. We also tried DeepLabHead(2048,2) as well but the results seemed to be inferior compared to those of DeepLabHead(2048,1). In addition, we found that normalizing pixel values of an input image showed no sign of performance improvement, so we omitted this step from further experiments.

3.2. Model Training. When optimizing a deep learning model for image segmentation, our objective function is the loss function that penalizes prediction errors for all image pixels. Given one 2D input image I, let $I_{x,y}$ be a pixel at 2D coordinate (x, y) and let $\mathscr{L}(p)$ be a loss value calculated for the p pixel; then, the constrained optimization for our segmentation model is

$$\min \sum_{p \in I_{x,y}} \mathscr{L}(p). \tag{1}$$

In an encoder-decoder network like DeepLabV3, the constrained optimization is mostly the same as (1) except that there are two subnetworks, the encoder and the decoder

Table 1: Comparison of state-of-the-art deep learning models in four high-level semantic segmentation datasets. Note that numeric data filled in this table refers to mIOU evaluation results collected from original papers.

Model	Publication	Type	mIOU PASCAL VOC 2012 (test set)	mIOU Pascal Context (val. set)	mIOU Cityscapes (test set)	mIOU ADE20K (val. set)
PSPNet [30]	CVPR 2017	CNN	85.4	—	80.2	—
DeepLabV3 [29]	arXiv 2017	CNN	85.7	—	81.3	—
EncNet [31]	CVPR 2018	CNN	85.9	—	—	44.7
DeepLabV3+ [32]	ECCV 2018	CNN	87.8	—	82.1	-
Swin Transformer [33]	ICCV 2021	Transformer	—	—	—	53.5
Segmenter [34]	ICCV 2021	Transformer	—	59.0	—	51.8
SETR [35]	CVPR 2021	Transformer	—	55.8	81.6	50.3
SegFormer [36]	NeurIPS 2021	Transformer	—	—	83.1	51.8

working together as shown in Figure 1. In an encoder-decoder network, an input image I is first passed through the encoder. Then, the encoder's outputs are fed as the decoder's input. Finally, outputs from the decoder are our segmentation results to be used in per-pixel loss computation. Nevertheless, as recent encoder-decoder networks may involve several skip connections and other complicated relationships between the two subnetworks, the above equation may be evolved into a much more sophisticated form.

Because the problem of pixel-level crack detection can be considered as the problem of per-pixel binary classification (crack or noncrack pixel), the default alternative of the loss function is the cross-entropy loss function. However, according to our experiments, no matter what training strategies we used on DeepLabV3-ResNet101, proper crack detection results were not obtained from the cross-entropy loss function. We believe that this might be an unwanted consequence of the severely imbalanced nature of low-level crack detection, where the number of crack pixels is extremely smaller than the number of noncrack pixels in most input images.

Since the common tricks of oversampling and undersampling cannot be used to balance the numbers of crack and noncrack pixels in this context of low-level crack detection, we explore other loss functions that prioritize prediction on the positive class (crack) over that on the negative class (noncrack). In this paper, the selected loss function is the binary focal loss function—a binary version of the famous focal loss [7, 8] computed as written in Equation 2. In the equation, y_i and $\hat{y}_i$ are ground-truth and predicted values of the i^{th} pixel, respectively; $\mathscr{L}_{bf+}$ is the loss value calculated from positive (crack) pixels, where $y_i = 1\,(y_+)$; $\mathscr{L}_{bf-}$ is the loss value calculated from negative (noncrack) pixels, where $y_i = 0\,(y-)$. The constant value of $\alpha = 3$ is used in all experiments. Also, three values of γ (i.e., 1, 2, and 4) will be experimented and referred to as $\mathscr{L}_{bf1}$, $\mathscr{L}_{bf2}$, and $\mathscr{L}_{bf4}$, respectively.

$$\begin{aligned}
\mathscr{L}_{bf} &= \mathscr{L}_{bf+} + \mathscr{L}_{bf-},\\
\mathscr{L}_{bf+} &= -\left(\frac{\sum_{i\in y_+}\left[(1-\hat{y}_i)^{\gamma}\cdot\log(\hat{y}_i)\right]}{\sum_{i\in y_+}(1-\hat{y}_i)^{\gamma}}\right),\\
\mathscr{L}_{bf-} &= -\alpha\left(\frac{\sum_{i\in y_-}\left[(\hat{y}_i)^{\gamma}\cdot\log(1-\hat{y}_i)\right]}{\sum_{i\in y_-}(\hat{y}_i)^{\gamma}}\right).
\end{aligned} \tag{2}$$

Before training downstream models upon the DeepLabV3-ResNet101 architecture on the low-level crack image dataset, there are two more training issues to be considered. First, should we train all layers in the model or should we train only the new layers in the DeepLabHead classifier? Second, should we continue training from the pretrained weights (pretrained on a subset of COCO train2017) or should we train all weights from scratch? For the first issue, our preliminary experiments revealed that training all layers yields better results than training only the classifier, so we will stick with this in all the following experiments. As for the second issue, we will conduct experiments both with and without pretrained weights and compare their results.

All experimental downstream models were trained on the same laptop computer with one Nvidia GeForce RTX 2080 GPU (8 GB RAM) using Python 3.8, CUDA Toolkit 10.2.89, cuDNN 7.6.5, PyTorch 1.7.1, and torchvision 0.8.2. The training was conducted for 300-epoch length with the batch size of 2 and the Stochastic Gradient Descent (SGD) optimizer (the initial learning rate of 0.0001, weight decay of 0.0005, and momentum of 0.9). The learning rate scheduler is used to multiply the current learning rate with 0.1 if the validation loss stays stable for 10 continuous epochs.

3.3. Dataset and Evaluation Metrics. For a deep learning model that works with natural images and requires high-level scene/object understanding, it is common to expect a training dataset with millions of images or more as the object/scene diversity is very high. However, the work in this paper deals with road surface images which are nonnatural images and need no high-level scene/object understanding. Hence, the number of training images needs not be that much; for example, the famous U-Net deep learning model [27] can do semantic segmentation on medical images using only 30 training images; the state-of-the-art DeepCrack [28] trains its deep learning model with 260 images from the CrackTree260 dataset. In order to compare our downstream models with the state-of-the-art DeepCrack (in Section 4), we use the same bundle of four crack datasets distributed by DeepCrack as well as their evaluation metrics as they were proved sufficient for this task of crack segmentation. Example images in the four crack datasets are displayed in Figure 2 and summarized in Table 2. As written in the table,

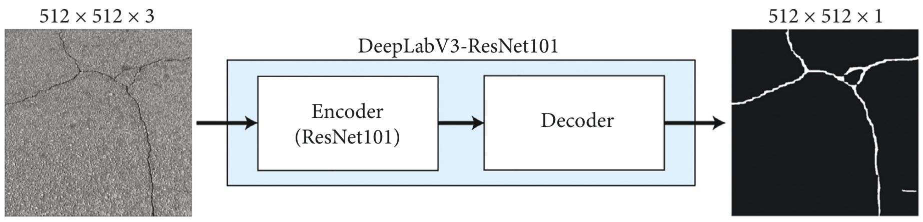

Figure 1: A high-level illustration of DeepCrackV3-ResNet101 used in this work. For more details of DeepLabV3-ResNet101, please refer to the original paper [29].

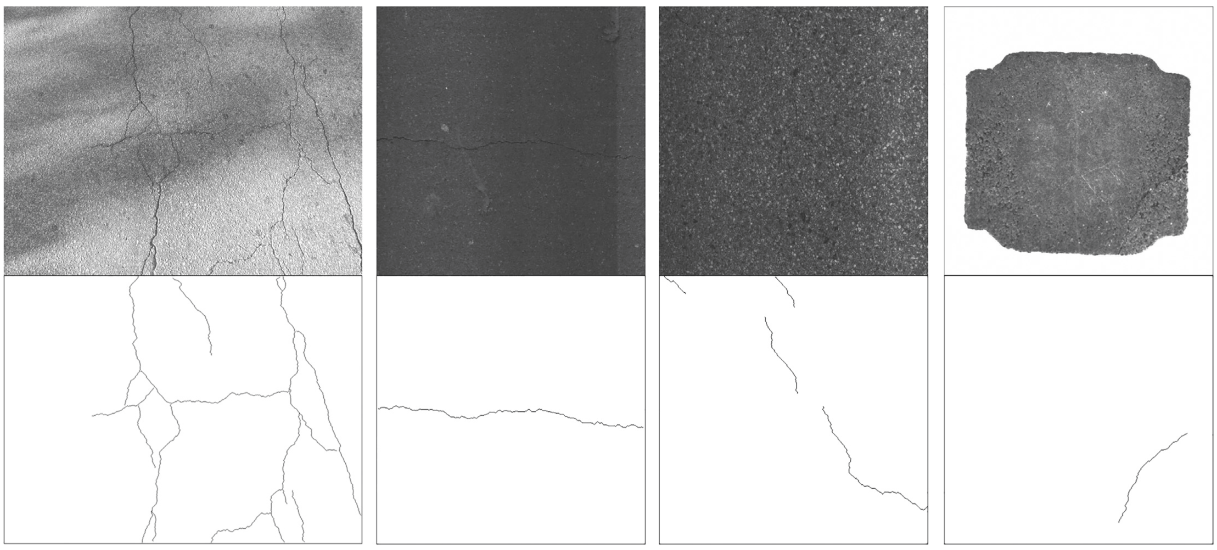

Figure 2: Examples of crack images (top row) and ground-truth images (bottom row) as provided by DeepCrack [28]. Columns from left to right are the datasets named CrackTree260, CRKWH100, CrackLS315, and Stone331, respectively. Note that images in the bottom row are inverted here for clear visualization.

only the CrackTree260 dataset is used for training and validating, whereas all four datasets are used for testing. One interesting fact about this bundle of crack datasets is that all ground-truth images are annotated with one-pixel-wide curves despite the actual width of cracks.

Because our experimental models are designed for one training sample which is a pair of one 512×512 RGB input image and one 512×512 binary ground-truth image, there are some preprocessing steps required to convert each sample to the model-ready format. Our preprocessing steps include image resizing and pixel rescaling. For the part of image resizing, we resample the RGB input image using pixel area relation and resize the binary ground-truth image using the nearest neighbor interpolation; the latter interpolation is selected in order not to introduce nonbinary pixels in the resized image. Then, during the training process, several techniques of data augmentation are applied on the fly to each training sample; this is for the purpose of avoiding overfitting and increasing image diversity. The augmentation includes random flip (both horizontal and vertical), random blur, random hue, random saturation, random brightness, random contrast, and random crop. Finally, when the one-channel greyscale segmentation map is predicted and outputted from our experimental model, it must be postprocessed to become a final binary output map. The postprocessing steps include the min-max normalization and the binarization. The final output map is a 512×512 one-channel binary image whose 0.0- and 1.0-pixel values refer to noncrack and crack pixels, respectively. Note that the image augmentation is partly done with OpenCV 4.5.1.48.

In the problem of high-level scene or object segmentation as listed in Table 1, mIOU is a popular evaluation metric. However, the situation is different in the problem of low-level crack detection, where each crack can be very thin and pixel-accurate crack localization is nontrivial. Similar to previous works in crack detection including DeepCrack [28], we do not evaluate our downstream models with mIOU but three F-measure-based values—the optimal dataset-scale F-measure (ODS F-measure), the optimal image-scale F-measure (OIS F-measure), and the average precision (AP or the area under the precision-recall curve regarding the ODS F-measure). The model evaluation is done with some flexibility, allowing the pixel that is predicted as a crack and is in the d-pixel proximity of the actual crack pixel to be counted as a true-positive pixel. This flexible evaluation is used in many previous works of low-level crack detection, including DeepCrack which used $d = 2$.

Table 2: Summary of the four crack datasets used in this paper.

Dataset	Resolution (pixels)	Number of images			
		Train.	Val.	Test	Total
CrackTree260	800 × 600, 960 × 720	200	20	40	260
CRKWH100	512 × 512	—	—	100	100
CrackLS315	512 × 512	—	—	315	315
Stone331	1024 × 1024	—	—	331	331

Table 3: Comparing our downstream models built upon DeepLabV3-ResNet101 with other works of deep-learning-based low-level crack detection on the same test datasets; the flexibility value of $d = 2$ is used in all evaluations. While ODS F-measure is reported with its corresponding precision (P) and recall (R) as written underneath, OIS F-measure is reported together with its standard deviation (written with a leading ± symbol) showing the dispersion of the optimal image-scale F-measure in a particular test dataset. For each training strategy of us, the best (maximum) value among the four loss functions is highlighted in bold.

	Modified U-Net [6]	DeepCrack [28]	Our downstream models built upon DeepLabV3-ResNet101							
			Trained from pretrained weights				Trained from random weights			
			$\mathscr{L}_{\mathrm{bce}}$	$\mathscr{L}_{\mathrm{bf1}}$	$\mathscr{L}_{\mathrm{bf2}}$	$\mathscr{L}_{\mathrm{bf4}}$	$\mathscr{L}_{\mathrm{bce}}$	$\mathscr{L}_{\mathrm{bf1}}$	$\mathscr{L}_{\mathrm{bf2}}$	$\mathscr{L}_{\mathrm{bf4}}$
CrackTree260										
OIS F-measure	0.3981	–	0.3081 ± .0919	0.8520 ± .0468	0.8596 ± .0521	**0.8752** ± .0312	0.3031 ± .0594	0.8542 ± .0227	0.8577 ± .0241	**0.8705** ± .0230
ODS F-measure	0.3940	–	0.2218 @P .1325 @R .6793	0.8354 @P .7706 @R .9120	0.8408 @P .7835 @R .9073	**0.8553** @P .8049 @R .9124	0.2721 @P .1789 @R .5684	0.8440 @P .7777 @R .9226	0.8449 @P .7870 @R .9120	**0.8530** @P .8056 @R .9064
AP	0.1683	–	0.1317	0.8347	0.8387	**0.8879**	0.1764	0.8595	0.8820	**0.8913**
CRKWH100										
OIS F-measure	0.3350	0.9170	0.1117 ± .1208	0.5321 ± .3660	**0.6107** ± .3404	0.5447 ± .3595	0.1681 ± .0983	**0.8131** ± .1282	0.8020 ± .1315	0.8111 ± .1276
ODS F-measure	0.3483	0.9095	0.1407 @P 0.0848 @R 0.4128	0.5226 @P 0.3934 @R 0.7785	0.4480 @P 0.3072 @R 0.8277	**0.6612** @P 0.5338 @R 0.8686	0.0951 @P 0.0595 @R 0.2369	**0.8166** @P 0.7530 @R 0.8919	0.8029 @P 0.7416 @R 0.8752	0.8011 @P 0.7429 @R 0.8693
AP	0.1242	0.9315	0.0642	0.3490	0.2629	**0.3938**	0.0421	**0.8362**	0.8215	0.8239
CrackLS315										
OIS F-measure	0.2517	0.8671	0.1371 ± 0.1009	0.6844 ± 0.2237	**0.7191** ± 0.1924	0.6761 ± 0.2467	0.1832 ± 0.1095	**0.7315** ± 0.1734	0.7177 ± 0.1808	0.7217 ± 0.1832
ODS F-measure	0.2648	0.8449	0.0791 @P 0.0560 @R 0.1345	0.7143 @P 0.6322 @R 0.8208	**0.7332** @P 0.6587 @R 0.8268	0.6698 @P 0.5656 @R 0.8210	0.1143 @P 0.0628 @R 0.6413	**0.7446** @P 0.6722 @R 0.8345	0.7255 @P 0.6597 @R 0.8059	0.7325 @P 0.6663 @R 0.8133
AP	0.1004	0.8772	0.0351	0.6378	**0.6893**	0.5178	0.0512	**0.7205**	0.7096	0.7138
Stone331										
OIS F-measure	0.4090	0.8751	0.0605 ± 0.0687	0.0480 ± 0.0200	0.0532 ± 0.0273	**0.0564** ± 0.0406	0.0453 ± 0.0309	0.6683 ± 0.1719	**0.7587** ± 0.1348	0.6578 ± 0.1976
ODS F-measure	0.4333	0.8559	0.0313 @P 0.0159 @R 0.8917	0.0340 @P 0.0173 @R 0.9817	**0.0358** @P 0.0182 @R 0.9662	0.0288 @P 0.0146 @R 0.9784	0.0277 @P 0.0140 @R 1.0000	0.6829 @P 0.6036 @R 0.7862	**0.7572** @P 0.7075 @R 0.8144	0.6822 @P 0.6264 @R 0.7488
AP	0.2374	0.8883	0.0126	0.0123	**0.0130**	0.0125	0.0098	0.6686	**0.7697**	0.6653

To perform the flexible evaluation, we use two ground-truth images to compute precision and recall; the two ground-truth images are one original ground-truth image (i.e., one-pixel wide crack annotation) and one flexible ground-truth image. For any predefined integer value of $d > 0$, the flexible ground-truth image is prepared by applying the morphological dilation to the original ground-truth image with the square kernel of size $(2 * d) + 1$. This means that if the original white pixel (i.e., crack pixel) has one-pixel width, using $d = 2$ will make the pixel become five-pixel wide in the flexible ground-truth image. Finally, when computing precision and recall values, the numbers of true-positive and false-positive pixels are counted

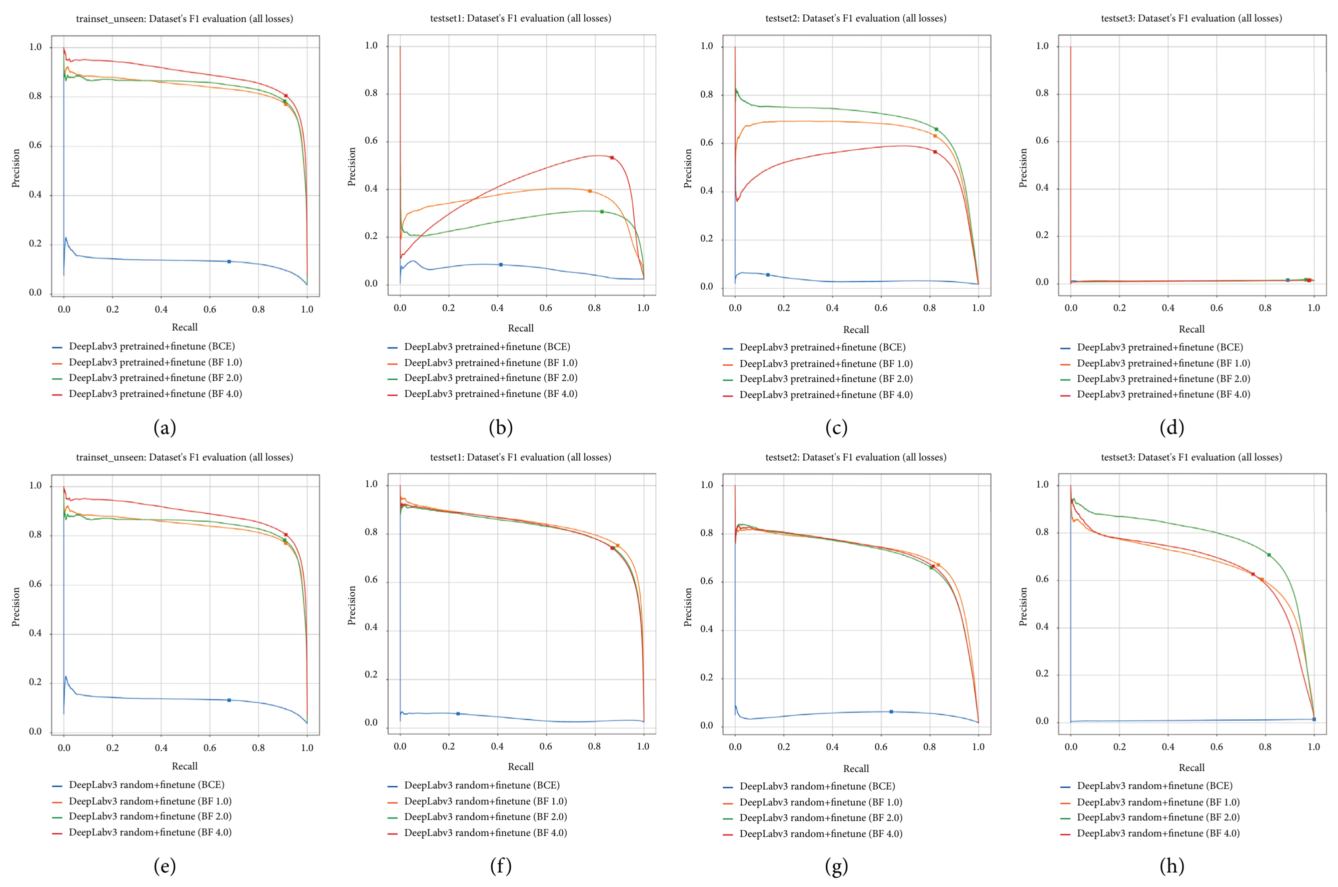

Figure 3: The precision-recall curves regarding our downstream models built upon DeepLabV3-ResNet101. The best F-measure (ODS F-measure) of each graph is marked with a small rectangle. (a–d) and (e–h) represent two training strategies which are training from the pretrained weights and training from random weights, respectively. Testing results are shown for CrackTree260 in (a, e), CRKWH100 in (b, f), CrackLS315 in (c, g), and Stone331 in (d, h).

on the flexible ground-truth image, whereas the number of false-negative pixels is counted on the original ground-truth image. Note that we use scikit-learn 0.23.2 for evaluation and matplotlib 3.3.1 for image as well as graph visualization.

3.4. Comparative Experiments. To compare our proposed models with other deep learning models, we choose our previous work of Modified U-Net [6] and the state-of-the-art DeepCrack [28]. Like our work, both are deep learning models that can be trained in an end-to-end manner and are proposed specifically for crack segmentation.

Although Modified U-Net and DeepCrack share the same concept of using the encoder-decoder architecture, their internal implementations are totally different. On one hand, Modified U-Net borrows the main concept from the famous U-Net [27] and applies some modifications to make it fit with the task of crack segmentation. This means that Modified U-Net follows the idea of symmetry U-shape encoder-decoder architecture with skip connections between same-level layers in the encoder and the decoder as well as a single loss computed only at the end of the decoder. On the other hand, DeepCrack is a newly designed architecture that has an appearance that is similar to that of U-Net but has no skip connection between the two subnetworks and uses a compound loss to train the model. Instead of propagating feature maps from the encoder and concatenating them directly to feature maps in the decoder, DeepCrack combines feature maps from same-level layers in the encoder and the decoder, fuses them at the middle, and produces an additional loss for each level. In terms of crack segmentation performances, DeepCrack achieves the state-of-the-art results in segmenting very fine and thin cracks, whereas Modified U-Net suffers from overly thick segmentation results (high recall but low precision). To get more precise results from Modified U-Net, a postprocessing step (e.g., edge thinning) may be required in addition.

Similar to Modified U-Net and DeepCrack, our downstream models regarding DeepLabV3-ResNet101 use the encoder-decoder architecture. However, designing and tuning Modified U-Net and DeepCrack is a process that requires researchers and practitioners to experiment and tweak every related design and hyperparameter by themselves; this can discourage many who are not familiar with these kinds of deep learning models for semantic segmentation. Our research goal is different as it is our main contribution to help researchers and practitioners by utilizing as much as possible from an off-the-shelf architecture officially available in a major deep learning framework. Hence, we choose to rely mainly on the fixed architecture of DeepLabV3-ResNet101, change only on

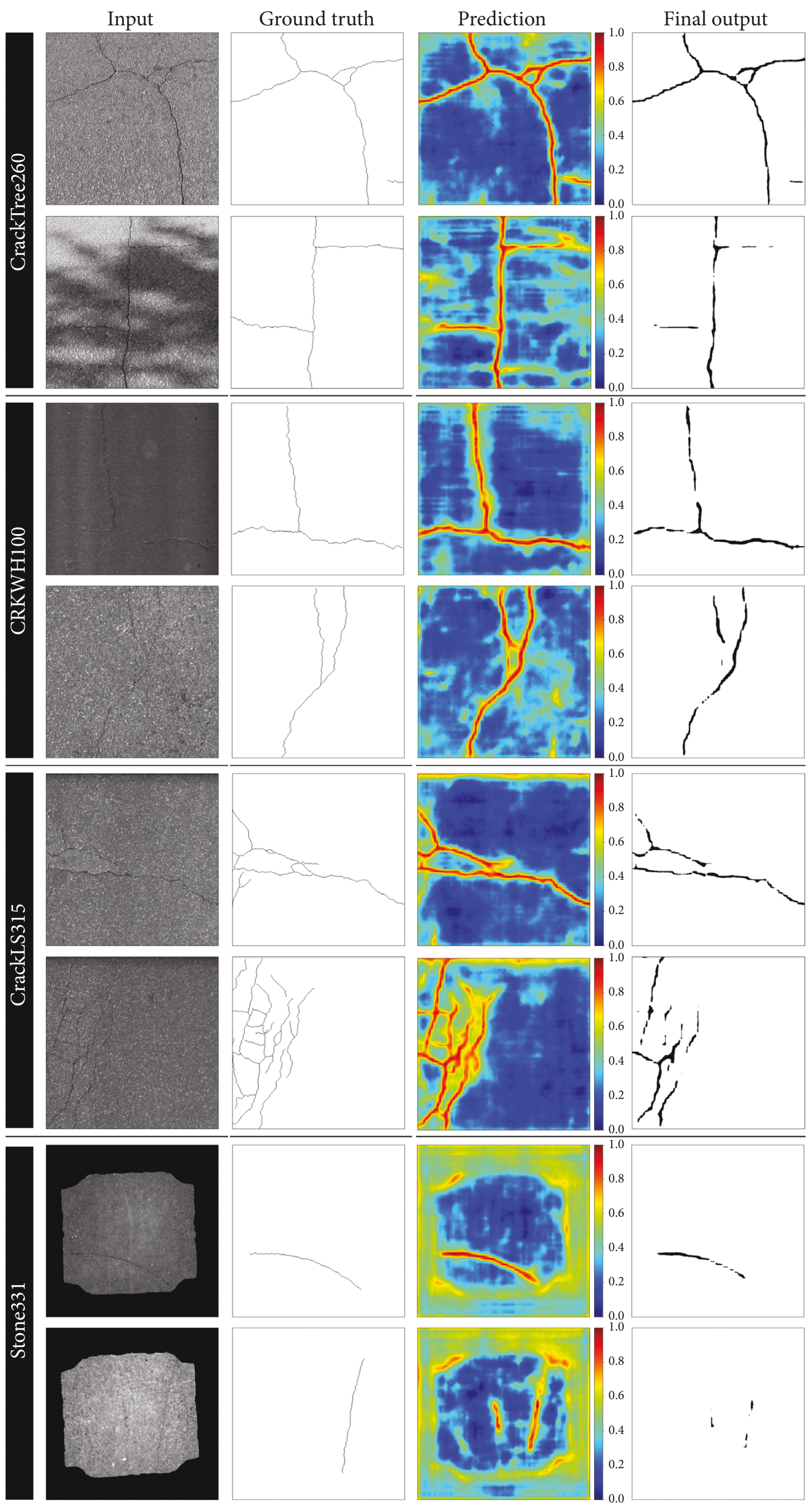

FIGURE 4: Some results from our downstream model that is trained by $\mathscr{L}_{bf2}$ from random weights. For better visualization, binary images in the second and fourth columns are inverted and greyscale images in the third column are colorized with the jet colormap in matplotlib.

the essential part of the classifier head, and tune only those hyperparameters that are fundamental to most deep learning practitioners (e.g., the loss function, trainable layers, and the decision whether to use pretrained weights).

4. Experimental Results

Table 3 compares our downstream models with the state-of-the-art DeepCrack model [28] as well as our previous model

Table 4: Time analysis regarding three model architectures based on input images of 512 × 512 resolution.

Model	Training time per 300 epochs	Inference Time per image	Inference FPS	Remark
DeepCrack [28]	—	0.153 s	6	One Nvidia GeForce GTX TITAN-X GPU (6.69 TFLOPS in FP32)
Modified U-Net [6]	2 h 15 m	0.040 s	25	One Nvidia GeForce RTX 2080 GPU (10.07 TFLOPS in FP32)
DeepLabV3-ResNet101 (ours)	6 h	0.078 s	13	One Nvidia GeForce RTX 2080 GPU (10.07 TFLOPS in FP32)

of Modified U-Net [6]. Except for DeepCrack whose reported performances in [28] are used as is, other models in the table are trained with the CrackTree260 dataset and tested with four crack datasets following the description in Table 2. Note that, to avoid dynamic results due to different random seed values, each of our experimental DeepLabV3-ResNet101 models in Table 3 is trained and evaluated only once using the same set of fixed seeding values.

According to Table 3, when evaluating our downstream models with unseen images from the CrackTree260 dataset, the performances of downstream models are not much different between the two training strategies. However, when changing to other test datasets, the training strategy of random weights performs better, particularly in the Stone331 dataset. This difference between the two training strategies in our downstream models is also shown in Figure 3, where (a)-(d) graphs belonging to the training strategy of pretrained weights reveal performance instability across different test datasets. Adding more epochs of training may allow the downstream models on the top row of the figure to fit more on the problem of pixel-level crack detection. However, doing this will increase computational resources. Our recommendation for the problem of low-level crack detection is that in case the base model is originally proposed for high-level scene/object segmentation, it is better to neglect the pretrained weights given with the base model and train our downstream models from random weights.

Among all models listed in Table 3, the best one is the state-of-the-art DeepCrack which wins in all indicators. Nevertheless, our downstream models trained with the binary focal loss function show reasonable performances especially when they are trained from random weights. Considering the three downstream models trained with $\mathscr{L}_{\mathrm{bf}}$ from random weights, although the winner in each test dataset is varied, the performances of the three models are not significantly different from one another in each indicator, except for the Stone331 dataset. In the Stone331 dataset, the downstream model trained with $\mathscr{L}_{bf2}$ is better than the other two by a significant margin. Because of this ability of good generalization among different test datasets, the downstream model trained with $\mathscr{L}_{bf2}$ from random weights is concluded as our best downstream model. As for bad performing models, two downstream models trained with the binary cross-entropy loss function are obviously the worst as they can barely detect or localize any crack. The second worst model is Modified U-Net as this model suffers from too thick segmentation results (high recall but low precision) in the context of one-pixel-wide ground-truth annotation; observing the cause of thick results is listed as future work in Modified U-Net's paper. Comparing Modified U-Net and our best downstream model trained with $\mathscr{L}_{bf2}$ from random weights, our downstream model beats Modified U-Net with $\tilde{}$2x (up to 7x) better evaluation results.

Figure 4 displays some segmentation results from the downstream model trained with $\mathscr{L}_{bf2}$ from random weights. Images in the fourth column are obtained by binarizing the corresponding images in the third column (i.e., the raw prediction images). In this figure, the threshold value used for binarizing images in each dataset is the one that produces the ODS F-measure in Table 3. Hence, it is the threshold value that tries to maximize the balance between precision and recall. The current segmentation images in the fourth column show a problem of crack discontinuity which might become better if we choose another threshold value that aims for higher recall. Because getting both high precision and high recall with a single-shot end-to-end deep learning model is challenging for very fine cracks, a specifically designed model like the state-of-the-art DeepCrack [28] may be required. However, for downstream models whose knowledge or architecture can be shared or transferred among different problem domains, additional post-processing steps like the edge thinning and edge linking in the classic Canny edge detector [39] should become useful to improve the final segmentation results without having to tweak or redesign the deep learning model's architecture.

Finally, in terms of time analysis, Table 4 shows a comparison between the three model architectures in our comparative experimental results; note that values regarding DeepCrack are from those reported in its original paper [28]. From the table, it is obvious that the least accurate Modified U-Net is the fastest and the least time-consuming model, whereas the most accurate DeepCrack is the slowest and the most time-consuming model (even after being compensated with its 1.5x lower GPU FLOPS).

5. Conclusion

This paper uses the popular concept of downstream models in the context of low-level crack detection. Based on the state-of-the-art DeepLabV3-ResNet101 architecture for semantic segmentation, our downstream models are trained with four alternative loss functions and two training strategies. Experimental results reveal that, for the same training setting, downstream models generalize to different datasets better when they are trained from random weights rather than from the pretrained weights. Also, using the binary focal loss function with $\gamma = 2$ yields the most compromised results in all test datasets.

This paper experimentally confirms that it is possible to use an off-the-shelf deep learning architecture for semantic segmentation as a base model and build downstream models upon it for low-level crack detection. Major architecture modification to the base model is not necessary but the special loss function is required to deal with the severely imbalanced nature of low-level crack detection. It is worth noting that the purpose of our downstream models is different from those of other previous works that specially design and fabricate their deep learning architectures from scratch for low-level crack detection. Our downstream models focus on reusing an existing deep learning architecture, including the one that is not proposed for low-level crack detection, reducing technical tasks, and accelerating the model training. Our future work is to additionally improve the performance of the downstream model without introducing any major modification to the model architecture. We plan to further apply advanced image processing techniques on the probability map provided by the downstream model to refine the results.

Data Availability

Previously reported crack datasets that were used to support this study are available at 10.1109/TIP.2018.2878966. The prior study and datasets are cited at relevant places within the text as references [28].

Conflicts of Interest

The author declares that no conflicts of interest.

Acknowledgments

The research presented herein was partially supported by a research grant from the Research Center, National Institute of Development Administration (NIDA), Bangkok, Thailand.

References

[1] L. Zhang, F. Yang, Y. D. Zhang, and Y. J. Zhu, "Road crack detection using deep convolutional neural network," in *Proceedings of the IEEE International Conference on Image Processing*, pp. 3708–3712, ICIP, Phoenix, AZ, USA, Sebtember2016.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hintodn, "ImageNet classification with deep convolutional neural networks," *International Conference on Neural Information Processing Systems (NIPS'12)*, vol. 1, pp. 1097–1105, 2012.

[3] M. Eisenbach, R. Stricker, D. Seichter et al., "How to get pavement distress detection ready for deep learning? A systematic approach," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 2039–2047, IJCNN, Anchorage, AK, USA, July 2017.

[4] Z. Fan, Y. Wu, J. Lu, and W. Li, "Automatic pavement crack detection based on structured prediction with the convolutional neural network," 2018, https://arxiv.org/abs/1802.02208.

[5] T. Siriborvornratanakul, "An automatic road distress visual inspection system using an onboard in-car camera," *Advances in Multimedia*, vol. 2018, Article ID 2561953, 10 pages, 2018.

[6] T. Siriborvornratanakul, "A deep learning based road distress visual inspection system using Modified U-Net," *in Lecture Notes in Computer Science*, vol. 13097, pp. 345–355, 2021.

[7] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, ICCV, Venice, Italy, September 2017.

[8] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020.

[9] Y. Cha, W. Choi, and O. Büyüköztürk, "Deep learning-based crack damage detection using convolutional neural networks," *Computer-Aided Civil and Infrastructure Engineering*, vol. 32, no. 5, pp. 361–378, 2017.

[10] C. Feng, M. Liu, C. Kao, and T. Lee, "Deep active learning for civil infrastructure defect detection and classification," in *Proceedings of the International Workshop on Computing in Civil Engineering*, pp. 298–306, IWCCE, Seattle, USA, June 2017.

[11] Y. Li, H. Li, and H. Wang, "Pixel-wise crack detection using deep local pattern predictor for robot application," *Sensors*, vol. 18, no. 9, 2018.

[12] H. Maeda, Y. Sekimoto, T. Seto, T. Kashiyama, and H. Omata, "Road damage detection and classification using deep neural networks with smartphone images," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 12, pp. 1127–1141, 2018.

[13] W. Liu, D. Anguelov, D. Erhan et al., "SSD: Single Shot MultiBox detector," in *Proceedings of the European Conference on Computer Vision*, pp. 21–37, ECCV, Springer, Cham, September2016.

[14] Y. Cha, W. Choi, G. Suh, S. Mahmoudkhani, and O. Buyukozturk, "Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 9, pp. 731–747, 2018.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[16] B. Ramalingam, A. A. Hayat, M. R. Elara et al., "Deep learning based pavement inspection using self-reconfigurable robot," *Sensors*, vol. 21, no. 8, 2021.

[17] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," 2016, https://arxiv.org/abs/1612.08242.

[18] N. Ma, X. Zhang, H. Zheng, and J. Sun, "ShuffleNet V2: practical guidelines for efficient CNN architecture design," in *Proceedings of the European Conference on Computer Vision*, pp. 122–138, ECCV, Springer, Cham, September2018.

[19] F. Chen and M. R. Jahanshahi, "Deep learning-based crack detection using convolutional neural network and naïve bayes data fusion," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 5, pp. 4392–4400, 2018.

[20] A. Zhang, K. C. Wang, B. Li et al., "Automated pixel-level pavement crack detection on 3D asphalt surfaces using a

deep-learning network," *Computer-Aided Civil and Infrastructure Engineering*, vol. 32, no. 10, pp. 805–819, 2017.

[21] A. Zhang, K. C. Wang, M. Asce et al., "Deep learning-based fully automated pavement crack detection on 3D asphalt surfaces with an improved CrackNet," *Journal of Computing in Civil Engineering*, vol. 32, no. 5, 2018.

[22] A. Zhang, K. C. Wang, Y. Fei et al., "Automated pixel-level pavement crack detection on 3D asphalt surfaces with a recurrent neural network," *Computer-Aided Civil and Infrastructure Engineering*, vol. 34, no. 3, pp. 213–229, 2019.

[23] J. Cheng, W. Xiong, W. Chen, Y. Gu, and Y. Li, "Pixel-level crack detection using U-Net," in *Proceedings of the IEEE Region 10 International Conference*, TENCON'18, Jeju, Korea (South), October2018.

[24] Z. Liu, Y. Cao, Y. Wang, and W. Wang, "Computer vision-based concrete crack detection using U-net fully convolutional networks," *Automation in Construction*, vol. 104, pp. 129–139, 2019.

[25] S. L. Lau, E. K. Chong, X. Yang, and X. Wang, "Automated pavement crack segmentation using U-Net-based convolutional neural network," *IEEE Access*, vol. 991 page, 2020.

[26] W. Qiao, H. Zhang, F. Zhu, and Q. Wu, "A crack identification method for concrete structures using improved U-Net convolutional neural networks," *Mathematical Problems in Engineering*, vol. 2021, Article ID 6654996, 16 pages, 2021.

[27] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," *Medical Image Computing and Computer-Assisted Intervention*, vol. 9351, pp. 234–241, 2015.

[28] Q. Zou, Z. Zhang, Q. Li, X. Qi, Q. Wang, and S. Wang, "DeepCrack: learning hierarchical convolutional features for crack detection," *IEEE Transactions on Image Processing*, vol. 28, no. 3, pp. 1498–1512, 2019.

[29] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017, https://arxiv.org/abs/1706.05587.

[30] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, Honolulu, HI, USA, July 2017.

[31] H. Zhang, K. Dana, J. Shi et al., "Context encoding for semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, CVPR, Salt Lake City, UT, USA, March 2018.

[32] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European Conference on Computer Vision*, ECCV, Munich, Germany, October 2018.

[33] Z. Liu, Y. Lin, Y. Cao et al., "Swin Transformer: hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, ICCV, 2021.

[34] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, "Segmenter: transformer for semantic segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, ICCV, 2021.

[35] S. Zheng, J. Lu, H. Zhao et al., "Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, CVPR, 2021.

[36] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "SegFormer: simple and efficient design for semantic segmentation with transformers," in *Proceedings of the Conference on Neural Information Processing Systems*, NeurIPS, December 2021.

[37] J. Zhang and J. Zhang, "An improved nondestructive semantic segmentation method for concrete dam surface crack images with high resolution," *Mathematical Problems in Engineering*, vol. 2020, Article ID 5054740, 14 pages, 2020.

[38] H. Fu, D. Meng, W. Li, and Y. Wang, "Bridge crack semantic segmentation based on improved Deeplabv3+," *Journal of Marine Science and Engineering*, vol. 9, no. 671, 2021.

[39] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.