Published in Image Processing On Line on 2020–09–21.
Submitted on 2019–10–13, accepted on 2020–05–25.
ISSN 2105–1232 © 2020 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials, software, datasets and online demo at
https://doi.org/10.5201/ipol.2020.282

An "All Terrain" Crack Detector obtained by Deep Learning on Available Databases

Sébastien Drouyer

Université Paris-Saclay, CMLA (UMR CNRS 8536), France
sebastien.drouyer@cmla.ens-cachan.fr

Abstract

We present a general deep learning method for detecting cracks on all sorts of surfaces. For making this method robust to different types of cracks and acquisition procedures, we have trained our method on four datasets - Crack500, DeepCrack, SDNet2018 and CrackForest. We have also labelled a part of the SDNet2018 dataset so that it contains semantic labels, as it originally only proposed crack/non-crack classifications on the image level. To validate our approach, we perform a cross-dataset study where we train the model on a subset of the datasets and test it on another subset. Results of this study show that training the model on these various datasets makes it more robust to new images, outperforming existing classical and deep learning methods. In order to make our method even more robust to different objects, scenes and illuminations, we have also added images from the Flickr website, leading to an important drop in false positives on extra dataset images. The network seems to function well on images not belonging to any of the datasets, and its publication in IPOL will allow users to enrich further training.

Source Code

The python source codes implementing the algorithms described in the paper, and the online demo, are accessible at the associated web page[1].

Supplementary Material

The parameters of the implemented neural network, the labels used for the SDNet2018 dataset, the filenames of the different datasets used for training, validation and test, and the URLs of the Flickr images used for data augmentation are provided as supplementary material at the associated web page.

Keywords: crack detection; semantic segmentation; deep neural networks; U-net

[1]https://doi.org/10.5201/ipol.2020.282

1 Introduction

Finding cracks on urban structures such as buildings, bridges, roads, and actually on any material subject to fatigue, is essential for identifying decaying structures and maintaining them. The surface to inspect is however extensive, making a manual inspection time-intensive and prone to errors. As a result, efforts have been made in the recent years to detect cracks in an automatic manner [4, 32, 18, 20, 35, 25, 22, 30, 29, 3, 8].

Although this task might be easy in simple cases such as wide cracks on white homogeneous walls, it becomes much more difficult when the studied surface is rough or textured, or when the crack is thin and not easily discernible. As classical approaches fail to produce satisfying results, machine learning and deep learning approaches have been proposed in recent years [18, 17, 15]. However, two major problems subside.

First, as these methods are specialized on specific areas (bridges or roads for instance) and specific acquisition procedures, their performance often drops when applied on different images. This can easily be observed by the variety of shapes, texture or illumination conditions of the images in different datasets (see Figure 1).

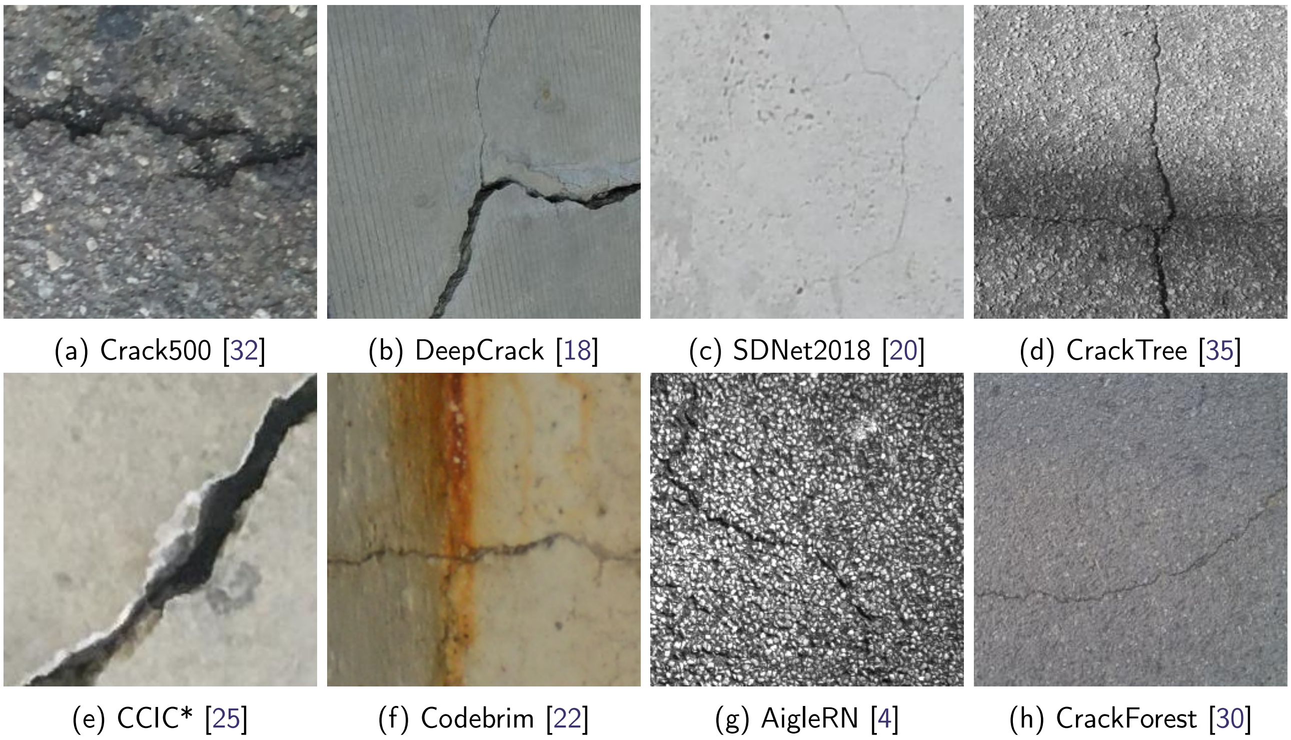

(a) Crack500 [32] (b) DeepCrack [18] (c) SDNet2018 [20] (d) CrackTree [35]
(e) CCIC* [25] (f) Codebrim [22] (g) AigleRN [4] (h) CrackForest [30]

Figure 1: Crops of different crack datasets. CCIC: Concrete Crack Images for Classification.

Secondly, some methods only classify images [20, 22, 25] as containing a crack or not. The shapes and widths of the cracks are not provided, although it can be an important information for assessing the cracks' severity.

The objective of this work is to propose a robust semantic segmentation algorithm detecting cracks on images (see Figure 2). For achieving this goal, we have first created a dataset merging the largest state of the art datasets: Crack500 [32], DeepCrack [18], SDNet2018 [20] and CrackForest [30]. A part of the SDNet2018 dataset [20] has also been improved: it initially provided only crack / non-crack classes for each image, but using manually corrected predictions of the deep learning algorithm trained on the other datasets, we now provide semantic segmentation labels. Those labels are available online.

This dataset was created so that our algorithm can be trained on a variety of crack detection problems. To evaluate its robustness, we have elaborated a cross-dataset study where the algorithm is trained on a subset of datasets and evaluated on another subset. We show that merging these datasets greatly enhances the robustness of the trained neural network. We also show that adding non-crack pictures (extracted from the Flickr website) increases the neural network performance, especially reducing the rate of false positives. The weights of the trained neural networks as well as the source code are available online.

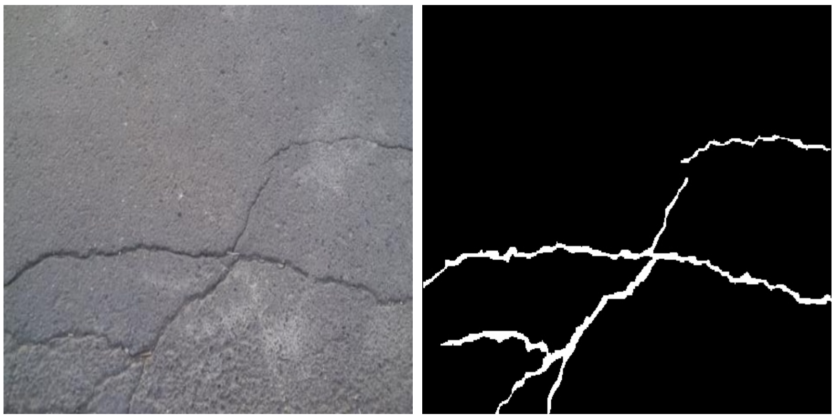


Figure 2: Objective of our method: create, from and input image (a), a semantic segmentation map indicating where the cracks are (b). Example from the CrackForest dataset [30].

2 Related Work

Researchers have been using image processing techniques to detect cracks on concrete materials for several decades [13, 23]. There is a general interest in detecting cracks on any type of surface, but some areas are more active than others. Most popular research areas are crack detection on road pavements [13, 23, 30, 8, 32, 35, 4] and crack detection on bridges [22, 27, 16, 7, 33]. There are however a variety of different applications such as crack detection on steel [26] or nuclear power plants [29]. Some methods try to detect cracks in a more general manner such as DeepCrack [18]. The authors of [14] provide a review on defect detection - including cracks - on concrete and asphalt civil infrastructures.

Edge detections algorithms, such as Sobel, Canny or the Fast Haar Transform have been studied for crack detection [1]. However, as shown in Figure 3, these operators work on easy examples such as dark cracks on light walls, but fail when the texture of the surface is too important. Moreover, they often need adapted preprocessing and post-processing steps. The same is true for threshold-based approaches [21] or percolation-based image processing techniques [31].

These methods fail as a semantic understanding of the scene is necessary to separate cracks and non-cracks. The concrete surface can indeed contain lines as in Figure 1b or have a very heterogeneous texture as shown in Figure 1g. Moreover, cracks can be particularly large as in Figure 1e, detecting them can therefore require a large scale analysis to differentiate them from dark walls.

As a consequence, methods leveraging machine learning have been developed. Abdel-Qader et al. [2] combine edge detectors and PCA to classify images as containing cracks or not. Shi et al. [30] use a random forest to classify each pixel as belonging to a crack or not. Mundt et al. [22] leverage convolutional neural networks to classify defects - cracks being among them - on pictures of bridges.

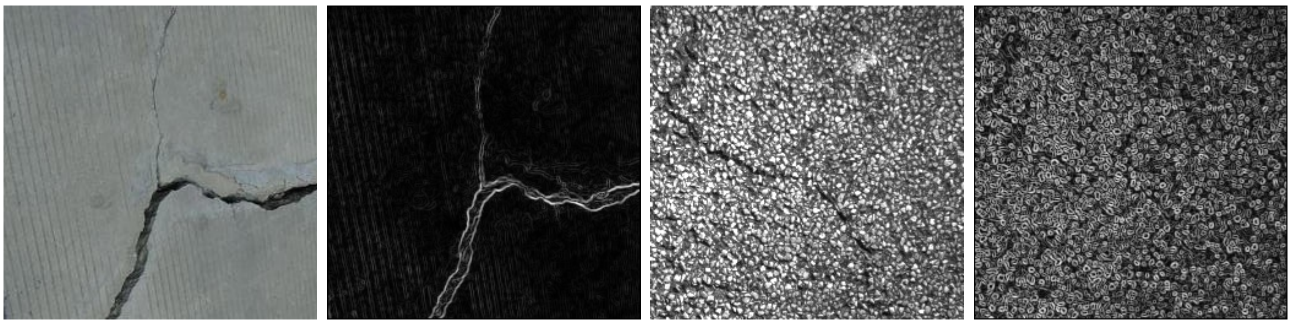

(a) Easy example (b) Sobel applied on (a) (c) Hard example (d) Sobel applied on (c)

Figure 3: Results of the Sobel operator on easy and hard examples.

Several methods [18, 17, 15] use deep learning semantic segmentation methods to segment images into crack / non-crack areas.

In order to train machine learning and deep learning methods, several datasets have been released by the scientific community. Datasets of road pavement images are particularly numerous [30, 8, 32, 35, 4, 6]. Among them, only [30], [32] and [4] provide semantic segmentation labels. CrackTree [35] only provides a fine line representation of the cracks, their widths are therefore not provided. GAPs [8] and AsphaltCrack [6] classify each image as containing a crack or not, no semantic segmentation annotations are provided.

Codebrim [22] is a dataset classifying defects on bridge images. SDNet2018 [20] and DeepCrack [18] are more general datasets and contain images from various areas (bridges, walls and pavements). However, only DeepCrack provides semantic segmentation labels; SDNet2018 classifies images as containing cracks or not.

3 Experimental Setup

3.1 Dataset Creation

The dataset was created by combining Crack500 [32], DeepCrack [18], SDNet2018 [20] and CrackForest [30]. The dataset was divided into three parts: training, validation and testing. For allowing an easier comparison with the state of the art, the initial repartition of each dataset was respected when possible: for instance, an image in the training part of the dataset of Crack500 stayed in the training part of the combined dataset. DeepCrack only provided training and validation subsets: a testing subset was created by separating the validation subset in two parts. SDNet2018 and CrackForest didn't provide any training, validation or testing subsets: they were randomly separated into three parts. Filenames used in each set are available online. The size of each dataset is shown in Table 1. There is an important imbalance between the datasets both in terms of number of images and number of pixels. We describe in Section 3.2 how we manage this imbalance.

Subset	Training	Validation	Test
Crack500	250 / 922M	50 / 184M	62 / 229M
SDNet2018	800 / 82M	200 / 13M	80 / 5M
CrackForest	81 / 12M	23 / 4M	14 / 2M
DeepCrack	300 / 63M	138 / 29M	56 / 12M

Table 1: Size of datasets in number of images / number of pixels.

As SDNet2018 didn't provide any semantic segmentation labels, we used our model trained on

Crack500, DeepCrack and CrackForest to produce semantic segmentation labels on a part of SD-Net2018 (1000 images). These labels were then manually corrected. As the dimensions of the images were different from one dataset to another, they were divided into patches of 256 $\times$ 256 pixels to make the learning process easier.

3.2 Model Architecture and Training

As explained in Section 2, segmenting an image into crack and non-crack areas requires to take into account lower scale and higher scale features. The chosen model architecture should therefore be able to take into account multi-scale features. A U-net-like neural network [28] is particularly adapted to such detections problems, as its encoder-decoder architecture allows to perceive the image both at fine and coarse levels. Our architecture is therefore similar, but we added batch normalization layers [11] to speed up the training and train more robust models. This approach is similar to SegNet [5], except that the number of convolutions is more limited and there is only one batch normalization before each max pool or up-convolution. This has been done to reduce the number of parameters and memory usage. The architecture is shown in Figure 4. The loss used for training is binary cross entropy

$$L = -\frac{\sum_{i,j} y[i,j]\log(p[i,j]) + (1-y[i,j])\log(1-p[i,j])}{w \times h},$$

where $y[i,j]$ is the ground truth label at pixel (i,j) (0 being non-crack, 1 being crack), $p[i,j]$ is the predicted probability that the pixel (i,j) is crack, w and h are the width and height of the patch.

The neural network takes a grey level image as input and outputs the semantic segmentation map which is a 2D matrix of values between 0 and 1 representing the likelihood that a pixel is in a crack. RGB images are converted to grey level images to maximize the generalisability of the model: the neural network is less likely in this case to be disturbed by colors that were never represented in the dataset.

During training, random flips, rotation, zooms (between -20% and +20%), illumination and contrasts changes were done in order to augment the merged dataset (see Algorithm 1). The size of the batches was set to 8, and they were constructed so that each dataset was represented approximately equally to prevent larger datasets to have a higher impact than smaller ones in the training process (see Algorithm 2). We used Adam [12] as the optimizer, with $\alpha = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$, as recommended in the original paper. The momentum of batch normalization layers was set to 0.99. We also used the `ReduceLROnPlateau` functionality of Keras[2]: during the training process, the learning rate is multiplied by 0.2 if a new low in the validation loss does not appear for 10 epochs.

3.3 Cross-Dataset Study

To estimate the generalisability of our model, we have conducted a cross-dataset study which consists of training the model on a subset of datasets and testing it on a different subset.

As described in Section 3.2, the output map of our neural network is a likelihood map where each pixel has a value comprised between 0 and 1 representing the likelihood that the pixel is in a crack. To evaluate these likelihood maps, we propose to use two metrics: a modified version of the F_1 score [10], and the Area Under the Curve of the ROC curve [9].

For the first metric, we transform our likelihood map into a binary map by applying a threshold $b(x,y) = l(x,y) > t$ with $t = 0.5$, l being the likelihood map and b the resulting binary map. A

[2]Keras, Chollet, F. and others, 2015, https://keras.io

Algorithm 1: Patch preprocessing during training

```
input  : patch
output: patch (processed)
avg = mean(patch)
patch -= avg
patch *= RandomNumberBetween 0 And 1 + 0.5
patch += avg + (RandomNumberBetween 0 And 1 - 0.5) × 0.3
if RandomCoinFlip = True then
    patch = VerticalFlip(patch)
if RandomCoinFlip = True then
    patch = HorizontalFlip(patch)
Apply random rotation on patch (set undefined pixels to 0)
Apply random zoom, between 0.8 and 1.2 on patch (set undefined pixels to 0)
return patches
```

Algorithm 2: Batch dataset balancing

```
input  : patchesListPerDataset, sizeOfBatch
output: patches
nbPatches=$\frac{\text{sizeOfBatch}}{\text{number of datasets}}$
patches = Empty Array()
foreach dataset in patchesListPerDataset do
    for i in range(0, nbPatches) do
        patches.append(pick random patch from patchesListPerDataset[dataset])
return patches
```

classical metric for measuring classification tasks is the F_1 score [10], which takes into account both the precision and the recall of the classifier

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}},$$

with

$$\text{precision} = \text{fraction of true positives among all detections} = \frac{tp}{tp + fp} = \frac{|\{b = 1\} \cap \{g = 1\}|}{|\{b = 1\}|},$$

$$\text{recall} = \text{fraction of true positives succesfully detected} = \frac{tp}{tp + fn} = \frac{|\{b = 1\} \cap \{g = 1\}|}{|\{g = 1\}|},$$

where b is the binary segmentation map ($\{b = 1\}$ is the set of points $\{(i, j),\ b(i, j) = 1\}$), g is the ground truth, and tp, fp and fn are respectively the number of true positives, false positives and false negatives. Although this metric is adapted to most classification tasks, it can be ill-adapted to semantic segmentation problems where thin objects are represented, as it is the case with cracks. Indeed, the width of cracks is generally between 2 and 5 pixels in the databases under study. As illustrated in Figure 5, if the width of the crack is 3 pixels and there is a shift of just one pixel between the detection and the ground truth, the precision, recall and F_1 scores all drop to $\frac{2}{3}$ where it would have been 1 if the semantic segmentation was aligned correctly. This drop is very important considering that, due to inaccuracies and inconsistencies intra and inter-dataset during the labeling process, the ground truth often exhibits such discrepancies. Moreover, it is not necessary to be precise down to the pixel level: a shift of a few pixels is not likely to modify the crack's diagnosis.

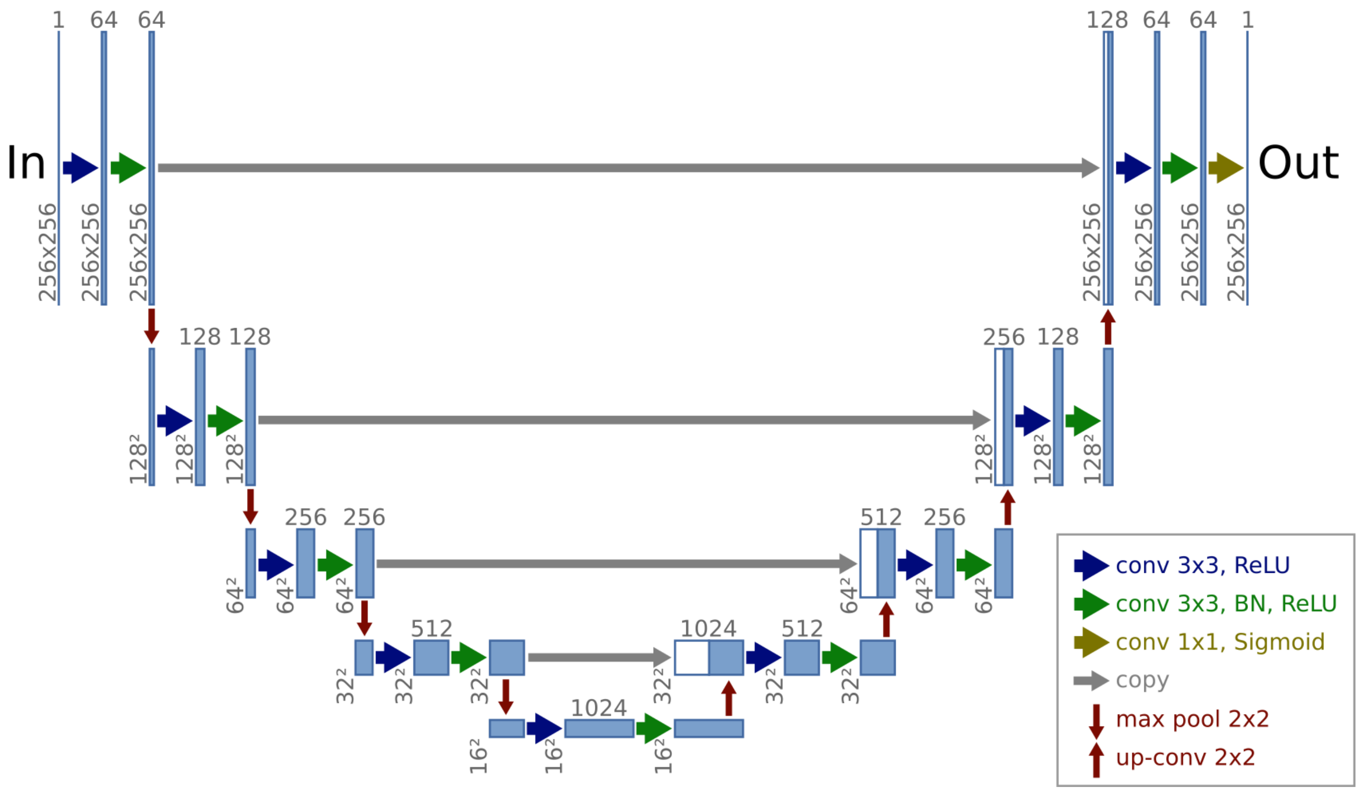


Figure 4: Architecture of our model. In: the input image. Out: the output semantic segmentation. BN: Batch Normalization.

We therefore introduce a new metric that does not penalize small spatial shifts and distortions in the prediction compared to the ground truth. The main idea is to consider a detection as correct if a positive in the ground truth exists at a distance strictly less than θ, and to consider a positive in the ground truth successfully detected if a detection exists at a distance $< \theta$. We therefore define θ-precision and θ-recall as:

$$\theta\text{-precision} = \text{fraction of detected points at a distance} < \theta \text{ to a positive in the ground truth},$$

$$\theta\text{-precision} = \frac{|\{b=1\} \cap \{\delta_\theta(g)| = 1\}}{|\{b=1\}|},$$

$$\theta\text{-recall} = \text{fraction of positives in the ground truth at a distance} < \theta \text{ of a detection},$$

$$\theta\text{-recall} = \frac{|\{\delta_\theta(b)=1\} \cap \{g=1\}|}{|\{g=1\}|},$$

where b is the binary segmentation map, g the ground truth and $\delta_\theta(x)$ the binary morphological dilation of size θ.

We can then define θ-F_1:

$$\theta\text{-}F_1 = 2 \times \frac{\theta\text{-precision} \times \theta\text{-recall}}{\theta\text{-precision} + \theta\text{-recall}}.$$

Figure 6a shows the relationship between θ and θ-precision, θ-recall and θ-F_1.

The second metric evaluates the likelihood map as a whole. The ROC curve represents for all possible thresholds the relationship between the true positive rate (ordinate) - the percentage of

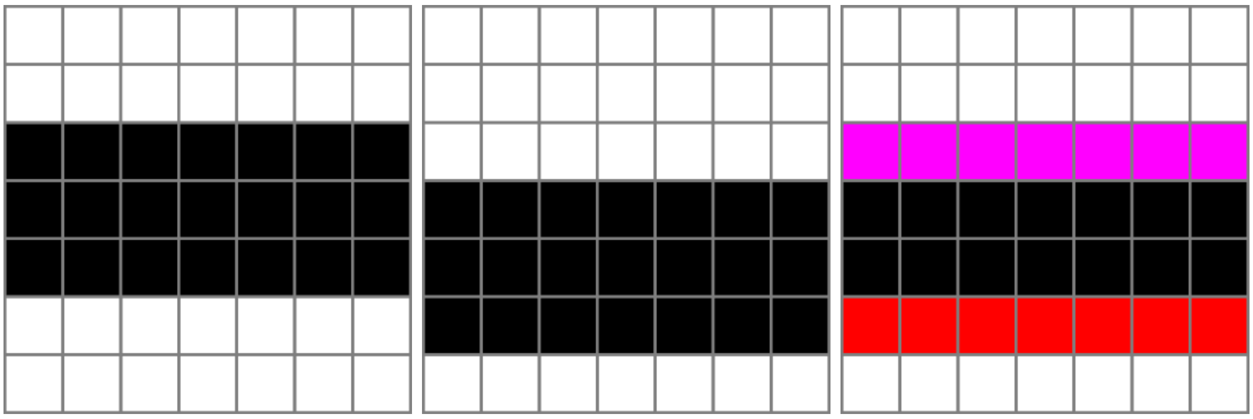


Figure 5: Limitation of the F1 score. TP: True Positive (black in (c)). FP: False Positive (red in (c)). FN: False Negative (purple in (c)).

pixels detected as cracks and labelled as cracks in the ground truth - and the false positive rate (abscissa) - the percentage of pixels detected as cracks but not labelled as cracks. An example of such a curve can be found in Figure 7. A good classification algorithm should maximize the true positive rate while minimizing the false positive rate, in other words maximize the Area Under the Curve (AUC). An area of 0.5 (or 50%) means that the false positive rate is increasing as fast as the true positive rate: in other words, the algorithm is not better than random draws. The best area values are close to 1 (or 100%) as it means that there exists a threshold detecting most positives with very few false positives.

Tables 2, 3 and 4 show the performance of all considered networks when trained on a single dataset and tested on another one. As expected, the results are best when a network is trained and tested on the same dataset. F_1 scores can be poor, especially when the model has been trained on one dataset and tested on another, but are significantly improved when using θ-F_1 with $\theta = 5$, suggesting that these low measured performances are mostly due to differences during the labelling process. Indeed, the ground truth represents cracks with different widths depending on the dataset. Overall, the θ-recall is lower than the θ-precision, implying that there is a higher number of missed cracks than false positives. Networks trained on other datasets seem to have an important drop in performance when tested on Crack500, suggesting that this dataset has some unique cracks and features. The model on CrackForest is the worst in terms of generalization, which can be explained by the fact that CrackForest contains the least number of images.

	Crack500			**SDNet2018**			**CrackForest**			**DeepCrack**		
	P.	*R.*	F1	*P.*	*R.*	F1	*P.*	*R.*	F1	*P.*	*R.*	F1
Crack500	*76%*	*58%*	66%	*53%*	*46%*	49%	*75%*	*37%*	49%	*74%*	*75%*	75%
SDNet2018	*37%*	*27%*	31%	*68%*	*64%*	66%	*57%*	*69%*	62%	*75%*	*62%*	68%
CrackForest	*67%*	*18%*	29%	*76%*	*26%*	38%	*78%*	*60%*	68%	*76%*	*34%*	47%
DeepCrack	*66%*	*34%*	44%	*53%*	*55%*	54%	*84%*	*26%*	40%	*74%*	*78%*	76%

Table 2: Cross dataset performance study: performance when training on a single dataset (1/3). The first column indicates the dataset on which the neural network has been trained. The first line indicates which dataset it has been tested on. The performance metrics are the classical precision (P.), recall (R.) and F_1: 0% represents the worst performance, 100% is the best performance.

Table 5 shows the AUC performance of the networks when trained on all datasets except one and tested on the remaining one. Although the performance is slightly worse compared to when the

	Crack500			**SDNet2018**			**CrackForest**			**DeepCrack**		
	θ-P.	*θ-R.*	*θ*-F1	*θ-P.*	*θ-R.*	*θ*-F1	*θ-P.*	*θ-R.*	*θ*-F1	*θ-P.*	*θ-R.*	*θ*-F1
Crack500	*92%*	*82%*	87%	*56%*	*63%*	59%	*99%*	*59%*	74%	*81%*	*91%*	86%
SDNet2018	*47%*	*43%*	45%	*77%*	*81%*	79%	*99%*	*81%*	89%	*87%*	*81%*	84%
CrackForest	*80%*	*38%*	52%	*79%*	*47%*	59%	*99%*	*93%*	96%	*84%*	*53%*	65%
DeepCrack	*83%*	*51%*	63%	*57%*	*77%*	65%	*99%*	*49%*	66%	*83%*	*90%*	86%

Table 3: Cross dataset performance study: performance when training on a single dataset (2/3). The first column indicates the dataset on which the neural network has been trained. The first line indicates which dataset it has been tested on. The performance metrics are the θ-precision (θ-P.), θ-recall (θ-*R.*) and θ-F1 introduced in Section 3.3 with $\theta = 5$: 0% represents the worst performance, 100% is the best performance.

	Crack500	**SDNet2018**	**CrackForest**	**DeepCrack**
Crack500	98.8%	90.9%	94.3%	97.5%
SDNet2018	84.4%	97.7%	94.8%	97.2%
CrackForest	73.8%	77.9%	98.9%	81.6%
DeepCrack	87.0%	93.3%	88.4%	97.6%

Table 4: Cross dataset performance study: performance when training on a single dataset (3/3). The first column indicates the dataset on which the neural network has been trained. The first line indicates which dataset it has been tested on. The performance metric is the Area Under Curve of the ROC curve [9]: 0% represents the worst performance, 100% is the best performance.

network is trained and tested on the same dataset, it is consistently better than when we train a network on one dataset and test it on another. Also, the worst combination achieves an AUC of 94.3%, whereas the worst AUC of the best network trained on a single dataset is 90.9%, suggesting that training on several datasets produces more robust networks. We have done this cross-validation for multiple architectures in Table 6. Our method performs better on the worst performing combination than any of the other tested architectures, suggesting that it generalizes better. Also, training our architecture on each dataset separately, then evaluating each neural network on the test set and combining the predictions using different operators (min, median, avg, max) doesn't produce better results.

Trained on	**Tested on**	**AUC**	**Diff. 1**	**Diff. 2**
SDNet2018+CrackForest+DeepCrack	Crack500	94.3%	-4.5%	+8.4%
CrackForest+Crack500+DeepCrack	SDNet2018	94.5%	-3.2%	+1.3%
SDNet2018+Crack500+DeepCrack	CrackForest	95.1%	-3.8%	+0.3%
SDNet2018+CrackForest+Crack500	DeepCrack	97.8%	+0.1%	+0.3%

Table 5: Cross dataset performance study: performance when training on several datasets using our approach. AUC: Area Under Curve of the ROC curve [9]: 0% represents the worst performance, 100% is the best performance. Diff. 1: how much the AUC has changed compared to when we trained the model on the dataset we tested it on. Diff. 2: how much the AUC has improved compared to the best model trained on a single dataset different from the one it was tested on.

3.4 Quantitative and Qualitative Results

Table 7 shows the AUC performance of multiple models (trained on all datasets when possible). The AUC performance of our proposed network is comprised between 97% and 99% depending on the dataset, and its performance is close to, or better, than the one of networks trained and tested on the same dataset.

Architecture	Min. AUC	Max. AUC
FCN-8 [19]	**94.1%**	97.2%
FCN-32 [19]	91.6%	95.3%
PSPNet [34]	91.3%	95.1%
SegNet [5]	90.3%	97.4%
Our method	**94.3%**	**97.8%**
Spe. (min.)	68.4%	88.1%
Spe. (median)	90.1%	97.6%
Spe. (avg.)	93.1%	**98.4%**
Spe. (max.)	92.8%	98.2%

Table 6: Cross dataset performance study: performance range when training on several datasets using different architectures. AUC: Area Under Curve of the ROC curve [9]: 0% represents the worst performance, 100% is the best performance. Min. AUC & Max. AUC : minimum and maximum AUC obtained when combining 3 datasets and testing on another. Spe. (method): for each sample, we run our architecture trained on each dataset, and aggregate the detection using the minimum, median, average and maximum.

Method	Crack500	SDNet2018	CrackForest	DeepCrack	All
CrackIT[24] *	76.1%	65.5%	77.5%	71.9%	72.0%
DeepCrack[18] **	95.2%	93.2%	97.0%	97.8%	96.0%
FCN 8 [19]	**97.4%**	**97.3%**	**98.0%**	**98.2%**	**97.9%**
FCN 32 [19]	95.8%	94.6%	95.2%	96.5%	95.8%
PSPNet [34]	**97.9%**	95.8%	94.4%	96.9%	96.8%
SegNet [5]	96.5%	97.0%	97.4%	95.9%	96.8%
Our method	97.2%	**98.1%**	**98.5%**	**98.6%**	**98.2%**

Table 7: Performance of the model trained on all datasets. AUC: Area Under Curve of the ROC curve [9]: 0% represents the worst performance, 100% is the best performance. Bold: best results. Gray bold: second best results. * The CrackIT method doesn't use machine learning, so it was not trained on the dataset. ** Contrary to all other deep learning methods in the table, DeepCrack was not trained on our dataset.

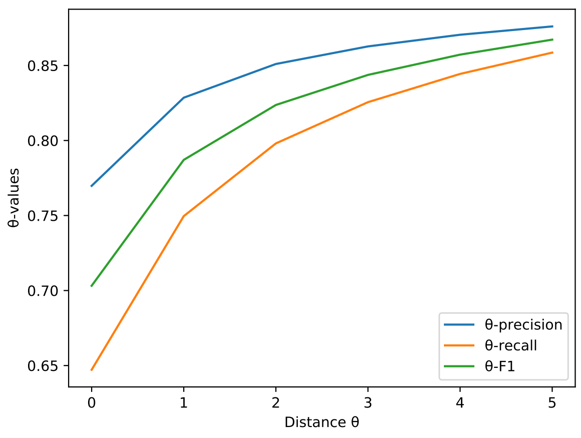
Table 8 shows the recall, precision and F_1 performance metrics as well as their θ counter-part. Figure 6b shows the relationship between θ and θ-F_1. Overall, there is an increase of 15 to 20 percentage points when comparing F_1 scores to θ-F_1 scores with $\theta = 5$: a large part of the errors measured by F_1 are due to small discrepancies between the binary map and the ground truth around cracks. Precision is here again larger than recall: there are more missed cracks than false positives. Our modified U-net method seems to perform better than other state of the art methods.

Method	Precision	Recall	F1	θ-Precision	θ-Recall	θ-F1
CrackIT	59%	26%	36%	68%	43%	52%
DeepCrack	50%	**80%**	*61%*	74%	**88%**	80%
FCN8	**76%**	52%	**61%**	**88%**	76%	**82%**
FCN32	73%	20%	31%	**88%**	31%	46%
PSPNet	65%	21%	32%	**84%**	35%	49%
SegNet	72%	47%	57%	**88%**	63%	74%
Our method	**77%**	**65%**	**70%**	**88%**	**86%**	**87%**

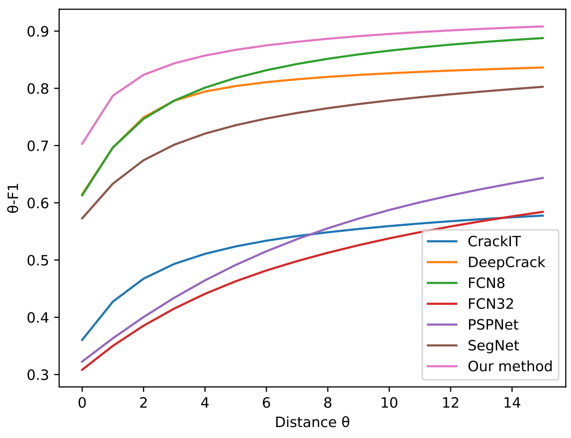
Table 8: Performance metrics for the binary segmentation map for several methods. $\theta = 5$. Bold: best results. Gray bold: second best results.

The ROC curve of the benchmark's methods on the test set is displayed in Figure 7.

Our method can also be used for classifying whole patches. For doing so, we define a patch as



(a) θ-precision, θ-recall and θ-F_1 for our method.



(b) θ-F_1 for all methods.

Figure 6: Relationship between θ-F_1 and θ.

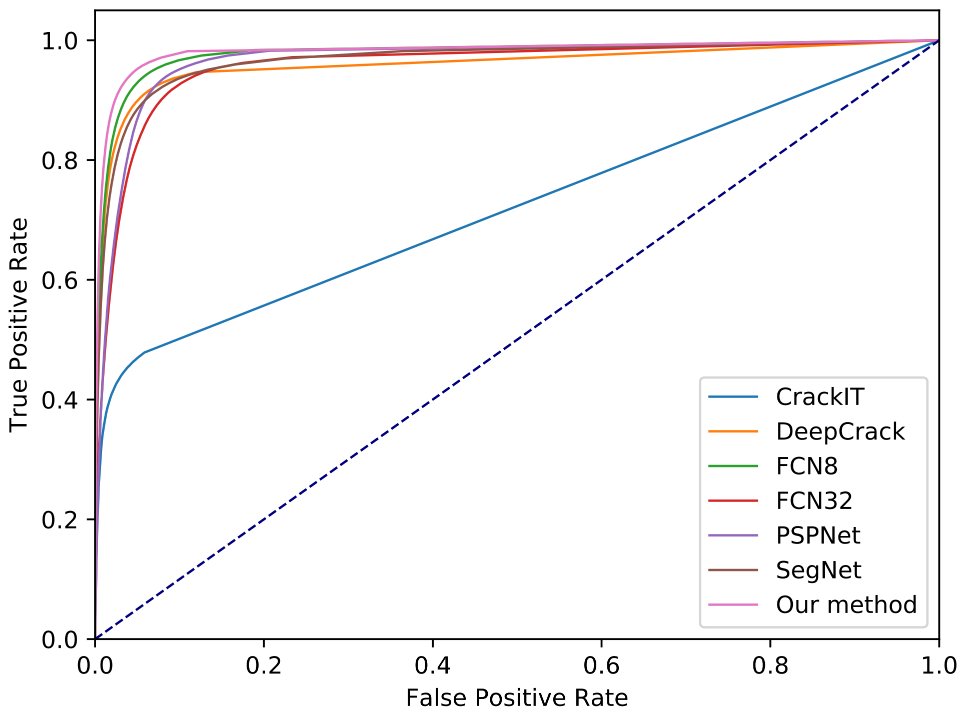
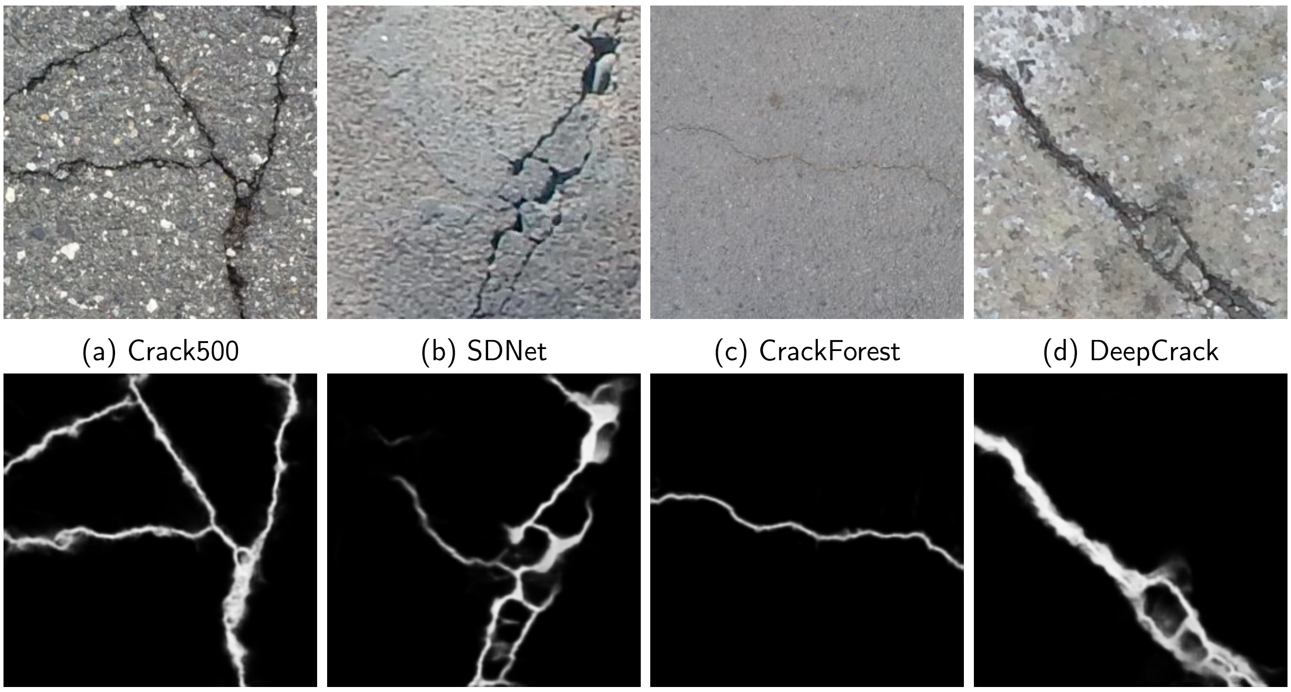


Figure 7: ROC curve of our benchmark on all datasets.

containing a crack if there exists a pixel inside it marked as belonging to a crack in the ground truth. If we define $S = \sum_{\forall x \in p} p(x)$, $p(x)$ being the prediction mask, the optimal threshold maximizing the F_1 score is $S > 176.09$, achieving an F_1 score of 89%, a precision of 92% and a recall of 87%.

Examples of predictions of our U-net on the merged dataset are shown in Figure 8. Figure 9 shows an example of prediction on an image of ground crack, a type of crack that the network was never trained on, suggesting that the network generalizes well.



(a) Crack500 (b) SDNet (c) CrackForest (d) DeepCrack

(e) Prediction on (a) (f) Prediction on (b) (g) Prediction on (c) (h) Prediction on (d)

Figure 8: Examples of predictions of the model trained on the merged dataset.

3.5 Robustness Enhancement using Extra-Dataset Images

We have shown in Section 3.4 that combining several datasets during training enhances the robustness of detectors. However, a limitation of the combined dataset we have constructed is that it focuses on images of concrete structures containing cracks. As it doesn't contain images of more general scenes, the trained detectors fail when tested on images containing different objects (see Figure 10).

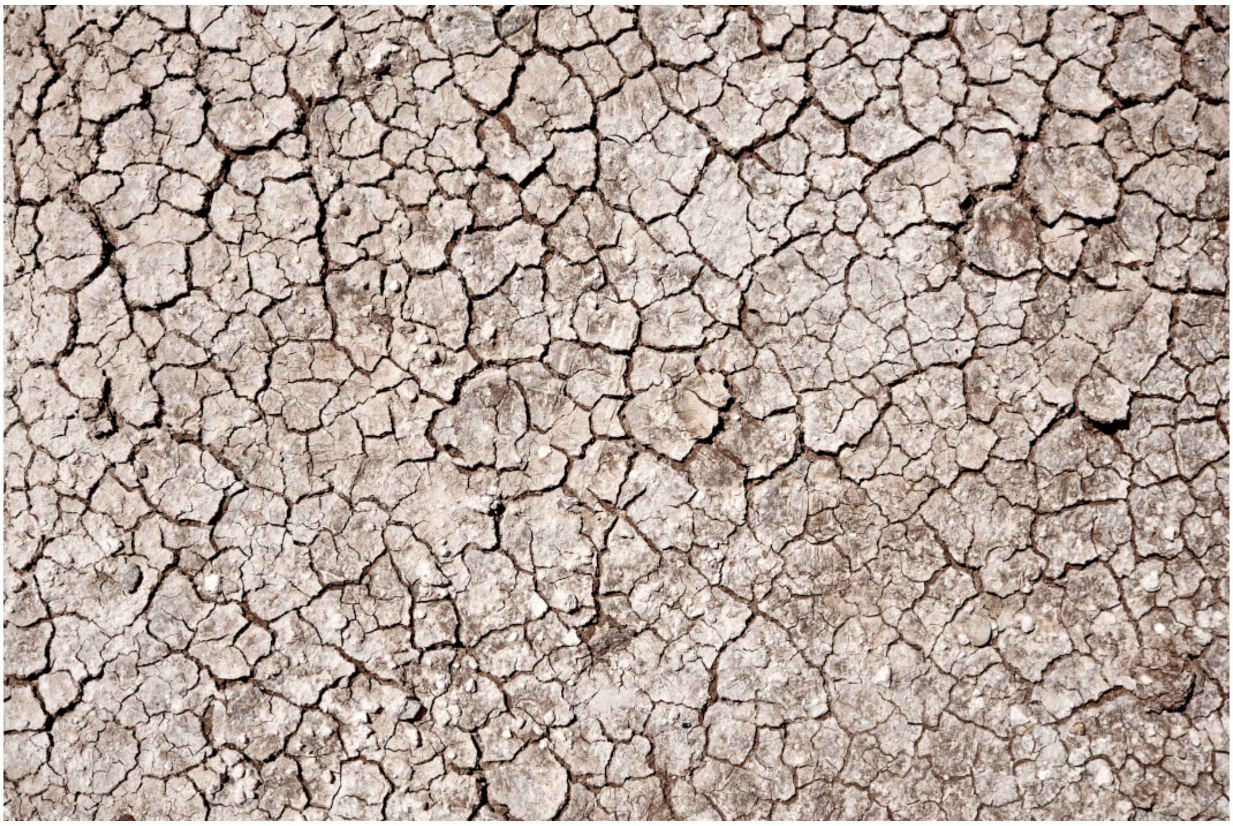
In order to make our crack detector more robust, we have added in our merged dataset an additional set of 3893 images from the Flickr[3] website (a list of image URLs will be made available). These images served as negative examples, and a manual check was done to check that they contained no cracks.

Some comparative results are shown in Figure 10 and Table 9. The detector trained on the augmented dataset is less sensitive than when trained on the original dataset (precision is higher and recall is lower): this is an expected effect of adding images with only negative examples. We have therefore shown in Table 9 the performance of the model trained on the original dataset with the standard threshold of 0.5 and an additional threshold of 0.75 resulting in a similar precision and recall than what is achieved by the model trained on the augmented dataset. The number of false crack detections on the Flickr dataset is much lower: there are, on average, 40 times less false positive pixels when using the model trained on the augmented dataset than the one on the original dataset. This observation remains true even after adjusting the threshold on the model trained on the original dataset: the number of false positive pixels is 12 times lower in that case.

3.6 A Few Notes about Inference

Although the neural network has been trained on 256×256 patches, it is important to note that the detector can be tested on larger images as well.

[3] `https://www.flickr.com/`


(a) Ground crack

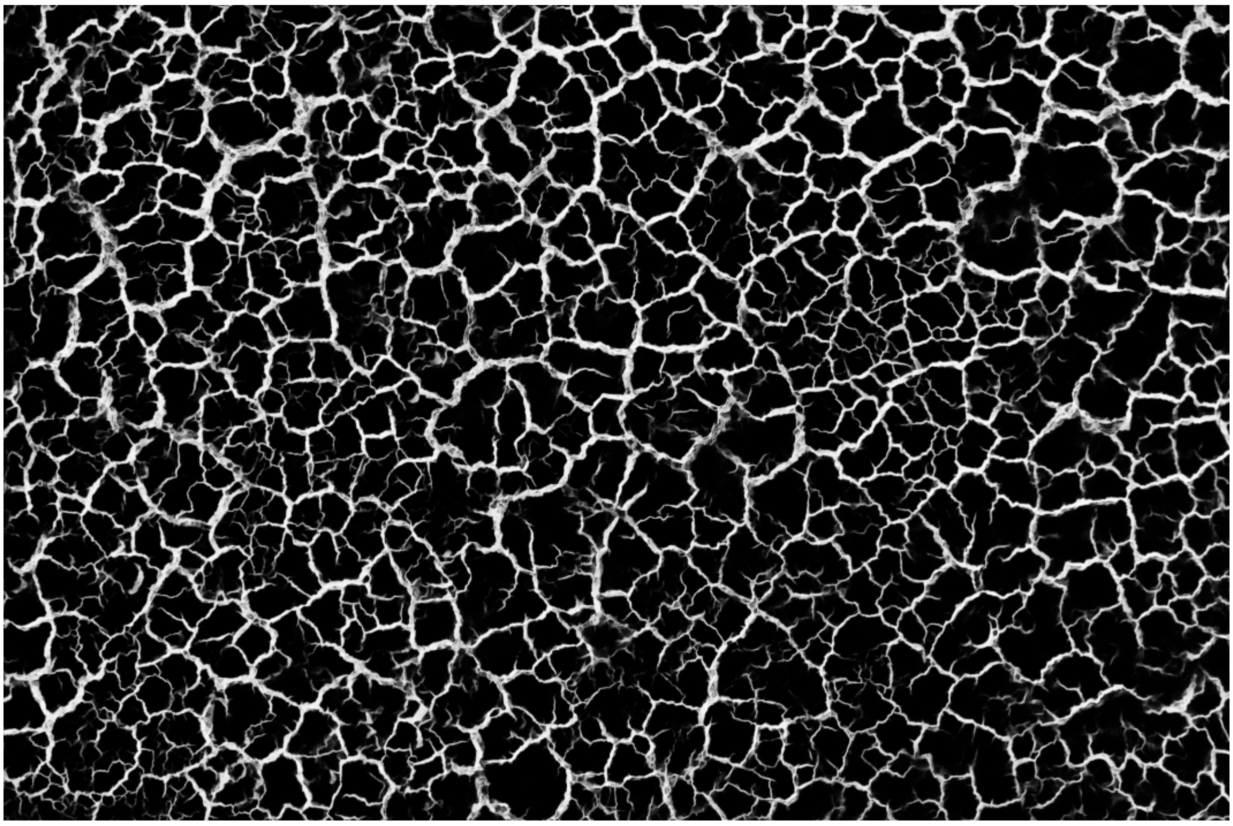

(b) Prediction on (a)

Figure 9: Example of prediction on an image of ground crack, a type of crack that the model was never trained on.

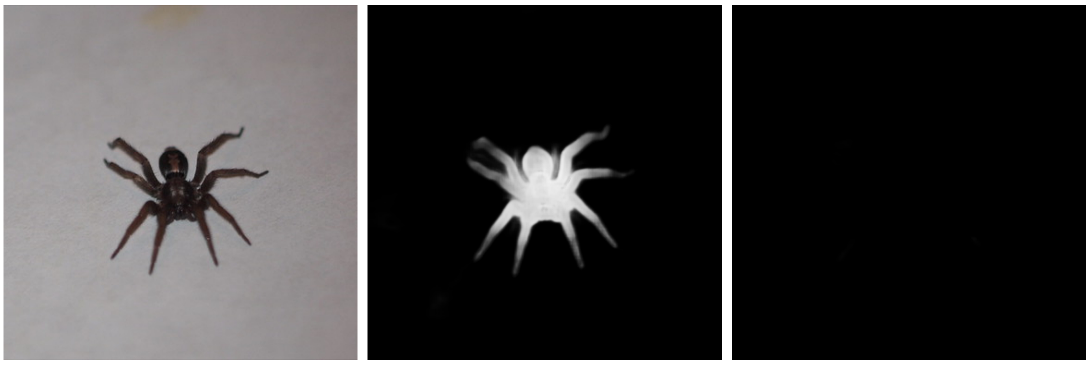

(a) Image of a spider (b) Detection when trained on original dataset (c) Detection when trained on augmented dataset

Figure 10: Example of problematic image: an image of a spider on a wall is detected as a crack when using U-net trained on the original merged dataset. However, when using the augmented dataset, our detector correctly classifies the spider as non crack.

Trained on	θ-Precision	θ-Recall	θ-F1	AFD
Original dataset (t = 0.50)	88%	86%	87%	3491
Original dataset (t = 0.75)	92%	70%	80%	1080
Augmented dataset (t = 0.50)	93%	70%	80%	88

Table 9: Performance metrics for the binary segmentation map for several methods. $\theta = 5$. θ-Precision, θ-Recall and θ-F1 are computed on the original dataset. t: threshold used for classifying pixels as cracks / non cracks. AFD: Average number of pixels per image falsely classified as cracks on the Flickr dataset.

A first approach can be to simply provide a larger image to the neural network. The only requirement in that case is that the images width and height must be multiples of 16 so that the output size is the same as the input size. A limitation of this approach is that the memory requirements for processing the image might exceed the GPU or RAM specifications.

Our algorithm therefore uses another approach. The neural network is evaluated on overlapping sliding windows of the image and predictions are then concatenated and aggregated as described in Algorithms 3 and 4. This approach allows us to be more permissive on the size of the image at the cost of a longer inference time.

Batch normalization layers are fixed during inference.

4 Conclusion

We have presented a general deep learning method for detecting cracks on textured surfaces. For making this method robust to different types of cracks and acquisition procedures, we have trained our method on four datasets - Crack500 [32], DeepCrack [18], SDNet2018 [20], CrackForest [30]. We have also labelled a part (1000 images) of the SDNet2018 dataset so that it contains semantic labels, as it originally only proposed crack / non-crack classifications on the image level. In order to make our method even more robust to different objects, scenes and illuminations, we have also added images from the Flickr website, leading to an important drop in false positives on extra dataset images.

To validate our approach, we have done a cross-dataset study where we trained the model on a subset of the datasets and tested it on another subset. Results of this study show that training the model on the several datasets makes it more robust to new images.

Overall, the network's predictions are satisfying even on images outside the merged dataset.

```
Algorithm 3: Sliding windows inference
  input  : image, neuralNetwork, windowSize, nbOutputLayers
  output: predictions
  // Image dimension is width × height
  // windowSize is 256 in our case.
  // nbOutputLayers is 1 in our case.
  windowWeights = GetWindowWeights(windowDimensions)
  // See Algorithm 4
  windows = get 50% overlapping sliding windows positions of size windowSize on image
  sumMat = matrix filled with zeros of dimensions width × height × nbOutputLayer
  foreach window in windows do
      sumMat[window] += windowWeights
  prediction = matrix filled with zeros of dimensions width × height × nbOutputLayer
  foreach window in windows do
      exTarget = extract window from image;
      exPrediction = neuralNetwork.predict(exTarget);
      // This is a simplification. If memory allows it, the best is to predict on batches.
      prediction[window] = exPrediction × windowWeights / sumMat[window]
  return patches
```

```
Algorithm 4: GetWindowWeights
  input  : windowSize, unreliableD
  output: windowsWeights
  // Predictions are not reliable near the borders of the patch, unreliableD is the distance at which we
  estimate it is not reliable.
  windowsWeights = binary matrix filled with zeros of dimensions windowSize × windowSize
  // For each pixel in windowsWeights, set its value to the distance to the nearest border.
  windowsWeights[1:-1, 1:-1] = 1
  windowsWeights = distanceTransform(windowsWeights)
  windowsWeights = (windowsWeights - unreliableD) / unreliableD
  windowsWeights[windowsWeights < epsilon] = epsilon
  // epsilon being a very small number, to handle images borders.
  windowsWeights[windowsWeights > 1] = 1
  // Pixel value is epsilon when distance to border is less than unreliableD.
  // Pixel value is 1 when distance to border is more than 2 × unreliableD.
  // Otherwise, it is between 0 and 1.
  return windowsWeights
```

Acknowledgment

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. Work partly financed by Office of Naval research grant N00014-17-1-2552, DGA Astrid project «filmer la Terre» n° ANR-17-ASTR-0013-01, DGA Defals challenge n° ANR-16-DEFA-0004-01.

Image Credits

Images from Crack500 [32], DeepCrack [18], SDNet2018 [20], CrackTree [35], CCIC [25], Codebrim [22], AigleRN [4] and CrackForest [30] datasets.

Designed by welcomia / Freepik[4]

Crop of picture by Louis (lostinfog) / Flickr[5]

References

[1] I. Abdel-Qader, O. Abudayyeh, and M.E. Kelly, *Analysis of edge-detection techniques for crack identification in bridges*, Journal of Computing in Civil Engineering, 17 (2003), pp. 255–263. https://doi.org/10.1061/(ASCE)0887-3801(2003)17:4(255).

[2] I. Abdel-Qader, S. Pashaie-Rad, O. Abudayyeh, and S. Yehia, *PCA-based algorithm for unsupervised bridge crack detection*, Advances in Engineering Software, 37 (2006), pp. 771–778. https://doi.org/10.1016/j.advengsoft.2006.06.002.

[3] R. Abraham, M. Bergounioux, and P. Debs, *Automatic Choice of the Threshold of a Grain Filter via Galton—Watson Trees: Application to Granite Cracks Detection*, Journal of Mathematical Imaging and Vision, 60 (2018), pp. 50–69. https://doi.org/10.1007/s10851-017-0743-3.

[4] R. Amhaz, S. Chambon, J. Idier, and V. Baltazart, *Automatic crack detection on two-dimensional pavement images: An algorithm based on minimal path selection*, IEEE Transactions on Intelligent Transportation Systems, 17 (2016), pp. 2718–2729. https://doi.org/10.1109/TITS.2015.2477675.

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla, *SegNet: A deep convolutional encoder-decoder architecture for image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 39 (2017), pp. 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615.

[6] A.J. Balaji, G.T. Balaji, M.S. Dinesh, B.B. Nair, and D.S.H. Ram, *Asphalt crack dataset*. http://dx.doi.org/10.17632/xnzhj3x8v4.2.

[7] S. Chanda, G. Bu, H. Guan, J. Jo, U. Pal, Y-C. Loo, and M. Blumenstein, *Automatic bridge crack detection–a texture analysis-based approach*, in Proceedings of IAPR Workshop on Artificial Neural Networks in Pattern Recognition, Springer, 2014, pp. 193–203. https://doi.org/10.1007/978-3-319-11656-3_18.

[8] M. Eisenbach, R. Stricker, D. Seichter, K. Amende, K. Debes, M. Sesselmann, D. Ebersbach, U. Stoeckert, and H-M. Gross, *How to get pavement distress detection ready for deep learning? a systematic approach*, in Proceedings of International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2039–2047. https://doi.org/10.1109/IJCNN.2017.7966101.

[9] T. Fawcett, *An introduction to ROC analysis*, Pattern recognition Letters, 27 (2006), pp. 861–874. https://doi.org/10.1016/j.patrec.2005.10.010.

[4]http://www.freepik.com
[5]https://flic.kr/p/cr8xx1

[10] C. Goutte and E. Gaussier, *A probabilistic interpretation of precision, recall and F-score, with implication for evaluation*, in Proceedings of European Conference on Information Retrieval, Springer, 2005, pp. 345–359. https://doi.org/10.1007/978-3-540-31865-1_25.

[11] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, arXiv preprint arXiv:1502.03167, (2015). https://arxiv.org/abs/1502.03167.

[12] D.P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014). https://arxiv.org/abs/1412.6980.

[13] K.R. Kirschke and S.A. Velinsky, *Histogram-based approach for automated pavement-crack sensing*, Journal of Transportation Engineering, 118 (1992), pp. 700–710. https://doi.org/10.1061/(ASCE)0733-947X(1992)118:5(700).

[14] C. Koch, K. Georgieva, V. Kasireddy, B. Akinci, and P. Fieguth, *A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure*, Advanced Engineering Informatics, 29 (2015), pp. 196–210. https://doi.org/10.1016/j.aei.2015.01.008.

[15] D. Lee, J. Kim, and D. Lee, *Robust concrete crack detection using deep learning-based semantic segmentation*, International Journal of Aeronautical and Space Sciences, 20 (2019), pp. 287–299. https://doi.org/10.1007/s42405-018-0120-5.

[16] B. Lei, N. Wang, P. Xu, and G. Song, *New crack detection method for bridge inspection using UAV incorporating image processing*, Journal of Aerospace Engineering, 31 (2018), p. 04018058. https://doi.org/10.1061/(ASCE)AS.1943-5525.0000879.

[17] Y. Li, H. Li, and H. Wang, *Pixel-wise crack detection using deep local pattern predictor for robot application*, Sensors, 18 (2018), p. 3042. https://doi.org/10.3390/s18093042.

[18] Y. Liu, J. Yao, X. Lu, R. Xie, and L. Li, *DeepCrack: A deep hierarchical feature learning architecture for crack segmentation*, Neurocomputing, 338 (2019), pp. 139–153. https://doi.org/10.1016/j.neucom.2019.01.036.

[19] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3431–3440. https://doi.org/10.1109/CVPR.2015.7298965.

[20] M. Maguire, S. Dorafshan, and R.J. Thomas, *SDNET2018: A concrete crack image dataset for machine learning applications*, (2018). https://digitalcommons.usu.edu/all_datasets/48/.

[21] A. Miyamoto, M-A. Konno, and E. Bruhwiler, *Automatic crack recognition system for concrete structures using image processing approach*, in Asian Journal of Information Technology, 2007. https://medwelljournals.com/abstract/?doi=ajit.2007.553.561.

[22] M. Mundt, S. Majumder, S. Murali, P. Panetsos, and V. Ramesh, *Meta-learning convolutional neural architectures for multi-target concrete defect classification with the concrete defect bridge image dataset*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 11196–11205. https://doi.org/10.1109/CVPR.2019.01145.

[23] H. Oh, N.W. Garrick, and L.E.K. Achenie, *Segmentation algorithm using iterative clipping for processing noisy pavement images*, in Imaging Technologies: Techniques and Applications in Civil Engineering. Second International Conference Engineering Foundation; and Imaging Technologies Committee of the Technical Council on Computer Practices, American Society of Civil Engineers, 1998.

[24] H. Oliveira and P.L. Correia, *CrackIT–An image processing toolbox for crack detection and characterization*, in Proceedings of IEEE international Conference on Image Processing (ICIP), IEEE, 2014, pp. 798–802. https://doi.org/10.1109/ICIP.2014.7025160.

[25] Ç.F. Özgenel and A.G. Sorguç, *Performance comparison of pretrained convolutional neural networks on crack detection in buildings*, in Proceedings of the International Symposium on Automation and Robotics in Construction (ISARC), vol. 35, IAARC Publications, 2018, pp. 1–8. https://doi.org/10.22260/ISARC2018/0094.

[26] M. Poncelet, G. Barbier, B. Raka, S. Courtin, R. Desmorat, J.C. Le-Roux, and L. Vincent, *Biaxial High Cycle Fatigue of a type 304L stainless steel: Cyclic strains and crack initiation detection by digital image correlation*, European Journal of Mechanics-A/Solids, 29 (2010), pp. 810–825. https://doi.org/10.1016/j.euromechsol.2010.05.002.

[27] P. Prasanna, K.J. Dana, N. Gucunski, B.B. Basily, H.M. La, R.S. Lim, and H. Parvardeh, *Automated crack detection on concrete bridges*, IEEE Transactions on Automation Science and Engineering, 13 (2014), pp. 591–599. https://doi.org/10.1109/TASE.2014.2354314.

[28] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in Proceedings of International Conference on Medical image Computing and Computer-Assisted Intervention, Springer, 2015, pp. 234–241. https://doi.org/10.1007/978-3-319-24574-4_28.

[29] S.J. Schmugge, L. Rice, N.R. Nguyen, J. Lindberg, R. Grizzi, C. Joffe, and M.C. Shin, *Detection of cracks in nuclear power plant using spatial-temporal grouping of local patches*, in Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2016, pp. 1–7. https://doi.org/10.1109/WACV.2016.7477601.

[30] Y. Shi, L. Cui, Z. Qi, F. Meng, and Z. Chen, *Automatic road crack detection using random structured forests*, IEEE Transactions on Intelligent Transportation Systems, 17 (2016), pp. 3434–3445. https://doi.org/10.1109/TITS.2016.2552248.

[31] T. Yamaguchi, S. Nakamura, and S. Hashimoto, *An efficient crack detection method using percolation-based image processing*, in Proceedings of IEEE Conference on Industrial Electronics and Applications (ICIEA), IEEE, 2008, pp. 1875–1880. https://doi.org/10.1109/ICIEA.2008.4582845.

[32] F. Yang, L. Zhang, S. Yu, D. Prokhorov, X. Mei, and H. Ling, *Feature pyramid and hierarchical boosting network for pavement crack detection*, IEEE Transactions on Intelligent Transportation Systems, (2019). https://doi.org/10.1109/TITS.2019.2910595.

[33] C.M. Yeum and S.J. Dyke, *Vision-based automated crack detection for bridge inspection*, Computer-Aided Civil and Infrastructure Engineering, 30 (2015), pp. 759–770. https://doi.org/10.1111/mice.12141.

[34] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, *Pyramid scene parsing network*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2881–2890. https://doi.org/10.1109/CVPR.2017.660.

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