Pixel-accurate road crack detection in presence of inaccurate annotations

Rodrigo Rill-García, Eva Dokládalová, Petr Dokládal

Pixel-accurate road crack detection in presence of inaccurate annotations

Rodrigo Rill-García[a,b,*], Eva Dokladalova[a], Petr Dokládal[c]

[a] *LIGM, Univ Gustave Eiffel, CNRS, ESIEE Paris, F-77454 Marne-la-Vallée*
[b] *Navier Laboratory, École des Ponts ParisTech, Univ Gustave Eiffel, CNRS, F-77454 Marne-La-Vallée, France*
[c] *Center for Mathematical Morphology, MINES Paris – PSL Research University, 35 rue Saint Honoré, Fontainebleau 77305, France*

Abstract

Recent road crack detection methods obtain appealing scores but typically allow a few pixel tolerance margin. This is acceptable for locating cracks, but not for measuring their width (indicator of the cracks' severity). Our baseline model, U-VGG19, obtains an F-score of 71.77% on CrackForest, which is superior to other approaches when no tolerance is admitted. However, increasing the scores without tolerance is difficult due to inaccurate annotations.

We propose a novel synthetic dataset, Syncrack, as a benchmark for the evaluation of training with inaccurate annotations. Our results show that inaccurate annotations have a detrimental impact on the F-measure, decreasing it by up to 20%. To overcome this, we study label noise correction techniques using weakly supervised learning. Training U-VGG19 with these corrected labels improves the results on Syncrack by up to 12%. Obtained results on the CrackForest and Aigle-RN datasets support that these approaches are useful for real-life data too.

Keywords: Crack Detection, Deep Learning, Defect Segmentation, Inaccurate Labels, Weakly Supervised Learning

1. Introduction

When monitoring road condition, it is crucial to inspect the cracks [1], one of the many roads' distress signs. By counting the cracks and determining their orientation, it is possible to determine the origin and mechanism of the damage (road aging, excessive traffic, climate, etc.). In addition to that, there is an interest to measure the cracks width, which is one of the indicators of the severity of the damage and future durability of many constructions, such as concrete structures [2]. Measuring the width requires a pixel-accurate crack detection, which is still a challenging task [3].

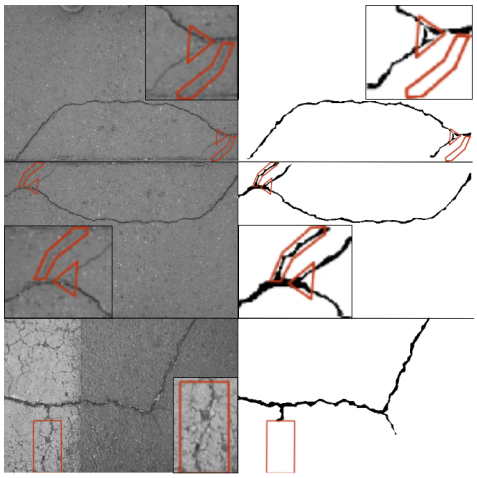
The major difficulties in road inspection are: 1) the boundary between crack and background is often fuzzy; 2) the background is composed of diverse textures; and 3) the cracks have complex geometric shapes. The problem becomes harder under uncontrolled acquisition conditions: irregular and variable illumination; the image resolution on the limit of thin cracks width; intrusive background textures due to external surface conditions (humidity, dirt, sand, oil spots) and objects (lane signs, manhole covers, leaves); etc.

All these constraints cause errors when it comes to manual annotation, which is a tedious and highly time-consuming task. There are annotation errors on two levels (see Fig. 1): a) on the level of objects (a missing label for a crack, or a label for a non-existent crack), and b) on the level of pixels (labels are either too wide, too thin or inaccurately placed).

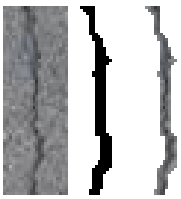
Because of inaccurate labeling, particularly on the level of pixels, searching for a perfect match of the detected cracks and the annotation is pointless. This motivated fundamental works such as CrackTree [4] to propose tolerance margins for evaluation: "a detected crack pixel is still considered to be a true positive if it is located no more than 2 pixels away from human annotated crack curves". Since then, many authors have opted to use this kind of tolerance margins for evaluation [4–13]. Nowadays, state-of-the-art methods using these tolerance margins exhibit the highest scores – overwhelming F-scores as high as 95% using Deep Learning [5, 14]. The works cited in this paragraph are discussed in detail in section 2. While this tolerant approach is sufficient for counting and locating cracks, it is not for measuring their width – an indicator of the cracks' age and severity [3].

Tolerance margins are lax and tolerant to errors on the pixel level, providing too optimistic scores. These margins have shown to create huge score gaps: from F-score=56.7% (0-pixels tolerance) to 80.0% (1-pixel tolerance). The typical 2-pixels margin increased the score even more: up to 87.0% [8]. Table 1 shows preliminary results with different tolerance margins, training the model proposed in this paper on public data. We can observe that increasing the tolerance margin increases the precision drastically. It has been shown that training with dilated annotations allows increasing the recall when evaluating on raw annotations. This increase in recall, however, comes with a decrease of precision [15] (because the predictions are wider than the original annotations). Thus, tolerance allows predicted cracks to be wider than reality, artificially preserving high

*Corresponding author: E-mail address: rodrigo.rill@esiee.fr



(a)



(b)

Figure 1: Examples of inaccurate annotations (from the CrackForest dataset). (a) Incongruent or fuzzy manual annotations. (b) A zoom to an example shows evidence of lack of accuracy (here, the manual crack mask is wider than the crack).

precision scores.

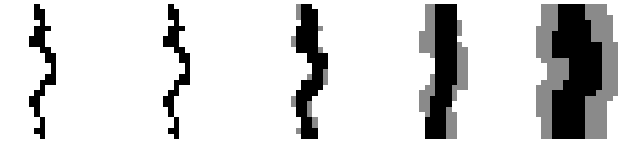
Evaluating without tolerance will provide a higher degree of confidence in the geometric properties obtained from detected cracks e.g. shape, length and width (see Fig. 2). However, increasing the scores without tolerance is difficult due to the inaccuracy of manual annotations. From a machine learning perspective, these inaccurate labels are a particular case of noisy data at the pixel level. Although some efforts have been made to learn in the presence of noise with some class imbalance [16], no strategy is usable on heavily class-imbalanced data. Indeed, cracks typically represent less than 1% of the pixels from road images. Such a rate of imbalance represents a considerable challenge to effectively train a model with inaccurate human labels.

In this paper, we explore the limitations of pixel-accurate crack segmentation methods based on fully convolutional neural networks (FCNN). Particularly, the limitations caused by the negative impact of inaccurate labels. Our contributions can be summarized as:

1. An extension of VGG19. We propose to use – as encoder – the convolutional layers from VGG19 i.e. the 19-layers version of the network originally proposed by the Visual Geometry Group at Oxford. Inspired by U-net, we build a novel symmetrical, u-shaped encoder-decoder network (referred to as U-VGG19) for supervised road pavement crack detection. We use transfer learning to reduce training costs and deal with few real-life annotated data.
2. A novel synthetic dataset, Syncrack. We develop this dataset and propose it to quantitatively measure the detrimental impact of inaccurate labels when having an accurate ground truth for evaluation. We are the first to provide this kind of analysis for road crack detection. From this we prove that training with inaccurate labels deteriorates the accuracy of the width of detected cracks.
3. A study of label correction methods. Previous works on road crack detection approach the problem of inaccurate annotations using tolerance margins for evaluation. Unlike them, we prove that correction approaches traditionally used in *weakly supervised* classification provide a significant improvement with respect to inaccurate annotations. Using these methods, we increase scores up to 12% in pixel-accurate evaluation (when no tolerance margins are allowed).

Table 1: Crack detection scores using different tolerance margins.

Metric	Tolerance margin			
	0px	1px	2px	5px
Precision	72.04	88.19	93.10	95.15
Recall	74.45	78.10	79.02	79.37
F-measure	73.23	82.84	85.48	86.55



(a) Crack (b) 0px (c) 1px (d) 2px (e) 5px

Figure 2: Evaluation with tolerance margins. Color code: (Black/White) Crack/No crack; (Grey) Ground truth for evaluation with tolerance margins. (a) A crack. (b) Perfect prediction. (c-e) Predictions inside their corresponding tolerance margin; all black pixels are true positives, but the geometry of the real crack is not respected.

Our quantitative results on Syncrack show how pixel-accurate crack segmentation can greatly benefit from weakly supervised learning approaches. This is mainly reflected in an increased precision score, which is related to more accurate crack width prediction. Furthermore, our results on public datasets support that these approaches are useful for real-life data too.

The rest of the paper is organized as follows. In section 2, we provide a review of the literature on crack detection, and on learning from inaccurate labels. The model proposed as baseline (U-VGG19), and the explored methods to correct labels, are introduced in section 3. The

Table 2: Summary of methods proposed before Deep Learning.

Method	Approach	Main weakness
Minimal Path Selection [6]	Image processing and graph representation	Non-robust to intrusive objects and overall noisy images
CrackTree [4]	Image processing, graph representation and tree generation	
Perception law [17]	Image processing, probability	
CrackIt [18]	Block intensity statistics, supervised machine learning	Dependent on handcrafted features for supervised training
CrackForest [7]	Random structured forests, SVM	

experiment setup used for training the baseline model, and label correction, is described in section 4. This section also brings a description of the proposed synthetic dataset, Syncrack. Then, we analyse the potential of using U-VGG19 as a baseline model in section 5, and we compare it with the state of the art on public datasets, with pixel-accurate evaluation. In section 6, we quantify the impact of inaccurate labels and the efficiency of different label correction methods on Syncrack and real-life data. The section 7 concludes the paper and suggests some future improvements.

2. Related Work

Automatic crack detection has been a topic of interest for many years. Table 2 summarizes some representative approaches based on traditional image processing and machine learning. However, this kind of approaches has been greatly surpassed by Deep Learning (DL).

2.1. Crack Detection Using Deep Learning

In 2016 and 2017, [19] and [20] respectively cropped high-resolution pavement images into patches, annotated binary as containing or not a crack, to train a CNN. Both networks beat conventional computer vision approaches. Later works such as [21] aimed for multi-class classification. This allowed not only identifying patches with cracks but identifying the type of crack inside the patch. In 2018, [20] fine-tuned AlexNet for concrete structures. To achieve more precise detections, they get crack-probability maps based on the average predicted probability of overlapping sliding windows. Similarly, [9] proposed a structured prediction approach by representing each pixel with a patch. Per input patch, a smaller patch was predicted and the output patches were used to build probability maps. For simultaneous distress segmentation, [22] used YOLOv2; this provided a set of bounding boxes, avoiding patches but lacking accurate location.

In 2019, [5] moved to FCNNs: U-net based networks allowed pixel-accurate segmentation. The same year, U-net was also introduced for concrete crack detection [23]. Independent U-net variants were used as generative models by [24] for generative adversarial networks (GANs). In 2020, [25] again combined a U-net with adversarial learning, replacing the last block of convolutional layers in the decoder by a discriminator. This network aimed to classify patches in a binary way.

A U-net variant inspired by PSPNet was proposed by [26]. This architecture replaces the blocks of convolutional layers in the encoder by multi-scale blocks, and the bottleneck by residual blocks. A similar network was proposed by [11], but adding attention gates between the encoder and the decoder before concatenating to only propagate relevant activations further. In [15], a self-attention layer is added on top of the last bottle-neck layer of a U-net. This network is one of the method's three components: image preprocessing, deep neural network and data augmentation (dilating the annotations used for training).

With a multi-scale approach too, [27] used an encoding FCNN to obtain different resolution feature maps. These maps are deconvoluted to calculate the loss during training and to fuse them together. The final fusion is post-processed to obtain refined predictions. A boosting method based on a U-net architecture was proposed by [28], assembling feature maps with different resolutions from the decoder, sharing a single fusion-loss function. The presence of a decoder avoids the need for post-processing. A similar approach was presented by [10], but using SegNet as a basis. This provided SegNet with the ability to decode using information from the encoder, similarly to U-net's skip connections. The architecture discussed in [28] was extended by [12] adding a multi-dilation module at the bottleneck. This module allows to obtain multiple context sizes' features by using dilation convolutions with different dilation rates. Another ensemble approach was proposed in [14], weighting multiple independent CNNs for the final ensemble. These CNNs return $(0, 1)^{n \times n}$ vectors from n×n patches.

Table 3 compares the top scores from DL methods presented in this section. As discussed before, scores based on tolerance margins are lax and tolerant to errors on the pixel level.

2.2. Weakly Supervised Learning

Weakly supervised learning is an intensively studied topic [30, 31]. It can be categorized into three main groups: noise-robust algorithm design, noise filtering, and noise-tolerant methods [30]. We will focus on the first two.

Robust algorithms like SVM or k-NN can reduce the negative impact of noisy labels, as long as the number of training samples is big enough [16]. Furthermore, combining SVM with techniques such as active learning provides a

Table 3: Comparison of deep-learning-based results using different scope evaluations on pavement images.

Evaluation strategy	Bounding box [22]	Patch [25, 29]	Tolerance margin [5, 14]	Pixel-accurate [24, 26, 28]
F-measure	>85 %	>90 %	>95 %	>70 %
Geometric accuracy (location/width)	★	★★	★★★	★★★★
Evaluation precision	Crack-box size dependent	Patch size dependent	Tolerance margin dependent	Pixel-accurate

way to train with semi-supervision. This reduces the labeling process to annotate a set of -greedily chosen- relevant samples [32]. However, traditional machine learning algorithms using handcrafted features have been overpassed by DL for crack detection.

Classifier ensembles (e.g. voting strategies [33, 34]) have shown an ability to improve the performance of models trained on inaccurate data. These ensemble outputs can be used further to post-process the predictions. On the other hand, single-classifier-based filters have been used successfully too [33, 35]. Using single-classifier outputs recursively is called self-training: each new model takes the output of the previous one as input, and it produces new (cleaner) labels. This strategy has been used to segment images using image-level labels [36].

Other approaches have been proposed to study the influence of single data pairs (input/label) during training to identify outliers (i.e. potential mislabels) [37, 38]. Nonetheless, influence functions require expensive second derivative calculations and assume model differentiability and convexity. Recent efforts [39, 40] have proposed to approximate these functions using second-order optimization techniques. This allows integrating influence scores for outlier detection when training CNNs. However, the detection of influential observations is complex. In addition, Lagrangian [41] and Sobolev gradient based optimizers [42] have been applied to solve different image segmentation problems with high performance. However, they generally cause high computational costs too.

Alternatively, efforts have been made towards building loss functions able to deal with outliers. For example, [43] proposes a framework with a noise-robust loss that allows updating the parameters of the network and correcting the inaccurate labels simultaneously. However, this kind of approaches, as well as influence-based ones, is typically used for image classification. For segmentation, we have a more complex problem. Instead of classifying images, we are classifying pixels: we make tens of thousands of predictions per image instead of one.

Some alternatives to this kind of problem have been proposed based on per-pixel difficulty. For example, for vessel segmentation from weak automatic annotations [44]. First, the loss focuses on easy-to-classify pixels. However, this has the potential risk of ignoring systematic biases in noisy labels (losing crucial pixels). Thus, an online active component is used to refine updated labels: a small number of valuable pixels with potentially incorrect labels are annotated and then manually refined in each iteration during training.

For vessel segmentation, choosing these valuable pixels is challenging because of the highly imbalanced foreground and background. Similarly, crack detection has an inherent severe class imbalance. Training in presence of class imbalance is difficult by itself; it is commonly done by oversampling the minority class. However, doing this in presence presence of label noise is still a challenging problem [45]. Particularly, severe class imbalance represents an open challenge also for all the weakly supervised learning approaches discussed so far.

After identifying the mislabeled samples, filtering is done by correcting those samples. This correction consists of relabeling or removing the identified mislabeled examples [31].

3. Proposed Approach

3.1. U-VGG19

U-net [46], a symmetrical auto-encoder, has seen success in pavement distress segmentation. Unlike a typical auto-encoder, it adds some skip connections to feed the decoder with the feature maps generated by the encoder at different resolutions.

On the other side, VGG has been an attractive feature extractor, also used to detect cracks. For example, [5] used the pre-trained VGG16 by substituting its dense layers with deconvolutional ones. For black box images, [47] built a generic decoder able to be concatenated with different encoders. Using a pre-trained VGG16 encoder allowed the model to surpass even a model based on ResNet152 (without using pre-trained ResNet weights). Similarly, [13] proposed an improved U-net by testing different depth encoders inspired by the VGG19 architecture. Furthermore, loss functions based on VGG-extracted feature maps have been used for texture generation. Particularly, to reduce the impact of limited amounts of available annotated data, [48] used a semantic texture generation approach for data augmentation. This improved the model's performance in real images.

Based on these reasons, we built a U-net-like network using the pre-trained VGG19 [49] as a basis. We use its

convolutional layers as encoder (Fig. 3.A), and we replace the dense layers with a decoder (symmetrical to the encoder, Fig. 3.B) for segmentation. Like the original U-net, we add a skip connection between the encoder and the decoder at each available resolution (see Fig. 3.C).

This network, referred to as U-VGG19, is used as baseline model for the experiments in this paper. All the convolutional layers use kernel size 3, stride 1, and a ReLu activation. Activation functions have a critical role in deep neural networks [50]. Even though different types of functions have been used in recent works [51], we employed ReLU in our work because of its efficiency and results. The only exception is the final layer (Fig. 3.D), using kernel size 1 with a sigmoid activation to obtain a single channel image with values in the range (0, 1) – crack probability. The downsampling uses max pooling; the upsampling uses nearest-neighbor interpolation. Since we do not use any dense layer, any input size can be used without resizing. The number of trainable parameters is 39,236,101.

3.2. Learning in Presence of Noise

Manual annotations are prone to many errors, especially at the pixel level. Training with these inaccurate annotations is a case of inaccurate supervision, a sub-case of *weakly supervised learning* [31]. In this paper, we explore noise-robust algorithms and noise filtering strategies for label correction [30]. Particularly, we look to improve crack prediction in terms of the cracks' real width. The core idea is 1) to get a set of new pseudo-labels from the original data and 2) to train a model with the help of these pseudo-labels.

From robust algorithms, k-NN has shown to reduce the negative impact of noisy labels, as long as the number of samples is big enough [16]. In this paper, we use U-VGG19 as a feature extractor: we use the feature maps from the second to last convolutional layer (Fig. 3.D) to represent each pixel as a 2D vector. Then, a k-NN algorithm with k=5 is used to assign new pseudo-labels per pixel. The algorithm is applied to each image individually, because of its scalability with respect to the number of pixels. A typical image for pavement crack detection contains $\approx$150k pixels, which fulfill the "big enough" requirement [16]. Increasing the number of images to be used simultaneously would increase drastically the processing time per image. We refer to this approach as *5-nn voting*.

It is also possible to use the predictions of the model for self-training. This means to train models recursively: each new model takes the output of the previous one as training input, and the new model predicts new pseudo-labels [39]. In this paper, we train U-VGG19 with manual labels and we use the trained model to predict new pseudo-labels per image. We refer to this approach as *self-training*.

Ensemble voting strategies have shown to improve the performance of models trained with noisy data too [33, 34]. Similarly to [52], we use a bagging strategy with a k-folds approach. Specifically, we train 10 different models with 10 different subsets (composed of 9 folds each). Once the 10 voters are trained, they are used to predict all the available images. Unlike [52], we get pseudo-labels per image using all the trained models simultaneously before training the final model. To do this, two ensemble strategies are used: *majority voting* and *consensus voting*.

The computational complexity of the four methods depends on the baseline model used. In this paper, our baseline model is U-VGG19, but any segmentation network could be used. We assume its complexity constant per pixel, and linear with the size of the image, denoted by $\mathcal{O}(n)$ with n being the number of pixels. The four methods can be repeated more than once, but there is no guarantee of improvement by increasing the number of iterations. Here we discuss the computational complexity of performing 1 iteration per method.

In the case of self-training, the method requires one full training of the baseline model and one prediction of the full dataset; no further operations are needed. For the voting methods, M instances of the baseline model (considered as weak learners) are each trained using $\frac{M-1}{M}$ of the dataset. With the predictions of the weak learners, the final pseudo-labels are obtained through voting. Each voting strategy can be solved in $\mathcal{O}$(M) per pixel [53]. Therefore, the complexity of voting is $\mathcal{O}$(Mn) per image. In the case of 5-nn voting, the final pseudo-labels are obtained using a k-NN algorithm on a d-dimensional space as projected by the baseline model. The projection is done in $\mathcal{O}$(n). Then, when using an efficient tree structure for the k-NN algorithm [54, 55] (available in [56]), the complexity of the tree construction is $\mathcal{O}$(dn log(n)). The k-NN prediction per pixel is performed in $\mathcal{O}$(k log(n)). Finally, the pseudo-labels for all the pixels are obtained in $\mathcal{O}$(kn log(n)) per image.

The pseudo-labels generated by the methods described before (see summary in Table 4) are used to filter (i.e. to correct) the original labels. This consists of removing or relabeling samples [31]. Evidence suggests that removing the identified mislabeled samples reduces the error in clean data with respect to relabeling them. Nevertheless, compared with removal methods, relabeling ones have their accuracy fall off much more slowly when increasing the label noise [35]. This may be explained by the limited number of remaining training samples after removal. In our context, with few labeled images and a very low rate of positive-class instances, the impact of removing data points could be worse. Thus we study both approaches.

To relabel, we simply replace the original labels by the new pseudo-labels. To remove, we use the pseudo-labels to weight pixels at loss calculation during training: we assign a weight of 1 to the pixels where both the raw and the pseudo-label agree, and 0 to all others.

4. Experimental Setup

4.1. Fully-convolutional Neural Network

The proposed architecture was implemented using Tensorflow 2.1.0 and a 4GB Nvidia GTX 1050. During train-

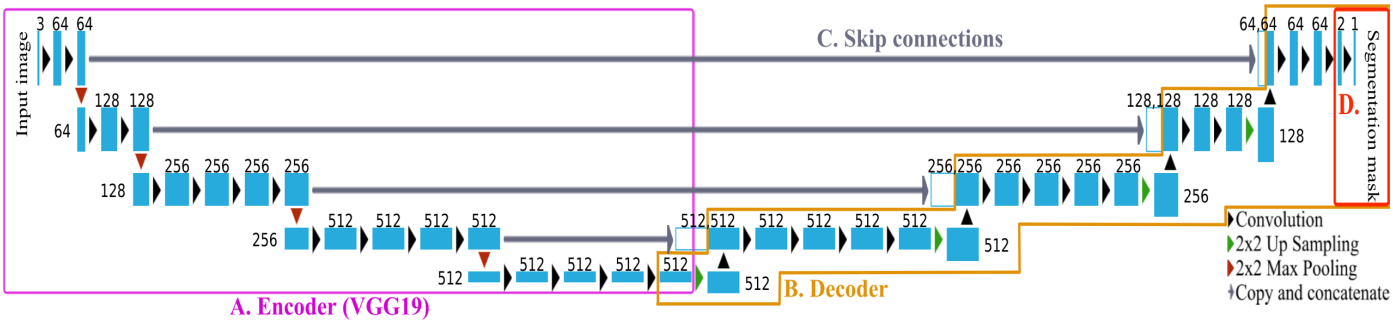


Figure 3: U-VGG19 architecture.

Table 4: Comprehensive comparison of the proposed pseudo-label generation methods. The number of pixels per image is denoted by n.

Method	Basis	Advantages	Disadvantages	Computational complexity
Self-training	Single-model prediction	Easy to implement, no hyperparameter selection	Amplification of the baseline model's bias	Cost of the baseline model – $\mathcal{O}$(n)
Majority voting	Ensemble of M classifiers using bagging	A balanced probability of discarding good data and retaining bad data [33]	Empirical selection of M	$\mathcal{O}$(Mn)
Consensus voting	Ensemble of M classifiers using bagging	Conservative in discarding good data [33]	Empirical selection of M	$\mathcal{O}$(Mn)
5-nn voting	k-NN algorithm on projected d-dimensional space	Robust to decision boundary overfitting	Empirical selection of k	- Pixel projection $\mathcal{O}$(n) - Tree creation $\mathcal{O}$(dn log(n)) - Pseudo-label generation $\mathcal{O}$(kn log(n))

ing, input images are cropped to 256×256 patches and fed as 4-patch batches. This setup is used to deal with datasets containing multiple-size images without resizing. Each model is trained using the Adam optimizer with a 10^{-4} learning rate and default parameters. The initial weights from the encoder are the weights from VGG19 pre-trained on ImageNet; from this starting point, the whole U-VGG19 is trained together.

For each dataset, available images are randomly split into 80% training and 20% validation using a fixed seed. Then, a model per dataset is trained. To refine the results at late epochs, we reduce the learning rate on validation loss plateau (by 2, with 5 epochs tolerance). To avoid overfitting, we add an early stop if the validation loss does not increase during 20 consecutive epochs. We report the scores calculated on the the validation split at the epoch with the minimum validation loss (validation images are fed without cropping using batch size 1).

4.2. Datasets

CrackForest Dataset (CFD): This public dataset contains 118 images collected from urban roads containing perturbations such as shadows, oil spots, and water stains in Beijing, China [7]. The original image size is 480×320. Two independent annotations are provided: borders and segmentation. As suggested by [26], we removed some images with clear annotation errors; we preserved a total of 108 images.

Aigle-RN: A subset from a bigger database presented in [6]. Unlike the other subsets (collected by laser), Aigle-RN is captured using cameras. It contains 38 annotated images collected at traffic speed for periodically monitoring the French pavement surface condition. They have been pre-processed to mitigate the influence of non-uniform lighting conditions.

Syncrack: This dataset (available at: `https://github.com/Sutadasuto/syncrack_generator/tree/demo`) provides synthetic RGB images of crack-like structures over pavement-like textures. Both the crack shapes and the pavement textures are generated randomly based on Perlin noise. To make crack detection challenging, the contrast of cracks is low and randomly variable along the crack. Furthermore, transition zones between cracks and background are added to emulate the boundary fuzziness of real-life pictures. Finally, small crack-like artifacts are added to avoid models to learn only from pixel intensities ignoring the spatial properties of actual cracks. To make this dataset more similar to a real-life one, the image size is 480×320 and the crack width is randomly chosen to be around 1-3 pixels (similarly to CFD). The dataset contains

500 images with their corresponding pixel-accurate annotations. Additionally, a noisy version (emulating real-life inaccurate annotations) is provided.

4.3. Label correction

For the generation of pseudo-labels, we first train U-VGG19 under the same setup described at subsection 4.1. The model obtained with the training split of a given dataset is used to get pseudo-labels for both the training and the validation splits of the very same dataset. The newly labeled training and validation splits are then used to train a new model. Finally, this model is used to predict the images from the validation split using the original raw labels for evaluation.

In the case of Syncrack, the training split is 20% and 80% for validation. This is meant 1) to have a number of training images similar to real-life datasets and 2) to have a larger number of validation images such that the confidence in the obtained results is higher. To have objective baselines about the effects of inaccurate annotations on crack detection, we use the noisy version of Syncrack.

To introduce the noise, the ground truth images were divided into patches. Per patch, an erosion or a dilation is randomly performed, using a disk with random radius. With this approach, some crack annotation segments become wider, others thinner, some remain untouched and some even disappear completely. Syncrack contains 0.46% crack pixels and 99.54% background pixels. After introducing noise, with "crack" as the positive class, the labels are composed of: 0.31% true positives, 0.25% false positives, 99.30% true negatives, and 0.14% false negatives. Thus, the dataset has a quasi symmetrical label noise (false positives and false negatives), and a severely imbalanced class representation, likewise in real road images.

4.4. Training Loss

In segmentation tasks, the binary cross-entropy (BCE) loss is typically used for training. However, this function exhibits a severe problem on highly imbalanced data like ours. As pointed out by [25], a naive FCNN approach will lead to the "all black" problem: the network will simply converge to treating the entire input image as background.

To solve this, classical approaches like class weighting have been used [28]. However, by overweighting the under-represented class (cracks), the model will have a bias towards false positives such as wider cracks and isolated noise. This behavior is contradictory with the pixel-accurate segmentation goal, so we adopted the approach proposed by [26]: a hybrid loss function using the Dice Score Coefficient (DSC). This score, used for segmentation evaluation, represents the ratio of the area of intersection of two objects to the total area.

DSC ranges from 0 to 1 (the greater, the better), but it has a singularity when both ground truth and prediction have no crack pixels. To deal with this, and to use the score as a loss function to minimize, the Dice loss (DICE) is defined as follows:

$$DICE = 1 - \frac{2 * GT * Pred + 1}{GT + Pred + 1} \tag{1}$$

DICE ranges from 0 (perfect prediction) to 1 (Pred and GT don't intersect at all). However, this function has convergence problems, sometimes falling into local optimum. To deal with this, our final loss introduces BCE as proposed by [26]:

$$Loss = BCE + \alpha * DICE \tag{2}$$

The constant weight α is a hyperparameter; after some preliminary experiments, we set α to 3. By minimizing this function, BCE helps to achieve convergence while DICE punishes the model for "all black" outputs.

4.5. Evaluation Scores

The first approaches to crack detection using DL performed patch classification. Given the inherent class imbalance, Precision, Recall and F-measure were often used as metric scores. These scores were extended to evaluations with tolerance margins, as well as to recent pixel-accurate works evaluating without tolerance, so we do as well. Similarly to [26], we report the DSC as a metric score more suited for binary segmentation.

We calculate the DSC per image, and we report the average over all the validation images. Precision and recall are calculated over all the validation pixels. To calculate the scores, the network's predictions are binarized using a threshold of 0.5.

5. Baseline Model Analysis and Comparison

To explore our label correction approaches, first it was necessary to test our baseline model. To do this, we trained U-VGG19 on public real-life datasets and compared our scores with state-of-the-art methods (when no tolerance is allowed).

Fig. 4 shows an example of U-VGG19 predictions in validation images from both CFD and Aigle-RN. Table 5 shows the scores obtained on the validation sets of both datasets, when training U-VGG19 on each one of them individually. Additionally, to analyze the ability of U-VGG19 to learn simultaneously from different information sources (cross-dataset generalization), we report the results obtained by using CFD and Aigle-RN as a single dataset.

It is important to highlight that the highest score is obtained in the CFD+Aigle-RN dataset. This suggests that our network benefits from the diversity of the images contained in both datasets. This boost can be also explained by the increased number of training images. The number of training images could also explain why the performance of U-VGG19 is lower in Aigle-RN compared to CFD (38 vs 108 labeled images).

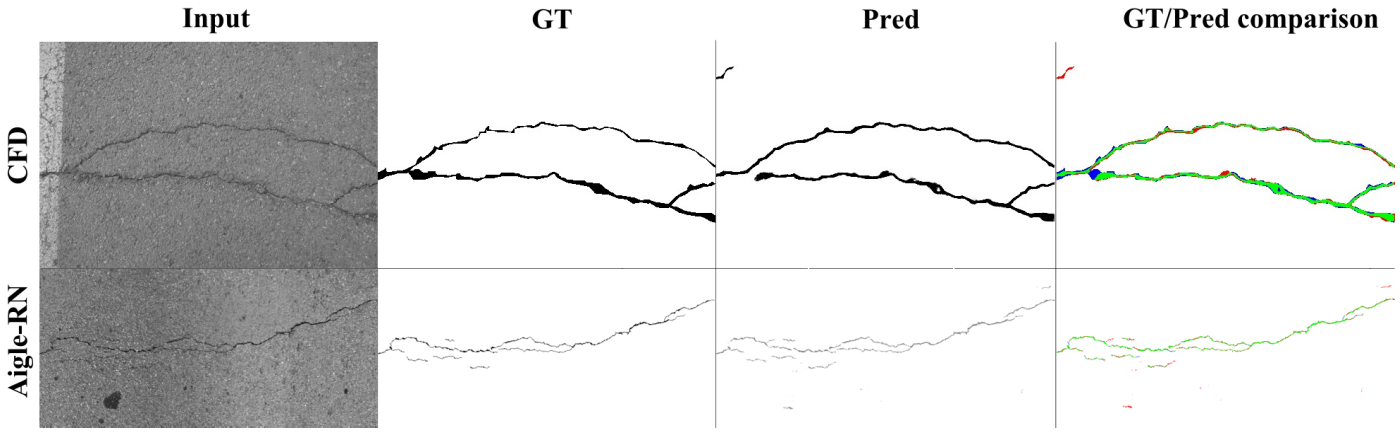


Figure 4: Qualitative performance on CFD and Aigle-RN. Comparison color code: (Green) True positives; (Blue) False negatives; (Red) False positives.

Table 5: Results on public pixel-accurate datasets using U-VGG19.

Dataset	Metrics	Score
CFD	Pr, Re (F)	72.23%, 71.31% (71.77%)
	DSC	70.80%
Aigle-RN	Pr, Re (F)	56.33%, 81.25% (66.53%)
	DSC	65.66%
CFD+Aigle-RN	Pr, Re (F)	72.04%, 74.45% (73.23%)
	DSC	72.01%

Table 6: Comparison of pixel-accurate F-scores on CFD.

Method	F-score
U-net [24]	60.48%
GANs [24]	64.13%
Multi-scale Convolutional Blocks [26][a]	74.19%
Feature Pyramid Hierarchical Boosting [28][b]	70.50%
Distribution equalization learning [15]	54.55%
– U-VGG19 (ours) –	71.77%

[a] Training and evaluation are done with a CFD+Aigle-RN dataset.
[b] GT and Pred are thinned to 1-pixel edges for evaluation.

Table 6 compares U-VGG19 with other pixel-accurate methods evaluated without tolerance. These state-of-the-art methods, similarly to ours, are based on U-net. We chose CrackForest as a reference dataset and F-measure as the score since they are the most popular in literature.

Our method is just below a U-net with multi-scale convolutional blocks [26]. However, those results were obtained by using a CFD+Aigle-RN dataset. By comparing them to our results using the same dataset fusion approach, we are below by less than 1% – see Table 5 (CFD+Aigle-RN F-score) and Table 6 (Multi-scale Convolutional Blocks). Nonetheless, their architecture is much more complex than ours; moreover, our training converged around the same number of epochs reported by them (∼90), implying an advantage in terms of hardware requirements and time. Furthermore, the transfer learning strategy of U-VGG19 outperforms other, more complex, approaches (GANs [24], multi-scale hierarchical boosting [28], distribution equalization learning [15]).

When we analyzed U-VGG19 predictions qualitatively, we observed two relevant types of error: 1) missing low-contrast thin cracks and 2) questionable cracks that could be or not annotated as cracks depending on the observer.

As suggested by [26], we extended our method by introducing data augmentation. Particularly, we aimed to improve the recall by solving the problem of missing thin cracks. Our data augmentation consisted of randomly transforming an image (and its corresponding annotation) immediately before feeding it to the neural network for training. To do this, a random value for each of the 6 following operations is chosen: adding noise, changing illumination, flipping, zooming, rotating and shearing. Every image undergoes the 6 operations in the given order.

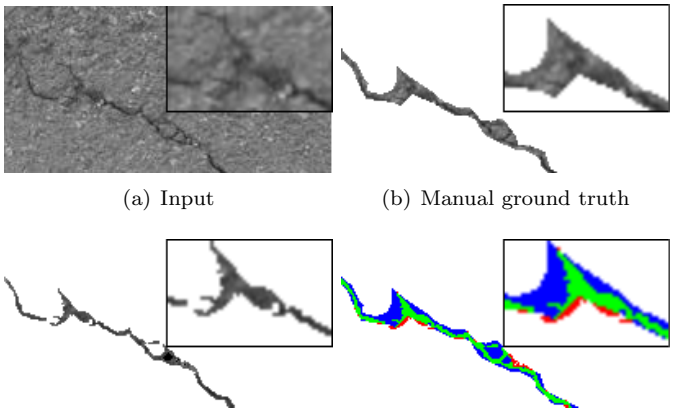
In Table 7, we can see that the data augmentation approach actually reduced our DSC. Even more, contrary to the expected outcome, the most affected score was the recall. When we analyzed the predictions of this new model, the score reduction seemed to be caused by questionable cracks.

Table 7: Results on CFD+Aigle-RN.

Method	Metric	Score
Multi-scale Convolutional Blocks [26]	Precision	72.44%
	Recall	76.02%
	DSC	72.09%
U-VGG19	Precision	72.04%
	Recall	74.45%
	DSC	72.01%
U-VGG19 with data augmentation	Precision	75.62%
	Recall	66.62%
	DSC	69.60%

These cracks are questionable both on the level of objects, and on the level of pixels: in Fig. 5, according to the manual annotation, the recall decreased; but, by looking

at the false negatives (in blue), it is clear that the predicted crack is closer to the real width and shape.



(a) Input (b) Manual ground truth

(c) U-VGG19 prediction (training with data augmentation) (d) Ground truth vs Prediction comparison

Figure 5: Comparison of manual (noisy) annotation and a prediction (more accurate in terms of width). The color code in (d) is the same as in Fig. 4.

To conclude, without accurate ground truth annotations, it is hard to objectively quantify the ability of the model to deal with noisy annotations. In the next section, we use Syncrack, a synthetic dataset with clean accurate annotations. By introducing noise to the labels, we quantify the effects of inaccurate annotations and label correction approaches.

6. Results and Analysis

In this section, we first quantify the detrimental impact of training with noisy annotations in the final prediction. To do this, we train U-VGG19 with the noisy version of Syncrack (inaccurate labels) and evaluate on the clean version (accurate labels). Afterwards, we train independent models using the pseudo-labels generated by each weakly supervised method (see summary in Table 4); we evaluate each of them using the clean version of Syncrack. This allows us to measure the improvement with respect to training with raw, inaccurate labels. Finally, from this analysis, we choose the best performing methods and test them on CFD.

To show that the scores obtained by training with pseudo-labels are stable, the results presented in this section are the average scores of 3 models trained independently per dataset. To begin, we trained U-VGG19 using Syncrack with clean (accurate) annotations as a baseline. Then, as a second baseline, we trained a fresh version of U-VGG19 with the noisy version of Syncrack and tested the model on clean data.

In Table 8, we can observe that training with noisy labels reduces drastically the performance of U-VGG19 on clean data: from DSC=82.58% down to 61.30%. To ensure that this behavior is not caused only by our architecture, we train two other publicly available networks on Syncrack: U-net-B, one of the approaches with the highest scores (F-score>95% [5]) in our literature review; and MultiResUnet [57], which has an architecture very similar to the one proposed by [26]. Both, U-Net-B and MultiResUnet, show again a severe drop of DSC when training with noisy data.

Table 8: Baselines on Syncrack using different networks.

Labels	Metric	U-VGG19 (ours)	U-Net-B [5]	MultiRes UNet [57]
Clean	DSC	82.58	75.36	63.23
	Pr	85.07	81.89	61.00
	Re	81.30	73.15	64.29
Noisy	DSC	61.30	57.11	51.23
	Pr	49.81	62.65	54.25
	Re	85.68	58.59	53.60
Noisy (with Data Augmentation)	DSC	66.47	53.48	60.63
	Pr	54.98	66.74	52.85
	Re	88.63	50.88	80.17

This implies that it is important to deal with inaccurate labels in order to take full advantage of the models' learning capacity.

The first tested approach was motivated by the observation made with respect to Fig. 5: it is possible that the expected generalization improvement caused by data augmentation is actually helpful to deal with inaccurate labels. Training U-VGG19 and MultiResUnet with noisy labels, but using data augmentation, actually improved the baseline of training with noisy data, as seen in Table 8. Since data augmentation is a common practice in DL, we use this score as a third baseline.

Given that U-VGG19 obtained the highest scores for all the baselines, we use it to test 4 label correction approaches:

1. *Self-training*: we use U-VGG19 trained with noisy labels (with data augmentation) to predict pseudo-labels.
2. *Majority voting*: we train 10 instances of U-VGG19 using subsets of the training split with noisy labels (without data augmentation). Pseudo-labels are generated by majority voting, using the 10 models as voters.
3. *Consensus voting*: we used the same 10 (previously trained) models, but generating the pseudo-labels with a consensus voting approach.
4. *5-nn voting*: we use U-VGG19[1] trained with noisy data (without data augmentation) as a feature extractor. Per image, we obtain pseudo-labels using a k-NN algorithm.

[1]Both for self-training and 5-nn voting, we chose the model with the highest validation score during training with noisy labels.

Table 9: Results on Syncrack's clean validation data training U-VGG19 with corrected labels. The best score per metric is highlighted in bold.

Correction method	Metric	Removing	Relabeling
Self-training	DSC	68.60	67.44
	Pr	58.38	55.90
	Re	87.05	**88.83**
Majority voting	DSC	63.69	63.41
	Pr	53.36	52.36
	Re	84.74	86.17
Consensus voting	DSC	64.34	64.24
	Pr	61.41	60.81
	Re	73.58	74.01
5-nn voting	DSC	71.57	**73.37**
	Pr	71.94	**72.53**
	Re	76.02	78.30

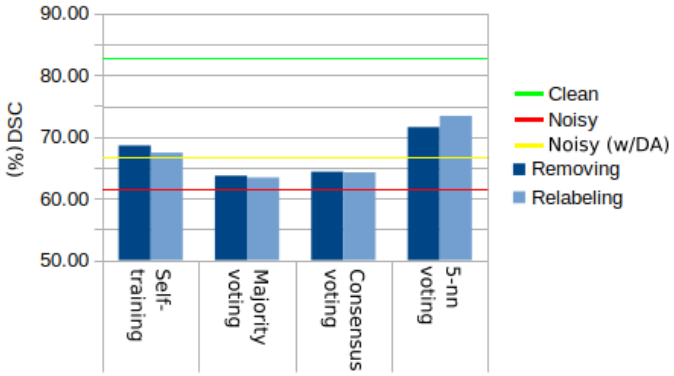


Figure 6: DSC on Syncrack's clean validation data using corrected labels.

The pseudo-labels generated by these methods are used to correct labels: relabeling the noisy annotations, and removing (ignoring) the pixels with a disagreement between noisy and pseudo labels. We show the results of both approaches in Table 9. In Fig. 6, it is easy to notice that both strategies have a very similar performance. Removing tends to be better most of the time, with 5-nn voting being the only exception.

This is particularly interesting since 5-nn voting is the best among the four proposed approaches. The DSC improved from 61.30% (training with noisy labels) to 73.37% (training with pseudo-labels): more than half the decrease caused by noisy labels. Comparing this with *training using noisy labels with data augmentation* (called *data augmentation* from now on), we see an increase from 66.47% to 73.37%.

The only other approach that surpassed the data augmentation baseline was self-training. This supports that, indeed, U-VGG19 is able to improve itself by training with its own predictions. Regarding the voting ensembles, both are below data augmentation but they surpass the noisy-label baseline. Particularly, consensus voting exhibits a slightly better score than majority voting (either by relabeling or removing).

In Tables 8 and 9, we highlight some behaviors:

1. Training with noisy labels may increase the recall but it decreases the precision with respect to training with clean data.
2. Using data augmentation tends to increase both precision and recall with respect to training with noisy labels.
3. Removing labels tends to increase the precision whereas relabeling the recall.
4. The behavior in 3 is analogous to the one expected from majority and consensus voting: consensus increases the precision and majority the recall.
5. All the explored methods improve the precision with respect to training with noisy data.

Behavior 1 suggests that introducing noisy labels during training makes the predicted cracks wider (precision decreases). On the other side, behavior 5 suggests that using weak supervision approaches deals with this problem (precision increases). Particularly, the method that obtained the highest DSC (5-nn voting, relabeling) achieved the highest precision.

Fig. 7 illustrates this. U-VGG19 is able to properly detect cracks as thin as 1 pixel, when trained with clean (accurate) labels. However, by training the same architecture with noisy (inaccurate) labels, the predicted crack is much wider than reality (3 times wider). After training the network with the pseudo-labels generated by 5-nn voting, most of the crack is detected with its original width.

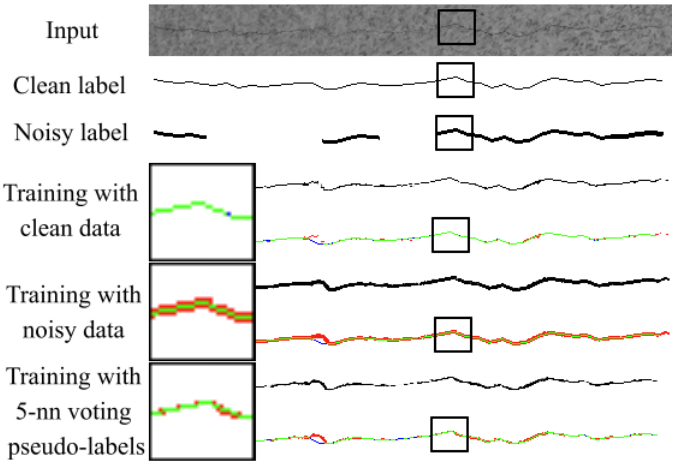


Figure 7: Prediction comparison of U-VGG19 on Syncrack. In the last three rows, black-lines show predictions and colored lines show the comparison with the clean label (the color code is the same as in Fig. 4).

The weak supervision approaches that provide the best scores on Syncrack (self-training and 5-nn voting) will now be tested on real data (CFD+Aigle-RN). For simplicity, we used only a relabeling strategy for training. This time, our only baselines are training with manual annotations and training with data augmentation. Table 10 shows the scores of the baselines and the two weak supervision approaches, using the original manual annotations as ground

Table 10: Results on CFD+Aigle-RN using label correction techniques.

Method	Metric	Score
U-VGG19	Precision	72.04%
	Recall	74.45%
	DSC	72.01%
U-VGG19 (using data augmentation)	Precision	75.62%
	Recall	66.62%
	DSC	69.60%
U-VGG19 (using self-training correction)	Precision	74.54%
	Recall	61.99%
	DSC	66.70%
U-VGG19 (using 5-nn correction)	Precision	74.23%
	Recall	71.73%
	DSC	71.53%

Calculated with respect to manual annotations.

truth for evaluation. Contrarily to the intuition, the DSC training with data augmentation was lower than when training without it. Furthermore, the DSC obtained by training with pseudo-labels is also lower than the raw manual labels baseline.

Nonetheless, under the context of noisy labels, this is not a surprise. As expected from the results obtained on Syncrack, the decrease of DSC is explained by a reduced recall in all the cases. Furthermore, in the three cases, the precision improves with respect to training with manual annotations. Manual annotations have a bias towards labeling cracks wider than reality. Learning this bias will improve the scores evaluating with those inaccurate annotations. A better crack width prediction, however, will decrease the recall score.

Unlike with Syncrack, we don't have accurate ground truth annotations to measure if our correction approaches are actually improving the prediction. To indirectly measure if the decrease of recall is due to refining the predicted cracks width, we analyzed the grayscale intensity of pixels labeled as crack. Dark pixels are more likely to be part of a crack; therefore, a more accurate crack segmentation should exhibit a lower average intensity. Furthermore, a wider segmentation should exhibit a higher standard deviation since it includes a variety of pixels from healthy pavement.

First, raw images were converted to grayscale –[0, 1]– without standardization. Then, we calculated the average and the standard deviation of the set of crack-labeled pixel intensities per image. Table 11 shows the average from this analysis in all the validation images. With respect to manual annotations, both average intensity and standard deviation are slightly lower in the U-VGG19 predictions. Furthermore, both average and standard deviation decrease even more in the predictions of U-VGG19 trained with data augmentation: the segmentation seems to improve despite achieving a slightly lower DSC.

For weak supervision approaches, we can observe the same behavior: the recall decreases, but the average intensity and standard deviation decreased too. Particularly, the average and standard deviation of self-training are lower than the ones of data augmentation. This is congruent with the results obtained from Syncrack, where self-training improved the baseline of data augmentation. However, the decrease of recall could be caused merely by missing entire cracks or crack segments. Therefore, a qualitative analysis is needed.

Table 11: Analysis of pixels predicted as cracks on the validation split of CFD+Aigle-RN.

Method	Average intensity	Intensity standard deviation
Manual annotations	0.3839	0.0811
U-VGG19	0.3791	0.0814
U-VGG19 with Data Augmentation	0.3673	0.0757
U-VGG19 with self-training correction	0.3595	0.0714
U-VGG19 with 5-nn correction	0.3760	0.0803

A comparison of the predictions obtained by the methods from Table 11 is shown in Fig. 8. We focus on self-training since it was the method with the lower average and standard deviation. Arrow A points a dark structure not labeled as crack by the original manual annotation but identified as a crack by the model trained with self-training. This ambiguity is shared with the structure pointed by arrow B. It looks similar (albeit a bit lighter) but the structure was manually annotated as crack, while not predicted as crack by U-VGG19. Both cases are examples of fuzziness caused by low-resolution images; correction of this kind of potential errors remains a challenge. However, arrows C and D point to zones where our label correction approach shows promising results. In case C, the manual annotation is much wider than the actual crack. As we advance along the columns, the prediction gets thinner without missing the visible crack, thus improving the predicted crack width. In case D, the manual annotation is slightly offset with respect to the given visible crack. As we move along the columns, both the shape and the location of the crack improve in our predictions.

Inaccurate annotations are a key element limiting supervised crack detection: 1) the resulting models are inherently biased, and the scores hit a ceiling because of the noise in the ground truth; 2) if we remove the bias from the training data, the discrepancy with the inaccurate ground truth annotations will increase and the scores will decrease. However, methods derived from the field of weakly supervised learning show themselves as very promising options to deal with this. Particularly, these approaches can help to relabel manual annotations into a more accurate ground truth in terms of crack width.

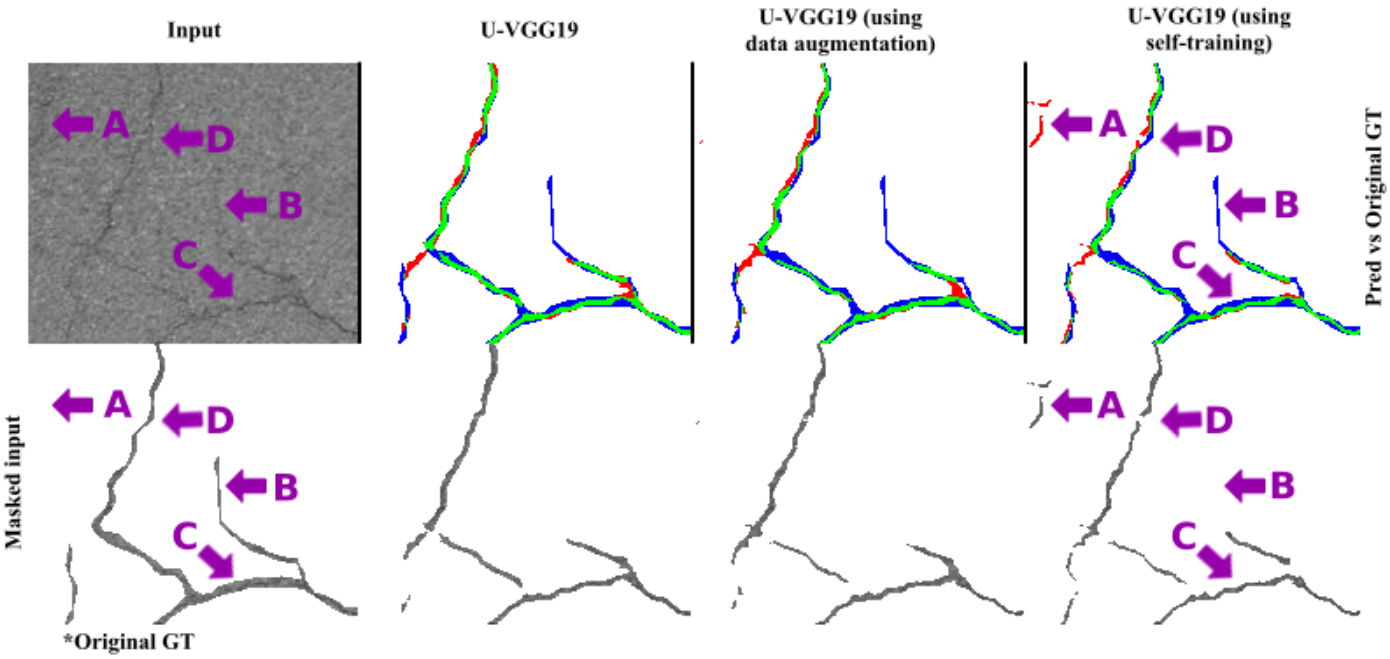


Figure 8: Prediction comparison training with manual annotations, data augmentation and corrected annotations. The color code is the same as in Fig. 4.

7. Conclusions and Future Work

In this paper, we approached the problem of inaccurate manual annotations under the scope of weakly supervised learning. This is possible because mislabels at pixel level (caused by inaccurate annotations) are a case of noisy labels.

First, to approach the crack detection task, we introduced U-VGG19 (a U-net with VGG19 as backbone). U-VGG19 obtained an F-score of 71.77% on the CrackForest dataset. This is competitive and often better than other, more complex, approaches when evaluating without tolerance margins.

Then, to evaluate the impact of inaccurate annotations, we introduced a novel synthetic dataset (Syncrack). This dataset contains pixel-accurate crack annotations, used as a training baseline for U-VGG19. After that, we introduced noise to the annotations to emulate real-life inaccurate annotations. Training with these noisy labels exhibited a very negative impact: a decrease from DSC=83% to 61% on accurate annotations.

To fill this gap, we tested label correction methods inspired by the field of weakly supervised learning. The most successful ones were self-training and 5-nn voting, improving the noisy-label results by up to 12% (from DSC=61% to 73%): more than half the decrease caused by inaccurate annotations. Self-training consists of using U-VGG19 trained with noisy labels to predict new pseudo-labels per image. 5-nn voting, instead, uses the same trained U-VGG19 to project the pixels from an image into a new 2D space; new pseudo-labels per pixel are then obtained using a k-NN algorithm. From both methods, 5-nn voting obtained the best result. 5-nn voting allows avoiding overfitted decision boundaries. However, the number of neighbors is a manual hyperparameter to tune. Increasing this value will increase the complexity without a guarantee of improving the expected results. On the other hand, self-training does not require parameter tuning, being the simplest method to implement. However, it is prone to overfit to its own biases.

We extended these methods to real road images (CrackForest + Aigle-RN). Since no accurate ground truth exists in this case, we provided indirect measurements intended to evaluate the quality of the predicted cracks width. All the provided results showed the promising performance of weak supervision methods to improve crack detection. Particularly, in terms of crack width.

Learning in the presence of noise is an extensively studied field. However, there are no strategies suitably studied for severely imbalanced classes such as in crack detection (with ratios as aggressive as 1/100). Based on the results presented in this paper, improving weak supervision approaches oriented to heavily imbalanced classes can work hand-to-hand with DL strategies, towards improving crack detection.

The code to reproduce our results is available at: `https://github.com/Sutadasuto/weak_supervision_crack_detection`

Acknowledgment

This work has been supported by the project DiXite. Initiated in 2018, DiXite (Digital Construction Site) is a

project of the I-SITE FUTURE, a French initiative to answer the challenges of sustainable city.

References

[1] E. Coquelle, J.-L. Gautier, P. Dokládal, Automatic assessment of a road surface condition, in: 7th Symposium on Pavement Surface Characteristics, Surf, Norfolk, Virginia, 2012.

[2] X. Yang, H. Li, Y. Yu, X. Luo, T. Huang, X. Yang, Automatic Pixel-Level Crack Detection and Measurement Using Fully Convolutional Network, Computer-Aided Civil and Infrastructure Engineering 33 (12) (2018) 1090–1109, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/mice.12412. doi:https://doi.org/10.1111/mice.12412.
URL http://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12412

[3] S. Bhat, S. Naik, M. Gaonkar, P. Sawant, S. Aswale, P. Shetgaonkar, A Survey On Road Crack Detection Techniques, in: 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1–6. doi:10.1109/ic-ETITE47903.2020.67.

[4] Q. Zou, Y. Cao, Q. Li, Q. Mao, S. Wang, CrackTree: Automatic crack detection from pavement images, Pattern Recognition Letters 33 (3) (2012) 227–238. doi:10.1016/j.patrec.2011.11.004.
URL http://www.sciencedirect.com/science/article/pii/S0167865511003795

[5] U. Escalona, F. Arce, E. Zamora, J. H. Sossa Azuela, Fully Convolutional Networks for Automatic Pavement Crack Segmentation, Computación y Sistemas 23 (2) (2019) 451–460–460, number: 2. doi:10.13053/cys-23-2-3047.
URL https://www.cys.cic.ipn.mx/ojs/index.php/CyS/article/view/3047

[6] R. Amhaz, S. Chambon, J. Idier, V. Baltazart, Automatic Crack Detection on Two-Dimensional Pavement Images: An Algorithm Based on Minimal Path Selection, IEEE Transactions on Intelligent Transportation Systems 17 (10) (2016) 2718–2729, conference Name: IEEE Transactions on Intelligent Transportation Systems. doi:10.1109/TITS.2015.2477675.

[7] Y. Shi, L. Cui, Z. Qi, F. Meng, Z. Chen, Automatic Road Crack Detection Using Random Structured Forests, IEEE Transactions on Intelligent Transportation Systems 17 (12) (2016) 3434–3445, conference Name: IEEE Transactions on Intelligent Transportation Systems. doi:10.1109/TITS.2016.2552248.

[8] D. Ai, G. Jiang, L. Siew Kei, C. Li, Automatic Pixel-Level Pavement Crack Detection Using Information of Multi-Scale Neighborhoods, IEEE Access 6 (2018) 24452–24463, conference Name: IEEE Access. doi:10.1109/ACCESS.2018.2829347.

[9] Z. Fan, Y. Wu, J. Lu, W. Li, Automatic Pavement Crack Detection Based on Structured Prediction with the Convolutional Neural Network, arXiv:1802.02208 [cs]ArXiv: 1802.02208.
URL http://arxiv.org/abs/1802.02208

[10] Q. Zou, Z. Zhang, Q. Li, X. Qi, Q. Wang, S. Wang, DeepCrack: Learning Hierarchical Convolutional Features for Crack Detection, IEEE Transactions on Image Processing 28 (3) (2019) 1498–1512, conference Name: IEEE Transactions on Image Processing. doi:10.1109/TIP.2018.2878966.

[11] J. König, M. David Jenkins, P. Barrie, M. Mannion, G. Morison, A Convolutional Neural Network for Pavement Surface Crack Segmentation Using Residual Connections and Attention Gating, in: 2019 IEEE International Conference on Image Processing (ICIP), 2019, pp. 1460–1464, iSSN: 2381-8549. doi:10.1109/ICIP.2019.8803060.

[12] Z. Fan, C. Li, Y. Chen, J. Wei, G. Loprencipe, X. Chen, P. Di Mascio, Automatic Crack Detection on Road Pavements Using Encoder-Decoder Architecture, Materials 13 (13) (2020) 2960, number: 13 Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/ma13132960.
URL https://www.mdpi.com/1996-1944/13/13/2960

[13] L. Zhang, J. Shen, B. Zhu, A research on an improved Unet-based concrete crack detection algorithm, Structural Health Monitoring (2020) 1475921720940068Publisher: SAGE Publications. doi:10.1177/1475921720940068.
URL https://doi.org/10.1177/1475921720940068

[14] Z. Fan, C. Li, Y. Chen, P. D. Mascio, X. Chen, G. Zhu, G. Loprencipe, Ensemble of Deep Convolutional Neural Networks for Automatic Pavement Crack Detection and Measurement, Coatings 10 (2) (2020) 152, number: 2 Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/coatings10020152.
URL https://www.mdpi.com/2079-6412/10/2/152

[15] J. Fang, B. Qu, Y. Yuan, Distribution equalization learning mechanism for road crack detection, Neurocomputing 424 (2021) 193–204. doi:10.1016/j.neucom.2019.12.057.
URL https://linkinghub.elsevier.com/retrieve/pii/S0925231219317667

[16] T. I. Cannings, Y. Fan, R. J. Samworth, Classification with imperfect training labels, arXiv preprint arXiv:1805.11505.

[17] P. Dokládal, Statistical Threshold Selection for Path Openings to Detect Cracks, Vol. 10225, 2017, pp. 369–380. doi:10.1007/978-3-319-57240-6_30.
URL https://hal-mines-paristech.archives-ouvertes.fr/hal-01478089

[18] H. Oliveira, P. L. Correia, CrackIT — An image processing toolbox for crack detection and characterization, in: 2014 IEEE International Conference on Image Processing (ICIP), 2014, pp. 798–802, iSSN: 2381-8549. doi:10.1109/ICIP.2014.7025160.

[19] L. Zhang, F. Yang, Y. Daniel Zhang, Y. J. Zhu, Road crack detection using deep convolutional neural network, in: 2016 IEEE International Conference on Image Processing (ICIP), 2016, pp. 3708–3712, iSSN: 2381-8549. doi:10.1109/ICIP.2016.7533052.

[20] B. Kim, S. Cho, Automated Vision-Based Detection of Cracks on Concrete Surfaces Using a Deep Learning Technique, Sensors 18 (10) (2018) 3452, number: 10 Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/s18103452.
URL https://www.mdpi.com/1424-8220/18/10/3452

[21] L. Guo, R. Li, B. Jiang, X. Shen, Automatic crack distress classification from concrete surface images using a novel deep-width network architecture, Neurocomputing 397 (2020) 383–392. doi:10.1016/j.neucom.2019.08.107.
URL https://linkinghub.elsevier.com/retrieve/pii/S0925231220304112

[22] V. Mandal, L. Uong, Y. Adu-Gyamfi, Automated Road Crack Detection Using Deep Convolutional Neural Networks, in: 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 5212–5215. doi:10.1109/BigData.2018.8622327.

[23] Z. Liu, Y. Cao, Y. Wang, W. Wang, Computer vision-based concrete crack detection using U-net fully convolutional networks, Automation in Construction 104 (2019) 129–139. doi:10.1016/j.autcon.2019.04.005.
URL https://www.sciencedirect.com/science/article/pii/S0926580519301244

[24] Z. Gao, B. Peng, T. Li, C. Gou, Generative Adversarial Networks for Road Crack Image Segmentation, in: 2019 International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1–8, iSSN: 2161-4407. doi:10.1109/IJCNN.2019.8851910.

[25] K. Zhang, Y. Zhang, H.-D. Cheng, CrackGAN: Pavement Crack Detection Using Partially Accurate Ground Truths Based on Generative Adversarial Learning, IEEE Transactions on Intelligent Transportation Systems (2020) 1–14Conference Name: IEEE Transactions on Intelligent Transportation Systems. doi:10.1109/TITS.2020.2990703.

[26] M. Sun, R. Guo, J. Zhu, W. Fan, Roadway Crack Segmentation Based on an Encoder-decoder Deep Network with Multi-scale Convolutional Blocks, in: 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), 2020, pp. 0869–0874. doi:10.1109/CCWC47524.2020.9031213.

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

URL https://linkinghub.elsevier.com/retrieve/pii/S0925231219300566

[28] F. Yang, L. Zhang, S. Yu, D. Prokhorov, X. Mei, H. Ling, Feature Pyramid and Hierarchical Boosting Network for Pavement Crack Detection, IEEE Transactions on Intelligent Transportation Systems 21 (4) (2020) 1525–1535, conference Name: IEEE Transactions on Intelligent Transportation Systems. doi:10.1109/TITS.2019.2910595.

[29] L. Pauly, H. Peel, S. Luo, D. Hogg, R. Fuentes, Deeper Networks for Pavement Crack Detection, 2017. doi:10.22260/ISARC2017/0066.

[30] B. Frénay, M. Verleysen, Classification in the presence of label noise: a survey, IEEE transactions on neural networks and learning systems 25 (5) (2013) 845–869.

[31] Z.-H. Zhou, A brief introduction to weakly supervised learning, National Science Review 5 (1) (2017) 44–53. arXiv:https://academic.oup.com/nsr/article-pdf/5/1/44/31567770/nwx106.pdf, doi:10.1093/nsr/nwx106.
URL https://doi.org/10.1093/nsr/nwx106

[32] W. Liu, L. Zhang, D. Tao, J. Cheng, Support vector machine active learning by Hessian regularization, Journal of Visual Communication and Image Representation 49 (2017) 47–56. doi:10.1016/j.jvcir.2017.08.001.
URL https://www.sciencedirect.com/science/article/pii/S1047320317301633

[33] C. E. Brodley, M. A. Friedl, Identifying mislabeled training data, Journal of artificial intelligence research 11 (1999) 131–167.

[34] V. S. Sheng, F. Provost, P. G. Ipeirotis, Get another label? improving data quality and data mining using multiple, noisy labelers, in: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, 2008, pp. 614–622.

[35] J. Young, J. Ashburner, S. Ourselin, Wrapper methods to correct mislabelled training data, in: 2013 International Workshop on Pattern Recognition in Neuroimaging, IEEE, 2013, pp. 170–173.

[36] G. Liang, X. Wang, Y. Zhang, N. Jacobs, Weakly-Supervised Self-Training for Breast Cancer Localization*, in: 2020 42nd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC), 2020, pp. 1124–1127, iSSN: 2694-0604. doi:10.1109/EMBC44109.2020.9176617.

[37] R. D. Cook, Detection of influential observation in linear regression, Technometrics 19 (1) (1977) 15–18. arXiv:https://doi.org/10.1080/00401706.1977.10489493, doi:10.1080/00401706.1977.10489493.
URL https://doi.org/10.1080/00401706.1977.10489493

[38] R. D. Cook, S. Weisberg, Residuals and influence in regression, New York: Chapman and Hall, 1982.

[39] D. Hao, L. Zhang, J. Sumkin, A. Mohamed, S. Wu, Inaccurate labels in weakly-supervised deep learning: Automatic identification and correction and their impact on classification performance, IEEE Journal of Biomedical and Health Informatics 24 (9) (2020) 2701–2710. doi:10.1109/JBHI.2020.2974425.

[40] P. W. Koh, P. Liang, Understanding black-box predictions via influence functions, arXiv preprint arXiv:1703.04730.

[41] H. Kervadec, J. Dolz, J. Yuan, C. Desrosiers, E. Granger, I. B. Ayed, Constrained Deep Networks: Lagrangian Optimization via Log-Barrier Extensions, arXiv:1904.04205 [cs]ArXiv: 1904.04205.
URL http://arxiv.org/abs/1904.04205

[42] E. Goceri, CapsNet topology to classify tumours from brain images and comparative evaluation, IET Image Processing 14 (5) (2020) 882–889. doi:10.1049/iet-ipr.2019.0312.
URL https://onlinelibrary.wiley.com/doi/10.1049/iet-ipr.2019.0312

[43] Q. Zhang, F. Lee, Y.-g. Wang, R. Miao, L. Chen, Q. Chen, An improved noise loss correction algorithm for learning from noisy labels, Journal of Visual Communication and Image Representation 72 (2020) 102930. doi:10.1016/j.jvcir.2020.102930.
URL https://www.sciencedirect.com/science/article/pii/S1047320320301619

[44] J. Zhang, G. Wang, H. Xie, S. Zhang, N. Huang, S. Zhang, L. Gu, Weakly supervised vessel segmentation in X-ray angiograms by self-paced learning from noisy labels with suggestive annotation, Neurocomputing 417 (2020) 114–127. doi:10.1016/j.neucom.2020.06.122.
URL https://www.sciencedirect.com/science/article/pii/S0925231220312261

[45] M. Koziarski, B. Krawczyk, M. Woźniak, Radial-Based oversampling for noisy imbalanced data classification, Neurocomputing 343 (2019) 19–33. doi:10.1016/j.neucom.2018.04.089.
URL https://linkinghub.elsevier.com/retrieve/pii/S0925231219301596

[46] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015.
URL https://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a/

[47] S. Bang, S. Park, H. Kim, H. Kim, Encoder–decoder network for pixel-level road crack detection in black-box images, Computer-Aided Civil and Infrastructure Engineering 34 (8) (2019) 713–727, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/mice.12440. doi:https://doi.org/10.1111/mice.12440.
URL http://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12440

[48] D. Mazzini, P. Napoletano, F. Piccoli, R. Schettini, A Novel Approach to Data Augmentation for Pavement Distress Segmentation, Computers in Industry 121 (2020) 103225. doi:10.1016/j.compind.2020.103225.
URL http://www.sciencedirect.com/science/article/pii/S0166361519310516

[49] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv:1409.1556 [cs]ArXiv: 1409.1556.
URL http://arxiv.org/abs/1409.1556

[50] E. Goceri, Analysis of Deep Networks with Residual Blocks and Different Activation Functions: Classification of Skin Diseases, in: 2019 Ninth International Conference on Image Processing Theory, Tools and Applications (IPTA), 2019, pp. 1–6, iSSN: 2154-512X. doi:10.1109/IPTA.2019.8936083.

[51] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, Activation functions: Comparison of trends in practice and research for deep learning, arXiv preprint arXiv:1811.03378.

[52] Y. Zhou, H. Yu, H. Shi, Study group learning: Improving retinal vessel segmentation trained with noisy labels (2021). arXiv:2103.03451.

[53] R. S. Boyer, J. S. Moore, Mjrty—a fast majority vote algorithm, in: Automated Reasoning, Springer, 1991, pp. 105–117.

[54] S. M. Omohundro, Five balltree construction algorithms, International Computer Science Institute Berkeley, 1989.

[55] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (9) (1975) 509–517.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[57] N. Ibtehaz, M. S. Rahman, Multiresunet : Rethinking the u-net architecture for multimodal biomedical image segmentation, Neural Networks 121 (2020) 74–87. doi:https://doi.org/10.1016/j.neunet.2019.08.025.
URL https://www.sciencedirect.com/science/article/pii/S0893608019302503