Appendix: Designing a Hybrid Neural System to Learn Real-world Crack Segmentation from Fractal-based Simulation

In this supplementary material, we provide additional details, experimental setup and descriptions for the various parts of our proposed systems (Cracktal and CAP-Net), as well as discuss additional results and highlight the limitations of our approach. The structure is as follows:

A. Experimental setup and implementation details of the Cracktal Simulator.

B. Additional training and design details of CAP-Net.

C. Additional results on in-distribution Cracktal data.

D. Further qualitative results on real world images.

E. Discussion of the limitations of our current approach and possible future improvements.

A. Cracktal Details

In this section, we discuss some fundamentals of the physics based workflow and the texture maps used in Cracktal. We also list the hyperparameters that have been used to generate our specific Cracktal data set.

A.1. Rendering Maps and Crack Simulation Model

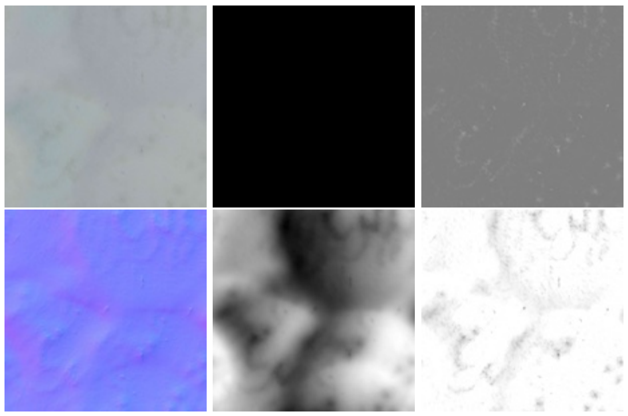

Figure 1. PBR Maps used to generate a concrete surface. Textures were compressed to view in PDF. From left to right, top to bottom: albedo, metallic, roughness, surface normal, height, and ambient occlusion map.

PBR Maps: Physics-based rendering (PBR) is an approach to rendering scenes that aims to accurately simulate the behavior of light in the real world. In PBR, materials are defined by their physical properties, such as color (in the case of dielectrics), roughness, and surface normals, which are used to calculate how light interacts with the material. In practice, PBR relies on a set of texture maps that define the physical properties of each object in the scene (see Figure 1). These texture maps include:

- Color/albedo map: This texture map defines the base color of the material for our case of the fully dielectric concrete (or alternatively reflectance value for metals), which represents the color that is observed under diffuse lighting conditions.
- Metallic Map: This map defines the metallicity of a material. Values range from non-metallic (zero values) to fully metallic (one values), and similarly to a mask, define how to treat the above color map's values in specific regions with respect to reflected color or specularity.
- Roughness Map: This map defines the smoothness of the material, with values ranging from very rough to very smooth. Whereas the amount of reflected light is always conserved, roughness determines diffuse scattering. That is, the reflected direction becomes increasingly random the rougher the surface.
- Normal Map: This map encodes the surface normals of a material in RGB, defining which direction a surface faces and thus affecting light and shadow calculations. Normal maps are typically created by capturing fine surface details through photogrammetry.
- Height map: This map is used to modify the surface geometry. It defines the height variations of the surface of an object in a grayscale map, where brighter values represent higher elevations and darker values represent lower ones. It is used in conjunction with other material properties, such as roughness and surface normal maps, to calculate how light interacts with the surface of an object and providing a more realistic "bumpmapped" look.

- Ambient Occlusion Map: This map defines the occluded areas on a surface, which are areas that receive less light due to being blocked or shadowed by other objects. AO maps help to add depth and realism to a material by accentuating the small details and crevices in a surface.

By using these texture maps in a PBR renderer, it is possible to create highly realistic materials that look similar to their real-world counterparts under different lighting conditions. For more detailed information, we defer to popular PBR guides, such as by Adobe's Substance[1].

Crack Model: As explained in the main body, the crack model is implemented using an irregular fractal model. In our experiments, the subdivision depth is set to 7. Furthermore, the width of the crack is obtained by generating a random kernel size (3 or 5) for a Gaussian blur and applying it to the crack map. Figure 2 illustrates the crack generation process. The obtained crack is then added to the scene.

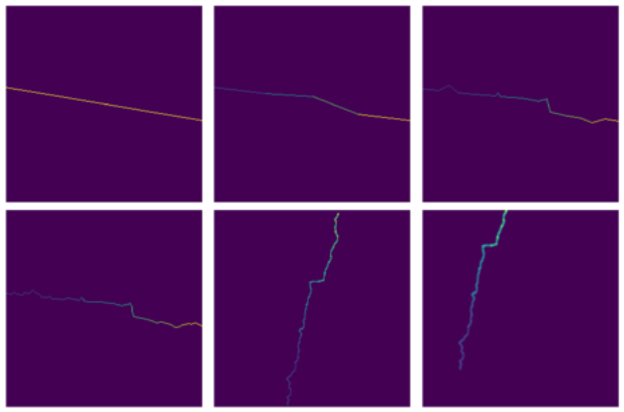
Figure 2. Crack Generation steps. A straight line is modified iteratively using the stochastic midpoint displacement algorithm. In the final steps the crack is translated, rotated and width is then added by applying a Gaussian blur filter.

A.2. Implementation Details

We generate our data using Blender Cycles PBR engine. Our simulator code is written using Blender's Python API, which provides a comprehensive set of functions and classes for interacting with the Cycles Engine. The code was scripted in blender version 2.79 on a Linux OS. We will open source the Cracktal code, the underlying textures, as well as the generated data for training and validation. With respect to surface alterations, the employed moss textures were downloaded from textures.com [2] and the graffiti textures from turbosquid.com [3].

[1]see https://substance3d.adobe.com/tutorials/courses/the-pbr-guide-part-2 PBR Guide 2018

[2]https://www.textures.com/search?q=moss

[3]https://www.turbosquid.com/FullPreview/490921

Rendering Hyperparameters: We set the sampling factor for the data generation to 20. When rendering an image, the rendering engine sends out rays from the camera into the scene and estimates the color of the first object that each specific ray intersects with. To reduce aliasing and noise, the renderer takes multiple samples per pixel and averages the estimated colors to obtain a more accurate representation of each pixel in the scene. Generally, higher sampling rates lead to more accurate and realistic results but also require more computational resources and time. We chose a rendering tile size of 256×256 (for parallelization) and a resolution of 2048×2048 for the final image.

B. CAP-Net

In this section, we present further details and visualization of the main modules in our proposed system, CAP-Net. We also document our training and validation procedure.

B.1. Pointwise mutual information

The basic assumption underlying the choice of a pointwise mutual information module as an inductive bias is that the statistical association between pixels belonging to the background texture is high, whereas for pixels belonging to anomalies their statistical association with neighboring pixels is low. We hypothesize that by leveraging these statistical associations between neighboring pixels for learning, we can achieve better generalization from simulated to real images, which is empirically corroborated in the main body. Figure 3 presents some visualization of the obtained pixel wise affinity scores. Here we see that cracks and other anomalies like surface holes, moss or graffiti edges are assigned lower scores.

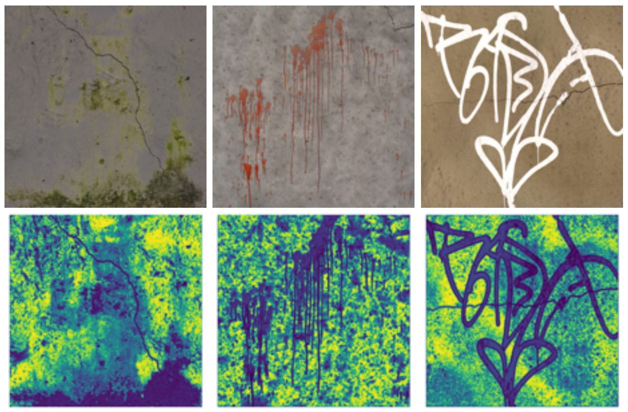
Figure 3. Examples of the affinity maps obtained by the pointwise mutual information module. Images are compressed for view in PDF. Upper row shows the original images from Cracktal dataset and the lower row is their corresponding affinity maps.

B.2. Adaptive Instance Normalization

Figure 4 visualizes some examples of images transformed by the AdaIN module. We use a similar design to that of the original AdaIn work [2]. The style transfer network takes a content image c and an arbitrary style image s as inputs, and synthesizes an output image that recombines the content and the style. A simple encoder-decoder architecture is used, in which the encoder f is fixed to the first four layers of VGG-19 pretrained on ImageNet. After encoding the content c and style s images to the latent feature space, both feature maps are passed to an AdaIN layer that aligns the mean and variance of the content feature map. A randomly initialized decoder is trained to map the obtained features to the image space, generating the stylized image. The loss function of the decoder is computed as follows:

$$\mathcal{L} = \lambda \cdot \mathcal{L}_s + \mathcal{L}_c \tag{1}$$

which is a weighted combination of the content loss $\mathcal{L}_c$ and the style loss $\mathcal{L}_s$ with the style loss weight λ. The content loss term $\mathcal{L}_c$ is computed as the Euclidean distance between the target features and the features of the output image. The style loss $\mathcal{L}_s$ is the Gram matrix loss between the decoded image and the style image s.

We set the style weight $\lambda = 10000$ and perform 1000 iteration steps to generate a stylized image. For each image in our training set, we generate 5 stylized images and pick randomly between them during training. As an optimizer for the decoder, we use LBFGS [4]. As observable in figure 4, the background texture is changed to match the style of the style image s, but the crack remains visible.

B.3. Implementation Details

In our experiments, we use a U-Net [6] architecture as the backbone of our semantic segmentation network. The encoders are implemented in PyTorch and trained for 30 epochs on two A100 Nvidia GPUs using Adam with an initial learning rate of 0.001 and weight decay of 0.0005, mini-batch size 32. We train the models using synthetic concrete images generated with our Cracktal simulator. We save the model with the highest validation accuracy on a validation set of synthetic images.

During training, we downsample the Cracktal images to a resolution of 512×512 and then randomly crop a 256×256 patch. We don't perform any data augmentation during training. The training set is composed of 8800 images, as this amount seems to saturate performance (see section C).

For evaluation on SegCODEBRIM, we process and segment the original images with resolution 1500×844 patch-wise. Each patch has a resolution of 512×512 and is then downsampled to 256×256 to input to the neural network.

For the pointwise mutual information estimation, we sample 10000 pixel pairs to perform the kernel density estimation, which is a good computational trade-off given that the input images have a 256×256 resolution. The distance between two pixels in a pair is between 1 and 8 pixels. As a kernel density estimator, we use gaussian KDE with automatic bandwidth estimation [7]. For each pixel, we estimate the affinity by calculating the PMI scores with its neighbouring pixels in a radius of 5 from the origin. As a hyperparamater for the PMI, we set $\tau = 2.25$.

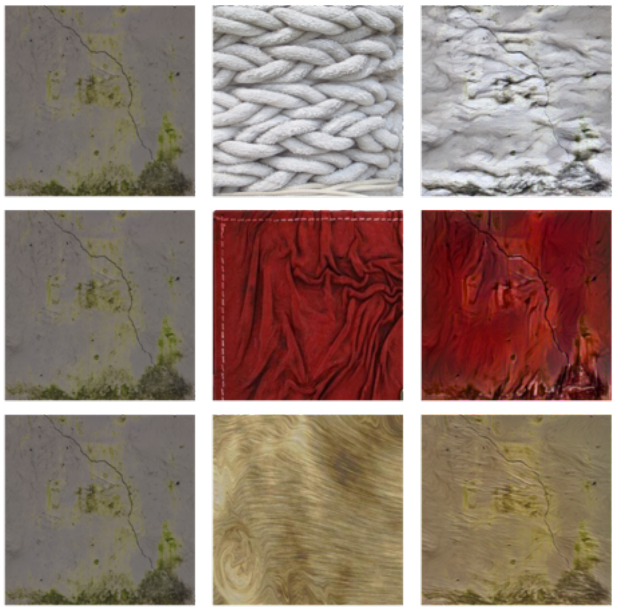
Figure 4. Adaptive Instance Normalization examples. Images are compressed for view in PDF. From left to right: original, style, and stylized images.

C. Further Details for Evaluation Metrics

The authors of [8] introduce a similarity measure centerlineDice (clDice), calculated by comparing the intersection of the prediction and ground truth masks and their morphological skeleta. Given two binary segmentation maps, ground truth GT and prediction P, S_{GT} and S_P are the respectively extracted skeletons. Subsequently, the fraction of S_X that lies within Y (Topology Precision), and vice-a-versa (Topology Sensitivity):

$$T_{prec}(S_P, GT) = \frac{| S_P \cap GT |}{S_P} \tag{2}$$

$$T_{sens}(S_{GT}, P) = \frac{| S_{GT} \cap P |}{S_{GT}} \tag{3}$$

These can then be used to define the clDice score:

$$clD(GT, P) = 2 \times \frac{T_{prec}(S_P, GT) \times T_{sens}(S_{GT}, P)}{T_{prec}(S_P, GT) + T_{sens}(S_{GT}, P)} \tag{4}$$

D. Additional Results

For our experiments in the main body, we have used a training set of 8800 synthetic images from the Cracktal simulator. Figure 5 illustrates the impact of training set size on the performance for the baseline U-Net model. Here we observe that after a set size of 4000 images the performance appears to stagnate on the $F1$ and $clDice$ metrics. On the HDF_euc, we observe an increase of almost 2 between set size 4000 and 8800. It appears that our Cracktal simulator generates a good variety of examples and that a training set of around 8800 is reasonable.

As a complement, table 1 further highlights the test set performance of different models on the in-distribution Cracktal data. Here we observe that the baseline U-Net slightly outperforms the other models on $F1$, $F1_{\theta=10}$, HDF_{RBF}. On the clDice and HDF_{Euc} metrics, U-Net also achieves the best mean value performance, but the obtained values lie within statistical deviation when contrasted with our CAP-Net design. As observed in the main body's results, the performance of U-Net drops more significantly on out-of-distribution data (multi-source set and SegCODE-BRIM) compared to our proposed model CAP-Net. This shows that in-distribution performance is not predictive of the performance on out-of-distribution data.

Furthermore, CAP-Net is more robust than the simple fully data-driven U-Net. Figure 6 contains multiple qualitative examples of the baseline U-Net and CAP-Net. Here it is evident that our design choices improve the performance overall and help detecting cracks that U-Net misses.

E. Limitations and Prospects

In this section, we highlight potential remaining challenges and areas of improvement in our approach.

Pointwise Mutual Information (PMI): The computed PMI values depend on the chosen hyperparameters, such as the image scale or neighborhood size. While some solutions have been presented in the literature to address some of these challenges, such as scale invariance [3], these come with a substantial computational overhead. In contrast, the simplified PMI estimation presented in this work offers a good trade-off in terms of computational efficiency, especially when deployed in conjunction with neural networks. We also note that if the input image consists of extensive amount of cracked surface, where the cracked pattern itself forms a regular texture, PMI will dampen the response for crack detection (which no longer constitutes an anomaly). However, in real-world applications, we are interested in early onset detection of crack formation, before it's too late and structural collapse is imminent. Hence this situation is unlikely to be faced in real-world practice.

Adaptive instance normalization (AdaIN): We made use of textures from the Describable Textures dataset [1] to perform the style transfer. One immediate prospect for improvement lies in leveraging a concrete inspection dataset, which could potentially offer a more diverse range of textures that are even more relevant to the task at hand.

The challenge of crack scale: Another possible challenge of our current system is the cracks' scale. If the cracks are too wide, our model has difficulty detecting it. This is mainly because we only generated thin cracks in Cracktal. Again, this focus stems from the fact that inspectors are mainly interested in the early formation of cracks in real world scenarios. Wider cracks are usually found in concrete surfaces where the inspectors are already aware that the surface is damaged and were structures are severely damaged, potentially beyond repair.

	Model	$F1(\uparrow)$	$F1_{\theta=10}(\uparrow)$	$clDice(\uparrow)$	$HDF_{Euc}(\downarrow)$	$HDF_{RBF}(\downarrow)$
Cracktal	U-Net [6]	75.3 ± 0.7	82.4 ± 0.2	88.2 ± 0.4	11.0 ± 0.9	15.3 ± 0.2
	Attn-U-Net [5]	73.3 ± 0.6	81.4 ± 0.1	86.3 ± 0.7	13.6 ± 1.1	17.6 ± 0.7
	Multi-U-Net (D-SN)	73.2 ± 1.1	81.4 ± 0.8	84.2 ± 1.5	13.6 ± 3.1	14.5 ± 2.7
	Multi-U-Net (PMI)	72.8 ± 4.1	80.4 ± 3.1	83.2 ± 7.3	14.3 ± 5.2	18.5 ± 6.1
	CAP-Net	72.2 ± 0.9	80.1 ± 0.4	86.9 ± 1.3	11.7 ± 1.5	17.6 ± 0.8

Table 1. Performance comparison of different models on a testset of Cracktal data.

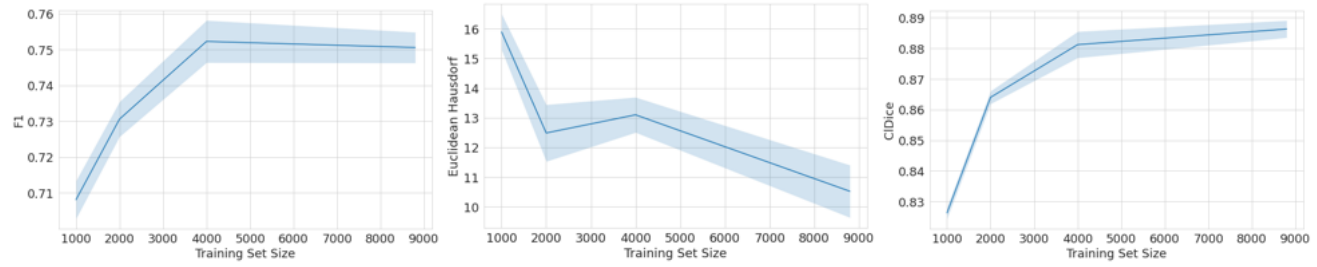

Figure 5. Evolution of segmentation performance in terms of training set size for baseline U-Net on Cracktal testset. We visualize the performance using different metrics. From left to right: F1, Hausdorff distance with euclidean measure, clDice.

References

[1] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2014. 4

[2] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017. 3

[3] Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H Adelson. Crisp boundary detection using pointwise mutual information. In *European Conference on Computer Vision*, pages 799–814. Springer, 2014. 4

[4] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989. 3

[5] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018. 5

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 234–241. Springer, 2015. 3, 5

[7] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015. 3

[8] Suprosanna Shit, Johannes C Paetzold, Anjany Sekuboyina, Ivan Ezhov, Alexander Unger, Andrey Zhylka, Josien PW Pluim, Ulrich Bauer, and Bjoern H Menze. cldice-a novel topology-preserving loss function for tubular structure segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16560–16569, 2021. 3

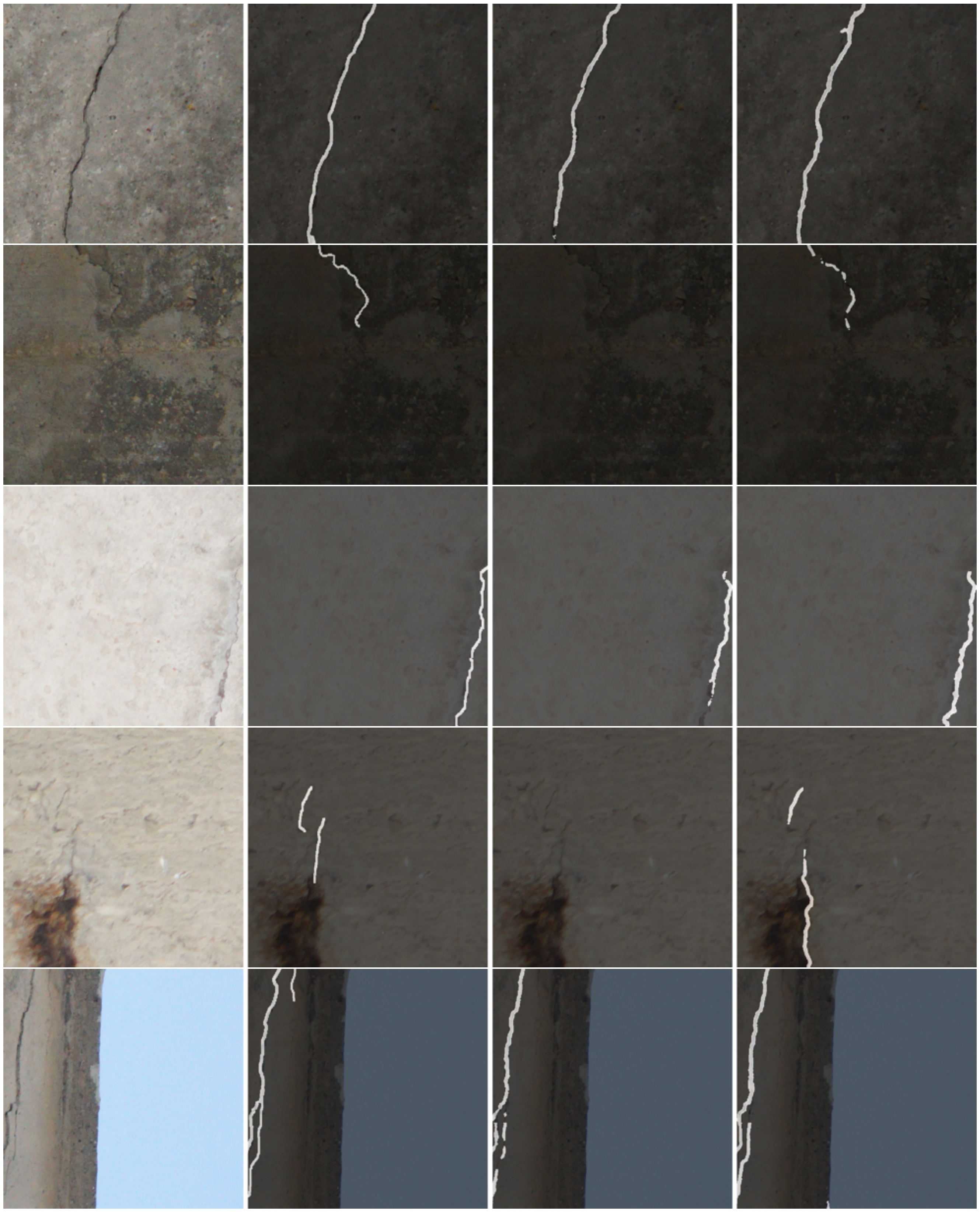
Figure 6. Qualitative examples on SegCODEBRIM from left to right: input image, ground-truth, U-Net, CAP-Net (ours). In general, CAP-Net segments cracks that U-Net often misses entirely. Images are compressed for view in pdf.