Computer Vision Approaches for Crack Detection

Marcus Hyge Ritika Giridhar Simen Nordraak Kjekshus Vithujan Paskaralingam

(Names ordered alphabetically)

Abstract

Cracks provide important information for evaluating the building condition and conducting the necessary building maintenance. This paper presents two methods to detect cracks present in a group of images, taken at random from an online dataset, using computer vision and unsupervised machine learning methods. The first method uses image convolution and convolution masks to detect the cracks, and the second method uses contours to detect the cracks. The method using convolution masks gave an accuracy of 70-76±4%, while the method using contours gave an accuracy of 92-98±0.8%. Covered issues include choice of the paper subject, background and literature review, method, implementation, results and further discussion.

Keywords—computer vision; machine learning; crack detection; PCA feature extraction

1. Introduction

The most commonly used building materials, namely masonry, concrete, mortar etc., are weak in tension and shear. Therefore, stresses of even a small magnitude, causing tension and shear stresses, can lead to cracking[5]. Cracks on material surfaces are one of the earliest indicators of structural deterioration, causing both environmental and structural damages. It is important to detect cracks as early as possible, with manual inspections being the most frequently used method. In this method, a sketch of the crack is drawn by hand, and the conditions of the irregularities are noted down. However, the manual approach requires a specialist, and is entirely dependent on the specialist's expertise and experience. Thus, this approach lacks impartiality in quantitative analysis, and may also result in some cracks going undetected. An alternative approach to manual inspection is an automatic crack detection system that works by leveraging different methods from computer vision like gradient thresholding and edge detection[13]. Using image processing techniques, the captured or scanned images of the infrastructure parts can be analyzed to identify any possible defects. Apart from image processing, machine learning methods are being increasingly applied to ensure better performance outcomes and robustness in crack detection[10]. In practice, crack detection is a very challenging problem because of the low contrast between the cracks and the surrounding surfaces, the intensity inhomogeneity along the cracks, and the possible shadows with similar intensity to the cracks[19].

Figure 1. Different positive and negative crack images from the dataset

This project was set as a learning opportunity to figure out how machine learning and computer vision algorithms work, and how to implement them to specific cases. We choose the crack-detection problem because of its interesting prospects, and the fact that we didn't need any special equipment. The project is divided into two main parts—one is the code[2, 3] and the other is the scientific report.

2. Background and Literature Review

There is a wide range of literature on crack detection. In these literatures, the unsupervised approaches can be divided into 4 categories- image thresholding-based methods, wavelet transform-based methods, anomaly detection-based methods and contour detection-based methods.

2.1 Image thresholding methods

In [6], the authors use preprocessing techniques which consists of binary segmentation, morphological operations and an algorithm that removes isolated dots and areas. Thresholding is then applied to the image to give crack candidates. While this approach is effective for longitudinal and transverse crack types, it is ineffective for alligator crack types.

2.2 Wavelet transform-based methods

In [14], a 2D continuous wavelet transform is used to build a complex coefficient map. Crack candidates can then be determined by the maximal wavelet coefficients through scales from the largest to the smallest one. Due to the anisotropy attributes of wavelet, this approach is ineffective to detect cracks with high curvature or low continuity traits.

2.3 Anomaly detection-based methods

In [15], the authors used Patch-PCA transformation to extract the features and create feature maps through convolutions. This is done to identify anomalies easier. In order to detect the anomalies, statistical analysis was performed using a contrario method. While this approach is a solution to detect cracks, it is not a very efficient solution as it lacks a reliable anomaly segmentation.

2.4 Contour detection-based methods

In [8], a contour-based algorithm is used to extract the boundaries of cracks. Before the use of the algorithm, a median filter is used to enchant the image by removing some noise. And after the use of the contour-algorithm, a support vector machine based on greedy search is used to fully eliminate noise from the image, leaving only the crack.

Our first method is based on convolution, which was inspired by [15]. After reading the article, we figured that we could use convolution to create a filter to find symmetrical boundary in images to find crack candidates. Thus, we opted to use this approach. However, after experimenting this approach we realized another method would be better suited for detecting cracks more accurately. Hence, our second approach is based on the use of contours, which was inspired by [8]. Both of our proposed methods are unsupervised, meaning there is no need for training data or prior knowledge of whether the image contains a crack or not.

3. Method

We used two methods to detect the cracks—one was by using convolution masks (v1), and the other was by using contours from the OpenCV library (v2).

3.1 Data preparation

The dataset is split equally into positive images (ones with cracks in them) and negative images (ones with no cracks in them). Performing the algorithms on 40,000 images would take up too much time and memory, so the functions *getIm()* and *getRandomImage()* were created to get a user specified number of random images each, from both the positive and the negative part of the dataset. Tests were often done on 100 or 500 images.

3.2 Image pre-processing

3.2.1 Using convolution masks

Images utilized by version 1 of the application need to have a square root for its x and y value. This is done by function *cutIm()* which cuts the ends of the image so that the closest x and y value for the image which has a square root is set.

The images, that are initially in RGB format, are converted into grayscale by combining the three-color channels in order to find luminance, the channels are each multiplied by a set value, before being added together. This is done in order to convert the image from a 3D RGB image into a 2D grayscale image, as grayscale simplifies the algorithm and reduces computational requirements[7, 16]. The function *grayscale()* is responsible for the conversion.

3.2.2 Using contours

The images are converted into 2D grayscale images using function *cv2.cvtColor()* from the OpenCV library.

3.2.3 PCA Feature Extraction

Another method that we implemented in version 1 was PCA feature extraction. PCA stands for “Principal Component Analysis”, and is a method to reduce the dimensionality of the data. By implementing and using the PCA as function *pcacomp()*, one can reduce the data and dimensionality to a much more reasonable size, so the algorithm can run faster, while also keeping most of the important data. In short, the unimportant parts in a dataset are removed, while keeping the important parts.

We know which part of the data is interesting by looking at the covariance matrix[1] and by finding the eigenvalues[17]. This gives a good representation of what data is important and should be saved.

The PCA was implemented in these steps-

1. Reshaping the data from RGB to grayscale to fit into the right dimensionality
2. Finding the covariance
3. Calculating the eigenvectors and eigenvalues
4. Creating the mask filter and applying it into to the original data

3.3 Crack classification

3.3.1 Using convolution masks

Image convolution is performed on the image array using eigenvalues[17] and eigenvectors[18]. Convolution provides a way of `multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality[4]. Image convolution is done using function *convolve()*.

Function *checkIfCrack()* takes an image, opened through the PIL library, as a parameter. If the gradient orientation pixel value is 0.0, then this is counted as a true value, indicating a crack may be present. Once a true or false value has been found for all pixels within the image, these values are then compared against a threshold value, meaning that a certain amount of true or false values must be present for the image to be recognized as a crack. The threshold value may be changed to improve accuracy, but is currently set at 0.1% difference required.

3.3.2 Using contours

Contours are a curve joining all the continuous points (along the boundary) having same color or intensity[11]. There are three arguments in *cv2.findContours()* function, first one is source image, second is contour retrieval mode, third is contour approximation method. The function outputs the contours and hierarchy, and each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object. After the number of contours have been found, the result is then compared to the total amount of pixels in the image, and whether the image contains a crack is decided based on whether the number of contours is greater than a certain percent of this value.

3.4 Printing the results

Finally, the results are printed, which shows how many total images there were (positive and negative), the number of false images (positive and negative), and the accuracy score.

4. Implementation

4.1 Google Colaboratory (Colab)

We used Colab to write the code so that everyone had easy access to the source code. In Colab, it's easy to make your own code snippets that don't interfere with each other. This meant that we could split the work and also work simultaneously. Colab also comes with a lot of the packages pre-installed, so all we had to do was import them without thinking about installing them on our own machine.

One of the problems that occurred was when we pair programmed, and sometimes one pair's code would overwrite the other pair's code. In hindsight, we probably could have used GitHub from the beginning, and split it up into different branches. That way, the pull requests would catch any removal of code. It's also easier when it comes to dataset handling than in Colab, as the images in Colab have to be uploaded after each session since Colab doesn't save it.

4.2 Data set

Finding a good data set is important for an algorithm and it`s subsequent training. We quickly decided that making our own data set would be too much work, so we chose to use a data set from the internet. We used Google's own dataset search and found a dataset called "Surface Crack Detection"[20]. We chose this dataset because of its massive collection of pictures, both with cracks and without. This dataset is also very popular, with 49 code examples shared and 6093 downloads on its original web page.

4.3 Patches

A technique that possibly could allow for higher accuracy within the crack detection algorithm is by splitting the images into smaller "patches" to allow the algorithm to judge a smaller part of the image, and thus remove some of the issues caused by noise within the images. This was originally implemented by analyzing the full image, and then also analyzing the fractions of this image. The image was split into four parts and each of these were then analyzed. This process was repeated until the size of the analyzed image was below a given threshold. This solution did not improve the accuracy of the algorithm, as the results were largely dependent on the first analysis when the whole image was taken into account. If no crack was discovered at this stage, it would not be discovered later. Seeing as its effect on the accuracy was negligible, and its effect on the performance was substantial, this solution was ultimately not utilized.

4.4 Git

Git was used for the latter part of the project when segmenting the code. It was important that the folders containing test data, and the structure of the files was kept, which Colab does not allow. The finished code for the project, along with a readme file detailing how to run the project, is found on GitHub[2]. Or alternatively at this link: https://github.com/MarcusStud/cv2206-project. Do note that the Git repo does not contain an executable, as the file sizes became too large for this to be a viable option.

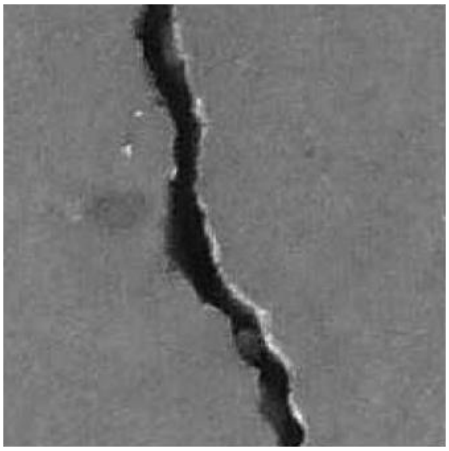
4.4 OneDrive

The final version of the project with an executable file is hosted on OneDrive[3], as it allowed for files of greater sizes than Git to be uploaded, and was a convenient option for hosting the final version of the project. The project may alternatively be found at this link: https://onedrive.live.com/?authkey=%21AN7jzq6U%2DbaEYn0&id=160AD1121AA684E7%2197008&cid=160AD1121AA684E7

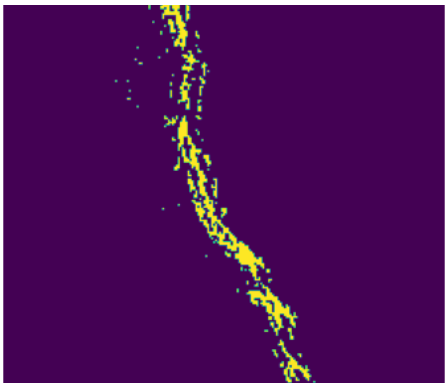
5. Results and Discussion

Both methods were tested on sample images of concrete walls which either had or did not have cracks present. Figures 2 and 4 show the way that the methods attempt to detect cracks in images where cracks are present, whereas figures 3 and 5 show how the methods will act when they receive an image where a crack is not present.

(a)



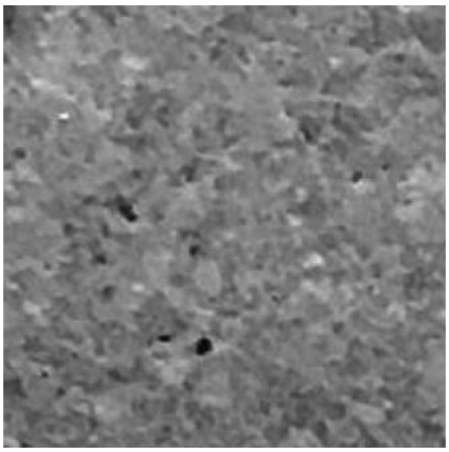
(b)



(c)

Figure 2. Displays crack detection method when a crack is present using convolution (a) Original image (b) Image converted to grayscale (c) Gradient orientation of the image

(a)

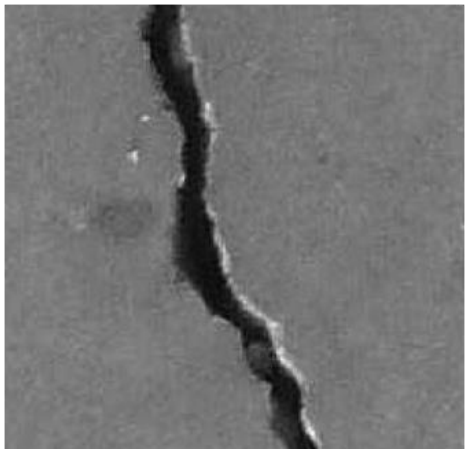


(b)

(c)

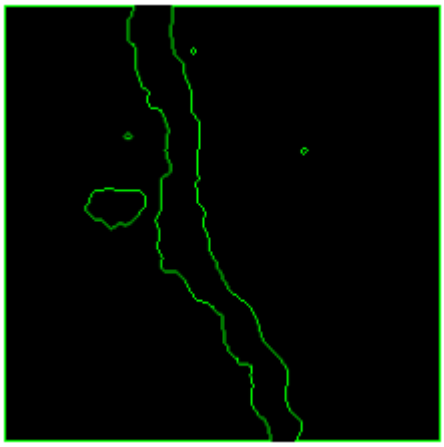
Figure 3. Displays crack detection method when a crack is not present using convolution (a) Original image (b) Image converted to grayscale (c) Gradient orientation of the image

(a)



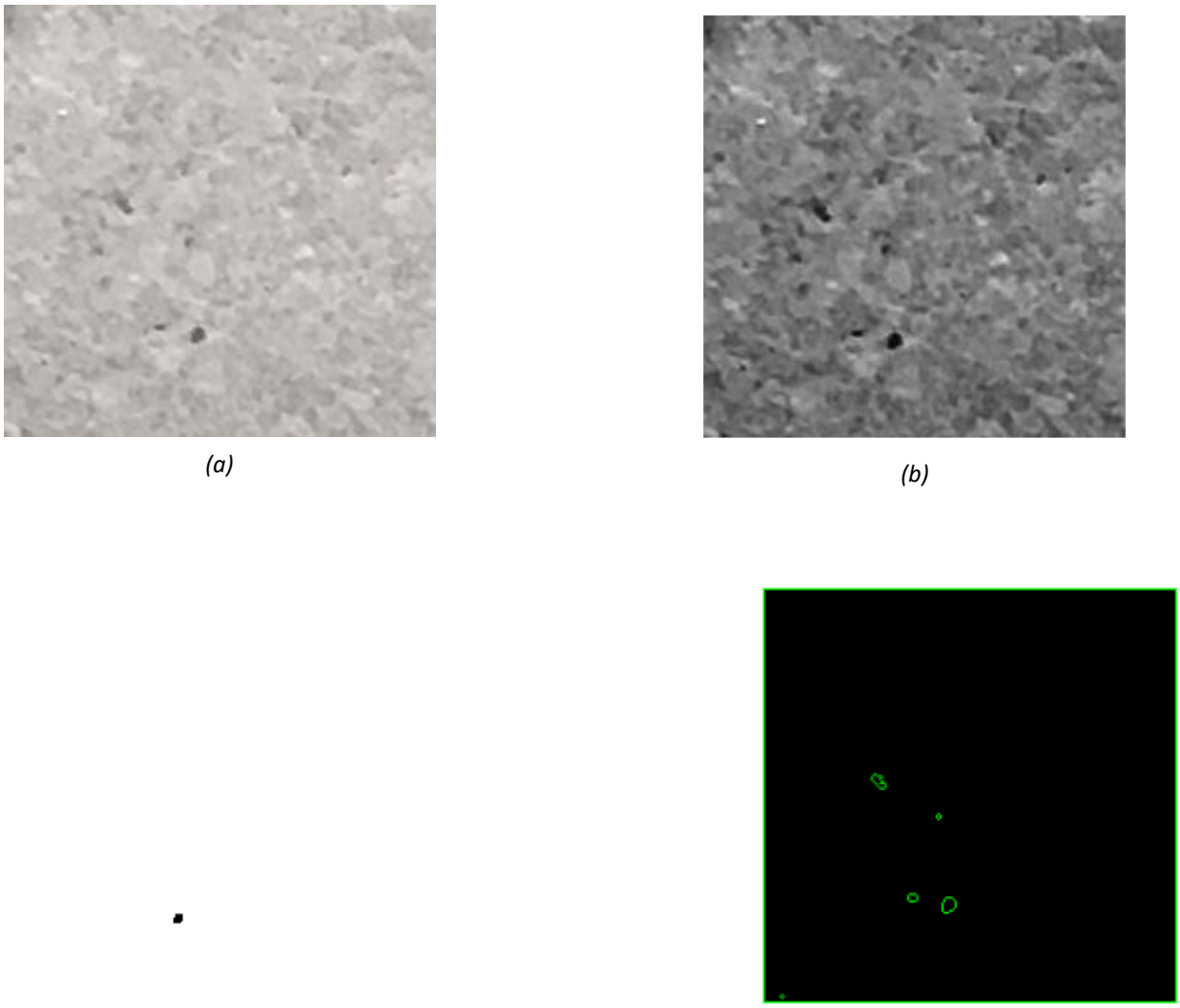
(b)

(c)



(d)

Figure 4. Displays crack detection method when a crack is present using contours (a) Original image (b) Image converted to grayscale (c) Binary inverse threshold of image (d) Contours found



(a) *(b)*

(c) *(d)*

Figure 5. Displays crack detection method when a crack is not present using contours (a) Original image (b) Image converted to grayscale (c) Binary inverse threshold of image (d) Contours found

Figures 6 and 7 show the confusion matrices for both methods. Based on the matrices, it can be concluded that both methods provided a higher accuracy than randomly guessing, and are thus successful in what they attempt to accomplish. The contour-based approach provides a markedly better accuracy than the convolution-based approach, which may also be discerned when looking at the F1-scores of the methods which are as follows:

Contour-based approach:

$$\frac{98.8}{98.8 + \left(\frac{1}{2}\right) * (7.6 + 1.2)} = \frac{247}{258} \approx 0.96$$

Convolution-based approach:

$$\frac{71.6}{71.6 + \left(\frac{1}{2}\right) * (23.6 + 28.4)} = \frac{179}{244} \approx 0.73$$

There are multiple aspects that affect the accuracy of the convolution-based approach, and there may therefore be ways of improving its accuracy. The gamma value of the image after it is converted to grayscale would affect the balance between false positives and false negatives, where a higher gamma value would lead to more false positives, but less false negatives. In addition, the threshold number of pixels that must be detected as cracks before the image is recognized as having a crack will also affect this.

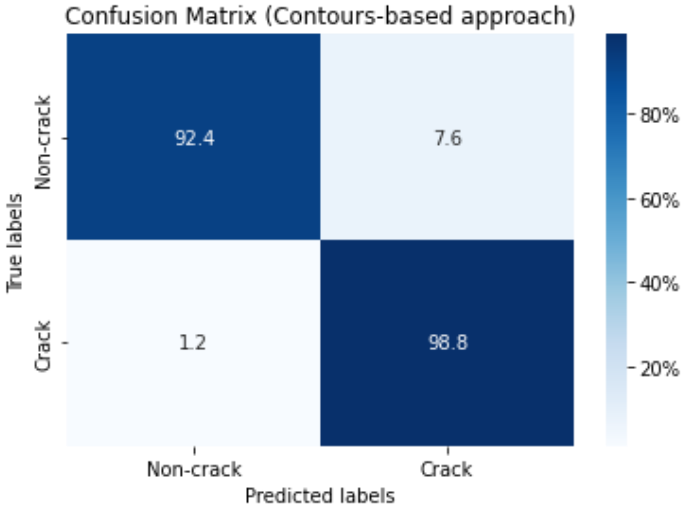


Figure 6

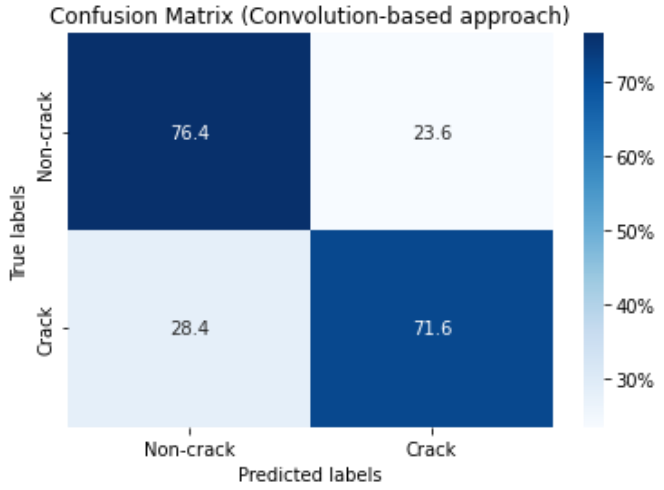


Figure 7

The contours-based approach needed 1.1 seconds to check 1000 images, whereas the convolution-based approach needed 235.3 seconds to check the same number of images. Both methods were timed using python's time library and were run on the same computer.

Another method to approach this problem would be to use supervised algorithms. According to [9,12], using deeply supervised methods to detect cracks on pavements using convolutional neural networks (CNN) provide high accuracy, robustness and efficiency. [9] have used a class-balanced cross-entropy loss function is designed to balance backgrounds and cracks by increasing the weight of crack pixel loss, while [12] have proposed a deeply supervised convolutional neural network for crack detection via a novel multiscale convolutional feature fusion module.

6. Conclusion

In this paper, two ways of detecting cracks have been proposed. The methods are contours-based and convolution-based, and provide an accurate prediction as to whether or not a crack is present within an image. The speed of the convolution-based approach leaves something to be desired, whereas the contours-based approach runs well. The classification of crack types and degrees of severity, which have high application values, are the problems that we would consider for our future work.

References

1. Cansiz S. (2021), *5 Things You Should Know About Covariance*, Accessed 03.05.2022
https://towardsdatascience.com/5-things-you-should-know-about-covariance-26b12a0516f1

2. Code on GitHub
https://github.com/MarcusStud/cv2206-project

3. Code on OneDrive
https://onedrive.live.com/?authkey=%21AN7jzq6U%2DbaEYn0&id=160AD1121AA684E7%2197008&cid=160AD1121AA684E7

4. Fisher R., Perkins S., Walker A., Wolfart E. (2003), *Convolution*, Accessed 29.04.2022
https://homepages.inf.ed.ac.uk/rbf/HIPR2/convolve.htm

5. Government of India (2004), *Cracks in Building (Causes and Prevention)*, Accessed 29.04.2022
https://rdso.indianrailways.gov.in/works/uploads/File/Handbook%20on%20Cracks%20in%20building%20(causes%2&%20prevention)(1).pdf

6. Huang W., Zhang N. (2012), *A novel road crack detection and identification method using digital image processing techniques*, 2012 7th International Conference on Computing and Convergence Technology (ICCCT), pp. 397-400
https://ieeexplore.ieee.org/document/6530365

7. Kanan C, Cottrell GW (2012), *Color-to-Grayscale: Does the Method Matter in Image Recognition?*, PLoS ONE 7(1): e29740
https://doi.org/10.1371/journal.pone.0029740

8. Li G., Zhao X., Du K., Ru F., Zhang, Y. (2017), *Recognition and evaluation of bridge cracks with modified active contour model and greedy search-based support vector machine*, *Automation in Construction*, Volume 78, pp. 51–61, ISSN 0926-5805
https://doi.org/10.1016/j.autcon.2017.01.019

9. Li H., Zong J., Nie J., Wu Z., Han H. (2021), *Pavement Crack Detection Algorithm Based on Densely Connected and Deeply Supervised Network*, in IEEE Access, vol. 9, pp. 11835-11842, doi: 10.1109/ACCESS.2021.3050401.
https://ieeexplore.ieee.org/abstract/document/9319182

10. Munawar S., Hammad A., Haddad A., Soares P., Waller T. (2021), *Image-Based Crack Detection Methods: A Review*, Infrastructures 6, no. 8: 115
https://doi.org/10.3390/infrastructures6080115

11. OpenCV, *Contours: Getting Started*, Accessed 02.05.2022
https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html

12. Qu Z., Cao C., Liu L., Zhou D.-Y. (2020), *A Deeply Supervised Convolutional Neural Network for Pavement Crack Detection with Multiscale Feature Fusion*, in IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2021.3062070.
https://ieeexplore.ieee.org/abstract/document/9378802?casa_token=zAAm8uwDHU0AAAAA:cHNbvN8FyHuyX-EJ5bX4vJiNARRya8Imm6MqMEeoYud7CX7JJAbeus7X_3fFYOpUc0fzcF_YXQ

13. Singh K., Anand A., Guruvayurappan S. (2019), *Surface Crack Detection using Computer Vision*, Accessed 26.04.22
https://www.wipro.com/engineeringNXT/surface-crack-detection-using-computer-vision

14. Subirats P., Dumoulin J., Legeay V., Barba D. (2006), *Automation of Pavement Surface Crack Detection using the Continuous Wavelet Transform*, 2006 International Conference on Image Processing, 2006, pp. 3037-3040, doi: 10.1109/ICIP.2006.313007.
https://ieeexplore.ieee.org/document/4107210

15. Tailanián, M., Musé, P., Pardo, Á. (2021) *A Multi-Scale A Contrario method for Unsupervised Image Anomaly Detection*, CoRR, abs/2110.02407
https://doi.org/10.48550/arXiv.2110.02407

16. Tan L., Jiang J. (2019), *Digital Signal Processing (Third Edition)*
https://doi.org/10.1016/C2017-0-02319-4

17. Weisstein, Eric W., *Eigenvalue*, From MathWorld—A Wolfram Web Resource, Accessed 26.04.2022
https://mathworld.wolfram.com/Eigenvalue.html

18. Weisstein, Eric W., *Eigenvector*, From MathWorld—A Wolfram Web Resource, Accessed 26.04.2022
https://mathworld.wolfram.com/Eigenvector.html

19. Zou Q., Cao Y., Li Q., Mao Q., Wang S. (2012), *CrackTree: Automatic crack detection from pavement images*, Pattern Recognition Letters, Volume 33, Issue 3, pp. 227-238, ISSN 0167-8655
https://doi.org/10.1016/j.patrec.2011.11.004

20. Özgenel F. (2019), *Concrete Crack Images for Classification*, Mendeley Data, V2, doi: 10.17632/5y9wdsg2zt.2
https://data.mendeley.com/datasets/5y9wdsg2zt/2