XLIII CILAMCE
21–25, November, 2022
FOZ DO IGUAÇU – PR – BRAZIL

Crack Detection In Concrete Using Artificial Intelligence With Deep Learning

Túlio de Araújo Vieira [1], *Lenildo Santos da Silva* [1], *Leonardo da Silveira Pirillo Inojosa* [1], *Márcio Augusto Roma Buzar* [1]

[1]*Dept. of Civil and Environmental Engineering, University of Brasília,*
Darcy Ribeiro University Campus, CEP 70910-900 , Asa Norte, Brasília-DF, Brazil
tuliovieira@unb.br, lenildo@unb.br, leinojosa@unb.br
[2]*Dept. of Project, University of Brasília*
A Campus Universitário Darcy Ribeiro, CEP 70910-900 , Asa Norte, Brasília-DF, Brazil
buzar@unb.br

Abstract. Artificial Intelligence (AI) is a field that has been drastically changing not only several areas of knowledge, but it also brings high expectations regarding the future of professions. While there are projections of great growth in demand for data scientists, there is the possible threat to unqualified labour, where AI can offer a low-cost alternative. Deep Learning is a subset of Machine Learning, which is a field dedicated to the study and development of machines [1], which can be seen as a stage of AI. Also called Deep Neural Network, refers to Artificial Neural Networks (ANN) with multiple layers. In recent decades, it has been considered one of the most powerfull tool, and has become very popular in literature as it is able to deal with a great amount of data. Interest in having deeper hidden layers began recently to overcome the performance of classical methods in different fields, especially in pattern recognition [1]. Neural networks are used in many areas, such as search algorithms on search engines, content recommendation algorithms, autonomous cars, speech recognition (audio), natural language recognition (text) and computer vision (images). The use in image recognition is mainly done with Convolutional Neural Networks. Thus, the present work intends to apply Convolutional Neural Networks for the detection of cracks in concrete structures through image processing, especially the obtained with drones. The detection of cracks by visual inspection can be a very laborious process, depending on the number of cracks and the difficulty of access, in addition to relying heavily on the subjectivity of the observer. Thus, several methods have been proposed to automate this process, which consist of image processing techniques. However, the implementation of these techniques is difficult when there are adverse conditions, such as changes in different textures [2]. It is in this sense that the use of neural networks brings the expectation of being a method appropriate in relation to the stability in the detection, even considering variations in the conditions for acquiring images, such as lighting, angle of acquisition, texture, dimensions, among others. This expectation is mainly due to the ability to automatically learn the characteristics relevant to the detection of cracks, whereas there are adverse conditions in the learning data.

Keywords: Crack Detection, Concrete, Artificial Intelligence.

1 Introduction

There are dozens of recent studies (between 2016 and 2019) that propose the detection of fissures using fully convolutional networks with semantic segmentation. This recent technique has represented a great advance in the field of detection of pathologies in reinforced concrete, asphalt, among others, because it is capable of defining a contour region of the irregular fissure.

1.1 Conventional Neural Networks

Conventional Neural Networks are formed by multiple perceptrons organized in multiple layers, forming a computational model capable of recognizing patterns based on learning data. Each data point has input information, which is measured numbers for solving a problem. These numbers can be the temperature of the day, the price of a stock on yesterday, or the hit index of a baseball player [1]. Based on this information, the neural network tries to make a prediction, for each prediction, the neural network makes a comparison as a real result, which constitutes the label or "feedback" of each set of input information. Based on this comparison, the neural network adapts to perform increasingly accurate predictions.

1.2 Convolutional Networks

The application of Convolutional Neural Networks (CNNs) has brought great results in the area of pattern recognition, including image and voice processing, autonomous car control and cancer detection. These achievements were the result of the development of new neural network architectures, which are able to perform tasks that were not possible with conventional models. Convolutional networks work with the same principles as a conventional neural network: perceptrons are organized in layers, taking several numerical values as input and generating one or more outputs. Error and activation functions are defined, and learning takes place with backpropagation. What changes is the structure of the network: instead of using fully connected layers, convolutional layers are used, which are connected locally. This drastically reduces the number of parameters required for learning (weights), which makes it possible for networks to learn much faster. Interspersed between convolutional layers, pooling layers are used, and at the end of the network, fully connected layers (dense layers) are used to generate the final output (Figure 1).

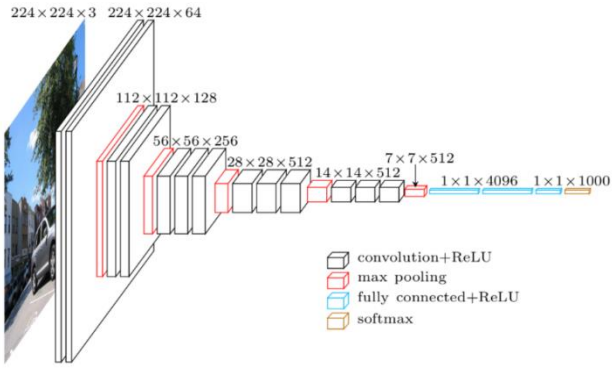

Figure 1. Organization of layers in a convolucinal network (VGG-16 network architecture). Source: Jordan, J. [3]

1.2.1 Tools

The development and training of the convolucinal neural network was carried out with Keras, which is a DeepLearning package for Python. This package, in turn, uses tensorflow to perform machine learning operations, allowing the researcher to worry only about the most important details of the implementation of the neural network (architecture, data, learning parameters). The development environment was the Jupyter Notebook.

Thus, Keraspermite allows you to implement each layer of a neural network with a line of code, explaining the type of layer (dense, convolutional, pooling, or activation layer) and its parameters (number of filters and their dimensions, type of activation). Finally, the model is compiled and several parameters are defined to improve learning, such as stochastic descending gradient, dropout, regularization of weights and momentum. Once the template is ready, simply save and use it for the desired implementation.

2 Building a model for crack detection

2.1 Model Architecture

The detection of the fissure consists of the delimitation of the region occupied by the fissure in the image, according to Figure 2.

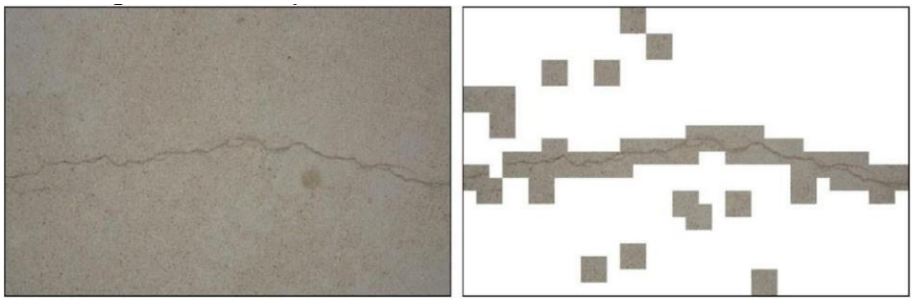

Figure 2. Crack detection using convolutional neural network. Source: Cha et al. [2]

Therefore, we aim at a computational vision of the fissure, which will allow future studies that can automate several processes in the area of study of pathologies, such as determination of crack opening, total length of cracks, total amount of injection material necessary for recovery of a cracked region, etc. To achieve this goal, the detection model will input an image with cracks, which will be divided into small images with dimensions of 227×227 pixels. For each of them, the neural network will make predictions regarding the presence or absence of fissure, which are the two categories under study. These predictions consist of values between 0 and 1, which represent the probability of the analyzed image belonging to the corresponding category (categorical cross entropy). The highest value between these two probabilities is taken as a result. Thus, it will be possible to delimit the region occupied by the fissure in the image, as presented by figures 3 and 4.

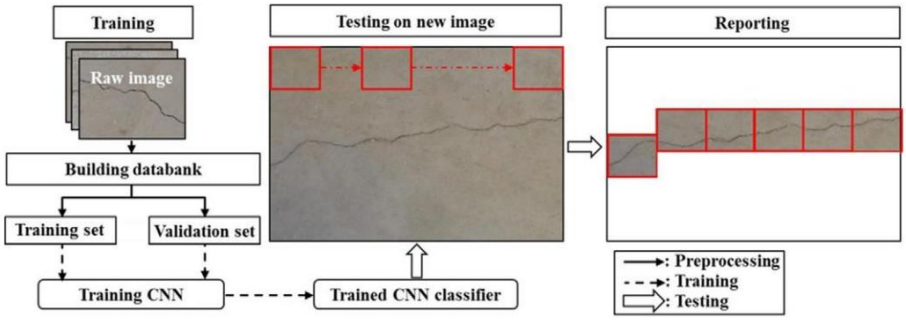

Figure 3. Methodology of the work, proposed by Cha et al. [2]

The architecture to be trained will be proposed by Cha et al. [2], whose layout and layer information are shown in Figure 4, respectively. The only change in this architecture is the input layer dimension of 227×227 instead of 256×256, the dimension used in Cha's study. This change was implemented to adapt the model to the database used, which has images with a resolution of 227×227, thus discarding the need for resizing images during the training process. In addition, the study of Dung, C. V., Anh, L. D. D. [4] reported good results using this resolution, which was another factor in favor of this choice.

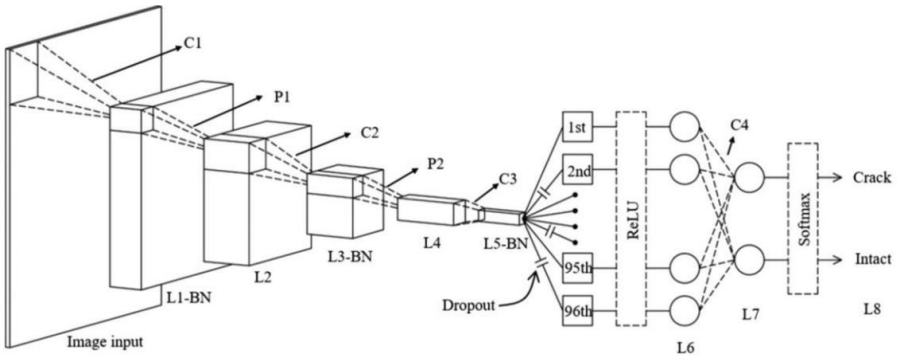

Figure 4. Convolutional network architectureSource: Cha et al. [2]

According to Cha et al. [2], this architecture demonstrated maximum training and validation accuracy of 98.22% in the 51st season and 97.95% in the 49th season, respectively. This metric was measured in 32,000 training images and 8,000 validation images. In addition, the proposed model uses several popular methods in the area of machine learning to avoid overfitting, such as stochastic gradient descent (SGD) and dropout. All of these measures are available for implementation with Keras. In addition to the layers explained in the present work, the model also implements batch normalization layers, which apply a standardization to its input, making its output have an average close to zero and standard deviation close to 1. These layers have the function of accelerating training and reducing the chance of overfitting, as well as enabling processing with values in a controlled range, which is usually the best choice when using numerical/computational methods.

2.2 Database

We used the open-source database created by Özgenel [5], which consists of 40,000 images with a resolution of 227x227 pixels, divided into two directories corresponding to the two labels of interest: "Positive", relative to the presence of fissure, and "Negative", relative to absence. Each label has the same amount of images (20,000 images contain fissure and 20,000 images do not).

Then, a Python script was written to divide the database into training, validation, and test sets at a 70%:15%:15% ratio at random. Each of these sets has the same number of images for both labels (with and without fissure).

It is important to note that the database used has more open fissures, different from the database used by Cha et al. [2], formed by images with more closed fissures. However, the use of training images with open fissures was also promising for the classification task, as shown in the results.

2.3 Preprocessing

Preprocessing training, validation, and testing images consists of two steps:

The first is abnormalization of pixel values so that they vary in the range from 0 to 1. Because each pixel has an intensity value from 0 to 255, just divide these values by 255. This normalization is quite common when working with images, and serves for weights to change evenly throughout learning.

The second step is to upload the images. When using large databases, it is impossible to load all training data at once because there is a RAM limit. Thus, it is necessary to load the images from the disk as the model is trained.

Therefore, the pre-processing of the images occurs as illustrated in Table 1.

Table 1. Pseudo-code of image preprocessing

A cada época de treinamento, repetir `steps_per_epoch` vezes:
Carregar `batch_size` imagens de treinamento do disco, aleatoriamente
Aplicar normalização
Realizar treinamento do modelo

The maximum value of *batch_size* depends on the amount of RAM available. Knowing that each image will result in an array of 3x227x227 dimensions (3 color channels, times 227 pixels tall, times 227 pixels wide), and that each element of this array will occupy 4 bytes float of 4x8 = 32 bits), it is estimated that each image will occupy 600 KB. Thus, a batch of 1024 images would take up 600 MB of memory, for example. To ensure that at each time the model has the chance to use all training images, it is usual to set the number of steps per season (steps_per_epoch) as the size of the training set divided by the size of the batch.

In the development of the model, we used *batch_size* = 1024, which results in values from *steps_per_epoch* equal to 28, 6 and 6 for the training, validation and test sets, respectively.

2.4 Training, Validation and Testing Results

The model was trained using training accuracy as a metric. The graph in Figure 5 shows the evolution of training and validation accuracy over 50 seasons.

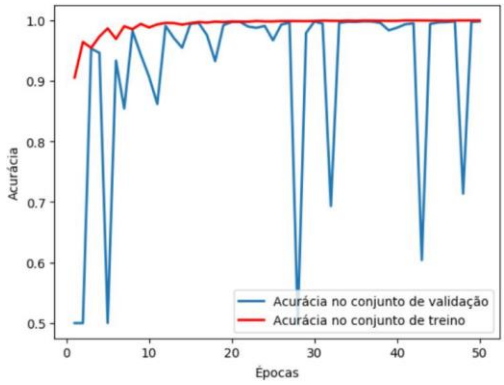

Figure 5. Evolution of training and validation accuracy

As can be seen from the chart above, the model reached great accuracy in a few seasons. This is probably due to the fact that the presence or not of fissures is relatively easy to be learned by the neural network used, because neural networks of similar size are able to deal with classification problems with many more categories and complexity of characteristics.

It can also be noted that the accuracy in the validation set fluctuated greatly at certain times. In some cases, this may be a sign of overfitting, characterizing a situation in which the model learns the training set very well, but does not generalize the learning to get it right in the validation set, which causes an oscillation in the accuracy of validation that tends to randomness.

However, in the present study, this oscillation in the accuracy of validation can be attributed to the fact that, throughout learning, the model was varying between several optimal solutions, some of which did not generalize well for the validation set. One solution to this problem would be to implement regularization of weights, or increase the ratio of dropout layers.

In any case, the model was able to achieve an optimal solution for both test and validation sets, with accuracy of 99.95% and 99.78%, respectively. The accuracy of the test at the end of the seasons was 99.8%.

However, the high validation and testing accuracy did not correspond to a good performance of the model in all situations, as explained in section 2.6. It is suspected that the database did not have a good variety in terms of crack opening and contrast. Thus, one should not take these metrics as good indicators of the generalization capacity of the model.

Next , the tests on images taken by mobile phone and found on the Internet are presented. An application has been developed that uses the trained model to detect cracks in photos of varied resolutions. Its development, as well as its limitations, are discussed in the following items.

2.5 Application Development

As an example of neural network application, an application that implements the model in a simple way was designed.

The application works as follows: the user chooses a photo of a concrete surface with cracks. Then chooses the size of the sliding window that will scroll through the photo (Figure 6, left).

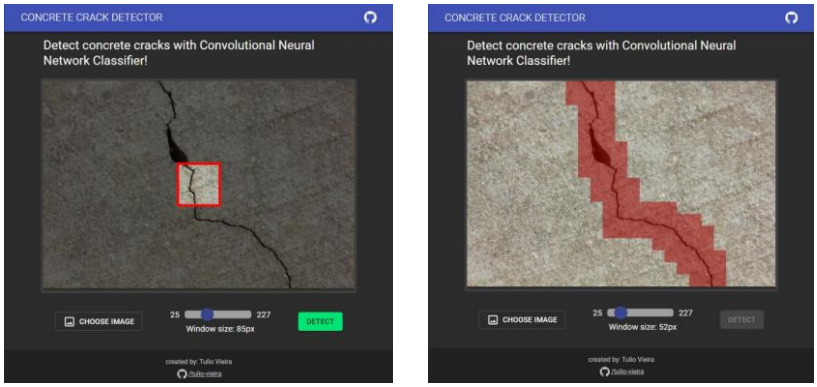

Figure 6. Demo of the developed application: settings and result. Authors' Images

This sliding window traverses the image using a step equal to half its dimension. In each position, the model makes a prediction regarding the absence or presence of fissure, and the results are presented graphically in order to form a region of detection of the fissure.

It is important to note that the window step was used to allow detections to overlap each other, which prevents a fissure from being found only in the corner of an image. This type of situation can make detection difficult, even if the model has been trained with an image bank that contains figures with cracks in the edges.

To ensure that the model works with images of various resolutions and window sizes, the window-generated image sections are resized to the resolution of 227×227 pixels, which is the input dimension of the neural network. The detection result is shown in Figure 6 on the right.

This app was developed to work in the browser, using tensorflowjs. This technology made it possible to convert the model developed in Keras to a model that can be used in javascript, as well as the implementation of the preprocessing steps, including the sliding window, in this language. However, bugs of tensorflowjs were found when we tried to use the application on less powerful phones and machines, as this technology is still new and needs further development.

The application can be accessed at this link:https://tulio-vieira.github.io/concrete-crack-detector-app. It is recommended to use the Google Chrome browser for use.

2.6 Limitations

The developed application does a good job to portray the situations in which the model has good performance or not. Basically, the more images generated by the sliding window are similar to training images, the better the detection. Thus, the model brings the best results in crack photos that have a sharp contrast between the fissure and intact surface, with a choice of window dimension that depicts a crack opening similar to that shown in Figure 7.

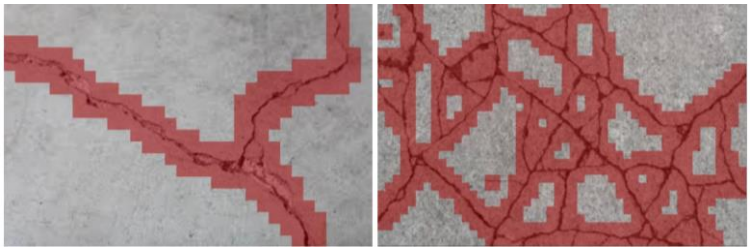

Figure 7. Crack detection under optimal conditions

Situations in which the concrete surface presents other artifacts, such as corrosion and stains or contrast that was not high enough for good detection, do not result in good results, because the model was not trained for such situations. An example can be seen in Figure 8.

Figure 8. False positives due to stains

Finally, it is recommended that the figure have an acceptable resolution, in order to generate appropriate clippings with dimensions close to 227 pixels.

3 Conclusions

The present work aimed to demonstrate a simple application of neural network (deep learning) for cleft detection, focusing on the classification of image clippings to generate a detection region. Using a relatively simple architecture, it was possible to generate good results, as long as the ideal detection conditions are present. These results can still be improved with the use of more complete databases, which contain cracks in stained surfaces and with less contrast, which can also be used in conjunction with open-source databases. In addition, it was possible to demonstrate the use of tensorflowjs, which made possible the development of a crack detection application in the browser. This technology greatly facilitated production, as it made it possible to automatically convert the trained model to the javascript environment. At the present time, the tensorflow development team has been releasing several versions of the technology, including versions for micro-controllers and other languages, which is further consolidating its position as the production platform for machine learning models. It is important to emphasize, however, that the present work focuses only on one of the tasks relevant to detection, which is the classification of the image. To achieve more sophisticated detections, it is necessary to implement models capable of performing the semantic localization and segmentation of the cracks. These tasks include not only the implementation of neural networks with complex architectures, but also advanced algorithms that work together with those networks. In the midst of this complexity, several authors have proposed several alternatives, with the most recent articles dating from this year of writing. The state of the art in the area of crack detection appears to be in the research of Young-Jin Cha [6], who developed an algorithm capable of detecting cracks at pixel level in real time, using an auto-encoder architecture.

Although the present work does not follow the latest developments in the area, it still presents the basic concepts of computer vision and machine learning, which are the basis for the development of more complex models and algorithms. It was also possible to portray issues related to the production of a machine learning model, whose code and implementation details can be found in the github repository:https://github.com/tulio-vieira/concrete-crack-detector-app. The notebooks used for training, in turn, can be found here: https://github.com/tulio-vieira/concrete-crack-detector-train.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] TRASK, A. (2019). Grokking Deep Learning. Manning Publications, Shelter Island, NY, 301p.
[2] CHA, Y.-J., CHOI, W., & BÜYÜKÖZTÜRK, O. (2017). Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. Computer-Aided Civil and Infrastructure Engineering, 32(5), 361–378.

[3] JORDAN, J. (2018). Common architectures in convolutional neural networks. Available in: https://www.jeremyjordan.me/convnet-architectures/. Accessed 01/12/2019.
[4] DUNG, C. V., ANH, L. D. (2018). Autonomous Concrete Crack Detection Using Deep Fully Convolutional Neural Network. Automation in Construction 99 (2019) 52–58.
[5] ÖZGENEL, Ç. F. (2018). Concrete Crack Images for Classification.Mendeley Data, v1. Available in: https://data.mendeley.com/datasets/5y9wdsg2zt/1. Accessed 03/12/2019.
[6] CHOI, W., CHA, Y.J. (2020). SDDNet: Real-time crack segmentation. IEEE Transections on Industrial Electronics, (IF: 7.515), 67(9), 8016-8025. Disponível em: https://ieeexplore.ieee.org/abstract/document/8863123. Acesso em 14/12/2020.