**Crack detection – technical report**

*Problem presentation*

With the increasing urban infrastructure, there is now more ground to cover than ever, so on-site human inspection is becoming harder and harder, current practice being time consuming, tedious, and subjective, involving human technicians reviewing the inspection videos and identifying cracks. The early detection of these cracks is paramount, as they degrade very quickly and form some nasty pot-holes.

We are in need of smart solutions, devices that are able to travel long distances (e.g.: drones) and detect cracks and imperfections in concrete roads, buildings, pipes etc.

Some big advantages of these solutions are maintenance cost reduction and the ability to inspect hard to reach places, like under bridges and tall buildings. These could be controlled remotely, massively reducing the risk of human injury when inspecting cracks.

*State-of-the-art*

The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. CNNs are multilayer perceptrons which have been regularized. Usually, MLPs imply fully connected networks, meaning that each neuron in a layer is connected to every single neuron in the next layer. This brings a disadvantage to MLP though: overfitting. This is why CNNs use convoluted layers.

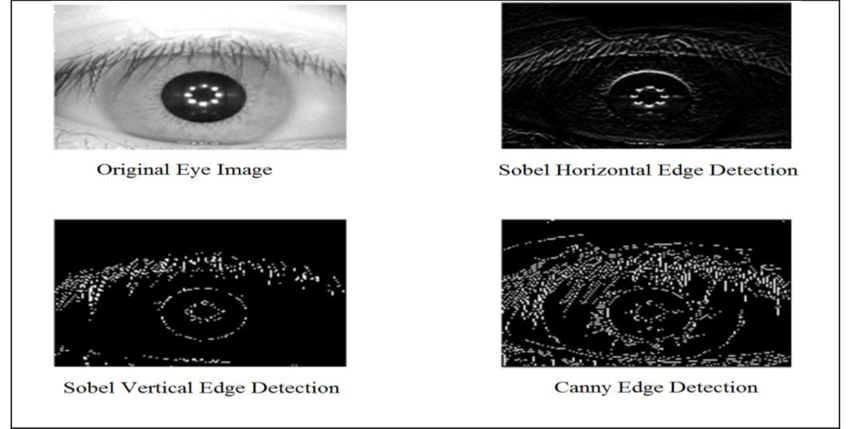
Convolutional neural networks, or CNNs, are a class of deep neural networks that are most commonly used for analyzing images. The difference of CNNs from other neural networks comes from a different approach to regularization: a CNN tries to assemble difficult patterns using smaller and simpler patterns. This is done by introducing convolution and pooling layers to the NN, which reduce the spatial size of the input array (image). Typically, neural networks use sigmoidal functions, but V. Nair and G.E. Hinton in 2010[[1]](#footnote-1) introduced a more efficient activation function called – Rectified Linear Unit (ReLU).

There are massive advantages to convolutional NNs:

* Boost in computation efficiency: in AlexNet (the name of a CNN, designed by Alex Krizhevsky in 2012) the convolutional layers comprised of 90% of the weights (~representational capacity) but contributed only to 10% of the computation; and the remaining (10% weights => less representation power, 90% computation) was eaten up by fully connected layers.
* Spatial information in images is retained more than in fully connected layers, so real world, unconstrained image segmentation has better results.
* It runs very well on all kinds of problems, even 1D problems like time series, and 3D image classification, because they have the same structure where we would like location invariant features. Even so, a CNN would be useless for totally unstructured data, like in a spreadsheet.

*Our solution*

Traditional methods of Image Processing Techniques like Canny and Sobel edge detection are prone to inefficiency because of lighting and shadow changes in varying real-life situations. Detecting these cracks is a challenging task since they are tiny, and noisy patterns exist on the components’ surfaces.



Neural Networks do better when it comes to these problems, being capable of learning image features automatically, with high effectiveness and robustness. They excel in eliminating the noisy features out of the resulting model, because they learn by the bulk of the images, and noisy images are usually outliers, so they will be mostly ignored.

Training of the network can be done beforehand (lengthy) and predicting cracks can be done quickly using a saved model which contains the features. The outstanding advantage of the proposed NN-based crack detection is that it spares multifarious work from features pre-extraction and calculation compared to traditional methods. Besides, the CNN needs not to convert the format of input images, but automatically learns crack features from images, which reduces workload of crack detection.

***Dataset***

The dataset[[2]](#footnote-2) contains concrete images having cracks. The data is collected from various METU Campus Buildings. The dataset is divided into two as negative and positive crack images for image classification.

Each class has 20000images with a total of 40000 images with 227 x 227 pixels with RGB channels, but we transformed them to grayscale, because colors are not really needed. The dataset is generated from 458 high-resolution images (4032x3024 pixel) with the method proposed by Zhang et al (2016)[[3]](#footnote-3). High-resolution images have variance in terms of surface finish and illumination conditions.

We updated the dataset by adding a couple hundred images of other cracks, in bathrooms, streets etc. Data augmentation is applied when training, done automatically at the start of each epoch for each image, so they are not stored on disk, which is a very good thing because we save space this way.

***Implementation details***

The API we used for training the NN is Keras, which is based on Tensorflow. The model is that of a convolutional NN, with max-pooling and dropout layers in between (a limitation of convolutional layers is that they record the precise position of features in the input).

The result of using a pooling layer and creating down sampled or pooled feature maps is a summarized version of the features detected in the input.

Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.

For the development of the front-end we used the Tkinter library that provides easy implementation for creating windows, buttons, input fields and make accessible the interaction with the back-end.

The application is structured into different windows, one as the main window in which the user selects an action (login/register), one in which the user inputs its credentials and one in which the user can either test or train the neural network.

The front-end also interacts with the database in order to store, retrieve and verify user information, also checking the type of user that has logged in order to determine his permissions.

The passwords are hashed using md5 for security purposes.

There are two types of users:

1. Normal users are able to be registered into the database by anyone using the application, they are able to upload an image to the program which will be tested using our trained model for the neural network in order to be shown a rate of the presence of cracks, are able to upload a folder in which every image will be tested (other types of files being ignored) and are also able to download the data provided by the neural network (layers, input/output shape, tested images, predictions, number of images with/without cracks) in either csv, xml of json formats.
2. Admin users are able to retrain the network after modifying the dataset, and they can also set the number of batches and epochs.

|  |  |
| --- | --- |
|  |  |

*Comparison with other solutions*

While other solutions do tend to be of more complexity regarding the network architecture and segmentation of images, they are also being evaluated on more complex datasets, and thus obtaining a lower precision overall. Some of these solutions include:

* Jenkins, Mark David, et al. "A Deep Convolutional Neural Network for Semantic Pixel-Wise Segmentation of Road and Pavement Surface Cracks." 2018 26th European Signal Processing Conference (EUSIPCO). IEEE, 2018. [1]

**Model used**: U-Net architecture with a final softmax layer that trains patches of the images from the original set in order to eliminate the useless background information before training.

**Results**: Image patches of size 48x48 are utilized and a total of 2000 random patches are extracted from each of the training images. The algorithm is evaluated on the CrackForest Dataset made of images of size 480x320. The network is trained on 100 of these images split into 80 training and 20 validation images. The implementation was carried out using Keras and Tensorflow and training was carried out over 100 epochs with a batch size of 34. Training takes approximately 3 hours and inference on the pre-processed test set of 18 images takes approximately 3 seconds. Precision: 92.46%

* Mosinska, Agata, et al. "Beyond the pixel-wise loss for topology-aware delineation." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018. [2]

**Model used**: A combination of the U-Net model and the VGG19 network in order to better determine possible errors that may appear on feature detection on the image.

**Results**: Images of cracks in road re split into 104 for training and 20 test images. The multiple shadows and cluttered background make their detection a challenging task. Patches of 450x450 pixels are used for training. The data is augmented mirroring and rotating the training images by 90◦, 180◦ and 270◦. Batch normalization is used for faster convergence and the Adam optimizer. Precision: 94.3%

* Escalona, Uriel, et al. "Fully Convolutional Networks for Automatic Pavement Crack Segmentation." Computación y Sistemas 23.2 (2019): 451-460. [3]

**Model used**: Three U-Net models, with 23, 11 and 7 convolutional layers in order to determine which one gives better results.

**Results**: Using the CFD dataset, made of images of size 480x320x3 (each having 3 different illuminations/shades) and the AigleRN dataset, made of 991x462 grey scale images, 100 images were used for training and 18 for testing. Precisions: 93.45%, 94.28%, 82.42%

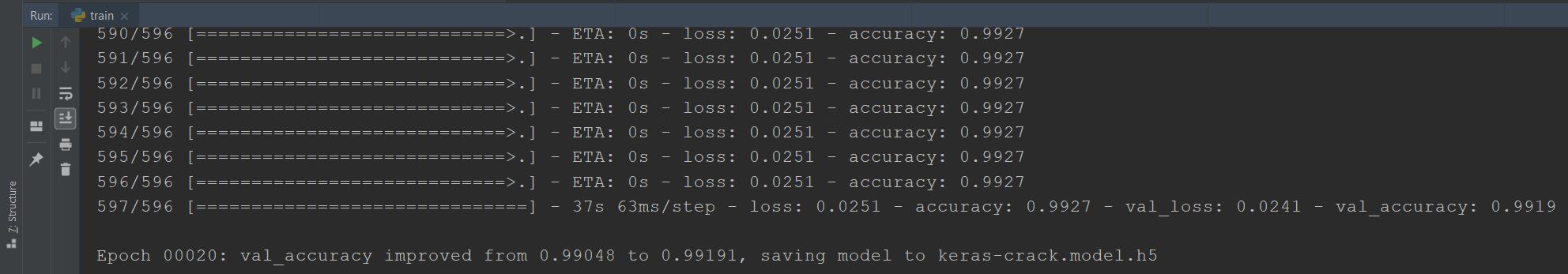
* Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *International Conference on Medical image computing and computer-assisted intervention*. Springer, Cham, 2015. [4]

**Model used**: U-Net on biomedical image segmentation

**Results**: Using 23 convolutional layers and augumented data by generating alternative images with deformations from a set of 30 512x512 images for training. Precision: 92.03%

*Results, evaluation*

Training the neural network on the given datasets (concrete crack images, crackconcrete, cracktile) we were able to obtain an accuracy of over 99% on the validation data, which is a subset of the training data (1000 images for each class separated from the training data, in which, 19000 images remain out of the 20000).



The test image is evaluated by being in the first place resized to a 128x128 resolution to which a chunk containing its data will be checked with the trained neural network model.

We verified and altered our implementation by using an AOP and MOP components and performing unit testing to check our connection to the database and the verification for the image testing algorithm.

For non-functional testing we used pgbench, a tool provided by PostGRE, to load test our database. For the laptop used, it turned out that a maximum 70 users at a time could be served. Any more would produce a fatal error.

A screen shot of a social media post

Description generated with very high confidence

To see the performance we used pgAdmin, the default monitoring tool provided by PostGRE.

A screenshot of a cell phone

Description generated with very high confidence

*Future work*

Regarding the possibilities to extend the application we can upgrade the interface to be more intuitive for the users, extend on the backend processing to include segmentation of images and be able to identify the segments that interests us, and upgrade the neural network to a more specific model that can better process more complicated images. Also, the app could also include a depth detection module or a crack outlining part.

*Conclusions*

Manual inspection of cracks or defects in concrete civilian buildings, bridges or dams is costly and time consuming. That is why there is interest in fully unmanned inspection of various kinds, such as detection of cracks or other defects in concrete walls/other structures.

With the advancement of autonomous devices, one could also install this crack detection module into an autonomous drone/robot that would examine various roads, structures, underground pipes, bridges, building infrastructures and so on. This can be even combined with a sorting algorithm, which would classify cracks and put a timestamp and location on the classification so we could have a hierarchy to follow.

*Bibliography*

[1] <https://www.eurasip.org/Proceedings/Eusipco/Eusipco2018/papers/1570437180.pdf>

[2] <http://openaccess.thecvf.com/content_cvpr_2018/papers/Mosinska_Beyond_the_Pixel-Wise_CVPR_2018_paper.pdf>

[3] <https://www.cys.cic.ipn.mx/ojs/index.php/CyS/article/viewFile/3047/2625>

[4] [https://arxiv.org/pdf/1505.04597.pdf)%e5%92%8c%5bTiramisu%5d(https://arxiv.org/abs/1611.09326.pdf](https://arxiv.org/pdf/1505.04597.pdf)%e5%92%8c%5bTiramisu%5d(https:/arxiv.org/abs/1611.09326.pdf)

1. <https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf> [↑](#footnote-ref-1)
2. 2018 – Özgenel, Ç.F., Gönenç Sorguç, A. “Performance Comparison of Pretrained Convolutional Neural Networks on Crack Detection in Buildings”, ISARC 2018, Berlin. [↑](#footnote-ref-2)
3. Lei Zhang , Fan Yang , Yimin Daniel Zhang, and Y. J. Z., Zhang, L., Yang, F., Zhang, Y. D., & Zhu, Y. J. (2016). Road Crack Detection Using Deep Convolutional Neural Network. In 2016 IEEE International Conference on Image Processing (ICIP). http://doi.org/10.1109/ICIP.2016.7533052 [↑](#footnote-ref-3)