

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communications Technology

Group Project Report

## **Surface Crack Classification**

Subject: Computer Vision

Group 5:

Pham Van Khoa - 20176796

Tran Hai Son - 20176861

Tran Le Hoang - 20176764

Vu Hai Dang - 20176711

Hoang Tuan Anh Van - 20170224

*Hanoi, December 2021*

## Table of contents

I. INTRODUCTION .....	3
1. Abstract .....	3
2. The Dataset .....	3
II. UNDERSTAND THE DATASET .....	3
1. Data distribution .....	3
2. Some data images .....	4
III. DIFFERENT APPROACHING.....	5
1. Convolutional Neural Network.....	5
1.1. Introduction .....	5
1.2. A Simple Convolutional Neural Network Architecture.....	6
1.3. Initializing parameters .....	8
1.4. Experiment results.....	8
1.5. Conclusion .....	10
2. VGG-16 .....	10
2.1. Introduction .....	10
2.2. VGG-16 architecture.....	11
2.3. Experiment results.....	13
2.3. Conclusion: .....	14
3. Image processing.....	14
3.1. Proof of concept with Image Processing.....	14
3.2. Experiment results.....	19
3.3. Conclusion: .....	20
IV. CONTRIBUTION .....	21
V. REFERENCES.....	21

# I. INTRODUCTION

## 1. Abstract

Concrete surface cracks are major defect in civil structures. Building Inspection which is done for the evaluation of rigidity and tensile strength of the building. Crack detection plays a key role in the building inspection, finding the cracks and determining the building health.

## 2. The Dataset

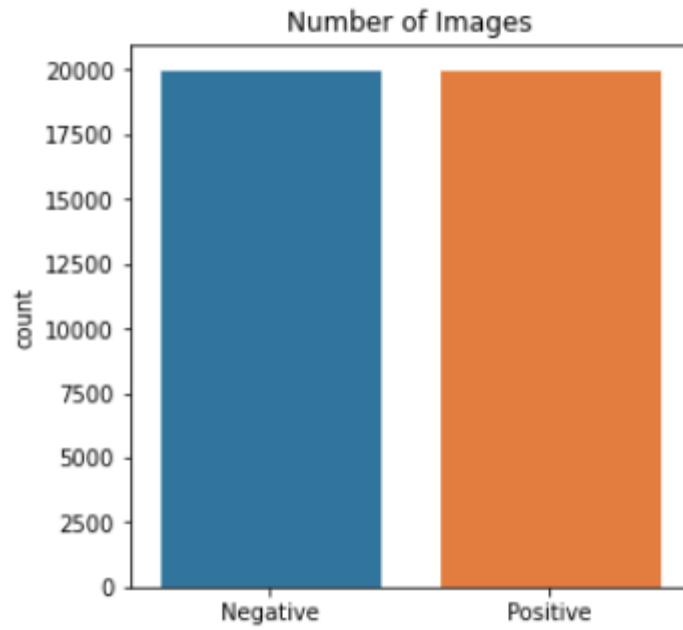
The dataset contains images of various concrete surfaces with and without crack. The image data are divided into two as negative (without crack) and positive (with crack) in separate folder for image classification. Each class has 20000 images with a total of 40000 images with 227 x 227 pixels with RGB channels. The dataset is generated from 458 high-resolution images (4032x3024 pixel) with the method proposed by Zhang et al (2016). High resolution images found out to have high variance in terms of surface finish and illumination condition. No data augmentation in terms of random rotation or flipping or tilting is applied.

# II. UNDERSTAND THE DATASET







(Data exploring and visualization)

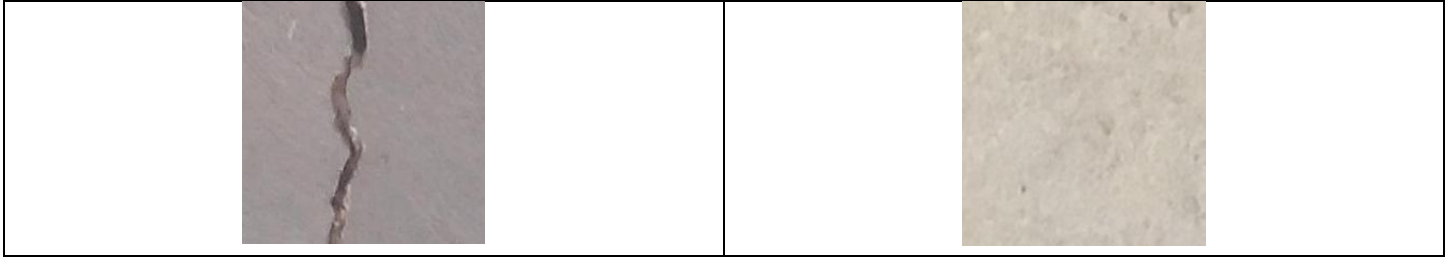
## 1. Data distribution

From the chart described below, we can conclude that this dataset is well-balanced between its two classes. Each class has 20.000 images with the total of 40.000 images for both classes. A well-balanced dataset like this would be very ideal to feed on deep learning neural networks.



## 2. Some data images

Positive			Negative		
					
					
					



From the above table that provides some images of both classes, only by naked eyes, we can see that images from these two classes are easily distinguishable. The crack from positive images are pretty clear, most of them have darker color with clear borders and edges, hence, this dataset problem may be well cracked with traditional image processing approach.

### III. DIFFERENT APPROACHING

(Simple Convolutional Neural Network, Image Processing)

#### 1. Convolutional Neural Network

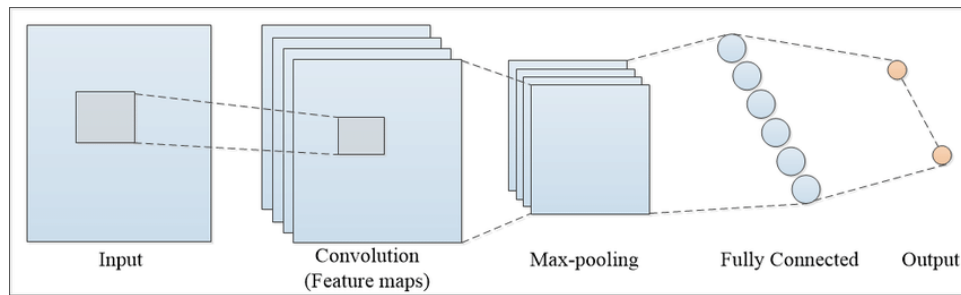
##### 1.1. Introduction

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network, most applied to analyze visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are only equivariant, as opposed to invariant, to translation. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

## 1.2. A Simple Convolutional Neural Network Architecture

### Architecture of a convolutional neural network with 3 layers:



### Main components:

- **Convolutional Layer:** Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.
- **Pooling Layer:** Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains valuable information. Spatial pooling can be of several types: Max Pooling, Average Pooling, Sum Pooling.
- **Fully Connected Layer (Dense):** Feeds all outputs from the previous layer to all its neurons, each neuron providing one output to the next layer.

### Our 11 layers convolutional neural network:

```

Model: "sequential"
-----
Layer (type)                 Output Shape              Param #
-----
conv2d (Conv2D)              (None, 120, 120, 64)     640
-----
max_pooling2d (MaxPooling2D) (None, 60, 60, 64)       0
-----
conv2d_1 (Conv2D)            (None, 60, 60, 64)       36928
-----
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)       0
-----
conv2d_2 (Conv2D)            (None, 30, 30, 128)     73856
-----
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 128)     0
-----
flatten (Flatten)            (None, 28800)            0
-----
dense (Dense)                (None, 256)              7373056
-----
dropout (Dropout)            (None, 256)              0
-----
batch_normalization (BatchNo (None, 256)              1024
-----
dense_1 (Dense)              (None, 2)                514
=====
Total params: 7,486,018
Trainable params: 7,485,506
Non-trainable params: 512

```

With our dataset, original images have the size of (227,227). To reduce computational cost and speed up training time, we will resize all images to size (120,120). After resized, we create our network by sequentially stack 3 pairs of convolutional and max-pooling layers.

- First conv2D: Filter with 64 kernels, each kernel with size (3,3), activation: relu.
- Second conv2D: Filter with 64 kernels, each kernel with size (3,3), activation: relu.
- Third conv2D: Filter with 128 kernels, each kernel with size (3,3), activation: relu.

We choose kernel of size (3,3) since our input images now has the size of only (120,120), we did not use (5,5) kernel since it requires much more parameter and time to learn, for our data, (3,3) kernel can guarantee our goal.

The number of kernels in each layer of a convolutional neural network is not arbitrary, it can be chosen either intuitively or empirically. Our dataset is in fact an “easy” dataset so we do not need

to add large number of kernels for each filter. After some training, we chose 64,64 and 128 as our number of kernels for layer conv2D 1 to 3, respectively.

For this network, besides essential layers like convolutional layer, dense layer and max pooling layer, we also add two additional layers:

- Drop Out: Dropout is a regularization method that during training phase, some number of layer outputs are randomly ignored or dropped out. This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on responsibility for the inputs. This helps in the case of overfitting.
- Batch Normalization: Batch Normalization is used to standardizes the inputs to a layer of each mini batch. This helps stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

### 1.3. Initializing parameters

For optimizer, we use Adam since it benefits from both Momentum and RMSprop. With momentum, this optimizer can help the model to overcome local minimum. On the other hand, with RMSprop, our model now has “friction” this help reduce the time our model needed to reach global minimum.

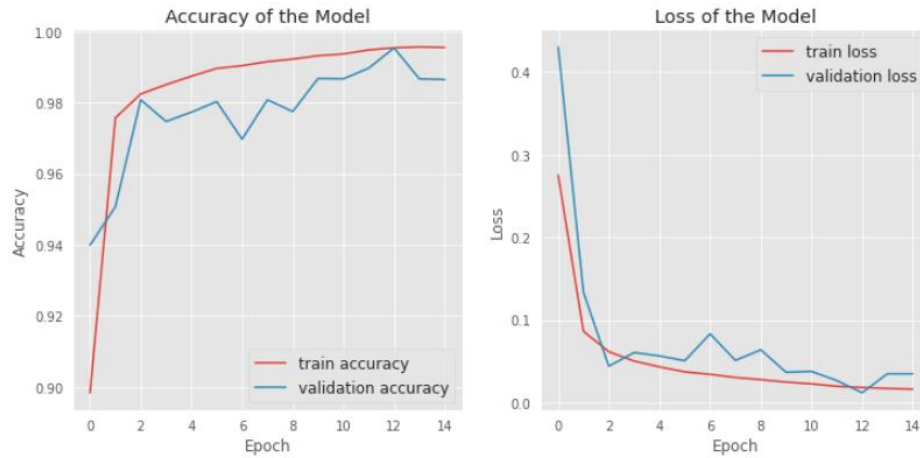
Optimizer: Adam; Learning rate: 0.00001; Loss: Sparse Categorical Cross entropy.

### 1.4. Experiment results

The dataset is splitted into 3 subsets which are: Train subset, Validation subset and Test subset with the corresponding ratio of 75/20/5 or 30000/8000/2000.

Without data augmentation, number of epochs: 15





	precision	recall	f1-score	support
Negative	0.99	1.00	1.00	1001
Positive	1.00	0.99	1.00	1001
accuracy			1.00	2002
macro avg	1.00	1.00	1.00	2002
weighted avg	1.00	1.00	1.00	2002
[[1000 1]				
[ 7 994]]				

With data augmentation including: shifting, zoom, rotation and flip;

Number of epochs: 200:



Found 2002 images belonging to 2 classes.					
	precision	recall	f1-score	support	
Negative	0.99	0.98	0.99	1001	
Positive	0.98	1.00	0.99	1001	
accuracy			0.99	2002	
macro avg	0.99	0.99	0.99	2002	
weighted avg	0.99	0.99	0.99	2002	
[[982 19]					
[ 5 996]]					

## 1.5. Conclusion

The result without augmentation is slightly better than one with augmentation. However, both ways still perform outstandingly with the F1-score of 99%.

The model with data augmentation might require more time and epochs to learn since the number data samples is significantly increased.

## 2. VGG-16

### 2.1. Introduction

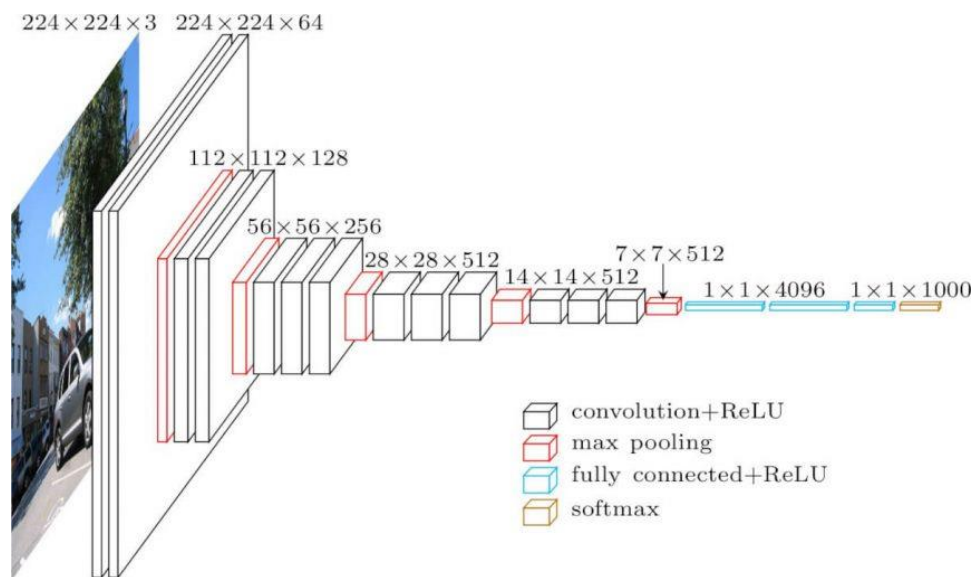
First proposed by K.Sumonyan and A.Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”, VGG-16 has been a simple yet widely used Convolutional Neural Network (CNN) Architecture

It was one of the renowned cnn models submitted to (ILSVRC) in the year 2014. VGG-16 made an improvement over AlexNet architecture by replacing 11 and 5 kernel-sized filters in the first and second convolutional layer, correspondingly, with multiple 3x3 kernel-sized filters one after another.

VGG16 is utilized in a variety of deep learning image classification techniques and is famous because of its ease of implementation.

## 2.2. VGG-16 architecture

### The original VGG-16:



The input to conv1 layer is of fixed size  $244 \times 244$  RGB image, which means that the number of channels is 3, making a tensor of  $(244, 244, 3)$

The tensor is then passed through a stack of convolutional (conv.) layers, where the kernels were used with a very small receptive field  $3 \times 3$ . More specifically, the first two layers are convolutional layers with 64  $3 \times 3$  filters each, using same padding hence a  $224 \times 224 \times 64$  volume. All hidden layers are equipped with the rectification (ReLU) non-linearity.

Then, a pooling layer is used which will reduce height and width of a tensor: from  $(224, 224, 64)$  to  $(112, 112, 64)$ . Then a couple of other conv layers are implemented, together with max-pooling layers.

The three last fully-connected layers contain two 4096 channels layers and a third layer contains 1000 channels for ILSVRC classification (one for each class). The final layer is the softmax layer.

In our specific project, transfer learning technique with pre-trained VGG16 model was utilized. A pre-trained model has been previously trained on a dataset and contains the weights and biases that represent the features of whichever dataset it was trained on. Learned features are often transferable to different data. For example, a model trained on a large dataset of bird images will contain learned features like edges or horizontal lines that you would be transferable your dataset. A summary of our model architecture are described as following:

In summary, we used pre-trained convnet from VGG-16 model and add 4 more layers:

- Flatten
- Dense with 256 node and 'relu' activation
- Dropout with the rate of 0.4
- Softmax layer with 2 node

```
Model: "model_3"
```

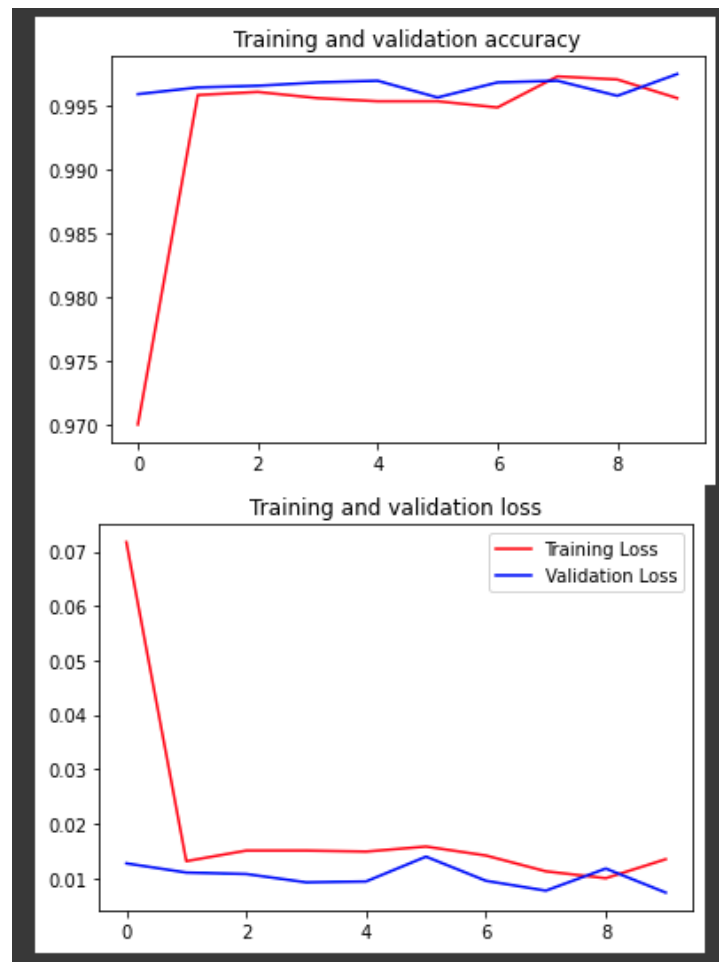
Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 128, 128, 3)	0
block1_conv1 (Conv2D)	(None, 128, 128, 64)	1792
block1_conv2 (Conv2D)	(None, 128, 128, 64)	36928
block1_pool (MaxPooling2D)	(None, 64, 64, 64)	0
block2_conv1 (Conv2D)	(None, 64, 64, 128)	73856
block2_conv2 (Conv2D)	(None, 64, 64, 128)	147584
block2_pool (MaxPooling2D)	(None, 32, 32, 128)	0
block3_conv1 (Conv2D)	(None, 32, 32, 256)	295168
block3_conv2 (Conv2D)	(None, 32, 32, 256)	590080
block3_conv3 (Conv2D)	(None, 32, 32, 256)	590080
block3_pool (MaxPooling2D)	(None, 16, 16, 256)	0
block4_conv1 (Conv2D)	(None, 16, 16, 512)	1180160
block4_conv2 (Conv2D)	(None, 16, 16, 512)	2359808
block4_conv3 (Conv2D)	(None, 16, 16, 512)	2359808
block4_pool (MaxPooling2D)	(None, 8, 8, 512)	0
block5_conv1 (Conv2D)	(None, 8, 8, 512)	2359808
block5_conv2 (Conv2D)	(None, 8, 8, 512)	2359808
block5_conv3 (Conv2D)	(None, 8, 8, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten_4 (Flatten)	(None, 8192)	0
dense_8 (Dense)	(None, 256)	2097408
dropout_4 (Dropout)	(None, 256)	0
dense_9 (Dense)	(None, 2)	514

```
=====
Total params: 16,812,610
Trainable params: 2,097,922
Non-trainable params: 14,714,688
```

### 2.3. Experiment results

The dataset is split into 3 subsets which are: Train subset, Validation subset and Test subset with the corresponding ratio of 75/20/5 or 30000/8000/2000.

Number of epochs is 10, and the result was already good at very first epochs and stay stable until the end.



	precision	recall	f1-score	support
Negative	1.00	1.00	1.00	1001
Positive	1.00	1.00	1.00	1001
accuracy			1.00	2002
macro avg	1.00	1.00	1.00	2002
weighted avg	1.00	1.00	1.00	2002
[[ 999    2]				
[    1 1000]]				

## 2.3. Conclusion:

The model predicted correctly almost all test images.

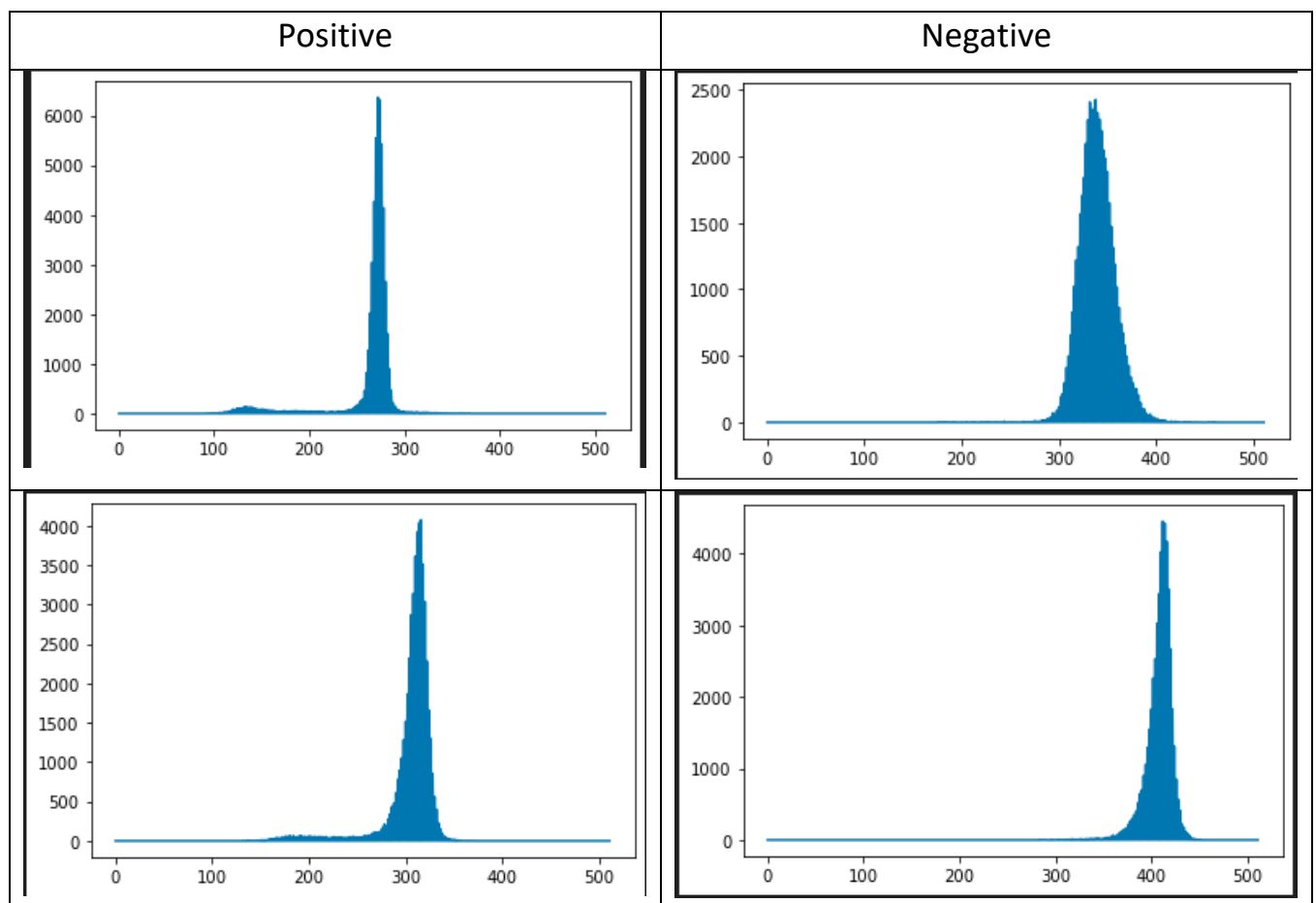
This result is slightly better than CNN stated above, but all metrics is almost 1 so it is hard to say which one is better.

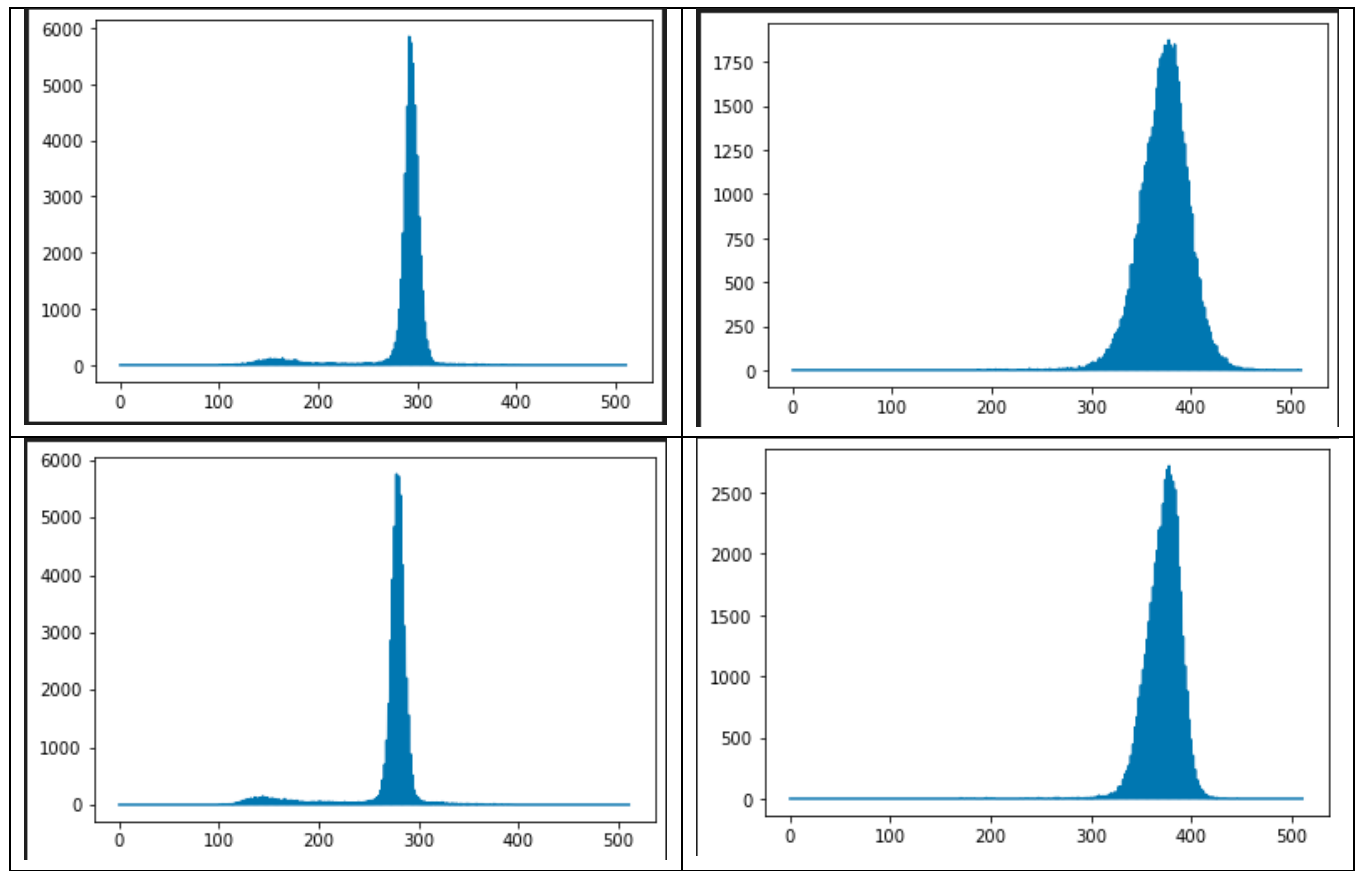
## 3. Image processing

### 3.1. Proof of concept with Image Processing

From the table that demonstrates several sample data in section II.2, we dive deeper into the dataset to have a conclusion that for every image that are labelled as Positive, the crack region has darker color and they take up a noticeable amount of space in the image. With this information, we decided to take advantages over the differential between pixel value of two regions: Region that contains the crack and the region that does not.


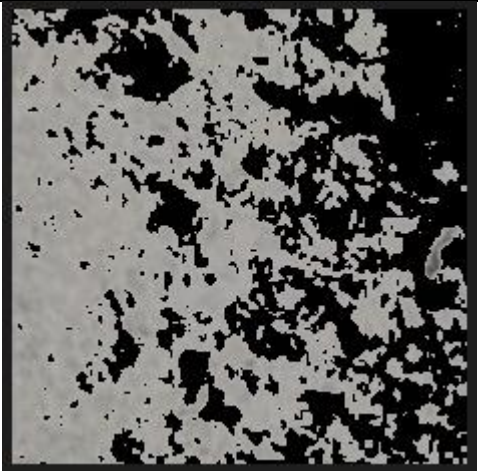


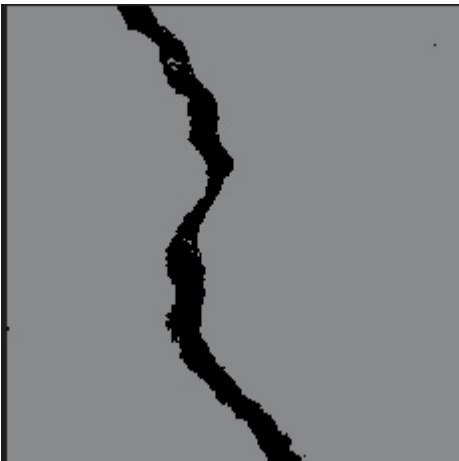
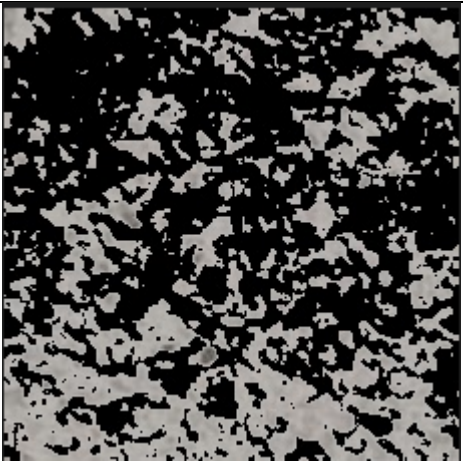
Firstly, we plot the histogram of some of the images that belongs to the 2 classes:



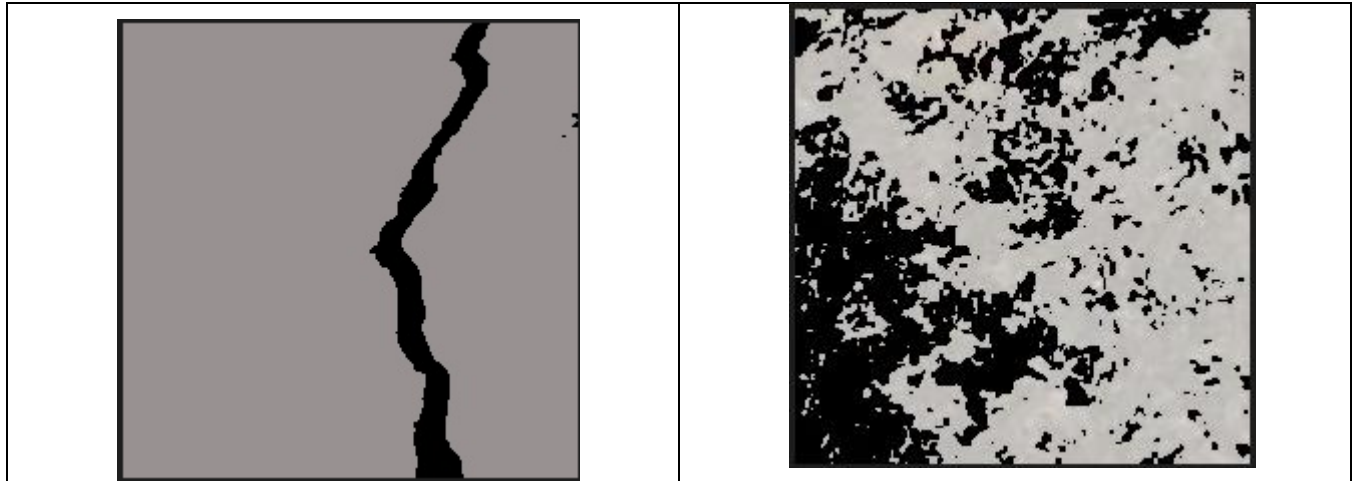


Between these histograms we plotted from some sample images of both classes, there is an important little detail that we want to focus on. All the histograms that belong to the Positive class have a bigger tale in comparison with those from the Negative class, this is because of the disparity between the number of pixels that have darker color or smaller value (a.k.a the crack region) and the number of pixels that have brighter color or larger value (a.k.a the normal region).

With image samples that has color disparity like this, we want to segment them based on color features. The process of segmentation by color can be achieved by using some of the clustering algorithms with pixel values as our criteria. For this particular situation, we would like to use KMeans with predefined number of K equals to 2 since we want to segment the image into 2 parts, one that has darker color and one that has brighter color. This whole process can be considered as the process of turning all RGB images into 2-color images.

Positive	Negative
	
	
	



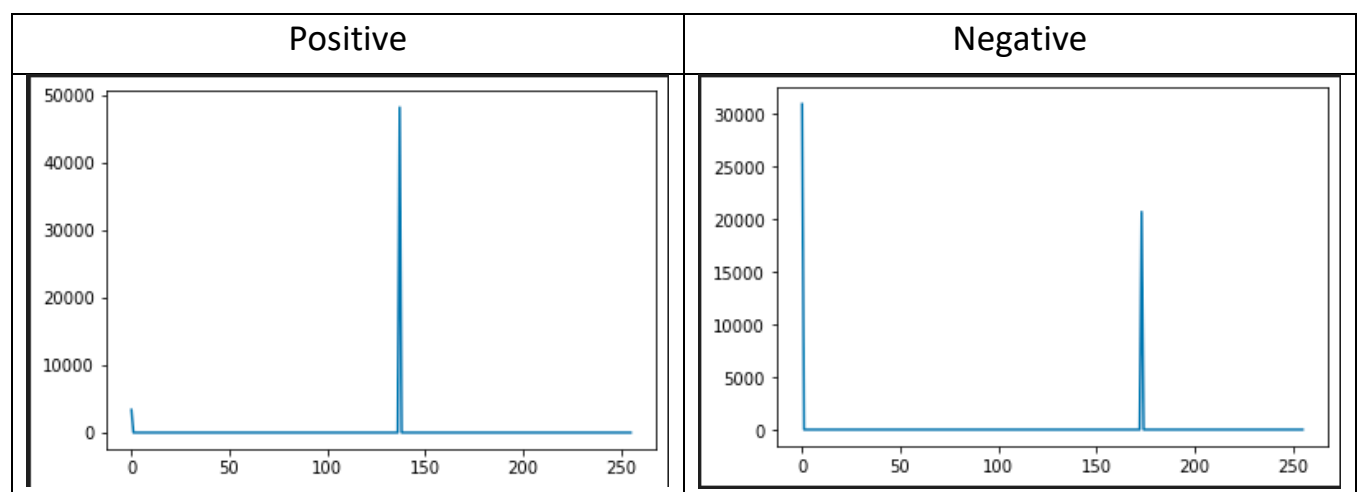


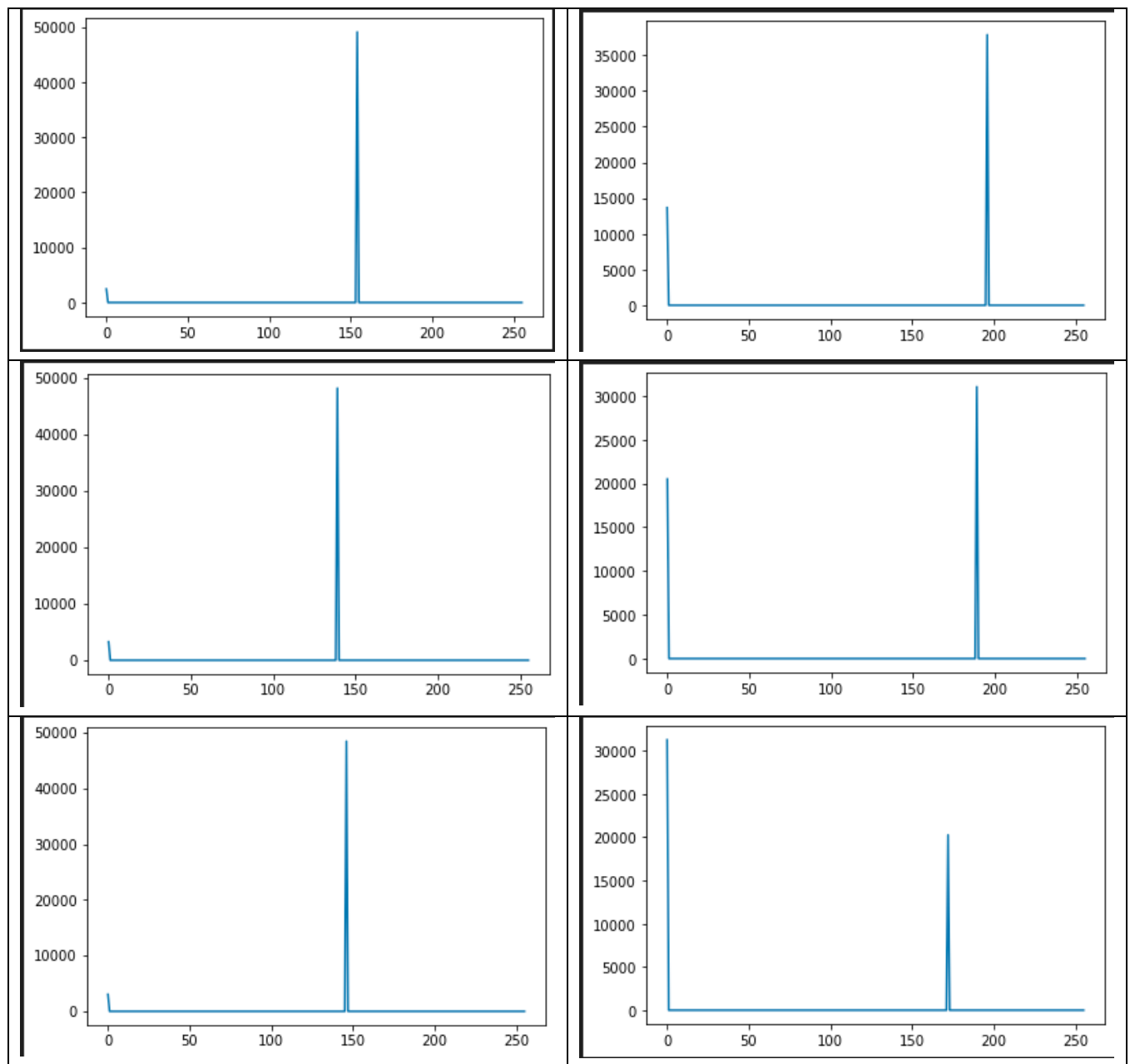
*Result of KMeans color clustering with K=2*

For the result images that belong to Negative class, since the original images does not have different regions with significantly distinct color range, the 2 cluster centroids we got from KMeans does not differ much from each other, that is why the number of dark pixels is nearly similar to the number of brighter pixels.

On the other hand, for the results we got from the Positive class, the crack area is well segmented, hence the number of dark pixels is much less than the number of brighter pixels.

To have a better visualization of these result images, we will again plot the histogram for each of them:





*Histogram of images after Kmeans*

The difference is now much easier to exploit. We will classify the images based on the disparity between the number of darker pixels and the number of brighter ones, which we call as diff-value. Our last step is to select a proper threshold so that any images that have diff-value higher than this threshold will be classified as Positive, and any images that have diff-value smaller than this threshold will be classified as Negative.

### 3.2. Experiment results

The threshold values are tuned by hand after several runs.

**OpenCV KMeans; number of K: 2; Initial centroids: Random; Threshold: 0.75**

	precision	recall	f1-score	support
Positive	0.65	0.95	0.77	20000
Negative	0.91	0.48	0.63	20000
accuracy			0.72	40000
macro avg	0.78	0.72	0.70	40000
weighted avg	0.78	0.72	0.70	40000
[[19001 999]				
[10328 9672]]				

Using Kmeans with random initial centroids yields a bad result. This is because of the KMeans itself, with randomly initial centroids, there will be cases that bad choices of initial centroids worsen the clustering performance of the KMeans. Furthermore, by randomly pick initial centroids, the results between different runs of KMeans are not the same, this has bad effect on how we choose a proper threshold later on. To counter this problem, we will try KMeans++ with better strategy to pick our first centroids.

**Sklearn KMeans; Number of K: 2; Initial centroids: KMeans++; Threshold: 0.8**

#### KMeans++:

K-Means++ is a smart centroid initialization technique and the rest of the algorithm is the same as that of K-Means. The steps to follow for centroid initialization are:

- Pick the first centroid point ( $C_1$ ) randomly.
- Compute distance of all points in the dataset from the selected centroid. The distance of  $x_i$  point from the farthest centroid can be computed by

$$d_i = \max_{(j:1 \rightarrow m)} ||x_i - C_j||^2$$

In which,  $d_i$  is the Distance of  $x_i$  point from the farthest centroid and  $m$  is the number of centroids already picked

Result:

	precision	recall	f1-score	support
Positive	0.93	0.95	0.94	20000
Negative	0.95	0.92	0.94	20000
accuracy			0.94	40000
macro avg	0.94	0.94	0.94	40000
weighted avg	0.94	0.94	0.94	40000
[[19064 936]				
[ 1516 18484]]				

The performance is now greatly improved with KMeans++. We can say that initial centroids have a major impact on KMeans performance.

### 3.3. Conclusion:

This method using Image processing and KMeans performs well and the best result reaches the F1-score of 94%, and Positive recall of 95%. We want to focus on the Positive recall accuracy here since it is particularly important due to the characteristics of this problem, we do not want to miss any surfaces that have crack, it may lead to severe consequences.

In comparison with the above methods using CNN and VGG-16, this approach performs worse. This is easily understandable since in the most cases, Deep Learning methods out-perform traditional image processing methods. Especially for this well-balanced dataset.

## IV. CONTRIBUTION

	Pham Van Khoa	Vu Hai Dang	Tran Le Hoang	Tran Hai Son	Hoang Tuan Anh Van
<b>CNN</b>	x	x		x	
<b>VGG-16</b>			x		x
<b>Image Processing</b>	x				
<b>Slides + Report</b>	x	x	x	x	x

## V. REFERENCES

- (1) <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- (2) <https://towardsdatascience.com/understanding-k-means-k-means-and-k-medoids-clustering-algorithms-ad9c9fbf47ca>
- (3) <https://arxiv.org/pdf/1409.1556.pdf>

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION - Karen Simonyan & Andrew Zisserman