UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Foundations of Deep Learning

Giovanni Cornacchia	830631
Alberto Gadda	824029
Paolo Guerini Rocco	826236

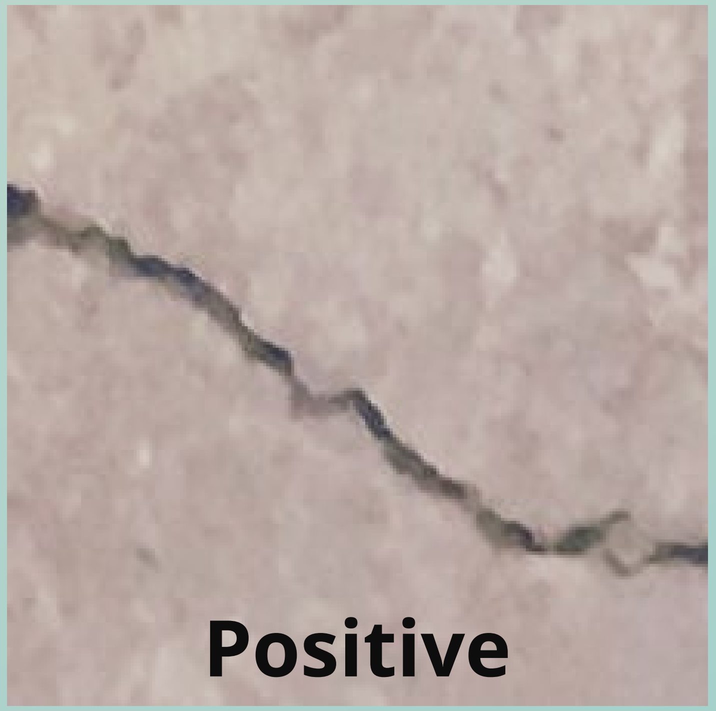
Introduction to the dataset

Image Data Classification - *Concrete Crack*

The Concrete Crack dataset contains images representing parts of roads, divided in two classes: "**Negative**" images without cracks and "**Positive**" images with cracks.

The dataset provides **20 000 RGB images** (227 x 227 x 3) **of each class**. The size of the dataset is 235 MB.

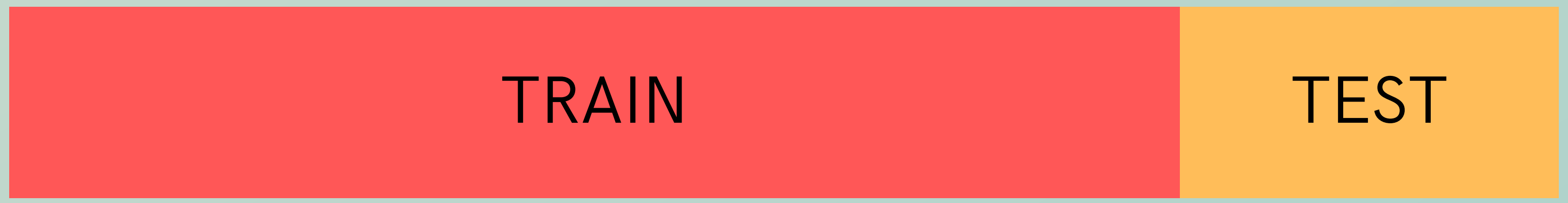
The goal of this project is to create the best architecture to properly classify the presence or absence of cracks in the images.



Training-Test split

The dataset was divided in two sections with balanced amount of positive and negative samples:

- Train: **75%** of the dataset (30 000 images)
- Test: **25%** of the dataset (10 000 images)



Data augmentation was applied to provide a greater variety of examples to the model so to ensure better training.

Data augmentation

The data augmentation process brings the training size from 30 000 to 56 000 images (**+86.7%**). Since the classes were already balanced, for each value of the response variable were created 13 000 new images.

This brings the augmentated training vs. test split to be 82.1% vs. 17.9%.

While the original images were kept intact, the augmented images were edited to create new and plausible samples.

Data augmentation procedure

The augmented data was obtained by manipulating the original images via the following steps in order:

- Edit **brightness** for the whole image (random value in range [-15; 15])
- Edit **contrast** for the whole image (random value in range [x0.7; x1.3])
- Add **white noise** to each value (random value in range [-30; 30])
- **Flip** the image by 90° in a random direction
- Force each value in range [0; 255]

Negative images were treated as described above, while for positive images there was an extra consideration based on the crack orientation (i.e. whether the crack is horizontal, vertical or diagonal).

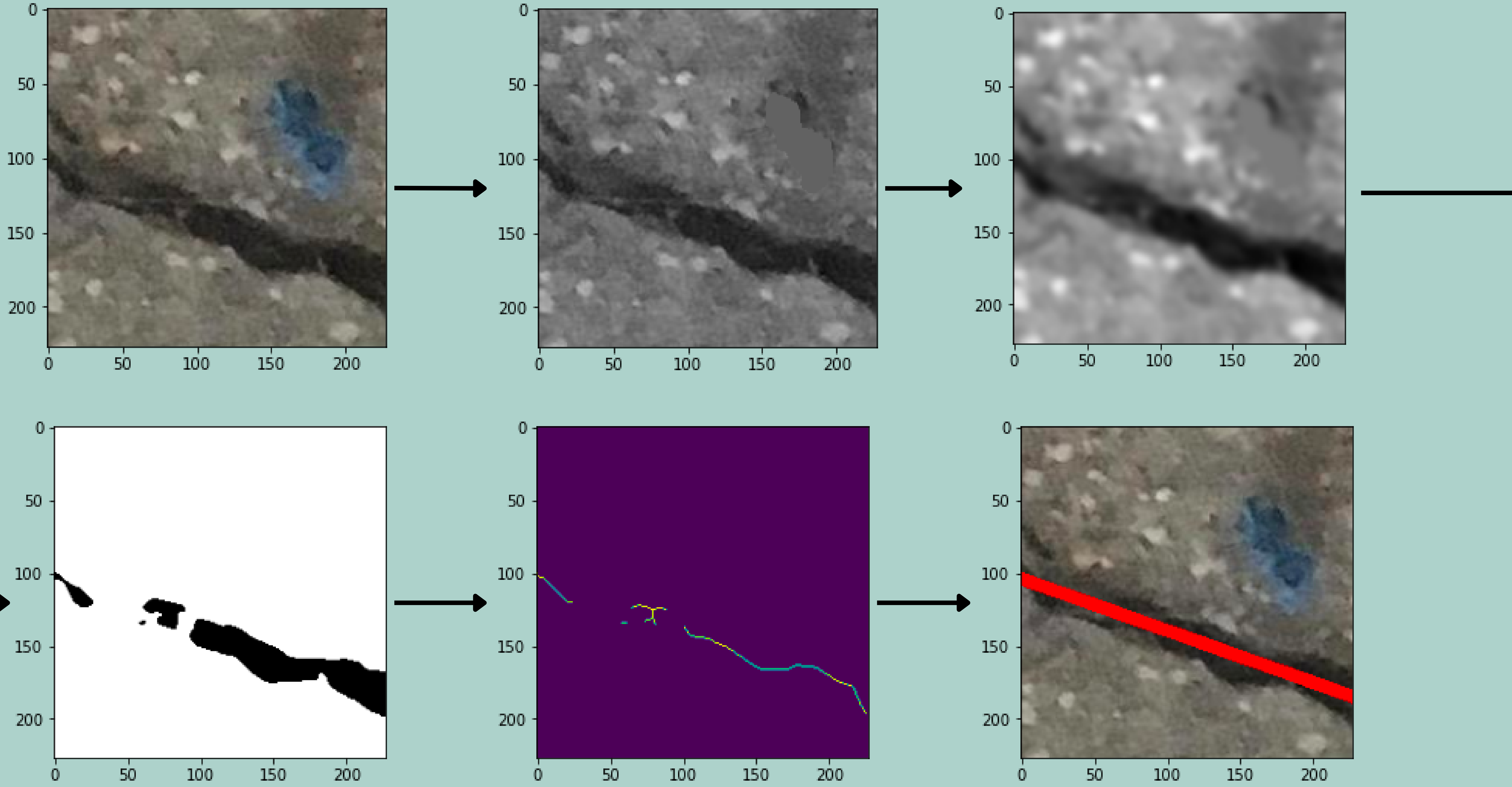
Crack orientation

When creating new positive images an extra effort was made to **balance the crack orientations** (e.g., if there are too many horizontal cracks and too few vertical ones, new vertical lines will be generated via flipping some horizontal ones by 90°).

The crack orientations found were in fact imbalanced:

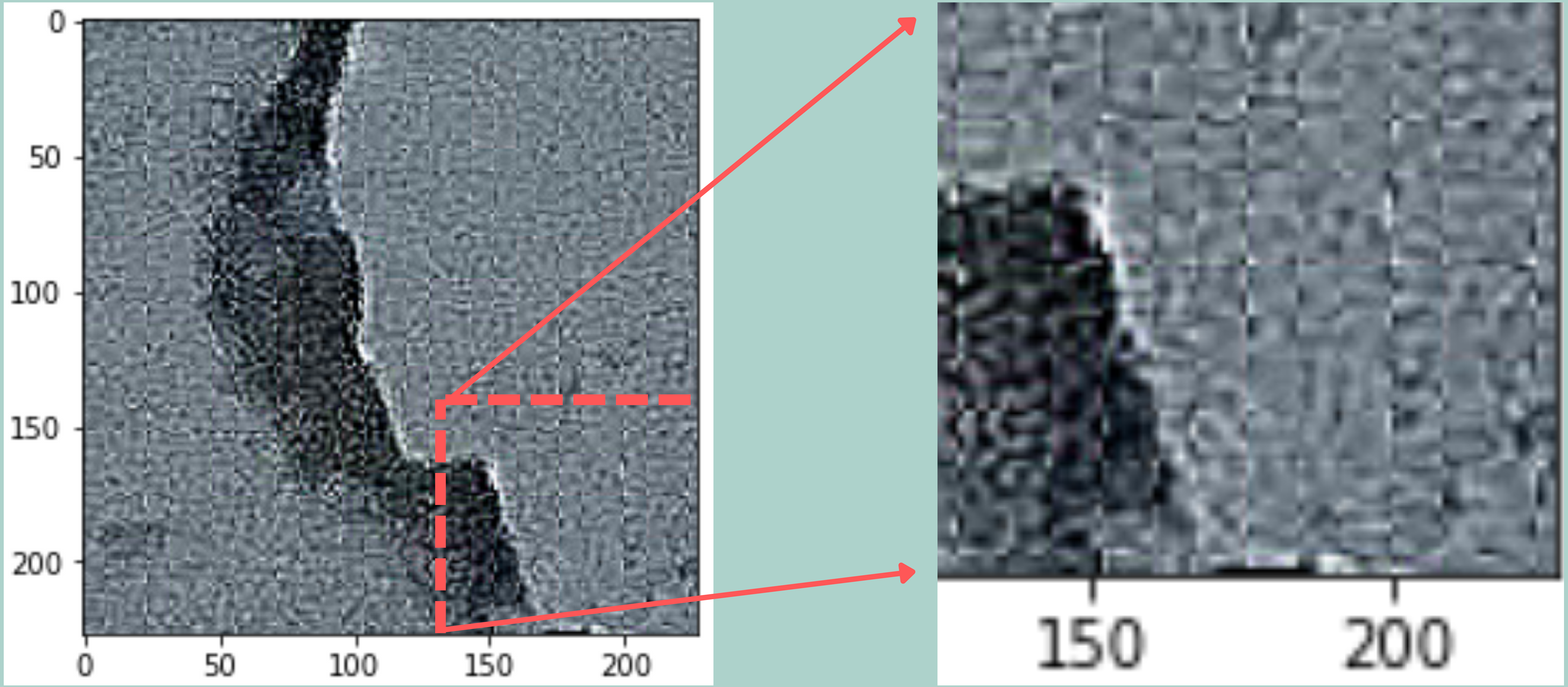
- 6545 horizontal
- 1171 vertical
- 3227 diagonal (bottom left - upper right)
- 4057 diagonal (bottom right - upper left)

How to retrieve the crack orientation?



Note on data quality

Via increasing the saturation it becomes evident that the images have a squared pattern in it (maybe caused by lossy compression?). This may lead to a poor performance if the images to classify in production will not to have this sort of defect.



Models

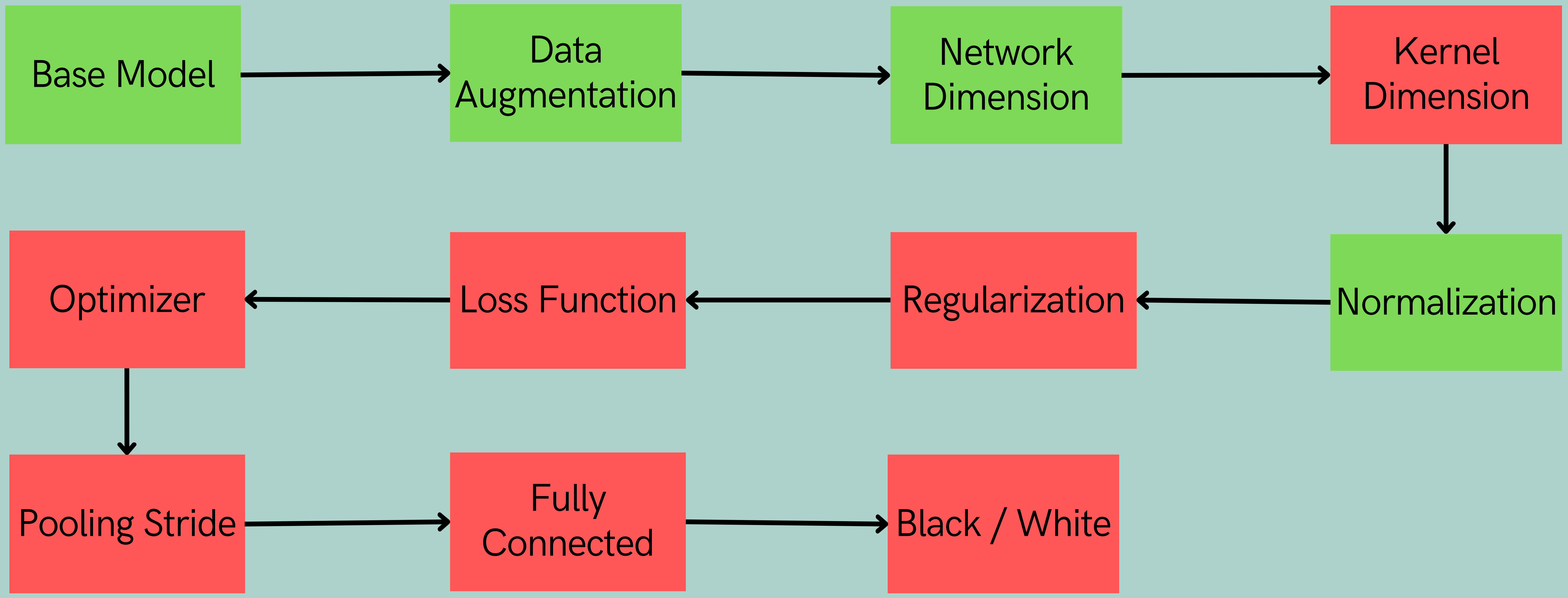
For the selection of the best architecture, the first model to be tested was a **very simple network** consisting of just two layers (32 and 64 nodes) that already produced very good performance (99.07% accuracy).

Since the initial performance was already so high, the possible **improvements were really limited**.

The metrics used to evaluate the models were mainly **accuracy** (since the classes are balanced) and **sensitivity** (since false negatives are far worse than false positives due to safety being a priority for a road).

Model choices

Multiple solutions were tested in order to find the best architecture. At each step the best performing architecture helped to understand what characteristics to keep or discard for the following attempts.

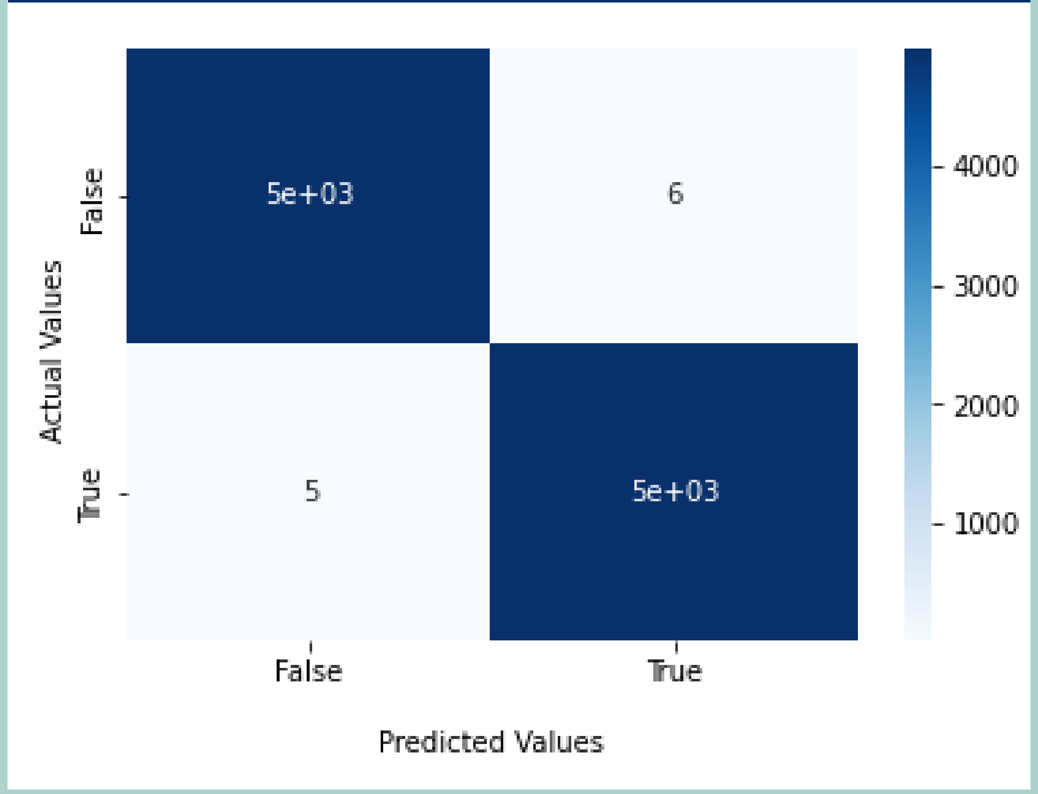


Best models

The **best three models** obtained perform as follows. The metrics are evaluated on the **validation set**.

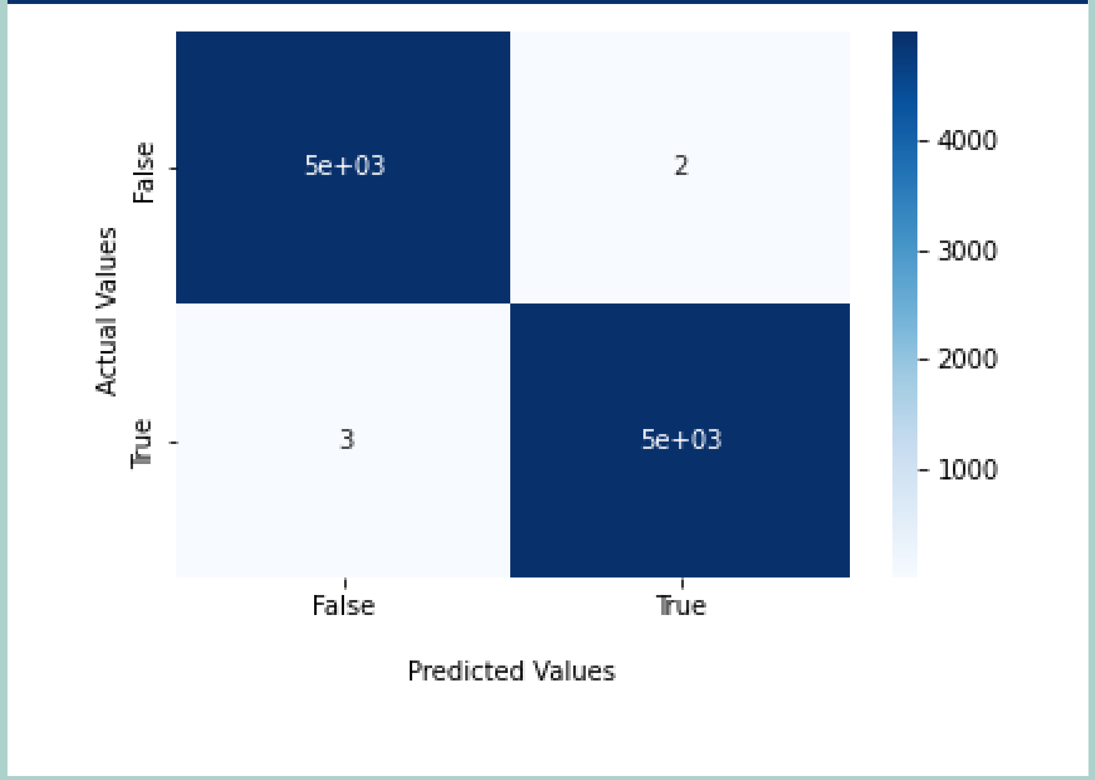
NORM

ACC=99.89% SENS=99.90%



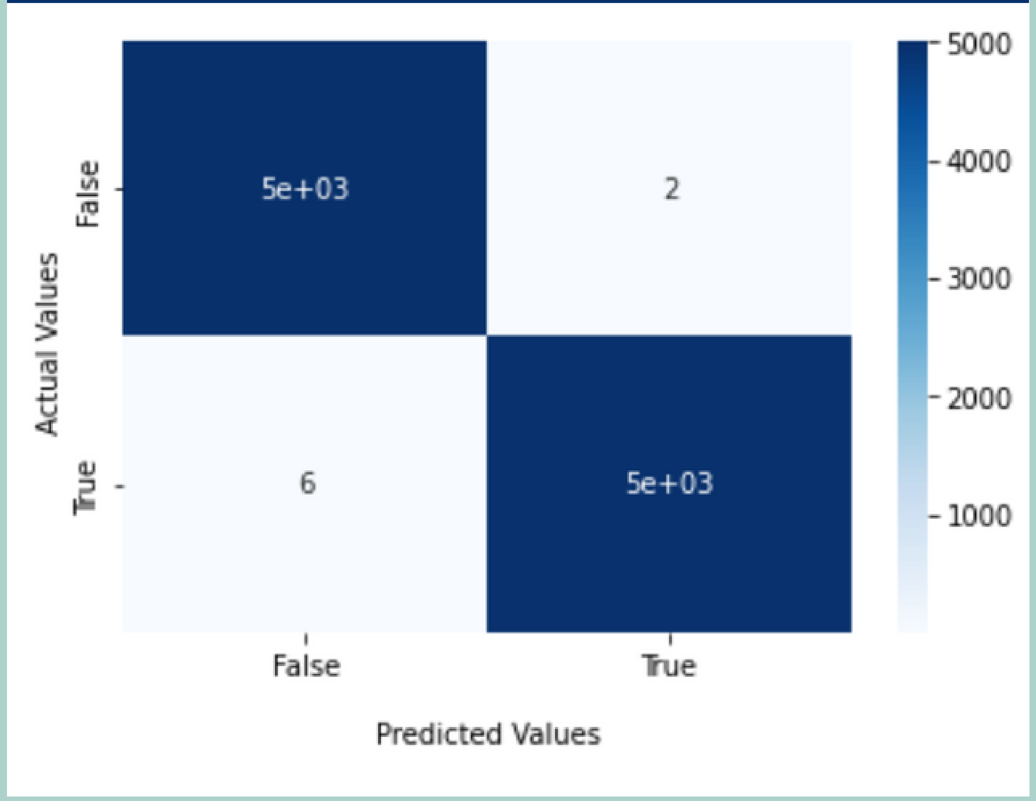
NORM_s4

ACC=99.95% SENS=99.94%



B/W

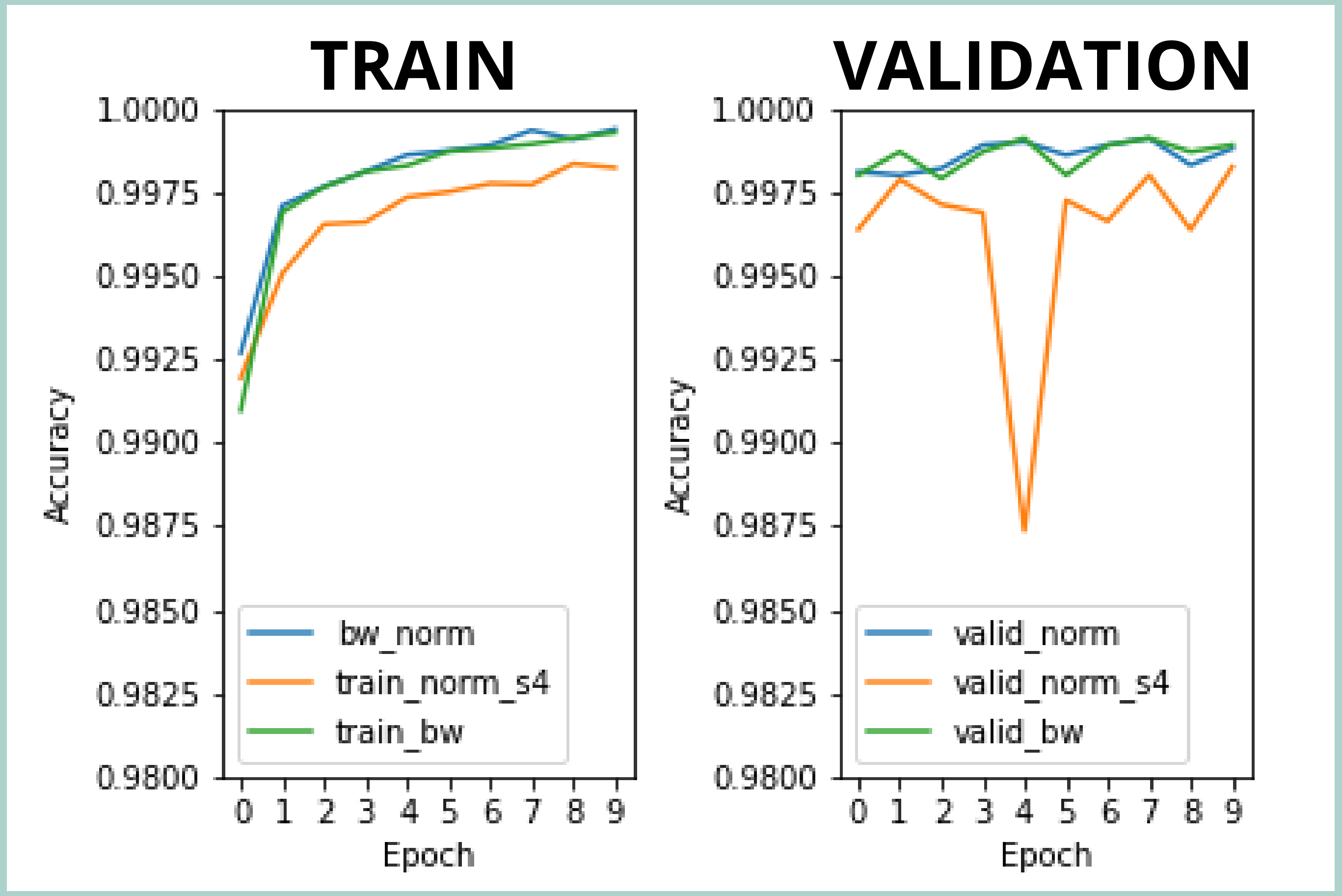
ACC=99.92% SENS=99.86%

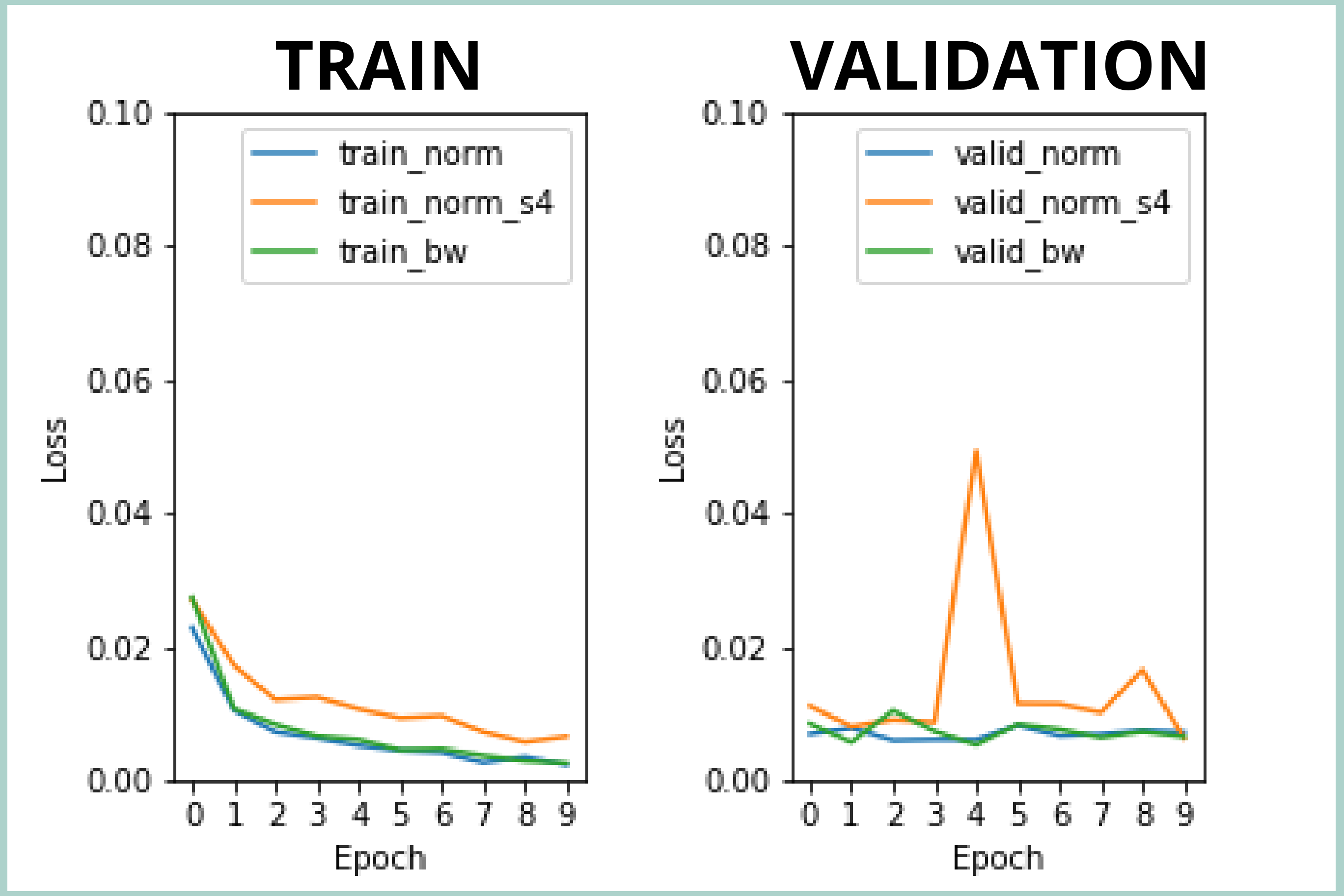


Visualization of the learning curves

The best model in terms of accuracy and sensitivity is "**NORM_s4**", even though given its worse training performance it could be poorly robust.

For this reason, we **keep all three models** and evaluate them on the test set.



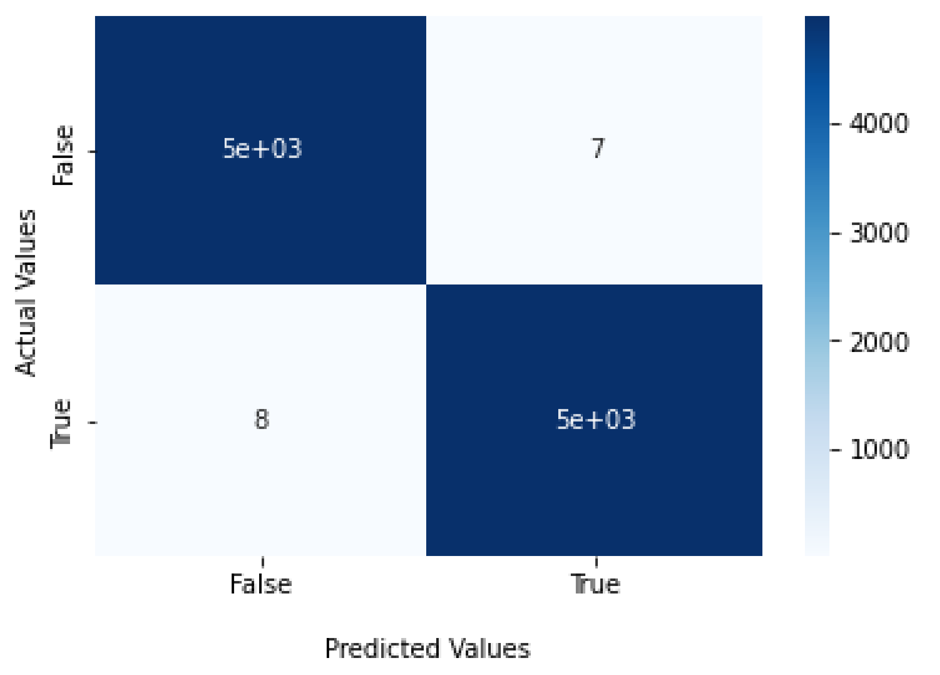


Best models on test set

The chosen models were then trained on the whole training set (validation included) for 40 epochs with early stopping. On the test set, the best performing model is "**NORM**".

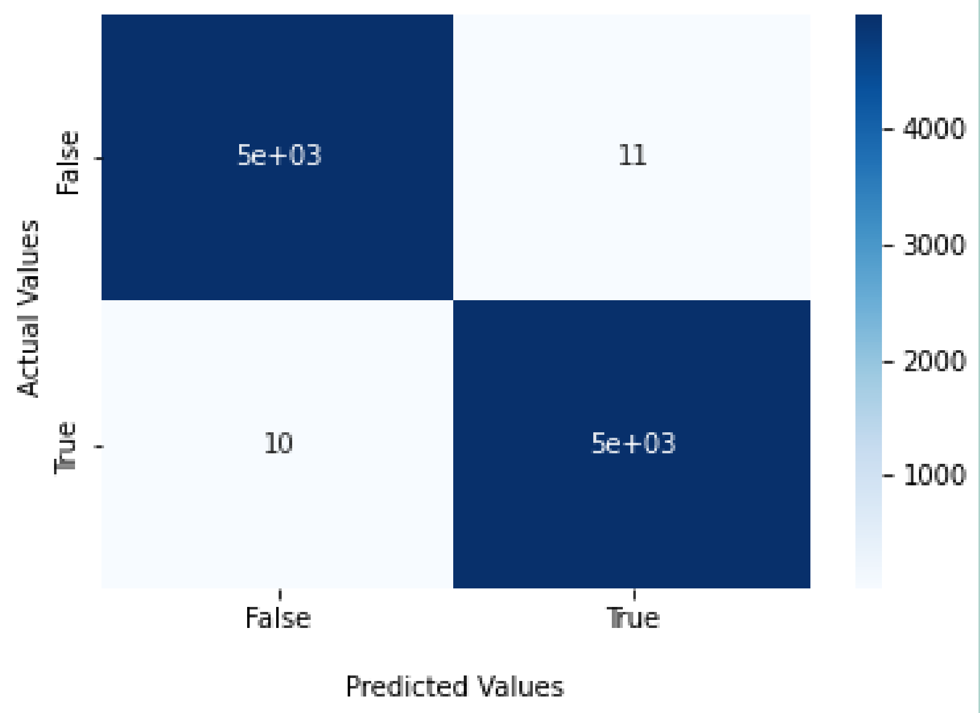
NORM

ACC=99.85%   SENS=99.84%



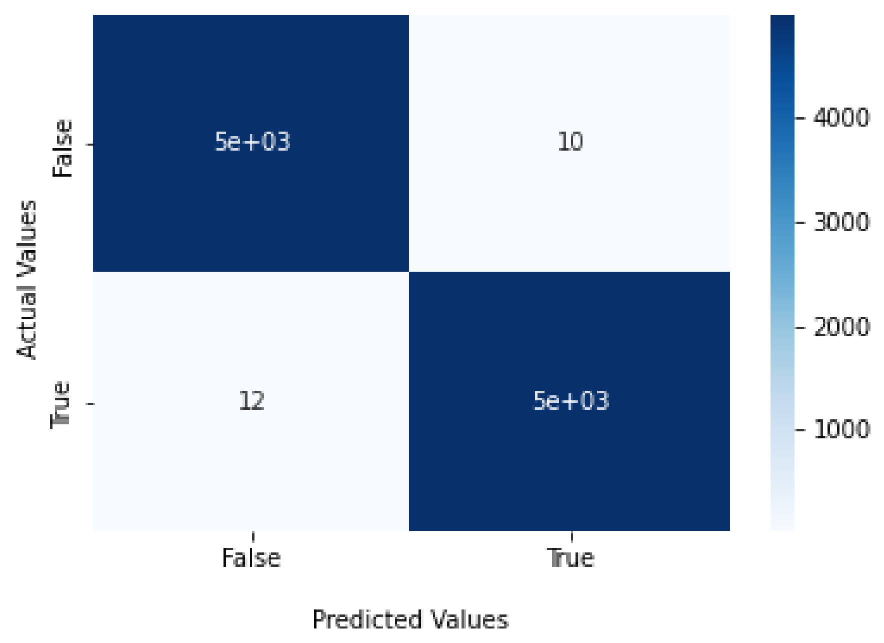
NORM_s4

ACC=99.79%   SENS=99.80%



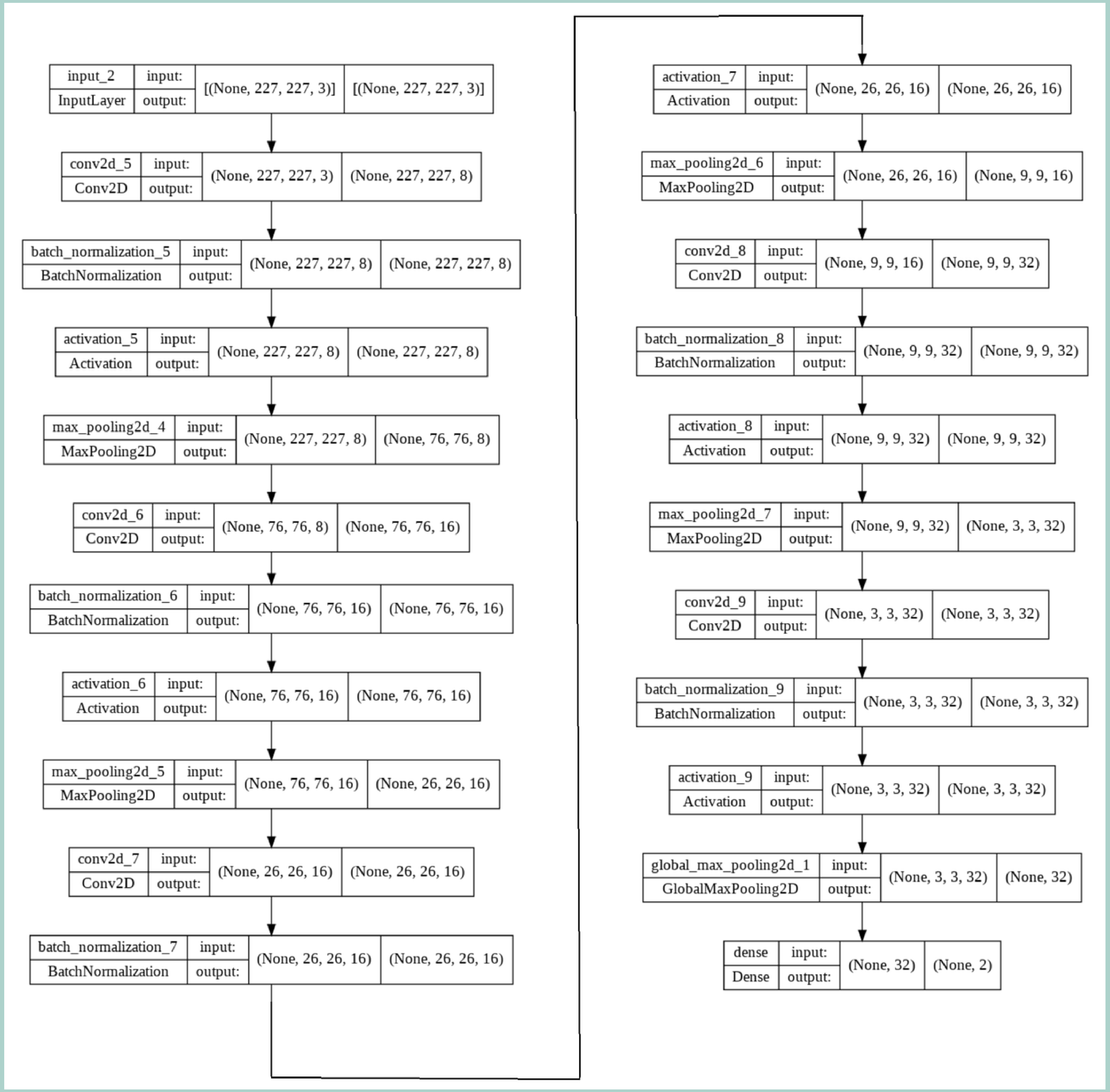
B/W

ACC=99.78%   SENS=99.76%



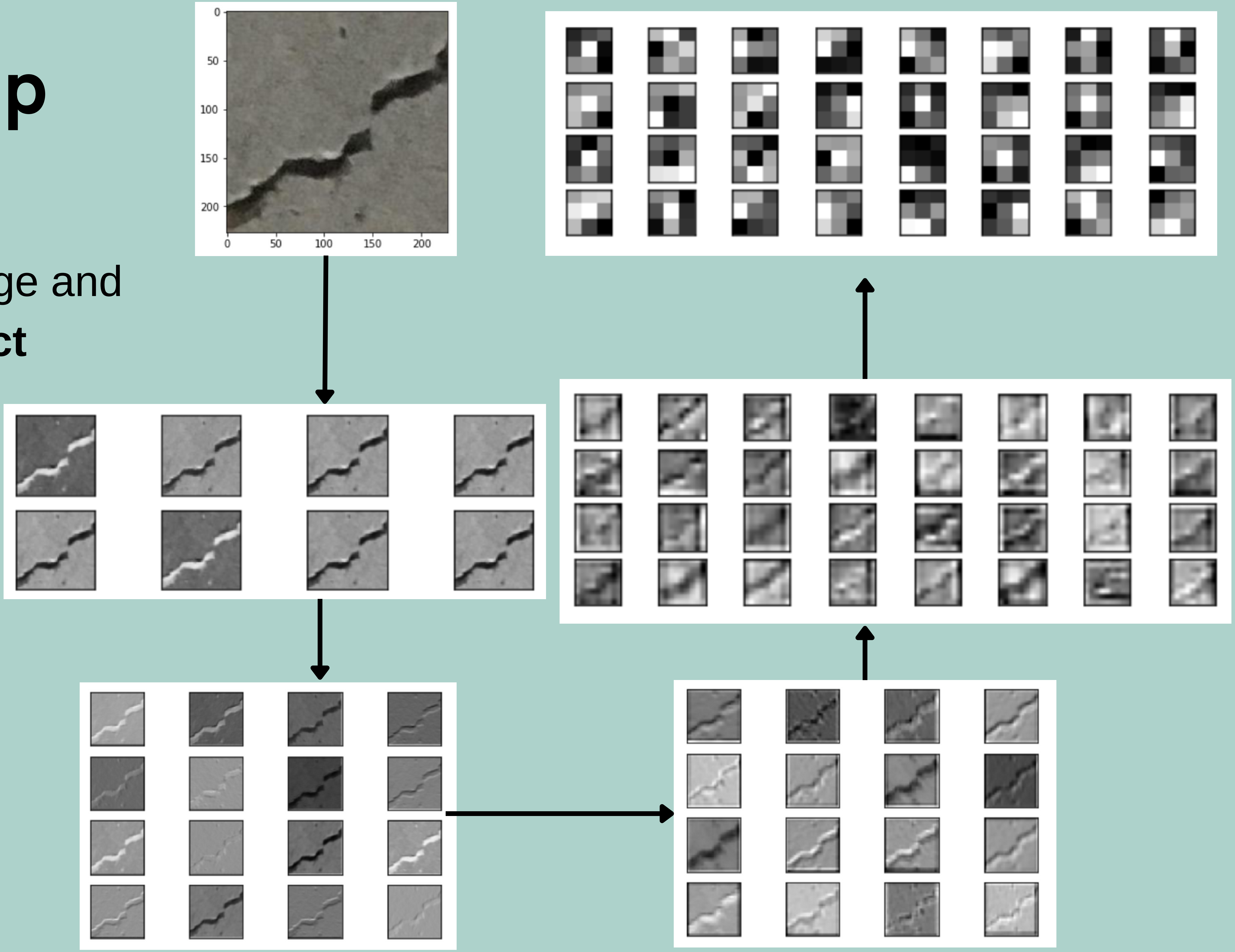
Architecture of the best model found

- 5 convolutional layers:
  - dimension: 8, 16, 16, 32, 32
  - kernel dimension: 3x3
- maxpooling (filter: 3x3, stride: 3)
- BatchNormalization
- layer activation function: ReLU
- final activation function: softmax
- 17 744 total parameters to estimate


input_2 | InputLayer | input: [(None, 227, 227, 3)] | output: [(None, 227, 227, 3)]
conv2d_5 | Conv2D | input: (None, 227, 227, 3) | output: (None, 227, 227, 8)
batch_normalization_5 | BatchNormalization | input: (None, 227, 227, 8) | output: (None, 227, 227, 8)
activation_5 | Activation | input: (None, 227, 227, 8) | output: (None, 227, 227, 8)
max_pooling2d_4 | MaxPooling2D | input: (None, 227, 227, 8) | output: (None, 76, 76, 8)
conv2d_6 | Conv2D | input: (None, 76, 76, 8) | output: (None, 76, 76, 16)
batch_normalization_6 | BatchNormalization | input: (None, 76, 76, 16) | output: (None, 76, 76, 16)
activation_6 | Activation | input: (None, 76, 76, 16) | output: (None, 76, 76, 16)
max_pooling2d_5 | MaxPooling2D | input: (None, 76, 76, 16) | output: (None, 26, 26, 16)
conv2d_7 | Conv2D | input: (None, 26, 26, 16) | output: (None, 26, 26, 16)
batch_normalization_7 | BatchNormalization | input: (None, 26, 26, 16) | output: (None, 26, 26, 16)
activation_7 | Activation | input: (None, 26, 26, 16) | output: (None, 26, 26, 16)
max_pooling2d_6 | MaxPooling2D | input: (None, 26, 26, 16) | output: (None, 9, 9, 16)
conv2d_8 | Conv2D | input: (None, 9, 9, 16) | output: (None, 9, 9, 32)
batch_normalization_8 | BatchNormalization | input: (None, 9, 9, 32) | output: (None, 9, 9, 32)
activation_8 | Activation | input: (None, 9, 9, 32) | output: (None, 9, 9, 32)
max_pooling2d_7 | MaxPooling2D | input: (None, 9, 9, 32) | output: (None, 3, 3, 32)
conv2d_9 | Conv2D | input: (None, 3, 3, 32) | output: (None, 3, 3, 32)
batch_normalization_9 | BatchNormalization | input: (None, 3, 3, 32) | output: (None, 3, 3, 32)
activation_9 | Activation | input: (None, 3, 3, 32) | output: (None, 3, 3, 32)
global_max_pooling2d_1 | GlobalMaxPooling2D | input: (None, 3, 3, 32) | output: (None, 32)
dense | Dense | input: (None, 32) | output: (None, 2)

Feature map example

how layers affect an image and make it more **abstract**



Thank you