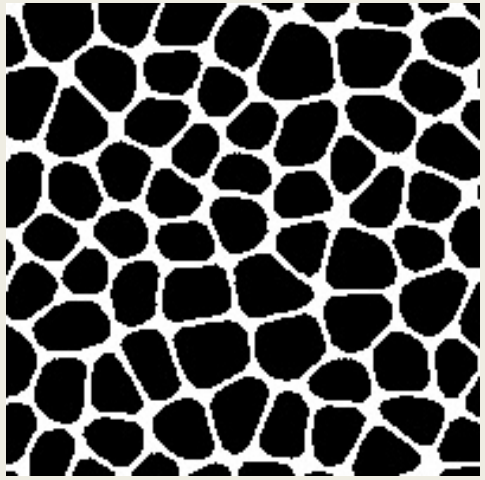
MICROSTRUCTURE RECONSTRUCTION

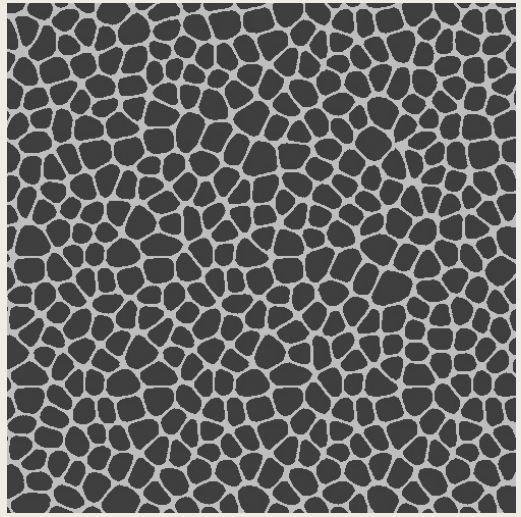
Using Artificial Neural Networks in TensorFlow

Kryštof Latka
Gymnázium Nový PORG, Praha

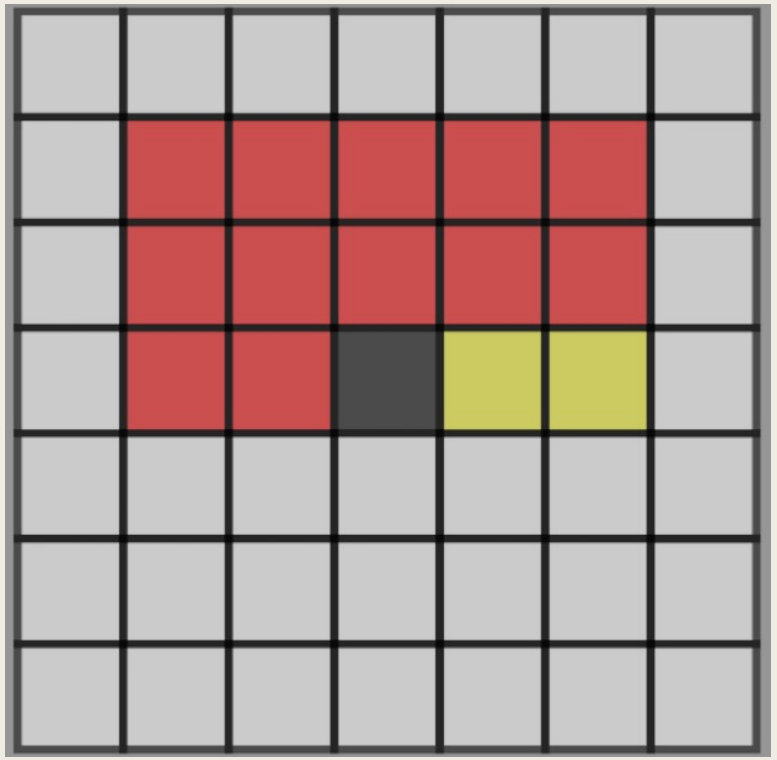
Objective

- To learn the key features of a microstructure using a neural network and to reconstruct the microstructure using those key features
- Key features are learned by taking a neighbourhood of the central pixel as an input and pairing it with a corresponding phase of the central pixel on the output
- For simplicity, we studied only two-phase materials
- We used Google's machine learning platform TensorFlow and Keras Sequential API built on top of it

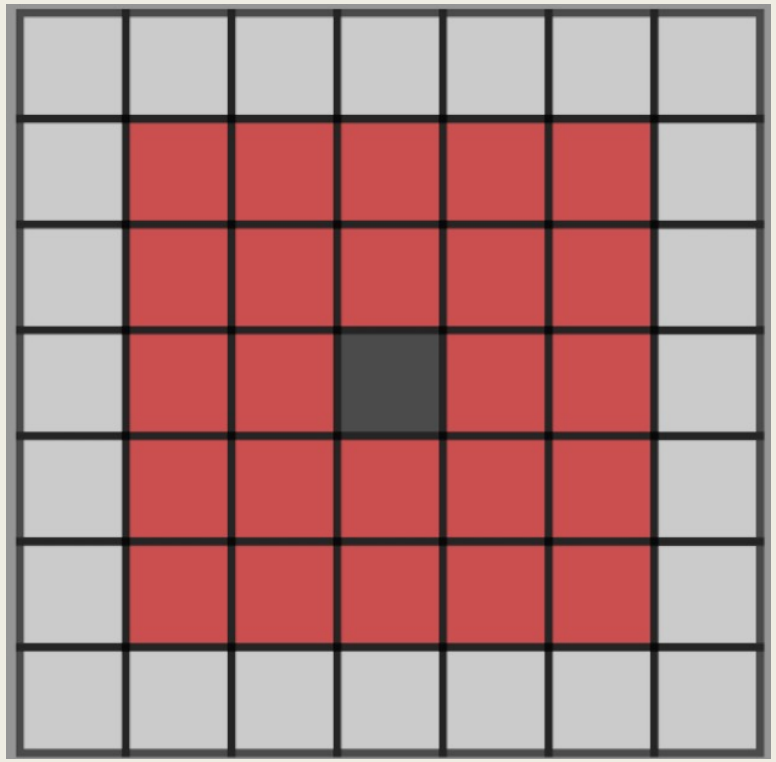




Data extraction



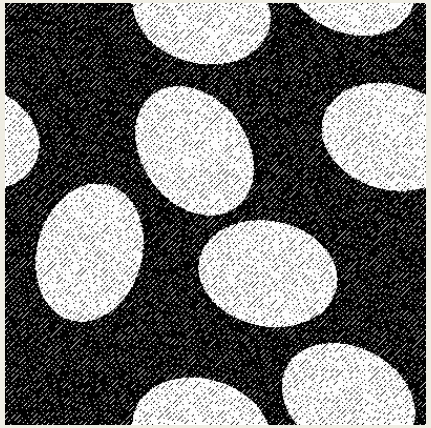
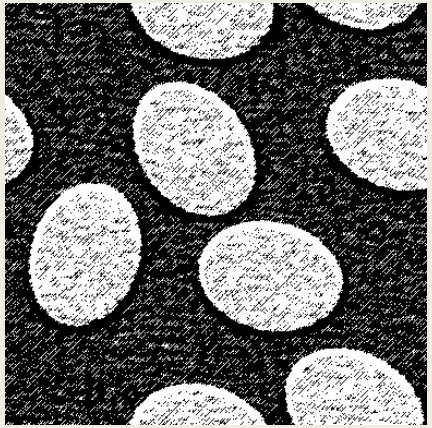
Causal neighbourhood



Non-causal neighbourhood

Initial mistakes

- We started with the non-causal approach, very simple neural network with no pooling or convolutional layers
- Reconstructions looked solid; however, this was an oversimplification => the model was not impacted by incorrect predictions on previous pixels

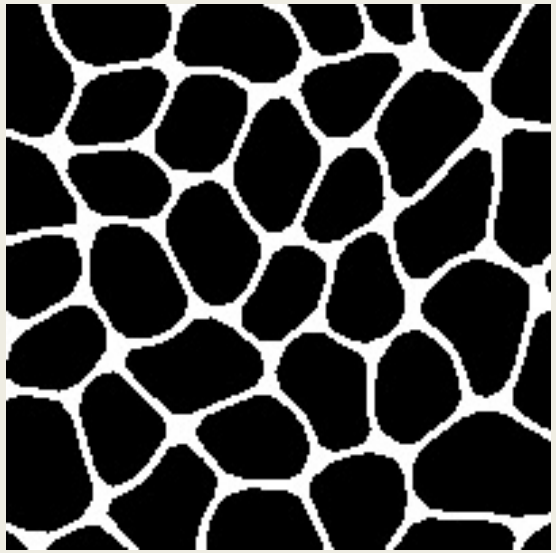
Original   Reconstructed

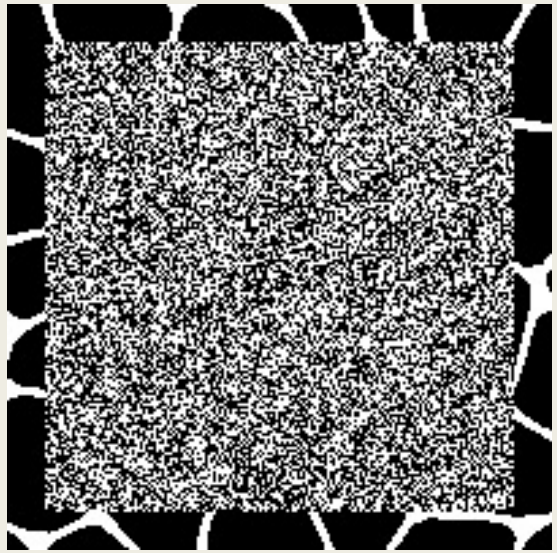
- Even though it made a mistake in a prediction, the neighbourhood of the subsequent pixel still had correct data
- We started using the causal approach, iterating through the image in a raster scan order, and predicting the pixel phase one by one

Unsuccessful reconstructions

- Once the model was modified to take into account its previous mistakes, it suddenly began to produce completely wrong outputs
- We pondered employing pooling layers

The edges must remain unchanged so that the causal model has some starting point





Pooling

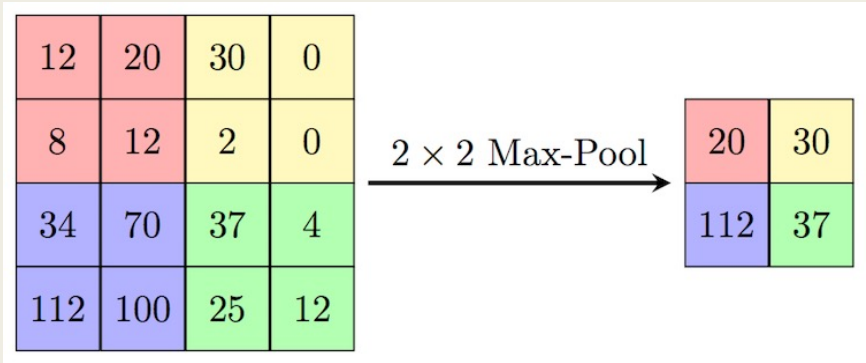
- Pooling reduces variance and computation complexity of an image => reduces image resolution
- Max Pooling: The maximum pixel value of the batch is selected to represent the phase
- Average Pooling: The average value of all the pixels in the batch is computed to represent the phase
- After the addition of pooling layers, we investigated how different configurations affect the causal model's accuracy, namely:

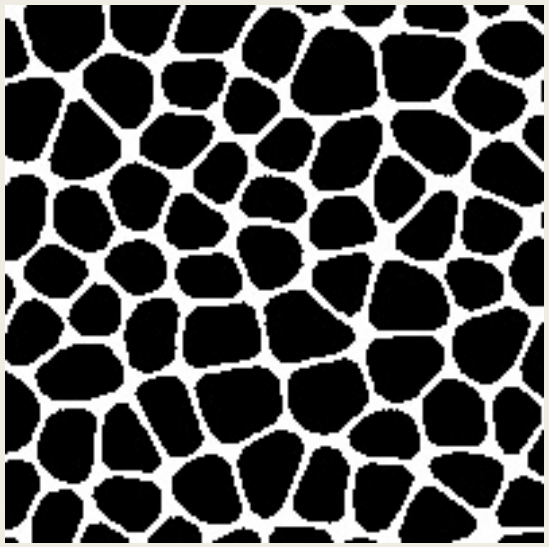
I. Number of densely connected layers

II. Number of neurons in each layer

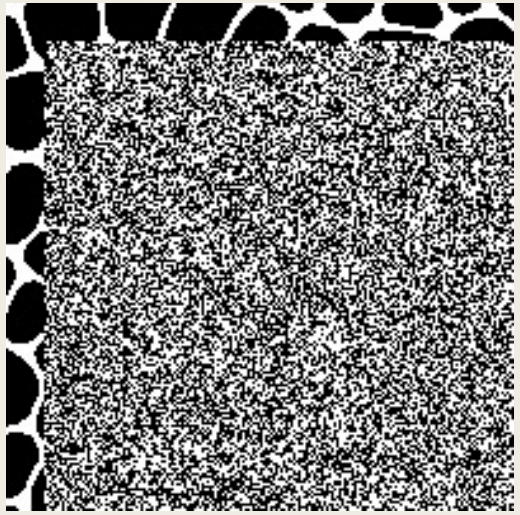
III. Window size of maximum pooling

IV. Neighbourhood window size

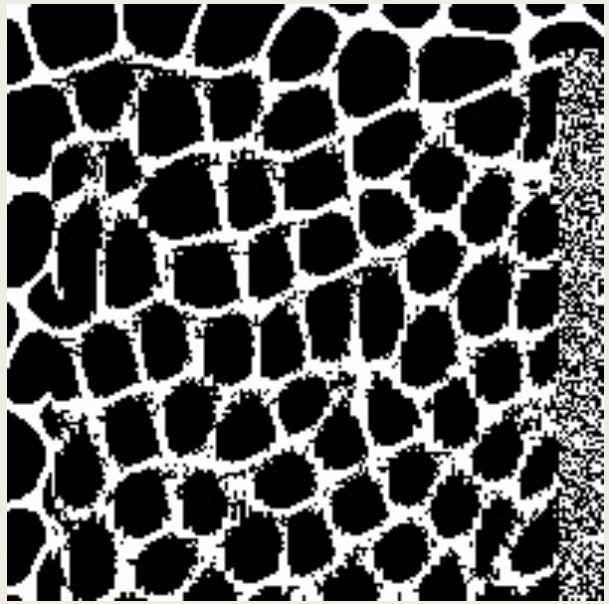




Generate random noise



Use the causal model



Smoothing/Denoising model

- Introducing a second model to smooth out the image after reconstruction
- Using the non–causal approach
- We include a small ”introduced noise” during the training of the model in order to make it more robust and less prone to mistakes
- We also assessed how different configurations affect the non-causal model’s accuracy:

I. Neighbourhood window size

II. Size of introduced noise

III. Average pooling vs maximum pooling layer

Quantification of results

- ε^{ϕ} error – based on difference in volume representation => not the most accurate metric, although some of the reconstructions had very similar volume representation to the original, the pattern was significantly different

$$\varepsilon^{\phi} = |\phi^{ref} - \phi^{orig}|$$

- ε^{S_2} error – We compute the two-point correlation function for both the reference sample and the original sample and calculate the difference. Then, we take the resulting matrix and compute the result as follows:

$$\varepsilon^{S_2} = \frac{||A||_F}{N_i \times N_j}$$

$$||A||_F = \sqrt{\sum_{i,j} (a_{i,j})^2}$$

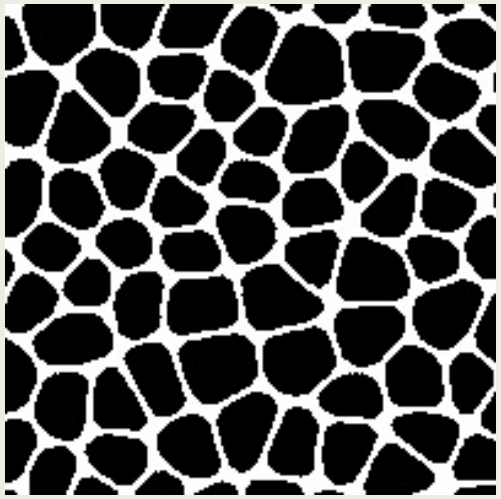
Quantification of results of the smoothing/denoising model

- ε^D error – used only for the smoothing model, assess the difference of the phase of the central pixel with the phases of its most closely neighbouring pixels
- Assumption => the number of pixels whose phase is different to that of the central pixel will be lower if the edges are properly smoothed out
- Even though this might not necessarily be true for some edge pixels, it will apply on a larger scale

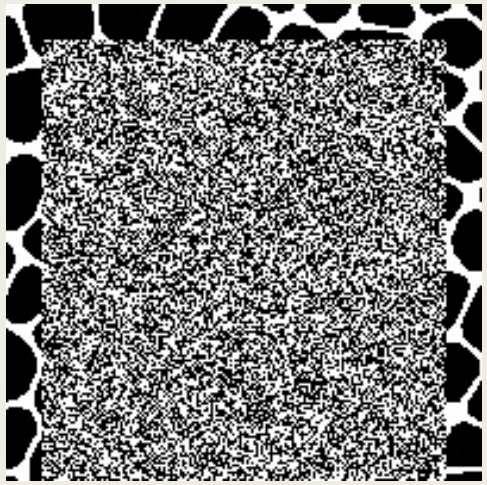
$$D_{ij} = \frac{\sum_{k,l=-1}^{1} |d_{i+k,j+l} - d_{ij}|}{8}$$

$$\varepsilon^D = \frac{\sum_{i,j=1}^{Ni,Nj} D_{ij}}{Ni \times Nj}$$

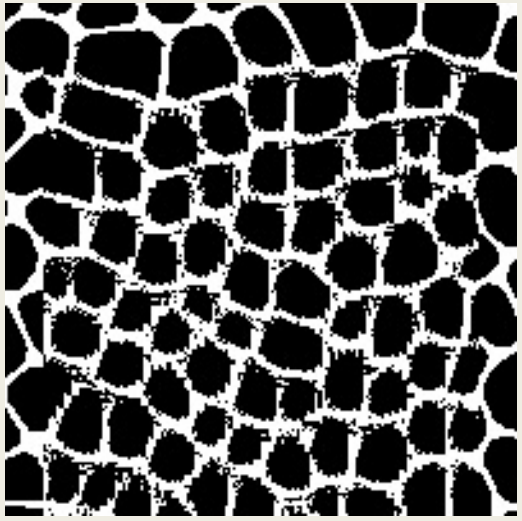
Overall process



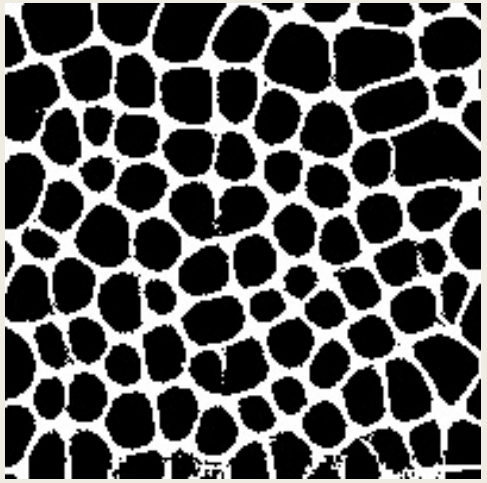
Generate
random noise



Use the
causal model



Use the non-
causal model



Data

Table 1 - Values of ε^{ϕ}

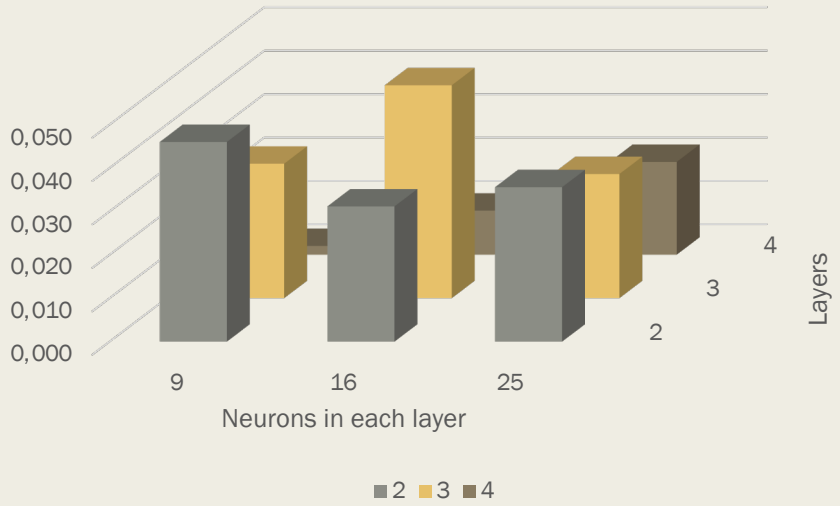


Table 2 – Values of ε^{S_2}

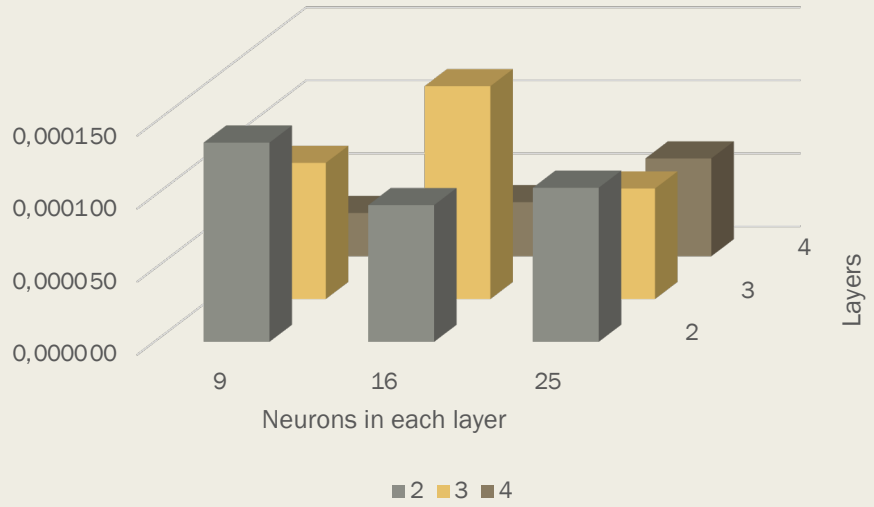


Table 3 – Values of ε^{ϕ}

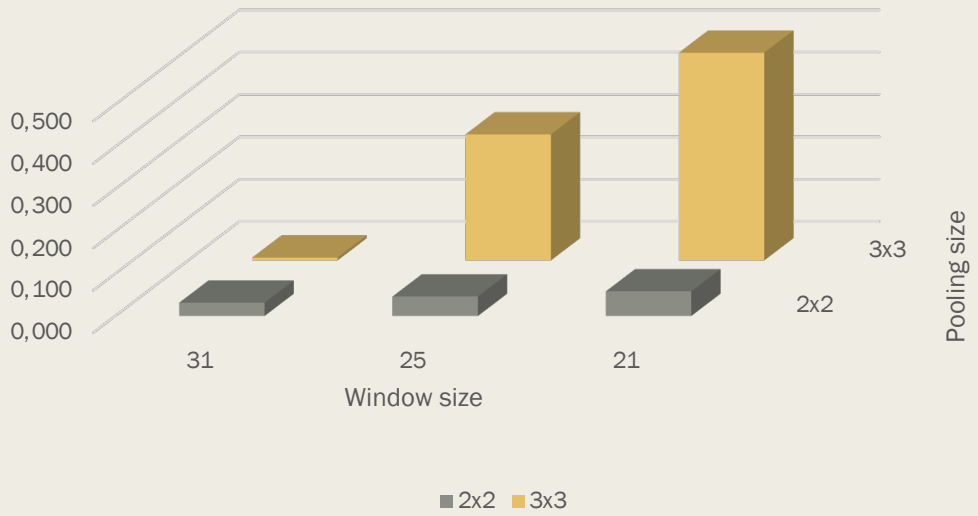
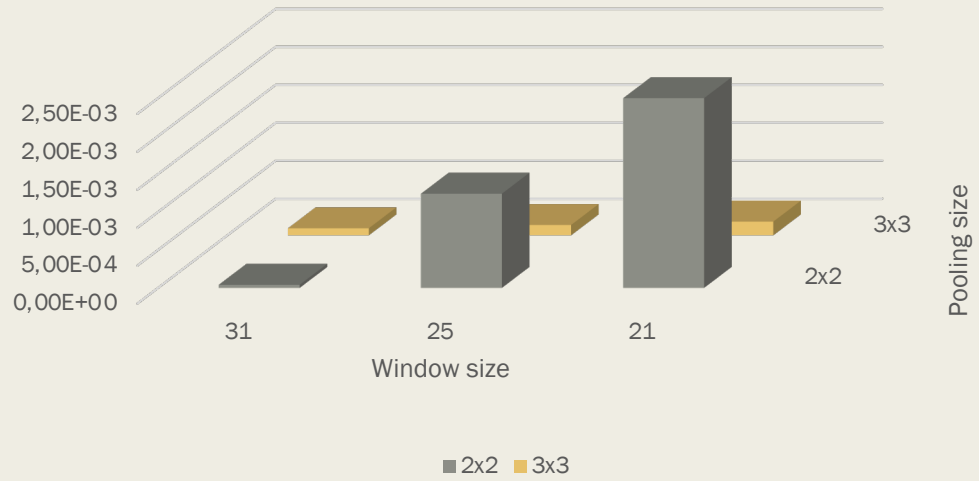
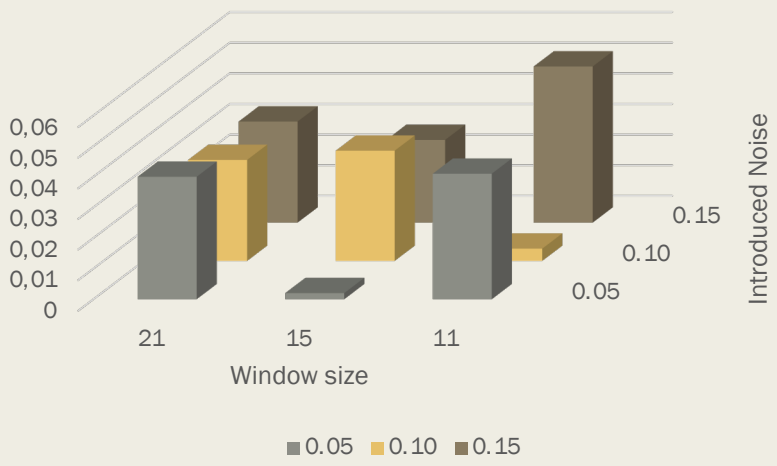


Table 4 – Values of ε^{S_2}



Smoothing Data

Table 5 – Values of ε^{ϕ}



Maximum pooling layer

Table 6 – Values of ε^{S_2}

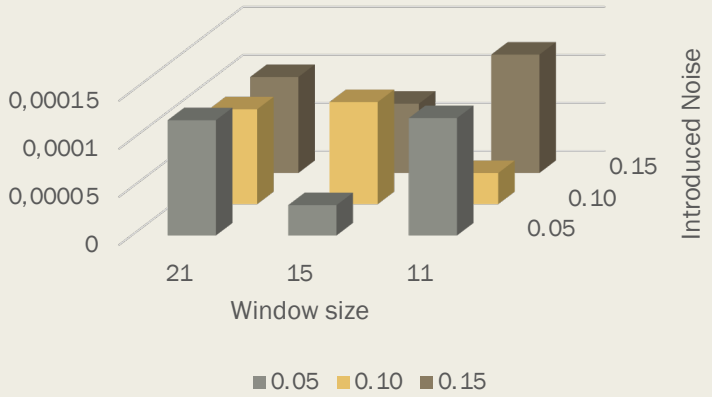
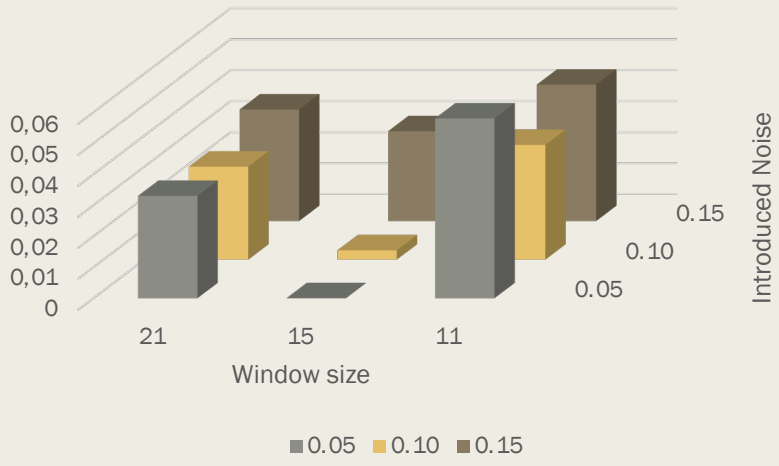
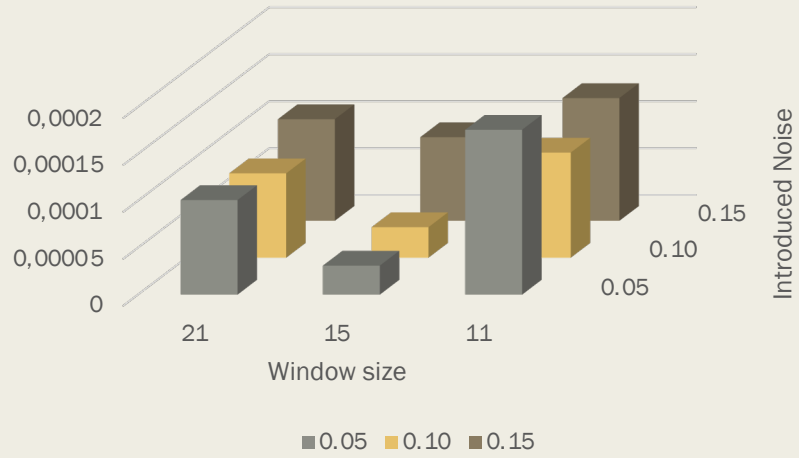


Table 8 – Values of ε^{ϕ}

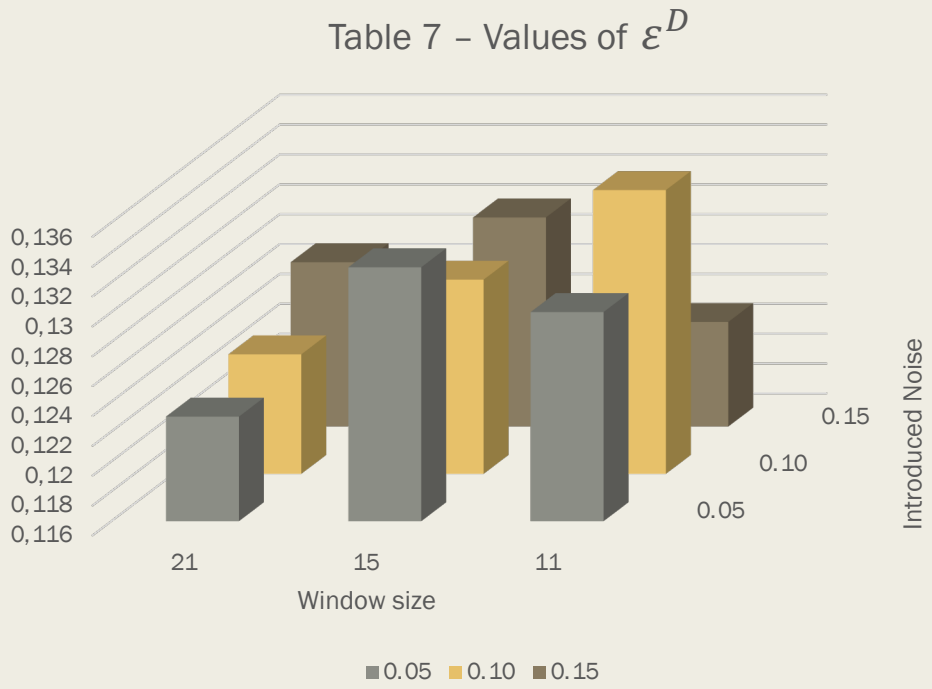


Average pooling layer

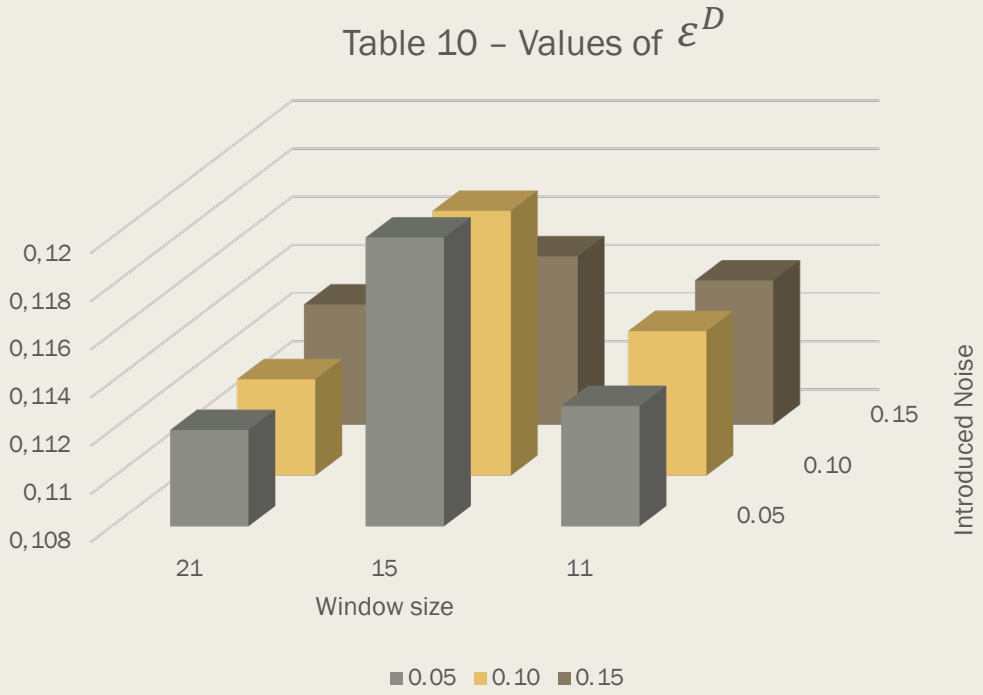
Table 9 – Values of ε^{S_2}



Smoothing Data



Maximum pooling layer

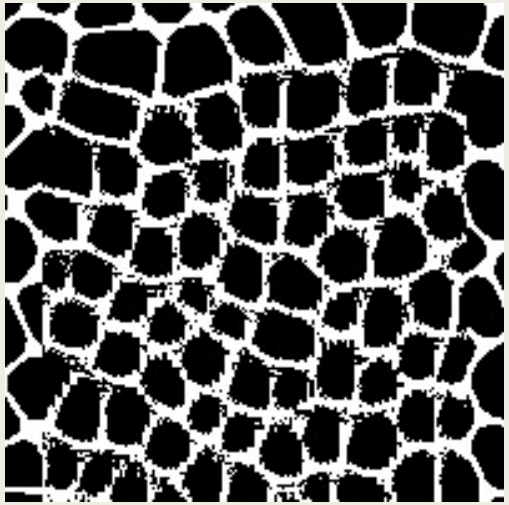


Average pooling layer

Discussion

- Discrepancy between the results of ε^{S_2} and the actual pattern

2 layers, 16 neurons, 31 window size, 3x3 pooling size



4 layers, 9 neurons, 31 window size, 3x3 pooling size

- ε^{D} values quite reliable to assess the accuracy of the smoothing models
- Average pooling layer more effective for smoothing model, ignores sharp features => allowed us to smooth out the edges

Conclusions

- Quite satisfying results given the relative simplicity of the Keras Sequential API we used
- A combination of causal and non-causal approach proved useful
- It was also preferable to use maximum pooling layer for the (causal) reconstruction model and an average pooling layer for the denoising (non-causal) model

- GitLab Repository https://gitlab.com/MartinDoskar/ai-based-reconstruction