Texture Generation with Neural Cellular Automata[1]

–

Introduction to Numerical Imaging

Grégroire Dutot [1] Hugo Simon [2]

[1]Ecole Polytechnique, France [2]Télécom Paris, France

06/02/2022

[1]Alexander Mordvintsev et al. (2021). *Texture Generation with Neural Cellular Automata*. arXiv: 2105.07299 [cs.AI].

Table of content

Texture Synthesis Using Convolutional Neural Networks

1. Neural Cellular Automata
   - Cellular Automata
   - Training procedure
   - First use cases
2. Contributions
   - Rotations
   - Learning Filters
   - Regularising the Loss
   - Texture Inpainting
   - Feature Visualization
3. Conclusion and perspectives

Cellular Automata

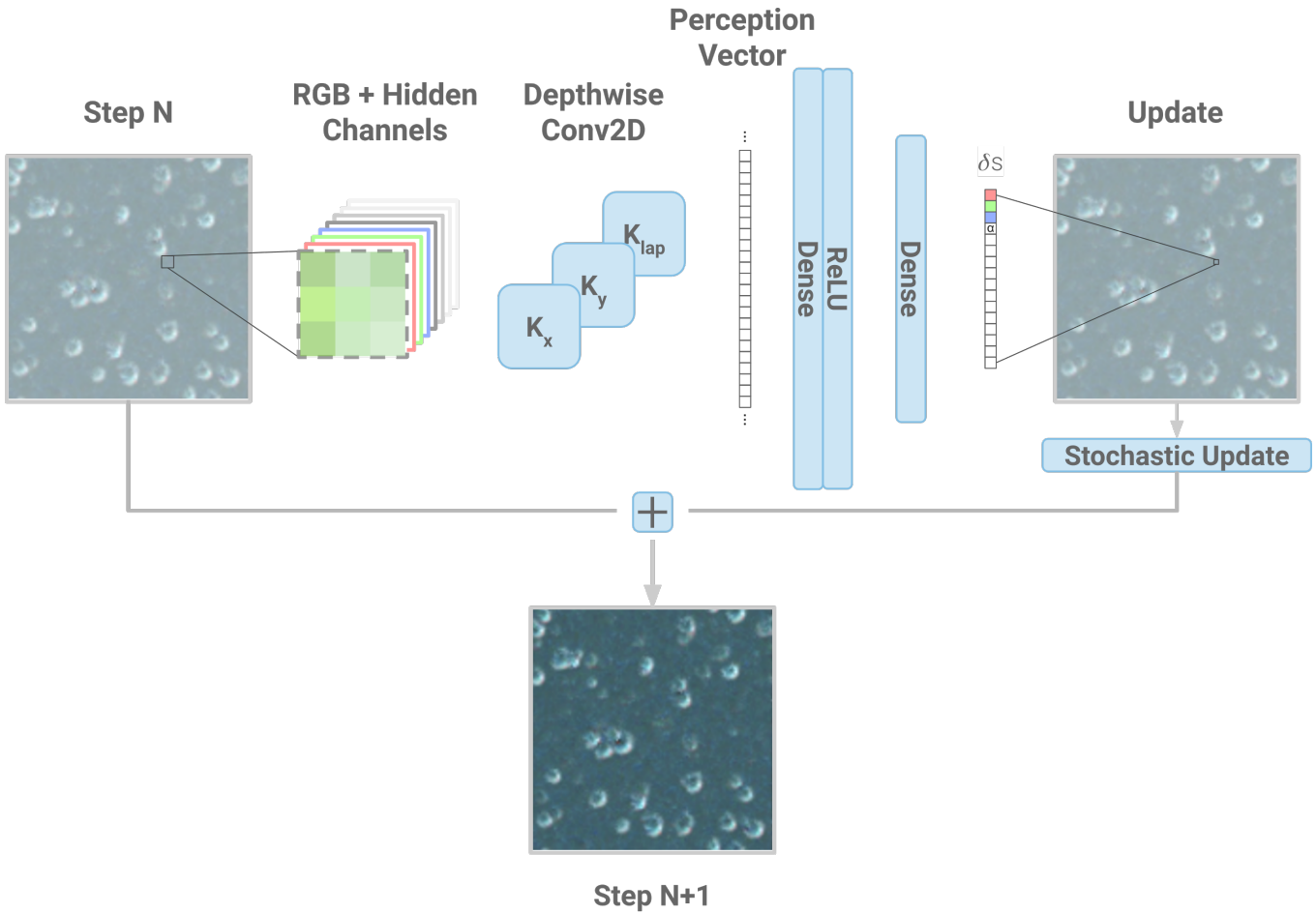
Each pixel has a state $s \in \mathbb{R}^k$ where the 3 first channels are RGB. Their update equation is $\frac{\partial \mathbf{s}}{\partial t} = f(\mathbf{s}, \nabla_x \mathbf{s}, \nabla_x^2 \mathbf{s})$, which becomes in the discrete case :

$$\mathbf{s}_{t+1} - \mathbf{s}_t = f(\mathbf{p}_t)\delta_{x,y,t}$$

- $\mathbf{p}_t = concat(\mathbf{s}_t, K_x * \mathbf{s}_t, K_y * \mathbf{s}_t, K_{lap} * \mathbf{s}_t)$ is the perception vector.
- $\delta_{x,y,t}$ is the update rate.
- K_x, K_y and K_{lap} are discrete derivation filters.

To Neural

The function f is learnt through a NN taking the form $f = relu(\mathbf{p}W_0 + b_0)W_1 + b_1$. The whole NCA becomes a RCNN


Step N
RGB + Hidden Channels
Depthwise Conv2D
K_{lap}
K_y
K_x
Perception Vector
ReLU Dense
Dense
δs
Update
Stochastic Update
Step N+1

The paper uses VGG-16 based Texture Loss (Gatys et al. 2015), which takes the form :

$$\mathcal{L}(x) = \sum_{\text{layers selectionnés } L} w_L \left\| G^L(x) - G^L(u) \right\|_F^2$$

with :

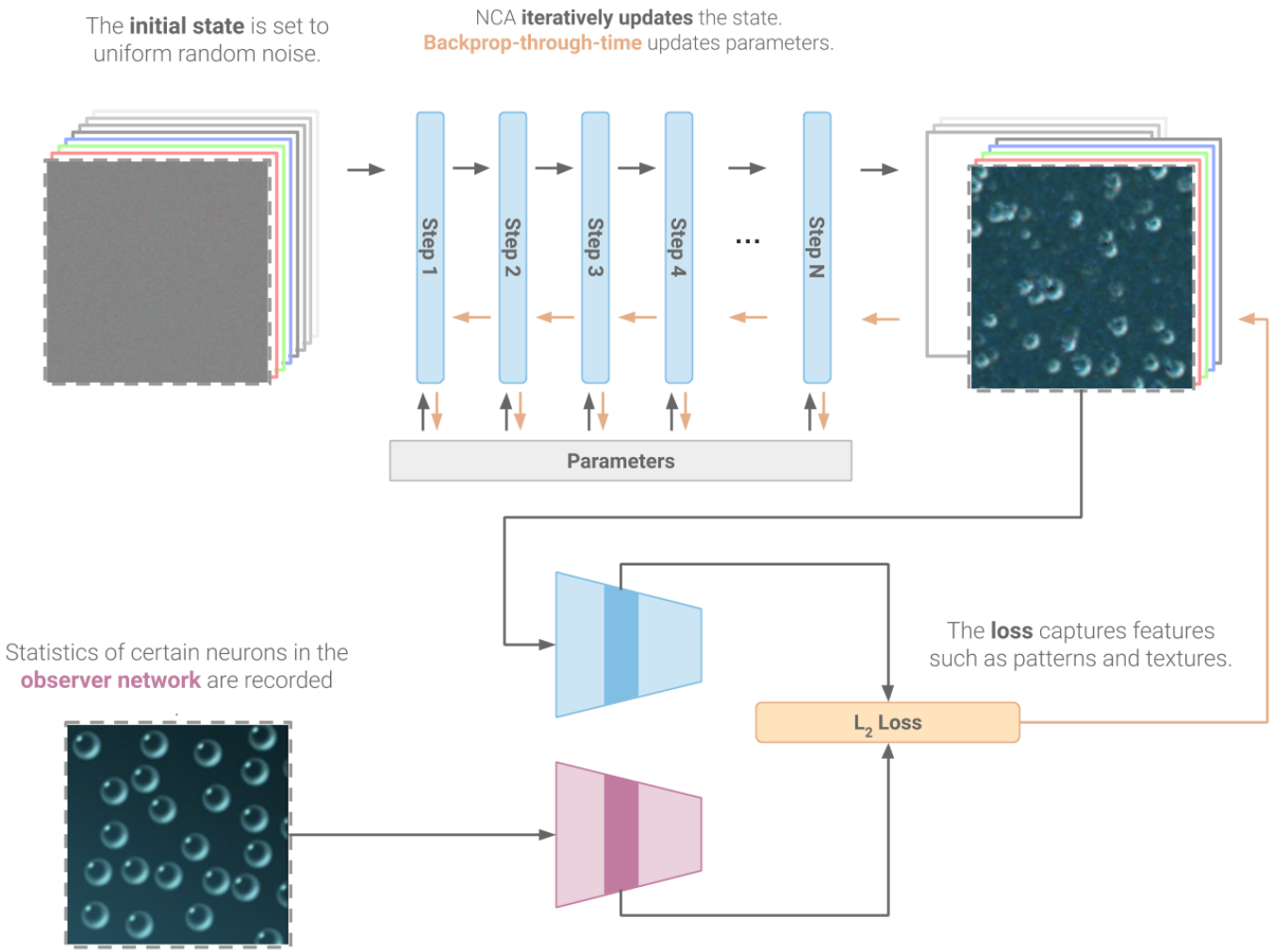
$$G^L(y) = \frac{1}{wh} \sum_{k \in \{0,\dots,w-1\} \times \{0,\dots,h-1\}} V^L(y)_k V^L(y)_k^T \in \mathbb{R}^{n \times n}.$$

Algorithm 1 Implementation of Neural Cellular Automata Texture Synthesis

Input: A texture example of size (3, H, W) and a NCA architecture

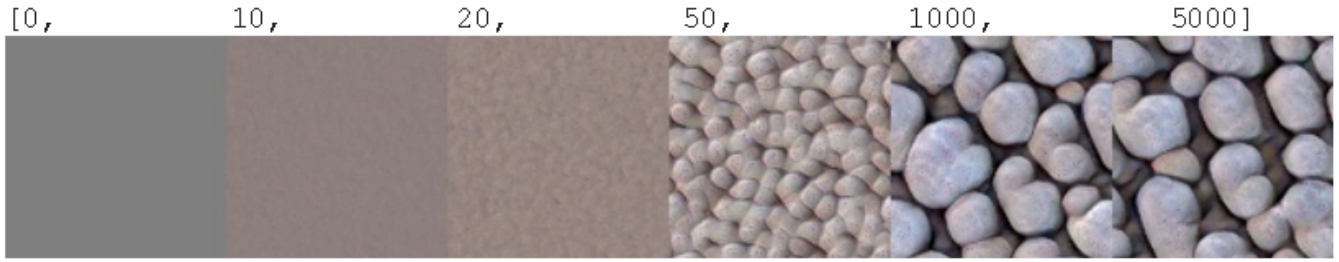
1. Randomly initialize a grid states pool of size (B, C, H, W).
2. Iterate until loss converge:
   - Sample b grid states from the pool and regularly reinitialize one
   - Iterate the current NCA on $N \overset{i.i.d.}{\sim} \mathcal{U}(\llbracket 32, 96 \rrbracket)$ steps for each grid states
   - Take first 3 channels of current grid steps as RGB for texture proposals
   - Compute VGG texture loss between proposals and example
   - Take gradient step on NCA parameters

Output: Dynamical process generating alike textures


The **initial state** is set to uniform random noise.
NCA **iteratively updates** the state.
Backprop-through-time updates parameters.
Step 1
Step 2
Step 3
Step 4
...
Step N
Parameters
Statistics of certain neurons in the **observer network** are recorded
The **loss** captures features such as patterns and textures.
L_2 Loss

We implemented the NCA on PyTorch (using a simplistic implementation provided by the authors).
Original image :

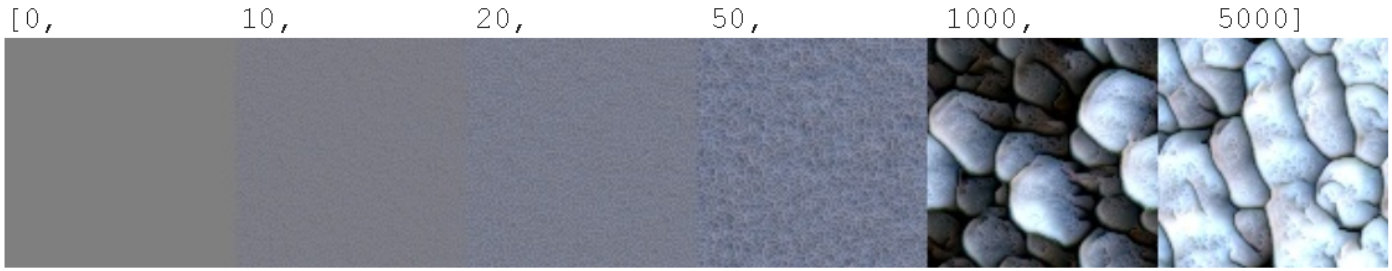
Results :



Organic behaviour and stable dynamic. (Maybe some colour difficulties).

- Robustness to degradation

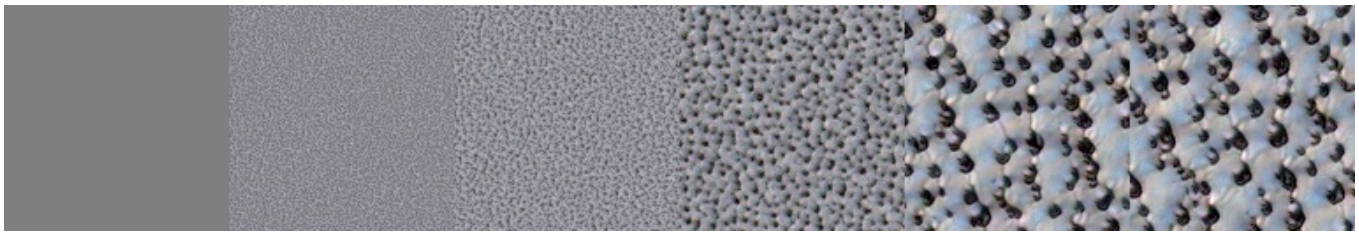
- Number of hidden variables (only RGB below) :



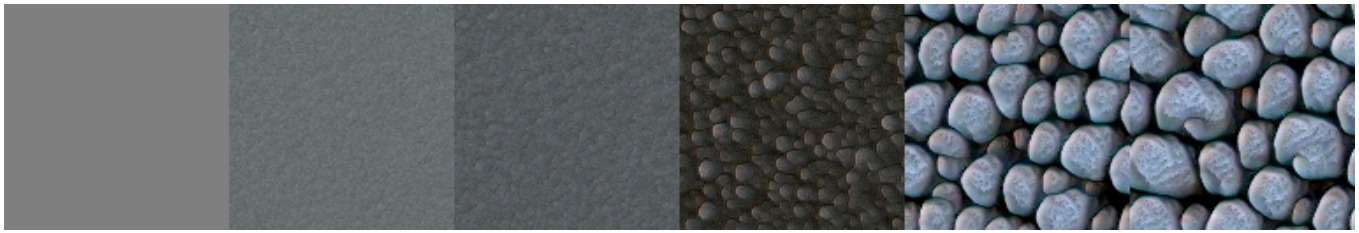
The hidden states of pixels are central in the determining of a good solution (which is a problem for inpainting).

The role of VGG layers

- First layer :



- Last layer :



It appears that the first layers are capturing the color texture of the pebbles while the last ones are capturing the general form : Hierarchical approach for texture synthesis.

Table of content

Texture Synthesis Using Convolutional Neural Networks

1. Neural Cellular Automata
   - Cellular Automata
   - Training procedure
   - First use cases

2. Contributions
   - Rotations
   - Learning Filters
   - Regularising the Loss
   - Texture Inpainting
   - Feature Visualization

3. Conclusion and perspectives

We notice in the algorithm design that the network learned in independent from the update filters. This allows to modify the filters of a trained cellular automata to change the "geometry" of the update equation.

We can for example apply a rotation to the gradients as such :

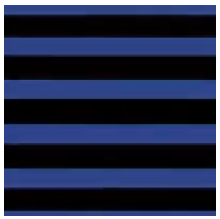
$$\begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} cos(\alpha_{x,y}) & sin(\alpha_{x,y}) \\ -sin(\alpha_{x,y}) & cos(\alpha_{x,y}) \end{bmatrix} \begin{bmatrix} K_x * \mathbf{s} \\ K_y * \mathbf{s} \end{bmatrix}$$

We implemented such rotation for position invariant and position dependent rotation.

Original image

Reconstruction (NCA)

Position invariant rotation



Position depedent rotation

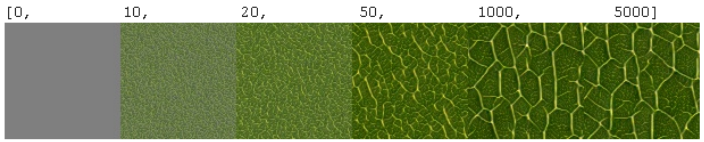
This property allows to change the geometry on which the NCA will behave without having to retrain it on new texture. (sometimes not convincing).

We implemented AdapativeCA, whose derivation filters are now part of optimised parameters. We studied three different modes :

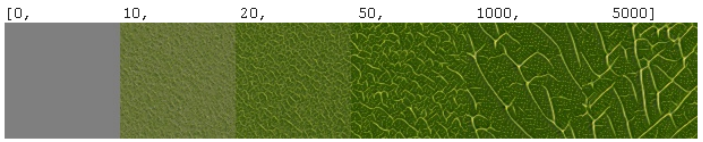
- **Mode 1 :** Only 9 parameters fixed (representing K_x) then we compute $K_y = K_x^T$ and $K_y = K_x^T$ et $K_{lap} = \frac{1}{2}(K_x \star K_x)_{::2} + (K_y \star K_y)_{::2}$
- **Mode 2 :** The three filters K_x, K_y, K_{lap} are learnt.
- **Mode 3 :** The filters are learnt differently for each state channel.

For the first two modes, we start we random initialisation of the weights and classical filters initialisation.

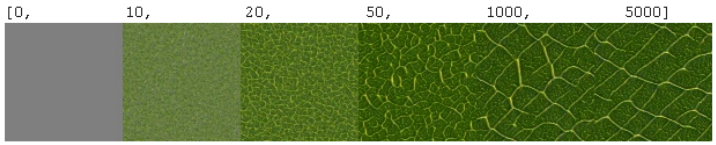
Mode 1 (deterministic initialisation)



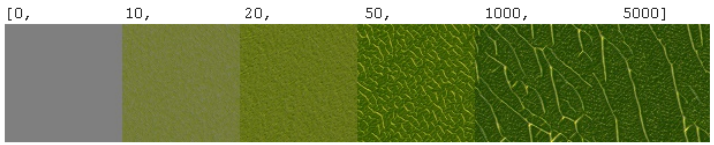
Mode 1 (random initialisation)



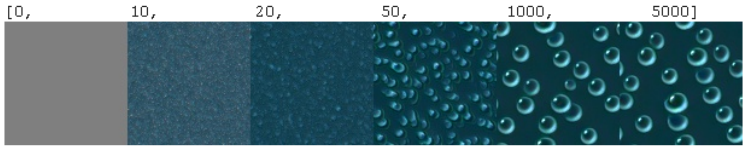
Mode 2 (deterministic initialisation)



Mode 2 (random initialisation)



Mode 1 (deterministic initialisation)



Mode 1 (random initialisation)

We can add some regularisation to the texture loss, to force some desired output properties.

- **Color regression :** We might want the texture to have the same color mean and color covariance as the targeted one. We denote :

$$m = \begin{pmatrix} m_r \\ m_g \\ m_b \end{pmatrix} = \frac{1}{MN} \sum_{t \in \Omega} h(t) \in \mathbb{R}^3$$

$$C_h = \frac{1}{MN} \sum_{t \in \Omega} \begin{pmatrix} h_r(t) - m_r \\ h_g(t) - m_g \\ h_b(t) - m_b \end{pmatrix} \begin{pmatrix} h_r(t) - m_r \\ h_g(t) - m_g \\ h_b(t) - m_b \end{pmatrix}^T \in \mathbb{R}^{3 \times 3}.$$

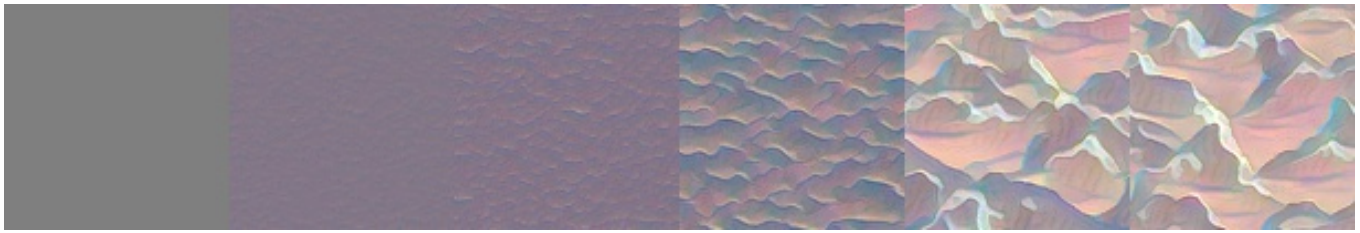
and modify the loss as follows :

$$\mathcal{L} + \lambda_{moy} \|m(x) - m(u)\|^2 + \lambda_{cov} \|C(x) - C(u)\|^2.$$

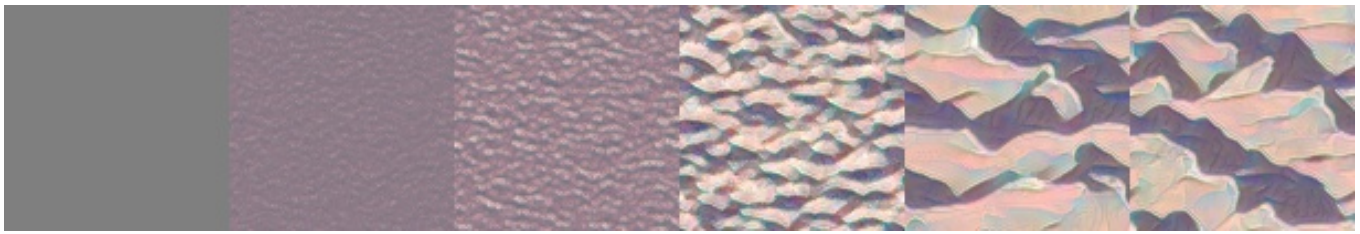
Results for color correction

- Original image :

- Without color correction :



- With color correction :



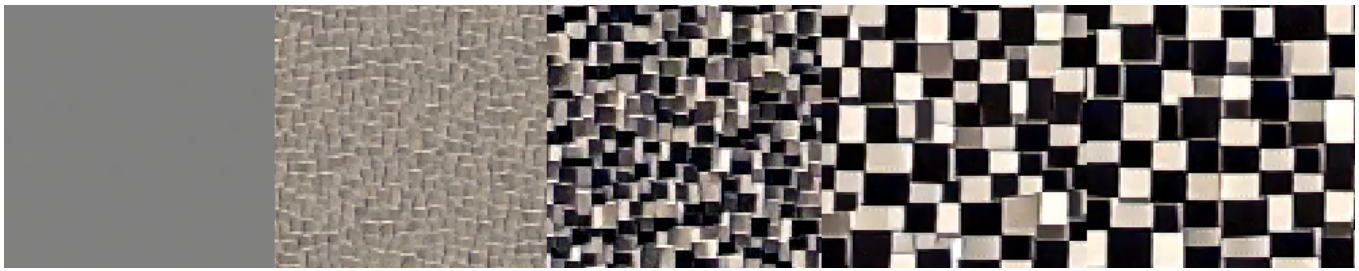
- **Spectral correction :** We might want the texture to have the same periodic property as the original one. Thus, we add the following term to the loss :

$$\lambda_{Fourier} \| |\hat{x}| - |\hat{u}| \|^2.$$

Results for spectral correction

- Original image :

- Without spectral correction :



- With spectral correction :

Texture Inpainting

Task consist of reconstructing in a coherent way a masked part of a texture while preserving the rest of the image.

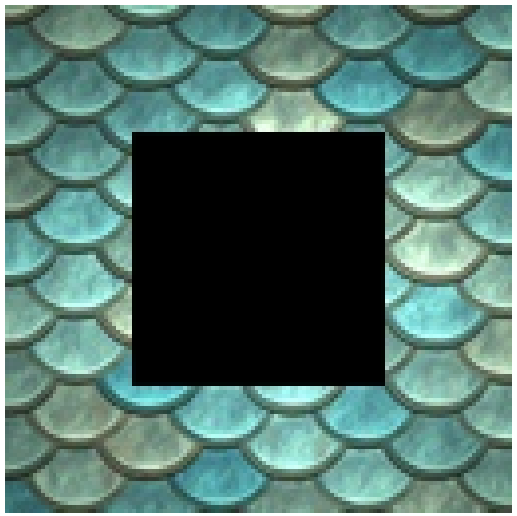
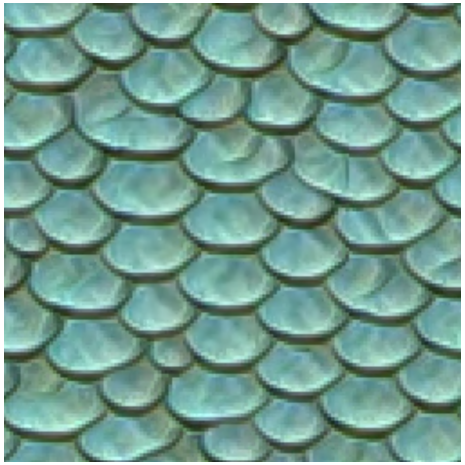


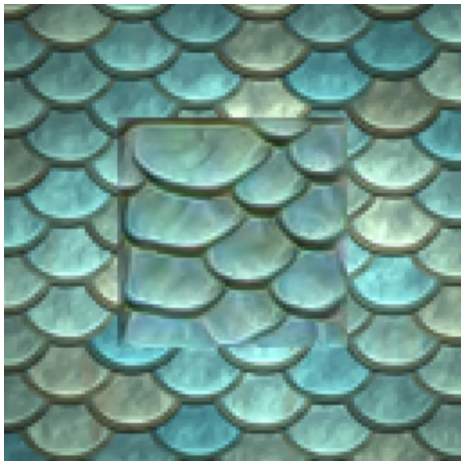
Figure 1: masked texture image

- First train NCA as before except the masked part is not taken into account in the loss to not interfere with texture learning.
- We then want to iterate NCA on the masked part to regenerate it. Unmasked part gives boundary conditions for RGB channels. But what boundary values should we use for underlying channels?
  - average grey
  - NCA iterations then fixation
  - NCA iterations, progressive fixation towards masked part

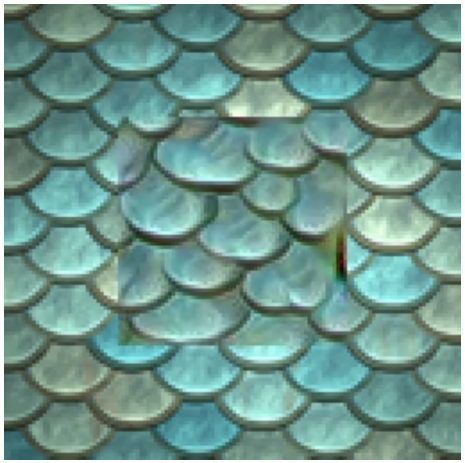
Results for texture inpainting



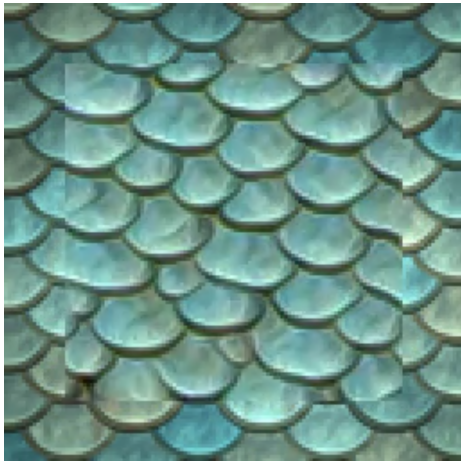
(a) reference: synthesizing unmasked part



(b) average grey



(c) NCA iterations then fixation



(d) NCA iterations, progressive fixation towards masked part

One way to visualise the behaviour of a previously trained neural network is to observe which type of input optimises a certain behaviour in its deep layers.

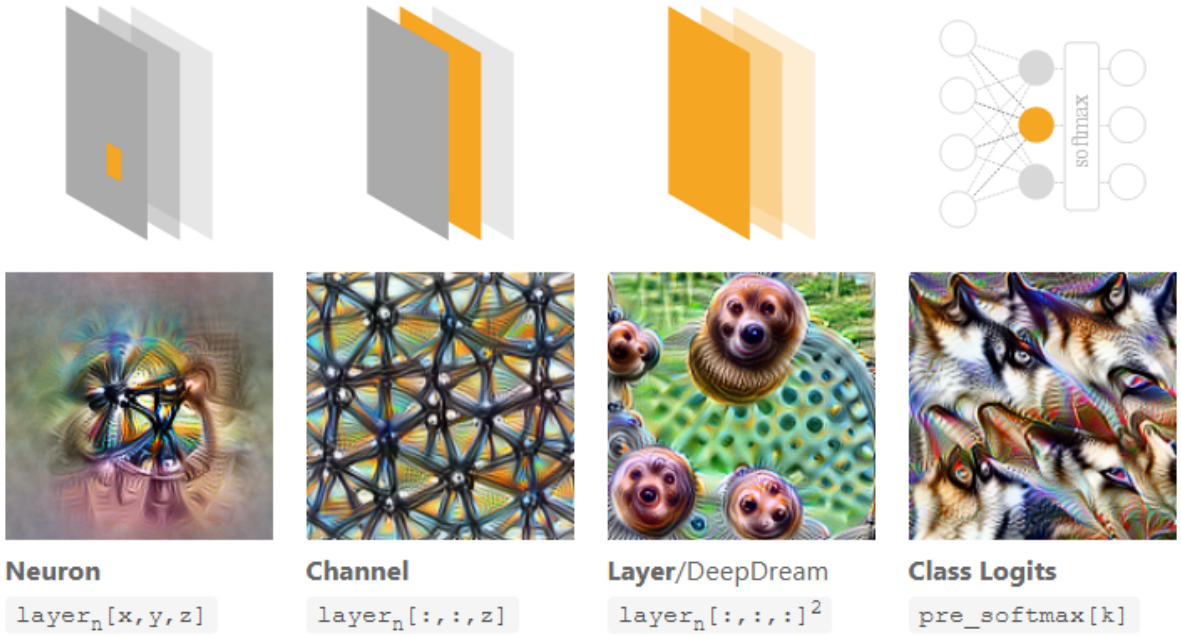


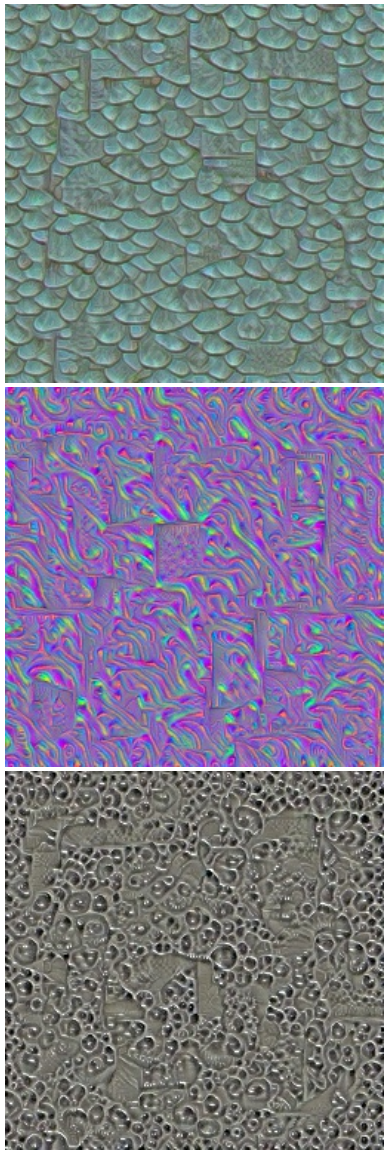
Figure 3: examples of feature visualization objectives

In the following, we are interested in two different objective functions.

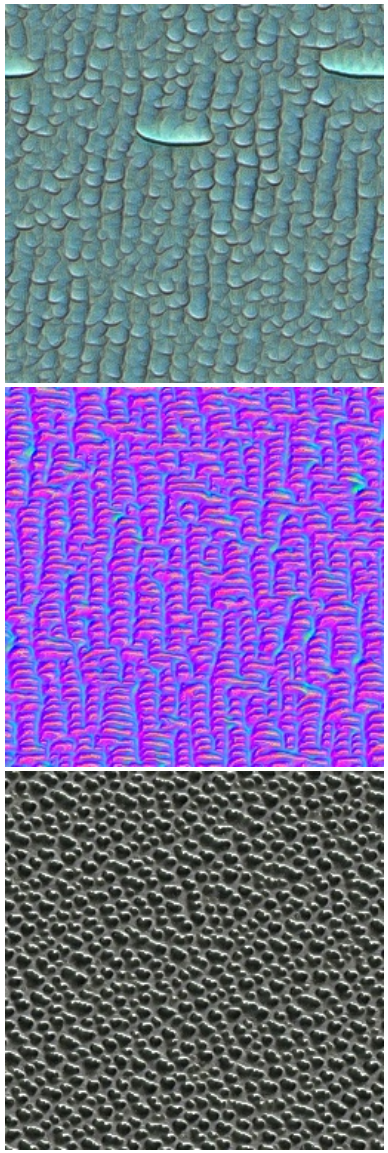
- minimizing the texture loss after 32 NCA iterations.
- maximizing energy of the last layer of NCA after 32 iterations.

Results for feature visualization

(a) initial texture image



(b) minimizing the texture loss after 32 NCA iterations



(c) maximizing energy of the last layer of NCA after 32 iterations

Table of content

Texture Synthesis Using Convolutional Neural Networks

1. Neural Cellular Automata
   - Cellular Automata
   - Training procedure
   - First use cases
2. Contributions
   - Rotations
   - Learning Filters
   - Regularising the Loss
   - Texture Inpainting
   - Feature Visualization
3. Conclusion and perspectives

Strengths

- Computationally efficient method for generating texture (parallelizable, reusable and few parameters)
- Loss adaptable to address precise issues
- Concept of NCA as a whole seems a good way to capture local interactions.

Weaknesses

- NCA is a local model. Can not reproduce images with global coherence, e.g. faces.
- To do inpainting, you need to find coherent boundary conditions!

Perspectives

- What is the theoretical form/properties of the solution space?

References

Gatys, Leon A. et al. (2015). *Texture Synthesis Using Convolutional Neural Networks*. arXiv: 1505.07376 [cs.CV].

Mordvintsev, Alexander et al. (2021). *Texture Generation with Neural Cellular Automata*. arXiv: 2105.07299 [cs.AI].

Thank you!