Visual texture synthesis with convolutional neural networks

Romain Petit

January 2019

Abstract

This project aims at studying the exemplar-based texture synthesis problem. Given observations of a texture, it consists in synthesizing a new signal that is pointwise different, but have the same perceptual properties. This task has many applications in computer graphics, but is also of great interest in the understanding of perception mechanisms, and has strong connections with the representation of stationary stochastic processes.

A popular approach to adress this task is to find a set of features that provide a compact representation of the process, and then synthesize the new signal by sampling from a distribution yielding the same features' value than the original signal.

The main contribution of the algorithms we study here, which only deals with visual textures, is to make use of features extracted from convolutional neural networks, that are known to provide rich image representations after being trained on large datasets. The article describing this new technique does not provide new theoretical insights on stochastic processes' representations, but rather gives empirical evidence that using the proposed features allows to capture the perceptual information in visual textures.

---

1 Introduction

To begin with, it shall be stressed that formally defining the exemplar-based synthesis problem, which consists in producing a texture that have identical perceptual properties than an observed one, is a challenge itself. This formulation indeed refers to two ill-defined notions : textures and perceptual similarity. According to Yves Meyer, textures are a "subtle balance between repetition and innovation". In computer vision, it is standard to model them as realizations of a stationary stochastic process. Since we will here focus only on visual textures, this is the viewpoint we will adopt. However, there is no consensus on the definition of perceptual similarities. As a result, the evaluation of synthesis algorithms remains purely qualitative. We are therefore left with a problem that can be reformulated as follows : given observations that are assumed to be sampled from a stationary process, how to generate a new signal while preserving pereceptual similarity ?

Two main families of algorithms to perform visual texture synthesis can be distinguished : the constraints-based and the resampling methods. The former focuses on findind relevant features that provide a compact representation of the observed process, and then perform synthesis by sampling a distribution yielding the

same value for these features than the original signal. The latter starts with a small randomly chosen area of the original texture, and then grows a new image by sampling from the conditional distribution of a pixel given all its neighbors synthesized so far, which is estimated on the original texture.

This last method, which was introduced in [Efros and Leung, 1999], is known to produce impressive results, but was originally extremly slow, and does not provide a real model for textures, but rather an algorithmic synthesis procedure. Variants have been proposed to speed up computations. They mainly rely on the idea of performing synthesis at patchs level, instead of working directly on pixels. A well-known example of these variants was introduced in [Efros and Freeman, 2001], and [Wei et al., 2009] provides a review of methods published before 2009. It shall be stressed here that these improvements made resampling techniques very effective and efficient, and that they should be preferred for applications in, for example, computer graphics, where the aim is only to produce high quality visual results. The main interest one can have in the constraints-based algorithms is that, unlike resampling methods, they do provide an actual parametric model for textures.

In this project, we will focus on a method belonging to the constraints-based family. They rely on the idea, introduced in [Julesz, 1962], that the perceptual information of a texture is contained in a finite number of statistics, such that two images sharing the same value for these statistics would be perceptually equivalent. A natural synthesis procedure can be derived from Julesz hypothesis : given a collection of statistics computed on a given image, find the set of distributions yielding the same value for these statistics, and sample from one of these distributions (more details on the choice of this distribution will be given later). The two main challenges while considering such methods are to find relevant statistics, that capture all perceptual information, and find an efficient sampling scheme to perform the synthesis.

In [Julesz, 1962], second order moments were presented as sufficient to summarize perceptual information. However, it soon appeared that higher order moments were required to fully represent some kind of stationary processes (for example non-Gaussian ones), and finding a suitable collection of statistics is still an active topic of research. Two main developments were made since Julesz's work. First, Markov random fields were proposed as a framework in which synthesis can be performed by sampling a distribution whose parameters are inferred from the original texture. The second idea was to use filter banks to extract a set of outputs from images, and then compute statistics on these responses' value.

Markov random fields have been for a long time a very popular tool in image analysis. However, their application to texture synthesis suffers from several drawbacks. Modeling an image as the realization of a Markov random field assumes that the distribution of a given pixel only depends on a set of neighbors. Increasing the size of this neighborhood allows to take higher order statistics into account, but also makes their estimation exponentially more complex. This thus prevents in practice the model from capturing long range dependencies.

The method we study here relies on statistics computed on images' responses to a filter bank. So far, [Portilla and Simoncelli, 2000] was probably one of the best performing of these algorithms. It relies on a steerable pyramid, which is a wavelet filter bank, and a set of hand-crafted statistics inspired from biological vision and audition. In [Bruna and Mallat, 2013], the use of the scattering transform for audio texture synthesis was investigated (see [Bruna, 2013] for a comprehensive presentation of the theory and its application to texture synthesis). This signal representation has been designed to have interesting invariance properties, and was successfully applied to several image and audio analysis tasks. Its computation is however not easy to implement, and until very recently, there did not exist any software library allowing to easily differentiate the transform.

In [Gatys et al., 2015], which is the method we will focus on, the authors propose to use representations learned by training convolutional neural networks on large image databases. Features extracted from these

networks are well-known for providing state of the art representations in a wide variety of computer vision tasks. Statistics can easily be computed on filters' activation yielded by passing an input image through the network. Aside from the outstanding results it produces, one of the interests of this technique is that its implementation in deep learning libraries is straightforward. These software packages are extensively used by a large community, and are thus well maintained and documented. Their main interest is to allow computations to be parallelized on graphics processing units (GPUs), which results in drastic speed ups. Even if a quantitative performance comparison with other methods can not be performed, this synthesis algorithm seems to show substantial improvements compared to other constraints-based techniques. The article's findings are purely experimental, and it does not provide any theoretical performance guarantee.

It should be noted that in December 2018, a new software package [Andén et al., 2018] providing a PyTorch implementation of the scattering transform was released. It would be interesting to use this new library to compare scattering transform-based synthesis algorithms and the method studied here. One of the main advantages of this transform is that its theoretical properties have been extensively studied. On the contrary, representations learned by training convolutional networks on large databases are still poorly understood.

2 Presentation of the synthesis method

As explained in the introduction, the synthesis method we study [Gatys et al., 2015] relies on features extracted from convolutional neural networks. These networks are built by composing several basic operations, which are in our case convolutions, poolings, and non-linear activation functions. Each layer of the network consists in convoluting or pooling the output of the previous layer, and then applying a non-linear activation function. Features extracted to perform texture synthesis are exactly the output of certain layers, which are empirically chosen to maximize the algorithm's performance. Formally, if the output of the l-th layer (or the l-th collection of feature maps) is denoted F_l, we have that $F_{l+1} = \rho(W_l F_l)$ where W_l denotes the linear operation performed at layer l (either a convolution or a pooling), and ρ the non-linear activation function.

Figure 1 provides a schematic description of the synthesis algorithm, which is as follows : the original texture is passed through the network, and layers' outputs are stored in feature maps $(F_l)_{l=1,\dots,L}$. The synthesized image is initialized (for example with a white noise), and then iteratively passed through the network. At each iteration, feature maps $(\hat{F}_l)$ and a perceptual loss $\mathcal{L}(F, \hat{F})$ are computed. The synthesized image is then modified to decrease this loss's value.

It shall be stressed that this optimization process does not have anything to do with neural network training. The network is pre-trained on a large image database (in our case ImageNet), and the only variables over which the optimization is performed are the synthesized image's pixels. As a consequence, it is straightforward to understand that the complexity of the synthesis process is directly linked to the size of the output image.

The perceptual loss measures the difference between statistics computed on the original and the synthesized image's features. As explained in the previous section, these statistics are designed to capture all the textures' perceptual information. In [Gatys et al., 2015], authors chose to use correlations between feature maps within a given layer (or Gram matrices). The Gram matrix corresponding to layer l, denoted $G^l = (G^l_{ij})$, is such that $G^l \in \mathbb{R}^{N_l \times N_l}$, where N_l is the number of filters in layer l. Its coefficients are defined as follows :

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk} \tag{1}$$

with F^l_{ik} the activation of the i-th filter in layer l at position k.[1]

[1] To be more precise, in two dimensional convolutional networks, feature maps are usually seen as tensors of size (N_l, H_l, W_l),

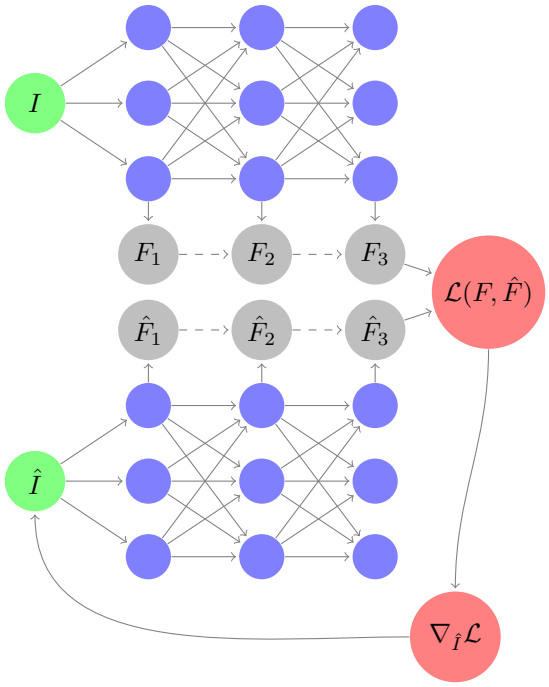


Figure 1 – Schematic description of the synthesis method

Gram matrices $(G^1, ..., G^L)$ provide a collection of statistics which the authors claim to be a good representation to capture visual textures' perceptual information. The total loss is then defined as :

$$\mathcal{L}(G, \hat{G}) = \sum_{l=1}^{L} w_l E_l \tag{2}$$

where w_l is the weight given to layer l (set to 0 or 1), and

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2 \tag{3}$$

where N_l is the number and M_l the size of features maps in layer l (i.e. $F_l \in \mathbb{R}^{N_l \times M_l}$). Its gradient with respect to the synthesized image $\hat{I}$ can be easily computed by backpropagation, which is efficiently implemented in deep learning libraries. In the end, this allows to iteratively modify the image to synthesize to decrease the perceptual loss at each stage.

The authors' claim that Gram matrices provide a good representation of textures is mainly motivated by empirical synthesis results, but these statistics however have an interesting property, which is that they do not take into account the spatial information contained in feature maps. The collection $(F_{i,k}^l)_k$ indeed corresponds to the response of different areas of the image to the i-th filter of layer l. Spatial information is thus encoded in feature maps by the position index, over which we sum in equation 1. This ensures that Gram matrices are invariant to a permutation of feature maps over their position index. Since textures are modeled as stationary processes, this kind of behaviour is desirable for a representation of the process.

where H_l and W_l are the height and width of feature maps at layer l. Since we flatten these last two dimensions to compute Gram matrices by performing matrix multiplication, it is convenient to denote $M_l = H_l \times W_l$ the size of feature maps and to write $F_l \in \mathbb{R}^{N_l \times M_l}$. The position index therefore accounts for vertical and horizontal position.

3 Re-implementation and experimental results

To perform my experiments, I decided to re-implement the synthesis method in Python using the PyTorch library, while the original code used the Caffe framework. My re-implementation, which is inspired by a PyTorch tutorial for style transfer [Jacq, 2017], can be found here. The main difference with Leon Gatys's implementation concerns the pixels' intensity management during the optimization. The original code used a box-constrained optimization algorithm (namely L-BFGS-B [Zhu et al., 1997]), which allows to keep the pixels' intensity between 0 and 1. Since, to my knowledge, PyTorch does not provide an implementation of this algorithm, I used an unconstrained variant (namely L-BFGS [Liu and Nocedal, 1989]), and added a post processing step at the end of the optimization. It consists in scaling the intensities between 0 and 1 with an affine transform, and then performing histogram specification, such that the original and the synthesized images have approximately the same color palette.

Their are numerous ways of performing histogram specification on color images, and a lot of work has been dedicated to find color transfer methods that do not produce undesired artifacts. This is however completely out of this project's scope, and for this reason I decided to rely on a simple approximate algorithm. It consists in matching the histogram of each color channel separately. This is done by computing the cumulative distribution function of each channel, for both images, and then approximately inverting the reference one. This allows to compute a change of contrast that ensures the image's histogram will be close to the target histogram.

The network used is a 19 layers VGG [Simonyan and Zisserman, 2014], pre-trained on ImageNet. As the network was trained on images with each color channel normalized by mean $(0.485, 0.456, 0.406)$, the same pre-processing must be applied to input images. Like in the original article, max pooling layers were replaced by average poolings, and layer weights used to define the loss in equation 2 were set to 1 for the first four pooling layers and the first convolution layer, and 0 for the others. The authors report having scaled the network filters' weights to constrain mean activations (see the last paragraph of section 2 in [Gatys et al., 2015]), but they provide almost no details or explanations on this matter. After a bit of research, it seems that they computed the mean activation of each filter across positions and all ImageNet validation images, and scaled weights in order to constrain these means to be 1. I chose not to perform such scaling since my implementation already produced results similar to Gatys's. Another reason for doing so is that [Ustyuzhaninov et al., 2017] (which will be discussed in details later) reports impressive results with filter weights randomly drawn, without any particular scaling.

Preliminary experiments suggested that partial failures occur for textures with strong structure, such as quasi periodic textures. To support this claim, I decided to compute images' autocorrelation, which quantifies the correlation between a signal and translated versions of itself. Given an image I, I computed the following quantitiy :

$$\rho(\tau_x, \tau_y) = \frac{\gamma(\tau_x, \tau_y)}{\gamma(0,0)} \text{ with } \gamma(\tau_x, \tau_y) = \sum_{x=0}^{N_x-1} \sum_{y=0}^{N_y-1} I(x,y) I(x+\tau_x, y+\tau_y) \tag{4}$$

which can be efficiently done using the fast Fourier transform. Indeed, we have :

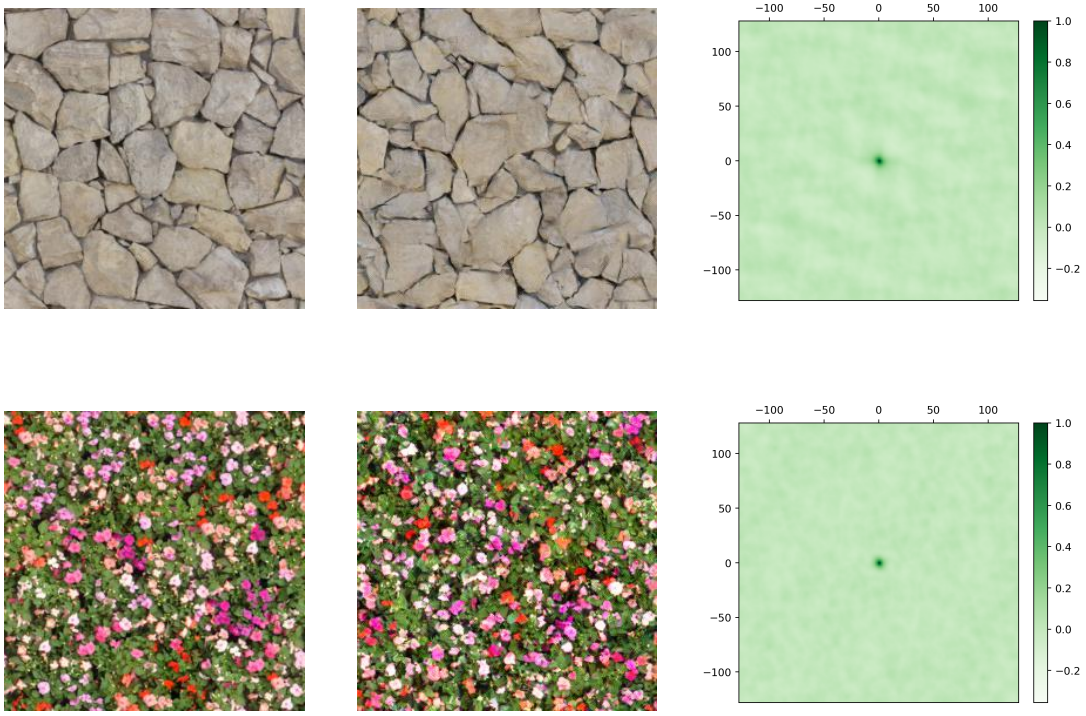
$$\gamma(\tau_x, \tau_y) = I \star I(-\bullet) = \mathcal{F}^{-1}(\mathcal{F}(I)\ \overline{\mathcal{F}(I)}) \tag{5}$$

where $\mathcal{F}$ denotes the Fourier transform. The autocorrelation of stationary processes has the following properties : it is maximal in $(0,0)$ and is symmetric with respect to the origin (i.e. $\rho(\tau_x, \tau_y) = \rho(-\tau_x, -\tau_y)$). The autocorrelation of a periodic signal is also periodic with same period.

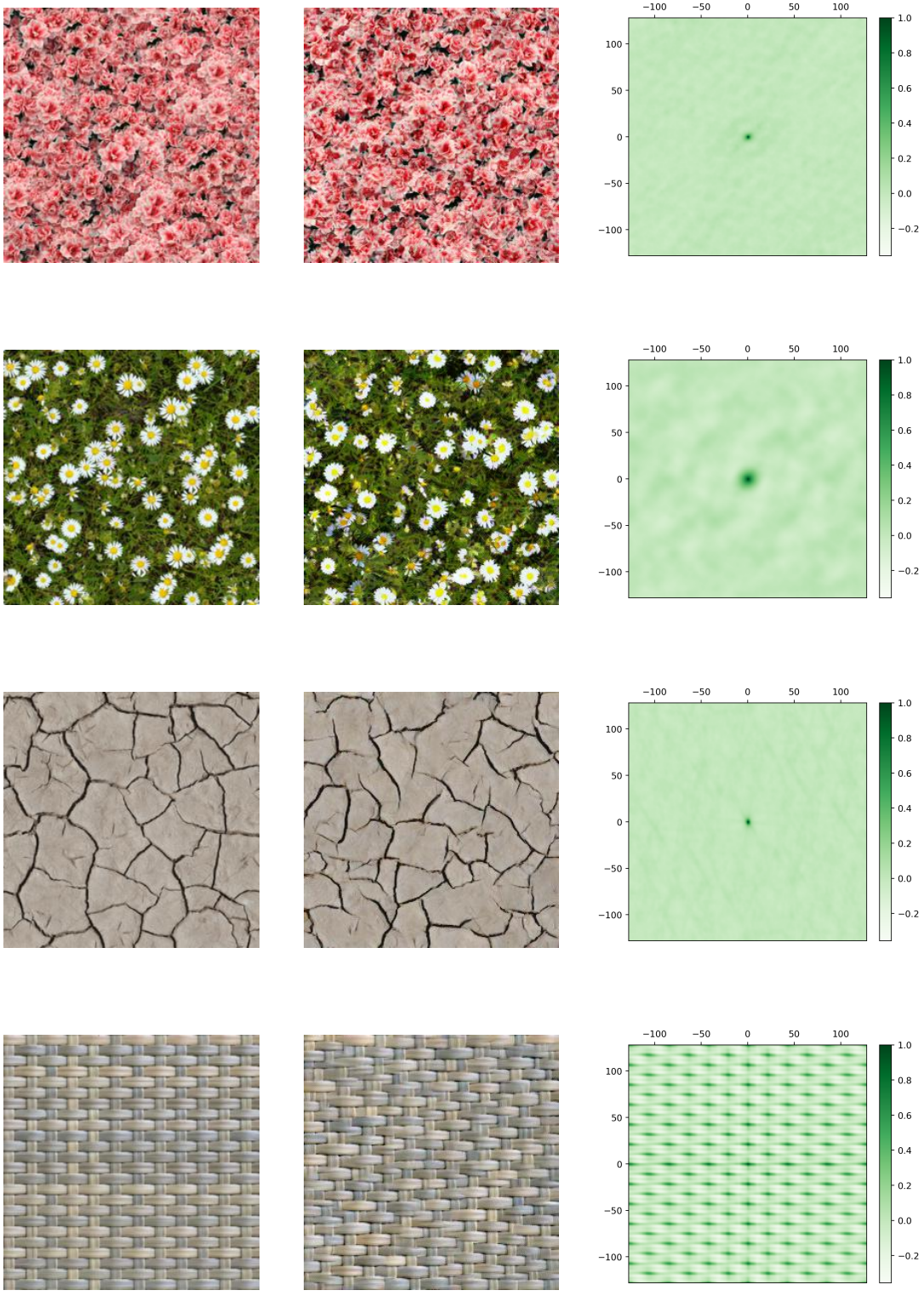
The results shown below were obtained using texture images from this database, that were cropped and resized to be of size 256×256. The perceptual loss was scaled by a 10^9 factor to avoid numerical issues,

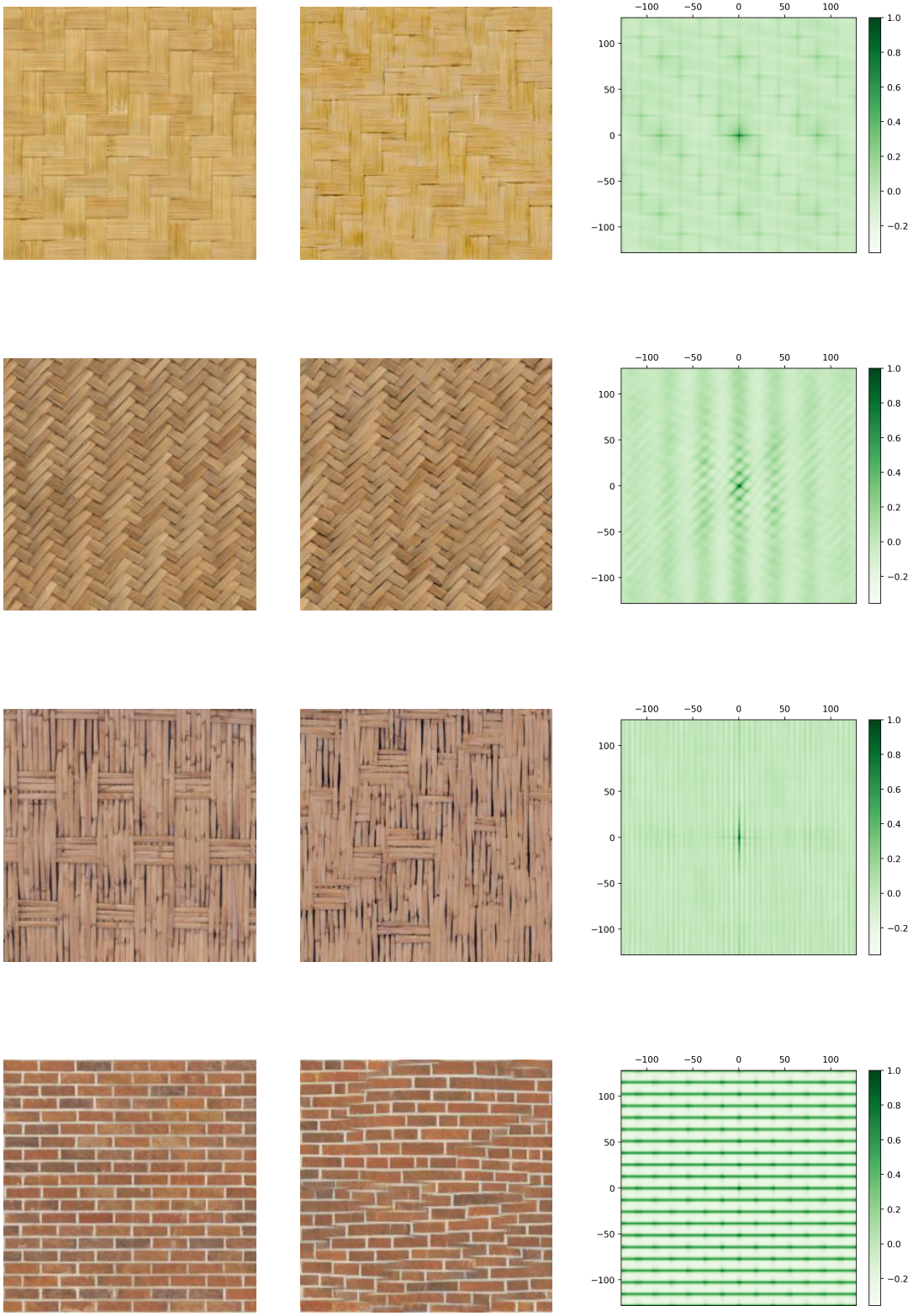
and 2000 optimization steps were performed. Synthesis were all run on a Tesla K40 GPU available through Google Colab, in approximately 10 minutes for each image. Each row of results consists in, from left to right, the original texture image, the synthesis result, and the original image's autocorrelation.
Synthesis results, which, as mentionned before, can not be quantitatively evaluated, seem to be consistent with Gatys's. As expected, partial failure cases exhibit specific autocorrelation patterns, with lots of regularly spaced autocorrelation extrema. On the contrary, the synthesis performs extremly well on textures whose autocorrelation have a unique global maximum at $(0, 0)$. This tends to indicate that the representation fails in capturing high level structures such as periodicity. Surprisingly, the original article does not include any failure analysis, even though the authors' example gallery contains numerous examples of ill-synthesized textures such as described here.

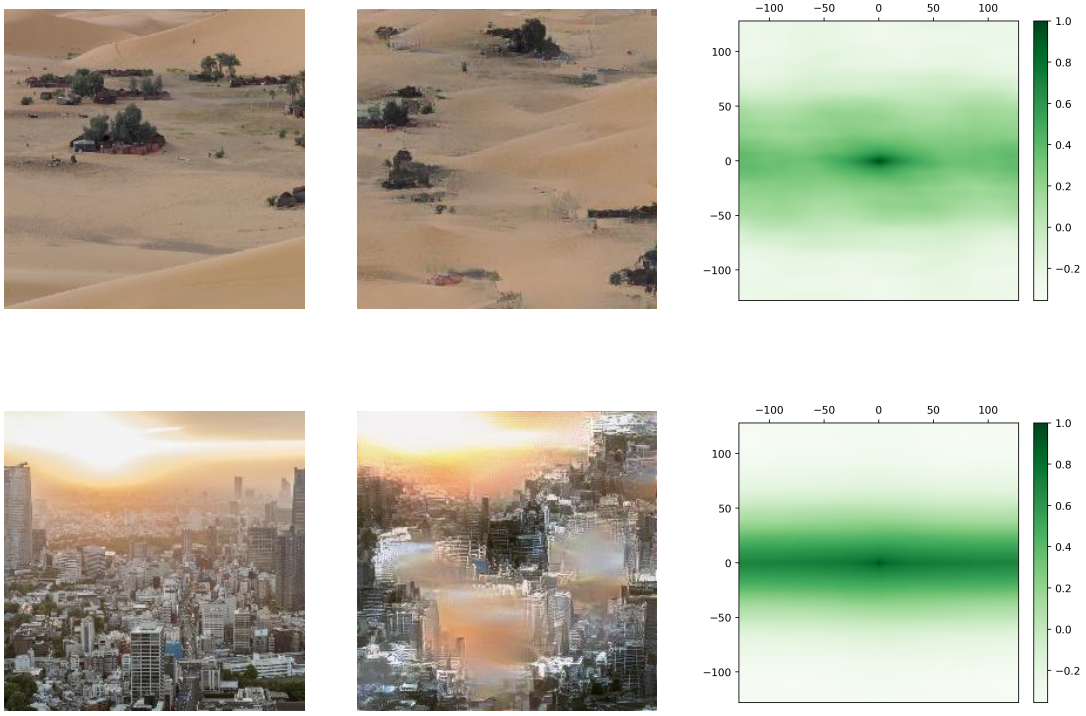
The last two images do not correspond to textures as we defined them, since they do not appear to be even approximately stationary[2]. An interesting thing to study is the sun's position in the last synthesized image. Surprisingly, it is approximately located just like in the original image, suggesting that some spatial information is encoded in the texture's representation. This phenomenon is also reported in the article, on completely different images. In every one of them, the object exhibiting this behaviour is always located on the image's booundary. Even if we saw that, after computing Gram matrices, we cannot seperate the activations of different image areas, that were initially stored in feature maps, it could happen that "features in the network explicitly encode information at the image boundaries", as suggested in [Gatys et al., 2015].



[2]In this case, it shall be stressed that autocorrelation, or at least our way of computing it, is not relevant anymore.


−100
−50
0
50
100
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


−100
−50
0
50
100
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



4 Conclusion and perspectives

The work done over the course of this project can be summarized as follows : the texture synthesis method proposed in [Gatys et al., 2015] was re-implemented using the PyTorch library, and experiments were run on a different database than the one used by the authors. As reported in the article, the method produces impressive visual results. One of its main interests is that it can be quickly implemented in any deep learning library, and can therefore benefit from GPU acceleration and automatic differentiation.

The main finding of this work is that the method is robust to slight modifications and small changes of hyperparameters. Indeed, I found that replacing the box-constrained optimization algorithm by an unconstrained one, and post processing the synthesized image by performing histogram specification did not alter the results' visual quality.

The article however does not provide any new theoretical insight on stationary processes' representation, and little work was done to analyze partial failure cases. Computing images' autocorrelation could be useful in understanding what kind of perceptual information the model fails to capture. Results presented in this report seem to suggest strong structures such as quasi periodicity are particularly difficult to take into account.

Several works have been proposed to improve the method we studied here. In [Berger and Memisevic, 2016], authors enriched the representation with spatial co-occurences of features. This allowed them to show better synthesis results on textures exhibiting strong structures, such as those described above in partial failures' analysis. [Ulyanov et al., 2016] present a variant that allows to perform synthesis using a convolutional

network, but this time in a feed-forward fashion. This greatly reduces the time required to synthesize an image, while producing visualy similar results.

In my opinion, the main limitation of this work is the following : it is suggested that the synthesis performance is closely linked to the representation extracted from convolutional neural networks. However, it was subsequently shown in [Ustyuzhaninov et al., 2017] that similar results could be obtained by using a single layer network with a large filter whose weights were randomly drawn. This indicates that what really captures perceptual information in textures is still unclear. In particular, it shows that neither the hierarchical structure of the representation, nor the fact filter weights were learned on a large image database are necessary to synthesize visualy convincing results.

Finally, I would personally be extremely interested to find out how a recent preprint [Mallat et al., 2018], that I was not able to study, relates to this last work. Its authors, who claim to have been inspired by these findings, derive a new signal representation that allows to reconstruct certain kind of images from a limited number of coefficients. The exact class of signals that can be represented this way is however still unclear.

References

[Andén et al., 2018] Andén, J., Andreux, M., Angles, T., Belilovsky, E., Eickenberg, M., Exarchakis, G., Huang, G., Leonarduzzi, R., Lostanlen, V., Oyallon, E., Thiry, L., and Zagoruyko, S. (2018). Kymatio, wavelet scattering in pytorch. https://www.kymat.io/.

[Berger and Memisevic, 2016] Berger, G. and Memisevic, R. (2016). Incorporating long-range consistency in cnn-based texture generation. *arXiv preprint arXiv:1606.01286*.

[Bruna, 2013] Bruna, J. (2013). *Scattering representations for recognition*. PhD thesis, Ecole Polytechnique X.

[Bruna and Mallat, 2013] Bruna, J. and Mallat, S. (2013). Audio texture synthesis with scattering moments. *arXiv preprint arXiv:1311.0407*.

[Efros and Freeman, 2001] Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM.

[Efros and Leung, 1999] Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *ICCV*, page 1033. IEEE.

[Gatys et al., 2015] Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28*.

[Jacq, 2017] Jacq, A. (2017). Neural style transfer using pytorch. https://pytorch.org/tutorials/advanced/neural_style_tutorial.html.

[Julesz, 1962] Julesz, B. (1962). Visual pattern discrimination. *IRE transactions on Information Theory*, 8(2):84–92.

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

[Mallat et al., 2018] Mallat, S., Zhang, S., and Rochette, G. (2018). Phase harmonics and correlation invariants in convolutional neural networks. *arXiv preprint arXiv:1810.12136*.

[Portilla and Simoncelli, 2000] Portilla, J. and Simoncelli, E. P. (2000). A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Ulyanov et al., 2016] Ulyanov, D., Lebedev, V., Vedaldi, A., and Lempitsky, V. S. (2016). Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, pages 1349–1357.

[Ustyuzhaninov et al., 2017] Ustyuzhaninov, I., Brendel, W., Gatys, L., and Bethge, M. (2017). What does it take to generate natural textures?

[Wei et al., 2009] Wei, L.-Y., Lefebvre, S., Kwatra, V., and Turk, G. (2009). State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*, pages 93–117. Eurographics Association.

[Zhu et al., 1997] Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560.