Visualization methods for convolutional networks and generative applications

Lucas Gerretsen
Ecole Normale Supérieure de Paris-Saclay
94230 Cachan
lucas.gerretsen2@gmail.com

Nathan Grinsztajn
Ecole Normale Supérieure de Paris-Saclay
94230 Cachan
nathan.grinsztajn@polytechnique.edu

Abstract

Deep learning has gained a huge popularity in the last years. Although they have undoubtedly shown great performances on tasks that were before considered to be hard, they have very low interpretability, which may cause some security or ethical issues. In this work, we review some recently proposed visualization methods to better understand deep neural networks. More specifically, we describe a class of inversion approaches and link them to algorithms for generating stylized images or pieces of art.

1. Introduction

Deep Neural Networks have gained a lot of attention in the past years due to recent empirical breakthroughs [5]: in-real-time object detection in images [7], image description through visual-semantic alignments [8], etc ... Deep Learning has found many applications in image processing, Natural Language Processing, and other various fields. One can argue that their success is due to their large abstraction capabilities with millions of parameters, combined with great computational potentials thanks to accelerated computing. However, it is well known that neural networks have few theoretical results and guaranties as compared to other common machine learning models, and their deep structure make them hard to apprehend. Recent studies aimed at improving the understanding of these black boxes by proposing visualization methods.

It is uneasy to understand how a neural network learns classes, and the specific features it uses. If a network is trained to discriminate animals for example, it has to use some internal representation of classes. Which means the network has some insights of what is a cat, for example. In this work, we'll try to visualise such representation by "inverting" convolutional network, and show how it can be used for image generation. We are first going to review some of these visualization methods for better understanding convolutional neural networks: by describing these methods, we are going to show that some of them share a common goal that is to invert information hidden in predictions or feature maps **2**. We will then describe a method which borrows some of these principles for generating artistic images by merging a *content* image with a *style* image **3**. Finally, we will propose a different approach for generating stylized images by class predictions **4**.

2. Deep inside Neural Networks

Recent research articles have presented various approaches to visualize intermediate representations of an image inside deep neural networks. Although these implementations share the same baseline, they widely differ on implementation details (loss, use of a prior, incremental or one-shot process...), which can lead to very different results.

2.1. Model-based inversion

Given a convolutional network trained on some dataset, the problem of "inverting" the model can be summarized with the following formula, where L_1 is a model-based loss (maximizing the activation of a given filter, maximizing the targeted class output...) and L_2 is a regularisation loss.

$$I_0 = \underset{I}{\operatorname{argmin}} L_1(I, model) + L_2(I) \tag{1}$$

From a practical point a view, I_0 can be computed by a simple gradient descent, or more sophisticated optimization algorithm.

In a convolutional network, it can make sens to visualize the first layers by simply plotting them as images, but this approach doesn't work for deeper layers, as it doesn't take into account the image embedding provided by the previous layers. (1) allows to visualize every layer of a convolutional network.

We use it with $L_2 = 0$ and L_1 being the mean output of a particular filter. We try this formula with different filters of inception-v3, varying the depth.

We can see on the Figure 1 that the style to which each of the selected layers fires are very dependent on their depth. From mainly colors (and a little bit of texture) for the first layer, it switches to more and more abstract representation.

The "Mixed0" seems to discriminate between lots of different texture, and the last layer shown seem to use also positions and shapes.

2.2. Class maximisation

Another interesting way to use the formula (1) is to maximise a specific class response. It allows to get insights how the neural network perceives the objects it has to classify, and in particular what internal representations of such objects are stored in memory.

First, we use a gradient descent algorithm (RMSprop) on a randomly generated image. Results are given in Figure 2. We can see that although the pictures are classified as belonging to particular classes, they are very far from looking like realistic pictures of the said classes. This sensitivity is a known weakness of convolutional neural networks, and is at the basis of adversarial attack threats [1].

But we can wonder if the softmax, which is the activation function of the last layer, is not preventing the network to output good class representations. Indeed, the softmax is an activation function which uses the output of every class to predict the belonging probabilities. Therefore maximising toward a specific class could just lead to minimizing toward the other classes.

To prevent such behaviour, we remove the last softmax activation, and replaced it with a simple linear activation. In order to get more realistic image, we use both a total variation loss and the $||.||_6$ norm. This last norm prevents individual pixels to take abnormally high values, as done in [2]. This combination of the two losses can be seen as the L_2 term in the formula (1).

We combine L_2 and L_1 as in (1) with a weight of 10 for L_2 and 1 for L_1. The figures gotten can be see in figure 3. We can see that, despite the poor first results, adding a regularization loss allows to have more realistic images. This means the neural network indeed has memorized strong insights of each classes, that can be revealed through simple gradient descent algorithms.

2.3. Iterative optimization for image reconstruction

Setups proposed by [6] [9] approximate an image, instead of optimizing a model as we have seen in **2.1**.

The goal here is, given a fixed image $\vec{c}$ and its fixed representation $\phi_c = \phi(c)$, to find an image $\vec{x}$ whose representation $\phi(\vec{x})$ is as close as possible to ϕ_c. This can be formulated as:

$$\vec{x}^* = \arg\min_{\vec{x}} L(\phi_c, \phi(\vec{x})) + \lambda R(\vec{x}) \tag{2}$$

where L is a loss function characterizing the similarity between 2 vectors, and R is an optional regularization function. After optimizing **2**, the obtained $\vec{x}^*$ is intended to be close to $\vec{c}$ with respect to the abstraction function ϕ and the loss L.

Some well chosen ϕ can help explore the information that are filtered inside deep neural networks, such as setting ϕ to a specific feature map. Moreover, provided that the whole objective is differentiable with respect to $\vec{x}$, this problem can be iteratively solved by gradient descent optimizers with random initialization for $\vec{x}$.

We set up an experiment to visualize reconstructed images after 200 iterations based on the concatenation of the feature maps at layer l, and setting L to be the Frobenius (or euclidean) distance. Increasing l for the same input image $\vec{c}$ gave images where information had visibly been lost. Fig. **4** shows that most of the information on the details and colors are lost from layer relu1_2 to layer relu5_2 of a pre-trained neural network VGG19 (ie 16 convolutional layers), leaving mostly information on the main shapes.

We also experimented the Total Variation norm proposed by [6] as the regularizer R. We show that the resulting reconstructed images have lower dissimilarity to the original image when we use this regularizer, as shown in Fig. **4**. Indeed, this norm was built by the authors with the intention to be low for realistic images.

3. Neural image stylizer

We are now going to present an artistic application of **2.3** proposed by [4]. Their goal was to create a stylized image $\vec{x}$ by merging feature representations of a *content* image $\vec{c}$ with a *style* images $\vec{s}$. It is also possible to use several style images, but we are describing the case of 1 for simpler notations in the following.

Their method was to use the first (eg 5) abstraction layers of a pre-trained convolutional networks and leave aside the next layers. Given this fixed neural network, fixed images $\vec{c}$ and $\vec{s}$, and an initial image $\vec{x}$, their idea is to iteratively minimize a *custom* loss function with respect to image $\vec{x}$ by gradient descent through the fixed neural network. This method is exactly an extension of [6] that we presented in **2**: more specifically, the first term of eq. 2 is called the *content loss*, and we add to this the so-called *style loss*.

For any image $\vec{a}$, we denote $A^l \in \mathbf{R}^{n_l \times m_l}$ the matrix made of the n_l flattened feature maps of $\vec{a}$ at layer l. This lets us define the content loss and the style loss:

$$\mathcal{L}_{content}(\vec{x}, \vec{c}, l) = \frac{1}{2}||X^l - C^l||_F^2 \tag{3}$$

$$\mathcal{L}_{style}(\vec{x}, \vec{s}, l) = \frac{1}{4N_l^2 M_l^2}||X^l.{X^l}^T - S^l.{S_l}^T||_F^2 \tag{4}$$

The authors proposed this latter loss since they argued that Gram matrices, which can be seen as auto-correlation matrices, capture well the style of an image in practice and forget about the raw contents of images.

Moreover, these losses can be summed over several layers l with different weights, giving more or less importance to the most abstract feature maps. This results in the following

total loss with hyper-parameters $\alpha, \beta, (v_l)$ and (w_l)

$$\mathcal{L}_{total} = \alpha \sum_l v_l \mathcal{L}_{content}(l) + \beta \sum_l w_l \mathcal{L}_{style}(l) \quad (5)$$

As mentioned before, involving only feature maps in the loss gives degrees of liberty to the algorithm during optimization. In the one hand, the content loss for layer $l = 0$ is convex with respect to image $\vec{x}$, so optimizing it would in theory give $\vec{x*} = \vec{c}$. On the other hand, theory and comparative experiments from section **2** showed that putting weight only on the deeper layers would make the task of minimizing the content loss $\mathcal{L}_{content}$ harder, since the search space is larger. This is why recent implementations of the above described algorithm add a third loss term to **5** for regularization purpose, weighted by a hyper-parameter γ, and using the Total Variation norm. This helps to ensure that the obtained image $\vec{x}$ is smooth, to improve its realistic aspect.

In our experiments presented in Fig. **5**, we used a pre-trained VGG19 which has 16 convolutional layers. The content loss took into account outputs of convolutional layers 10 and 14 equally, and the style loss took outputs of convolutional layers 1, 3, 5, 9 and 13 with decreasing weight of discount factor 0.7. Initial images $\vec{x}$ were set to $\vec{c}$ with negligible noise to vary final outputs, and images we show are the result of 500 iterations of Adam optimization. Ratios between the 3 terms of the loss were tuned and set so that $\beta = 3.10^3\alpha$ and $\gamma = 10\alpha$.

A great advantage of this method is that it is fast to set up since the neural networks used can be trained from any supervised task: we indeed only need their abstraction layers, and not the final fully connected layers which are dependent on the classification task.

4. Image generation with neural network reversion

4.1. Deep Dream

Here we explore yet an other application of the formula (1) : Deep Dream. It is an unsupervised image generation technique first described by Google in [3].

It basically consists in choosing $L_2 = 0$, and L_1 as the weighted sum of the norm of several filter outputs. The optimisation technique is simply a stochastic gradient descent.

What changes from the layer visualization technique we used in the first part is that:

- It combined several filters outputs instead of choosing only one.
- It maximising its square norm, instead of the coefficient sum.
- It uses a simple stochastic gradient descent.
- It uses some additional rendering tricks, known to produce better-looking outputs. It uses jittering (so that the model doesn't saturate at some particular places of the picture), and combines it with zooming-dezooming cycle with the same purpose.

It is unsupervised in the sens that depending on the filters and the weights chosen, the result widely varies. Our results can be seen in 6. When superficial layers are selected, we get new low-level features, like motifs, abstract patterns and color changes. When using deeper layers, we can see high-level identifiable features like eyes in (d).

4.2. "Lucid" Dreams

Deep Dream has the drawback of being unsupervised, and thus it is very uneasy to control how an image is going to change according to some particular layers. This necessitates lots of trial and errors, the vast majority of generated pictures being uninteresting. Here, we try to combine the insights of Deep Dream, which are the jitterring and the zooming, to the class generation method described in the first section.

This allows to have some control over the output image: the shape and features that will be generated depend on the class chosen. Our results can be seen on Figure 7.

We can notice several things. First, the network uses the original image and twists some of the object into bird-like shapes, like the tree in the background. We can also see that the pictures seem to converge, since there are very little variations between the last two images.

This process could probably be finetuned, since there are a lot of parameters to take into account (optimizer used, jittering and zooming, loss weights...).

5. Discussions

In this work, we have looked at several ways of using the formula (1) to invert the representation of a neural network.

We've seen that this formula can be used to get some insights about the internal representations stored in the memory of a network, and can shed some light on how the network converts training data into knowledge. We've also seen that these representations are noisy and very different from how humans see the world, which can lead to potential adversarial attacks.

In the following part, we've seen how the same principle, with additional tricks, can be used to generate artistic images, using the knowledge encoded in the neural network weights. Some of these techniques are well known, like stylizers or Deep Dream, and some are new trials which can probably be finetuned, like Lucid Dream.

It would be interesting also to use this technique with a network pre-trained on Imagenet and finetuned for an other task, to see if the internal knowledge learnt by the network during the finetuning phase suffers from the initial training

on other classes. If it is not the case, it would also allow to generate yet more shapes associated with the new classes.

On the whole, this work aims at showing how reversing neural network can be use to understand better how deep neural networks work, and at raising question about a possible use of computer vision in art in general.

References

[1] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *CoRR*, 2018.

[2] R. Kotikalapudi and contributors, *Keras-vis*, `https://github.com/raghakot/keras-vis`, 2017.

[3] C. O. Alexander Mordvintsev and M. Tyka, *Inceptionism: Going deeper into neural networks*, `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`, 2015.

[4] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, 2015.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, no. 7553, p. 436, 2015.

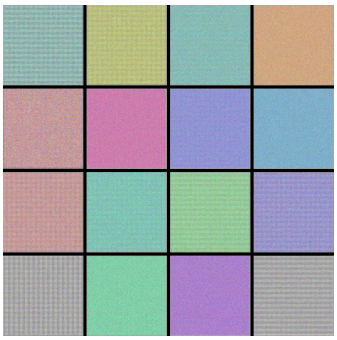
[6] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5188–5196.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

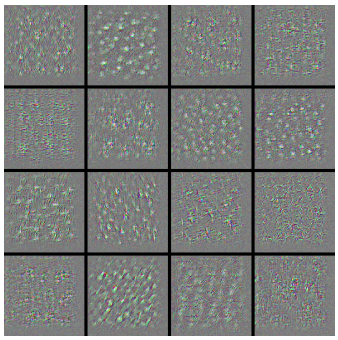
[8] A. Karpathy and F. Li, "Deep visual-semantic alignments for generating image descriptions," *CoRR*, 2014.

[9] K. Simonyan, A. Vedaldi, and A. Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps*, 2013.

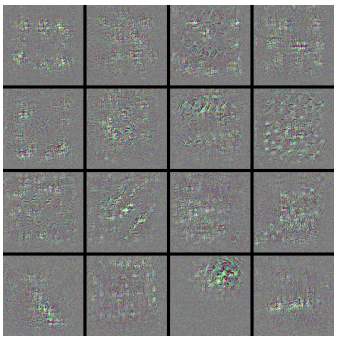
Appendix



(a)

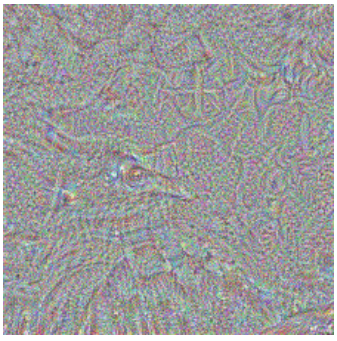


(b)

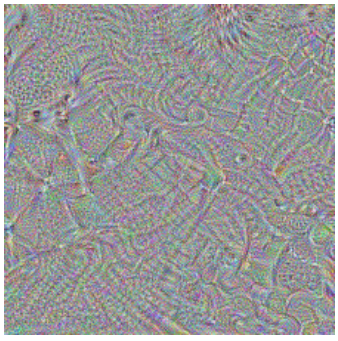


(c)

Figure 1: Image reconstruction of several filters of inception-v3 trained on ImageNet. (a): first filters. (b): the "Mixed0" layer, roughly one-third deep in the network. (c): "Mixed4" layer, one of the last layers.

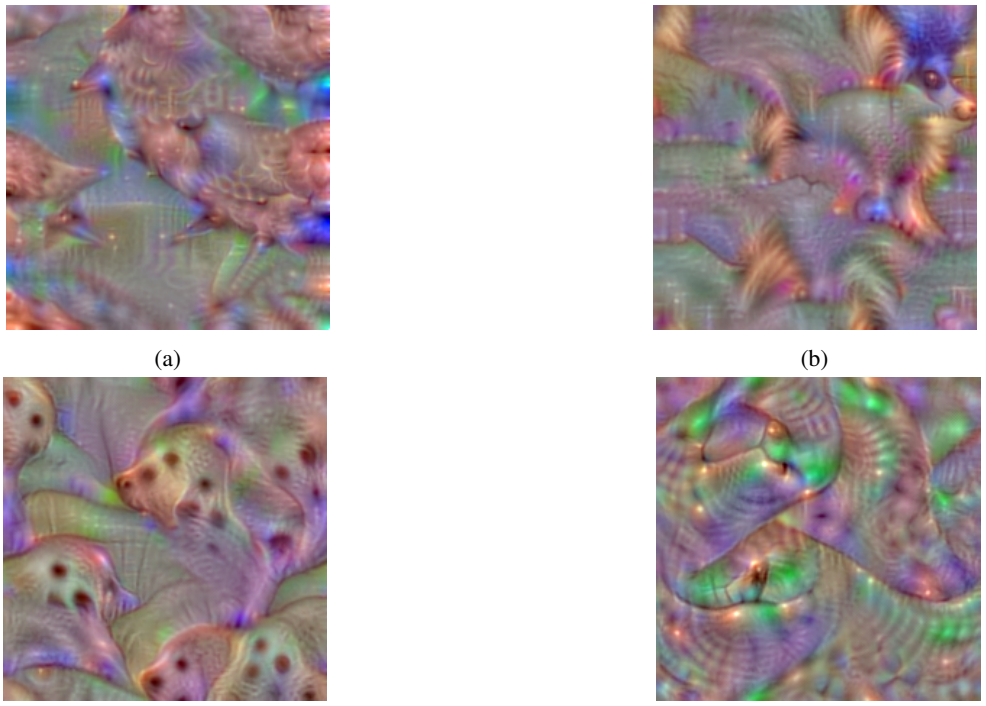


(a)



(b)

Figure 2: Reconstruction of two classes of inception-v3 trained on ImageNet, from random noise. (a): index 20, water ouzel, probability: 0.999. (b): index 251, dalmatians, probability: 0.999.



(a) (b) (c) (d)

Figure 3: Reconstruction of of four classes of inception-v3 trained on ImageNet, from random noise, with regularization losses. (a): index 20, water ouzel, (b): index 323, monarch butterfly, (c), index 251, dalmatians, (d): index 55, green snake.

(a) (b)

Figure 4: Image reconstruction by iterative optimization using neural network VGG19. **Left:** Example reconstructed solutions from various feature maps ϕ as representations. Loss includes TV regularization (bottom) or not (top). From left to right: layers relu1_2, relu2_2, relu3_2, relu4_2, relu5_2. **Right:** Normalized Frobenius similarity between the original image and their reconstructions from various layers, and with the help or not of TV regularization. Similarity is normalized to be in $[0; 1]$.

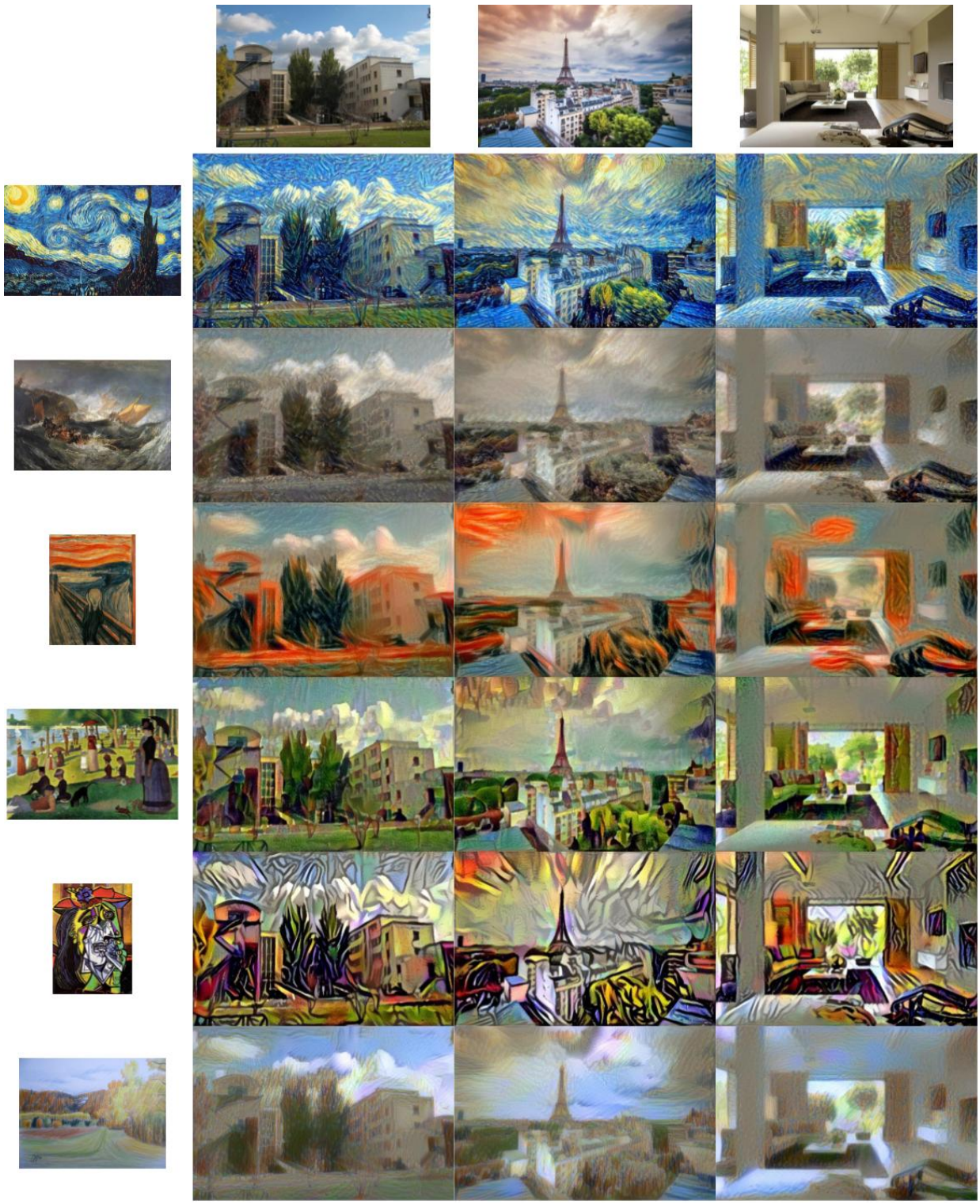
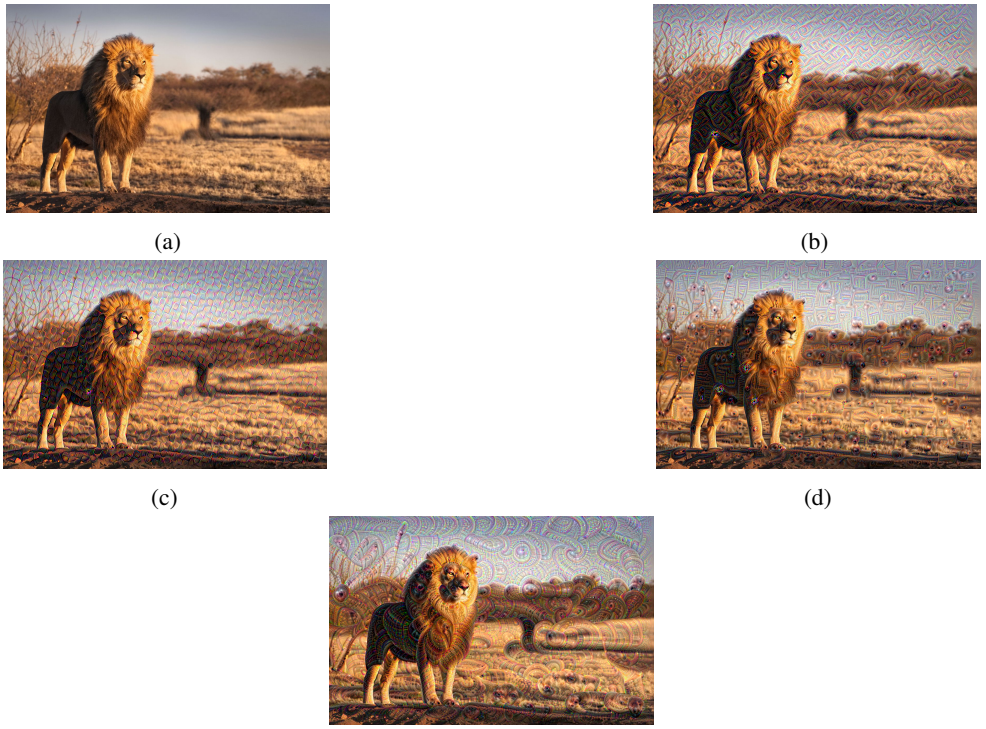


Figure 5: Example of stylized images using a pre-trained version of VGG19. The content loss used layers relu4_2 and relu5_2 equally; the style loss used relu1_1, relu2_1, relu3_1, relu4_1, relu5_1 with decreasing weight of factor 0.7. Best viewed in numeric version. **Column-wise:** input *content* image; from left to right: Université Paris-Saclay, panoramic view of Paris, indoor scene. **Row-wise:** input *style* image, either paintings or drawing; from top to bottom: *The Starry Night* by Vincent van Gogh (1889), *The Shipwreck of the Minotaur* by J.M.W. Turner (1805), *Der Schrei* by Edvard Munch (1893), *Un dimanche après-midi à l'Île de la Grande Jatte* by Georges Seurat (1886), *The Weeping Woman* by Pablo Picasso (1937), a pencil drawing *La falaise en automne* by "Malela" (2011).



(a) (b) (c) (d) (e)

Figure 6: Images generated with the Deep Dream protocol. (a): the original image, (b) and (c) are created to maximise some of the first layers, (d) and (e) are maximizing deeper layers.

(a) (b) (c) (d) (e) (f) (g) (h)

Figure 7: Images generated with the "lucid" dream technique, for the index class 20 (water ouzel). The pictures are respectively iteration 0, 7, 15, 30, 50, 80, 110, 180