Understanding *Context encoders: Feature Learning by Inpainting*

Quentin LEROY
MVA ENS Paris-Saclay
qleroy@ens-paris-saclay.fr

Bastien PONCHON
MVA ENS Paris-Saclay
bponchon@ens-paris-saclay.fr

Abstract

*This report presents our work on [9]. In this paper, the authors propose a novel unsupervised feature learning method using inpainting. Inpainting is the task of filling a potentially large region of an image (one fourth for example). The method is **Context Encoder**, a convolutional neural network trained for inpainting. Trained to create plausible content for the missing part of an image based on the surroundings, the network learns a representation of the input that captures the appearances and semantics of visual structures. The features learned by the network are shown to be effective for other tasks than inpainting that usually requires supervised learning for example segmentation, detection and classification.*

1. Introduction

Convolutional Neural Networks (ConvNets or CNN) were originally found to be effective for the classification task under a supervised setting. A tremendous amount of later works further proved that they also excel for other computer vision tasks. In the present work the authors explore CNN trained in a unsupervised manner. The network is asked to completely restore a large part of the input image that is missing (for example a 64×64 square in the center of an 128×128 image). This enforces the network to holistically understand the semantics of the scene, learns high-level features over the whole spatial extent of the image in order to provide content for the missing part that semantically complies with the context of the scene. This is achieved thanks to an autoencoder-like architecture. The network is composed of an encoder that transforms the input image into a fixed small number of features followed by a decoder that creates out of these features the content of the missing part. The parameters are learned with a reconstruction loss as well as an adversarial loss.

2. The Context-Encoder Model

2.1. Architecture

The original architecture of Context encoders proposed by the authors was to use an autoencoder-like structure to code the context of an input image as a 6x6x256 feature map (9126 hidden features). The **encoder** was to be derived from the *AlexNet* architecture and composed of five convolutional layers. The output of the encoder is connected to the input of the decoder by a channel-wise fully-connected layer (6*6*256=9126 activations layers), in order to propagate the information within each feature map. The **decoder** follows with five up-convolutional layers that augment the dimensions of the latent representation in-between the encoder and the decoder (at the interface of the encoder and the decoder).

2.2. Loss function

2.2.1 Reconstruction Loss

The choice of the loss function is strongly related with the task at hand namely inpainting or put it differently filling a hole missing in the input image. Let x designate the input image and $\hat{M}$ the binary mask such that $\hat{M} * x$ (Hadamard product) selects the missing region. The reconstruction loss is the L2 distance between the content generated by the network and the original region of the image that has been masked to the network. More formally :

$$\mathcal{L}_{rec}(x) = ||\hat{M} * (x - F((1 - \hat{M}) * x))||_2^2$$

Note that the mask $\hat{M}$ can be of any size and shape, covering up to $\frac{1}{4}$ of the image.

This pixel-wise distance does not encourage the network to produce details with high precision but rather create content that overall capture the structure of the original region (*example to put it in perspective: take an image I whose pixels are independently drawn from $\mathcal{N}(0, 1)$; this resembles a texture. The image M whose pixels are zero everywhere is much closer to I in L2 distance that another image I' textured just as I*). Indeed, there are often multiple equally plausible ways to fill a missing image region which

are consistent with the context. The reconstruction loss will tend to average all these modes in the predicted distribution on the image space. To overcome the blurry result favored by the reconstruction loss the authors propose to incorporate an adversarial loss to the total loss.

2.2.2 The GAN framework

The adversarial loss is based on Generative Adversarial Networks. GANs were introduced by [6] as deep generative models able to learn to represent an estimate, either explicitly or implicitly, of some distribution $p_{data}(X)$ on some space X, given a training set composed of samples x drawn from this distribution.

As explained in [5], this is done through a two player game between two parametrized and differentiable (with regards to their inputs and their parameters) functions: a generator G and a discriminator D.

The **Generator** $G : Z \longrightarrow X$ takes as input some z drawn from a prior distribution p_Z on a space Z and tries to produce an output $G(z) \in X$ drawn from a probability distribution p_{model} such that that p_{model} is close from p_{data}, that is to say such that $G(z)$ seems to have been drawn from the distribution $p_{data}(X)$.

The **Discriminator** $D : X \longrightarrow [0, 1]$ on the other hand takes as input an element $x' \in X$ and outputs the probability $D(x)$ that x comes from the training set, that is to say, has been drawn from the true distribution $p_{data}(X)$, rather than having been generated by G, and therefore drawn from the distribution $p_{model}(X)$. This translate by the maximization of the loss function $\mathcal{L}_{adv}$ given by the following log likelihood (or the minimization of the corresponding negative log likelihood):

$$\begin{aligned}\mathcal{L}_{adv}(G, D, Z, X) =& E_{x\sim p_{data(X)}}\left(logD(x)\right) \\ &+ E_{z\sim p_Z}\left(log(1 - D(G(z)))\right)\end{aligned}$$

The generator tries to fool the discriminator, and therefore tries to minimize this very value $\mathcal{L}_{adv}(G, D, Z, X)$. The solution of this minimax game (or zero-sum game, as the generator and the discriminator try to minimize two opposite cost functions) is given by: $min_G max_D \mathcal{L}_{adv}(G, D, Z, X)$.

2.2.3 Adversarial Loss

In the case of our inpainting task, the target domain X is the space of complete(d) images, with p_data represented by the set of training images. The source domain Z is the space of context images (image with a missing masked region). The encoder-decoder pipeline F aims at generating a complete image out of an incomplete one and therefore can be seen as a generator.

The authors hence introduce a corresponding discriminator network D that aims at predicting whether the input fed is a real sample or is a sample coming from the distribution of the content generated by the encoder-decoder pipeline. The discriminator is a five layer convolutional network that takes as input the context either completed with the ouptut of the encoder-decoder pipeline (which is the generated missing area) or the ground truth missing area. The adversarial loss is then:

$$\begin{aligned}\mathcal{L}_{adv}(F, D) =& \mathbb{E}_{X\in\mathcal{X}}[\log D(x)] \\ &+ \log(1 - D(F((1 - \hat{M}) * x)))\end{aligned}$$

The generator and the discriminator are updated simultaneously at each epoch by gradient descent, to maximize $\mathcal{L}_{adv}$ for D and minimize the joint loss for F.

The **joint loss** that the encoder-decoder generator tries to minimize is a weighted combination of the reconstruction loss and the adversarial loss:

$$\mathcal{L} = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}_{adv}$$

Note that unlike the base GAN framework, the loss that the generator of the context encoder tries to minimize is not strictly equal to the opposite.

2.2.4 Non convergence

As stated in [5], instability and non convergence of the learning procedure is one of the main issues in the GAN framework. Indeed, as both the generator and the discriminator try to minimize opposite cost functions, the updates of one can sometimes undo the progress made by the other when updating its own parameters. These alternative updates may lead to oscillation in the loss function $\mathcal{L}$ (one player trying to maximize it while the other try to minimize it) slowing down the learning procedure which may not even end up any close to an equilibrium state.

This is actually an issue the authors of our paper faced. Indeed the context-encoder presented in section 2.1 does not converge with the presented adversarial training. The authors had thus to resort to two different architectures:

1. *Architecture for semantic Inpainting*: Exchanging the encoder-decoder architecture with a one with a smaller hidden feature representation (4000 instead of 9216 features). This architecture no longer uses AlexNet for the encoder, but uses adversarial training, giving compelling realistic results in the inpainting tasks. This architecture shown in Figure 1 is the one for which we had a pre-trained model to work with.

2. *Architecture for Feature Learning*: Using the same architecture for the context-encoder as described in Sec-

tion 2.1 but without the discriminator and the adversarial training. This architecture is shown in Figure 2. The main incentive for using this architecture was its compatibility with prior works in terms of hidden feature representation (the size of the output code of the encoder). This allowed the authors to compare the context encoder with other encoding baselines, and to use the output of the pre-trained encoder as input to classification detection and segmentation methods to verify and compare the relevance of the feature representation learned by the context-encoder.

3. Implementation Details

As we were short on time and computer resources (according to the authors, the model took between 16 hours and one month on Titan-X GPU , depending on the size of the training set) we used the code and the two Torch pretrained models available at github.com/pathak22/context-encoder to understand and experiment the abilities and limitations of the Context-Encoder. We implemented our experiments in Lua Torch. These available pre-trained models had the *architecture for semantic Inpainting* described in Section 2.2.4 and illustrated in Figure 1, that is to say with adversarial training but with only 4000 features in the feature representation layer between the encoder and the decoder. These pre-trained models had been trained on two different data sets: the first one on a subset of ImageNet images, and the second one on a set of Paris street view images.

3.1. Masking techniques

This architecture was only trained for semantic inpainting images where a fixed size central region was dropped. The architecture is similar for inpainting of random regions. The difference is that in this case a different mask size and shape is generated for each training sample. The authors note that this masking technique for training outperforms central region masks as in the latter case the model learns features that are more specific to the mask, since it is the same for all training images.

3.2. Batch Normalization

An important architecture element that is only shortly mentioned in the paper is Batch Normalization, a common practice in deep learning, introduced in [7]. The idea behind Batch Normalization is to reduce the *Internal Covariate Shift*, that is to say the change in the distribution means and variance of the inputs of each layer of the network due to the iterative updates of the parameters of the previous layers during training. This is done by introducing normalization layers in the architectures before non-linear layers (such as ReLU and Sigmoid) which fix the mean and the variance of each feature in the layer over the batch of training inputs to the network. This is done by subtracting to each feature its mean over the input-batch and then divide it by its variance over the input-batch. The results are then given a fixed mean and variance, which are two parameters (for each feature) learned during training, enabling the network to keep its representation ability. This practice enables to speed up the training by using higher learning rates and be less careful about optimization. It also sometimes eliminates the need for Dropout to prevent overfitting.

4. Experiments

4.1. Examples

We first run our pre-trained models on 5 different datasets, to test their limits and abilities. We used the following data sets:

1. Images from the ImageNet dataset ([2]), that was also used to train one of the two models.

2. Images of Paris Street views ([3]), that was also used to train one of the two models.

3. The Brodatz album of texture images from [1] in order to test how the models preform on texture synthesis (a task for which there exist many efficient methods).

4. Dali painting images from Painting-91 data set from [4], to see if the context encoding is robust in the application of less realistic non photograph images.

5. Binary mire images that we have generated to test the performance of the model on simple geometric shapes (lines, circles, squares, etc.).

We compared the performance of the two pretrained models with the IPOL implementation of the Non-Local patch-based algorithm for inpainting from [8]. For the latter, we used the default patch size, which is 7.

The results are shown in Figures 3, 4, 5, and 6.

In these four figures, the first column shows the ground truth image, the second and third column show the out of the context-encoders pre-trained on the Imagenet and on the Paris Street View datasets respectively, and the fourth column shows the output of the Non-Local patch-based algorithm.

It can be seen from Figure 3 that the context encoder yields compelling results on the inpainting of simple shapes, even though the inability to produce the exact white color makes the reconstruction apparent. We can also note that the context-encoder trained on ImageNet images performs better than the other one, probably due to the grater size and diversity of its training set. The patch-based algorithm produce images that cannot be differentiated from

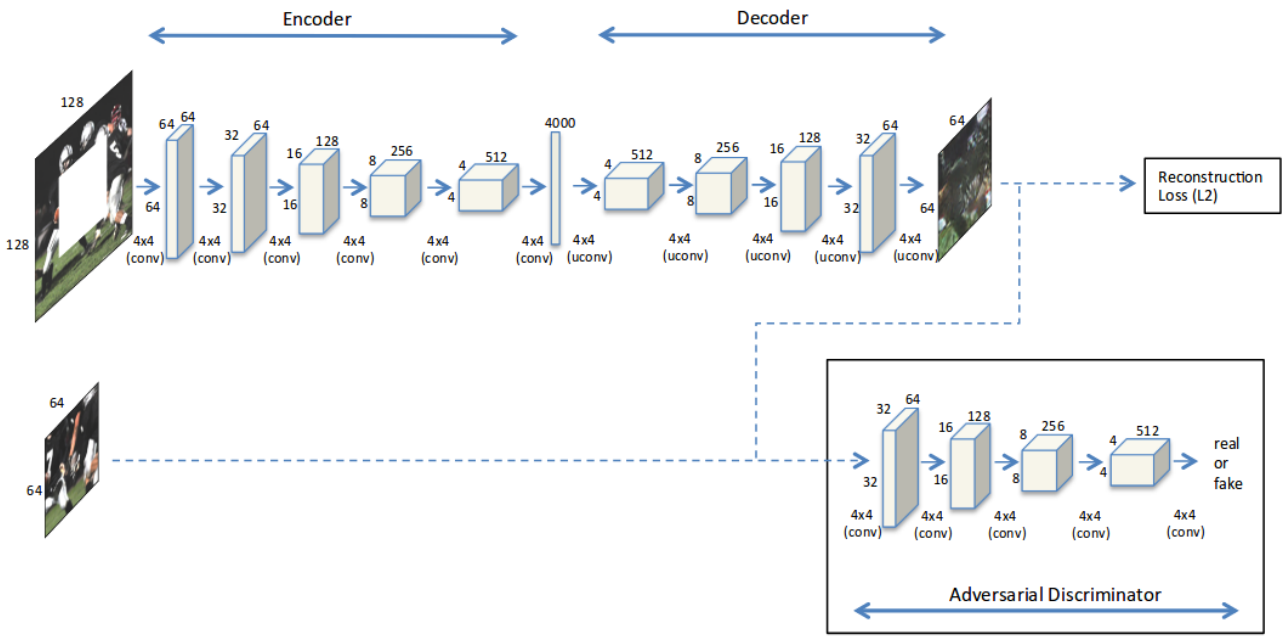

Figure 1. Context encoder trained with joint reconstruction and adversarial loss for **semantic inpainting**. This illustration is shown for center region dropout. Similar architecture holds for arbitrary region dropout as well

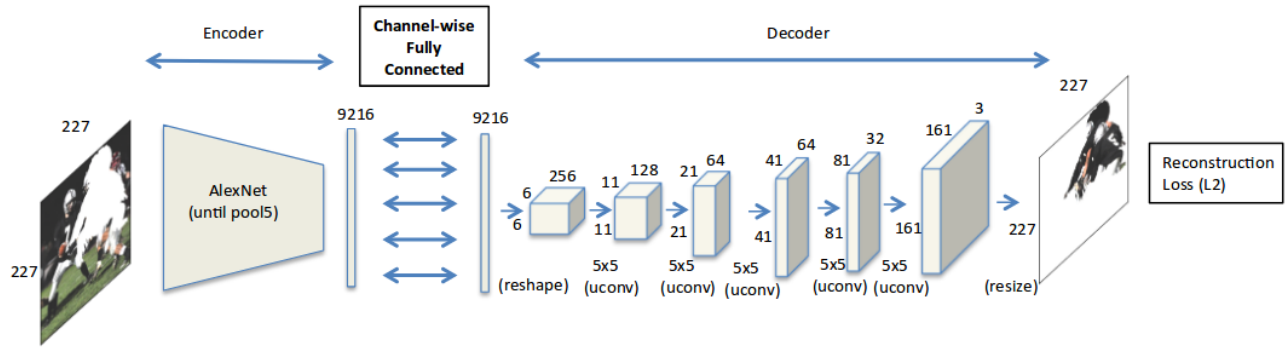

Figure 2. Context encoder trained with reconstruction loss for **feature learning**. Shown for arbitrary region dropouts in the input.

non-reconstructed images but fails to reproduce the actual circle and angle shapes (this may be achieved by choosing a smaller patch size for the algorithm).

The results of our two models on images from the same data sets from which the training images were taken are shown in Figure 4. The results are quite compelling, especially on paris street views and on simple structures of ImageNet images, but rarely producing seamless inpainting. Logically enough, each model performs better than the other on images similar to the ones it was trained on. The patch-based algorithm usually produce unrealistic images, as it tries to paste patches from other various locations in the context.

As shown in Figure 5, we also tested the context-encoder on texture images, where the patch-based algorithm largely outperforms the context-encoder models.

Finally, we tested the context encoder on Dali paintings to see how the models performed on less realistic images. The model trained on ImageNet images actually produce images that could be mistaken for actual Dali paintings, even though they are different from the originals.

4.2. Impacts of the various layers

In order to understand the roles of the various layers in the context-encoder network, we reproduced the results on ImageNet and ParisStreetViews while setting to random (but with the same mean and variance) the weights of each layer succesively. The results are shown in Figures 7 and 8.

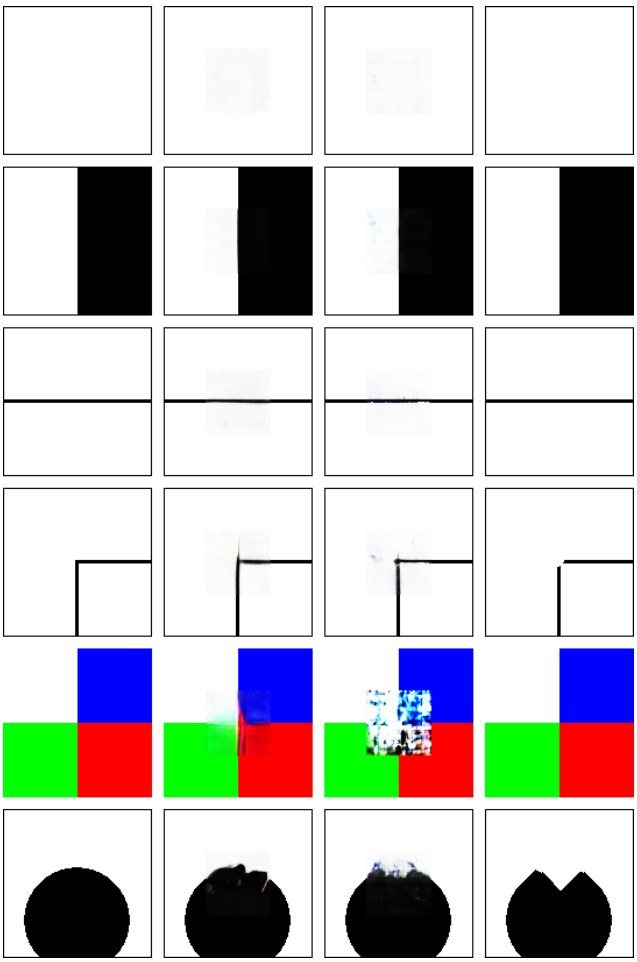

Figure 3. Results on some simple binary images. **First column**: original images. **Second column**: ImageNet model. **Third column**: Paris Street View model. **Fourth column**: Non-local patch-based.

Each row corresponds to the output of the models when the weights of one of the convolutional layers was randomly modified. The top rows shows the results when the shallower layers are randomized, while the bottom rows show the results when the deepest layers are randomized. We see that the deepest the randomized layer, the more readable the produced image. This makes senses as randomizing shallow layer impacts the subsequent layers outputs.

4.3. Visualization of the first layer

Figures 9 and 10 show respectively the 64 convolutional filters of the first layer and the corresponding 64 features maps. The filters are of size 4x4. We can see from both figures that the first layer aims at capturing some basic geometric features such as edges and convex shapes.

4.4. Visualization of the hidden features

This section proposes a way to visualize the high-dimensional 4000 features. An image is passed through Context Encoder. We retain the `code`, that is the 4000 features at the interface of the encoder and the decoder. In [9], the authors use these features for segmentation and classification, with the slight but important difference that they passed the images through a Context Encoder trained for random block inpainting, a network that is not available from their GitHub repository. Hopefully the features learned with the central block inpainting are also useful. Figure 11 and Figure 12 show interesting results. In Figure 11 the images are from the few datasets discussed above, the images are very different in nature, from synthetic images to natural images of animals or street buildings along with textures. The 2D-visualization has this appealing property that it separates mire images from texture images and almost texture images from natural images. Figure 12 shows the results on other images. These images are taken from recvis17/assignment2, there are instances of two objects: aeroplanes and motorbikes. The thumbnails scatter across the 2D-space almost seperating the two classes. But looking more closely we remark that in fact it may separate the images according to colors; we see a motorbike entangled with all the aeroplanes with blue background and some aeroplanes with no blue background within the motorbikes thumbnails. Actually the center of the image is occluded to the network and it must inpaint it, it just so happens that the aeroplanes or motorbikes are mostly located at the center of the images and are almost always in the center region that is hidden to the network. This little experiment is convincing in that the central region cropping model is not the best suited for feature learning, rather a random region cropping model would most probably occlude only a part of objects of interest in the image and indeed learn interesting features about them by inpainting the hidden region in a unsupervised manner.

4.5. Conclusion

We presented our work and understanding on the paper *Context-encoder: Feature Learning by Inpainting*. The presented method encode in a non supervised fashion the context of the image by enforcing the network to perform an supervised inpainting task of a masked region in the image. It is very interesting to that the learned features are indeed relevant (as shown by the tests in the paper) and leads to better results (when plugged to classification, detection ad segmentation techniques) that the feature representation in the auto-encoder (which have more information as input, since we do not mask a part of the image). However, in addition to the fact that the authors finally resort to presenting two different architectures, one for the semantic inpainting and the other for feature learning, some other critics could

Figure 4. Results on Imagenet and Paris Street View images. **First column**: original images. **Second column**: ImageNet model. **Third column**: Paris Street View model. **Fourth column**: Non-local patch-based.

be held against this methods.

First we were quite disappointed not to have the ability to perform some tests on a model pre-trained to inpaint randomly masked region. Indeed, having experienced that changing the color of the masked region having quite important impacts on the output of the model, we wonder how the architecture works for random region inpainting. Will the model inpaint all uniform regions with a similar color as the mask ? How does the model deal with the activation produced by the masked area (which is irrelevant to the context encoding).

Finally, we believe this paper raises the question of the definition of the concept of *context*. Indeed, in most images we had to deal with, the most relevant objects and scenes were occluded when applying the mask to the image before feeding it as input to the context encoder. We hence noticed that some different images representing different different objects and scenes could fit in similar context (and therefore be associated with the same code), as explained in section 4.4 with the planes and motorbikes classification. Is encoding the context relevant if it is disconnected from the actual semantic of the scene ? This could explain why the method still stands way behind supervised methods for task such as classification.

References

[1] P. Brodatz. *Textures: A Photographic Album for Artists and Designers*. 1966.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes paris look like paris? *ACM Transactions on Graphics (SIGGRAPH)*, 31(4):101:1–101:9, 2012.

[4] J. v. d. W. M. F. Fahad Khan, Shida Beigpour. Painting-91: A large scale database for computational painting cate-

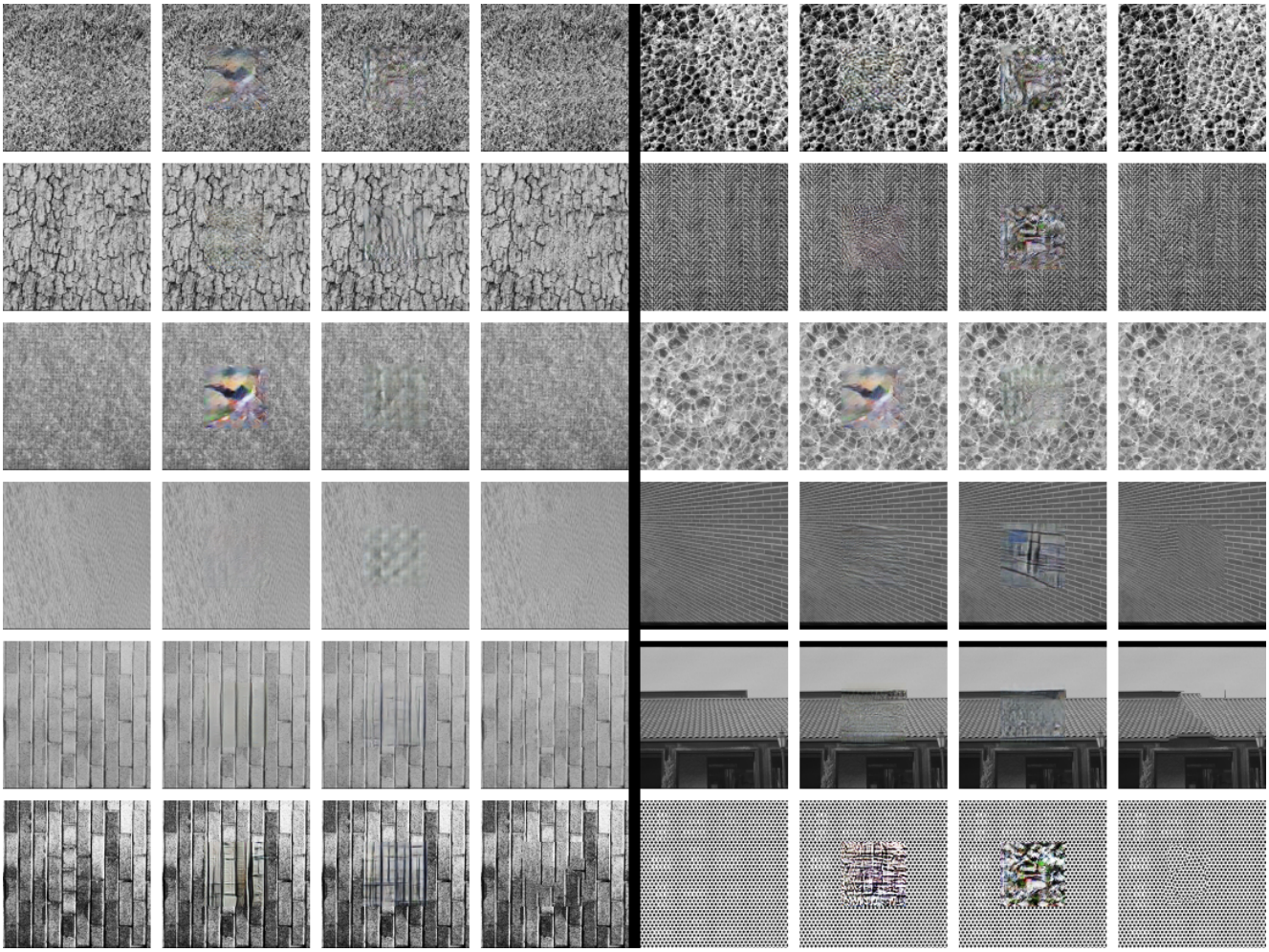

Figure 5. Some examples on various datasets. **First column**: original images. **Second column**: ImageNet model. **Third column**: Paris Street View model. **Fourth column**: Non-local patch-based.

gorization. *Categorization, Machine Vision and Application (MVAP)*, 25(6):1385-1397, 2014.

[5] I. J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[8] A. Newson, A. Almansa, Y. Gousseau, and P. Prez. Non-Local Patch-Based Image Inpainting. *Image Processing On Line*, 7:373–385, 2017.

[9] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. *CoRR*, abs/1604.07379, 2016.

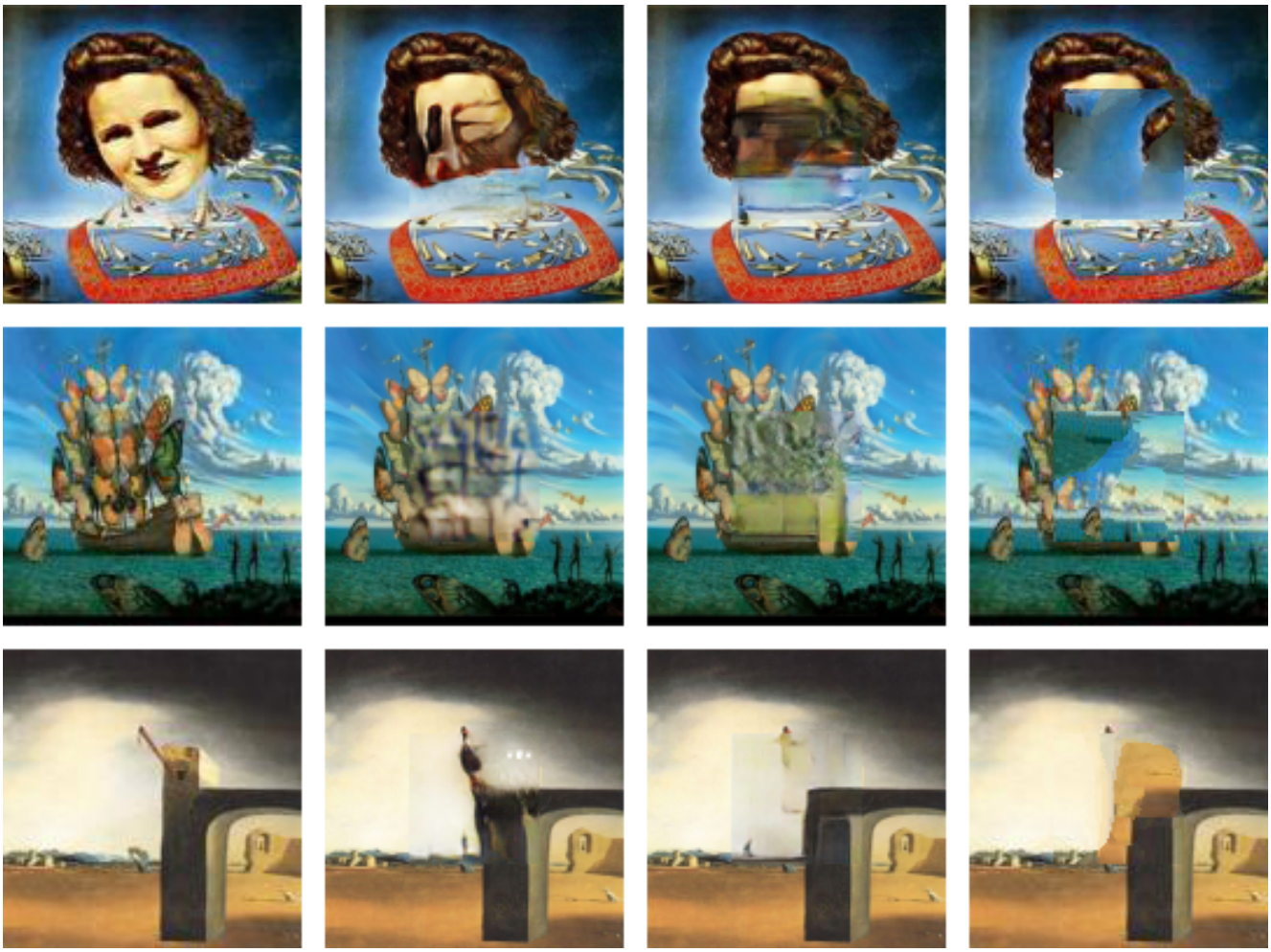

Figure 6. Some examples on various datasets. **First column**: original images. **Second column**: ImageNet model. **Third column**: Paris Street View model. **Fourth column**: Non-local path-based.

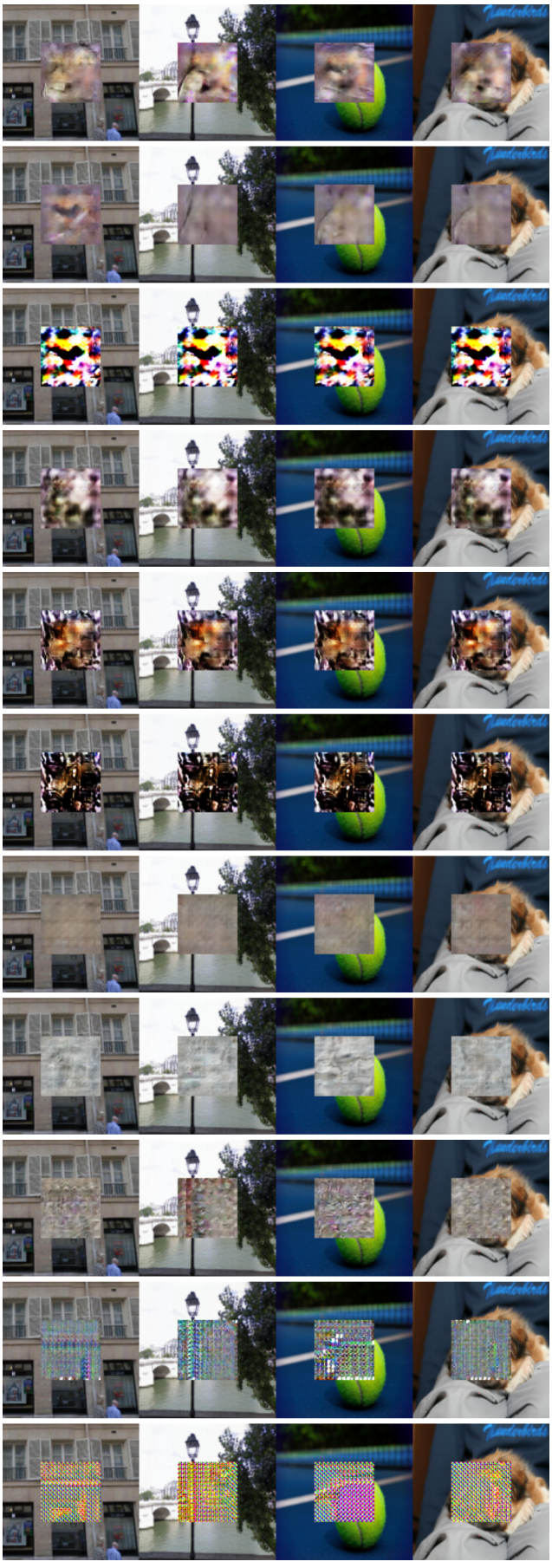

Figure 7. Outputs of the ImageNet pre-trained model with randomized weights. Each line correspond to the output of the network where the weights of one of the 11 convolutional layers were randomized.

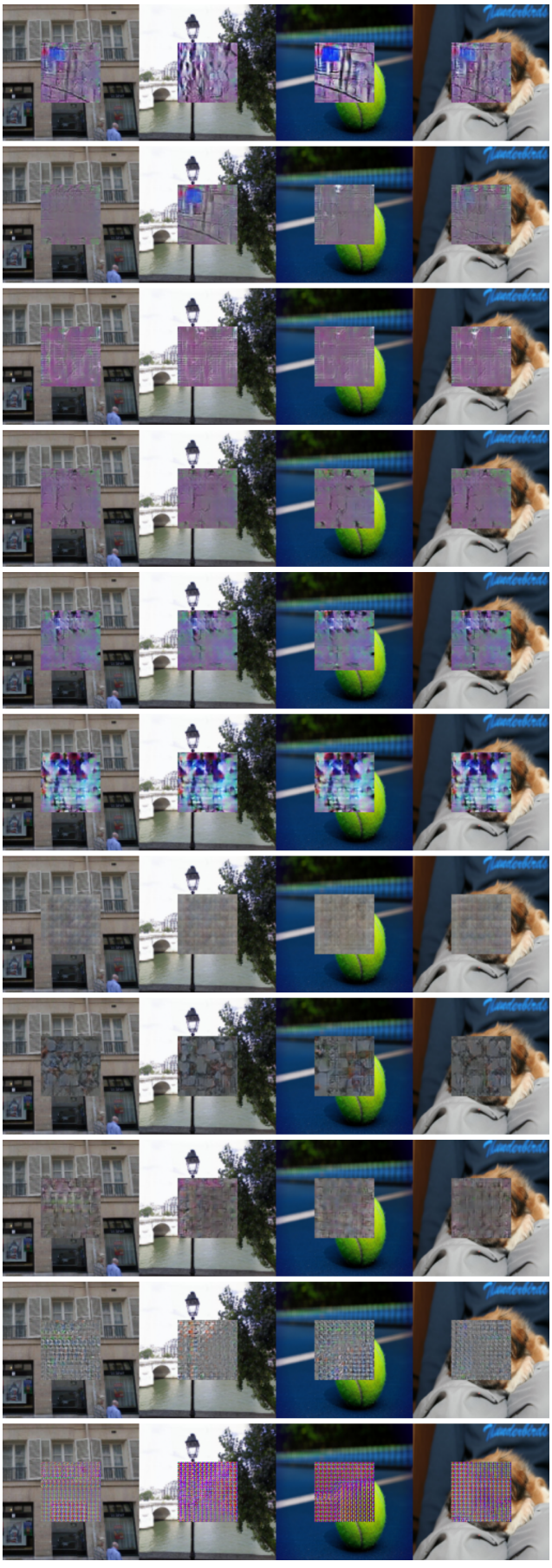

Figure 8. Outputs of the ParisStreetView pre-trained model with randomized weights. Each line correspond to the output of the network where the weights of one of the 11 convolutional layers were randomized.

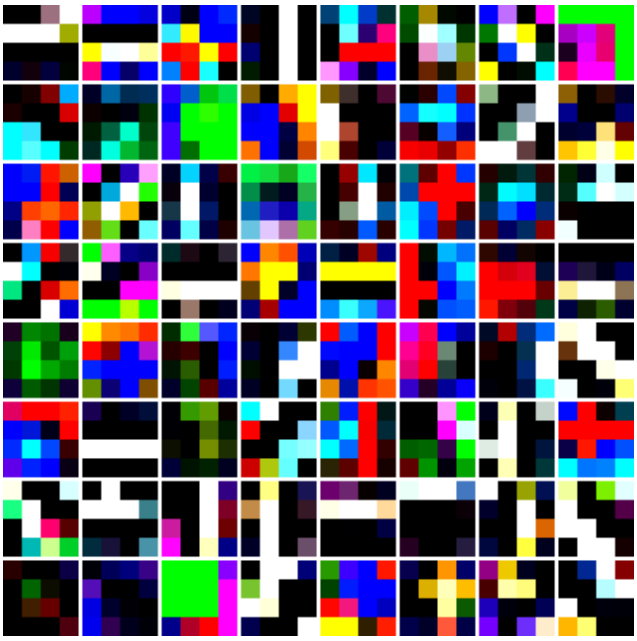

Figure 9. Visualization of the 64 filters of the first convolutional layer. (3 channels)

Figure 10. Visualization of the 64 feature maps produced by the first convolutional layer for the dog image of Figure 4

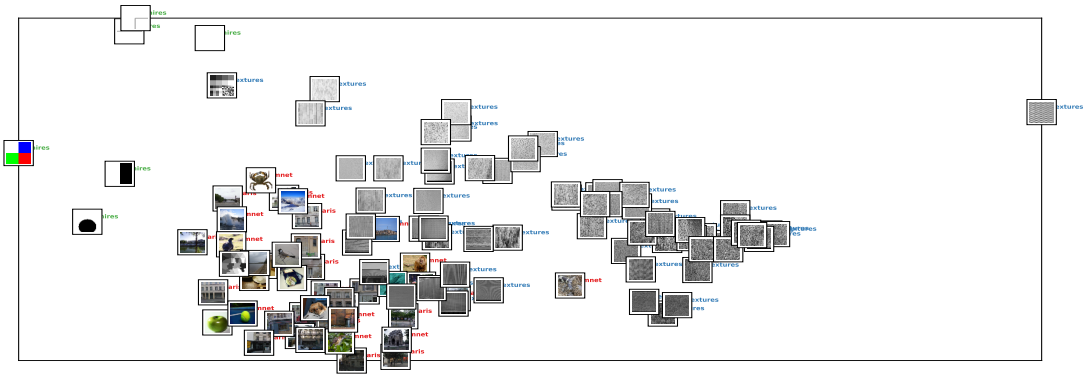

Figure 11. 112 images from the four datasets discussed in the section 4.1

Figure 12. Some images from recvis17/assignment2. 2113 images were passed through Context Encoder and the 4000 features at the interface of the encoder and the decoder were retained. The thumbnails in the figure are displayed in the 2D space with coordinates computed with t-SNE.