Texture Networks: Feed-forward Synthesis of Textures and Stylized Images

Dmitry Ulyanov DMITRY.ULYANOV@SKOLTECH.RU
Skolkovo Institute of Science and Technology & Yandex, Russia

Vadim Lebedev VADIM.LEBEDEV@SKOLTECH.RU
Skolkovo Institute of Science and Technology & Yandex, Russia

Andrea Vedaldi VEDALDI@ROBOTS.OX.AC.UK
University of Oxford, United Kingdom

Victor Lempitsky LEMPITSKY@SKOLTECH.RU
Skolkovo Institute of Science and Technology, Russia

Abstract

Gatys et al. recently demonstrated that deep networks can generate beautiful textures and stylized images from a single texture example. However, their methods requires a slow and memory-consuming optimization process. We propose here an alternative approach that moves the computational burden to a learning stage. Given a single example of a texture, our approach trains compact feed-forward convolutional networks to generate multiple samples of the same texture of arbitrary size and to transfer artistic style from a given image to any other image. The resulting networks are remarkably light-weight and can generate textures of quality comparable to Gatys et al., but hundreds of times faster. More generally, our approach highlights the power and flexibility of generative feed-forward models trained with complex and expressive loss functions.

1. Introduction

Several recent works demonstrated the power of deep neural networks in the challenging problem of *generating images*. Most of these proposed **generative networks** that produce images as output, using feed-forward calculations from a random seed; however, very impressive results were obtained by (Gatys et al., 2015a;b) by using **networks descriptively**, as image statistics. Their idea is to reduce image generation to the problem of sampling at random from the set of images that match a certain statistics. In **texture synthesis** (Gatys et al., 2015a), the reference statistics is extracted from a single example of a visual texture, and the goal is to generate further examples of that texture. In **style transfer** (Gatys et al., 2015b), the goal is to match simultaneously the visual style of a first image, captured using some low-level statistics, and the visual content of a second image, captured using higher-level statistics. In this manner, the style of an image can be replaced with the one of another without altering the overall semantic content of the image.

Matching statistics works well in practice, is conceptually simple, and demonstrates that off-the-shelf neural networks trained for generic tasks such as image classification can be re-used for image generation. However, the approach of (Gatys et al., 2015a;b) has certain shortcomings too. Being based on an *iterative optimization procedure*, it requires backpropagation to gradually change the values of the pixels until the desired statistics is matched. This iterative procedure requires several seconds in order to generate a relatively small image using a high-end GPU, while scaling to large images is problematic because of high memory requirements. By contrast, feed-forward generation networks can be expected to be much more efficient because they require a single evaluation of the network and do not incur in the cost of backpropagation.

In this paper we look at the problem of achieving the synthesis and stylization capability of descriptive networks using feed-forward generation networks. Our contribution is threefold. First, we show for the first time that a generative approach can produce textures of the quality and diversity comparable to the descriptive method. Second, we propose a generative method that is two orders of magnitude faster and one order of magnitude more memory efficient than the

The source code and pretrained models are available at https://github.com/DmitryUlyanov/texture_nets

Input Gatys et al. Texture nets (ours) Input Gatys et al. Texture nets (ours)

Figure 1. Texture networks proposed in this work are feed-forward architectures capable of learning to synthesize complex textures based on a single training example. The perceptual quality of the feed-forwardly generated textures is similar to the results of the closely related method suggested in (Gatys et al., 2015a), which use slow optimization process.

descriptive one. Using a single forward pass in networks that are remarkably compact make our approach suitable for video-related and possibly mobile applications. Third, we devise a new type of multi-scale generative architecture that is particularly suitable for the tasks we consider.

The resulting fully-convolutional networks (that we call *texture networks*) can generate textures and process images of arbitrary size. Our approach also represents an interesting showcase of training conceptually-simple feed-forward architectures while using complex and expressive loss functions. We believe that other interesting results can be obtained using this principle.

The rest of the paper provides the overview of the most related approaches to image and texture generation (Sect. 2), describes our approach (Sect. 3), and provides extensive extensive qualitative comparisons on challenging textures and images (Sect. 4).

2. Background and related work

Image generation using neural networks. In general, one may look at the process of generating an image $\mathbf{x}$ as the problem of drawing a sample from a certain distribution $p(\mathbf{x})$. In texture synthesis, the distribution is induced by an example texture instance $\mathbf{x_0}$ (*e.g.* a polka dots image), such that we can write $\mathbf{x} \sim p(\mathbf{x}|\mathbf{x_0})$. In style transfer, the distribution is induced by an image $\mathbf{x_0}$ representative of the visual style (*e.g.* an impressionist painting) and a second image $\mathbf{x}_1$ representative of the visual content (*e.g.* a boat), such that $\mathbf{x} \sim p(\mathbf{x}|\mathbf{x_0}, \mathbf{x}_1)$.

(Mahendran & Vedaldi, 2015; Gatys et al., 2015a;b) reduce this problem to the one of finding a *pre-image* of a certain image statistics $\Phi(\mathbf{x}) \in \mathbb{R}^d$ and pose the latter as an optimization problem. In particular, in order to synthesize a texture from an example image $\mathbf{x_0}$, the pre-image problem is:

$$\operatorname*{argmin}_{\mathbf{x}\in\mathcal{X}} \|\Phi(\mathbf{x}) - \Phi(\mathbf{x_0})\|_2^2. \tag{1}$$

Importantly, the pre-image $\mathbf{x} : \Phi(\mathbf{x}) \approx \Phi(\mathbf{x_0})$ is usually not unique, and sampling pre-images achieves diversity. In practice, samples are extracted using a local optimization algorithm $\mathcal{A}$ starting from a random initialization $\mathbf{z}$. Therefore, the generated image is the output of the function

$$\operatorname*{localopt}_{\mathbf{x}\in\mathcal{X}}(\|\Phi(\mathbf{x}) - \Phi(\mathbf{x_0})\|_2^2; \mathcal{A}, \mathbf{z}), \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \Sigma). \tag{2}$$

This results in a distribution $p(\mathbf{x}|\mathbf{x_0})$ which is difficult to characterise, but is easy to sample and, for good statistics Φ, produces visually pleasing and diverse images. Both (Mahendran & Vedaldi, 2015) and (Gatys et al., 2015a;b) base their statistics on the response that $\mathbf{x}$ induces in deep neural network layers. Our approach reuses in particular the statistics based on correlations of convolutional maps proposed by (Gatys et al., 2015a;b).

Descriptive texture modelling. The approach described above has strong links to many well-known models of visual textures. For texture, it is common to assume that $p(\mathbf{x})$ is a stationary *Markov random field* (MRF). In this case, the texture is ergodic and one may considers local spatially-invariant statistics $\psi \circ F(\mathbf{x}; i), i \in \Omega$, where i denotes a spatial coordinate. Often F is the output of a bank of linear filters and ψ an histogramming operator. Then the spatial average of this local statistics on the prototype texture $\mathbf{x_0}$ approximates its sample average

$$\phi(\mathbf{x_0}) = \frac{1}{|\Omega|}\sum_{i=1}^{|\Omega|} \psi \circ F(\mathbf{x_0}; i) \approx \underset{\mathbf{x}\sim p(\mathbf{x})}{E}[\psi \circ F_l(\mathbf{x}; 0)]. \quad (3)$$

The FRAME model of (Zhu et al., 1998) uses this fact to induce the maximum-entropy distribution over textures $p(\mathbf{x}) \propto \exp(-\langle \lambda, \phi(\mathbf{x}) \rangle)$, where λ is a parameter chosen so that the marginals match their empirical estimate, i.e. $E_{\mathbf{x}\sim p(\mathbf{x})}[\phi(\mathbf{x})] = \phi(\mathbf{x_0})$.

A shortcoming of FRAME is the difficulty of sampling from the maxent distribution. (Portilla & Simoncelli, 2000) addresses this limitation by proposing to directly find images $\mathbf{x}$ that match the desired statistics $\Phi(\mathbf{x}) \approx \Phi(\mathbf{x_0})$, pioneering the pre-image method of (1).

Where (Zhu et al., 1998; Portilla & Simoncelli, 2000) use linear filters, wavelets, and histograms to build their texture statistics, (Mahendran & Vedaldi, 2015; Gatys et al., 2015a;a) extract statistics from pre-trained deep neural networks. (Gatys et al., 2015b) differs also in that it considers the style transfer problem instead of the texture synthesis one.

Generator deep networks. An alternative to using a neural networks as descriptors is to construct generator networks $\mathbf{x} = \mathbf{g}(\mathbf{z})$ that produce *directly* an image $\mathbf{x}$ starting from a vector of random or deterministic parameters $\mathbf{z}$.

Approaches such as (Dosovitskiy et al., 2015) learn a mapping from deterministic parameters $\mathbf{z}$ (*e.g.* the type of object imaged and the viewpoint) to an image $\mathbf{x}$. This is done by fitting a neural network to minimize the discrepancy $\|\mathbf{x}_i - \mathbf{g}(\mathbf{z}_i)\|$ for known image-parameter pairs $(\mathbf{x}_i, \mathbf{z}_i)$. While this may produce visually appealing results, it requires to know the relation $(\mathbf{x}, \mathbf{z})$ beforehand and cannot express any diversity beyond the one captured by the parameters.

An alternative is to consider a function $\mathbf{g}(\mathbf{z})$ where the parameters $\mathbf{z}$ are unknown and are sampled from a (simple) random distribution. The goal of the network is to map these random values to plausible images $\mathbf{x} = \mathbf{g}(\mathbf{z})$. This requires measuring the quality of the sample, which is usually expressed as a distance between $\mathbf{x}$ and a set of example images $\mathbf{x}_1, \ldots, \mathbf{x}_n$. The key challenge is that the distance must be able to generalize significantly from the available examples in order to avoid penalizing sample diversity.

Generative Adversarial Networks (GAN; (Goodfellow et al., 2014)) address this problem by training, together with the generator network $\mathbf{g}(\mathbf{z})$, a second *adversarial* network $f(\mathbf{x})$ that attempts to distinguish between samples $\mathbf{g}(\mathbf{z})$ and natural image samples. Then f can be used as a measure of quality of the samples and $\mathbf{g}$ can be trained to optimize it. LAPGAN (Denton et al., 2015) applies GAN to a Laplacian pyramid of convolutional networks and DCGAN (Radford et al., 2015) further optimizes GAN and learn is from very large datasets.

Moment matching networks. The maximum entropy model of (Zhu et al., 1998) is closely related to the idea of *Maximum Mean Discrepancy* (MMD) introduced in (Gretton et al., 2006). Their key observation the expected value $\mu_p = E_{\mathbf{x}\sim p(\mathbf{x})}[\phi(\mathbf{x})]$ of certain statistics $\phi(\mathbf{x})$ uniquely identifies the distribution p. (Li et al., 2015; Dziugaite et al., 2015) derive from it a loss function alternative to GAN by comparing the statistics averaged over network samples $\frac{1}{m}\sum_{i=1}^{m} \phi \circ \mathbf{g}(\mathbf{z}_i)$ to the statistics averaged over empirical samples $\frac{1}{m}\sum_{i=1}^{m} \phi(\mathbf{x}_i)$. They use it to train a *Moment Matching Network* (MMN) and apply it to generate small images such as MNIST digits. Our networks are similar to moment matching networks, but use very specific statistics and applications quite different from the considered in (Li et al., 2015; Dziugaite et al., 2015).

3. Texture networks

We now describe the proposed method in detail. At a high-level (see Figure 2), our approach is to train a feed-forward **generator network** $\mathbf{g}$ which takes a noise sample $\mathbf{z}$ as input and produces a texture sample $\mathbf{g}(\mathbf{z})$ as output. For style transfer, we extend this *texture network* to take both a noise sample $\mathbf{z}$ and a content image $\mathbf{y}$ and then output a new image $\mathbf{g}(\mathbf{y}, \mathbf{z})$ where the texture has been applied to $\mathbf{y}$ as a visual style. A separate generator network is trained for each texture or style and, once trained, it can synthesize an arbitrary number of images of arbitrary size in an efficient feed-forward manner.

A key challenge in training the generator network $\mathbf{g}$ is to construct a loss function that can assess automatically the quality of the generated images. For example, the key idea of GAN is to *learn* such a loss along with the generator network. We show in Sect. 3.1 that a very powerful loss can be derived from pre-trained and fixed **descriptor networks** using the statistics introduced in (Gatys et al., 2015a;b). Given the loss, we then discuss the architecture of the generator network for texture synthesis (Sect. 3.2) and then generalize it to style transfer (Sect 3.3).

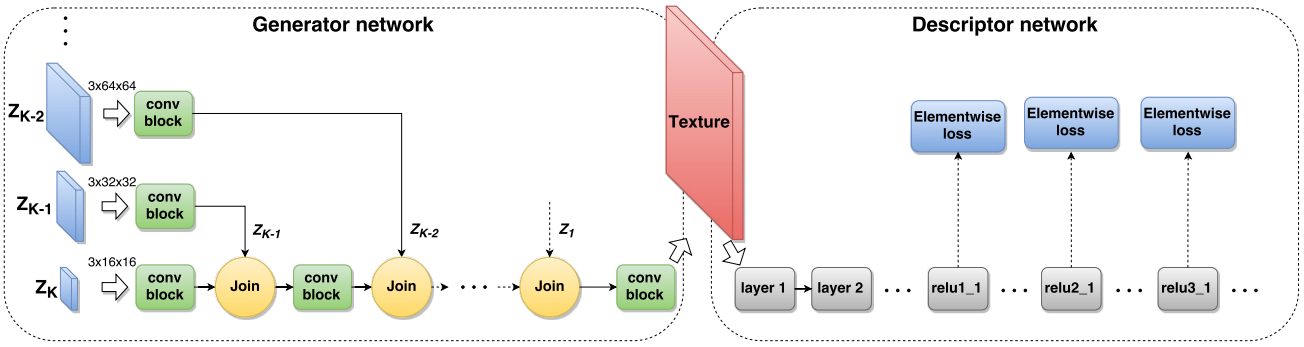

Figure 2. Overview of the proposed architecture (texture networks). We train a *generator network* (left) using a powerful loss based on the correlation statistics inside a fixed pre-trained *descriptor network* (right). Of the two networks, only the generator is updated and later used for texture or image synthesis. The `conv` block contains multiple convolutional layers and non-linear activations and the `join` block upsampling and channel-wise concatenation. Different branches of the generator network operate at different resolutions and are excited by noise tensors z_i of different sizes.

3.1. Texture and content loss functions

Our loss function is derived from (Gatys et al., 2015a;b) and compares image statistics extracted from a fixed pre-trained descriptor CNN (usually one of the VGG CNN (Simonyan & Zisserman, 2014; Chatfield et al., 2014) which are pre-trained for image classification on the ImageNet ILSVRC 2012 data). The descriptor CNN is used to measure the mismatch between the *prototype* texture $\mathbf{x_0}$ and the generated image $\mathbf{x}$. Denote by $F_i^l(\mathbf{x})$ the i-th map (feature channel) computed by the l-th convolutional layer by the descriptor CNN applied to image $\mathbf{x}$. The *Gram matrix* $G^l(\mathbf{x})$ is defined as the matrix of scalar (inner) products between such feature maps:

$$G_{ij}^l(\mathbf{x}) = \langle F_i^l(\mathbf{x}), F_j^l(\mathbf{x}) \rangle . \tag{4}$$

Given that the network is convolutional, each inner product implicitly sums the products of the activations of feature i and j at all spatial locations, computing their (unnormalized) empirical correlation. Hence $G_{ij}^l(\mathbf{x})$ has the same general form as (3) and, being an orderless statistics of local stationary features, can be used as a texture descriptor.

In practice, (Gatys et al., 2015a;b) use as texture descriptor the combination of several Gram matrices $G^l, l \in L_T$, where L_T contains selected indices of convolutional layer in the descriptor CNN. This induces the following *texture loss* between images $\mathbf{x}$ and $\mathbf{x_0}$:

$$\mathcal{L}_T(\mathbf{x}; \mathbf{x_0}) = \sum_{l \in L_T} \|G^l(\mathbf{x}) - G^l(\mathbf{x_0})\|_2^2 . \tag{5}$$

In addition to the texture loss (5), (Gatys et al., 2015b) propose to use as *content loss* the one introduced by (Mahendran & Vedaldi, 2015), which compares images based on the output $F_i^l(\mathbf{x})$ of certain convolutional layers $l \in L_C$ (without computing further statistics such as the Gram matrices). In formulas

$$\mathcal{L}_C(\mathbf{x}; \mathbf{y}) = \sum_{l \in L_C} \sum_{i=1}^{N_l} \|F_i^l(\mathbf{x}) - F_i^l(\mathbf{y})\|_2^2 , \tag{6}$$

where N_l is the number of maps (feature channels) in layer l of the descriptor CNN. The key difference with the texture loss (5) is that the content loss compares feature activations at corresponding spatial locations, and therefore preserves spatial information. Thus this loss is suitable for content information, but not for texture information.

Analogously to (Gatys et al., 2015a), we use the texture loss (5) alone when training a generator network for texture synthesis, and we use a weighted combination of the texture loss (5) and the content loss (6) when training a generator network for stylization. In the latter case, the set L_C does not includes layers as shallow as the set L_T as only the high-level content should be preserved.

3.2. Generator network for texture synthesis

We now discuss the architecture and the training procedure for the generator network $\mathbf{g}$ for the task of texture synthesis. We denote the parameters of the generator network as θ. The network is trained to transform a noise vector $\mathbf{z}$ sampled from a certain distribution $\mathcal{Z}$ (which we set to be uniform i.i.d.) into texture samples that match, according to the texture loss (5), a certain prototype texture $\mathbf{x_0}$:

$$\theta_{\mathbf{x_0}} = \operatorname*{argmin}_{\theta} E_{\mathbf{z} \sim \mathcal{Z}} \left[\mathcal{L}_T(\mathbf{g}(\mathbf{z}; \theta), \mathbf{x_0}) \right] . \tag{7}$$

Network architecture. We experimented with several architectures for the generator network $\mathbf{g}$. The simplest are chains of convolutional, non-linear activation, and upsampling layers that start from a noise sample $\mathbf{z}$ in the form of

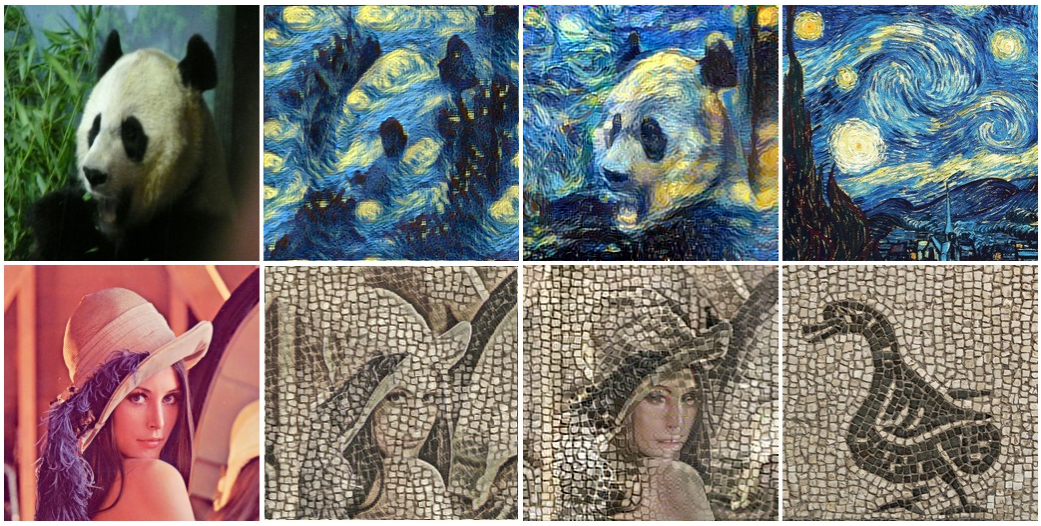

Figure 3. Our approach can also train feed-forward networks to transfer style from artistic images (left). After training, a network can transfer the style to any new image (e.g. right) while preserving semantic content. For some styles (bottom row), the perceptual quality of the result of our feed-forward transfer is comparable with the optimization-based method (Gatys et al., 2015b), though for others the results are not as impressive (top row and (Supp.Material)).

a small feature map and terminate by producing an image. While models of this type produce reasonable results, we found that *multi-scale architectures* result in images with smaller texture loss and better perceptual quality while using fewer parameters and training faster. Figure 2 contains a high-level representation of our reference multi-scale architecture, which we describe next.

The reference texture $\mathbf{x_0}$ is a tensor $\mathbb{R}^{M\times M\times 3}$ containing three color channels. For simplicity, assume that the spatial resolution M is a power of two. The input noise $\mathbf{z}$ comprises K random tensors $\mathbf{z}_i \in \mathbb{R}^{\frac{M}{2^i}\times\frac{M}{2^i}}$, $i = 1, 2, \dots, K$ (we use $M = 256$ and $K = 5$) whose entries are i.i.d. sampled from a uniform distribution. Each random noise tensor is first processed by a sequence of convolutional and non-linear activation layers, then upsampled by a factor of two, and finally concatenated as additional feature channels to the partially processed tensor from the scale below. The last full-resolution tensor is ultimately mapped to an RGB image $\mathbf{x}$ by a bank of 1×1 filters.

Each convolution block in Figure 2 contains three convolutional layers, each of which is followed by a ReLU activation layer. The convolutional layers contain respectively 3×3, 3×3 and 1×1 filters. Filers are computed densely (stride one) and applied using circular convolution to remove boundary effects, which is appropriate for textures. The number of feature channels, which equals the number of filters in the preceding bank, grows from a minimum of 8 to a maximum of 40. The supplementary material specifies in detail the network configuration which has only $\sim$65K parameters, and can be compressed to $\sim$300 Kb of memory.

Upsampling layers use simple nearest-neighbour interpolation (we also experimented strided full-convolution (Long et al., 2015; Radford et al., 2015), but the results were not satisfying). We found that training benefited significantly from inserting batch normalization layers (Ioffe & Szegedy, 2015) right after each convolutional layer and, most importantly, right before the concatenation layers, since this balances gradients travelling along different branches of the network.

Learning. Learning optimizes the objective (7) using stochastic gradient descent (SGD). At each iteration, SGD draws a mini-batch of noise vectors $\mathbf{z}_k, k = 1, \dots, B$, performs forward evaluation of the generator network to obtained the corresponding images $\mathbf{x}_k = \mathbf{g}(\mathbf{z}_k, \theta)$, performs forward evaluation of the descriptor network to obtain Gram matrices $G^l(\mathbf{x}_k), l \in L_T$, and finally computes the loss (5) (note that the corresponding terms $G^l(\mathbf{x_0})$ for the reference texture are constant). After that, the gradient of the texture loss with respect to the generator network parameters θ is computed using backpropagation, and the gradient is used to update the parameters. Note that LAPGAN (Denton et al., 2015) also performs multi-scale processing, but uses layer-wise training, whereas our generator is trained end-to-end.

3.3. Style transfer

In order to extend the method to the task of image stylization, we make several changes. Firstly, the generator net-

Figure 4. Further comparison of textures generated with several methods including the original statistics matching method (Portilla & Simoncelli, 2000) and the DCGAN (Radford et al., 2015) approach. Overall, our method and (Gatys et al., 2015a) provide better results, our method being hundreds times faster.

work $\mathbf{x} = \mathbf{g}(\mathbf{y}, \mathbf{z}; \theta)$ is modified to take as input, in addition to the noise variable $\mathbf{z}$, the image $\mathbf{y}$ to which the noise should be applied. The generator network is then trained to output an image $\mathbf{x}$ that is close in content to $\mathbf{y}$ and in texture/style to a reference texture $\mathbf{x_0}$. For example, $\mathbf{y}$ could be a photo of a person, and $\mathbf{x_0}$ an impressionist painting.

Network architecture. The architecture is the same as the one used for texture synthesis with the important difference that now the noise tensors $\mathbf{z}_i, i = 1, \ldots, K$ at the K scales are concatenated (as additional feature channels) with downsampled versions of the input image $\mathbf{y}$. For this application, we found beneficial to increased the number of scales from $K = 5$ to $K = 6$.

Learning. Learning proceeds by sampling noise vectors $\mathbf{z}_i \sim \mathcal{Z}$ and natural images $\mathbf{y}_i \sim \mathcal{Y}$ and then adjusting the parameters θ of the generator $\mathbf{g}(\mathbf{y}_i, \mathbf{z}_i; \theta)$ in order to minimize the combination of content and texture loss:

$$\theta_{\mathbf{x_0}} = \underset{\theta}{\operatorname{argmin}} \; E_{\mathbf{z}\sim\mathcal{Z};\, \mathbf{y}\sim\mathcal{Y}} \left[\mathcal{L}_T \left(\mathbf{g}(\mathbf{y}, \mathbf{z}; \theta),\, \mathbf{x_0}\right) + \alpha\, \mathcal{L}_C \left(\mathbf{g}(\mathbf{y}, \mathbf{z}; \theta),\, \mathbf{y}\right) \right]. \tag{8}$$

Here $\mathcal{Z}$ is the same noise distribution as for texture synthesis, $\mathcal{Y}$ empirical distribution on naturals image (obtained from any image collection), and α a parameter that trades off preserving texture/style and content. In practice, we found that learning is surprisingly resilient to overfitting and that it suffices to approximate the distribution on natural images $\mathcal{Y}$ with a very small pool of images (e.g. 16). In fact, our qualitative results degraded using too many example images. We impute this to the fact that stylization by a convolutional architecture uses local operations; since the same local structures exist in different combinations and proportions in different natural images $\mathbf{y}$, it is difficult for local operators to match in all cases the overall statistics of the reference texture $\mathbf{x_0}$, where structures exist in a fixed arbitrary proportion. Despite this limitation, the perceptual quality of the generated stylized images is usually very good, although for some styles we could not match the quality of the original stylization by optimization of (Gatys et al., 2015b).

4. Experiments

Further technical details. The generator network weights were initialized using Xavier's method. Training used *Torch7*'s implementation of Adam (Kingma & Ba, 2014), running it for 2000 iteration. The initial learning rate of 0.1 was reduced by a factor 0.7 at iteration 1000 and then again every 200 iterations. The batch size was set to 16. Similar to (Gatys et al., 2015a), the texture loss uses the layers $L_T = \{\texttt{relu1\_1}, \texttt{relu2\_1}, \texttt{relu3\_1}, \texttt{relu4\_1}, \texttt{relu5\_1}\}$ of VGG-19 and the content loss the layer $L_C = \{\texttt{relu4\_2}\}$. Fully training a single model required just two hours on an NVIDIA Tesla K40, and visually appealing results could be obtained much faster, after just a few epochs.

Texture synthesis. We compare our method to (Gatys et al., 2015a;b) using the popular implementation of (Johnson, 2015), which produces comparable if not better results

Figure 5. Our architecture for image stylization takes the content image and the noise vector as inputs. By scaling the input noise by different factors k we can affect the balance of style and content in the output image without retraining the network.

than the implementation eventually released by the authors. We also compare to the DCGAN (Radford et al., 2015) version of adversarial networks (Goodfellow et al., 2014). Since DCGAN training requires multiple example images for training, we extract those as sliding 64×64 patches from the 256×256 reference texture $\mathbf{x_0}$; then, since DCGAN is fully convolutional, we use it to generate larger 256×256 images simply by inputting a larger noise tensor. Finally, we compare to (Portilla & Simoncelli, 2000).

Figure 4 shows the results obtained by the four methods on two challenging textures of (Portilla & Simoncelli, 2000). Qualitatively, our generator CNN and (Gatys et al., 2015a)'s results are comparable and superior to the other methods; however, the generator CNN is much more efficient (see Sect. 4.1). Figure 1 includes further comparisons between the generator network and (Gatys et al., 2015a) and many others are included in the supplementary material.

Style transfer. For training, example natural images were extracted at random from the ImageNet ILSVRC 2012 data. As for the original method of (Gatys et al., 2015b), we found that style transfer is sensitive to the trade-off parameter α between texture and content loss in (6). At test time this parameter is not available in our method, but we found that the trade-off can still be adjusted by changing the magnitude of the input noise $\mathbf{z}$ (see Figure 5).

We compared our method to the one of (Gatys et al., 2015b; Johnson, 2015) using numerous style and content images, including the ones in (Gatys et al., 2015b), and found that results are qualitatively comparable. Representative comparisons (using a fixed parameter α) are included in Figure 3 and many more in the supplementary material. Other qualitative results are reported in Figure 7.

4.1. Speed and memory

We compare quantitatively the speed of our method and of the iterative optimization of (Gatys et al., 2015a) by measuring how much time it takes for the latter and for our generator network to reach a given value of the loss $\mathcal{L}_T(\mathbf{x}, \mathbf{x_0})$. Figure 6 shows that iterative optimization requires about 10 seconds to generate a sample $\mathbf{x}$ that has a loss comparable to the output $\mathbf{x} = \mathbf{g}(\mathbf{z})$ of our generator network. Since an evaluation of the latter requires ∼20ms, we achieve a $500\times$ speed-up, which is sufficient for *real-time applications* such as video processing. There are two reasons for this significant difference: the generator network is much smaller than the VGG-19 model evaluated at each iteration of (Gatys et al., 2015a), and our method requires a single network evaluation. By avoiding backpropagation, our method also uses significantly less memory (170 MB to generate a 256×256 sample, vs 1100 MB of (Gatys et al., 2015a)).

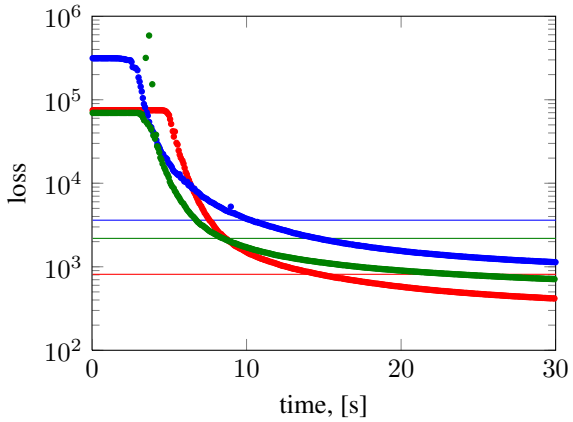

Figure 6. The objective values (log-scale) within the optimization-based method (Gatys et al., 2015a) for three randomly chosen textures are plotted as functions of time. Horizontal lines show the style loss achieved by our feedforward algorithm (mean over several samples) for the same textures. It takes the optimization within (Gatys et al., 2015a) around 10 seconds (500x slower than feedforward generation) to produce samples with comparable loss/objective.

5. Discussion

We have presented a new deep learning approach for texture synthesis and image stylization. Remarkably, the ap-

Figure 7. Stylization results for various styles and inputs (one network per row). Our approach can handle a variety of styles. The generated images are of `256x256` resolution and are computed in about 20 milliseconds each.

proach is able to generate complex textures and images in a purely feed-forward way, while matching the texture synthesis capability of (Gatys et al., 2015a), which is based on multiple forward-backward iterations. In the same vein as (Goodfellow et al., 2014; Dziugaite et al., 2015; Li et al., 2015), the success of this approach highlights the suitability of feed-forward networks for complex data generation and for solving complex tasks in general. The key to this success is the use of complex loss functions that involve different feed-forward architectures serving as "experts" assessing the performance of the feed-forward generator.

While our method generally obtains very good result for texture synthesis, going forward we plan to investigate better stylization losses to achieve a stylization quality comparable to (Gatys et al., 2015b) even for those cases (*e.g.* Figure 3.top) where our current method achieves less impressive results.

References

Chatfield, Ken, Simonyan, Karen, Vedaldi, Andrea, and Zisserman, Andrew. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

Denton, Emily L., Chintala, Soumith, Szlam, Arthur, and Fergus, Robert. Deep generative image models using a laplacian pyramid of adversarial networks. *CoRR*, abs/1506.05751, 2015.

Dosovitskiy, Alexey, Springenberg, Jost Tobias, and Brox, Thomas. Learning to generate chairs with convolutional neural networks. In *Proc. Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 1538–1546, 2015.

Dziugaite, Gintare Karolina, Roy, Daniel M., and Ghahramani, Zoubin. Training generative neural networks via maximum mean discrepancy optimization. *CoRR*, abs/1505.03906, 2015.

Gatys, Leon, Ecker, Alexander S, and Bethge, Matthias. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems, NIPS*, pp. 262–270, 2015a.

Gatys, Leon A., Ecker, Alexander S., and Bethge, Matthias. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015b.

Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems,NIPS*, pp. 2672–2680, 2014.

Gretton, Arthur, Borgwardt, Karsten M, Rasch, Malte, Schölkopf, Bernhard, and Smola, Alex J. A kernel method for the two-sample-problem. In *Advances in neural information processing systems,NIPS*, pp. 513–520, 2006.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. International Conference on Machine Learning, ICML*, pp. 448–456, 2015.

Johnson, Justin. neural-style. `https://github.com/jcjohnson/neural-style`, 2015.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Li, Yujia, Swersky, Kevin, and Zemel, Richard S. Generative moment matching networks. In *Proc. International Conference on Machine Learning, ICML*, pp. 1718–1727, 2015.

Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,CVPR*, pp. 3431–3440, 2015.

Mahendran, Aravindh and Vedaldi, Andrea. Understanding deep image representations by inverting them. 2015.

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

Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Supp.Material.

Zhu, S. C., Wu, Y., and Mumford, D. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *IJCV*, 27(2), 1998.

Supplementary material

Below, we provide several additional qualitative results demonstrating the performance of texture networks and comparing them to Gatys et al. approaches.

5.1. A note on generator architecture

Since the generator is only restricted to produce good images in terms of texture loss, nothing stops it from generating samples with small variance between them. Therefore, if a model archives lower texture loss it does not implicate that this model is preferable. The generator should be powerful enough to combine texture elements, but not too complex to degrade to similar samples. If degrading effect is noticed the noise amount increasing can help in certain cases. Figure 9 shows a bad case, with too much iteration performed. This degrading effect is similar to overfitting but there is no obvious way to control it as there is no analogue of validation set available.

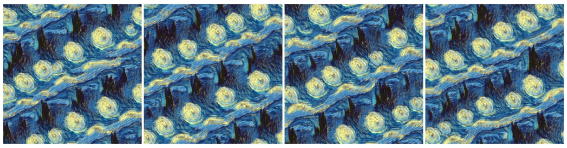

Figure 9. The example of an overfitted generator. Although every "star" differs from each other in details you may notice a common structure. This also illustrates generators love (when overfitted) for diagonal, horizontal and vertical lines as `3x3` filters are used.

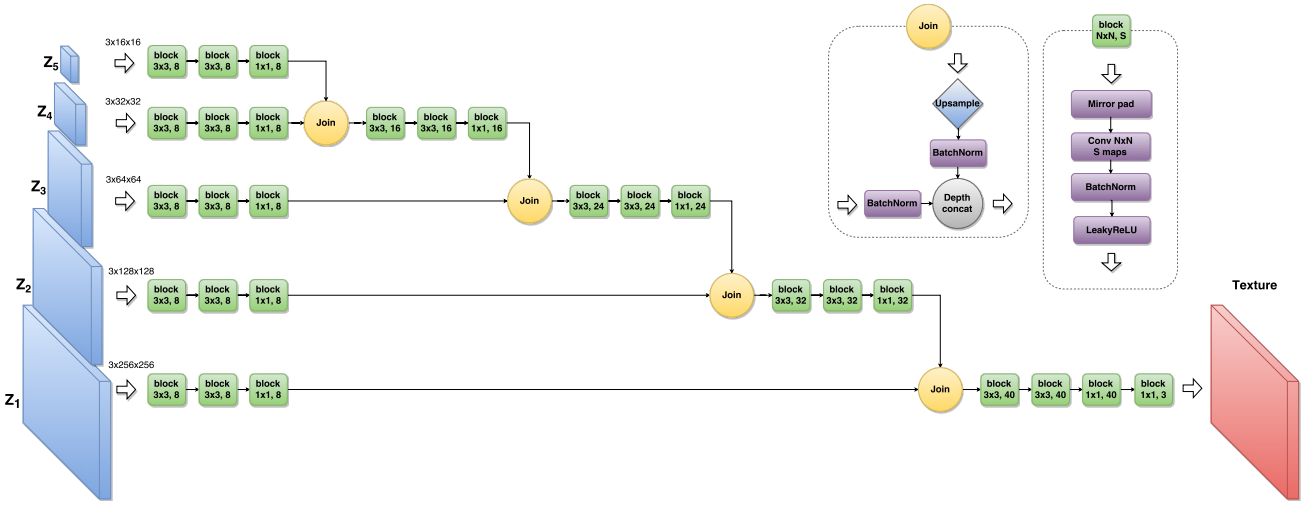

Figure 8. A more exact scheme of the architecture used for texture generation in our experiments.

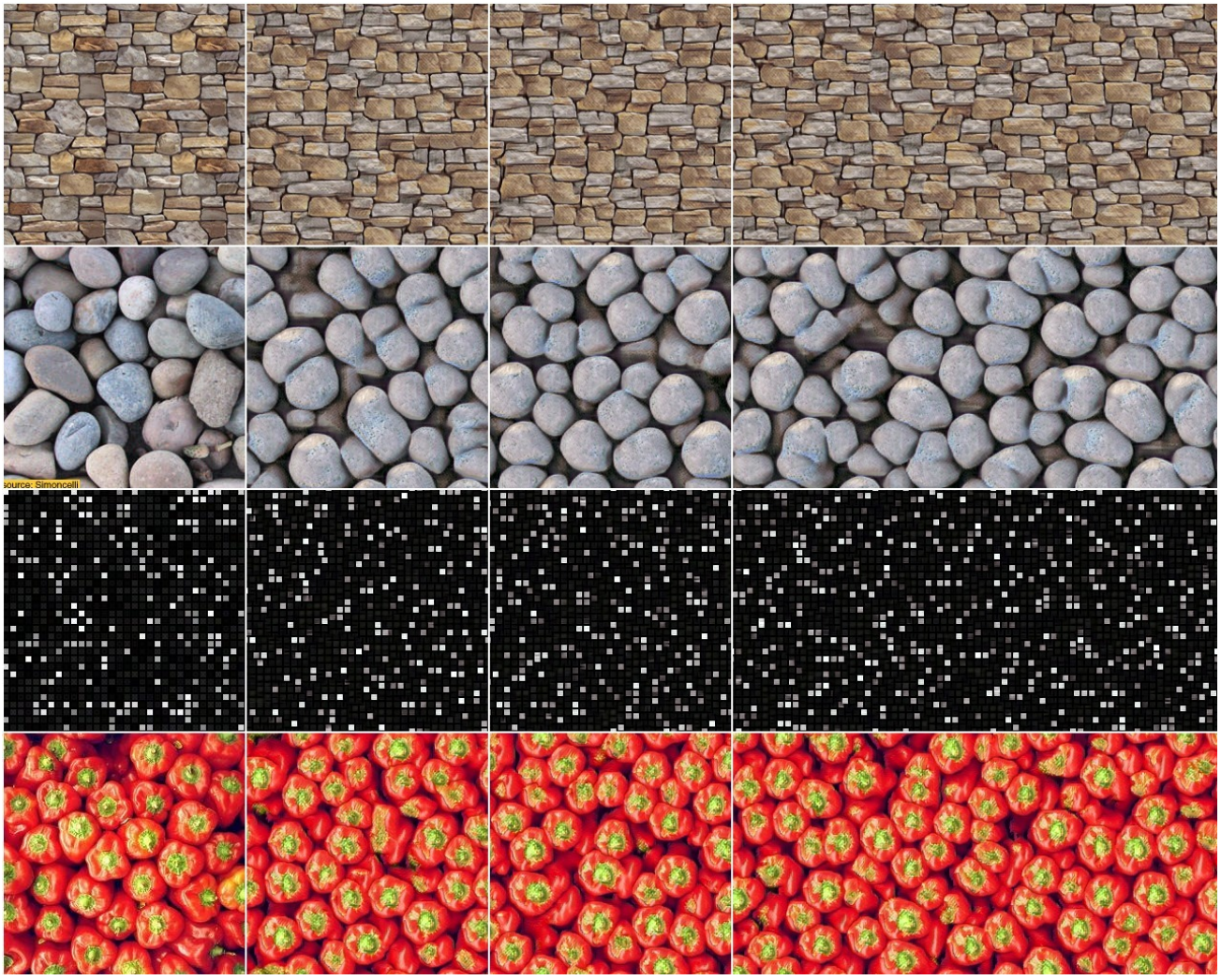

Figure 10. Various samples drawn from three texture networks trained for the samples shown on the left. While training was done for `256x256` samples, in the right column we show the generated textures for a different resolution.

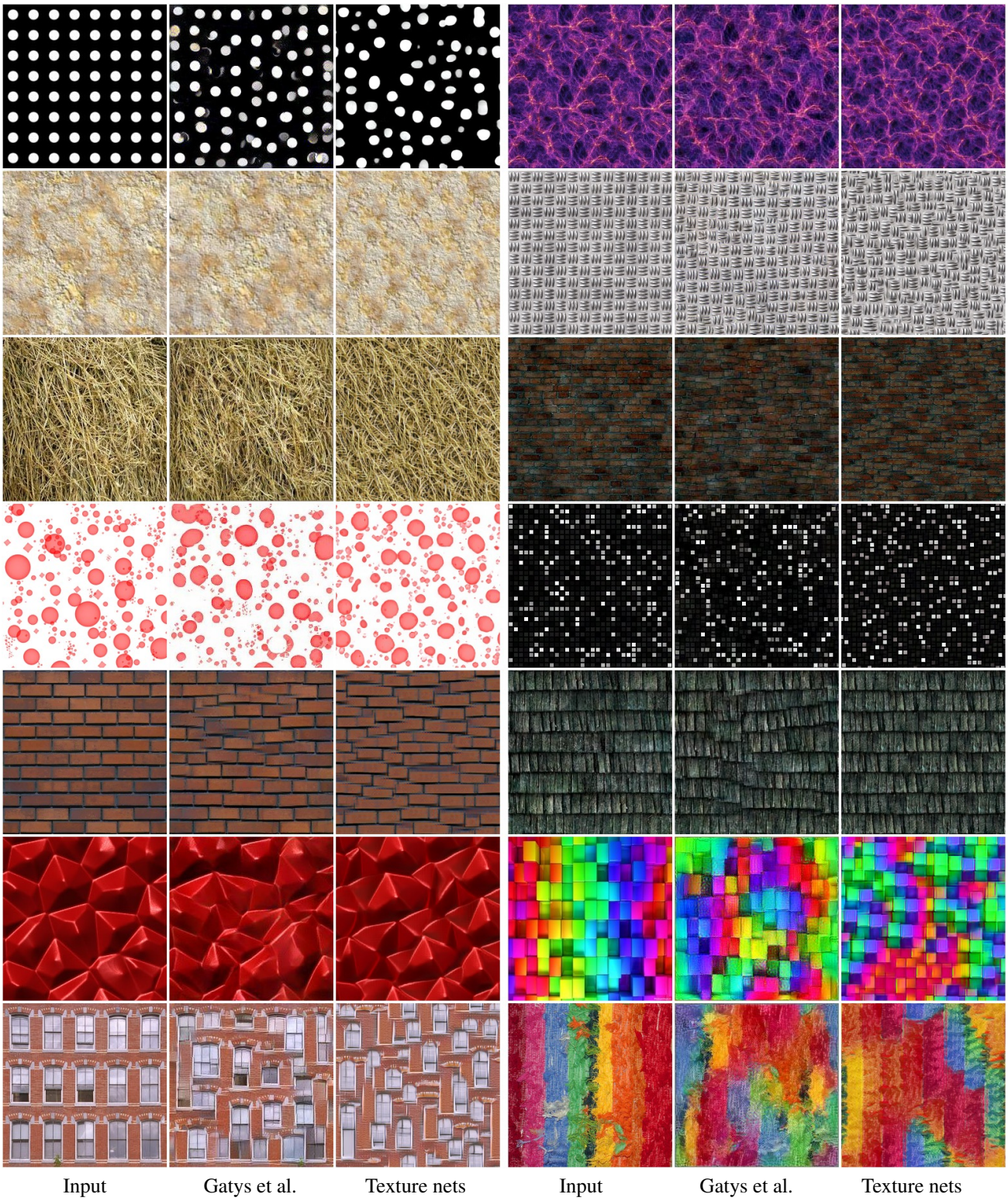

Figure 11. More comparison with Gatys et al. for texture synthesis.

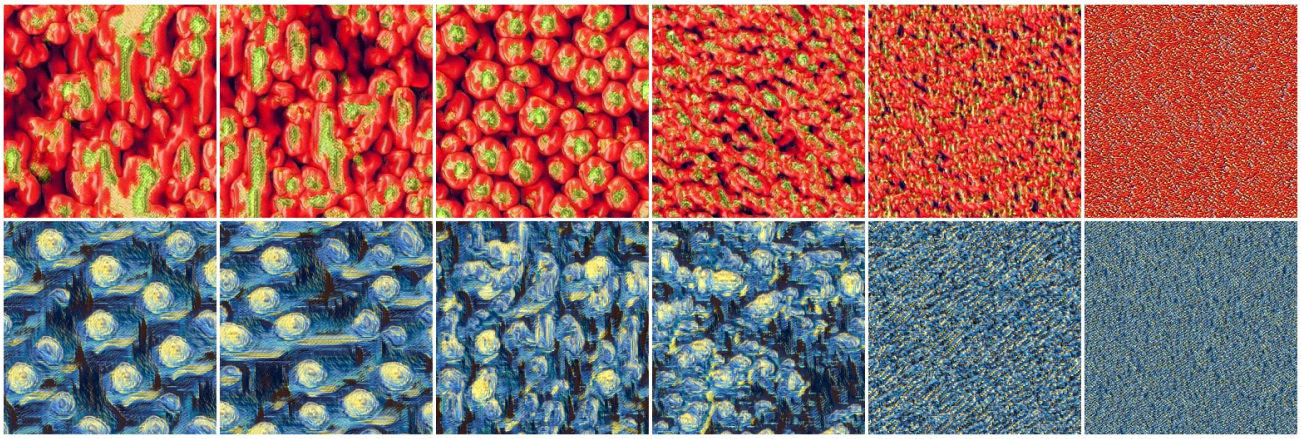

Figure 12. Ablation example: here we generate examples from the pretrained networks after zeroing out all inputs to the multi-scale architecture except for one scale. This let us analyze the processes inside the generator. For peppers image we observe that depth $K = 4$ is enough for generator to perform well. Texture elements in starry night are bigger therefore the deepest input blob is used. Note that the generator is limited by the VGG network capacity and cannot capture larger texture elements than the receptive field of the last convolution layer.

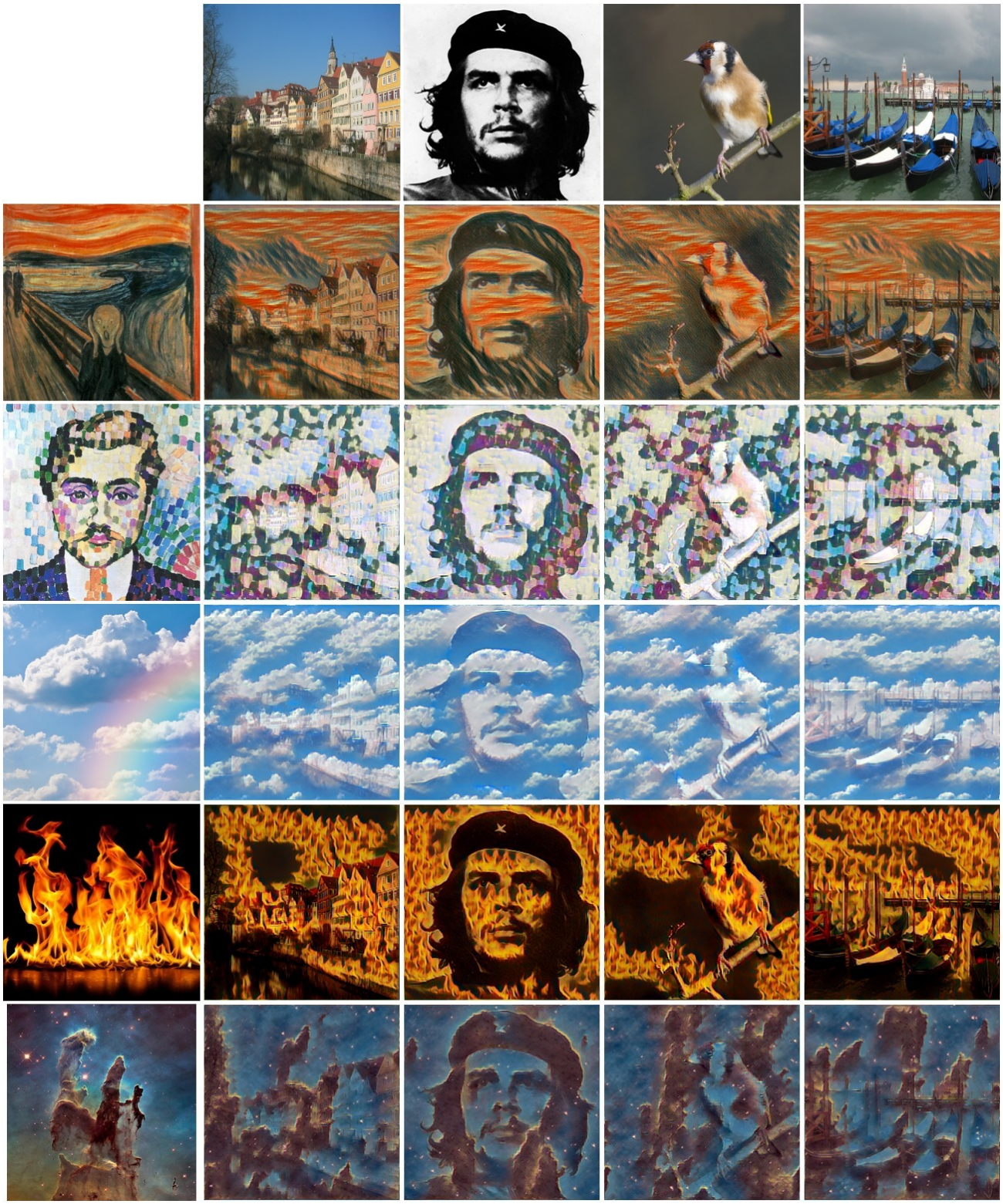

Figure 13. Style transfer results for more styles. As before, each row represents a network trained with particular style with the original style image on the left, and columns correspond to different content images, with the original image on the top.

Figure 14. Being fully convolutional, our architecture is not bound to image resolution it was trained with. Above, a network trained to stylize 256×256 images was applied to 1024×768 reproduction of Pieter Bruegel's "The Tower of Babel".

Figure 15. More style transfer comparisons with Gatys et al. For the styles above, the results of our approach are inferior to Gatys et al. It remains to be seen if more complex losses or deeper networks can narrow down or bridge this performance gap.