Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis

Dmitry Ulyanov
Skolkovo Institute of Science and Technology & Yandex
dmitry.ulyanov@skoltech.ru

Andrea Vedaldi
University of Oxford
vedaldi@robots.ox.ac.uk

Victor Lempitsky
Skolkovo Institute of Science and Technology
lempitsky@skoltech.ru

Abstract

The recent work of Gatys et al.*, who characterized the style of an image by the statistics of convolutional neural network filters, ignited a renewed interest in the texture generation and image stylization problems. While their image generation technique uses a slow optimization process, recently several authors have proposed to learn generator neural networks that can produce similar outputs in one quick forward pass. While generator networks are promising, they are still inferior in visual quality and diversity compared to generation-by-optimization. In this work, we advance them in two significant ways. First, we introduce an instance normalization module to replace batch normalization with significant improvements to the quality of image stylization. Second, we improve diversity by introducing a new learning formulation that encourages generators to sample unbiasedly from the Julesz texture ensemble, which is the equivalence class of all images characterized by certain filter responses. Together, these two improvements take feed forward texture synthesis and image stylization much closer to the quality of generation-via-optimization, while retaining the speed advantage.*

1. Introduction

The recent work of Gatys *et al.* [4, 5], which used deep neural networks for texture synthesis and image stylization to a great effect, has created a surge of interest in this area. Following an eariler work by Portilla and Simoncelli [15], they generate an image by matching the second order moments of the response of certain filters applied to a reference texture image. The innovation of Gatys *et al.* is to use non-linear convolutional neural network filters for this purpose.

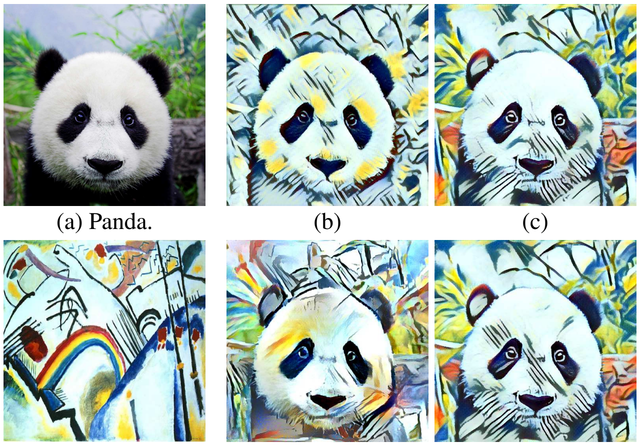

(a) Panda. (b) (c)
(d) Style. (e) (f)

Figure 1: Which panda stylization seems the best to you? Definitely not the variant (b), which has been produced by a state-of-the-art algorithm among methods that take no longer than a second. The (e) picture took several minutes to generate using an optimization process, but the quality is worth it, isn't it? We would be particularly happy if you chose one from the rightmost two examples, which are computed with our new method that aspires to combine the quality of the optimization-based method and the speed of the fast one. Moreover, our method is able to produce diverse stylizations using a single network.

Despite the excellent results, however, the matching process is based on local optimization, and generally requires a considerable amount of time (tens of seconds to minutes) in order to generate a single textures or stylized image.

In order to address this shortcoming, Ulyanov *et al.* [19] and Johnson *et al.* [8] suggested to replace the optimization process with feed-forward generative convolutional networks. In particular, [19] introduced *texture networks* to generate textures of a certain kind, as in [4], or to apply

The source code is available at https://github.com/DmitryUlyanov/texture_nets a certain texture style to an arbitrary image, as in [5]. Once trained, such texture networks operate in a feed-forward manner, three orders of magnitude faster than the optimization methods of [4, 5].

The price to pay for such speed is a reduced performance. For texture synthesis, the neural network of [19] generates good-quality samples, but these are not as diverse as the ones obtained from the iterative optimization method of [4]. For image stylization, the feed-forward results of [19, 8] are qualitatively and quantitatively worse than iterative optimization. In this work, we address both limitations by means of two contributions, both of which extend beyond the applications considered in this paper.

Our first contribution (section 4) is an architectural change that significantly improves the generator networks. The change is the introduction of an **instance-normalization layer** which, particularly for the stylization problem, greatly improves the performance of the deep network generators. This advance significantly reduces the gap in stylisation quality between the feed-forward models and the original iterative optimization method of Gatys *et al.*, both quantitatively and qualitatively.

Our second contribution (section 3) addresses the limited diversity of the samples generated by texture networks. In order to do so, we introduce a new formulation that learns generators that **uniformly sample the Julesz ensemble** [20]. The latter is the equivalence class of images that match certain filter statistics. Uniformly sampling this set guarantees diverse results, but traditionally doing so required slow Monte Carlo methods [20]; Portilla and Simoncelli, and hence Gatys *et al.*, cannot sample from this set, but only find individual points in it, and possibly just one point. Our formulation minimizes the Kullback-Leibler divergence between the generated distribution and a quasi-uniform distribution on the Julesz ensemble. The learning objective decomposes into a loss term similar to Gatys *et al.* minus the entropy of the generated texture samples, which we estimate in a differentiable manner using a non-parametric estimator [12].

We validate our contributions by means of extensive quantitative and qualitative experiments, including comparing the feed-forward results with the gold-standard optimization-based ones (section 5). We show that, combined, these ideas dramatically improve the quality of feed-forward texture synthesis and image stylization, bringing them to a level comparable to the optimization-based approaches.

2. Background and related work

Julesz ensemble. Informally, a *texture* is a family of visual patterns, such as checkerboards or slabs of concrete, that share certain local statistical regularities. The concept was first studied by Julesz [9], who suggested that the visual system discriminates between different textures based on the average responses of certain image filters.

The work of [20] formalized Julesz' ideas by introducing the concept of *Julesz ensemble*. There, an image is a real function $x : \Omega \to \mathbb{R}^3$ defined on a discrete lattice $\Omega = \{1, \dots, H\} \times \{1, \dots, W\}$ and a texture is a distribution $p(x)$ over such images. The local statistics of an image are captured by a bank of (non-linear) filters $F_l : \mathcal{X} \times \Omega \to \mathbb{R}$, $l = 1, \dots, L$, where $F_l(x, u)$ denotes the response of filter F_l at location u on image x. The image x is characterized by the spatial average of the filter responses $\mu_l(x) = \sum_{u \in \Omega} F_l(x, u)/|\Omega|$. The image is perceived as a particular texture if these responses match certain characteristic values $\bar{\mu}_l$. Formally, given the loss function,

$$\mathcal{L}(x) = \sum_{l=1}^{L} (\mu_l(x) - \bar{\mu}_l)^2 \quad (1)$$

the Julesz ensemble is the set of all texture images

$$\mathcal{T}_\epsilon = \{x \in \mathcal{X} : \mathcal{L}(x) \le \epsilon\}$$

that approximately satisfy such constraints. Since all textures in the Julesz ensemble are perceptually equivalent, it is natural to require the texture distribution $p(x)$ to be uniform over this set. In practice, it is more convenient to consider the exponential distribution

$$p(x) = \frac{e^{-\mathcal{L}(x)/T}}{\int e^{-\mathcal{L}(y)/T}\, dy}, \quad (2)$$

where $T > 0$ is a temperature parameter. This choice is motivated as follows [20]: since statistics are computed from spatial averages of filter responses, one can show that, in the limit of infinitely large lattices, the distribution $p(x)$ is zero outside the Julesz ensemble and uniform inside. In this manner, eq. (2) can be though as a uniform distribution over images that have a certain characteristic filter responses $\bar{\mu} = (\bar{\mu}_1, \dots, \bar{\mu}_L)$.

Note also that the texture is completely described by the filter bank $F = (F_1, \dots, F_L)$ and their characteristic responses $\bar{\mu}$. As discussed below, the filter bank is generally fixed, so in this framework different textures are given by different characteristics $\bar{\mu}$.

Generation-by-minimization. For any interesting choice of the filter bank F, sampling from eq. (2) is rather challenging and classically addressed by Monte Carlo methods [20]. In order to make this framework more practical, Portilla and Simoncelli [16] proposed instead to heuristically sample from the Julesz ensemble by the optimization process

$$x^* = \operatorname*{argmin}_{x \in \mathcal{X}} \mathcal{L}(x). \quad (3)$$

If this optimization problem can be solved, the minimizer x^* is by definition a texture image. However, there is no reason why this process should generate fair samples from the distribution $p(x)$. In fact, the only reason why eq. (3) may not simply return *always the same* image is that the optimization algorithm is randomly initialized, the loss function is highly non-convex, and search is local. Only because of this eq. (3) may land on different samples x^* on different runs.

Deep filter banks. Constructing a Julesz ensemble requires choosing a filter bank F. Originally, researchers considered the obvious candidates: Gaussian derivative filters, Gabor filters, wavelets, histograms, and similar [20, 16, 21]. More recently, the work of Gatys *et al.* [4, 5] demonstrated that much superior filters are automatically learned by deep convolutional neural networks (CNNs) even when trained for apparently unrelated problems, such as image classification. In this paper, in particular, we choose for $\mathcal{L}(x)$ the *style loss* proposed by [4]. The latter is the distance between the empirical correlation matrices of deep filter responses in a CNN.[1]

Stylization. The texture generation method of Gatys *et al.* [4] can be considered as a direct extension of the texture generation-by-minimization technique (3) of Portilla and Simoncelli [16]. Later, Gatys *et al.* [5] demonstrated that the same technique can be used to generate an image that mixes the statistics of two other images, one used as a texture template and one used as a content template. Content is captured by introducing a second loss $\mathcal{L}_{\text{cont.}}(x, x_0)$ that compares the responses of deep CNN filters extracted from the generated image x and a content image x_0. Minimizing the combined loss $\mathcal{L}(x) + \alpha\mathcal{L}_{\text{cont.}}(x, x_0)$ yields impressive *artistic images*, where a texture $\bar{\mu}$, defining the artistic style, is fused with the content image x_0.

Feed-forward generator networks. For all its simplicity and efficiency compared to Markov sampling techniques, generation-by-optimization (3) is still relatively slow, and certainly too slow for real-time applications. Therefore, in the past few months several authors [8, 19] have proposed to *learn generator neural networks* $g(z)$ that can directly map random noise samples $z \sim p_z = \mathcal{N}(0, I)$ to a local minimizer of eq. (3). Learning the neural network g amounts to minimizing the objective

$$g^* = \operatorname*{argmin}_g \mathop{\mathbb{E}}_{p_z} \mathcal{L}(g(z)). \tag{4}$$

While this approach works well in practice, it shares the same important limitation as the original work of Portilla and Simoncelli: there is no guarantee that samples generated by g^* would be fair samples of the texture distribution (2). In practice, as we show in the paper, such samples tend in fact to be not diverse enough.

Both [8, 19] have also shown that similar generator networks work also for stylization. In this case, the generator $g(x_0, z)$ is a function of the content image x_0 and of the random noise z. The network g is learned to minimize the sum of texture loss and the content loss:

$$g^* = \operatorname*{argmin}_g \mathop{\mathbb{E}}_{p_{x_0}, p_z} [\mathcal{L}(g(x_0, z)) + \alpha\mathcal{L}_{\text{cont.}}(g(x_0, z), x_0)]. \tag{5}$$

Alternative neural generator methods. There are many other techniques for image generation using deep neural networks.

The Julesz distribution is closely related to the FRAME maximum entropy model of [21], as well as to the concept of *Maximum Mean Discrepancy* (MMD) introduced in [7]. Both FRAME and MMD make the observation that a probability distribution $p(x)$ can be described by the expected values $\mu_\alpha = \mathbb{E}_{x \sim p(x)}[\phi_\alpha(x)]$ of a sufficiently rich set of statistics $\phi_\alpha(x)$. Building on these ideas, [14, 3] construct generator neural networks g with the goal of minimizing the discrepancy between the statistics averaged over a batch of generated images $\sum_{i=1}^N \phi_\alpha(g(z_i))/N$ and the statistics averaged over a traning set $\sum_{i=1}^M \phi_\alpha(x_i)/M$. The resulting networks g are called *Moment Matching Networks* (MMN).

An important alternative methodology is based on the concept of Generative Adversarial Networks (GAN; [6]). This approach trains, together with the generator network $g(z)$, a second *adversarial* network $f(\cdot)$ that attempts to distinguish between generated samples $g(z), z \sim \mathcal{N}(0, I)$ and real samples $x \sim p_{\text{data}}(x)$. The adversarial model f can be used as a measure of quality of the generated samples and used to learn a better generator g. GAN are powerful but notoriously difficult to train. A lot of research is has recently focused on improving GAN or extending it. For instance, LAPGAN [2] combines GAN with a Laplacian pyramid and DCGAN [17] optimizes GAN for large datasets.

3. Julesz generator networks

This section describes our first contribution, namely a method to learn networks that draw samples from the Julesz ensemble modelling a texture (section 2), which is an intractable problem usually addressed by slow Monte Carlo methods [21, 20]. Generation-by-optimization, popularized by Portilla and Simoncelli and Gatys *et al.*, is faster, but can only find one point in the ensemble, not sample from it, with scarce sample diversity, particularly when used to train feed-forward generator networks [8, 19].

Here, we propose a new formulation that allows to train generator networks that *sample the Julesz ensemble*, gener-

[1] Note that such matrices are obtained by averaging local non-linear filters: these are the outer products of filters in a certain layer of the nerual network. Hence, the style loss of Gatys *et al.* is in the same form as eq. (1).

ating images with high visual fidelity as well as high diversity.

A generator network [6] maps an i.i.d. noise vector $z \sim \mathcal{N}(0, I)$ to an image $x = g(z)$ in such a way that x is ideally a sample from the desired distribution $p(x)$. Such generators have been adopted for texture synthesis in [19], but without guarantees that the learned generator $g(z)$ would indeed sample a particular distribution.

Here, we would like to sample from the Gibbs distribution (2) defined over the Julesz ensemble. This distribution can be written compactly as $p(x) = Z^{-1} e^{-\mathcal{L}(x)/T}$, where $Z = \int e^{-\mathcal{L}(x)/T}\, dx$ is an intractable normalization constant.

Denote by $q(x)$ the distribution induced by a generator network g. The goal is to make the target distribution p and the generator distribution q as close as possible by minimizing their Kullback-Leibler (KL) divergence:

$$
\begin{aligned}
KL(q\|p) &= \int q(x) \ln \frac{q(x)Z}{p(x)}\, dx \\
&= \frac{1}{T} \mathop{\mathbb{E}}_{x\sim q(x)} \mathcal{L}(x) + \mathop{\mathbb{E}}_{x\sim q(x)} \ln q(x) + \ln(Z) \qquad (6) \\
&= \frac{1}{T} \mathop{\mathbb{E}}_{x\sim q(x)} \mathcal{L}(x) - H(q) + \text{const.}
\end{aligned}
$$

Hence, the KL divergence is the sum of the expected value of the style loss $\mathcal{L}$ and the negative entropy of the generated distribution q.

The first term can be estimated by taking the expectation over generated samples:

$$
\mathop{\mathbb{E}}_{x\sim q(x)} \mathcal{L}(x) = \mathop{\mathbb{E}}_{z\sim \mathcal{N}(0,I)} \mathcal{L}(g(z)). \qquad (7)
$$

This is similar to the reparametrization trick of [11] and is also used in [8, 19] to construct their learning objectives.

The second term, the negative entropy, is harder to estimate accurately, but simple estimators exist. One which is particularly appealing in our scenario is the Kozachenko-Leonenko estimator [12]. This estimator considers a *batch* of N samples $x_1, \dots, x_n \sim q(x)$. Then, for each sample x_i, it computes the distance ρ_i to its nearest neighbour in the batch:

$$
\rho_i = \min_{j\neq i} \|x_i - x_j\|. \qquad (8)
$$

The distances ρ_i can be used to approximate the entropy as follows:

$$
H(q) \approx \frac{D}{N} \sum_{i=1}^{N} \ln \rho_i + \text{const.} \qquad (9)
$$

where $D = 3WH$ is the number of components of the images $x \in \mathbb{R}^{3\times W\times H}$.

An energy term similar to (6) was recently proposed in [10] for improving the diversity of a generator network in a adversarial learning scheme. While the idea is superficially similar, the application (sampling the Julesz ensemble) and instantiation (the way the entropy term is implemented) are very different.

Learning objective. We are now ready to define an objective function $E(g)$ to learn the generator network g. This is given by substituting the expected loss (7) and the entropy estimator (9), computed over a batch of N generated images, in the KL divergence (6):

$$
\begin{aligned}
E(g) = \frac{1}{N} \sum_{i=1}^{N} \Big[&\frac{1}{T} \mathcal{L}(g(z_i)) \\
&- \lambda \ln \min_{j\neq i} \|g(z_i) - g(z_j)\| \Big] \qquad (10)
\end{aligned}
$$

The batch itself is obtained by drawing N samples $z_1, \dots, z_n \sim \mathcal{N}(0, I)$ from the noise distribution of the generator. The first term in eq. (10) measures how closely the generated images $g(z_i)$ are to the Julesz ensemble. The second term quantifies the lack of diversity in the batch by mutually comparing the generated images.

Learning. The loss function (10) is in a form that allows optimization by means of Stochastic Gradient Descent (SGD). The algorithm samples a batch $z_1, \dots, z_n$ at a time and then descends the gradient:

$$
\begin{aligned}
\frac{1}{N} \sum_{i=1}^{N} \Big[&\frac{d\mathcal{L}}{dx^\top} \frac{dg(z_i)}{d\theta^\top} \\
&- \frac{\lambda}{\rho_i} (g(z_i) - g(z_{j_i^*}))^\top \left(\frac{dg(z_i)}{d\theta^\top} - \frac{dg(z_{j_i^*})}{d\theta^\top} \right) \Big] \qquad (11)
\end{aligned}
$$

where θ is the vector of parameters of the neural network g, the tensor image x has been implicitly vectorized and j_i^* is the index of the nearest neighbour of image i in the batch.

4. Stylization with instance normalization

The work of [19] showed that it is possible to learn high-quality texture networks $g(z)$ that generate images in a Julesz ensemble. They also showed that it is possible to learn good quality stylization networks $g(x_0, z)$ that apply the style of a fixed texture to an arbitrary content image x_0.

Nevertheless, the stylization problem was found to be harder than the texture generation one. For the stylization task, they found that learning the model from too many example content images x_0, say more than 16, yielded poorer *qualitative* results than using a smaller number of such examples. Some of the most significant errors appeared along the border of the generated images, probably due to padding and other boundary effects in the generator network. We conjectured that these are symptoms of a learning problem too difficult for their choice of neural network architecture.

A simple observation that may make learning simpler is that the result of stylization should not, in general, depend on the contrast of the content image but rather should match the contrast of the texture that is being applied to it. Thus, the generator network should discard contrast information in the content image x_0. We argue that learning to discard contrast information by using standard CNN building block is unnecessarily difficult, and is best done by adding a suitable layer to the architecture.

To see why, let $x \in \mathbb{R}^{N \times C \times W \times H}$ be an input tensor containing a batch of N images. Let x_{nijk} denote its $nijk$-th element, where k and j span spatial dimensions, i is the feature channel (i.e. the color channel if the tensor is an RGB image), and n is the index of the image in the batch. Then, contrast normalization is given by:

$$\begin{aligned} y_{nijk} &= \frac{x_{nijk} - \mu_{ni}}{\sqrt{\sigma_{ni}^2 + \epsilon}}, \\ \mu_{ni} &= \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{nilm}, \\ \sigma_{ni}^2 &= \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} (x_{nilm} - \mu_{ni})^2. \end{aligned} \tag{12}$$

It is unclear how such as function could be implemented as a sequence of standard operators such as ReLU and convolution.

On the other hand, the generator network of [19] does contain a normalization layers, and precisely *batch normalization* (BN) ones. The key difference between eq. (12) and batch normalization is that the latter applies the normalization to a whole batch of images instead of single ones:

$$\begin{aligned} y_{nijk} &= \frac{x_{nijk} - \mu_{i}}{\sqrt{\sigma_{i}^2 + \epsilon}}, \\ \mu_{i} &= \frac{1}{HWN} \sum_{n=1}^{N} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{nilm}, \\ \sigma_{i}^2 &= \frac{1}{HWN} \sum_{n=1}^{N} \sum_{l=1}^{W} \sum_{m=1}^{H} (x_{nilm} - \mu_{i})^2. \end{aligned} \tag{13}$$

We argue that, for the purpose of stylization, the normalization operator of eq. (12) is preferable as it can normalize each individual content image x_0.

While some authors call layer eq. (12) contrast normalization, here we refer to it as *instance normalization* (IN) since we use it as a drop-in replacement for batch normalization operating on individual instances instead of the batch as a whole. Note in particular that this means that instance normalization is applied throughout the architecture, not just at the input image—fig. 2 shows the benefit of doing so.

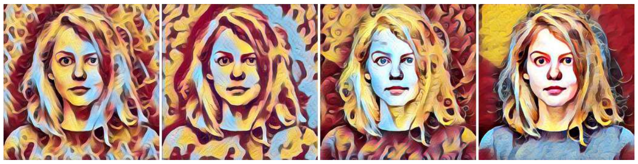
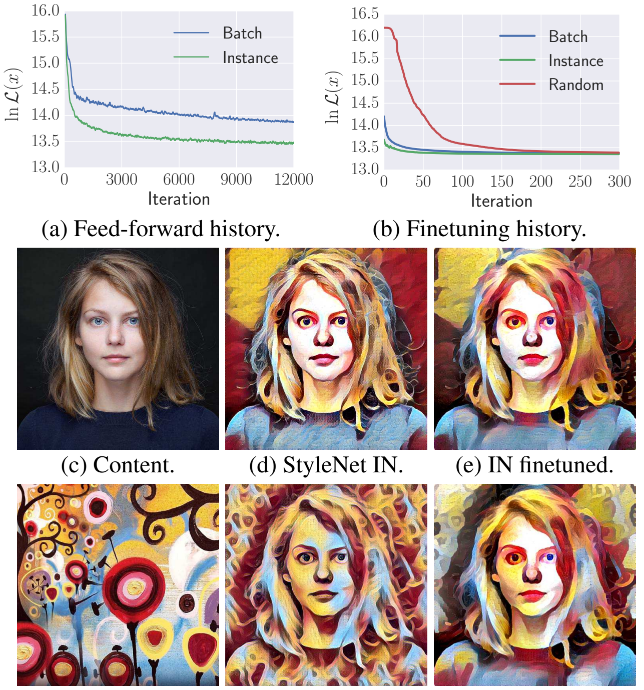


Figure 2: Comparison of normalization techniques in image stylization. From left to right: BN, cross-channel LRN at the first layer, IN at the first layer, IN throughout.



(a) Feed-forward history. (b) Finetuning history.

(c) Content. (d) StyleNet IN. (e) IN finetuned.

(f) Style. (g) StyleNet BN. (h) BN finetuned.

Figure 3: (a) learning objective as a function of SGD iterations for StyleNet IN and BN. (b) Direct optimization of the Gatys *et al*. for this example image starting from the result of StyleNet IN and BN. (d,g) Result of StyleNet with instance (d) and batch normalization (g). (e,h) Result of finetuning the Gatys *et al*. energy.

Another similarity with BN is that each IN layer is followed by a scaling and bias operator $s \odot \mathbf{x} + b$. A difference is that the IN layer is applied at test time as well, unchanged, whereas BN is usually switched to use accumulated mean and variance instead of computing them over the batch.

IN appears to be similar to the layer normalization method introduced in [1] for recurrent networks, although it is not clear how they handle spatial data. Like theirs, IN is a generic layer, so we tested it in classification problems as well. In such cases, it still work surprisingly well, but not as well as batch normalization (e.g. AlexNet [13] IN has 2-3% worse top-1 accuracy on ILSVRC [18] than AlexNet BN).

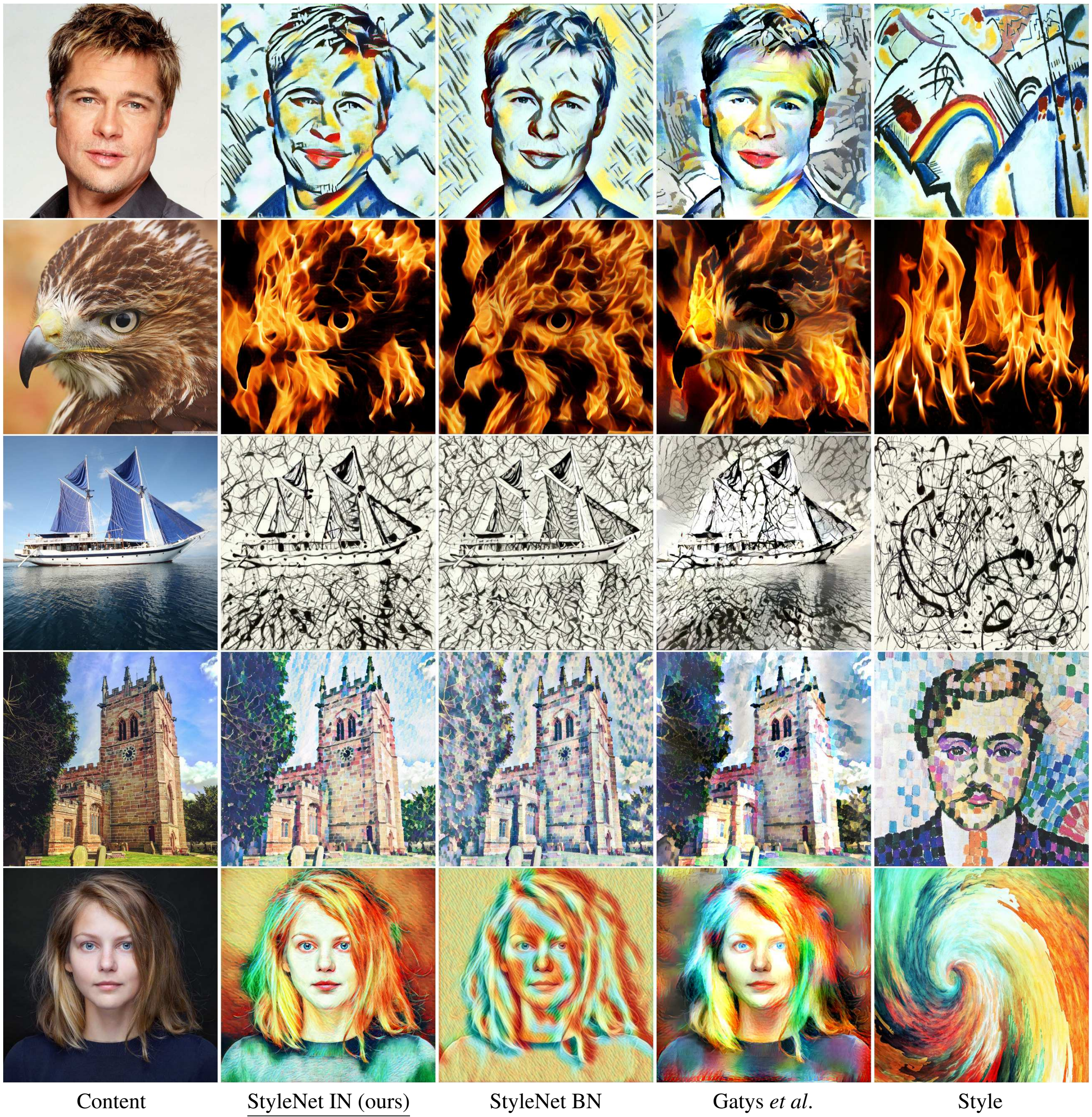


Figure 4: Stylization results obtained by applying different textures (rightmost column) to different content images (leftmost column). Three methods are compared: StyleNet IN, StyleNet BN, and iterative optimization. StyleNet BN is similar to [19] and [8] but trained on larger images (`512x` compared to `256x` in [19, 8]) for a fair comparison with StyleNet IN. We compare to original [19, 8] in supmat.

5. Experiments

In this section, after discussing the technical details of the method, we evaluate our new texture network architectures using instance normalization, and then investigate the ability of the new formulation to learn diverse generators.

5.1. Technical details

Network architecture. Among two generator network architectures, proposed previously in [19, 8], we choose the residual architecture from [8] for all our style transfer experiments. We also experimented with architecture from [19] and observed a similar improvement with our method, but use the one from [8] for convenience. We call it *StyleNet* with a postfix BN if it is equipped with batch normalization or IN for instance normalization.

For texture synthesis we compare two architectures: the multiscale fully-convolutional architecture from [19] (TextureNetV1) and the one we design to have a very large receptive field (TextureNetV2). TextureNetV2 takes a noise vector of size 256 and first transforms it with two fully-connected layers. The output is then reshaped to a 4×4

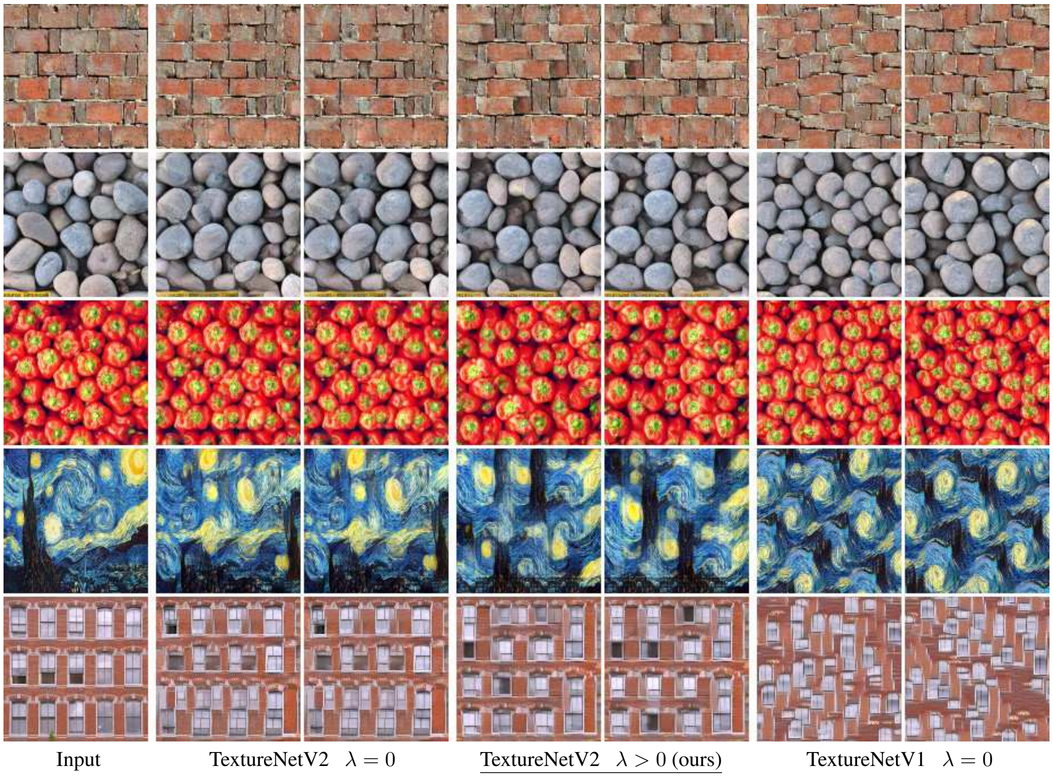

Input
TextureNetV2 $\lambda = 0$
TextureNetV2 $\lambda > 0$ (ours)
TextureNetV1 $\lambda = 0$

Figure 5: The textures generated by the high capacity Texture Net V2 without diversity term ($\lambda = 0$ in eq. (10)) are nearly identical. The low capacity TextureNet V1 of [19] achieves diversity, but has sometimes poor results. TextureNet V2 with diversity is the best of both worlds.

image and repeatedly upsampled with fractionally-strided convolutions similar to [17]. More details can be found in the supplementary material.

Weight parameters. In practice, for the case of $\lambda > 0$, entropy loss and texture loss in eq. (10) should be weighted properly. As only the value of $T\lambda$ is important for optimization we assume $\lambda = 1$ and choose T from the set of three values $(5, 10, 20)$ for texture synthesis (we pick the higher value among those not leading to artifacts – see our discussion below). We fix $T = 10000$ for style transfer experiments. For texture synthesis, similarly to [19], we found useful to normalize gradient of the texture loss as it passes back through the VGG-19 network. This allows rapid convergence for stochastic optimization but implicitly alters the objective function and requires temperature to be adjusted. We observe that for textures with flat lightning high entropy weight results in brightness variations over the image fig. 7. We hypothesize this issue can be solved if either more clever distance for entropy estimation is used or an image prior introduced.

5.2. Effect of instance normalization

In order to evaluate the impact of replacing batch normalization with instance normalization, we consider first the problem of *stylization*, where the goal is to learn a generator $x = g(x_0, z)$ that applies a certain texture style to the content image x_0 using noise z as "random seed". We set $\lambda = 0$ for which generator is most likely to discard the noise.

The StyleNet IN and StyleNet BN are compared in fig. 3. Panel fig. 3.a shows the training objective (5) of the networks as a function of the SGD training iteration. The objective function is the same, but StyleNet IN converges much faster, suggesting that it can solve the stylization problem more easily. This is confirmed by the stark difference in the qualitative results in panels (d) end (g). Since the StyleNets are trained to minimize in one shot the same objective as the iterative optimization of Gatys *et al.*, they can be used to *initialize* the latter algorithm. Panel (b) shows the result of applying the Gatys *et al.* optimization starting from their random initialization and the output of the two StyleNets. Clearly both networks start much closer to an optimum than random noise, and IN closer than BN. The difference is qualitatively large: panels (e) and (h) show the change in the StyleNets output after finetuning by iterative optimization of the loss, which has a small effect for the IN variant, and a much larger one for the BN one.

Similar results apply in general. Other examples are shown in fig. 4, where the IN variant is far superior to BN and much closer to the results obtained by the much slower

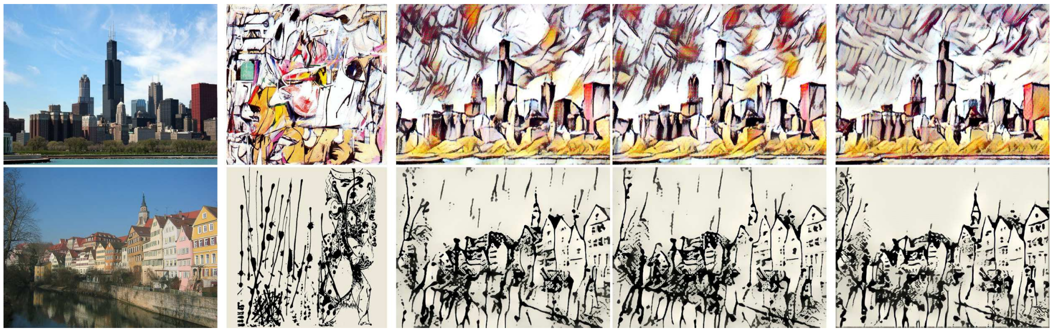


Figure 6: The StyleNetV2 $g(x_0, z)$, trained with diversity $\lambda > 0$, generates substantially different stylizations for different values of the input noise z. With $\lambda = 0$ generator tends to ignore noise channels when trained for sufficiently long time thus producing almost the same stylization for different noise z.

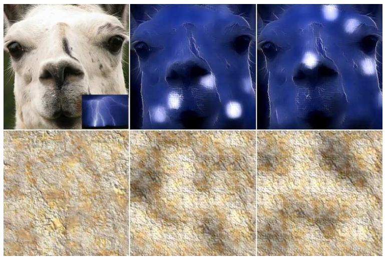


Figure 7: Negative examples. If the diversity term λ is too high for the learned style, the generator tends to generate artifacts in which brightness is changed locally (spotting) instead of (or as well as) changing the structure.

iterative method of Gatys *et al*. StyleNets are trained on images of a fixed sized, but since they are convolutional, they can be applied to arbitrary sizes. In the figure, the top tree images are processed at 512×512 resolution and the bottom two at 1024×1024. In general, we found that higher resolution images yield visually better stylization results.

While instance normalization works much better than batch normalization for stylization, for texture synthesis the two normalization methods perform equally well. This is consistent with our intuition that IN helps in normalizing the information coming from content image x_0, which is highly variable, whereas it is not important to normalize the texture information, as each model learns only one texture style.

5.3. Effect of the diversity term

Having validated the IN-based architecture, we evaluate now the effect of the entropy-based diversity term in the objective function (10).

The experiment in fig. 5 starts by considering the problem of texture generation. We compare the new high-capacity TextureNetV2 and the low-capacity TextureNetsV1 texture synthesis networks. The low-capacity model is the same as [19]. This network was used there in order to force the network to learn a non-trivial dependency on the input noise, thus generating diverse outputs even though the learning objective of [19], which is the same as eq. (10) with diversity coefficient $\lambda = 0$, tends to suppress diversity. The results in fig. 5 are indeed diverse, but sometimes of low quality. This should be contrasted with TextureNetV2, the high-capacity model: its visual fidelity is much higher, but, by using the same objective function [19], the network learns to generate a single image, as expected. TextureNetV2 with the new diversity-inducing objective ($\lambda > 0$) is the best of both worlds, being both high-quality and diverse.

The experiment in fig. 6 assesses the effect of the diversity term in the stylization problem. The results are similar to the ones for texture synthesis and the diversity term effectively encourages the network to learn to produce different results based on the input noise.

One difficultly with texture and stylization networks is that the entropy loss weight λ must be tuned for each learned texture model. Choosing λ too small may fail to learn a diverse generator, and setting it too high may create artifacts, as shown in fig. 7.

6. Summary

This paper advances feed-forward texture synthesis and stylization networks in two significant ways. It introduces instance normalization, an architectural change that makes training stylization networks easier and allows the training process to achieve much lower loss levels. It also introduces a new learning formulation for training generator networks to sample uniformly from the Julesz ensemble, thus explicitly encouraging diversity in the generated outputs. We show that both improvements lead to noticeable improvements of the generated stylized images and textures, while keeping the generation runtimes intact.

Acknowledgements. VL was supported by the Ministry of Education and Science of the Russian Federation (grant 14.756.31.0001).

References

[1] L. J. Ba, R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 5

[2] E. L. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, pages 1486–1494, 2015. 3

[3] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. In *UAI*, pages 258–267. AUAI Press, 2015. 3

[4] L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems, NIPS*, pages 262–270, 2015. 1, 2, 3

[5] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. 1, 2, 3

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014. 3, 4

[7] A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola. A kernel method for the two-sample-problem. In *Advances in neural information processing systems,NIPS*, pages 513–520, 2006. 3

[8] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II*, pages 694–711, 2016. 1, 2, 3, 4, 6

[9] B. Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290(5802):91–97, 1981. 2

[10] T. Kim and Y. Bengio. Deep directed generative models with energy-based probability estimation. *arXiv preprint arXiv:1606.03439*, 2016. 4

[11] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. 4

[12] L. F. Kozachenko and N. N. Leonenko. Sample estimate of the entropy of a random vector. *Probl. Inf. Transm.*, 23(1-2):95–101, 1987. 2, 4

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 5

[14] Y. Li, K. Swersky, and R. S. Zemel. Generative moment matching networks. In *Proc. International Conference on Machine Learning, ICML*, pages 1718–1727, 2015. 3

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

[16] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *IJCV*, 2000. 2, 3

[17] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. 3, 7

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 5

[19] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1349–1357, 2016. 1, 2, 3, 4, 5, 6, 7, 8

[20] S. C. Zhu, X. W. Liu, and Y. N. Wu. Exploring texture ensembles by efficient markov chain monte carlotoward a atrichromacyo theory of texture. *PAMI*, 2000. 2, 3

[21] S. C. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *IJCV*, 27(2), 1998. 3