Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks

Chuan Li(✉) and Michael Wand(✉)

Institut for Informatik, University of Mainz, Mainz, Germany
cl.chuanli@gmail.com, wandm@uni-mainz.de

Abstract. This paper proposes Markovian Generative Adversarial Networks (MGANs), a method for training generative networks for efficient texture synthesis. While deep neural network approaches have recently demonstrated remarkable results in terms of synthesis quality, they still come at considerable computational costs (minutes of run-time for low-res images). Our paper addresses this efficiency issue. Instead of a numerical deconvolution in previous work, we precompute a feed-forward, strided convolutional network that captures the feature statistics of *Markovian patches* and is able to directly generate outputs of arbitrary dimensions. Such network can directly decode brown noise to realistic texture, or photos to artistic paintings. With adversarial training, we obtain quality comparable to recent neural texture synthesis methods. As no optimization is required at generation time, our run-time performance (0.25 M pixel images at 25 Hz) surpasses previous neural texture synthesizers by a significant margin (at least 500 times faster). We apply this idea to texture synthesis, style transfer, and video stylization.

Keywords: Texture synthesis · Adversarial generative networks

1 Introduction

Image synthesis is a classical problem in computer graphics and vision [5]. The key challenges are to capture the structure of complex classes of images in a concise, learnable model, and to find efficient algorithms for learning such models and synthesizing new image data. Most traditional "*texture synthesis*" methods address the complexity constraints using Markov random field (MRF) models that characterize images by statistics of local patches of pixels.

Recently, generative models based on deep neural networks have shown exciting new perspectives for image synthesis [7,8]. Deep architectures capture appearance variations in object classes beyond the abilities of pixel-level approaches. However, there are still strong limitations of how much structure can

Electronic supplementary material The online version of this chapter (doi:10.1007/978-3-319-46487-9_43) contains supplementary material, which is available to authorized users.

B. Leibe et al. (Eds.): ECCV 2016, Part III, LNCS 9907, pp. 702–716, 2016.
DOI: 10.1007/978-3-319-46487-9_43

be learned from limited training data. This currently leaves us with two main classes of "deep" generative models: (1) *full-image models* that generate whole images [3,8], and (2) *Markovian models* that also synthesize textures [7,15].

The first class, full-image models, are often designed as specially trained auto-encoders [12]. Results are impressive but limited to rather small images (typically around 64×64 pixels). The second class, the deep Markovian models, capture the statistics of local patches only and assemble them to high-resolution images. Consequently, the fidelity of details is good, but additional guidance is required if non-trivial global structure should be reproduced [1,5,7,9,15]. Our paper addresses this second approach of deep Markovian texture synthesis.

Previous neural methods of this type [7,15] are built upon a deconvolutional framework [18,25]. This naturally provides blending of patches and permits reusing the intricate, emergent multi-level feature representations of large, discriminatively trained neural networks like the VGG network [21], repurposing them for image synthesis. As a side note, we will later observe that this is actually crucial for high-quality result (Fig. 10). Gatys et al. [7] pioneer this approach by modeling patch statistics with a global Gaussian models of the higher-level feature vectors, and Li and Wand [15] utilize dictionaries of extended local patches of neural activation, trading-off flexibility for visual realism. Unfortunately, the run-time costs of the deconvolution approach are very high, requiring iterative back-propagation in order to estimate a pre-image (pixels) of the feature activations (higher network layer). In the case of [15], a high-end GPU needs several minutes to synthesize low-resolution images (such as a 512-by-512 pixels image).

The objective of our paper is therefore to improve the efficiency of deep Markovian texture synthesis. The key idea is to precompute the inversion of the network by fitting a strided convolutional network [20] to the inversion process, which operates purely in a feed-forward fashion. Despite being trained on patches of a fixed size, the resulting network can generate images of arbitrary dimension, yielding an efficient texture synthesizer of a specific style[1].

We train the convolutional network using adversarial training [20], which permits maintaining image quality similar to the original, expensive optimization approach. As result, we obtain significant speed-up: Our GPU implementation computes 512×512 images within 40 ms (on an nVidia TitanX). The key limitation, of course, is to precompute the feed-forward convolutional network for each texture style. Nonetheless, this is still an attractive trade-off for many potential applications, for example from the area of artistic image or video stylization.

2 Related Work

Deconvolutional neural networks have been introduced to visualize deep features and object classes. Zeiler and Fergus [25] back-project neural activations to highlight pixels. Mahendran and Vedaldi [17] reconstruct images from the neural encoding in intermediate layers. Recently, effort are made to improve the

[1] See supplementary material and code at: https://github.com/chuanli11/MGANs.

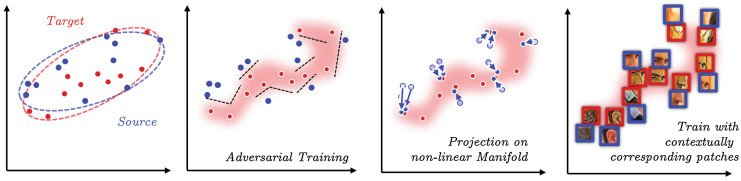

Fig. 1. Motivation: real world data does not always comply with a Gaussian distribution (first), but a complex nonlinear manifold (second). We adversarially learn a mapping to project contextually related patches to that manifold.

efficiency and accuracy of visualization [19,24]. Mordvintsev et al. have raised wide attention by showing how deconvolution of class-specific activations can create hallucinogenic imagery from discriminative networks [18]. The astonishing complexity of the obtained visual patterns has immediately spurred hope for new generative models: Gatys et al. [6,7] drove deconvolution by global covariance statistics of feature vectors on higher network layers, obtaining unprecedented results in artistic style transfer. However, enforcing per-feature-vector statistics permits a mixing of feature patterns that never appear in actual images and limit plausibility of the learned texture. This can be partially addressed by replacing point-wise feature statistics by statistics of spatial patches of feature activations [15]. This permits photo-realistic synthesis in some cases, but also reduces invariance because the simplistic dictionary of patches introduces rigidity.

Full image methods employ specially trained auto-encoders as generative networks [12]. For example, the Generative Adversarial Networks use two networks, one as the discriminator and other as the generator, to iteratively improve the model by playing a minimax game [8]. This model is extended to work with a Laplacian pyramid [3]. Very recently, Radford et al. [20] propose a set of architectural refinements[2] that stabilized the performance of this model, and show that the generators have vector arithmetic properties. One important strength of adversarial networks is that it offers perceptual metrics [4] that allows auto-encoders to be training more efficiently.

In very recent, two concurrent work, Ulyanov et al. [22] and Johnson et al. [10] propose fast implementations of Gatys et al.'s approach. Both of their methods employ precomputed decoders trained with a perceptual texture loss and obtain significant run-time benefits (higher decoder complexity reduces their speed-up a bit). The main difference in our paper is the use of Li and Wand's [15] feature-patch statistics as opposed to learning Gaussian distributions of individual feature vectors, which provides some benefits in reproducing textures more faithfully.

3 Model

Let us first conceptually motive our method. Statistics based methods [7,22] match the distributions of source (input photo or noise signal) and target

[2] Strided convolution, ReLUs, batch normalization, removing fully connected layers.

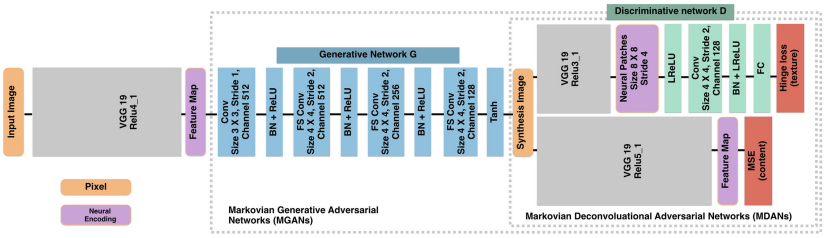

Fig. 2. Our model contains a generative network (blue blocks) and a discriminative network (green blocks). We apply the discriminative training on Markovian neural patches (purple block as the input of the discriminative network.). (Color figure online)

(texture) with a Gaussian model (Fig. 1, first). However, real world data does not always comply with a Gaussian distribution. Instead it can follow a complicated non-linear manifold. Adversarial training [8] recognizes such manifold (Fig. 1, second), and strengthens its generative power with projections (Fig. 1, third). We apply adversarial training on contextually corresponding Markovian patches (Fig. 1, fourth), so learning can focus on the mapping between different depictions of the same context, rather than the mixture of context and depictions.

Figure 2 visualizes our pipeline, which extends the patch-based synthesis algorithm of Li and Wand [15]. We first replace their patch dictionary (nearest-neighbor search) with a continuous discriminative network D (green blocks) that learns to distinguish actual feature patches (on VGG_19 layer Relu3_1, purple block) from inappropriately synthesized ones. A second comparison (pipeline below D) with a VGG_19 encoding of the same image on the higher, more abstract layer Relu5_1 can be optionally used for guidance. If we run deconvolution on the VGG networks (with the gradient from the discriminator and optionally from the guidance content), we obtain deconvolutional image synthesizer, which we call *Markovian Deconvolutional Adversarial Networks* (MDANs).

MDANs are very slow. Therefore we aim for an additional generative network G (blue blocks). It takes a VGG_19 layer Relu4_1 encoding of an image and decodes it to pixels. During training we do not change the *VGG_19* network (gray blocks), and only optimize D and G. We denote the overall architecture by *Markovian Generative Adversarial Networks* (MGANs).

3.1 Markovian Deconvolutional Adversarial Networks (MDANs)

MDANs synthesize textures with a deconvolutional process that is driven by adversarial training: a discriminative network D (green blocks in Fig. 2) is trained to distinguish between "neural patches" from the synthesis image and from the example image. We use regular sampling on layer *relu3_1* of *VGG_19* output (purple block). It outputs a classification score $s = \pm 1$ for each neural patch, indicating how "real" the patch is (with $s = 1$ being real). For each patch sampled

Fig. 3. Un-guided texture synthesis using MDANs. For each case the first image is the example texture, and the other two are the synthesis results. Image credits: [23]'s "Ivy", flickr user erwin brevis's "gell", Katsushika Hokusai's "The Great Wave off Kanagawa", Kandinsky's "Composition VII".

from the synthesized image, $1-s$ is its texture loss to minimize. The deconvolution process back-propagates this loss to pixels. Like Radford et al. [20] we use batch normalization and leaky ReLU to improve the training of D.

Formally, we denote the example texture image by $\mathbf{x}_t \in \mathbb{R}^{w_t \times h_t}$, and the synthesized image by $\mathbf{x} \in \mathbb{R}^{w \times h}$. We initialize $\mathbf{x}$ with random noise for un-guided synthesis, or an content image $\mathbf{x}_c \in \mathbb{R}^{w \times h}$ for guided synthesis. The deconvolution iteratively updates $\mathbf{x}$ so the following energy is minimized:

$$\mathbf{x} = \arg\min_{x} E_t(\Phi(\mathbf{x}), \Phi(\mathbf{x}_t)) + \alpha_1 E_c(\Phi(\mathbf{x}), \Phi(\mathbf{x}_c)) + \alpha_2 \Upsilon(\mathbf{x}) \tag{1}$$

Here E_t denotes the texture loss, in which $\Phi(\mathbf{x})$ is $\mathbf{x}$'s feature map output from layer *relu3_1* of *VGG_19*. We sample patches from $\Phi(\mathbf{x})$, and compute E_t as the Hinge loss with their labels fixed to one:

$$E_t(\Phi(\mathbf{x}), \Phi(\mathbf{x}_t)) = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - 1 \times s_i) \tag{2}$$

Here s_i denotes the classification score of i-th neural patch, and N is the total number of sampled patches in $\Phi(\mathbf{x})$. The discriminative network is trained on the fly: Its parameters are randomly initialized, and then updated after each deconvolution, so it becomes increasingly smarter as synthesis results improve.

The additional regularizer $\Upsilon(\mathbf{x})$ in Eq. 1 is a smoothness prior for pixels [17]. It is defined as $\sum_{i,j}((x_{i,j+1}-x_{i,j})^2+(x_{i+1,j}-x_{i,j})^2)$, where $x_{i,j}$ is the color value of pixel at i-th row and j-th column. This term penalizes the color difference between adjacent pixels.

Using E_t and $\Upsilon(\mathbf{x})$ can synthesize random textures (Fig. 3). By minimizing an additional content loss E_c, the network can generate an image that is contextually related to a guidance image $\mathbf{x}_c$ (Fig. 4). This content loss is the Mean Squared Error between two feature maps $\Phi(\mathbf{x})$ and $\Phi(\mathbf{x}_c)$. We set the weights with $\alpha_1 = 1$ and $\alpha_2 = 0.0001$, and minimize Eq. 1 using back-propagation with ADAM [11] (learning rate 0.02, momentum 0.5). Notice each neural patch receives its own

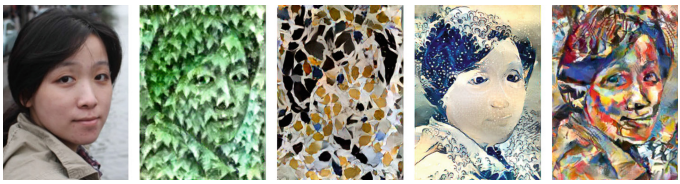

Fig. 4. Guided texture synthesis using MDANs. The reference textures are the same as in Fig. 3.

output gradient through the back-propagation of D. In order to have a coherent transition between adjacent patches, we blend their output gradient like texture optimization [13] did.

3.2 Markovian Generative Adversarial Networks (MGANs)

MDANs require many iterations and a separate run for each output image. We now train a variational auto-encoder (VAE) that decodes a feature map directly to pixels. The target examples (textured photos) are obtained from the MDANs. Our generator G (blue blocks in Fig. 2) takes the layer *relu4_1* of *VGG_19* as the input, and decodes a picture through a ordinary convolution followed by a cascade of fractional-strided convolutions (FS Conv). Although being trained with fixed size input, the generator naturally extends to arbitrary size images.

As Dosovitskiy and Brox [4] point out, it is crucially important to find a good metric for training an auto-encoder: Using the Euclidean distance between the synthesized image and the target image at the pixel level (Fig. 5, pixel VAE) yields an over-smoothed image. Comparing at the neural encoding level improves results (Fig. 5, neural VAE), and adversarial training improves the reproduction of the intended style further (Fig. 5, MGANs).

Our approach is similar to classical Generative Adversarial Networks (GANs) [8], with the key difference of not operating on full images, but neural patches from the *same* image. Doing so utilizes the contextual correspondence between the patches, and makes learning easier and more effective in contrast to learning the distribution of a object class [8] or a mapping between contextually irrelevant data [22]. In additional we also replace the Sigmoid function and the binary cross entropy criteria from [20] by a max margin criteria (Hinge loss). This avoids the vanishing gradient problem when learning D. This is more problematic in our case than in Radfort et al.'s [20] because of less diversity in our training data. Thus, the Sigmoid function can be easily saturated.

Figure 5 (MGANs) shows the results of a network that is trained to produce paintings in the style of Picasso's "Self-portrait 1907". For training, we randomly selected 75 faces photos from the CelebA data set [16], and in additional to it 25 non-celebrity photos from the public domain. We resize all photos so that the maximum dimension is 384 pixels. We augmented the training data by generating 9 copies of each photo with different rotations and scales. We regularly sample

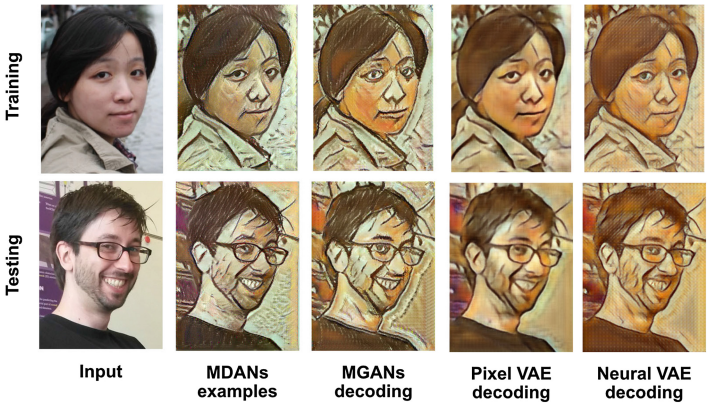

Fig. 5. Our MGANs learn a mapping from *VGG_19* encoding of the input photo to the stylized example (MDANs). The reference style texture for MDANs is Pablo Picasso's "self portrait 1907". We compare the results of MGANs to Pixel VAE and Neural VAE in with both training and testing data.

subwindows of 128-by-128 croppings from them for batch processing. In total we have 24,506 training examples, each is treated as a training image where neural patches are sampled from its *relu3_1* encoding as the input of D.

Figure 5 (top row, MGANs) shows the decoding result of our generative network for a training photo. The bottom row shows the network generalizes well to test data. Notice the MDANs image for the test image is never used in the training. Nonetheless, direct decoding with G produces very good approximation of it. The main difference between MDANs and MGANs is: MDANs preserve the content of the input photo better and MGANs produce results that are more stylized. This is because MGANs was trained with many images, hence learned

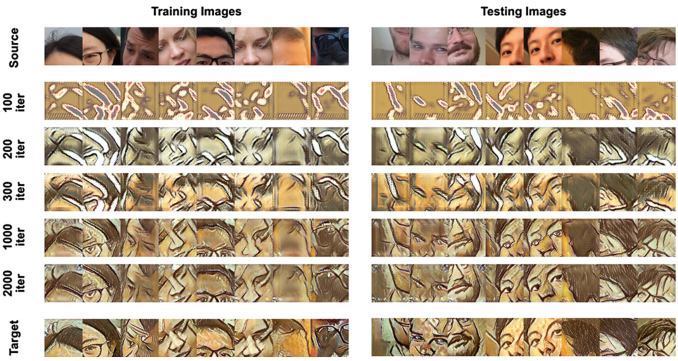

Fig. 6. Intermediate decoding results during the training of MGANs. The reference style texture for MDANs is Pablo Picasso's "self portrait 1907".

the most frequent features. Another noticeable difference is MDANs create more natural backgrounds (such as regions with flat color), due to its iterative refinement. Despite such flaws, the MGANs model produces comparable results with a speed that is 500 times faster.

Figure 6 shows some intermediate results MGANs. It is clear that the decoder gets better with more training. After 100 batches, the network is able to learn the overall color, and where the regions of strong contrast are. After 300 batches the network started to produce textures for brush strokes. After 1000 batches it learns how to paint eyes. Further training is able to remove some of the ghosting artifacts in the results. Notice the model generalizes well to testing data (right).

4 Experimental Analysis

We conduct empirical study with some hyper-parameters (layers for classification, patch size) and the complexity of the model (number of layers in the network, number of channels in each layer). While there may not be a universal optimal design for all textures, our study shed some light on how the model generally behaves. For fair comparison, the example textures in this study are fixed to 128-by-128 pixels, and synthesis output are fixed to 256-by-256 pixels.

Visualizing decoder features: We visualize the learned filters of decoder G in Fig. 7. These features are directly decoded from a one-hot input vector. Individual patches are similar to, but not very faithfully matching the example textures (due to the semi-distributed nature of the encoding). Nonetheless, the similarity seems to be strong enough for synthesizing new images.

Parameters: Here we experiment different input layers for the discriminative network. To do so we run unguided texture synthesis with discriminator D taking layer *relu2_1*, *relu3_1*, and *relu4_1* of *VGG_19* as the input. We use patch sizes of 16, 8 and 4 respectively for the three options, so they have the same receptive field of 32 image pixels (ignoring padding). The first three results in Fig. 8 shows the results: Lower layers (*relu2_1*) produce sharper appearances but at the cost of losing the structure. Higher layer (*relu4_1*) preserves coarse structure better but at the risk of being too rigid for guided scenarios. Layer *relu3_1* offers a good balance between quality and flexibility. We then show the influence of

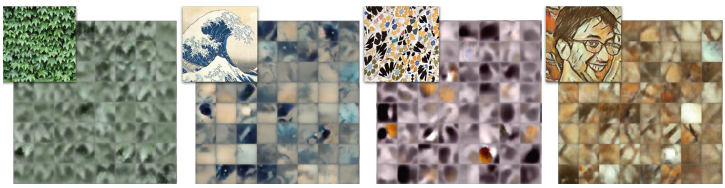

Fig. 7. Visualizing the learned features in the generative networks. Image credits: [23]'s "Ivy", flickr user erwin brevis's "gell", Katsushika Hokusai's "The Great Wave off Kanagawa", and Norman Jaklin.

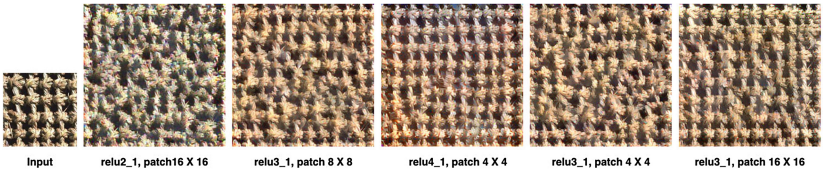

Fig. 8. Different layers and patch sizes for training the discriminative network. Input image credit: "ropenet" from the project link of [14].

patch size: We fix the input layer of D to be *relu3_1*, and compare patch size of 4 and 16 to with the default setting of 8. The last two results in Fig. 8 shows that such changes also affect the rigidity of the model: smaller patches increase the flexibility and larger patches preserve better structure.

Complexity: We now study the influence of (1) the number of layers in the networks and (2) the number of channels in each layer. We first vary D by removing the convolutional layer. Doing so reduces the depth of the network and in consequence the synthesis quality (first column, Fig. 9). Bringing this convolutional layer back produces smoother synthesis (second column, Fig. 9). However, quality does not obviously improves with more additional layers (third column, Fig. 9). Testing D with 4, 64, and 128 channels for the convolutional layer, we observe that in general less channels leads to worse results (fourth column, Fig. 9), but there is no significance difference between 64 channels and 128 channels (second column v.s. fifth column). The optimal complexity also

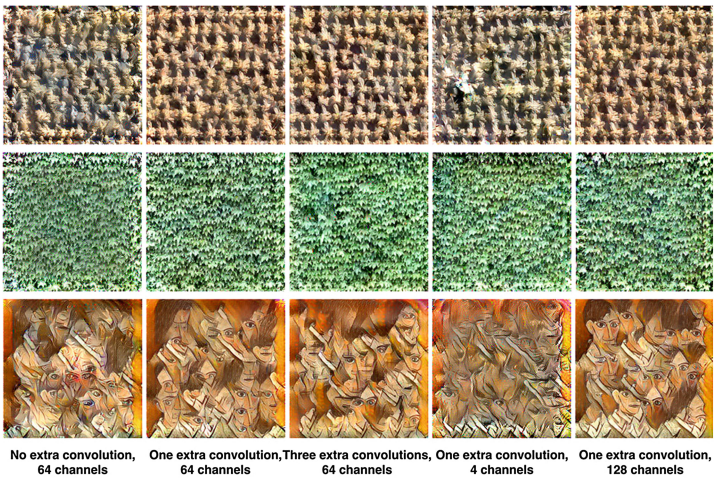

Fig. 9. Different depths for training the discriminative netowrk. The input textures are "ropenet" from the project link of [14,23]'s "Ivy", and Pablo Picasso's "self portrait 1907".

depends on the actual texture. For example, the ivy texture is rather simple, so the difference between 4 channels and 64 channels are only marginal.

Next, we fix the discriminative network and vary G. We notice some quality loss when removing the first convolutional layer from G, or reducing the number of channels for all layers, and very limited improvement from a more complex design. However the difference is not very significant. This is likely because of all these networks are driven by the same D. The reluctance of further improvement indicates there might be non-trivial information from the deconvolutional process that can not be approximated by a feed forward process.

Initialization. Usually, networks are initialized with random values. However we found D has certain generalization ability. Thus, for transferring the same texture to different images with MDANs, a previously trained network can serve as initialization. Figure 10 shows initialization with pre-trained discriminative network (that has already transferred 50 face images) produces good result with only 50 iterations. In comparison, random initialization does not produce comparable quality even after the first 500 iterations. It is useful to initialize G with an auto-encoder that directly decodes the input feature to the original input photo. Doing so essentially approximates the process of inverting *VGG_19*, and let the whole adversarial network to be trained more stably.

The role of VGG: We also validate the importance of the pre-trained *VGG_19* network. As the last two pictures in Fig. 10 show, training a discriminative network from scratch (from pixel to class label [20]) yields significantly worse results. This has also been observed by Ulyanov et al. [22]. Our explanation is that much

Fig. 10. Different initializations of the discriminative networks. The reference texture is Pablo Picasso's "self portrait 1907".

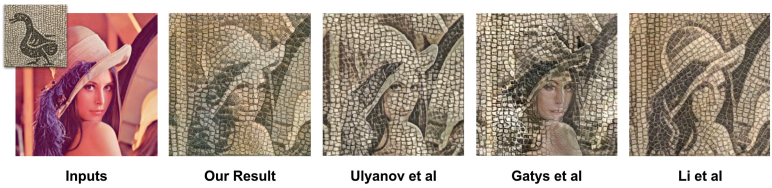

Fig. 11. Comparisons with previous methods. Results of Ulyanov et al. [22], Gatys et al. [7] and input images are from [22].

of the statistical power of VGG_19 stems from building shared feature cascades for a diverse set of images, thereby approaching human visual perception more closely than a network trained with a limited example set.

5 Results

We train each model with 100 randomly selected ImageNet images and a single example texture. We first produce 100 transferred images using MDANs, then regularly sample 128-by-128 image croppings as training data for MGANs. In total we have around 16k samples. Each epoch min-batches through all samples in random order (about 12 min). We train each texture for upto five epochs.

Figure 11 compares our results with other methods. We observe that our method has a very different character in comparison to global statistics based models [7,22]: It transfers texture more coherently, such as the hair and the eyes of Lena was consistently mapped to dark textures. In contrast, the Gaussian model [7,22] failed to keep such consistency, in particular the eyes in [22]'s result and the entire face in [7]'s result are not textured. The patch based approach [15] produces the most coherent synthesis, due to the use of non-parametric sampling. However, their method requires patch matching so is significantly slower (generate this 384-by-384 picture in 110 s). Our method and Ulyanov et al. [22] run at the same level of speed; both bring significantly improvement of speed over Gatys et al. [7] (500 times faster) and Li and Wand [15] (5000 times faster).

Figure 12 further discuss the difference between the Gaussian based method [22] and our method[3]. In general [22] produces more faithful color distributions in respect to the style image. It also texture the background better (the starry night), whereas our method suffers due to the suppression from the VGG network. On the other hand, our method produces more coherent texture transfer for salient foreground objects, such as the facade in both examples. In comparison [22] produces either too much or too little textures in such complex regions.

MGANs can decode noise input into texture (Fig. 13): Perlin noise[4] images are forwarded through *VGG_19* to generate feature maps for the decoder. To our surprise, the model that was trained with ImageNet images is able to decode such features maps to plausible textures. This shows the generalization ability of our model. Figure 13 shows our video decoding result. As a feed-forward process our method is not only faster but also relatively more temporally coherent than per-frame based deconvolutional methods (Fig. 14).

[3] Since Ulyanov et al. [22] and Johnson et al. [10] are very similar approaches, here we only compare to one of them [22]. The main differences of [10] are: (1) using a residual architecture instead of concatenating the outputs from different layers; (2) no additional noise in the decoding process.

[4] We need to use "brown" noise with spectrum decaying to the higher frequencies because flat "white" noise creates an almost flat response in the encoding of the VGG network. Somer lower-frequency structure is required to trigger the feature detectors in the discriminative network.

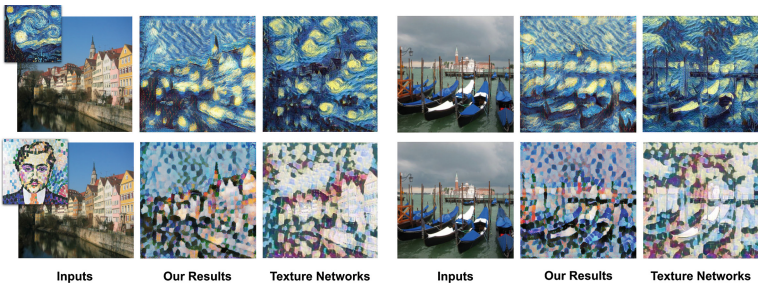

Fig. 12. More comparisons with Texture Networks [22]. Results of [22] and input images are from [22].

Fig. 13. Generate random textures by decoding from Brown noise.

Last but not the least, we provide details for the time/memory usage of our method. The time measurement is based on a standard benchmark framework [2]: Our speed is at the same level as the concurrent work by Ulyanov et al. [22], who also use a feed-forward approach, perform significantly faster than previous deconvolution based approaches [7,15]. More precisely, both our method and Ulyanov et al. [22] are able to decode 512-by-512 images at 25 Hz, while [22] leads the race by a very small margin. The time cost of both methods scale linearly with the number of pixels in the image. For example, our method cost 10 ms for a 256-by-256 image, 40 ms for a 512-by-512 image, and 160 ms for a 1024-by-1024 image. Both methods show a very significant improvement in speed over previous deconvolutional methods such as Gatys et al. [7] and Li and Wand [15]: about 500 times faster than Gatys et al. [7], and 5000 times faster than Li and Wand [15]. In the meantime our method is also faster than most traditional pixel based texture synthesizers (which rely on expensive nearest-neighbor searching). A possible exceptions would be a GPU implementation of "Patch Match" [1], which could run at comparable speed. However, it provides the quality benefits (better blending, invariance) of a deep-neural-network method (as established in previous work [7,15]). Memory-wise, our generative model takes 70 Mb memory for its parameters(including the *VGG* network till layer Relu4_1). At runtime, the required memory to decode a image linearly depends on the image's size: for a 256-by-256 picture it takes about 600 Mb, and for a 512-by-512 picture it requires about 2.5 Gb memory. Notice memory usage can be reduced by subdividing the input photo into blocks and run the decoding in a scanline fashion. However, we do not further explore the optimization of memory usage in this paper.

Fig. 14. Decoding a 1080-by-810 video. We achieved the speed of 8 Hz. Input video is credited to flickr user macro antonio torres.

6 Limitation

Our current method works less well with non-texture data. For example, it failed to transfer facial features between two difference face photos. This is because facial features can not be treated as textures, and need semantic understanding (such as expression, pose, gender etc.). A possible solution is to couple our model with the learning of object class [20] so the local statistics is better conditioned. For synthesizing photo-realistic textures, Li and Wand [15] often produces better results due to its non-parametric sampling that prohibits data distortion. However, the rigidity of their model restricts its application domain. Our method works better with deformable textures, and runs significantly faster.

Our model has a very different character compared to Gaussian based models [7,22]. By capturing a global feature distribution, these other methods are able to better preserve the global "look and feels" of the example texture. In contrast, our model may deviate from the example's global color distribution.

Since our model learns the mapping between different depictions of the same content, it requires features highly invariant features. For this reason we use the pre-trained *VGG_19* network. This makes our method weaker in dealing with highly stationary backgrounds (sky, out of focus region etc.) due to their weak activation from *VGG_19*. We observed that in general statistics based methods [7,22] generate better textures for areas that has weak content, and our method works better for areas that consist of recognizable features. We believe it is valuable future work to combine the strength of both methods.

Finally, we discuss the noticeable difference between the results of MDANs and MGANs. The output of MGANs is often more consistent with the example texture, this shows MGANs' strength of learning from big data. MGANs has weakness in flat regions due to the lack of iterative optimization. More sophisticated architectures such as the recurrent neural networks can bring in state information that may improve the result.

7 Conclusion

The key insight of this paper is that adversarial generative networks can be applied in a Markovian setting to learn the mapping between different depictions of the same content. We develop a fully generative model that is trained from a single texture example and randomly selected images from ImageNet. Once

trained, our model can decode brown noise to realistic texture, or photos into artworks. We show our model has certain advantages over the statistics based methods [7,22] in preserving coherent texture for complex image content.

Our method is only one step in the direction of learning generative models for images. For future work one can study the broader framework in a big-data scenario to learn not only Markovian models but also include coarse-scale structure models. This additional invariance to image layout could open up ways to also use more training data for the Markovian model, thus permitting more complex decoders with stronger generalization capability over larger classes.

Acknowledgments. This work has been partially supported by the Intel Visual Computing Institute and the Center for Computational Science Mainz. We like to thank Bertil Schmidt and Christian Hundt for providing additional computational resources; and Dmitry Ulyanov, Norman Jaklin for sharing results and input images.

References

1. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: PatchMatch: a randomized correspondence algorithm for structural image editing. In: SIGGRAH, pp. 24:1–24:11 (2009)
2. Chintala, S.: Easy benchmarking of all publicly accessible implementations of convnets (2015). https://github.com/soumith/convnet-benchmarks
3. Denton, E.L., Fergus, R., Szlam, A., Chintala, S.: Deep generative image models using a Laplacian pyramid of adversarial networks. In: NIPS (2015)
4. Dosovitskiy, A., Brox, T.: Generating images with perceptual similarity metrics based on deep networks. CoRR abs/1602.02644 (2016). http://arxiv.org/abs/1602.02644
5. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: SIGGRAPH, pp. 341–346 (2001)
6. Gatys, L.A., Ecker, A.S., Bethge, M.: Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. In: NIPS, May 2015. http://arxiv.org/abs/1505.07376
7. Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style (2015). arXiv preprint http://arxiv.org/abs/1508.06576
8. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NIPS, pp. 2672–2680 (2014)
9. Hertzmann, A., Jacobs, C.E., Oliver, N., Curless, B., Salesin, D.H.: Image analogies. In: SIGGRAPH, pp. 327–340 (2001)
10. Johnson, J., Alahi, A., Li, F.F.: Perceptual losses for real-time style transfer and super-resolution. CoRR abs/1603.08155, March 2016. http://arxiv.org/abs/1603.08155v1
11. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980
12. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. CoRR abs/1312.6114 (2013). http://arxiv.org/abs/1312.6114
13. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. SIGGRAPH **24**(3), 795–802 (2005)

14. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: image and video synthesis using graph cuts. ACM Trans. Graph. **22**(3), 277–286 (2003)
15. Li, C., Wand, M.: Combining Markov random fields and convolutional neural networks for image synthesis. CoRR abs/1601.04589 (2016). http://arxiv.org/abs/1601.04589
16. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: ICCV (2015)
17. Mahendran, A., Vedaldi, A.: Understanding deep image representations by inverting them. In: CVPR (2015)
18. Mordvintsev, A., Olah, C., Tyka, M.: Inceptionism: going deeper into neural networks (2015). http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html
19. Nguyen, A.M., Yosinski, J., Clune, J.: Multifaceted feature visualization: uncovering the different types of features learned by each neuron in deep neural networks. CoRR abs/1602.03616 (2016). http://arxiv.org/abs/1602.03616
20. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. CoRR abs/1511.06434 (2015). http://arxiv.org/abs/1511.06434
21. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR (2014). http://arxiv.org/abs/1409.1556
22. Ulyanov, D., Lebedev, V., Vedaldi, A., Lempitsky, V.: Texture networks: feed-forward synthesis of textures and stylized images. CoRR abs/1603.03417, March 2016. http://arxiv.org/abs/1603.03417v1
23. Xie, J., Lu, Y., Zhu, S.C., Wu, Y.N.: A theory of generative convnet. CoRR arXiv:1602.03264 (2016). http://arxiv.org/abs/1602.03264
24. Yosinski, J., Clune, J., Nguyen, A.M., Fuchs, T., Lipson, H.: Understanding neural networks through deep visualization. CoRR abs/1506.06579 (2015). http://arxiv.org/abs/1506.06579
25. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014, Part I. LNCS, vol. 8689, pp. 818–833. Springer, Heidelberg (2014)