Texture attribute synthesis and transfer using feed-forward CNNs

Thomas Irmer
Ruhr University Bochum
thomas.irmer@rub.de

Tobias Glasmachers
Ruhr University Bochum
tobias.glasmachers@ini.rub.de

Subhransu Maji
University of Massachusetts
smaji@cs.umass.edu

Abstract

We present a novel technique for texture synthesis and style transfer based on convolutional neural networks (CNNs). Our method learns feed-forward image generators that correspond to specification of styles and textures in terms of high-level describable attributes such as 'striped', 'dotted', or 'veined'. Two key conceptual advantages over template-based approaches are that attributes can be analyzed and activated individually, while a template image necessarily represents a simultaneous specification of many attributes, and that attributes can combine aspects of many texture templates allowing flexibility in the generation process. Once the attribute-wise networks are trained, applications to texture synthesis and style transfer are fast, allowing for real-time video processing.

1. Introduction

Texture synthesis techniques can be roughly divided into two fundamentally different approaches: (1) *procedural texture synthesis*, which generates textures using a set of predefined parameters and some randomness, and (2) *template-based texture synthesis*, which takes a template image as input and produces similar-looking textures.

Procedural methods are useful when a reference texture image (or template) is not available [7, 26]. These are based on hand-crafted mathematical functions or algorithms that use randomness to generate a certain type of texture such as wood, marble, or clouds. Such models also have category-specific parameters which can be adjusted to generate a wide range of textures. However, these parameters rarely have an intuitive meaning, which makes them hard to adjust. In contrast, template-based methods can generate very realistic looking textures [8, 14, 23, 29]. Nevertheless the availability of such a template is mandatory.

In this paper we aim for a third approach which overcomes these shortcomings, in the following sense. Humans often think about textures in terms of natural language, e.g., in terms of high-level attributes. Say, a designer wants to create a *veined texture*, but without any additional constraint given by a concrete template image. Current procedural models based on simple mathematical models are not capable of generating textures based on high-level attributes. We propose a method, which synthesizes a texture, or transfers it to an image, given just a texture attribute like *veined*. It uses a convolutional neural network (CNN) architecture in a *generative manner* such that the resulting images are classified as the desired attributes. The classifier itself is based on models trained on publicly-available texture datasets, hence the proposed models can be trained without direct human supervision. Once trained, these models are fast since they process the image in a feed-forward manner allowing real-time processing of high-resolution images or video.

Convolutional neural networks are powerful image processors. While they are primarily known for their capabilities for various image recognition tasks [17, 15, 25], some recent works have shown that CNNs are effective as *texture descriptors* [10, 21, 4] and *texture generators* [6, 28, 16]. In these approaches, texture generation is template based: for each style or texture template, a new network is trained. The proposed approach generalizes this to the setting where instead of a reference style image, we train networks that generate images corresponding to various texture attributes such as *veined*.

Aside from computational advantages, this approach results in networks representing and generating textures with interpretable attributes. We believe that this property is key for the practical utility of the method in a creative environment. Moreover, these attributes combine aspects of style common across many images, allowing greater flexibility in the generated style. For example, in order to make an image more 'cracked', the orientation and scale of the cracks must match the structures within the image. A single cracked image template may only have cracks at a single scale or orientation making the style transfer difficult. On the other hand, if the goal is to simply generate a cracked image, the generator can achieve the desired effect by flexibly choosing parts from different cracked images. Our results show that such parts are essentially embodied in a classifier trained to recognize a set of cracked images.

2. Convolutional Neural Networks

A feed-forward neural network is a parameterized, non-linear map [1]. The information processing is organized into layers. Data, for example raw RGB image pixel values, is propagated from the input layer through so-called hidden layers to the output layer. Layers are collections of artificial neurons, each of which is connected to neurons from previous layers. Each neuron receives a linear combination of the activations of these neurons it is connected to, parameterized by connection weights, which are the trainable parameters of the model. The neuron computes its activation by applying a non-linear transfer function to its input. The number n of layers, the number of neurons M_l per layer (layer size), the connectivity pattern between layers, and the type of transfer function are hyperparameters of the network. Once the hyperparameters are set, the weights are trained by minimizing the (regularized) empirical risk functional defined over a training set of labeled images, usually with a variant of stochastic gradient descent [2].

In convolutional neural networks (CNNs), layers have a spatial layout, just like input images [18, 9]. The flow of information is furthermore restricted to local interactions, e.g., with convolution and pooling filters applied uniformly across a whole layer, resembling visual fields found in early visual cortex.

The neural activation patterns arising in higher layers are intermediate representations of the input. They are sometimes referred to as the network's *feature maps*. Following the notation of Gatys et al., each layer l with N_l filters generates N_l feature maps of size M_l when vectorized. All feature maps can then be written as a matrix $F_l \in \mathbb{R}^{N_l \times M_l}$.

CNNs are known to be powerful object detectors [5, 17], therefore a standard hypothesis is that the feature maps of layers close to the output encode valuable high-level information. In recent years, huge deep networks were trained on large data sets for various detection and classification tasks. The hidden layers of these networks are frequently applied in other contexts as general-purpose image descriptors. These descriptors were found to contain information about many aspects of an image, including its texture [10].

3. Related Work

A remarkable discovery about the descriptive power of CNNs was made by Gatys et al. [10]. Their idea follows work by Portilla and Simoncelli [24], who extracted differently sized features homogeneously from a template image and formed spatial summary statistics from those features. In contrast to the original approach, Gatys et al. used a pre-trained VGG network [27] instead of hand-crafted features–namely the Gram matrices of the VGG's feature maps.

A natural goal for a texture descriptor is spatial invariance. This is achieved by considering the Gram matrices $G^l_{ij} \in \mathbb{R}^{N_l \times N_l}$, which are defined by

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}, \tag{1}$$

where G^l_{ij} is the inner product between feature maps i and j in layer l. A set of Gram matrices extracted from some hand-chosen layers forms the texture descriptor. This descriptor has the desirable property of being spatially invariant because the inner products forming the entries of the Gram matrices blend information form all locations in the feature maps together.

To generate new textures which look alike a template image $\vec{x}$, Gatys et al. match the descriptor of that template image by minimizing the following objective $\mathcal{L}_{\text{texture}}$, starting from a noise image $\hat{\vec{x}}$:

$$\mathcal{L}_{\text{texture}}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l \tag{2}$$

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - \hat{G}^l_{ij})^2, \tag{3}$$

where $w_l \in \{0, 1\}$ are weight factors used to select the descriptive layers.

In a follow-up work Gatys et al. [11] additionally added another loss term to their method, which aims to preserve the *content* of the input image. This makes it possible to not only generate new texture images starting from noise but also *transfer textures* to real images. As the loss term for the content they simply chose the squared-loss on a single layer on the feature maps directly:

$$\mathcal{L}_{\text{content}}(\vec{x}, \hat{\vec{x}}) = \frac{1}{2} \sum_{i,j} (F_{ij} - \hat{F}_{i,j})^2. \tag{4}$$

By optimizing a weighted combination

$$\mathcal{L}_{\text{total}}(\vec{x}, \hat{\vec{x}}) = \alpha \mathcal{L}_{\text{content}}(\vec{x}, \hat{\vec{x}}) + \beta \mathcal{L}_{\text{texture}}(\vec{x}, \hat{\vec{x}}) \tag{5}$$

of the *texture* and *content* loss terms they yield appealing results with transferring textures to any given input image.

Another work which exploits the descriptive power of CNNs has been published by Lin et al. [21]. They explore to what extend those Gram matrices–which they refer to as *bilinear features*–are suitable for classification. They train linear classifiers for texture attributes on top of the extracted bilinear features for a set of layers L and obtain attribute prediction probabilities C_l on the selected layers $l \in L$. Similar to Gatys et al., they then generate new texture images by optimizing an input image $\hat{\vec{x}}$ until it converges to the same texture attribute prediction as a given template $\vec{x}$:

$$\mathcal{L}_{\text{attribute}}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} c(C_l, \hat{C}_l) \tag{6}$$

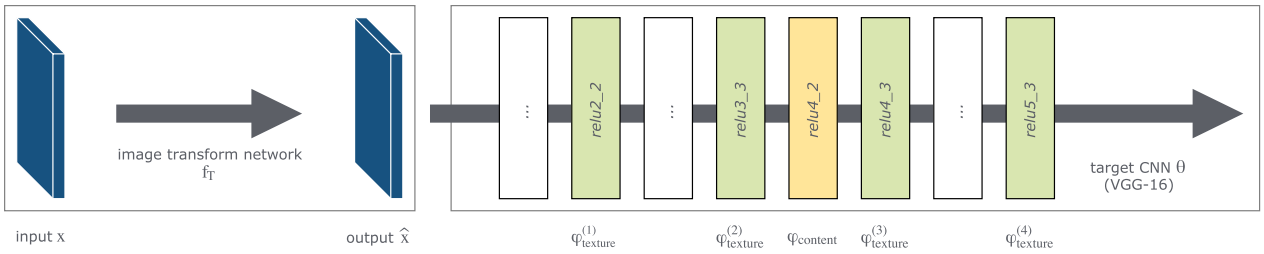

Figure 1. Overview of the proposed architecture. The *image transform network* f_T takes an image $\vec{x}$ of arbitrary size and aspect ratio and outputs an image $\hat{\vec{x}}$ with the same dimensions. The attached *target network* θ serves as the descriptor network where the labels are extracted from. We use the VGG-16 network [27], trained to classify the texture images from the DTD database [3] into it's 47 categories.

where c is a cost function such as the negative log-likelihood of the label C.

All of the above mentioned works reveal a lot about the descriptive power of CNNs and yield amazing results in texture synthesis and texture transfer to other images. However, both methods are slow since they use iterative optimization processes.

Two recent works by Ulyanov et al. [28] and Johnson et al. [16] moved this slow iterative optimization process to the training phase of a neural network. They showed that the optimization which has previously been done by methods such as gradient descent can also be done by a trained CNN instantly. In that case one can use the loss in eq. (5) as objective for a CNN. This requires training a network per texture template, which usually takes longer than the optimization-based approach but can be used in real-time after training. Although their results cannot match the quality of the previous methods yet, they show a major boost in computational performance and memory efficiency. Nevertheless their methods are also template based and have to train a single network for each texture template image, which makes it cumbersome to incorporate new textures.

4. Texture Attribute Generator

Here we describe the details of our approach, which enables us to transfer selected texture attributes like *veined* to any input image, while the content of the image is still well preserved, and synthesize new texture images of the selected attribute from nothing but noise.

Our method draws on various aspects of previous work on texture synthesis and texture transfer. Essential are the texture descriptors proposed by Gatys et al. [10], the classification scores introduced by Lin et al. [20], and the feed-forward architectures as described by Johnson et al. [16]. We aim to generalize template-based methods and to drastically speed up the generations process.

4.1. Method Overview

The overall architecture is shown in figure 1. To enforce our method to be independent of image dimensions we use an image transformation network f_T, which compresses the input to a shallower representation and scales it up to it's original dimensions. The architecture is derived from [16] and a detailed network architecture is given in table 1. During training the image dimensions are as stated in table 1, but at test time any image size can be used. For the descriptor CNN we use a pre-trained network θ. For a fixed texture attribute t, we extract a set of target features $\phi^{(1)}_{\text{texture}}(t), \phi^{(2)}_{\text{texture}}(t), \ldots, \phi^{(n)}_{\text{texture}}(t)$ from θ's n layers. The network f_T maps an input image $\vec{x}$ to the output $\hat{\vec{x}} = f_T(\vec{x})$. Then $\hat{\vec{x}}$ is fed through θ and produces a set of feature responses $\phi^{(1)}_{\text{texture}}(\hat{\vec{x}}), \phi^{(2)}_{\text{texture}}(\hat{\vec{x}}), \ldots, \phi^{(n)}_{\text{texture}}(\hat{\vec{x}})$ and a content response $\phi_{\text{cont}}(\hat{\vec{x}})$.

We then minimize the loss function $\mathcal{L}_{\text{total}}$ which represents the loss on the texture features and the content features combined with an image prior.

$$\begin{aligned}\mathcal{L}_{\text{total}}(\vec{x}, \hat{\vec{x}}) = &\sum_{i=1}^{n} \lambda^{(i)}_{\text{texture}} \cdot \mathcal{L}^{(i)}_{\text{texture}}(t, \hat{\vec{x}}) \\ &+ \lambda_{\text{content}} \cdot \mathcal{L}_{\text{content}}(\vec{x}, \hat{\vec{x}}) \\ &+ \lambda_{\text{prior}} \cdot \mathcal{L}_{\text{prior}}(\hat{\vec{x}})\end{aligned} \tag{7}$$

$\mathcal{L}_{\text{total}}$ consists of three weighted parts: the texture feature loss $\mathcal{L}_{\text{texture}}$, the content feature loss $\mathcal{L}_{\text{content}}$, and a prior for natural images $\mathcal{L}_{\text{prior}}$, for which we chose the *total variation norm* (TV norm):

$$\mathcal{L}^{(l)}_{\text{texture}}(t, \hat{\vec{x}}) = \left(\phi^{(l)}_{\text{texture}}(t) - \phi^{(l)}_{\text{texture}}(\hat{\vec{x}})\right)^2 \tag{8}$$

$$\mathcal{L}_{\text{content}}(\vec{x}, \hat{\vec{x}}) = \left(\phi_{\text{content}}(\vec{x}) - \phi_{\text{content}}(\hat{\vec{x}})\right)^2 \tag{9}$$

$$\mathcal{L}_{\text{prior}}(\hat{\vec{x}}) = \sum_{i,j} \Big(\left(\hat{\vec{x}}_{i,j+1} - \hat{\vec{x}}_{ij}\right)^2 \tag{10}$$

$$+ \left(\hat{\vec{x}}_{i+1,j} - \hat{\vec{x}}_{ij}\right)^2 \Big)^{\beta/2} \tag{11}$$

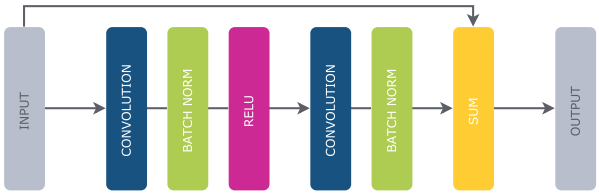

Figure 2. Modified Residual Block by [12]

Descriptor CNN and Training Targets The network θ is the VGG-16 network [27] trained to classify texture attributes using the bilinear features from a subset of layers. So the texture targets $\phi^{(l)}_{\text{texture}}(t)$ on layer l for a given texture attribute t are the classification scores of the linear classifier of that network for this attribute on the bilinear features. The content targets $\phi_{\text{content}}(\vec{x})$ are the feature maps of one of θ's layers without any modification.

The size of $\phi^{(l)}_{\text{texture}}(t)$ is the same for any layer of θ and depends on the database θ has been trained on. With n different categories $\phi^{(l)}_{\text{texture}}(t)$ would be a vector with n elements where each element is 1 if it corresponds to t and 0 otherwise. The content targets $\phi_{\text{content}}(\vec{x})$ are the feature maps taken from layer *relu4_2* of the VGG-16 which has $\frac{\textit{image height}}{8} \times \frac{\textit{image width}}{8} \times 512$ elements.

4.2. Network Architecture

The network architecture of the image transform network is shown in table 1. Each convolutional layer (conv) is followed by a batch normalization layer (bnorm) and a leaky ReLU (relu) with $a = 0.01$. The last layer is followed by a scaled sigmoid non-linearity instead of a ReLU. The scaled sigmoid version ensures that the output is in the range $[0, 255]$, which is important because θ has been trained on pixel intensities in that range. The residual blocks (res-conv), which have been introduced by [13], are used in a modified manner as described by [12] (see figure 2). The idea behind using residual blocks is, that the network has to achieve two competitive goals: preserving the content of the image and applying the texture. The skip connections provide an easy way for the network to forward all needed information about the content to the output. The convolutional layer in the residual blocks can then take care of the texture transfer.

Layer name	Filter size	Feature map size
Input		$224 \times 224 \times 3$
conv_1	$9 \times 9 \times 3 \times 32$	$224 \times 224 \times 32$
conv_2	$3 \times 3 \times 32 \times 64$	$112 \times 112 \times 64$
conv_3	$3 \times 3 \times 64 \times 128$	$56 \times 56 \times 128$
res-conv_4	$3 \times 3 \times 128 \times 128$	$56 \times 56 \times 128$
res-conv_5	$3 \times 3 \times 128 \times 128$	$56 \times 56 \times 128$
res-conv_6	$3 \times 3 \times 128 \times 128$	$56 \times 56 \times 128$
res-conv_7	$3 \times 3 \times 128 \times 128$	$56 \times 56 \times 128$
res-conv_8	$3 \times 3 \times 128 \times 128$	$56 \times 56 \times 128$
deconv_9	$4 \times 4 \times 128 \times 64$	$112 \times 112 \times 64$
deconv_{10}	$4 \times 4 \times 64 \times 32$	$224 \times 224 \times 32$
conv_{11}	$9 \times 9 \times 32 \times 3$	$224 \times 224 \times 3$
Output		$224 \times 224 \times 3$

Table 1. Detailed network architecture for texture transfer and generator network

The first three convolution layers increase the number of channels and down-sample the image. The number of channels has a direct impact on the learning capacity of the network. The more channels and thus filters, the more information it can encode. There is a trade-off between the computational cost of using many layers and the quality of the output. The down-sampling might seem superfluous at first, but it reduces the number of activations and gives us the opportunity to create a deeper network than without down-sampling. If we reduce the input size by a factor of two we can use four times more filters at the same computational effort or we could use four times more layers with the same number of filters [16].

Another important property of the down-sampling operations is the increase of the receptive field of later layers. Whenever we down-sample a feature map, the next layer's receptive field size represents a twice as big area of the input as without down-sampling. This, however, comes with the cost of up-sampling the feature map to it's original size, which tends to introduce artifacts.

4.3. Training

For each texture attribute (category) in a dataset we train a separate network. Before we start the training we need to set the target values for the texture attributes $\phi^{(l)}_{\text{texture}}(t)$. These are the classification scores taken from the descriptor network θ, pre-trained on texture classification. The descriptor network θ is the VGG-16 network, which has been fine-tuned for texture attribute classification. The attribute classification has been done by extracting the bilinear features from a set of layers, normalizing them by the signed square root and l_2, using a convolutional layer to form the classification vector and finally applying a soft-max layer. The loss, used to train this network on these features, was the *negative log-likelihood* on the attribute labels. The texture features $\phi^{(l)}_{\text{texture}}(t)$ are extracted from the subset of layers: *relu2_2, relu3_3, relu4_3, relu5_3*. The specific choice for the feature layers is not as important as choosing layers, which represent feature maps of different spatial size. This ensures that texture features of all sizes can be transferred [10]. After setting the targets for the given attribute t we can start the training procedure.

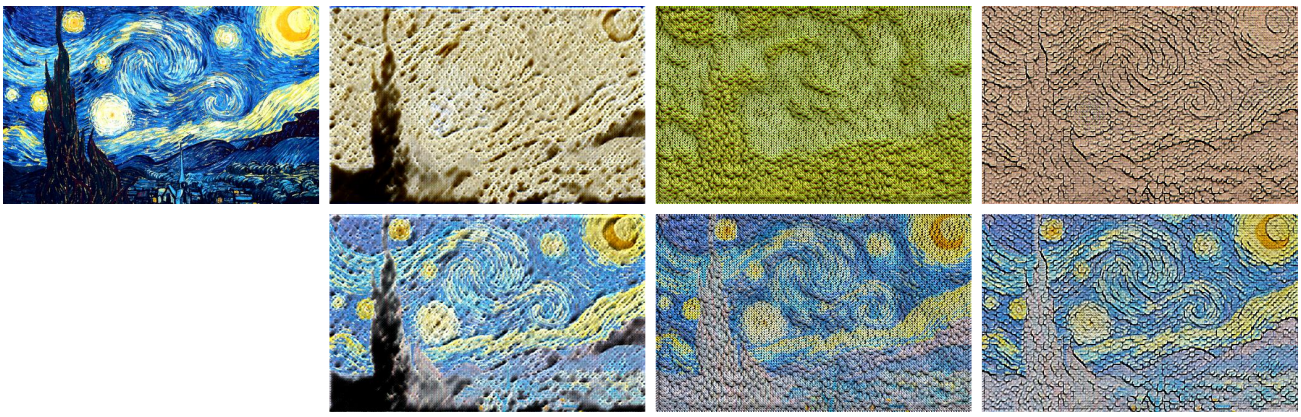

Figure 3. top left: network input *The Starry Night* by Vincent van Gogh, 1889, top row: network outputs for attributes *porous, knitted, cracked*, bottom row: network outputs with color from input image and luminance from network outputs

For the training we used the MS COCO dataset[1] [19], using SGD with momentum and a mini-batch size of 20 images per iteration. The learning rate was set to 10^{-12} and the momentum term was 0.9. We did not use weight decay, dropout or any other form of regularization. The objective weights λ were set as follows: $\lambda_{\text{content}} = 1$, $\lambda_{\text{prior}} = 10^{-1}$, $\lambda^{(l)}_{\text{texture}} = 0$ for the first 1,000 iterations and $\lambda^{(\text{relu2\_2})}_{\text{texture}} = 1 \cdot 10^{10}, \lambda^{(\text{relu3\_3})}_{\text{texture}} = 4 \cdot 10^{10}, \lambda^{(\text{relu4\_3})}_{\text{texture}} = 8 \cdot 10^{10}, \lambda^{(\text{relu4\_3})}_{\text{texture}} = 64 \cdot 10^{10}$ for the next 2,000 iterations. This staged training makes the network learn the optimal reconstruction first and then apply the texture. We found that this gives better results than training on all objectives from the beginning. The huge magnitude difference between the content and texture weights is caused by the range of the target values. The classification scores for the texture attributes are between 0 and 1, whereas the content feature maps are of size $28 \times 28 \times 512$ with a range of (0,255) for each activation, which results in huge values when applying the loss function. The values for the texture weights (1, 4, 8, 64) are hand-picked and have to be optimized in the future but yielded the best results.

Before running the image transform network f_T in each iteration we have to set the content target features ϕ_{content} by running θ forward and saving the feature maps for the content target layer, which we decided to be *relu4_2*. This is a good trade-off between reconstruction quality and feature map size. Earlier feature maps yield better reconstruction but have more activations which makes it computational and memory-wise more expensive to use earlier layers. In our case we can live with the slightly worse reconstruction because we do not want to reconstruct the content perfectly anyway.

The chain of processing steps for each mini-batch of RGB images is as follows:

- We rescale all images to a size of 224×224 pixels. The RGB values of these images are in the range $[0, 255]$.
- We subtract the channel-wise RGB mean, provided by the VGG-16's training data, and then compute the content target features.
- We feed these images through f_T to obtain the reconstruction $\hat{\vec{x}}$.
- These are propagated through θ to obtain the content features ϕ_{content} and the texture scores $\phi^{(l)}_{\text{texture}}$.
- On those we compute the losses $\mathcal{L}_{\text{content}}$ and $\mathcal{L}_{\text{texture}}$, which we back-propagate through θ, each weighted by its λ-term.
- After this back-propagation we get the loss of the content features and the texture scores with respect to $\hat{\vec{x}}$ to which we add the natural image prior loss $\mathcal{L}_{\text{prior}}$. This combined loss is then used to back-propagate through f_T.

At this point we have all gradients needed to perform the next SGD step.

The training dataset of MS COCO has 82,783 images. That means that with a mini-batch size of 20 images we would need 4,140 iterations to run a single epoch over the whole dataset. Because the training yields promising results after just a few iterations we do not use SGD in terms of epochs over the whole dataset but only look at individual iterations. That means that the network has not even seen the whole data set after just 3,000 iterations.

5. Empirical Evaluation

For training our networks we chose the following parameters:

The descriptive target network θ, was chosen to be the VGG-16 network [27], pre-trained on texture classification using the bilinear features by Lin et al. [20]. The texture dataset, which has been used to train θ, was the DTD (De-

[1] MS COCO Dataset - http://mscoco.org

Figure 4. top left: network input *Self-Portrait* by Pablo Picasso, 1907, top row: network outputs for attributes *braided, fibrous, interlaced, veined*, bottom row: network outputs with color from input image and luminance from network outputs

scribable Texture Dataset)[2], which consists of 5,640 images, evenly distributed over 47 categories, which are the following:

> *banded, blotchy, braided, bubbly, bumpy, chequered, cobwebbed, cracked, crosshatched, crystalline, dotted, fibrous, flecked, frilly, gauzy, grid, grooved, honeycombed, interlaced, knitted, lacelike, lined, marbled, matted, meshed, paisley, perforated, pitted, pleated, polka-dotted, porous, potholed, scaly, smeared, spiralled, sprinkled, stained, stratified, striped, studded, swirly, veined, waffled, woven, wrinkled, zigzagged* [3]

Since we aim to transfer textures to natural images and synthesize new textures we will mostly rely on qualitative inspection of the results since it there is no measure which tells us something about the naturalness as humans perceive images.

5.1. Texture Transfer

Figures 3 and 4 show examples of textures transfered to van Gogh's *Starry Night* and Picassos's *self-portrait*. As we can see in the first row of each figure, the color information of the input image gets lost, which is why we show a post-processed version of the output aside all generated results. This post-processing step consists of using the luminance of the output image and the color and saturation of the input image, which preserves the original color.

As we see in figures 3 and 4, this method can produce results, that contain information about the content of the input image as well as the attribute of the given texture. If we look at the first output - *the porous starry night* - we see that the input's color information gets lost but we still see a strong similarity with the original input. Most people would agree that these *starry nights* look *porous*, *knitted* and *cracked*. Also the self-portrait by Picasso is clearly recognizable after the texture transfer. Although those results are appealing not all texture attributes from the DTD dataset yield similarly good results.

Figure 5 shows examples for attributes that produce interesting images but do not look very natural. Although we can clearly see, that features of the given attribute appear in the output and they are oriented along the input features–like it's edges–they do not look very realistic. Another observation is, that in all texture attributes only a few actual texture features appear. They come in different scales but most parts of the output image consist of similar elements.

5.2. Texture Synthesis

Besides texture transfer we are also able to synthesize textures from just noise. The synthesis can be achieved by simply using noise as input instead of a natural image. Many choices for the noise type work but we found that Perlin noise [22] yields the best results. Figure 6 shows some

[2]DTD - http://www.robots.ox.ac.uk/~vgg/data/dtd/

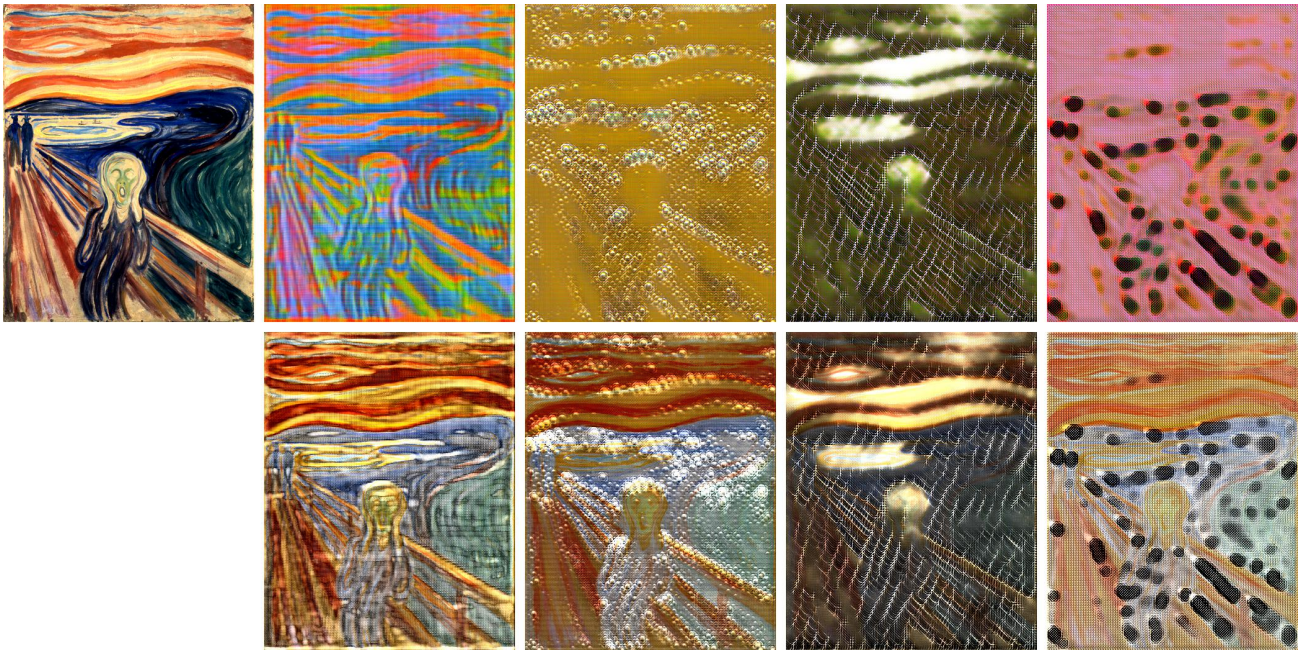

Figure 5. top left: network input *Der Schrei* by Edvard Munch, 1910, top row: network outputs for attributes *banded, bubbly, cobwebbed, dotted*, bottom row: network outputs with color from input image and luminance from network outputs

of the categories from DTD synthesized using Perlin color noise images of 192×192 pixels.

We can see that not all categories look natural but those who do look very promising indeed. These synthesized textures can be generated at any size only depending on the size of the noise input but all texture features (bubbles in *bubbly*, honeycombs in *honeycombed*, veins in *veined*, ...) will stay at the same scale. This is because the filters, learned by our networks, have a fixed size. So if the bubbles in the *bubbly* texture are about 32×32 pixels, they will be that size no matter how big the input noise is.

Training Iterations and Texture Scale One will easily realize that the effect of the texture transferred to an input depends on how long we train the generative network. The texture attribute becomes stronger and the content less recognizable the longer we train a network. Figure 7 shows how strong the texture appears after different iterations of the training. However the training behaves differently for other attributes–some might have a very strong effect in early iterations whereas others need much longer training. Another observation is the effect of the size of the input image. The texture features always appear at the same scale because the filters, learned by the generative network, have fixed size. Therefore the size of the input determines the textured look of the output.

6. Conclusion

We have presented a novel method for style transfer and texture generation. A key property of the method is that textures can be specified in terms of interpretable, high-level attributes, such as *striped*, *marbled*, or *veined*. This is achieved by training independent generative convolutional neural networks for each attribute.

In contrast to procedural methods, our texture generator enjoys the flexibility of a data-driven, neural networks based approach. Hence, an arbitrary attribute can be specified by simply providing a training set, and without any expert knowledge of how to generate the texture's structure algorithmically. In contrast to methods based on a single template image, the potentially large training set allows to focus on a single attribute, shared by all training instances, while a single image necessarily represents characteristics of multiple attributes.

In our system, texture generation and style transfer are feed forward processes. This makes the method capable of real-time video processing, which can be a decisive advantage over optimization-based approaches. We believe that already the demonstration that attribute-based style transfer is possible in a purely feed-forward manner is a significant contribution. The quality of the results is already quite pleasing. However, since we do not yet reach the perfection of far more compute-intensive optimization-based approaches, we still see room for future improvements.

Figure 6. Synthesized textures using 192×192 pixel RGB Perlin noise images as input. The texture attributes that have been used to generate those textures are (*top left to bottom right*): *braided, bubbly, bumpy, chequered, cracked, crystalline, fibrous, honeycombed, interlaced, knitted, marbled, paisley, scaly, swirly, veined, waffled, woven* and *zigzagged*

Figure 7. top left: input image showing portrait of *Sintel*–the protagonist from the identically named open source movie, others: texture attribute *veined* transferred to *Sintel* after training iterations 30, 60, 90, 120 and 150.

Figure 8. top left: input image showing portrait of *Sintel*, others: texture attribute *veined* at training iteration 120 transferred to *Sintel* at resolutions 128×92, 256×183, 512×366, 1024×761 and 2048×1461.

Acknowledgments

This research was supported in part by the National Science Foundation (NSF) grant IIS-1617917, and a Faculty gift from Facebook. Some of the GPUs used in this research were generously donated by NVIDIA.

References

[1] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. 2

[2] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012. 2

[3] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 3606–3613, 2014. 3, 6

[4] M. Cimpoi, S. Maji, I. Kokkinos, and A. Vedaldi. Deep filter banks for texture recognition, description, and segmentation. *International Journal of Computer Vision*, 118(1):65–94, 2016. 1

[5] D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. A committee of neural networks for traffic sign classification. In *The 2011 International Joint Conference on Neural Networks, IJCNN 2011, San Jose, California, USA, July 31 - August 5, 2011*, pages 1918–1921, 2011. 2

[6] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1538–1546, 2015. 1

[7] D. S. Ebert. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003. 1

[8] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, pages 1033–1038, 1999. 1

[9] L. Fei-Fei, A. Karpathy, and J. Johnson. Cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io, 2016. 2

[10] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 262–270, 2015. 1, 2, 3, 4

[11] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2414–2423, 2016. 2

[12] S. Gross and M. Wilber. Training and investigating residual nets. http://torch.ch/blog/2016/02/04/resnets.html, 2016. 4

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016. 4

[14] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1995, Los Angeles, CA, USA, August 6-11, 1995*, pages 229–238, 1995. 1

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, pages 675–678, 2014. 1

[16] 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, 3, 4

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012. 1, 2

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2

[19] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, pages 740–755, 2014. 5

[20] T. Lin and S. Maji. Visualizing and understanding deep texture representations. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2791–2799, 2016. 3, 5

[21] T. Lin, A. Roy Chowdhury, and S. Maji. Bilinear CNN models for fine-grained visual recognition. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1449–1457, 2015. 1, 2

[22] K. Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1985, San Francisco, California, USA, July 22-26, 1985*, pages 287–296, 1985. 6

[23] J. Portilla and E. P. Simoncelli. Texture modeling and synthesis using joint statistics of complex wavelet coefficients. In *IEEE workshop on statistical and computational theories of vision*, 1999. 1

[24] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–70, 2000. 2

[25] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014, Columbus, OH, USA, June 23-28, 2014*, pages 512–519, 2014. 1

[26] J. Schpok, J. Simons, D. S. Ebert, and C. D. Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, San Diego, CA, USA, July 26-27, 2003*, pages 160–166, 2003. 1

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 2, 3, 4, 5

[28] 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, 3

[29] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000, New Orleans, LA, USA, July 23-28, 2000*, pages 479–488, 2000. 1