This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

StyleBank: An Explicit Representation for Neural Image Style Transfer

Dongdong Chen[1], Lu Yuan[2], Jing Liao[2], Nenghai Yu[1], Gang Hua[2]
[1]University of Science and Technology of China [2]Microsoft Research, Beijing, China
cd722522@mail.ustc.edu.cn, {luyuan, jliao, ganghua}@microsoft.com, ynh@ustc.edu.cn

Abstract

We propose StyleBank*, which is composed of multiple convolution filter banks and each filter bank explicitly represents one style, for neural image style transfer. To transfer an image to a specific style, the corresponding filter bank is operated on top of the intermediate feature embedding produced by a single auto-encoder. The StyleBank and the auto-encoder are jointly learnt, where the learning is conducted in such a way that the auto-encoder does not encode any style information thanks to the flexibility introduced by the explicit filter bank representation. It also enables us to conduct incremental learning to add a new image style by learning a new filter bank while holding the auto-encoder fixed. The explicit style representation along with the flexible network design enables us to fuse styles at not only the image level, but also the region level. Our method is the first style transfer network that links back to traditional texton mapping methods, and hence provides new understanding on neural style transfer. Our method is easy to train, runs in real-time, and produces results that qualitatively better or at least comparable to existing methods.*

1. Introduction

Style transfer is to migrate a style from an image to another, and is closely related to texture synthesis. The core problem behind these two tasks is to model the statistics of a reference image (texture, or style image), which enables further sampling from it under certain constraints. For texture synthesis, the constraints are that the boundaries between two neighboring samples must have a smooth transition, while for style transfer, the constraints are that the samples should match the local structure of the content image. So in this sense, style transfer can be regarded as a generalization of texture synthesis.

Recent work on style transfer adopting Convolutional Neural Networks (CNN) ignited a renewed interest in this problem. On the machine learning side, it has been shown that a pre-trained image classifier can be used as a feature extractor to drive texture synthesis [11] and style transfer [12]. These CNN algorithms either apply an iterative optimization mechanism [12], or directly learn a feed-forward generator network [19, 37] to seek an image close to both the content image and the style image – all measured in the CNN (*i.e.*, pre-trained VGG-16 [36]) feature domain. These algorithms often produce more impressive results compared to the texture-synthesis ones, since the rich feature representation that a deep network can produce from an image would allow more flexible manipulation of an image.

Notwithstanding their demonstrated success, the principles of CNN style transfer are vaguely understood. After a careful examination of existing style transfer networks, we argue that the content and style are still coupled in their learnt network structures and hyper-parameters. To the best of our knowledge, an explicit representation for either style or content has not yet been proposed in these previous neural style transfer methods.

As a result, the network is only able to capture a specific style one at a time. For a new style, the whole network has to be retrained end-to-end. In practice, this makes these methods unable to scale to large number of styles, especially when the style set needs to be incrementally augmented. In addition, how to further reduce run time, network model size and enable more flexibilities to control transfer (*e.g.*, region-specific transfer), remains to be challenges yet to be addressed.

To explore an explicit representation for style, we reconsider neural style transfer by linking back to traditional texton (known as the basic element of texture) mapping methods, where mapping a texton to the target location is equivalent to a convolution between a texton and a Delta function (indicating sampling positions) in the image space.

Inspired by this, we propose *StyleBank*, which is composed of multiple convolution filter banks and each filter bank represents one style. To transfer an image to a specific style, the corresponding filter bank is convolved with the intermediate feature embedding produced by a single auto-encoder, which decomposes the original image into multiple feature response maps. This way, for the first time, we provide a clear understanding of the mechanism underneath neural style transfer.

The *StyleBank* and the auto-encoder are jointly learnt in our proposed feed-forward network. It not only allows us to simultaneously learn a bundle of various styles, but also enables a very efficient incremental learning for a new image style. This is achieved by learning a new filter bank while holding the auto-encoder fixed.

We believe this is a very useful functionality to recently emerged style transfer mobile applications (*e.g.*, Prisma) since we do not need to train and prepare a complete network for every style. More importantly, it can even allow users to efficiently create their own style models and conveniently share to others. Since the part of our image encoding is shared for variant styles, it may provide a faster and more convenient switch for users between different style models.

Because of the explicit representation, we can more conveniently control style transfer and create new interesting style fusion effects. More specifically, we can either linearly fuse different styles altogether, or produce region-specific style fusion effects. In other words, we may produce an artistic work with hybrid elements from van Gogh's and Picaso's paintings.

Compared with existing neural style transfer networks [19, 37], our proposed neural style transfer network is unique in the following aspects:

- In our method, we provide an explicit representation for styles. This enables our network to completely decouple styles from the content after learning.
- Due to the explicit style representation, our method enables region-based style transfer. This is infeasible in existing neural style transfer networks, although classical texture transfer methods were able to achieve it.
- Our method not only allows to simultaneously train multiple styles sharing a single auto-encoder, but also incrementally learn a new style without changing the auto-encoder.

The remainder of the paper is organized as follows. We summarize related work in Section 2. We devote Section 3 to the main technical design of the proposed StyleBank Network. Section 4 discusses about new characteristics of the proposed StyleBank Network when compared with previous work. We present experimental results and comparisons in Section 5. And finally we conclude in Section 6.

2. Related Work

Style transfer is very related to texture synthesis, which attempts to grow textures using non-parametric sampling of pixels [8, 39] or patches [7, 25] in a given source texture. The task of style transfer can be regarded as a problem of texture transfer [7, 10, 9], which synthesizes a texture from a source image constrained by the content of a target image. Hertzman et al. [16] further introduce the concept of image analogies, which transfer the texture from an already stylised image onto a target image. However, these methods only use low-level image features of the target image to inform the texture transfer.

Ideally, a style transfer algorithm should be able to extract and represent the semantic image content from the target image and then render the content in the style of the source image. To generally separate content from style in natural images is still an extremely difficult problem before, but the problem is better mitigated by the recent development of Deep Convolutional Neural Networks (CNN) [21].

DeepDream [1] may be the first attempt to generate artistic work using CNN. Inspired by this work, Gatys et al. [12] successfully applies CNN (pre-trained VGG-16 networks) to neural style transfer and produces more impressive stylization results compared to classic texture transfer methods. This idea is further extended to portrait painting style transfer [35] and patch-based style transfer by combining Markov Random Field (MRF) and CNN [22]. Unfortunately, these methods based on an iterative optimization mechanism are computationally expensive in run-time, which imposes a big limitation in real applications.

To make the run-time more efficient, more and more works begin to directly learn a feed-forward generator network for a specific style. This way, stylized results can be obtained just with a forward pass, which is hundreds of times faster than iterative optimization [12]. For example, Ulyanov et al. [37] propose a texture network for both texture synthesis and style transfer. Johnson et al. [19] define a perceptual loss function to help learn a transfer network that aims to produce results approaching [12]. Chuan et al. [23] introduce a Markovian Generative Adversarial Networks, aiming to speed up their previous work [22].

However, in all of these methods, the learnt feed-forward networks can only represent one specific style. For a new style, the whole network has to be retrained, which may limit the scalability of adding more styles on demand. In contrast, our network allows a single network to simultaneously learn numerous styles. Moreover, our work enables incremental training for new styles.

At the core of our network, the proposed *StyleBank* represents each style by a convolution filter bank. It is very analogous to the concept of "texton" [30, 41, 24] and filter bank in [42, 29], but *StyleBank* is defined in feature embedding space produced by auto-encoder [17] rather than image space. As we known, embedding space can provide compact and descriptive representation for original data [4, 32, 40]. Therefore, our *StyleBank* would provide a better representation for style data compared to predefined dictionaries (such as wavelet [31] or pyramid [15]).

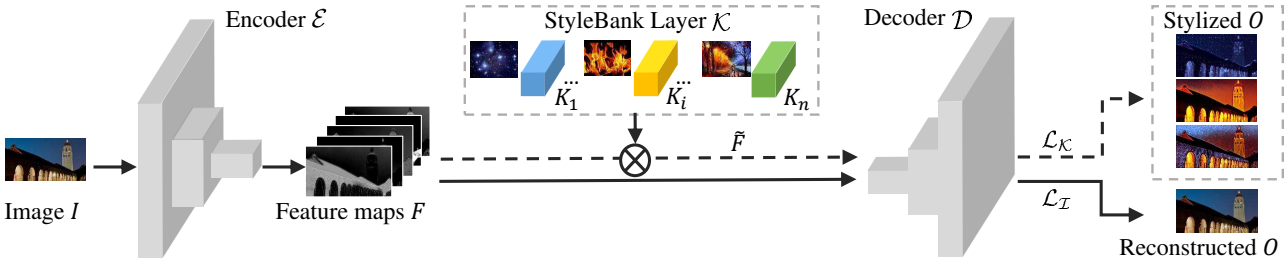


Figure 1. Our network architecture consists of three modules: image encoder $\mathcal{E}$, StyleBank layer $\mathcal{K}$ and image decoder $\mathcal{D}$

3. StyleBank Network

3.1. StyleBank

At its core, the task of neural style transfer requires a more explicit representation, like texton [30, 24] (known as the basic element of texture) used in classical texture synthesis. It may provide a new understanding for the style transfer task, and then help design a more elegant architecture to resolve the coupling issue in existing transfer networks [19, 37], which have to retrain hyper-parameters of the whole network for each newly added style end-to-end.

We build a feed-forward network based on a simple image auto-encoder (shown in Figure 1), which would first transform the input image (*i.e.*, the *content* image) into the feature space through the encoder subnetwork. Inspired by the texton concept, we introduce *StyleBank* as style representation by analogy, which is learnt from input styles.

Indeed, our StyleBank contains multiple convolution filter banks. Every filter bank represents one kind of style, and all channels in a filter bank can be regarded as bases of style elements (*e.g.*, texture pattern, coarsening or softening strokes). By convolving with the intermediate feature maps of content image, produced by auto-encoder, *StyleBank* would be mapped to the content image to produce different stylization results. Actually, this manner is analogy to texton mapping in image space, which can also be interpreted as the convolution between texton and Delta function (indicating sampling positions).

3.2. Network Architecture

Figure 1 shows our network architecture, which consists of three modules: image encoder $\mathcal{E}$, *StyleBank* layer $\mathcal{K}$ and image decoder $\mathcal{D}$, which constitute two learning branches: auto-encoder (*i.e.*, $\mathcal{E} \rightarrow \mathcal{D}$) and stylizing (*i.e.*, $\mathcal{E} \rightarrow \mathcal{K} \rightarrow \mathcal{D}$). Both branches share the same encoder $\mathcal{E}$ and decoder $\mathcal{D}$ modules.

Our network requires the *content* image I to be the input. Then the image is transformed into multi-layer feature maps F through the encoder $\mathcal{E}$: $F = \mathcal{E}(I)$. For the auto-encoder branch, we train the auto-encoder to produce an image that is as close as possible to the input image, *i.e.*, $O = \mathcal{D}(F) \rightarrow I$. In parallel, for the stylizing branch, we add an intermediate *StyleBank* layer $\mathcal{K}$ between $\mathcal{E}$ and $\mathcal{D}$. In this layer, *StyleBank* $\{K_i\}, (i = 1, 2, ..., n)$, for n styles would be respectively convolved with features F to obtain transferred features $\widetilde{F}_i$. Finally, the stylization result O_i for style i is achieved by the decoder $\mathcal{D}$: $O_i = \mathcal{D}(\widetilde{F}_i)$.

In this manner, contents could be encoded to the auto-encoder $\mathcal{E}$ and $\mathcal{D}$ as much as possible, while styles would be encoded into *StyleBank*. As a result, content and style are decoupled from our network as much as possible.

Encoder and Decoder. Following the architecture used in [19], the image encoder $\mathcal{E}$ consists of one stride-1 convolution layer and two stride-2 convolution layers, symmetrically, the image decoder $\mathcal{D}$ consists of two stride-$\frac{1}{2}$ fractionally strided convolution layers and one stride-1 convolution layer. All convolutional layers are followed by instance normalization [38] and a ReLU nolinearity except the last output layer. Instance normalization has been demonstrated to perform better than spatial batch normalization [18] in handling boundary artifacts brought by padding. Other than the first and last layers which use 9×9 kernels, all convolutional layers use 3×3 kernels. Benefited from the explicit representation, our network can remove all the residual blocks [14] used in the network presented in Johnson et al. [19] to further reduce the model size and computation cost without performance degradation.

StyleBank Layer. Our architecture allows multiple styles (by default, 50 styles, but there is really no limit on it) to be simultaneously trained in the single network at the beginning. In the *StyleBank* layer $\mathcal{K}$, we learn n convolution filter banks $\{K_i\}, (i = 1, 2, ...n)$ (referred as *StyleBank*). During training, we need to specify the i-th style, and use the corresponding filter bank K_i for forward and backward propagation of gradients. At this time, transferred features $\widetilde{F}_i$ is achieved by

$$\widetilde{F}_i = K_i \otimes F, \tag{1}$$

where $F \in \mathcal{R}^{c_{in} \times h \times w}$, $K_i \in \mathcal{R}^{c_{out} \times c_{in} \times k_h \times k_w}$, $\widetilde{F} \in \mathcal{R}^{c_{out} \times h \times w}$, c_{in} and c_{out} are numbers of feature channels for F and $\widetilde{F}$ respectively, (h, w) is the feature map size, and (k_w, k_h) is the kernel size. To allow efficient training

of new styles in our network, we may reuse the encoder $\mathcal{E}$ and the decoder $\mathcal{D}$ in our new training. We fix the trained $\mathcal{E}$ and $\mathcal{D}$, and only retrain the layer $\mathcal{K}$ with new filter banks starting from random initialization.

Loss Functions. Our network consists of two branches: auto-encoder (*i.e.*, $\mathcal{E} \rightarrow \mathcal{D}$) and stylizing (*i.e.*, $\mathcal{E} \rightarrow \mathcal{K} \rightarrow \mathcal{D}$), which are alternatively trained. Thus, we need to define two loss functions respectively for the two branches.

In the auto-encoder branch, we use MSE (Mean Square Error) between input image I and output image O to measure an *identity loss* $\mathcal{L}_{\mathcal{I}}$:

$$\mathcal{L}_{\mathcal{I}}(I, O) = \|O - I\|^2. \tag{2}$$

At the stylizing branch, we use *perceptual loss* $\mathcal{L}_{\mathcal{K}}$ proposed in [19], which consists of a content loss $\mathcal{L}_c$, a style loss $\mathcal{L}_s$ and a variation regularization loss $\mathcal{L}_{tv}(O_i)$:

$$\mathcal{L}_{\mathcal{K}}(I, S_i, O_i) = \alpha\mathcal{L}_c(O_i, I) + \beta\mathcal{L}_s(O_i, S_i) + \gamma\mathcal{L}_{tv}(O_i) \tag{3}$$

where I, S_i, O_i are the input content image, style image and stylization result (for the i-th style) respectively. $\mathcal{L}_{tv}(O_i)$ is a variation regularizer used in [2, 19]. $\mathcal{L}_c$ and $\mathcal{L}_s$ use the same definition in [12]:

$$\begin{aligned}\mathcal{L}_c(O_i, I) &= \sum\nolimits_{l\in\{l_c\}} \|F^l(O_i) - F^l(I)\|^2 \\ \mathcal{L}_s(O_i, S_i) &= \sum\nolimits_{l\in\{l_s\}} \|G_i(F^l(O_i)) - G_i(F^l(S_i))\|^2\end{aligned} \tag{4}$$

where F^l and G_i are respectively feature map and Gram matrix computed from layer l of VGG-16 network [36](pre-trained on the ImageNet dataset [34]). $\{l_c\}, \{l_s\}$ are VGG-16 layers used to respectively compute the content loss and the style loss.

Training Strategy. We employ a $(T+1)$-step alternative training strategy motivated by [13] in order to balance the two branches (auto-encoder and stylizing). During training, for every $T+1$ iterations, we first train T iterations on the branch with $\mathcal{K}$, then train one iteration for auto-encoder branch. We show the training process in Algorithm 1.

Algorithm 1 Two branches training strategy. Here λ is the tradeoff between two branches. $\Delta_{\theta_{\mathcal{K}}}$ denote gradients of filter banks in $\mathcal{K}$. $\Delta^{\mathcal{K}}_{\theta_{\mathcal{E},\mathcal{D}}}, \Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}}$ denote gradients of $\mathcal{E}, \mathcal{D}$ in stylizing and auto-encoder branches respectively.

for every $T+1$ iterations **do**
  // Training at branch $\mathcal{E} \rightarrow \mathcal{K} \rightarrow \mathcal{D}$:
  for $t = 1$ to T **do**
    • Sample m images $X = \{x_i\}$ and style indices $Y = \{y_i\}$, $i \in \{1, ..., m\}$ as one mini-batch.
    • Update $\mathcal{E}, \mathcal{D}$ and $\{K_j\}, j \in Y$:
      $\Delta^{\mathcal{K}}_{\theta_{\mathcal{E},\mathcal{D}}} \leftarrow \bigtriangledown_{\theta_{\mathcal{E},\mathcal{D}}}\mathcal{L}_{\mathcal{K}}$
      $\Delta_{\theta_{\mathcal{K}}} \leftarrow \bigtriangledown_{\theta_{\mathcal{K}}}\mathcal{L}_{\mathcal{K}}$
  end for
  // Training at branch $\mathcal{E} \rightarrow \mathcal{D}$:
  • Update $\mathcal{E}, \mathcal{D}$ only:
    $\Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}} \leftarrow \bigtriangledown_{\theta_{\mathcal{E},\mathcal{D}}}\mathcal{L}_{\mathcal{I}}$
    $\Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}} \leftarrow \lambda\frac{\|\Delta^{\mathcal{K}}_{\theta_{\mathcal{E},\mathcal{D}}}\|}{\|\Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}}\|}\Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}}$
end for

3.3. Understanding StyleBank and Auto-encoder

For our new representation of styles, there are several questions one might ask:

1) ***How does StyleBank represent styles?***

After training the network, each style is encoded in one convolution filter bank. Each channel of filter bank can be considered as dictionaries or bases in the literature of representation learning method [3]. Different weighted combinations of these filter channels can constitute various style elements, which would be the basic elements extracted from the style image for style synthesis. We may link them to "textons" in texture synthesis by analogy.

For better understanding, we try to reconstruct style elements from a learnt filter bank in an exemplar stylization image shown in Figure 2. We extract two kinds of representative patches from the stylization result (in Figure 2(b))– stroke patch (indicated by red box) and texture patch (indicated by green box) as an object to study. Then we apply two operations below to visualize what style elements are learnt in these two kinds of patches.

First, we mask out other regions but only remain these corresponding positions of the two patches in feature maps (as shown in Figure 2(c)(d)), that would be convolved with the filter bank (corresponding to a specific style). We further plot feature responses in Figure 2(e) for the two patches along the dimension of feature channels. As we can observe, their responses are actually sparsely distributed and some peak responses occur at individual channels. Then, we only consider non-zero feature channels for convolution and their convolved channels of filter bank (marked by green and red colors in Figure 2(f)) indeed contribute to a certain style element. Transferred features are then passed to the decoder. Recovery style elements are shown in Figure 2(g), which are very close in appearance to the original style patches (Figure 2(i)) and stylization patches (Figure 2(j)).

To further explore the effect of kernel size (k_w, k_h) in the StyleBank, we set a comparison experiment to train our network with two different kernel size of $(3, 3)$ and $(7, 7)$. Then we use similar method to visualize the learnt filter banks, as shown in Fig. 3. Here the green and red box indicate representative patches from (3,3) and (7,7) kernels

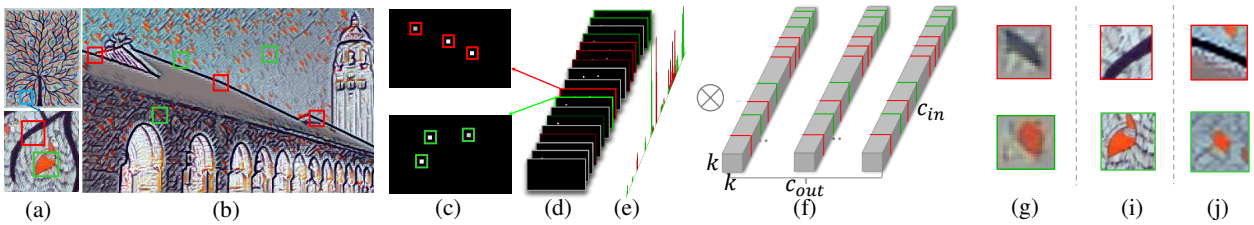


Figure 2. Reconstruction of the style elements learnt from two kinds of representative patches in an exemplar stylization image.

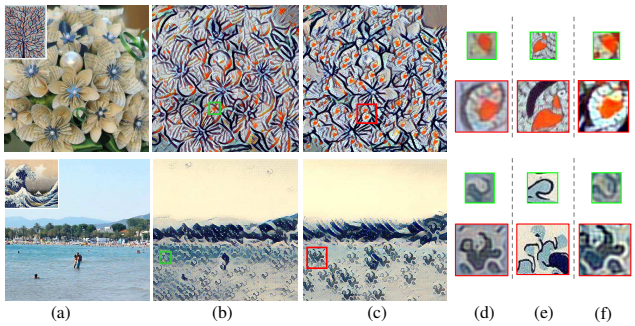


Figure 3. Learnt style elements of different StyleBank kernel sizes. (b) and (c) are stylization results of $(3, 3)$ and $(7, 7)$ kernels respectively. (d), (e) and (f) respectively show learnt style elements, original style patches and stylization patches.

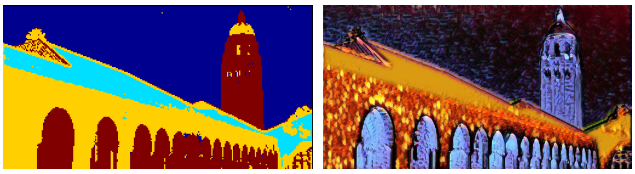


Figure 4. k-means clustering result of feature maps(left) and corresponding stylization result(right).

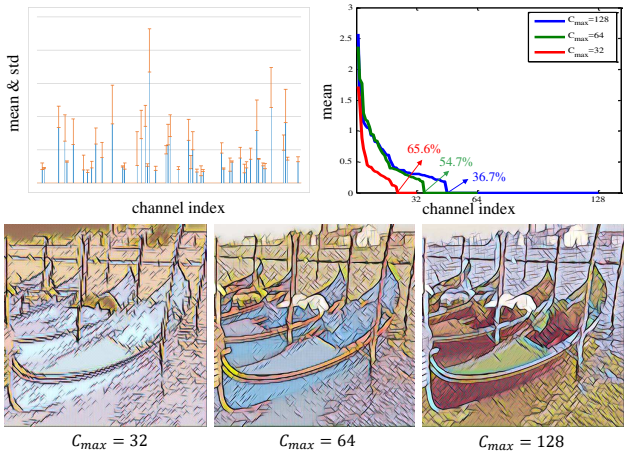


Figure 5. Sparsity analysis. Top-left: means and standard deviations of per-channel average response; top-right: distributions of sorted means of per-channel average response for different model sizes ($C_{max} = 32, 64, 128$); bottom: corresponding stylization results.

respectively. After comparison, it is easy to observe that bigger style elements can be learnt with larger kernel size. For example, in the bottom row , bigger sea spray appears in the stylization result with (7,7) kernels. That suggests our network supports the control on the style element size by tuning parameters to better characterize the example style.

2) *What is the content image encoded in?*

In our method, the auto-encoder is learnt to decompose the content image into multi-layer feature maps, which are independent of any styles. When further analyzing these feature maps, we have two observations.

First, these features can be spatially grouped into meaningful clusters in some sense (*e.g.*, colors, edges, textures). To verify this point, we extract each feature vector at every position of feature maps. Then, an unsupervised clustering (*e.g.*, K-means algorithms) is applied to all feature vectors (based on L2 normalized distance). Finally, we can obtain the clustering results shown in left of Figure 4, which suggests a certain segmentation to the content image.

Comparing the right stylization result with left clustering results, we can easily find that different segmented regions are indeed rendered with different kinds of colors or textures. For regions with the same cluster label, the filled color or textures are almost the same. As a result, our auto-encoder may enable region-specific style transfer.

Second, these features would distribute sparsely in channels. To exploit this point, we randomly sample 200 content images, and for each image, we compute the average of all non-zero responses at every of 128 feature channels (in the final layer of encoder). And then we plot the means and standard deviations of those per-channel averages among 200 images in the top-left of Figure 5. As we can see, valuable responses consistently exist at certain channels. One possible reason is that these channels correspond to specific style elements for region-specific transfer, which is in consistency with our observation in Figure 2(e).

The above sparsity property will drive us to consider smaller model size of the network. We attempt to reduce all channel numbers in our auto-encoder and StyleBank layer by a factor of 2 or 4. Then the maximum channel number C_{max} become 64, 32 respectively from the original 128. We also compute and sort the means of per-channel averages, as plotted in the top-right of Fig. 5. We can observe that the

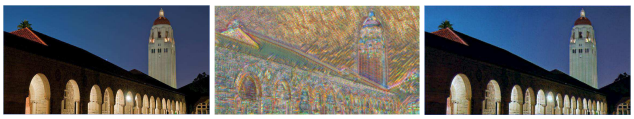
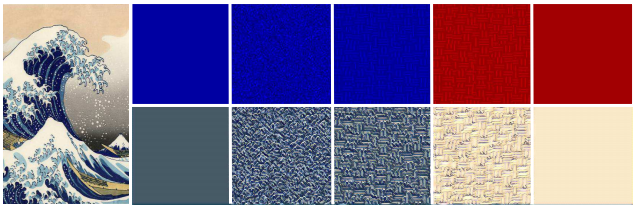


Figure 6. Illustration of the effects of two branches. The middle and right ones are reconstructed input image (left) with and without auto-encoder branch during training.



(a) (b) (c) (d) (e) (f)

Figure 7. Stylization result of a toy image, which consists of four parts of different color or different texture.

final layer of our encoder still maintains the sparsity even for smaller models although sparsity is decreased in smaller models ($C_{max} = 32$). On the bottom of Figure 5, we show corresponding stylization results of $C_{max} = 32, 64, 128$ respectively. By comparison, we can notice that $C_{max} = 32$ obviously produces worse results than $C_{max} = 128$ since the latter may encourage better region decomposition for transfer. Nevertheless, there may still be a potential to design a more compact model for content and style representation. We leave that to our future exploration.

3) *How are content and style decoupled from each other?*

To further know how well content is decoupled from style, we need to examine if the image is completely encoded in the auto-encoder. We compare two experiments with and without the auto-encoder branch in our training. When we only consider the stylizing branch, the decoded image (shown in the middle of Figure 6) produced by solely auto-encoder without $\mathcal{K}$ fails to reconstruct the original input image (shown in the left of Figure 6), and instead seems to carry some style information. When we enable the auto-encoder branch in training, we obtain the final image (shown in the right of Figure 6) reconstructed from the auto-encoder, which has very close appearance to the input image. Consequently, the content is explicitly encoded into the auto-encoder, and independent of any styles. This is very convenient to carry multiple styles learning in a single network and reduce the interferences among different styles.

4) *How does the content image control style transfer?*

To know how the content controls style transfer, we consider a toy case shown in Figure 7. On the top, we show the input toy image consisting of five regions with variant colors or textures. On the bottom, we show the output stylization result. Below are some interesting observations:

- For input regions with different colors but without textures, only a purely color transfer is applied (see Figure 7 (b)(f)).
- For input regions with the same color but different textures, the transfer consists of two parts: the same color transfer and different texture transfer influenced by appearance of input textures. (see Figure 7 (c)(d)).
- For input regions with different colors but the same textures, the results have the same transferred textures but different target colors (see Figure 7 (d)(e)).

4. Capabilities of Our Network

Because of an explicit representation, our proposed feed-forward network provides additional capabilities, when compared with previous feed-forward networks for style transfer. They may bring new user experiences or generate new stylization effects compared to previous methods.

4.1. Incremental Training

Previous style transfer networks (*e.g.*, [19, 37, 22]) have to be retrained for a new style, which is very inconvenient. In contrast, an iterative optimization mechanism [12] provides an online-learning for any new style, which would take several minutes for one style on GPU (*e.g.*, Titan X). Our method has virtues of both feed-forward networks [19, 37, 22] and iterative optimization method [12]. We enable an incremental training for new styles, which has comparable learning time to the online-learning method [12], while preserving efficiency of feed-forward networks [19, 37, 22].

In our configuration, we first jointly train the auto-encoder and multiple filter banks (50 styles used at the beginning) with the strategy described in Algorithm 1. After that, it allows to incrementally augment and train the *StyleBank* layer for new styles by fixing the auto-encoder. The process converges very fast since only the augmented part of the *StyleBank* would be updated in iterations instead of the whole network. In our experiments, when training with Titan X and given training image size of 512, it only takes around 8 minutes with about $1,000$ iterations to train a new style, which can speed up the training time by $20 \sim 40$ times compared with previous feed-forward methods.

Figure 8 shows several stylization results of new styles by incremental training. It obtains very comparable stylization results to those from fresh training, which retrains the whole network with the new styles.

4.2. Style Fusion

We provide two different types of style fusion: linear fusion of multiple styles, and region-specific style fusion.

Linear Fusion of Styles. Since different styles are encoded into different filter banks $\{K_i\}$, we can linearly fuse

Figure 8. Comparison between incremental training (*Left*) and fresh training (*Right*). The target styles are shown on the top-left.

Figure 9. Results by linear combination of two style filter banks.

multiple styles by simply linearly fusing filter banks in the *StyleBank* layer. Next, the fused filter bank is used to convolve with content features F:

$$\widetilde{F} = (\sum_{i=1}^{m} w_i * K_i) \otimes F \qquad \sum_{i=1}^{m} w_i = 1, \tag{5}$$

where m is the number of styles, K_i is the filter bank of style i. $\widetilde{F}$ is then fed to the decoder. Figure 9 shows such linear fusion results of two styles with variant fusion weight w_i.

Region-specific Style Fusion. Our method naturally allows a region-specific style transfer, in which different image regions can be rendered by various styles. Suppose that the image is decomposed into n disjoint regions by automatic clustering (*e.g.*, K-means mentioned in Section 3.3 or advanced segmentation algorithms [5, 33]) in our feature space, and M_i denotes every region mask. The feature maps can be described as $F = \sum_{i=1}^{m}(M_i \times F)$. Then region-specific style fusion can be formulated as Equation (6):

$$\widetilde{F} = \sum_{i=1}^{m} K_i \otimes (M_i \times F), \tag{6}$$

where K_i is the i-th filter bank.

Figure 10 shows such a region-specific style fusion result which exactly borrows styles from two famous paintings of Picasso and Van Goph. Superior to existing feed-forward networks, our method naturally obtains image decomposition for transferring specific styles, and passes the network only once. On the contrary, previous approaches have to pass the network several times and finally montage different styles via additional segmentation masks.

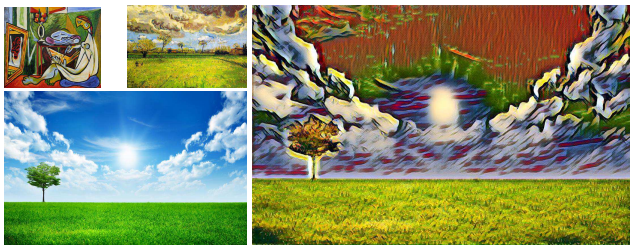


Figure 10. Region-specific style fusion with two paintings of Picasso and Van Gophm, where the regions are automatically segmented with K-means method.

5. Experiments

Training Details Our network is trained on 1000 content images randomly sampled from Microsoft COCO dataset [27] and 50 style images (from existing papers and the Internet). Each content image is randomly cropped to 512×512, and each style image is scaled to 600 on the long side. We train the network with a batch size of 4 ($m = 4$ in Algorithm 1) for $300k$ iterations. And the Adam optimization method [20] is adopted with the initial learning rate of 0.01 and decayed by 0.8 at every $30k$ iterations. In all of our experiments, we compute content loss at layer $relu4\_2$ and style loss at layer $relu1\_2$, $relu2\_2$, $relu3\_2$, and $relu4\_2$ of the pre-trained VGG-16 network. We use $T = 2, \lambda = 1$ (in Algorithm 1) in our two branches training.

5.1. Comparisons

In this section, we compare our method with other CNN-based style transfer approaches [12, 19, 37, 6]. For fair comparison, we directly borrow results from their papers. It is difficult to compare results with different abstract stylization, which is indeed controlled by the ratio α/β in Equation (3) and different work may use their own ratios to present results. For comparable perception quality, we choose different α, β in each comparison. More results are available in our supplementary material[1].

Compared with the Iterative Optimization Method. We use $\alpha/\beta = 1/100$ (in Equation (3)) to produce comparable perceptual stylization in Fig. 11. Our method, like all other feed-forward methods, creates less abstract stylization results than optimization method [12]. It is still difficult to judge which one is more appealing in practice. However, our method, like other feed-forward methods, could be hundreds of times faster than optimization-based methods.

Compared with Feed-forward Networks. In Figure 12 and Figure 13, we respectively compare our results with two feed-forward network methods [19, 37]. We use $\alpha/\beta = 1/50$ (in Equation (3)) in both comparisons. Ulyanov et al.

[1] http://www.dongdongchen.bid/pubs/sbk_supp.pdf

Inputs Our Results Gatys *et al.*'s Results

Figure 11. Comparison with optimization-based method [12].

Inputs Our Results Johnson *et al.*'s Results

Figure 12. Comparison with the feed-forward network in [19].

Inputs Our Results Ulyanov *et al.*'s Results

Figure 13. Comparison with the feed-forward network in [37].

[37] design a shallow network specified for the texture synthesis task. When it is applied to style transfer task, the stylization results are more like texture transfer, sometimes randomly pasting textures to the content image. Johnson et al. [19] use a much deeper network and often obtain better results. Compared with both methods, our results obviously present more region-based style transfer, for instance, the portrait in Figure 13, and river/grass/forest in Fig. 12.

Inputs Our Results Dumoulin *et al.*'s Results

Figure 14. Comparison with the synchronal learning [6],

Moreover, different from their one-network-per-style training, all of our styles are jointly trained in a single model.

Compared with other Synchronal Learning. Dumoulin et al., in their very recent work [6], introduce the "conditional instance normalization" mechanism derived from [38] to jointly train multiple styles in one model, where parameters of different styles are defined by different instance normalization factors (scaling and shifting) after each convolution layer. However, their network does not explicitly decouple the content and styles as ours. Compared with theirs, our method seems to allow more abilities of region-specific transfer. As shown in Fig. 14, our stylization results better correspond to the natural regions of content images. In this comparison, we use $\alpha/\beta = 1/25$ (in Equation (3)).

6. Discussion and Conclusion

In this paper, we have proposed a novel explicit representation for style and content, which can be well decoupled by our network. The decoupling allows faster training (for multiple styles, and new styles), and enables new interesting style fusion effects, like linear and region-specific style transfer. More importantly, we present a new interpretation to neutral style transfer which may inspire other understandings for image reconstruction, and restoration.

There are still some interesting issues for further investigation. For example, the auto-encoder may integrate semantic segmentation [28, 26] as additional supervision in the region decomposition, which would help create more impressive region-specific transfer. Besides, our learnt representation does not fully utilize all channels, which may imply a more compact representation.

Acknowledgement. This work is partially supported by National Natural Science Foundation of China (NSFC, NO.61371192)

References

[1] M. T. Alexander Mordvintsev, Christopher Olah. Inceptionism: Going deeper into neural networks, 2015.

[2] H. A. Aly and E. Dubois. Image up-sampling using total-variation regularization with a new observation model. *IEEE Transactions on Image Processing*, 14(10):1647–1659, 2005.

[3] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155, 2003.

[5] Y. Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *Proceedings of 8th IEEE International Conference on Computer Vision*, volume 1, pages 105–112. IEEE, 2001.

[6] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. *arXiv preprint arXiv:1610.07629*, 2016.

[7] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 341–346. ACM, 2001.

[8] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, volume 2, pages 1033–1038. IEEE, 1999.

[9] M. Elad and P. Milanfar. Style-transfer via texture-synthesis. *arXiv preprint arXiv:1609.03057*, 2016.

[10] O. Frigo, N. Sabater, J. Delon, and P. Hellier. Split and match: Example-based adaptive patch sampling for unsupervised style transfer. In *Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

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

[12] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

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

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[15] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd a Annual Conference on Computer Graphics and Interactive Techniques*, pages 229–238. ACM, 1995.

[16] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 327–340. ACM, 2001.

[17] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[19] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *arXiv preprint arXiv:1603.08155*, 2016.

[20] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[22] C. Li and M. Wand. Combining markov random fields and convolutional neural networks for image synthesis. *arXiv preprint arXiv:1601.04589*, 2016.

[23] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. *arXiv preprint arXiv:1604.04382*, 2016.

[24] Y. Li, T. Wang, and H.-Y. Shum. Motion texture: a two-level statistical model for character motion synthesis. In *ACM Transactions on Graphics (ToG)*, volume 21, pages 465–472. ACM, 2002.

[25] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (ToG)*, 20(3):127–150, 2001.

[26] D. Lin, J. Dai, J. Jia, K. He, and J. Sun. Scribblesup: Scribble-supervised convolutional networks for semantic segmentation. *arXiv preprint arXiv:1604.05144*, 2016.

[27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[28] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the 25th IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[29] Y. Lu, S.-C. Zhu, and Y. N. Wu. Learning frame models using cnn filters. *arXiv preprint arXiv:1509.08379*, 2015.

[30] J. Malik, S. Belongie, J. Shi, and T. Leung. Textons, contours and regions: Cue integration in image segmentation. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, volume 2, pages 918–925. IEEE, 1999.

[31] 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.

[32] S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee. Deep visual analogy-making. In *Advances in Neural Information Processing Systems 28*, pages 1252–1260. 2015.

[33] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM transactions on graphics (ToG)*, volume 23, pages 309–314. ACM, 2004.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein,

et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[35] A. Selim, M. Elgharib, and L. Doyle. Painting style transfer for head portraits using convolutional neural networks. *ACM Transactions on Graphics (ToG)*, 35(4):129, 2016.

[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[37] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *arXiv preprint arXiv:1603.03417*, 2016.

[38] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[39] L.-Y. 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*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000.

[40] T. Xue, J. Wu, K. L. Bouman, and W. T. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. *arXiv preprint arXiv:1607.02586*, 2016.

[41] S.-C. Zhu, C.-E. Guo, Y. Wang, and Z. Xu. What are textons? *International Journal of Computer Vision*, 62(1-2):121–143, 2005.

[42] S. C. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.