

# Explicit Filterbank Learning for Neural Image Style Transfer and Image Processing

Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu and Gang Hua, *IEEE Fellow*

**Abstract**—Image style transfer is to re-render the content of one image with the style of another. Most existing methods couple content and style information in their network structures and hyper-parameters, and learn it as a black-box. For better understanding, this paper aims to provide a new explicit decoupled perspective. Specifically, we propose *StyleBank*, which is composed of multiple convolution filter banks and each filter bank explicitly represents one style. To transfer an image to a specific style, the corresponding filter bank is operated on the intermediate feature produced by a single auto-encoder. The *StyleBank* and the auto-encoder are jointly learnt in such a way that the auto-encoder does not encode any style information. This explicit representation also enables us to conduct incremental learning to add a new style and 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 provides new understanding on neural style transfer. We further apply this general filterbank learning idea to two different multi-parameter image processing tasks: edge-aware image smoothing and denoising. Experiments demonstrate that it can achieve comparable results to its single parameter setting counterparts.

**Index Terms**—Image Processing and Computer Vision, Style Transfer

## 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 both 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 [1] and style transfer [2]. These CNN algorithms either apply an iterative optimization mechanism [2], [3], [4], or directly learn a feed-forward generator network [5], [6], [7] 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 [8]) feature domain. These algorithms often produce more impressive results compared to the traditional texture-synthesis based 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 existing neural style transfer methods. In addition, how to further enable more flexibilities to control transfer (*e.g.*, region-specific transfer), remain 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. However in these methods, simply stitching these textons in the original image space often suffer from boundary discontinuity artifacts. To alleviate this problem, some advanced seamless stitching algorithms (*e.g.*, graphcut) need to be used.

Inspired by this, we propose *StyleBank*, which is composed of multiple convolution filter banks and each filter bank learns to represent one style. For the above second issue, we further adopt feature space to do convolution rather than the original image space, and use a following decoder to automatically learn to solve the seamless stitching problem. Combining these two points, to transfer an image to a specific style, the corresponding filter bank is first convolved with the intermediate feature embedding produced by an encoder, which decomposes the original image into multiple feature response maps. Then the transformed (convolved) feature will be further fed into the following decoder to obtain the final specific stylization result. This way, for the first time, we provide a clear understanding of the mechanism underneath neural style transfer.

To explicitly decouple the content and style, we further propose a new “T+I” training strategy to jointly train the *StyleBank* and the auto-encoder (encoder+decoder) in an alternative way. Experiments demonstrate that 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

- Dongdong Chen and Nenghai Yu are with Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei, Anhui 230026, China.  
E-mail: cd722522@mail.ustc.edu.cn, ynh@ustc.edu.cn
- Lu Yuan is with Microsoft Research, Redmond, Washington 98052, USA.  
E-mail: luyuan@microsoft.com
- Jing Liao is with Department of Computer Science, City University of Hong Kong  
E-mail: jingliao@cityu.edu.hk
- Gang Hua is with Wormpex AI Research LLC, WA 98004, US.  
E-mail: ganghua@gmail.com
- Jing Liao and Gang Hua are the corresponding authors.

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 Picasso’s paintings. To better understand the underlying working principles, we also use some visualization strategies to investigate what the network learns and provide some new understandings.

Because the proposed filterbank learning algorithm is essentially a natural and general way to achieve multiple functionalities within one single network. We further apply this idea to two different multi-parameter image processing tasks: image smoothing, and denoising. Since these types of operators often contain some parameters to control the final results, some previous methods often train multiple different models for each specific parameter setting. By using the proposed filterbank learning idea, we can also use different filter banks for different parameter settings and enables multiple settings within one single network too. Experiments demonstrate that it can achieve comparable results to its single parameter setting counterparts.

To sum up, our contributions can be summarized in four-fold as below:

- 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 naturally provides some flexibilities like region-based style transfer, which is infeasible in existing neural style transfer networks.
- We have provided very detailed analysis of the working principles of our style transfer network and linked it back to some traditional texture synthesis methods, which may inspire more research works in this field.
- The proposed filterbank learning idea is very general. We have applied this idea to two different multi-parameter image processing tasks, and achieved comparable performance to their single parameter setting counterparts.

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 Style-Bank 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. To demonstrate the generality of our method, we further apply this idea to two different image processing tasks in Section 6. And finally we conclude in Section 7.

## 2 RELATED WORK

Style transfer is very related to texture synthesis, which attempts to grow textures using non-parameteric sampling of pixels [9], [10] or patches [11], [12] in a given source texture. The task of style transfer can be regarded as a problem of texture transfer [11],

[13], [14], which synthesizes a texture from a source image constrained by the content of a target image. Hertzman et al. [15] further introduce the concept of image analogies, which transfer the texture from an already stylized image onto a target image. However, these methods only use low-level image features of the target image to inform the texture transfer.

Ideally, style transfer algorithms 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) [16].

DeepDream [17] may be the first attempt to generate artistic work using CNN. Inspired by this work, Gatys et al. [2] successfully apply 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 [18] and patch-based style transfer by combining Markov Random Field (MRF) and CNN [3]. In work [4], Liao *et al.* reformulate style transfer as a special visual attribute transfer problem and propose a new technique named “deep image analogy” to solve it. 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[2]. For example, Ulyanov et al.[6] propose a texture network for both texture synthesis and style transfer. Johnson et al.[5] define a perceptual loss function to help learn a transfer network that aims to produce results approaching [2]. Chuan et al. [7] introduce a Markovian Generative Adversarial Networks, aiming to speed up their previous work [3]. 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.

Very recently, many following works are proposed to enable multiple styles [19], [20], [21] or arbitrary styles [22], [23], [24]. In multiple style transfer methods, they often use one-hot vector [20] or different shifting/scaling parameters of instance normalization layers [19] to represent different styles. While for most arbitrary style transfer methods, they often use a pre-trained VGG network and transform the content feature based on the style feature statistics, then retrained a new decoder to decode the transformed features back to the image space. Though these methods can achieve quite good stylization results, they do not provide an explicit and compact way to represent styles. Therefore, the underlying working principles of these methods are still vague. However, our method aims to decouple the content and style in an explicit way.

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” [25], [26], [27] and filter bank in [28], [29], but *StyleBank* is defined in feature embedding space produced by auto-encoder [30] rather than image space. As we known, embedding space can provide compact and descriptive representation for original data [31], [32], [33]. Therefore, our

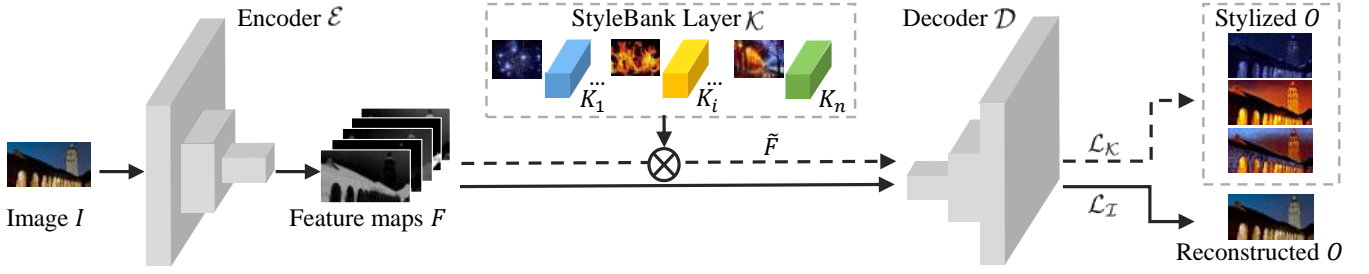


Fig. 1. Our network architecture consists of three modules: image encoder  $\mathcal{E}$ , StyleBank layer  $\mathcal{K}$  and image decoder  $\mathcal{D}$

*StyleBank* would provide a better representation for style data compared to predefined dictionaries (such as wavelet [34] or pyramid [35]).

### 3 STYLEBANK NETWORKS

#### 3.1 StyleBank

At its core, the task of neural style transfer requires a more explicit representation, like texton [25], [27] (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 [5], [6], 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). The difference is that we adopt this operation in the feature space but not the image space. Because the following decoder can learn to solve the seamless stitching problem, it can avoid boundary discontinuity artifacts without the need of extra post processing.

#### 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*  $\{K_i\}$ , ( $i = 1, 2, \dots, n$ ),

TABLE 1  
The default network configuration details.

Layer type	channel number	kernel size	stride
Conv	32	$9 \times 9$	1
Conv	64	$3 \times 3$	2
Conv	128	$3 \times 3$	2
Stylebank	128	$3 \times 3$	1
Deconv	64	$3 \times 3$	2
Deconv	32	$3 \times 3$	2
Conv	3	$9 \times 9$	1

for  $n$  styles would be respectively convolved with features  $F$  to obtain transferred features  $\tilde{F}_i$ . Finally, the stylization result  $O_i$  for style  $i$  is achieved by the decoder  $\mathcal{D}$ :  $O_i = \mathcal{D}(\tilde{F}_i)$ .

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 [5], the image encoder  $\mathcal{E}$  consists of one stride-1 convolution layer (channel number is 32) and two stride-2 convolution layers (channel numbers are 64 and 128 respectively), 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 [36] and a ReLU nonlinearity except the last output layer. Instance normalization has been demonstrated to perform better than spatial batch normalization [37] in handling boundary artifacts brought by padding. Other than the first and last layers which use  $9 \times 9$  kernels, all convolutional layers use  $3 \times 3$  kernels. In our default simplified version, we remove all the residual blocks used in the network presented in [5] to reduce the model size and computation cost. The detailed configuration parameters are shown in Table 1.

**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  $\{K_i\}$ , ( $i = 1, 2, \dots, n$ ) (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  $\tilde{F}_i$  is achieved by

$$\tilde{F}_i = K_i \otimes F, \quad (1)$$

where  $F \in \mathcal{R}^{c_{in} \times h \times w}$ ,  $K_i \in \mathcal{R}^{c_{out} \times c_{in} \times k_h \times k_w}$ ,  $\tilde{F} \in \mathcal{R}^{c_{out} \times h \times w}$ ,  $c_{in}$  and  $c_{out}$  are numbers of feature channels for  $F$  and  $\tilde{F}$  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*  $\mathcal{L}_{\mathcal{I}}$ :

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

At the stylizing branch, we use *perceptual loss*  $\mathcal{L}_{\mathcal{K}}$  proposed in [5], which consists of a content loss  $\mathcal{L}_c$ , a style loss  $\mathcal{L}_s$  and a variation regularization loss  $\mathcal{L}_{tv}(O_i)$ :

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

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 [38], [5].  $\mathcal{L}_c$  and  $\mathcal{L}_s$  use the same definition in [2]:

$$\begin{aligned} \mathcal{L}_c(O_i, I) &= \sum_{l \in \{l_c\}} \|F^l(O_i) - F^l(I)\|^2 \\ \mathcal{L}_s(O_i, S) &= \sum_{l \in \{l_s\}} \|G(F^l(O_i)) - G(F^l(S_i))\|^2 \end{aligned} \quad (4)$$

where  $F^l$  and  $G$  are respectively feature map and Gram matrix computed from layer  $l$  of pretrained VGG-16 network [8].  $\{l_c\}, \{l_s\}$  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 [39] 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  $\lambda$  is the tradeoff between two branches.  $\Delta_{\theta_{\mathcal{K}}}$  denote gradients of filter banks in  $\mathcal{K}$ .  $\Delta_{\theta_{\mathcal{E}, \mathcal{D}}}^{\mathcal{K}}, \Delta_{\theta_{\mathcal{E}, \mathcal{D}}}^{\mathcal{I}}$  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
       $Y = \{y_i\}, i \in \{1, \dots, m\}$  as one mini-batch.
    • Update  $\mathcal{E}, \mathcal{D}$  and  $\{K_j\}, j \in Y$ :
       $\Delta_{\theta_{\mathcal{E}, \mathcal{D}}}^{\mathcal{K}} \leftarrow \nabla_{\theta_{\mathcal{E}, \mathcal{D}}} \mathcal{L}_{\mathcal{K}}$ 
       $\Delta_{\theta_{\mathcal{K}}} \leftarrow \nabla_{\theta_{\mathcal{K}}} \mathcal{L}_{\mathcal{K}}$ 
  end for
  // Training at branch  $\mathcal{E} \rightarrow \mathcal{D}$ :
  • Update  $\mathcal{E}, \mathcal{D}$  only:
     $\Delta_{\theta_{\mathcal{E}, \mathcal{D}}}^{\mathcal{I}} \leftarrow \nabla_{\theta_{\mathcal{E}, \mathcal{D}}} \mathcal{L}_{\mathcal{I}}$ 
     $\Delta_{\theta_{\mathcal{E}, \mathcal{D}}}^{\mathcal{I}} \leftarrow \lambda \frac{\|\Delta_{\theta_{\mathcal{E}, \mathcal{D}}}^{\mathcal{K}}\|}{\|\Delta_{\theta_{\mathcal{E}, \mathcal{D}}}^{\mathcal{I}}\|} \Delta_{\theta_{\mathcal{E}, \mathcal{D}}}^{\mathcal{I}}$ 
end for

```

---

**Extension to Hierarchy Stylebank.** In the above default simplified design, each style is only represented by one single filter bank.

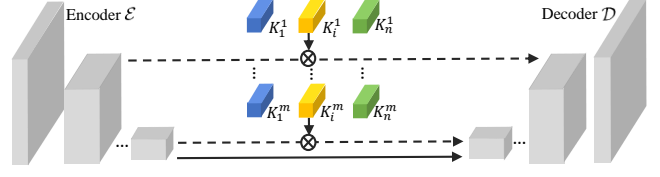


Fig. 2. The overall network structure for the hierarchy stylebank design.

However, in traditional image processing field, images are often decomposed in a hierarchy way (e.g., Gaussian Pyramid, Laplacian Pyramid) and each hierarchy level represents one specific level of image information. In fact, features extracted by existing deep neural networks also follow a similar hierarchy way. Inspired by this, we also try another hierarchy stylebank design as shown in Figure 2. In this setting, each style will be represented by multiple hierarchy filter banks ( $K_i^1, \dots, K_i^m$ ). Correspondingly, we will extract different levels of intermediate features ( $F^1, \dots, F^m$ ), and obtain transferred feature  $\tilde{F}_i^j$  by convolving  $K_i^j$  with  $F^j$ , formally:

$$\tilde{F}_i^j = K_i^j \otimes F^j, j \in \{1, \dots, m\} \quad (5)$$

Note that in this setting, we add the removed five residual blocks back into the auto-encoder network (three into encoder, and two into decoder) to get more hierarchy features. By default, we set  $m$  as 2, and regard the first 1/2 downsampled feature maps and the 1/4 downsampled output feature of the encoder as  $F^1, F^2$  respectively. The transferred feature  $\tilde{F}_i^1$  is first concatenated with the symmetric decoder feature then reduce its channel dimension from 128 back to 64 by using an extra  $1 \times 1$  convolution layer.

### 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 [40]. 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 3. We extract two kinds of representative patches from the stylization result (in Figure 3(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 3(c)(d)), that would be convolved with the filter bank (corresponding to a specific style). We further plot feature responses in Figure 3(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



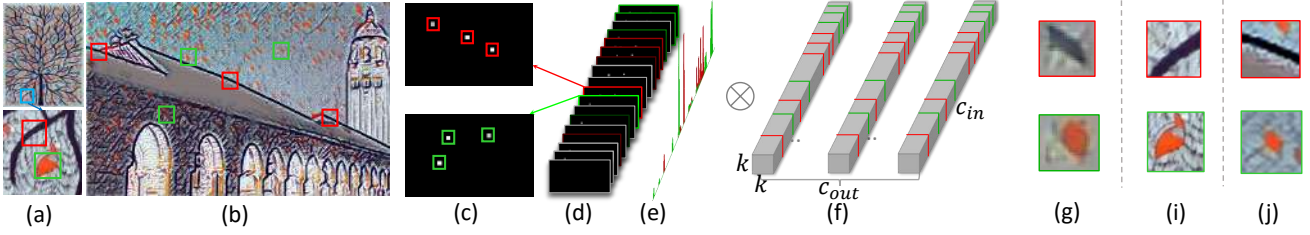


Fig. 3. Reconstruction of the style elements learnt from two kinds of representative patches in an exemplar stylization image.

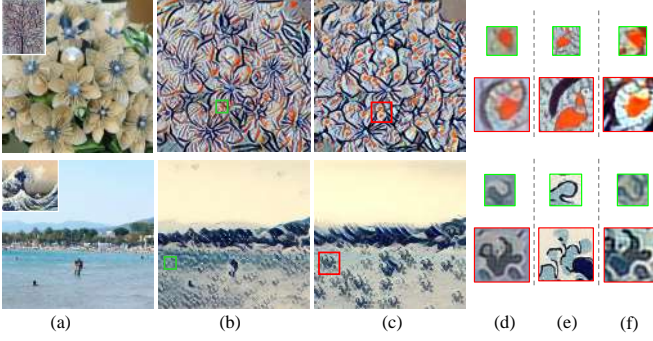


Fig. 4. Learnt style elements of different StyleBank kernel sizes. (b) and (c) are stylization results of  $(3, 3)$  and  $(7, 7)$  kernels respectively. (d), (e) and (f) respectively show learnt style elements, original style patches and stylization patches.

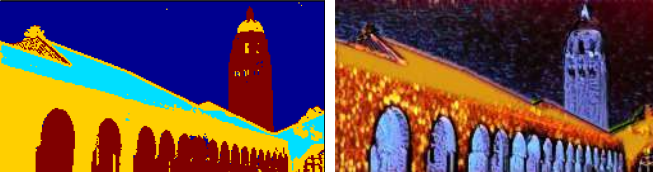


Fig. 5. k-means clustering result of feature maps(left) and corresponding stylization result(right).

convolution and their convolved channels of filter bank (marked by green and red colors in Figure 3(f)) indeed contribute to a certain style element. Transferred features are then passed to the decoder. Recovery style elements are shown in Figure 3(g), which are very close in appearance to the original style patches (Figure 3(i)) and stylization patches (Figure 3(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. 4. Here the green and red box indicate representative patches from  $(3, 3)$  and  $(7, 7)$  kernels 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.

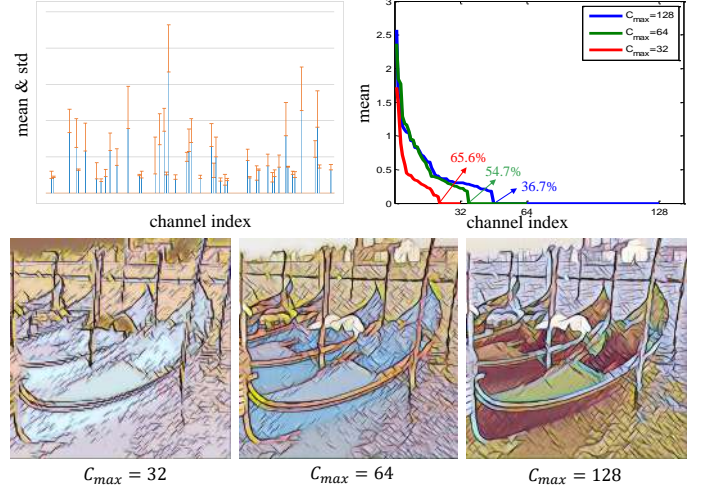


Fig. 6. Sparsity analysis. Top-left: means and standard deviations of per-channel average response; top-right: distributions of sorted means of per-channel average response for different model sizes ( $C_{max} = 32, 64, 128$ ); bottom: corresponding stylization results.

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 5, 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 6. 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 3(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



Fig. 7. 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.

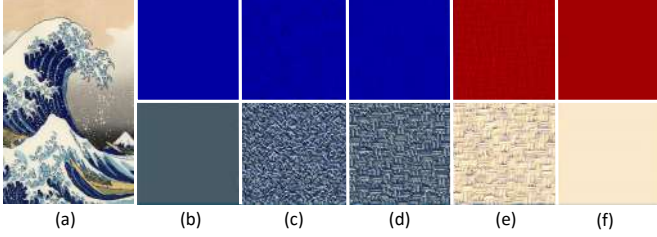


Fig. 8. Stylization result of a toy image, which consists of four parts of different color or different texture.

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. 6. We can observe that the 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 6, 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.

### 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 7) produced by solely auto-encoder without  $\mathcal{K}$  fails to reconstruct the original input image (shown in the left of Figure 7), 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 7) 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 8. 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 8 (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 8 (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 8 (d)(e)).

### 5) How to visualize all the learned style elements?

In Section 3.3, we have provided one way to visualize the learned style elements at one specific position. To further study how many different style elements are learned for different styles, we adopt another simple but effective visualization way. Because style transfer is essentially to convert content patches to style elements, the network itself should have learned a set of style elements for all different kinds of content patches. Therefore, to visualize all the learned style elements, we use a noise image to approximate the content patch distribution. Our hypothesis is that it should be able to cover all the possible local patch distributions as long as the noise sampling procedure is random and comprehensive enough. Specifically, we feed one large noise image (pixel values are uniformly sampled in  $[0, 255]$ ) into the network and see its corresponding stylization results.

In Figure 9, we provide the stylization results of two noise images for two different styles. It can be seen that our network has learned a set of representative style elements for each style. And for different noise images, the learned style elements are overall similar. This experiment also demonstrates that our method can be potentially used for texture synthesis by feeding noise images.

## 4 CAPABILITIES OF OUR NETWORK

Because of the explicit representation mechanism, our proposed feed-forward network also provides additional capabilities, which may bring new user experiences and generate new stylization effects.

### 4.1 Incremental Training

Some previous style transfer networks (e.g., [5], [6], [3]) have to be retrained for a new style, which is very inconvenient. In contrast, an iterative optimization mechanism [2] 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 [5], [6], [3] and iterative optimization method [2]. We enable an incremental training for new styles, which has comparable learning time to the online-learning method [2], while preserving efficiency of feed-forward networks [5], [6], [3].

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 ~ 40 times compared with previous feed-forward methods.

Figure 10 shows several stylization results of new styles by incremental training. It obtains very comparable stylization results to those from fresh training, which retrain the whole network with the new styles.





Fig. 9. To visualize the learned style elements, we feed different noise images into our network and get their corresponding stylization results. It shows that they contain different types of style elements, and the recovered style elements of different noise images are overall similar.



Fig. 10. Comparison between incremental training (*Left*) and fresh training (*Right*). The target styles are shown on the top-left.



Fig. 11. Results by linear combination of two style filter banks.

## 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  $\{K_i^1, \dots, K_i^m\}$ , we can linearly fuse multiple styles by simply linearly fusing filter banks in the *StyleBank* layer. Next, the fused filter bank is used to convolve with content features  $\{F^1, \dots, F^m\}$ :

$$\tilde{F}^j = \left( \sum_{i=1}^n w_i * K_i^j \right) \otimes F^j \quad \sum_{i=1}^n w_i = 1, j \in \{1, \dots, m\} \quad (6)$$

where  $m$  is the number of styles,  $K_i^j$  is the filter bank of style  $i$  at feature level  $j$ .  $\tilde{F}^j$  is then fed to the decoder. Figure 11 shows such linear fusion results of two different 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 [41], [42]) in our feature space, and  $M_i$  denotes every region mask. The feature maps at level  $j$  can be described as

$F^j = \sum_{i=1}^n (M_i \times F^j)$ . Then region-specific style fusion can be formulated as Equation (7):

$$\tilde{F}^j = \sum_{i=1}^n K_i^j \otimes (M_i^j \times F^j), \quad j \in \{1, \dots, m\} \quad (7)$$

where  $K_i^j$  is the  $i$ -th filter bank at level  $j$ .

Figure 12 shows such two region-specific style fusion results. The left case borrows styles from two famous paintings of Picasso and Van Gogh, while the right two styles are both from Van Gogh. 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.

## 5 EXPERIMENTS

**Training Details.** Our network is trained on 1000 content images randomly sampled from Microsoft COCO dataset [43] and 50 style images (from existing papers and the Internet). Each content image is randomly cropped to  $512 \times 512$ , 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 [44] 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

Though the goal of this paper is to provide better understanding for style transfer but not to achieve state-of-the-art stylization results, we still compare our method with some existing CNN-based style transfer approaches, including optimization based methods [2], Per-Style-Per-Model methods [5], [6], Multi-Style-Per-Model methods [19], [20], [21] and Arbitrary-Style-Per-Model methods [24], [22], [23]. For fair comparison, we directly borrow results from their papers or run their release codes. Because it is difficult to compare results with different abstract stylization, which is indeed controlled by the ratio  $\alpha/\beta$  in Equation (3) and different work may use their own ratios to present results, we choose different  $\alpha, \beta$  in each comparison for comparable perception quality. In the below comparison results, “-S” and “-H” indicate the default single filterbank and hierarchy stylebank setting respectively.

**Compared with the Iterative Optimization Method.** We use  $\alpha/\beta = 1/100$  (in Equation (3)) to produce comparable perceptual



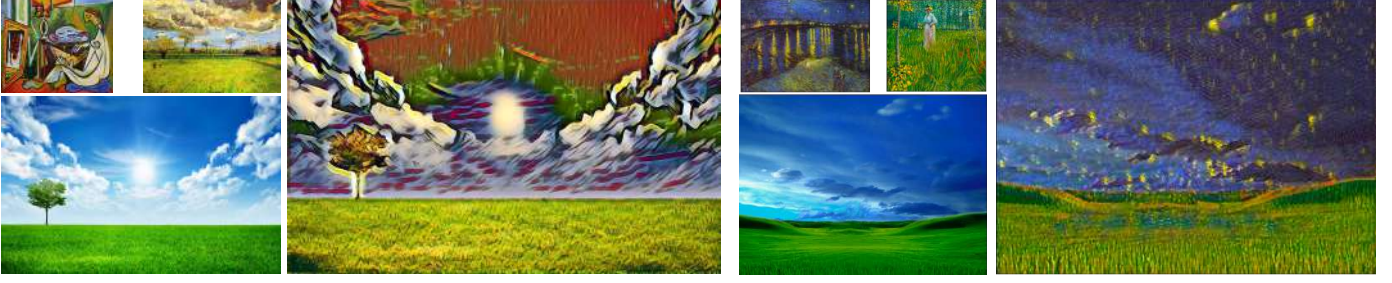


Fig. 12. Region-specific style fusion results with two paintings. The left two paintings are from Picasso and Van Gogh respectively, while the right two are both from Van Gogh. Here the regions are automatically segmented with K-means method.



Fig. 13. Comparison with optimization-based method [2].



Fig. 15. Comparison with the Per-Style-Per-Model method [5].



Fig. 14. Comparison with the Per-Style-Per-Model method [6].

stylization in Fig. 13. Our method, like all other feed-forward methods, creates less abstract stylization results than optimization method [2]. It is still difficult to judge which one is more appealing in practices. However, our method, like other feed-forward methods, could be hundreds of times faster than optimization-based methods.

**Compared with Per-Style-Per-Model methods.** In Figure 14 and Figure 15, we respectively compare our results with two Per-Style-Per-Model methods [5], [6]. We use  $\alpha/\beta = 1/50$  (in Equation (3)) in both comparisons. Ulyanov et al. [6] 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. [5] 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 14, and river/grass/forest in Fig. 15.

**Compared with Multi-Style-Per-Model and Arbitrary-Style-**

**Per-Model methods.** Very recently, many Multi-Style-Per-Model and Arbitrary-Style-Per-Model methods have been proposed. For the former type, existing methods often use one-hot vector [20] or one-dimensional shifting/scaling parameters of instance normalization layer [19] to indicate different styles and jointly train them within one single network. For the latter type, most of them directly transform the content features based on the first and second statistics of style features, then retrain a decoder to decode the transformed content features back to the image space to get the stylization results. In this experiment, we will compare our method with these two different types of methods both qualitatively and quantitatively, including Multi-Style-Per-Model methods [19], [20], [21] and Arbitrary-Style-Per-Model methods [22], [23], [24].

In Figure 17, we directly use the trained models and example images from [45] to generate the stylizations of these baseline methods. It can be seen that the results of different Multi-Style-Per-Model methods [20], [21] are often comparable but may learn different style elements from one specific style image. For Arbitrary-Style-Per-Model methods, though they can handle arbitrary styles, their stylization results are often blurry with distorted content structures and broken style elements. After checking a large number of stylization results, we find our method and Dumoulin et al.'s method [19] are slightly better, but our results are more region-based. More importantly, compared to all of these methods, instead of only enabling multiple or arbitrary styles within one network, we aim to provide an explicit and compact way to represent styles and understand the underlying working principles more clearly.

Besides the above qualitative evaluation, we also conduct an user study for evaluation. Specifically, we first select 10 content images and 10 style images to generate total 100 stylization





Fig. 16. Comparison results of hierarchy (third row) and single level stylebank (second row). It shows that adopting hierarchy stylebank can help to capture the style elements better.

TABLE 2

User study results of different stylization methods. “M” represents Multi-Style-Per-Model methods and “A” represents Arbitrary-Style-Per-Model methods. It shows that our method achieves comparable good stylization results to method [19] and outperforms other methods by a large margin.

Type	method	“best” ratio
M	Dumounlin et al. [19]	41%
M	Li et al. [20]	1%
M	Zhang and Dana [21]	5%
M	<b>Our</b>	<b>44%</b>
A	Chen and Schimdt [22]	3%
A	Huang and Belongie [23]	6%
A	Li et al. [24]	0%

results, then ask 30 users (17 females and 13 males from 18 to 30 years old) to select the best method for each content-style pair with the question “which stylization result is the best?”. As shown in Table 2, our method and Dumoulin et al.’s method have comparable “best” ratios, which are better than other methods. Need to note that since participants are only allowed to choose the best stylization results subjectively, there may exist some uncertainties when the stylization results of some methods are comparable. Besides, we find users are more inclined to choosing the stylization results that look overall bright and colorful, which may be not that fair if some methods like [21] capture the relatively dark style elements even though they are all correct.

## 5.2 Results of Hierarchy Stylebank

In our default simplified design, each style is only represented by one single level filter bank and this filter bank is only convolved with the single level of content features. Compared to this single level setting, the proposed hierarchy setting may be able to capture the texture elements better. To verify this point, we also train this hierarchy setting with the same style images set and compare the final stylization results. In Figure 16, two example stylization results are given. It can be seen that utilizing the hierarchy filter bank is very helpful to capture different scales of texture elements and produce better stylization results.

## 6 EXTENSION TO IMAGE PROCESSING TASKS

Though our method is motivated for style transfer task, the proposed filterbank learning idea is essentially a natural and general way to achieve multiple functionalities/tasks within one single network. To demonstrate its generalization ability, we further apply this idea to two different multi-parameter image processing tasks: image smoothing, and denoising. In these two tasks, there exist some parameters to control the final results (e.g., smoothness degree, noise level). With the development of deep learning, many methods have used neural networks to accelerate or improve these tasks. However, some of them still train multiple different models for different parameter settings, which is both time-consuming and storage-consuming. In the following part, we will try to use different filter banks for different parameter settings and enables multiple settings within one single network. Because we do not need to decouple the content and style like the above style transfer task, the auto-encoder branch is disabled in this experiment.

### 6.1 Image Smoothing

Image smoothing has been a very active and fundamental problem in the computer vision field, which can be used for unwarranted texture removal and image beautification in real applications. In the past decades, a lot of edge-aware image smoothing algorithms have been proposed, like  $L_0$  smoothing [46], RGF [47] and RTV [48]. To obtain satisfactory results, these methods often provide some hyper-parameters to tweak the final effects. Despite of their impressive results, these methods are often based on a time-consuming optimization procedure. To accelerate these algorithms, some deep neural networks [49], [50] are proposed to approximate them. However, some of these methods are designed to approximate the effects of one specific hyper-parameter setting or one specific algorithm. Therefore, for different parameter settings or algorithms, multiple different models need to be retrained.

Considering that different smoothing algorithms or parameter settings may share some common low-level or high-level image statistics, we try to use different filter banks to represent different algorithms and parameter settings. Without losing generality, we consider three famous edge-aware smoothing algorithms ( $L_0$ [46], RGF[47], RTV[48]) and five parameter settings for each algorithm in one single network, thus using total 15 filter banks.

By default, we directly use the backbone network structure proposed in [51] but remove the weight learning sub network. To support multiple algorithms/parameter settings, we try two different filter bank settings: the default single filter bank setting and the hierarchy filter bank setting. Following the same training strategy as [49], [50], [51], we first use PASCAL VOC dataset [52] to pre-generate the training image pairs by running the original algorithms, then train the network with these image pairs in a supervised way. The total image number is about 17k, and 500 images are randomly selected as the test set.

In Table 3, we first compare our method with the default single algorithm and parameter setting baseline without the extra filterbank layer (“baseline”). To further avoid the performance gain from the extra parameters in the filterbank layer, we further test another baseline (“baseline\*”) with one filterbank layer but training for one single setting. On one hand, it can be seen that using the proposed filter bank learning idea can achieve slightly better or at least very comparable performance to the single setting baseline in terms of PSNR and SSIM. On the other hand, the hierarchy filter bank is better than the single version. The underlying reason



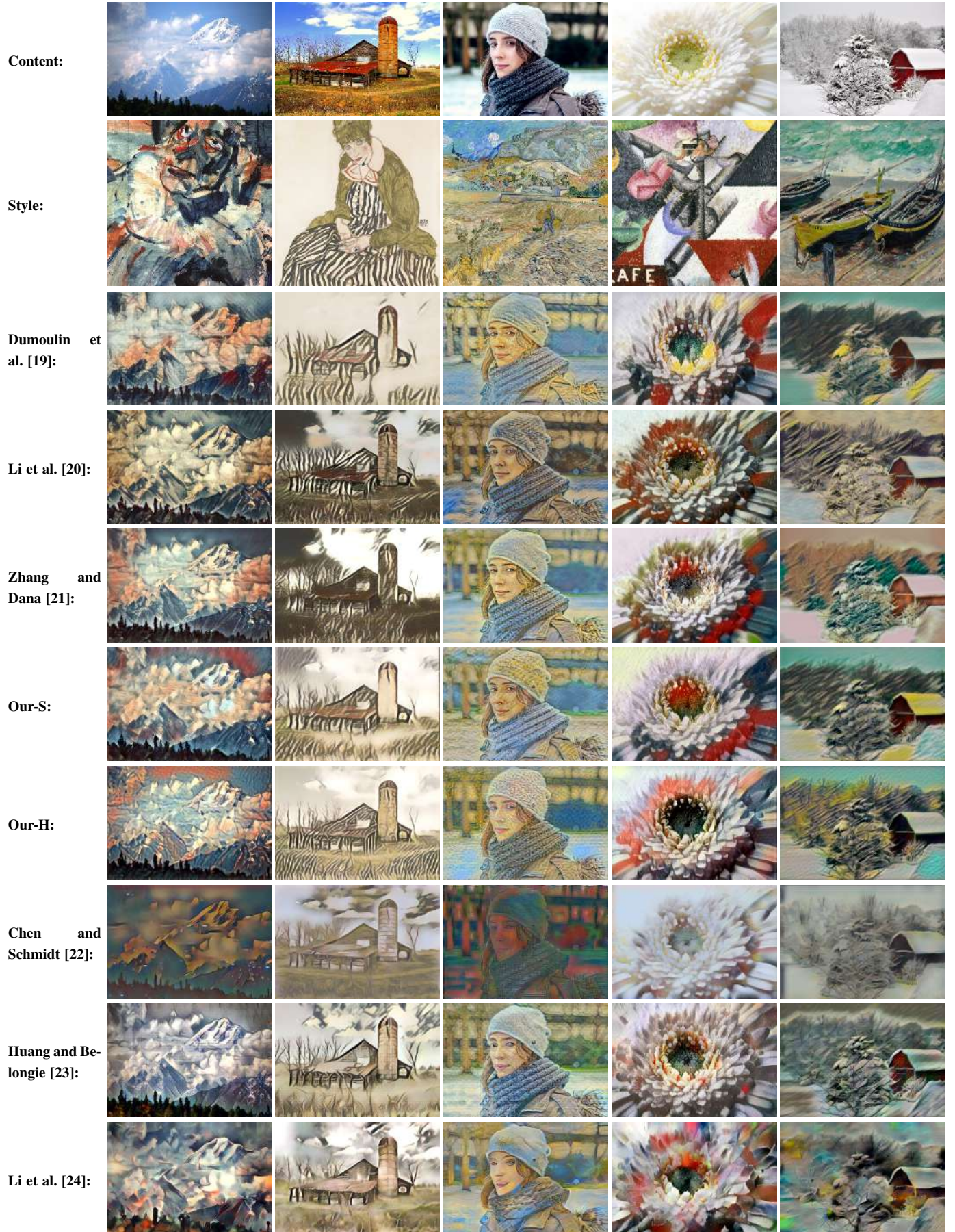


Fig. 17. Some visual comparison results of different methods: the above four methods are Multi-Style-Per-Model based, while the below three are Arbitrary-Style-Per-Model based methods. It can be seen that the results from Li et al. [20] and Zhang and Dana [21] often suffer from color drift problems, and all Arbitrary-Style-Per-Model methods will generate broken and blurry texture elements. By comparison, we find that Dumoulin et al.'s method [19] and our method can generate visually comparable results, but our results are more region-based.



may be that hierarchy filter bank uses both low-level and high-level image information, which is important for edge-aware image smoothing algorithms. Some visual examples with running time details are given in Figure 19, which further demonstrates that our method is able to achieve very good smoothing effects for different algorithms and hyper-parameters. Besides, compared to the running time of obtaining the groundtruth results, our method is significantly faster.

## 6.2 Image Denoising

Since the realistic image capturing environment is not perfect, the captured images are often degraded with some noises. In the past, many different image denoising algorithms are proposed by leveraging some image priors [53], [54] or large-scale training datasets [56], [55]. However, some of the deep neural networks based methods [56], [55] still train one network for one specific noise level. In this paper, though we also only consider additive white Gaussian noise (AWGN), we use different filter banks to represent different noise levels and train one single network for multiple different noise levels.

Following the same training and evaluation strategy as previous methods, we consider gray image denoising of three different noise levels ( $\sigma = 15, 25, 50$ ) and evaluate the final performance on the BSD68 dataset. To train the network, we also use the aforementioned PASCAL VOC dataset to pre-generate three different noise levels of image pairs.

Table 4 shows the detailed evaluation results on the BSD68 dataset. By using the proposed filter bank learning idea, our method is not only able to handle three different noise levels within one single network but also achieves very close results to the single setting baselines (“baseline\*” and “baseline” are with and without the extra filterbank layer respectively). However, different from the above edge-aware smoothing, the performance of “hierarchy filter bank” is almost same as the “single filter bank” version. The possible underlying reason is that AWGN is relatively local and does not require too much hierarchy image information, so the single filter bank setting is enough. We also provide the results of some previous state-of-the-art methods. Among them, BM3D [53] and WNNM [54] are optimization based methods, while DCDP [55] and DnCNN [56] are learning based methods. Surprisingly, our method is even better than these four methods, which may attribute to the better network structure.

We further show two visual comparison results in Figure 18. These two examples are with noise level 25 and 50 respectively. It shows that our method can produce good denoising results for different noise levels, which are even better than some previous state-of-the-art methods. Especially in the second example, we can find that, though the ground texture is very similar as the noise pattern, our method can still differentiate them well and preserve the original ground texture while removing the noises.

## 7 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 provide very detailed analysis and present a new interpretation to neutral style transfer, which may inspire more understandings rather than leave the network as a blackbox. To demonstrate the generalization

ability of the proposed idea in supporting multiple functionalities within one single network, we further extend this idea to image smoothing and image denoising task. Experiments demonstrate that the proposed idea not only supports multiple algorithms or parameter settings within one single network but also achieves comparable results to their single parameter counterparts.

Though this is an extension of our preliminary work [57], there are still some interesting problems for further investigation. For example, the auto-encoder may integrate semantic segmentation [58], [59] as additional supervision in the region decomposition, which would help create more impressive region-specific transfer. It is also interesting to incorporate the proposed technique into video and stereoscopic applications [60], [61].

**Acknowledgement.** This work was supported partly by National Key R&D Program of China Grant 2018AAA0101400, and NSFC Grant 61629301, U163620.

## REFERENCES

- [1] L. Gatys, A. S. Ecker, and M. Bethge, “Texture synthesis using convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 262–270.
- [2] L. A. Gatys, A. S. Ecker, and M. Bethge, “A neural algorithm of artistic style,” *arXiv preprint arXiv:1508.06576*, 2015.
- [3] C. Li and M. Wand, “Combining markov random fields and convolutional neural networks for image synthesis,” *arXiv preprint arXiv:1601.04589*, 2016.
- [4] J. Liao, Y. Yao, L. Yuan, G. Hua, and S. B. Kang, “Visual attribute transfer through deep image analogy,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 120, 2017.
- [5] J. Johnson, A. Alahi, and L. Fei-Fei, “Perceptual losses for real-time style transfer and super-resolution,” *arXiv preprint arXiv:1603.08155*, 2016.
- [6] 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.
- [7] C. Li and M. Wand, “Precomputed real-time texture synthesis with markovian generative adversarial networks,” *arXiv preprint arXiv:1604.04382*, 2016.
- [8] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [9] A. A. Efros and T. K. Leung, “Texture synthesis by non-parametric sampling,” in *Proceedings of the 7th IEEE International Conference on Computer Vision*, vol. 2. IEEE, 1999, pp. 1033–1038.
- [10] 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*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 479–488.
- [11] 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*. ACM, 2001, pp. 341–346.
- [12] 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)*, vol. 20, no. 3, pp. 127–150, 2001.
- [13] 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.
- [14] M. Elad and P. Milanfar, “Style-transfer via texture-synthesis,” *arXiv preprint arXiv:1609.03057*, 2016.
- [15] 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*. ACM, 2001, pp. 327–340.
- [16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
- [17] M. T. Alexander Mordvintsev, Christopher Olah. (2015) Inceptionism: Going deeper into neural networks. [Online]. Available: <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>
- [18] A. Selim, M. Elgharib, and L. Doyle, “Painting style transfer for head portraits using convolutional neural networks,” *ACM Transactions on Graphics (ToG)*, vol. 35, no. 4, p. 129, 2016.



TABLE 3

Quantitative comparison results of using the proposed filter bank learning idea for edge-aware smoothing task. Total 15 different settings (three different algorithms and five hyper-parameter settings for each algorithm) are tested, where each setting is represented with one filter bank. “our-s” means the results of using single filter bank, while “our-h” means the results of using hierarchy filter banks. It can be seen that our method can achieve slightly better or at least comparable results to the single parameter setting baseline and the hierarchy version is also better than the single version. Here “baseline\*” and “baseline” mean the results with and without the extra filterbank layer in baseline network respectively.

		$L_0$					RGF					RTV				
		$\gamma$	baseline	baseline*	our-s	our-h	$\gamma$	baseline	baseline*	our-s	our-h	$\gamma$	baseline	baseline*	our-s	our-h
PSNR	0.002	40.51	40.18	40.99	41.22	1	41.40	41.55	41.26	41.62	0.005	41.59	41.05	41.48	41.61	
	0.005	38.91	38.63	39.16	39.43	3	38.75	38.84	38.59	38.67	0.01	41.25	41.41	41.32	41.44	
	0.01	37.54	37.60	37.52	37.85	5	38.39	38.40	38.02	38.11	0.02	41.59	41.64	40.94	41.12	
	0.05	34.30	34.09	33.91	34.23	7	37.87	38.14	37.37	37.48	0.03	41.37	41.35	40.42	40.61	
	0.10	32.15	32.49	31.94	32.24	9	37.21	37.29	36.62	36.70	0.05	40.91	40.90	39.11	39.34	
	ave.	36.68	36.60	36.70	36.99	ave.	38.72	38.84	38.37	38.52	ave.	41.34	41.27	40.65	40.82	
SSIM	0.002	0.986	0.985	0.991	0.992	1	0.994	0.993	0.994	0.995	0.005	0.990	0.989	0.992	0.993	
	0.005	0.985	0.984	0.989	0.990	3	0.987	0.987	0.989	0.990	0.01	0.990	0.991	0.993	0.993	
	0.01	0.983	0.984	0.986	0.987	5	0.986	0.985	0.986	0.987	0.02	0.992	0.992	0.992	0.993	
	0.05	0.979	0.979	0.975	0.977	7	0.984	0.985	0.984	0.985	0.03	0.992	0.992	0.992	0.992	
	0.10	0.973	0.976	0.966	0.968	9	0.982	0.983	0.980	0.981	0.05	0.992	0.992	0.990	0.991	
	ave.	0.981	0.982	0.981	0.983	ave.	0.987	0.987	0.987	0.988	ave.	0.991	0.991	0.992	0.992	

TABLE 4

Quantitative comparison results of using the proposed filter bank learning idea for the gray image denoising task on the BSD68 dataset. Three different noise levels ( $\sigma = 15, 25, 50$ ) are trained within one single network, where each noise level is represented with one filter bank. It shows that our single filter bank (“our-s”) and hierarchy filter bank setting (“our-h”) can achieve almost same results with the single setting baseline, which are even better than some state-of-the-art methods. Here “baseline\*” and “baseline” mean the results with and without the extra filterbank layer in baseline network respectively.

PSNR/SSIM	BM3D[53]	WNNM [54]	DCDP [55]	DnCNN [56]	baseline	baseline*	our-s	our-h
15	31.07 / 0.8707	31.37 / 0.8759	31.63 / 0.8869	31.60 / 0.8847	31.67 / 0.8905	31.66 / 0.8905	31.64 / 0.8891	31.65 / 0.8891
25	28.57 / 0.7967	28.83 / 0.8028	29.15 / 0.8188	29.15 / 0.8132	29.22 / 0.8295	29.22 / 0.8293	29.21 / 0.8284	29.21 / 0.8284
50	25.62 / 0.6662	25.87 / 0.6635	26.19 / 0.6870	26.21 / 0.6870	26.32 / 0.7261	26.33 / 0.7260	26.30 / 0.7245	26.30 / 0.7254

- [19] V. Dumoulin, J. Shlens, and M. Kudlur, “A learned representation for artistic style,” *Proc. of ICLR*, vol. 2, 2017.
- [20] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, “Diversified texture synthesis with feed-forward networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3920–3928.
- [21] H. Zhang and K. Dana, “Multi-style generative network for real-time transfer,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 0–0.
- [22] T. Q. Chen and M. Schmidt, “Fast patch-based style transfer of arbitrary style,” *arXiv preprint arXiv:1612.04337*, 2016.
- [23] X. Huang and S. Belongie, “Arbitrary style transfer in real-time with adaptive instance normalization,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1501–1510.
- [24] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, “Universal style transfer via feature transforms,” in *Advances in neural information processing systems*, 2017, pp. 386–396.
- [25] 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*, vol. 2. IEEE, 1999, pp. 918–925.
- [26] S.-C. Zhu, C.-E. Guo, Y. Wang, and Z. Xu, “What are textons?” *International Journal of Computer Vision*, vol. 62, no. 1-2, pp. 121–143, 2005.
- [27] 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)*, vol. 21, no. 3. ACM, 2002, pp. 465–472.
- [28] 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*, vol. 27, no. 2, pp. 107–126, 1998.
- [29] Y. Lu, S.-C. Zhu, and Y. N. Wu, “Learning frame models using cnn filters,” *arXiv preprint arXiv:1509.08379*, 2015.
- [30] G. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504 – 507, 2006.
- [31] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, “A neural probabilistic language model,” *Journal of Machine Learning Research*, vol. 3, no. Feb, pp. 1137–1155, 2003.
- [32] S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee, “Deep visual analogy-making,” in *Advances in Neural Information Processing Systems* 28, 2015, pp. 1252–1260.
- [33] 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.
- [34] J. Portilla and E. P. Simoncelli, “A parametric texture model based on joint statistics of complex wavelet coefficients,” *International Journal of Computer Vision*, vol. 40, no. 1, pp. 49–70, 2000.
- [35] 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*. ACM, 1995, pp. 229–238.
- [36] D. Ulyanov, A. Vedaldi, and V. Lempitsky, “Instance normalization: The missing ingredient for fast stylization,” *arXiv preprint arXiv:1607.08022*, 2016.
- [37] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
- [38] H. A. Aly and E. Dubois, “Image up-sampling using total-variation regularization with a new observation model,” *IEEE Transactions on Image Processing*, vol. 14, no. 10, pp. 1647–1659, 2005.
- [39] 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*, 2014, pp. 2672–2680.
- [40] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

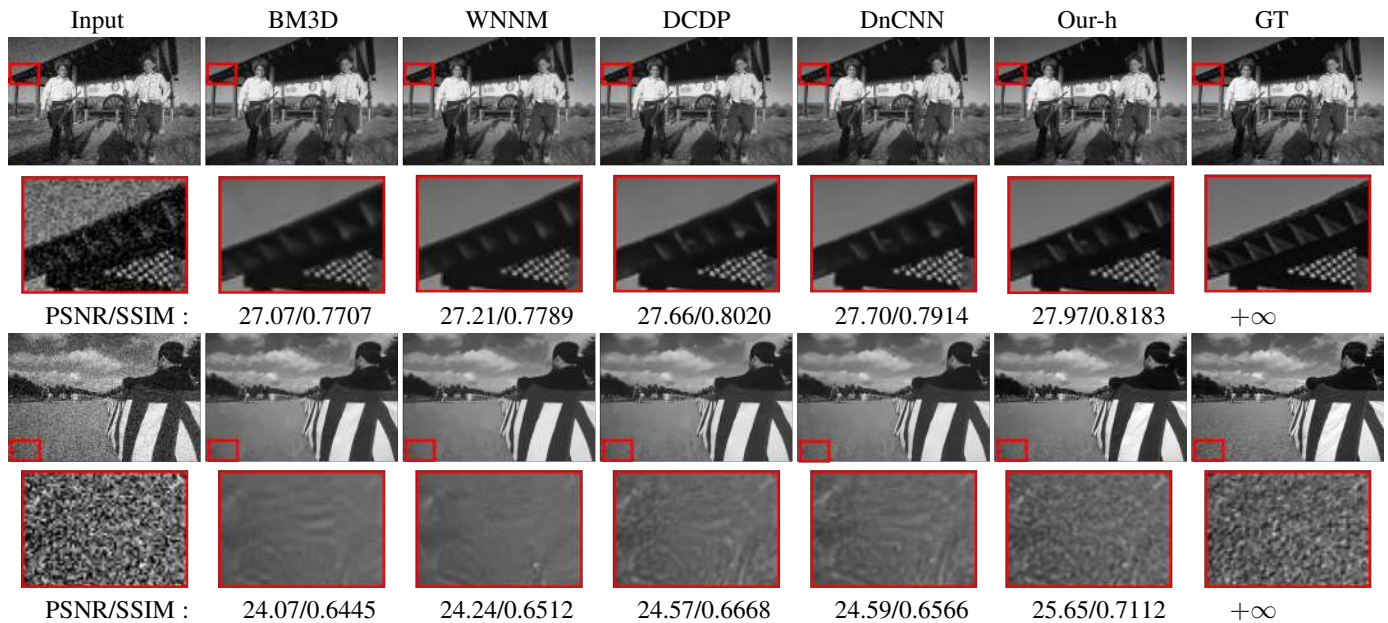


Fig. 18. Visual results of using the proposed filter bank learning idea for multiple noise level image denoising. The two cases are with noise level 25 and 50 respectively. It shows that our method can achieve very good denoising results of different noise levels, which even outperforms some famous state-of-the-art methods.

- [41] 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*, vol. 1. IEEE, 2001, pp. 105–112.
- [42] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," in *ACM transactions on graphics (ToG)*, vol. 23, no. 3. ACM, 2004, pp. 309–314.
- [43] 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*. Springer, 2014, pp. 740–755.
- [44] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [45] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *arXiv preprint arXiv:1705.04058*, 2017.
- [46] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via l0 gradient minimization," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6. ACM, 2011, p. 174.
- [47] Q. Zhang, X. Shen, L. Xu, and J. Jia, "Rolling guidance filter," in *European Conference on Computer Vision*. Springer, 2014, pp. 815–830.
- [48] L. Xu, Q. Yan, Y. Xia, and J. Jia, "Structure extraction from texture via relative total variation," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, p. 139, 2012.
- [49] L. Xu, J. Ren, Q. Yan, R. Liao, and J. Jia, "Deep edge-aware filters," in *International Conference on Machine Learning*, 2015, pp. 1669–1678.
- [50] Q. Fan, J. Yang, G. Hua, B. Chen, and D. Wipf, "A generic deep architecture for single image reflection removal and image smoothing," in *Proceedings of the 16th International Conference on Computer Vision (ICCV)*, 2017, pp. 3238–3247.
- [51] Q. Fan, D. Chen, L. Yuan, G. Hua, N. Yu, and B. Chen, "Decouple learning for parameterized image operators," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 442–458.
- [52] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," 2012.
- [53] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *TIP*, 2007.
- [54] S. Gu, L. Zhang, W. Zuo, and X. Feng, "Weighted nuclear norm minimization with application to image denoising," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 2862–2869.
- [55] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep cnn denoiser prior for image restoration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3929–3938.
- [56] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.
- [57] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua, "Stylebank: An explicit representation for neural image style transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1897–1906.
- [58] 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*, 2015, pp. 3431–3440.
- [59] 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.
- [60] D. Chen, J. Liao, L. Yuan, N. Yu, and G. Hua, "Coherent online video style transfer," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1105–1114.
- [61] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua, "Stereoscopic neural style transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6654–6663.

























































$\gamma$	input	0.002	0.005	0.010	0.050	0.100
$L_0$ -GT						
Time:		0.6973s	0.7351s	0.6777s	0.5759s	0.5388s
$L_0$ -baseline						
PSNR/Time:		40.13/0.0050s	38.28/0.0050s	36.54/0.0050s	33.59/0.0050s	30.94/0.0050s
$L_0$ -Our-h						
PSNR/Time:		40.82/0.0050s	38.93/0.0050s	37.04/0.0050s	34.21/0.0050s	31.04/0.0050s
$\gamma$	input	1	3	5	7	9
RGF-GT						
Time:		0.9465s	1.11s	1.40s	1.64s	1.95s
RGF-baseline						
PSNR/Time:		38.73/0.0050s	35.49/0.0050s	34.71/0.0050s	33.72/0.0050s	32.14/0.0050s
RGF-Our-h						
PSNR/Time:		38.38/0.0050s	35.10/0.0050s	34.28/0.0050s	33.16/0.0050s	31.77/0.0050s
$\gamma$	input	0.005	0.010	0.020	0.030	0.050
RTV-GT						
Time:		3.04s	0.40s	0.24s	0.21s	0.20s
RTV-baseline						
PSNR/Time:		43.30/0.0050s	42.76/0.0050s	43.32/0.0050s	42.83/0.0050s	42.39/0.0050s
RTV-Our-h						
PSNR/Time:		43.53/0.0050s	42.92/0.0050s	42.49/0.0050s	41.86/0.0050s	40.41/0.0050s

Fig. 19. Visual results of using the proposed filter bank learning idea for edge-aware smoothing task. It shows that our method can achieve very good smoothing effects for different algorithms and hyper-parameters while being much faster.