Controllable Image Processing via Adaptive FilterBank Pyramid

Dongdong Chen, Qingnan Fan, Jing Liao, Angelica Aviles-Rivero, Lu Yuan, Nenghai Yu, Gang Hua, *Fellow, IEEE*

***Abstract*—Traditional image processing operators often provide some control parameters to tweak the final results. Recently, different convolutional neural networks have been used to approximate or improve these operators. However, in those methods, one single model can only handle one operator of a specific parameter value and does not support parameter tuning. In this paper, we propose a new plugin module, *"Adaptive Filterbank Pyramid"*, which can be inserted into a backbone network to support multiple operators and continuous parameter tuning. Our module explicitly represents one operator with one filterbank pyramid. To generate the results of a specific operator, the corresponding filterbank pyramid is convolved with the intermediate feature pyramid produced by the backbone network. The weights of the filterbank pyramid are directly regressed by another sub-network, which is jointly trained with the backbone network and adapted to the input parameter, thus enabling continuous parameter tuning. We applied the proposed module for a large variety of image processing tasks, including image smoothing, image denoising, image deblocking, image enhancement and neural style transfer. Experiments show that our method is generalized to different types of image processing tasks and different backbone network structures. Compared to the single-operator-single-parameter baseline, our method can produce comparable results but is significantly more efficient in both training and testing.**

I. Introduction

Image processing is an important and fundamental research field in computer vision, which includes lots of tasks (e.g. image smoothing, restoration, stylization). Over the last decades, many different algorithms (referred as "operators") have been proposed for these tasks, such as L_0 smoothing [26], BM3D [7] and neural style transfer [15], [3]. Most of these operators provide some control parameters to tweak the final results. For example, a hyper-parameter Θ is used to control the smoothing degree in [26], and the style weight in [15] determines how much the input image should be stylized. Tuning these control parameters are especially useful for users to get their desired results in real applications.

Dongdong Chen and Nenghai Yu are with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei, Anhui 230026, China. E-mail:cddlyf@gmail.com, ynh@ustc.edu.cn

Qingnan Fan is with the Computer Science Department, Stanford University, Stanford, California 94305, US. Email: fqnchina@gmail.com

Jing Liao is with the Department of Computer Science, City University of Hong Kong. Email: jingliao@cityu.edu.hk

Angelica I. Aviles-Rivero is with the DAMTP and DPMMS, University of Cambridge, Email: ai323@cam.ac.uk

Lu Yuan is with Microsoft Cloud AI, Redmond, Washington 98052, USA. Email: luyuan@microsoft.com

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.

Despite their success, some traditional operators are very time-consuming and the results of some operators are still not good enough because only low-level statistics are used. Inspired by the tremendous success of deep learning techniques, some recent works [4], [27], [13], [22] have investigated deep neural networks (DNNs) to accelerate or improve these traditional operators. These methods often pre-generate a large set of training image pairs by running a traditional operator with specific parameters, then train a network to approximate this operator quickly as a regression problem. Thanks to the powerful learning capacity of deep neural networks, huge improvements have been achieved by these methods. However, without a good adaptive mechanism, they often have two important limitations for real applications.

The first limitation is that existing models do not support parameter tuning (parameter-specific). For different parameter values of an operator, separated models need to be retrained, which is very time and storage consuming. Even worse, these control parameters are often continuous and users may want to tweak them to achieve satisfactory results, e.g., tuning the stylization degree in style transfer. However, it is unrealistic to train numerous models for densely sampled parameter values. The underlying reason for this limitation is that current vanilla DNNs do not have good adaptive mechanisms, which means that their weights are fixed once trained. Thus during runtime, given an input image, only one processed result can be obtained.

The second limitation is that one model can only handle one specific operator (operator-specific), which means different models need to be retrained for different operators. Take the style transfer as an example, only one style is learned in one network. Considering the limited storage budget in real applications, this limitation is a fatal obstacle. However, different operators within the same class (e.g. different smoothing operators, or different styles in neural style transfer) often leverage some common image statistics (e.g. edges, semantic information). Thus, it is possible to train them in one single network without quality degradation.

Inspired by the "Stylebank" idea in [4] and the decouple learning mechanism in [12], this paper proposes a new plugin module *"Adaptive Filterbank Pyramid"* to address the above limitations. This plugin module can be inserted into the backbone network to support multiple operators and continuous parameter tuning. Specifically, as shown in Figure 1, we use a shared encoder and decoder for all operators and explicitly represent each operator with a *filterbank pyramid*. During runtime, to generate the results of one specific operator, its corresponding filterbank pyramid is convolved with the inter-

mediate feature pyramid produced by the backbone encoder.

To support parameter tuning, the weights of the filterbank pyramid will be directly regressed with another weight sub-network, whose input is the control parameters. In this way, when users select different control parameter values, the weight sub-network will dynamically change the weights of the filterbank pyramid to produce different processing results. Another bonus is that the intermediate feature pyramids can be reused when tuning parameters for one operator, which can save about half of the computation cost.

To demonstrate the effectiveness of our proposed *Adaptive FilterBank Pyramid*, we first implement our idea to approximate different image smoothing operators simultaneously within one single network, which includes bilateral filter [24], L_0 smoothing [26], relative total variation (RTV) regularization [28], rolling guidance filter (RGF) [31] and weighted median filter (WMF) [32]. Some of these smoothing operators are very slow because of their optimization procedure. Compared to the single-operator-single-parameter baseline, our method can obtain comparable results both qualitatively and quantitatively, but ours is significantly more efficient in both training and testing by supporting multiple different image operators in a single model. As a plugin module, we also demonstrate the proposed *Adaptive FilterBank Pyramid* can be easily inserted into different backbone auto-encoder networks.

To show the generalization capabilities of our approach to other image processing tasks that need parameter tunning, we further apply our method to image denoising, image deblocking, image enhancement [14], and neural style transfer [4], [15]. Experiments show that the proposed *Adaptive Filterbank Pyramid* is very flexible and generalizable, which can handle a wide range of applications.

Overall, our contributions are three-fold:

- We propose a new module, call *Adaptive FilterBank Pyramid*, to explicitly represent different image processing operators and support multi-operator training in one single model.
- The weights of the proposed filterbank pyramid are designed to be the output of another weight sub-network. By feeding different parameter values into this sub-network, the proposed filterbank pyramid can dynamically adjust its behavior and enable real-time continuous parameter tuning.
- We demonstrate the generality and effectiveness of the proposed method with different backbone network structures and a large variety of image processing tasks. Experiments show that it can generate comparable results to the single-operator-single-parameter baseline, but is significantly more efficient.

II. Related Work

A. Traditional Image Processing Operators

Image processing has been an active and fundamental research field in computer vision for a very long time. It includes different types of processing tasks, such as image smoothing, image enhancement, and image restoration. By considering different image priors under these tasks, lots of image processing operators have been proposed. For example, different edge-preserving image smoothing techniques are well-studied in [10], [14], [17], [24]. Spatial relationship and redundancy are explored in [1], [7] for image denoising, [14] uses a weighted least squares optimization framework for progressive coarsening of images and multi-scale detail extraction. To obtain a satisfactory result, a large part of these operators provide some controllable hyper-parameters, e.g. the smoothing parameter to control the final smoothness in [26]. Supporting continuous parameter tuning is one focus of our method in this paper, and generalization ability to different types of tasks is another focus.

B. Deep-Learning-based Approaches

Recently, deep-learning-based approaches have achieved great success in both image recognition tasks [20], [18] and image generation tasks [8], [16]. Since many traditional operators are based on a time-consuming optimization procedure and only use some low-level image statistics, different deep networks [27], [13], [22], [30], [2] have been proposed to accelerate or improve them. Considering the underlying task-specific priors, many useful strategies have been incorporated in the network design to improve the performance, like dilated convolution [29] to increase the receptive field or feeding the edge map into the network as the extra auxiliary information [13]. Despite their success in terms of quality and computational time, their networks are designed to train one model for a single image operator with a specific parameter value.

To support multiple operators and parameters in the deep network, as pointed out in [3], a naive naive idea is to add extra input channels to indicate different filters and parameters, and then let the network to learn them as a black-box. Compared to this perspective, our *Adaptive FilterBank Pyramid* is a more explicit and explainable representation, which can help to achieve better results and save testing time. To allow continuous parameter tuning, Fan et al. [12] propose a decouple learning algorithm to learn from the operator parameters and dynamically adjust the network weights for the image operator. Our adaptive weights idea is inspired by this idea. However, unlike [10], where all the convolutional weights are learned, we only learn the weights of filterbank pyramid. And the backbone auto-encoder is shared, so it is more efficient and storage saving. Compared to [5] and [12], the proposed *Adaptive FilterBank Pyramid* is a novel plugin module to support both multi-operator training and continuous parameter tuning.

C. Multi-style transfer with Stylebank learning.

Our method is also related to the multi-style transfer method [4], which represents each style with one stylebank kernel. However, our method differs from [4] in three aspects: 1) we aim to propose a general plugin module for different types of image processing tasks including style transfer. 2) Only a single-level convolution kernel is used in [4], but our filterbank pyramid is designed to convolve with different levels of features. This is important to many image processing tasks (e.g. edge-aware filtering), which often utilizes both low-level

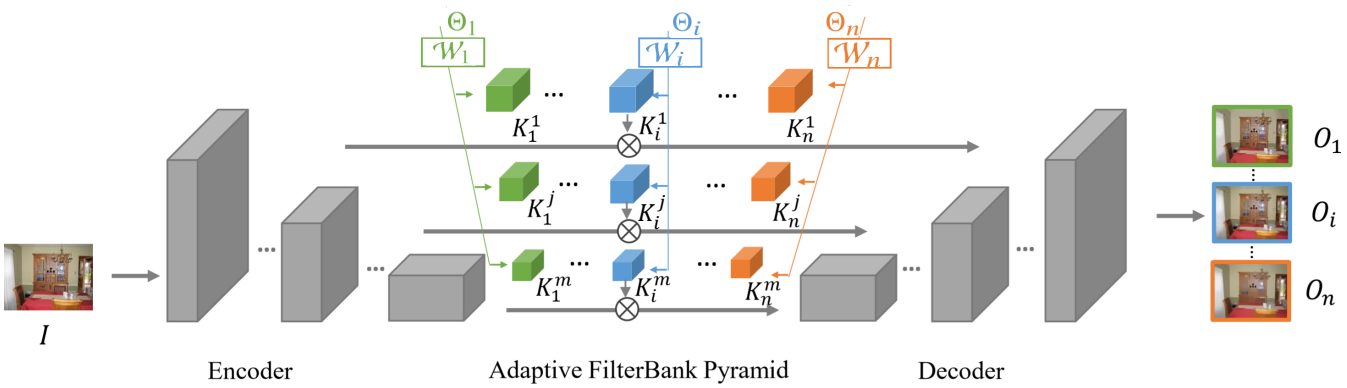


Fig. 1. Our network architecture consists of three modules: an encoder, the new proposed *Adaptive FilterBank Pyramid* and a decoder. Each image operator is represented by its corresponding filter bank pyramid, which is convolved with different level intermediate feature maps. To enable continuous parameter tuning, the weights of each filter bank pyramid are the output of another weight sub-network $\mathcal{W}_i$ and will be adaptively changed during runtime by feeding different Θ_i.

edge features and high-level semantic information. 3) The stylebank [4] is fixed once trained for a specific style, and no parameter tuning is allowed to change the stylization results. In the experiment part, we show that our method can generalize well to neural style transfer while enabling more capabilities.

III. Method

A. Overview of Our Method

Given an input image I, a specific image operator $\mathcal{H}_i$, and the corresponding tunable parameters Θ_i for $\mathcal{H}_i$, the processed result is denoted as $O_i = \mathcal{H}_i(I, \Theta_i)$. For different $\mathcal{H}_i$, Θ_i could be a single variable or a vector containing multiple variables, which controls a desirable effect such as smoothness strength of the filter, stylization degree or stroke size in neural style transfer. Unlike most previous methods which use a network $\mathcal{N}$ to approximate one specific filter $\mathcal{H}_i$ with one specific Θ_i^{const}, we aim to design a new plugin module, which can be inserted into a backbone network $\mathcal{N}$ to support multiple image operators $\mathcal{H}_1, ..., \mathcal{H}_n$ within a single model and enable tuning $\Theta_1, ..., \Theta_n$ continuously during runtime.

To achieve that, we propose the *"Adaptive FilterBank Pyramid"* $\mathcal{K}_1, ..., \mathcal{K}_n$ to represent different image operators while all these operators share a common base network $\mathcal{N}$, then the processed results of operator $\mathcal{H}_i$ are represented as:

$$\begin{aligned} O_i &= \mathcal{N}(\mathcal{K}_i, \Theta_i, I) \\ \mathcal{K}_i &= f(\Theta_i) \end{aligned} \tag{1}$$

To enable parameter tuning, we model the weights of $\mathcal{K}_i$ as the function of Θ_i. Moreover, we use another weight sub-network $\mathcal{W}_i$ to directly learn the specific formulation of f rather than design it in the handcrafted manner, *i.e.*, $f = \mathcal{W}_i$. During runtime, by feeding different input parameters Θ_i to $\mathcal{W}_i$, the weights of its corresponding filterbank pyramid $\mathcal{K}_i$ will be dynamically adjusted and change the functionality of $\mathcal{N}$ to generate different processed results corresponding to Θ_i.

B. Details of Network Structure

Our overall network structure is shown in Figure 1, which consists of three modules: an encoder $\mathcal{E}$, the proposed *Adaptive FilterBank Pyramid* set $\{\mathcal{K}_1, ..., \mathcal{K}_n\}$ for different filters, and a decoder $\mathcal{D}$. Given an input image I, the encoder $\mathcal{E}$ first encodes I into multi-level feature maps pyramid denoted as $\mathcal{F} = (F^1, ..., F^m)$, where F^l is the feature at level l. Then, we select the adaptive filterBank pyramid $\mathcal{K}_i = (K_i^1, ..., K_i^m)$ corresponding to the specific image operator $\mathcal{H}_i$ to convolve with the feature maps pyramid, finally the transformed feature pyramid is fed into the decoder to get the final processing result O_i. As described before, the weights of the adaptive filterBank pyramid $\mathcal{K}_i$ are designed as the output of another weight sub-network $\mathcal{W}_i$, which are adapted to the input parameter Θ_i.

a) Encoder and Decoder: As a flexible plugin, the proposed *Adaptive Filterbank pyramid* can be used in different types of backbone networks if they follow auto-encoder like structures. To apply the proposed filterbank pyramid to intermediate feature maps, given a specific backbone network $\mathcal{N}$, we split it into an encoder $\mathcal{E}$ and decoder $\mathcal{D}$. In this paper, we adopt a similar backbone network $\mathcal{N}$ as [12] by default, which consists of a total of 20 convolutional layers with 3x3 kernel size. Considering the effectiveness of residual learning, the intermediate 14 layers are formed as 7 residual blocks. The first three layers are vanilla convolutional layers that downsample the dimension of the feature maps by 1/2 to increase the receptive field and save intermediate computation cost. Symmetrically, the third-to-last layer is a stride-$\frac{1}{2}$ fractionally strided convolution layer to upsample the downsampled feature maps to the original image resolution, followed by two convolution layers. Except for the last convolution layer, we put an instance normalization layer [25] and ReLU layer after all the former convolutional layers. To further increase the receptive field, different increased dilation factors [29] are used in the convolution layers. Empirically, the dilation factors are set as $(2, 4, 4, 8, 8, 16, 1)$ for the 7 residual blocks respectively.

To split the whole network $\mathcal{N}$ as encoder $\mathcal{E}$ and decoder $\mathcal{D}$, we regard all the layers before the fourth residual block as the encoder and the remaining layers as the decoder. Detailed network configurations is given in Table I. As both the low-level features like edges and high-level features are important to most image processing tasks (e.g. edge-aware image

TABLE I
THE DETAILED ENCODER AND DECODER CONFIGURATION USED FOR THE DEFAULT AUTO-ENCODER LIKE BACKBONE STRUCTURE. THE PREFIX $\mathcal{E}-$ AND $\mathcal{D}-$ REPRESENT TO WHICH PART EACH MODULE BELONGS.

Layer type	channel	dilation	kernel	stride	repeat
$\mathcal{E}-$Conv	64	1	3×3	1	2
$\mathcal{E}-$Conv	64	1	3×3	2	1
$\mathcal{E}-$Residual block	64	2	3×3	1	1
$\mathcal{E}-$Residual block	64	4	3×3	1	2
$\mathcal{E}-$Residual block	64	8	3×3	1	1
$\mathcal{D}-$Residual block	64	8	3×3	1	1
$\mathcal{D}-$Residual block	64	16	3×3	1	1
$\mathcal{D}-$Residual block	64	1	3×3	1	1
$\mathcal{D}-$Deconv	64	1	4×4	2	1
$\mathcal{D}-$conv	64	1	3×3	1	1
$\mathcal{D}-$conv	3	1	1×1	1	1

filtering), we build a feature pyramid $(F^1, ..., F^m)$ and add several connections between the encoder and decoder, which is different from [4], [12]. This feature map pyramid $(F^1, ..., F^m)$ is then convolved with the operator-specific adaptive filterbank pyramid $(K_i^1, ..., K_i^m)$.

$$\begin{aligned}(F^1, ..., F^m) &= \mathcal{E}(I),\\ (F_i^1, ..., F_i^m) &= (F^1 \otimes K_i^1, ..., F^m \otimes K_i^m),\\ O_i &= \mathcal{D}(F_i^1, ..., F_i^m)\end{aligned} \tag{2}$$

where $\otimes$ denotes the convolution operation. $(F_i^1, ..., F_i^m)$ are the operator-specific transformed feature maps and fed into the decoder $\mathcal{D}$ to get the final filtering result O_i.

b) Adaptive FilterBank Pyramid: Assuming that we have n different image operators $\mathcal{H}_1, ..., \mathcal{H}_n$ to approximate, we represent each operator $\mathcal{H}_i$ with one corresponding *FilterBank* pyramid $(K_i^1, ..., K_i^m)$, where K_i^l is the filter bank kernel at level l. In order words, if total n operators are considered, n different *Filterbank* pyramids will be leveraged. To make each *Filterbank* pyramid $(K_i^1, ..., K_i^m)$ adaptive to different input parameter Θ_i, we use another weight sub-network $\mathcal{W}_i$ to directly regress its weight values, whose input is the tunable parameter Θ_i.

$$(K_i^1, ..., K_i^m) = \mathcal{W}_i(\Theta_i) \tag{3}$$

In the training stage, all these weight sub-networks are jointly trained with the backbone encoder and decoder network. Specifically, at each iteration, we randomly sample some input images from the training dataset as a batch. For each image, we further randomly sample an image operator and a parameter value. This batch of images and sampled parameters are fed into the backbone network and weight sub-networks respectively to predict the target filtering results.

Since these weight sub-networks and the backbone network are coupled in the same computation graph, the gradient of the backbone network will be back-propagated to these sub-networks to guide their training. Once these sub-networks are well-trained, they will dynamically change the weights of their corresponding filterbank pyramid with different user-selected parameters. And because the filterbank pyramid is inserted between the encoder and decoder, the feature maps generated by the encoder can be reused when switching image operators

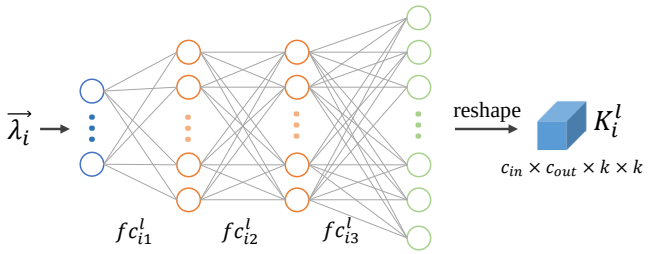


Fig. 2. The diagram of the weight sub-network to generate K_i^l, which simply consists of three fully connected layers.

or tuning parameter for the same input image, which is around two times faster than a re-evaluation of the whole network.

c) Weight Sub-network: Each operator-specific weight sub-network $\mathcal{W}_i$ adopts a similar network structure, except that the input dimension may not be the same because of different control parameter dimension. As shown in Figure 2, it simply consists of three fully connected layers, between which two nonlinear ReLU layers are inserted. The first and second hidden unit number is 8, and the third hidden unit number depends on the detailed weight dimension of the filterBank pyramid. In our case, $K_i^1, ..., K_i^m$ are often with a dimension of $c_{out} \times c_{in} \times k \times k$, where c_{out}, c_{in}, k denote the output channel number, input channel number and kernel size respectively.

$$K_i^l = fc_{i3}^l(\sigma(fc_{i2}^l(\sigma(fc_{i1}^l(\Theta_i))))) \tag{4}$$

d) Loss Function: For the image smoothing, restoration and enhancement tasks, we follow the same strategy as [27], [13], we simply use the mean squared Euclidean loss (L2 loss) to train our network:

$$\begin{aligned}O_i &= \mathcal{D}(\mathcal{E}(\mathcal{I}, \nabla_{\mathcal{I}}) \otimes \mathcal{W}_i(\Theta_i))\\ \mathcal{L} &= \|O_i - Y_i\|^2\end{aligned} \tag{5}$$

where O_i is the predicted processing result corresponding to one random sampled filter and discrete parameter by combination of Equation (2) and Equation (3), Y_i is the ground truth. To accelerate the training procedure, these training image pairs are pre-generated. Though we simply adopt L2 loss here, our proposed method is also general to other loss functions like L1 and auxiliary perceptual loss [19] to better maintain the image structures.

For neural style transfer, we adopt the same loss function in [15], [19], which is the weighted sum of two parts: the *content loss* $\mathcal{L}_c$ to preserve the structure of the input content image I and the *style loss* $\mathcal{L}_i^s$ to encourage the style fidelity to the i-th target reference style image I_i^s. And we set the style weight β_i as the input control parameter Θ_i, which determines the final stylization degree. And in the training stage, we will also dynamically change the loss weight of $\mathcal{L}_s$ with different input β_i as below.

$$\mathcal{L} = \alpha \mathcal{L}_c(O_i, I) + \beta_i \mathcal{L}_s(O_i, I_i^s) \tag{6}$$

e) Extra Edge Input: For the image smoothing, restoration and enhancement tasks, [13], [12] demonstrate that using the edge information of the input image I as the extra channel will help the network to preserve the original image structures

and generate better results. Therefore, we also pre-calculate the edge map $\nabla_{\mathcal{I}}$ of $\mathcal{I}$ as an extra input of the network. Note that all the baselines, in the experiments, use this extra input for a fair comparison.

$$\nabla_{\mathcal{I}}(x,y) = \frac{1}{4}|\mathcal{I}_{x,y} - \mathcal{I}_{x-1,y}| + |\mathcal{I}_{x,y} - \mathcal{I}_{x+1,y}| + |\mathcal{I}_{x,y} - \mathcal{I}_{x,y-1}| + |\mathcal{I}_{x,y} - \mathcal{I}_{x,y+1}| \quad (7)$$

IV. Experiments

In this section, we demonstrate the effectiveness and superiority of our *"Adaptive FilterBank Pyramid"* with respect to state-of-the-art baseline methods. Firstly, we adopt it to approximate six different edge-aware image smoothing operators, including bilateral filter [24], WLS [14], L_0 [26], RTV [28], RGF [31] and WMF [32], to prove the effectiveness and feasibility of our method and conduct some ablation analysis. Some of these operators like RTV are also learned in previous baseline methods [5], [13], [12], [27] because of their slow running speed. Secondly, we further apply our method to other wide range of image processing tasks, including image denoising, image deblocking, image enhancement and neural style transfer, to show the powerful flexibility and generality of the proposed plugin module.

A. Implementation Details.

For image smoothing, denoising, deblock and enhancement tasks, we use the PASCAL VOC dataset [11] to pre-generate the training image pairs as [12]. The total image number is about 17k, and we randomly select 500 images as the test set. Since our network is designed to process an image within a continuous parameter range rather than a specific parameter value, we generate our dataset by sampling different parameters randomly. In particular, for each image, we randomly sample six different parameter values. Although it is not able to cover the entire parameter space for a single image, it should be enough by using the above large scale dataset. For the sampling strategy, different from [12], we sample parameter values in the parameter range uniformly. For neural style transfer task, we follow the same training configuration as [19] but regard the style weight as the control parameter.

By default, we use the Adam optimizer to train our network with a batch size of 16 for 100 epochs. The initial learning rate is 0.01 and decreased by a gamma multiplier of 0.1 at every 40 epochs. All the different operators are sampled with the same probability at each iteration. Considering the tradeoff between performance gain and complexity as shown in following analysis Section IV-C-b, the default pyramid level m is 2 in this paper, and F^1, F^2 are the output of the third convolution layer and the fourth residual block respectively.

B. Comparisons on Image Smoothing Tasks

a) Comparison to Single-Operator-Single-Parameter Baseline: Since our goal is to train multiple different image operators within one single network while enabling the continuous parameter tuning, our first baseline is the method which trains single model for one image operator of one specific parameter value. Since the final absolute performance often depends on the backbone network, the same default backbone is also used for the baseline. To measure the performance difference, the PSNR and SSIM error metrics are adopted.

For our baseline, we train the single model with five different discrete parameter values for each operator. Though our method supports continuous parameters, only the results on these discrete parameter values are compared. As shown in Table II, our method can achieve comparable results with the single-operator-single-parameter baseline, but we train all these operators within a single model while enabling continuous parameter tuning. We have shown some visual comparison results in Figure 3 along with the ground truth. Obviously, our method can produce similar high-quality results for a wide range of parameter values and different operators without the need of retraining many separable models for each parameter of each operator. They are almost identical to that generated by the single model baseline and ground truth operators.

b) Comparison to Multi-Operator Baseline: We compare our method against two recent methods [5], [12] which have shown their simple extensions to multiple operators and parameter tuning. Chen et al. [5] propose to add two extra input channels to indicate different operators and parameters, which is the most naive and straightforward way, then let the network learn like a black-box. As shown in Table IV, our results are much better than [5]. This further demonstrates the superiority of our explicit filterbank pyramid representation to the naive black-box manner of [5].

In [12], multiple different weight learning networks are used to learn all the convolutional weights of the backbone network. If the backbone network consists of n convolutional layers, n different weight subnets are needed, which is very storage consuming. By contrast, our method only needs a fixed number (e.g. 6) of sub-networks to regress the weights of the filter pyramid. This is an important advantage, especially for a deeper backbone network. Moreover, since the encoder and decoder are shared by different operators in our method, this helps to explore and utilize the common information of these filters. For convenience, we directly cite the results from [12]. By leveraging different levels of feature and the explicit representation for each operator, our method significantly outperforms [12].

c) Comparison with State-of-the-art Methods: Although the goal of this paper is to achieve a good balance between better results and enabling more capabilities for approximating image operators, we still provide some comparisons to show the position of our method with our default configuration. As shown in Table VI, we compare our method with previous state-of-the-art methods [27], [13] which are designed for some specific image operators. Though they are trained for only one operator with one specific parameter value, our method can achieve very comparable results as [13], which are much better than [27].

d) Speed Comparison: For the training speed with the same backbone, since the single-operator-single-parameter baseline needs to train separated models for each operator of each parameter value, the training time is roughly $m * n$ (m

TABLE II
QUANTITATIVE COMPARISON TO THE SINGLE-OPERATOR-SINGLE-PARAMETER BASELINE USING THE DEFAULT BACKBONE. IT SHOWS THAT OUR METHOD CAN ACHIEVE VERY COMPARABLE RESULTS WITH THE SINGLE-OPERATOR-SINGLE-PARAMETER BASELINE WHILE ENABLING MULTIPLE OPERATORS AND DYNAMIC CONTINUOUS PARAMETER TUNING WITHIN ONE SINGLE NETWORK.

	Bilateral			L_0			RGF			RTV			WLS		
metric	Θ	baseline	our	Θ	baseline	our	Θ	baseline	our	Θ	baseline	our	Θ	baseline	our
PSNR	0.05	42.42	41.32	0.002	40.51	39.58	1	41.40	40.36	0.005	41.59	40.87	0.1	43.80	42.93
	0.10	39.53	39.92	0.005	38.91	38.45	3	38.75	38.64	0.01	41.25	40.85	0.5	42.55	42.55
	0.20	40.78	39.19	0.01	37.54	37.32	5	38.39	38.06	0.02	41.59	40.78	1.0	41.85	41.86
	0.40	41.52	39.81	0.05	34.30	34.32	7	37.87	37.36	0.03	41.37	40.48	3.0	40.41	40.30
	0.60	41.65	40.70	0.10	32.15	32.24	9	37.21	36.50	0.05	40.91	39.45	5.0	40.18	39.42
	ave.	41.18	40.19	ave.	36.68	36.18	ave.	38.72	38.18	ave.	41.34	40.49	ave.	41.76	41.41
SSIM	0.05	0.992	0.992	0.002	0.986	0.987	1	0.994	0.990	0.005	0.990	0.989	0.1	0.994	0.993
	0.10	0.990	0.991	0.005	0.985	0.986	3	0.987	0.987	0.01	0.990	0.990	0.5	0.993	0.992
	0.20	0.992	0.990	0.01	0.983	0.985	5	0.986	0.984	0.02	0.992	0.991	1.0	0.991	0.992
	0.40	0.992	0.990	0.05	0.979	0.981	7	0.984	0.982	0.03	0.992	0.991	3.0	0.988	0.989
	0.60	0.991	0.990	0.10	0.973	0.976	9	0.982	0.979	0.05	0.992	0.991	5.0	0.988	0.987
	ave.	0.991	0.991	ave.	0.981	0.983	ave.	0.987	0.984	ave.	0.991	0.990	ave.	0.991	0.991

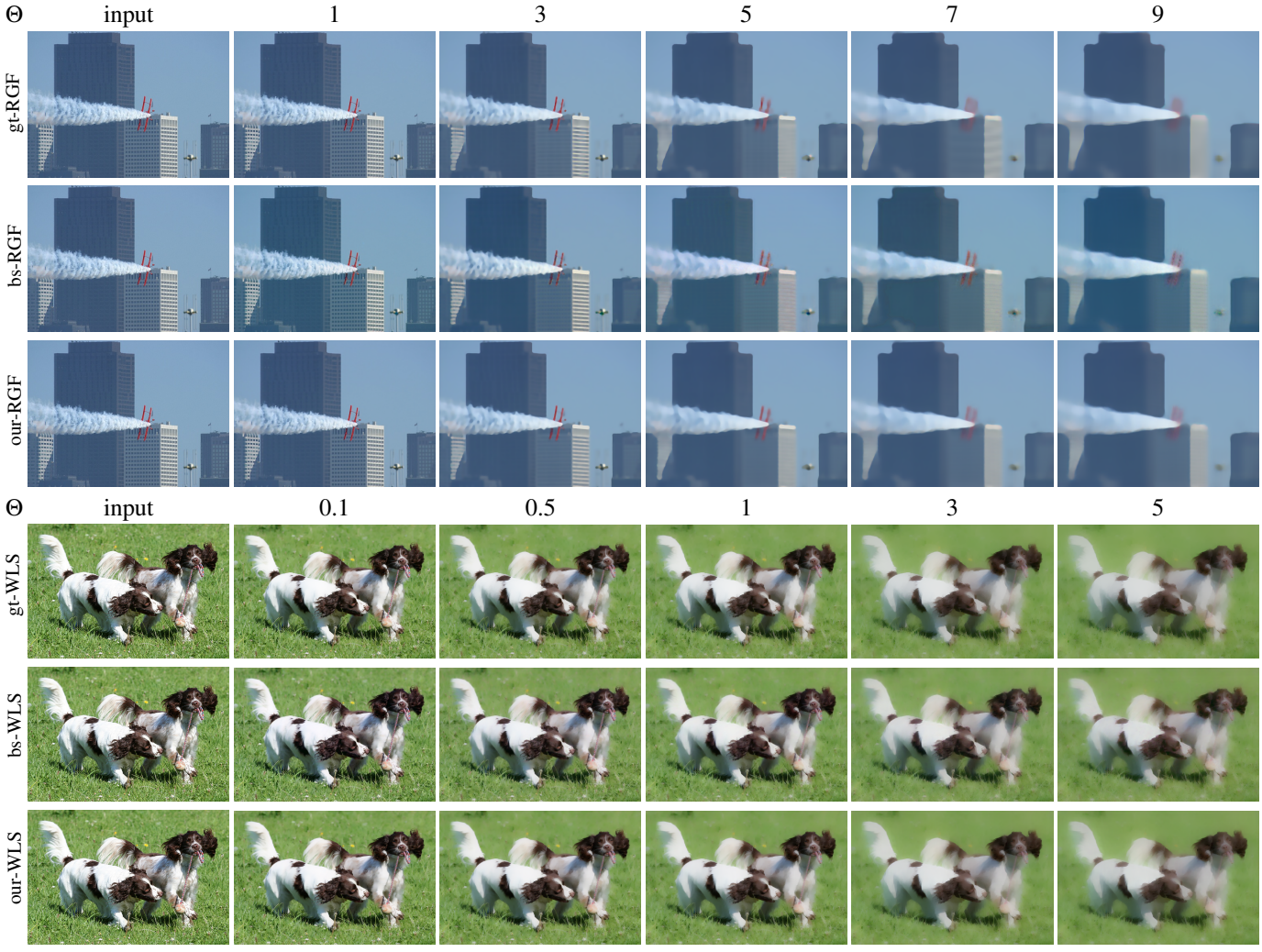


Fig. 3. Visual comparison results among the ground truth operator('gt-*'), the single-operator-single-parameter baseline ('bs-*) and our method ('our-'). Obviously, with different input control parameters for each image operator, our method can produce different visual plausible results without the need of retraining multiple models for each parameter values, which are almost identical to that generated by the baseline method and ground truth operators.

TABLE III
TO SHOW THE GENERALITY OF OUR *FilterBank* PYRAMID, WE ALSO PROVIDE COMPARISON RESULTS WITH THE SINGLE-OPERATOR-SINGLE-PARAMETER BASELINE AS TABLE II BUT USING A DIFFERENT BONE [5].

		Bilateral			L_0			RGF			RTV			WLS	
metric	Θ	baseline	our	Θ	baseline	our	Θ	baseline	our	Θ	baseline	our	Θ	baseline	our
PSNR	0.05	36.83	39.56	0.002	36.80	37.15	1	36.80	38.99	0.005	34.93	37.38	0.1	38.14	40.82
	0.10	36.29	37.50	0.005	33.71	35.89	3	34.25	36.16	0.01	33.99	34.49	0.5	36.72	39.41
	0.20	35.80	36.15	0.01	33.04	34.49	5	33.90	35.17	0.02	33.39	35.99	1.0	36.15	38.41
	0.40	36.36	36.52	0.05	30.33	31.22	7	33.29	34.43	0.03	33.41	35.31	3.0	35.64	36.59
	0.60	37.58	37.18	0.10	28.81	29.40	9	33.08	33.50	0.05	33.44	34.16	5.0	34.95	35.67
	ave.	36.83	37.38	ave.	32.54	33.63	ave.	34.26	35.65	ave.	33.83	35.46	ave.	36.32	38.18
SSIM	0.05	0.976	0.986	0.002	0.968	0.977	1	0.980	0.988	0.005	0.959	0.974	0.1	0.981	0.990
	0.10	0.977	0.984	0.005	0.952	0.973	3	0.962	0.974	0.01	0.954	0.973	0.5	0.976	0.986
	0.20	0.976	0.980	0.01	0.952	0.968	5	0.958	0.968	0.02	0.954	0.972	1.0	0.973	0.983
	0.40	0.977	0.979	0.05	0.938	0.957	7	0.953	0.964	0.03	0.956	0.970	3.0	0.969	0.975
	0.60	0.979	0.979	0.10	0.933	0.941	9	0.952	0.957	0.05	0.958	0.964	5.0	0.964	0.970
	ave.	0.977	0.982	ave.	0.949	0.963	ave.	0.961	0.970	ave.	0.956	0.971	ave.	0.973	0.981

TABLE IV
COMPARISON WITH MULTI-OPERATOR BASELINE [5] USING THE SAME BACKBONE. NOTABLY, OUR METHOD ARE MUCH BETTER.

		Bilater	L0	RGF	RTV	WLS
PSNR	[5]	35.03	31.82	33.17	33.51	35.26
	our	37.38	33.63	35.65	35.46	38.18
SSIM	[5]	0.970	0.945	0.950	0.949	0.966
	our	0.982	0.963	0.970	0.971	0.981

TABLE V
COMPARISON WITH THE MULTI-OPERATOR BASELINE [12], WHICH SHOWS OUR METHOD CAN ACHIEVE BETTER RESULTS THAN [12].

		L0	RGF	RTV	WLS	WMF
PSNR	[12]	33.54	35.90	37.69	38.02	36.46
	our	36.18	38.18	40.49	41.41	38.93
SSIM	[12]	0.972	0.976	0.982	0.983	0.970
	our	0.983	0.984	0.990	0.991	0.983

TABLE VI
COMPARISONS WITH STATE-OF-THE-ART SINGLE-OPERATOR-SINGLE-PARAMETER METHODS. WE CAN ACHIEVE COMPARABLE RESULTS AS [13], WHICH ARE BETTER THAN [27].

PSNR/SSIM	[27]	[13]	our
L_0	31.66 / 0.966	37.10 / 0.989	37.32 / 0.985
WLS	33.92 / 0.963	41.39 / 0.994	41.86 / 0.992

is parameter value number, and n is operator number) longer than our method. For the testing speed, thanks to our explicit representation design, our method can reuse the intermediate features. It only needs to rerun the layers after the proposed filterbank pyramid when users want to tune the parameters or switch filters. In contrast, previous methods [5], [12] need to rerun the whole network. Therefore our method is roughly two times faster when the encoder and decoder have similar computation cost. Moreover, the method of [12] also needs to run the weight sub-networks of all the layers.

C. Ablation Study

a) Generality to Different Backbones: Besides the default backbone, we also train our framework with a different backbone proposed in [5]. For this backbone, we regard the output of the second and fourth convolution layer as F^1, F^2, and insert the adaptive filterbank pyramid. As shown in Table III, with the backbone of [5], our proposed method achieves even better results than the single-operator-single-parameter baseline, which is quite surprising. One possible reason is that the newly added pyramid layers bring this gain. To demonstrate our hypothesis, we also add this pyramid layer into the original backbone network of [5] and retrain the single-operator-single-parameter baseline. The results (PSNR/SSIM) of the new baseline of each operator are: Bilateral (37.75/0.982), L_0 (33.79/0.960), RGF (35.45/0.972), RTV(36.06/0.973), WLS (37.85/0.979), which are better than the original baseline. On one hand, it demonstrates that our pyramid design is beneficial for image filtering. On the other hand, our results are still very close to this new baseline, which demonstrates our generality to different backbones.

b) Importance of FilterBank *Pyramid:* Feature pyramid demonstrated its effectiveness in many previous recognition tasks like [23]. To show the advantage of filterbank pyramid versus single scale filterbank for image filtering, we conduct a comparison experiment where only one filterbank is utilized (pyramid level is 1). The results are shown in Table VII at the row labeled as "*-our-p1". By comparison, our pyramid design achieves better results for the two different backbones [5], [13]. We further increase the pyramid level from two (default value) to three. We find that the performance ("*-our-p3") can be further boosted. But considering both the model complexity and the performance gain, we use pyramid level as two by default. This experiment further indicates that leveraging both the low-level and high-level features is very important to image operators.

c) Performance change with different task number: Since the goal of this paper is to jointly train multiple different

TABLE VII
COMPARISON RESULTS OF DIFFERENT PYRAMID LEVEL. NET1 IS THE BACKBONE USED IN [5], AND NET2 IS THE OUR DEFAULT BACKBONE. "-OUR-P1" TO "-OUR-P3" REFER TO THE RESULTS OF INCREASING THE PYRAMID LEVEL FROM 1 TO 3.

PSNR/SSIM	Bilateral	L_0	RGF	RTV	WLS	Ave.
Net1-our-p1	36.47 / 0.979	32.82 / 0.959	34.42 / 0.962	34.80 / 0.966	37.08 / 0.979	35.12 / 0.969
Net1-our-p2	37.38 / 0.982	33.63 / 0.963	35.65 / 0.970	35.46 / 0.971	38.18 / 0.981	36.06 / 0.973
Net1-our-p3	37.50 / 0.982	33.94 / 0.964	35.70 / 0.969	36.07 / 0.971	38.36 / 0.981	36.31 / 0.973
Net2-our-p1	39.97 / 0.991	36.01 / 0.982	37.93 / 0.984	40.49 / 0.991	41.16 / 0.991	39.11 / 0.988
Net2-our-p2	40.20 / 0.991	36.53 / 0.984	38.18 / 0.985	40.55 / 0.990	41.51 / 0.991	39.39 / 0.988
Net2-our-p3	40.48 / 0.991	36.78 / 0.984	38.35 / 0.985	40.76 / 0.991	41.61 / 0.991	39.60 / 0.988

TABLE VIII
COMPARISON RESULTS OF L_0 FILTER WHEN SIMULTANEOUSLY TRAINED WITH DIFFERENT NUMBERS OF IMAGE OPERATORS. IT SHOWS THAT THE OVERALL PERFORMANCE OF L_0 FILTER IS QUITE STABLE WHEN SIMULTANEOUSLY TRAINED WITH DIFFERENT IMAGE OPERATORS.

#Task	1	2	3	4	5
PSNR	33.79	33.46	33.75	33.69	33.63
SSIM	0.960	0.963	0.964	0.964	0.963

operators within one single network, we are very interested in how the performance changes with different operator numbers. To study this effect, we use L_0 filter as an example and jointly train it with different numbers of operators (from 1 to 5). As shown in Table VIII, the performance of L_0 is very stable when jointly trained with different numbers of filters. This further demonstrates the power and generality of our proposed framework, which leverages the common properties of different image operators very well. Note that the backbone of [5] is used in this experiment.

d) Incremental training for a new operator:: In real systems, given a well trained model, we may want to add one new operator to it. Then we have two solutions: 1) mix this operator with existing operators and train the model from scratch. 2) keep the existing backbone fixed and only train an extra filterbank pyramid for this operator. Compared to the former solution, the latter one is more flexible but challenging. To show the performance difference, we use the WMF operator [32] as an example and add it into our default model that already includes five operators. Table IX is the detailed comparison results. It can be seen that, the incremental training solution can achieve pretty good results. However, it is a little worse than the first joint training strategy and the single-operator-single-parameter baseline.

D. Extension to Other Tasks

After the demonstration of the effectiveness of our method on smoothing operators, we will extend our method to two image restoration tasks (i.e. denoising and deblocking), image enhancement and neural style transfer respectively in the following part to demonstrate the powerful flexibility and generality of our method. Note that, for these tasks, the goal of these experiments is not to achieve state-of-the-art performance but to show the proposed plugin module can support parameter tuning within one single network rather than multiple independent models.

a) Image Denoising: As most previous methods, the additive white Gaussian noise is considered in this paper. During training, we randomly select different noise levels in a range of [5,70] continuously and let the network learn to denoise different levels of noisy images. Our baseline is three separated models trained for the specific noise level $15, 25, 50$ following the similar evaluation strategy as previous denoising methods [7], [21], [30]. In this experiment, we consider gray image denoising and compare our method to our baseline and some previous state-of-the-art methods on the widely-used BSD68 dataset. As shown in Table X, our method can not only handle noisy images of any noise level in a continuous range but also achieve comparable results to the single-operator-single-parameter baseline ("single") that is trained for one specific noise level. More surprisingly, it is even better than some previous task-specific methods BM3D[7], UNLNet[21], we think the main reason is that our default backbone is better than the simple network structures used in these methods. The running speed of different methods is given in the last row of Table X.

In Figure 4, three visual denoising results are provided to compare our method to some previous state-of-the-art denoising methods, including BM3D [7], UNLNet [21] and DnCNN [30]. The first example image and the last two images are with a moderate noise level (25) and a strong noise level (50) from the BSD68 dataset respectively. It can be seen that our method can remove the noises of different levels and even recover the sharp image edges better than these methods.

In Table XI, we give more dense comparison results with our single model baseline. On the one hand, it shows that our method can handle all different noise levels very well. On the other hand, we find our method is even slightly better than the single model baseline for some specific noise levels, which may attribute to the better generalization ability from joint training. We further give one denoising example in Figure 5 and its result is consistent with the above quantitative results.

b) Image Deblocking: In real applications, JPEG compression is often used to compress the image into a smaller size to save bandwidth and storage. However, this compression procedure will cause some artifacts because of the high-frequency information loss, which can be seen in the second column in Figure 6. Image deblocking is the process aiming to remove the block artifacts of a JPEG compressed image. In the past, some methods have been proposed but are often designed to handle one specific compression quality factor.

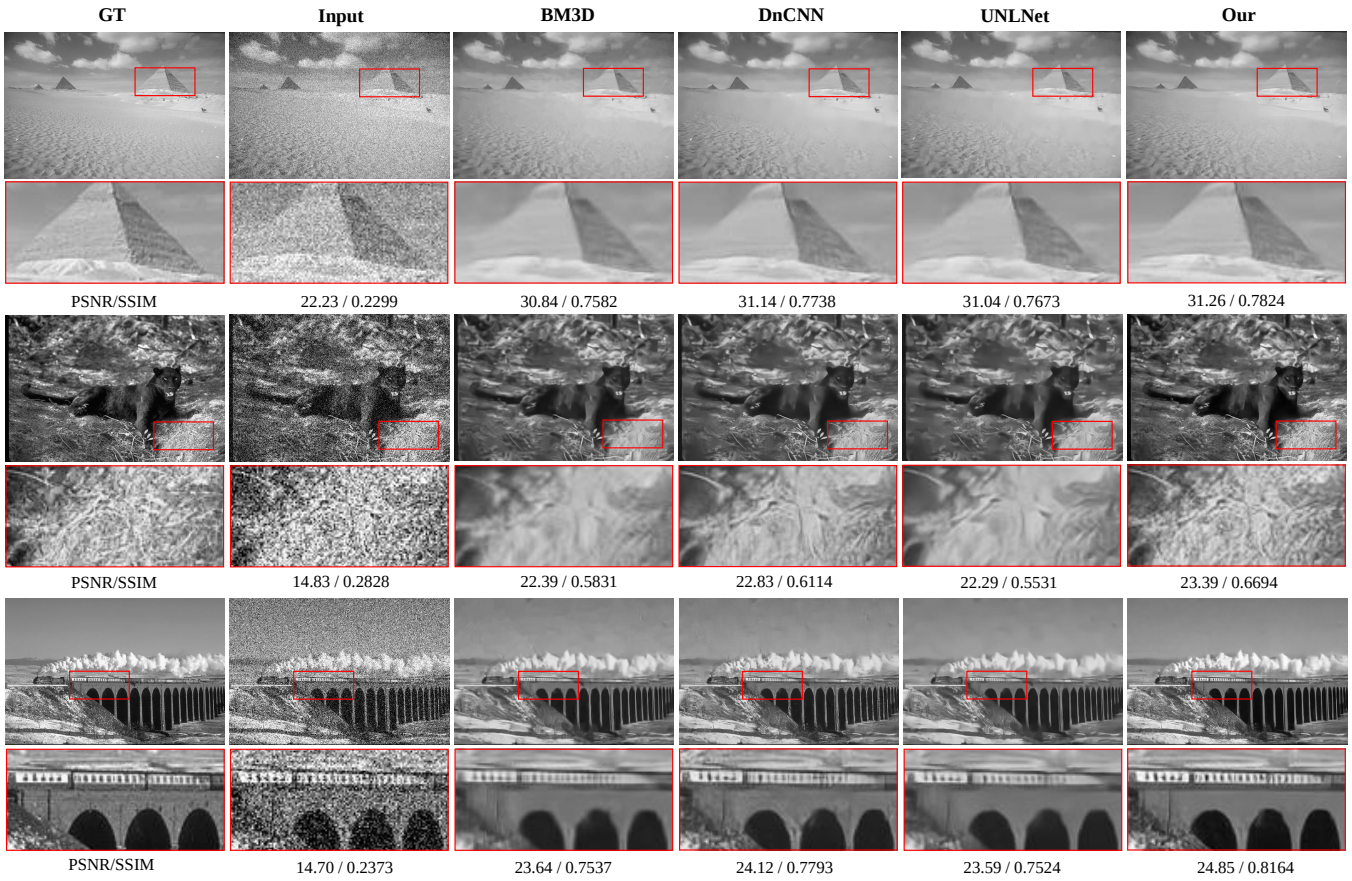


Fig. 4. Visual comparisons with some previous state-of-the-art methods for image denoising. The noise image of the first row is with the noise level of 25, and the last two rows are with noise level 50. It shows that our method supports denoising of different continuous noise levels and our denoising results are even better than some previous task- and parameter-specific methods.

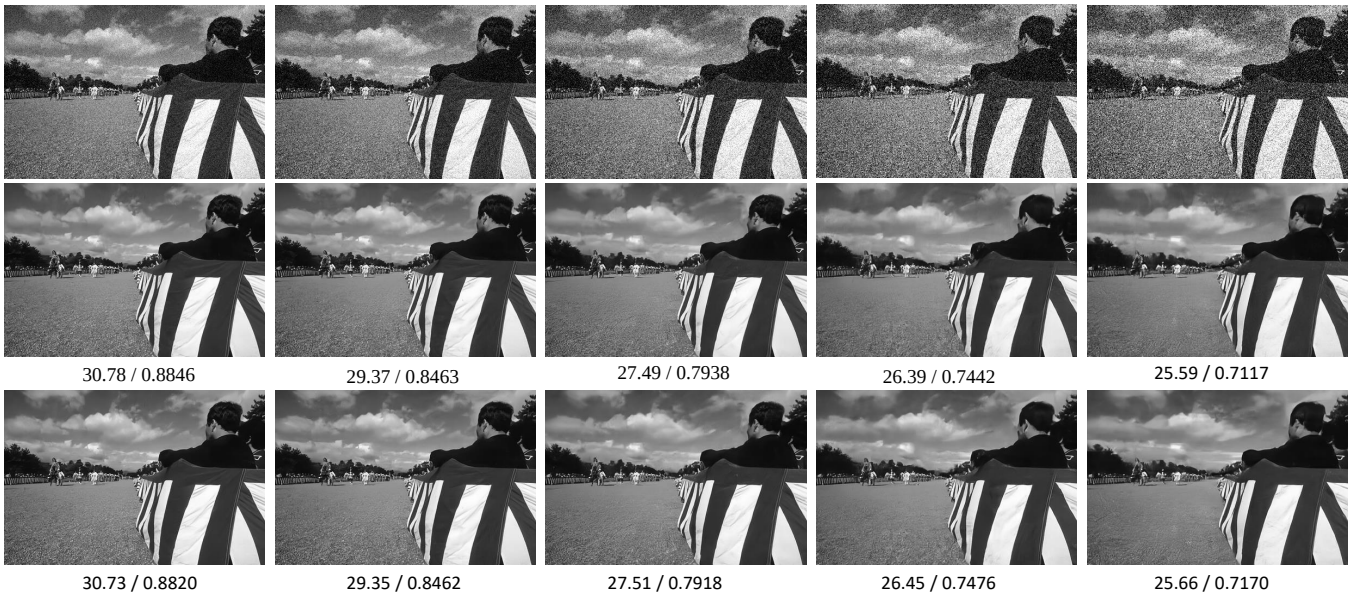


Fig. 5. One denoising example of different noise levels (left to right: 15,20,30,40,50). The first to last rows are input noise images, denoising results of the single-parameter baseline and our method respectively. The numbers below the images indicate the PSNR and SSIM. It shows that our method can handle different noise levels and can achieve comparable results to the single-parameter baseline.

TABLE IX
INCREMENTAL TRAINING RESULTS ("OUR-INCRE") FOR A NEWLY ADDED OPERATOR WMF. IT SHOWS THAT OUR METHOD CAN ACHIEVE PRETTY GOOD TRAINING RESULTS BY ONLY ADDING A NEW FILTERBANK PYRAMID WHILE KEEPING ALL OTHER REMAINING PARTS FIXED.

Θ	1	3	5	7	9	Avg
baseline	40.74 / 0.9909	39.85 / 0.9867	40.61 / 0.9888	40.33 / 0.9881	40.00 / 0.9869	40.31 / 0.9883
our	38.26 / 0.9863	39.30 / 0.9873	39.52 / 0.9879	39.19 / 0.9873	38.38 / 0.9852	38.93 / 0.9868
our-incre	35.85 / 0.9729	36.38 / 0.9714	36.19 / 0.9696	35.54 / 0.9661	34.78 / 0.9619	35.75 / 0.9684

TABLE X
QUANTITATIVE COMPARISONS FOR GRAY IMAGE DENOISING ON THE BSD68 DATASET. OUR METHOD CAN ACHIEVE COMPARABLE RESULTS TO THE SINGLE MODEL TRAINED ON ONE SPECIFIC LEVEL, AND IS EVEN BETTER THAN SOME TASK-SPECIFIC METHODS.

σ	BM3D [7]	DnCNN[30]	UNLNet [21]	our	our-single
15	31.07	31.60	31.47	31.65	31.68
25	28.57	29.15	28.96	29.24	29.22
50	25.62	26.21	26.04	26.35	26.32
speed(s)	1.34(cpu)	0.033(gpu)	0.5(gpu)	0.041 (gpu)	0.041(gpu)

TABLE XI
QUANTITATIVE COMPARISONS FOR GRAY IMAGE DENOISING ON THE BSD68 DATASET. OUR METHOD CAN ACHIEVE COMPARABLE RESULTS TO THE SINGLE MODEL TRAINED ON ONE SPECIFIC LEVEL, AND IS EVEN BETTER THAN SOME TASK-SPECIFIC METHODS.

σ	15	20	25	30	35	40	45	50
our	31.65	30.26	29.24	28.43	27.78	27.24	26.76	26.35
our-single	31.68	30.25	29.22	28.40	27.74	27.19	26.70	26.32

Like denoising, by incorporating the proposed plugin module, we want the network to be able to handle any compression quality factor in a continuous range. In Table XII, we compare our method on the LEVEL1 dataset to three JPEG deblocking methods including ARCNN[9], TNRD[6] and DnCNN [30]. It shows that our results are very close to the single-operator-single-parameter baseline, and even better than some previous methods [9], [6], [30].

Three visual deblock results are given in Figure 6, where the first example is with JPEG quality factor 20 and the latter two examples are with JPEG quality factor 10. It can be easily found that our method can recover cleaner and sharper structures of arbitrary JPEG compression levels, which further demonstrate our generalization ability.

c) Image Enhancement: In contrast to image smoothing, the goal of this task is to enhance the image details. In [14], Farbman et al. use the weighted least squares optimization framework for progressive coarsening of images and multi-scale detail extraction. In this experiment, we also use the default backbone network to approximate the operator proposed in [14] but enabling tuning the saturation factor to obtain different enhancement results. Some visual results are displayed in Figure 7.

d) Neural Style Transfer: For the original optimization based neural style transfer [15] method, we can control the final stylization degree by using different style weight β. However, many feed-forward network-based methods like [19], [4] are only designed to stylize images into one specific degree because no adaptive module exists in their networks. In this experiment, we have tried to insert the proposed plugin module into fast style transfer network [19]. Experiments show that we can make it possible to dynamically adjust the stylization degree continuously during runtime for [19]. In Figure 8, one example with different stylization degrees is given.

TABLE XII
QUANTITATIVE COMPARISONS FOR IMAGE DEBLOCKING ON THE LEVEL1 DATASET. OUR METHOD IS VERY COMPARABLE TO THE SINGLE MODEL BASELINE AND EVEN BETTER THAN SOME STATE-OF-THE-ART METHODS SPECIALLY DESIGNED FOR DEBLOCKING.

	Quality	ARCNN[9]	TNRD[6]	DnCNN [30]	our	our-single
PSNR	10	28.96	29.15	29.19	29.71	29.75
	20	31.29	31.46	31.59	32.08	32.10
SSIM	10	0.8076	0.8111	0.8123	0.8256	0.8269
	20	0.8733	0.8769	0.8802	0.8889	0.8891
speed (s)		0.01(gpu)	0.021(gpu)	0.033(gpu)	0.041(gpu)	0.041(gpu)

V. CONCLUSION

In this paper, we propose a novel plugin module call *"Adaptive FilterBank Pyramid"* that can be inserted into any backbone network to enable multiple operator training and continuous parameter tuning for controllable image processing. We applied the proposed module to different kinds of image operators and backbone network structures. Experiments demonstrate the strong generalization ability and effectiveness of our module, i.e. it can achieve comparable results with the single-operator-single-parameter baseline but is significantly more efficient in both training and testing. In the future, we will try to apply this idea to more tasks including face attribute transfer and multi-modal image generation.

Acknowledgement. This work was supported partly by the NSF Grant U1636201, Exploration Fund Project of USTC under Grant YD3480002001, Hong Kong ECS grant No. 21209119, Hong Kong UGC and Start-up grant No. 7200607, CityU of Hong Kong. Gang Hua is partially supported by National Key R&D Program of China Grant 2018AAA0101400 and NSFC Grant 61629301.

REFERENCES

[1] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65. IEEE, 2005.

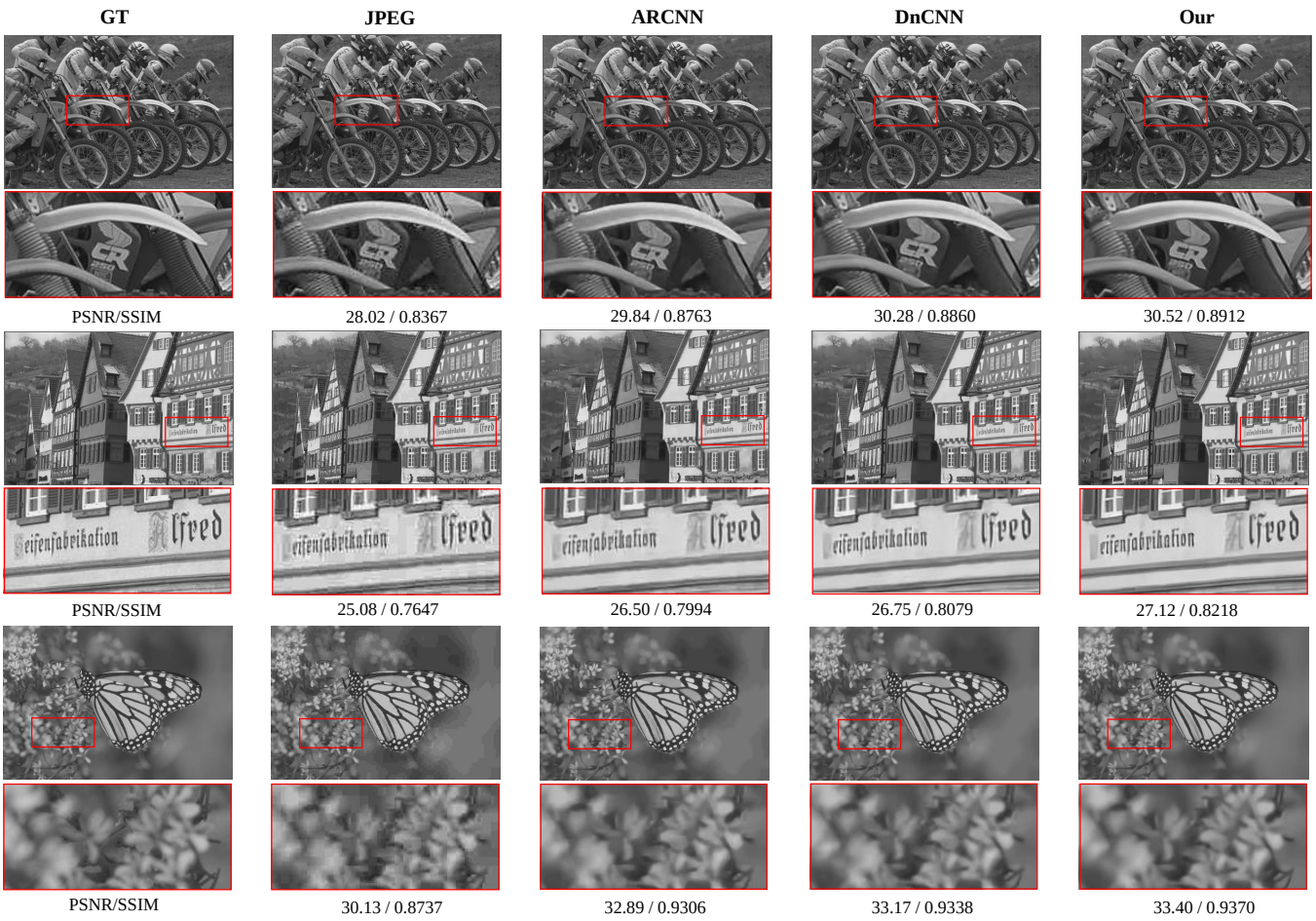


Fig. 6. Visual comparisons with some previous state-of-the-art method of JPEG deblocking. The first example is with quality factor 20 and the last two examples are with quality factor 10, which further demonstrate the generalization ability of our method to JPEG deblocking of arbitrary quality factors.

Fig. 7. Some visual results for image enhancement. By using the proposed plugin module, it supports continuous parameters tuning.

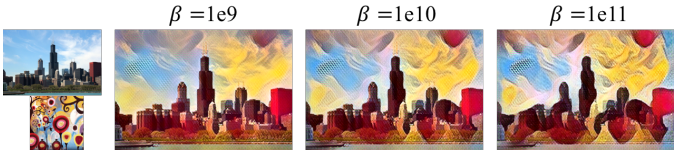


Fig. 8. Some visual results of fast neural style transfer that can continuously adjust stylization degree by using our plugin module in the single-style transfer network [19].

[2] D. Chen, M. He, Q. Fan, J. Liao, L. Zhang, D. Hou, L. Yuan, and G. Hua. Gated context aggregation network for image dehazing and deraining. *WACV 2019*, 2018.

[3] 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*, pages 1105–1114, 2017.

[4] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua. Stylebank: An explicit representation for neural image style transfer. In *Proc. CVPR*, volume 1, page 4, 2017.

[5] Q. Chen, J. Xu, and V. Koltun. Fast image processing with fully-convolutional networks. In *IEEE International Conference on Computer Vision*, volume 9, pages 2516–2525, 2017.

[6] Y. Chen and T. Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *TPAMI*, 2017.

[7] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *TIP*, 2007.

[8] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[9] C. Dong, Y. Deng, C. Change Loy, and X. Tang. Compression artifacts reduction by a deep convolutional network. In *CVPR*, 2015.

[10] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *ACM transactions on graphics (TOG)*, volume 21, pages 257–266. ACM, 2002.

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results, 2012.

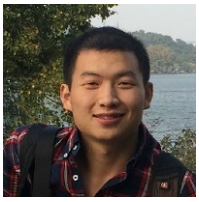
[12] Q. Fan, D. Chen, L. Yuan, G. Hua, N. Yu, and B. Chen. Decouple learning for parameterized image operators. In *ECCV 2018, European Conference on Computer Vision*, 2018.

[13] 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)*, pages 3238–3247, 2017.

[14] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. In *ACM Transactions on Graphics (TOG)*, volume 27, page 67. ACM, 2008.

[15] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

[16] 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.
[17] K. He, J. Sun, and X. Tang. Guided image filtering. In *European conference on computer vision*, pages 1–14. Springer, 2010.
[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
[19] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.
[20] 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.
[21] S. Lefkimmiatis. Universal denoising networks: A novel cnn architecture for image denoising. In *CVPR*, 2018.
[22] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep joint image filtering. In *European Conference on Computer Vision*, pages 154–169. Springer, 2016.
[23] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017.
[24] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.
[25] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, volume 1, page 3, 2017.
[26] L. Xu, C. Lu, Y. Xu, and J. Jia. Image smoothing via l 0 gradient minimization. In *ACM Transactions on Graphics (TOG)*, volume 30, page 174. ACM, 2011.
[27] L. Xu, J. Ren, Q. Yan, R. Liao, and J. Jia. Deep edge-aware filters. In *International Conference on Machine Learning*, pages 1669–1678, 2015.
[28] L. Xu, Q. Yan, Y. Xia, and J. Jia. Structure extraction from texture via relative total variation. *ACM Transactions on Graphics (TOG)*, 31(6):139, 2012.
[29] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.
[30] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *TIP*, 2017.
[31] Q. Zhang, X. Shen, L. Xu, and J. Jia. Rolling guidance filter. In *European Conference on Computer Vision*, pages 815–830. Springer, 2014.
[32] Q. Zhang, L. Xu, and J. Jia. 100+ times faster weighted median filter (wmf). In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2830–2837, 2014.



Dongdong Chen is a Ph.D. student from the University of Science and Technology of China. He is also a joint phd between his university and Microsoft Asia. His research interests mainly include image generation, image restoration, low-level image processing and high-level image recognition tasks.

Qingnan Fan is a Postdoctoral Scholar in the Computer Science Department of Stanford University. He received his PhD degree from Shandong University in 2019. His research interests mainly include image/video processing and 3D vision.

Jing Liao is an Assistant Professor with the Department of Computer Science, City University of Hong Kong (CityU). Prior to that, she was a Researcher at Visual Computing Group, Microsoft Research Asia. She received dual Ph.D. degrees from Zhejiang University and Hong Kong UST. Her primary research interests fall in the fields of Computer Graphics, Computer Vision, Image/Video Processing, Digital Art and Computational Photography.

Angelica Aviles-Rivero is currently a post-doctoral researcher (Research Associate) at DPMMS, University of Cambridge. She received her Ph.D. degree (in 2017) in Computer Vision and Robotics at the UPC-BarcelonaTech, Spain under the supervision of Prof. A. Casals. Her research lies at the intersection of computational mathematics and machine learning for applications to large-scale real world problems.

Lu Yuan received his PhD degree from the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology in 2009. Before that, he received his MS degree at TsingHua University. Now he is a Senior Research Manager in Microsoft Redmond. His research interests include computer vision, applied machine learning and computational photography.

Nenghai Yu is a full Professor at University of Science and Technology of China. He is also the director of Information Processing Center of USTC,deputy director of academic committee of School of Information Science and Technology. He received the Ph.D. degree from USTC in 2004. He was a visiting scholar in Institute of Production Technology, Faculty of Engineering, University of Tokyo, in 1999 and did cooperative research as the senior visiting scholar in Dept. of Electrical Engineering, Columbia University, from Apr. to Oct. 2008. His research focuses on image processing and video analysis, multimedia communication, media content security, Internet information retrieval, data mining and content filtering , network communication and security.

Gang Hua is the Vice President and Chief Scientist of Wormpex AI Research. Before that, he was the Principal Researcher/Research Manager at Microsoft Research between 2015 to 2018. He was an Associate Professor of Computer Science in Stevens Institute of Technology between 2011 and 2015, while holding an Academic Advisor position at IBM T. J. Watson Research Center. He has published more than 150 peer reviewed papers in top conferences such as CVPR/ICCV/ECCV, and top journals such as T-PAMI and IJCV. To date He holds 18 issued U.S Patents and also has 14 more U.S. Patents Pending. He is an IEEE Fellow, an IAPR Fellow, and an ACM Distinguished Scientist. His research focuses on artificial intelligence, computer vision, pattern recognition, machine learning, and robotics, with primary applications in the cloud and mobile intelligence domain.