# A General Decoupled Learning Framework for Parameterized Image Operators

Qingnan Fan, Dongdong Chen, *Member, IEEE*, Lu Yuan, Gang Hua, *Senior Member, IEEE*, Nenghai Yu and Baoquan Chen

**Abstract**—Many different deep networks have been used to approximate, accelerate or improve traditional image operators. Among these traditional operators, many contain parameters which need to be tweaked to obtain the satisfactory results, which we refer to as "parameterized image operators". However, most existing deep networks trained for these operators are only designed for one specific parameter configuration, which does not meet the needs of real scenarios that usually require flexible parameters settings. To overcome this limitation, we propose a new decoupled learning algorithm to learn from the operator parameters to dynamically adjust the weights of a deep network for image operators, denoted as the *base* network. The learned algorithm is formed as another network, namely the *weight learning* network, which can be end-to-end jointly trained with the *base* network. Experiments demonstrate that the proposed framework can be successfully applied to many traditional parameterized image operators. To accelerate the parameter tuning for real-time scenarios, the proposed framework can be further extended to dynamically change the weights of only one single layer of the *base* network while sharing most computation cost. We demonstrate that this real-time extension of the proposed decoupled learning framework even outperforms the state-of-the-art real-time methods.

**Index Terms**—Image Processing and Computer Vision, Filtering, Restoration, Smoothing

## 1 Introduction

Image operators are fundamental building blocks for many computer vision tasks, such as image filtering and restoration. To obtain the desired results, many of these operators contain some parameters that need to be tweaked. We refer them as "parameterized image operators" in this paper. For example, parameters controlling the smoothness strength are widespread in most smoothing methods, and a parameter denoting the target upsampling scalar is always used in image super resolution.

Recently, many CNN based methods [1], [2], [3] have been proposed to approximate, accelerate or improve these parameterized image operators and achieved significant progress. However, we observe that the networks in these methods are often only trained for one specific parameter configuration, such as edge-preserving filtering [1] with a fixed smoothness strength, or super resolving low-quality images [2] with a particular downsampling scale. Many different models need to be retrained for different parameter settings, which is both storage-consuming and time-consuming. It also prohibits these deep learning solutions from being applicable and extendable to a much broader corpus of images.

In fact, given a specific network structure, when training separated networks for different parameter configurations $\overrightarrow{\gamma}_k$ as [1], [2], [3], the learned weights $W_k$ are highly unconstrained and probably very different for each $\overrightarrow{\gamma}_k$. But can we find a common convolution weight space for different configurations by explicitly building their relationships? Namely, $W_k = h(\overrightarrow{\gamma}_k)$, where $h$ can be a linear or non-linear function. In this way, we can adaptively change the weights of the single target network based on $h$ in the runtime, thus enabling continuous parameter control.

To verify our hypothesis, we propose the first decoupled learning framework for parameterized image operators by decoupling the weights from the target network structure. Specifically, we employ a simple *weight learning* network $\mathcal{N}_{weight}$ as $h$ to directly learn the convolution weights of one task-oriented *base* network $\mathcal{N}_{base}$. These two networks can be trained end-to-end. During the runtime, the *weight learning* network will dynamically update the weights of the *base* network according to different input parameters, thus making the *base* network generate different objective results. This should be a very useful feature in scenarios where users want to adjust and select the most visually pleasant results interactively.

We demonstrate the effectiveness of the proposed framework for many different types of applications, such as edge-preserving image filtering with different degrees of smoothness, image super resolution with different scales of blurring, and image denoising with different magnitudes of noise. We also demonstrate the extensibility of our proposed framework on multiple input parameters for a specific application, and combination of multiple different image processing tasks. Experimental results demonstrate that the proposed framework is able to achieve almost as good results as the one solely trained for a single parameter value.

Despite of its generality and flexibility, all the convolution weights of $\mathcal{N}_{base}$ need to be updated during the runtime. In the other words, the whole network needs to be re-evaluated when a new parameter is selected. This is very time-consuming and unacceptable in real user scenarios. To be adaptive for practical applications, the proposed framework can be further extended to


- *Qingnan Fan is with Department of Computer Science and Technology, Shandong University, Qingdao, Shandong 266237, China. E-mail: fqnchina@gmail.com*
- *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 and Gang Hua are with Microsoft Research, Redmond, Washington 98052, USA. E-mail: {luyuan,ganghua}@microsoft.com*
- *Baoquan Chen is with Peking University, Beijing 100871, China. E-mail: baoquan@pku.edu.cn*

dynamically change the weights of only one single layer while sharing most of the computation, which make it feasible to run on both GPU and CPU devices in real time. With our default $\mathcal{N}_{base}$ network structure, experiments demonstrate this real-time version outperforms the state-of-the-art real-time methods [4], [5] by a large margin.

As an extra bonus, the proposed framework makes it easy to analyze the underlying working principle of the trained task-oriented network by visualizing different parameters. The knowledge gained from this analysis may inspire more promising research in this area. To sum up, the contributions of this paper lie in the following four aspects.

- We propose the first decoupled learning framework for parameterized image operators, where a *weight learning* network is learned to adaptively predict the weights for the task-oriented *base* network in the runtime.
- We show that the proposed framework can be learned to incorporate many different parameterized image operators and achieve very competitive performance with the one trained for a single specific parameter or operator.
- We extend our framework to enable real-time parameter tuning for real user scenarios, which outperforms many state-of-the-art methods by a large margin.
- We provide a unique perspective to understand the working principle of the trained task-oriented network with some valuable analysis and discussion, which may inspire more promising research in this area.

## 2 Related Work

In the past decades, many different image operators have been proposed for low level vision tasks. Previous work [6], [7], [8], [9] proposed different priors to smooth images while preserving salient structures. Some papers [10], [11] utilized the spatial relationship and redundancy to remove unpleasant noise in the image. Other papers [12], [13], [14] aimed to recover a high-resolution image from a low-resolution image. Among them, many operators are allowed to tune some built-in parameters to obtain different results, which is the focus of this paper.

With the development of deep learning techniques, many different neural networks are proposed to approximate, accelerate and improve these operators [1], [3], [15], [16], [17]. But their common limitation is that one model can only handle one specific parameter setting. To enable all other parameters, enormous different models need to be retrained, which is both storage-consuming and time-consuming. By contrast, our proposed framework allows us to input continuous parameters to dynamically adjust the weights of the task-oriented *base* network. Moreover, it can even be applied to multiple different parameterized operators with one single network.

Recently, Chen *et al.* [18] conducted a naive extension for parameterized image operators by concatenating the parameters as extra input channels to the network. Compared to their method, where both the network structure and weights maintain the same for different parameters, the weights of our *base* network are adaptively changed. Experimentally we find our framework outperforms their strategy by integrating multiple image operators. By decoupling the network structure and weights, our proposed framework also makes it easier to analyze the underlying working principle of the trained task-oriented network, rather than leaving it as a black box as in many previous works like [18].

To enable real-time image processing, a simple scheme to accelerate an operator is to apply it at a low-resolution image then upsample the result by reintroducing high-resolution details, which is used in bilateral upsampling [19] and the fast guided filter [20]. To unify and generalize these two methods, [4] presents bilateral guided upsampling, which is further extended to [5] by incorporating such a technique into an end-to-end trained deep network. Though [4] and [5] are able to run very efficiently on even CPU devices, their resultant image quality is not good enough. By contrast, our real-time extension directly runs the approximated operators on the original image resolution by sharing most of the computation costs for different operators, which not only enables real-time operation on CPU devices but is also able to demonstrate its superior performance over these recent state-of-the-art approaches [4], [5] by a large margin.

Our method is also related to evolutionary computing and meta learning. Schmidhuber [21] suggested the concept of fast weights in which one network can produce context-dependent weight changes for a second network. Some other works [22], [23], [24] casted the design of an optimization algorithm as a learning problem, Recently, Ha *et al.* [25] proposed to use a static hypernetwork to generate weights for a convolutional neural network on MNIST and Cifar classification. They also leverage a dynamic hypernetwork to generate weights of recurrent networks for a variety of sequence modelling tasks. The purpose of their paper is to exploit weight sharing property across different convolution layers. But in our cases, we pay more attention to the common shared property among numerous input parameters and many different image operators.

## 3 Method

### 3.1 Problem Definition and Motivation

The input color image and the target parameterized image operators are denoted as $\mathcal{I}$ and $f(\overrightarrow{\gamma}, \mathcal{I})$ respectively. $f(\overrightarrow{\gamma}, \mathcal{I})$ transforms the content of $\mathcal{I}$ locally or globally without changing its dimension. $\overrightarrow{\gamma}$ denotes the parameters which determine the transform degree of $f$ and may be a single value or a multi-value vector. For example, in $L_0$ smoothing[26], $\overrightarrow{\gamma}$ is the balance weight controlling the smoothness strength, while in RTV filter [8], it includes one more spatial gaussian variance. In most cases, $f$ is a highly nonlinear process and solved by iterative optimization methods, which is very slow in runtime.

Our goal is to implement parameterized operator $f$ with a base convolution network $\mathcal{N}_{base}$. In previous methods like [3], [17], given a specific network structure of $\mathcal{N}_{base}$, separated networks are trained for different parameter configurations $\overrightarrow{\gamma}_k$. In this way, the learned weights $\overrightarrow{W}_k$ of these separated networks are highly unconstrained and probably very different. But intuitively, for one specific image operator, the weights $\overrightarrow{W}_k$ of different $\overrightarrow{\gamma}_k$ might be related. So retraining separated models is too redundant. Motivated by this, we try to find a common weight space for different $\overrightarrow{\gamma}_k$ by adding a mapping constraint: $\overrightarrow{W}_k = h(\overrightarrow{\gamma}_k)$, where $h$ can be a linear or non-linear function.

In this paper, we directly learn $h$ with another *weight learning* network $\mathcal{N}_{weight}$ rather than design it by handcraft. Assuming $\mathcal{N}_{base}$ is a fully convolutional network containing a total of $n$ convolution layers, we denote their weights as $\overrightarrow{W}_k = (W_1, W_2, ..., W_n)$ respectively, then

$$(W_1, W_2, ..., W_n) = \mathcal{N}_{weight}(\overrightarrow{\gamma}) \tag{1}$$

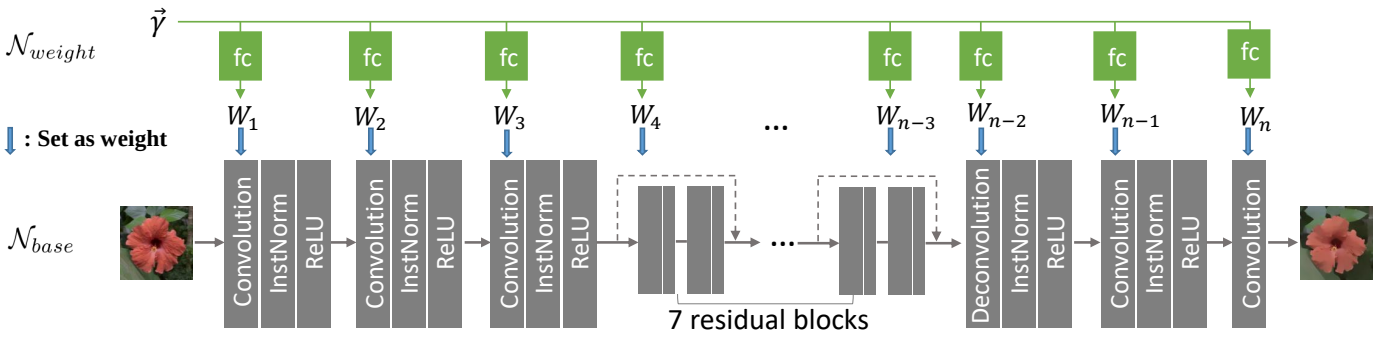



Fig. 1. Our system consists of two networks: the above *weight learning* network $\mathcal{N}_{weight}$ is designed to learn the convolution weights for the bottom *base* network $\mathcal{N}_{base}$. Given a parameterized image operator constraint by $\overrightarrow{\gamma}$, these two networks are jointly trained, and $\mathcal{N}_{weight}$ will dynamically update the weights of $\mathcal{N}_{base}$ for different $\overrightarrow{\gamma}$ in the inference stage.

where the input of $\mathcal{N}_{weight}$ is $\overrightarrow{\gamma}$ and the outputs are these weight matrices. In the training stage, $\mathcal{N}_{base}$ and $\mathcal{N}_{weight}$ can be jointly trained. In the inference stage, given different input parameter $\overrightarrow{\gamma}$, $\mathcal{N}_{weight}$ will adaptively change the weights of the target base network $\mathcal{N}_{base}$, thus enabling continuous parameter control.

Besides the original input image $\mathcal{I}$, the computed edge maps are shown to be a very important input signal for the target *base* network in [1]. Therefore, we also pre-calculate the edge map $E$ of $\mathcal{I}$ and concatenate it to the original image as an extra input channel:

$$E_{x,y} = \frac{1}{4}\sum_{c}(|\mathcal{I}_{x,y,c} - \mathcal{I}_{x-1,y,c}| + |\mathcal{I}_{x,y,c} - \mathcal{I}_{x+1,y,c}| + |\mathcal{I}_{x,y,c} - \mathcal{I}_{x,y-1,c}| + |\mathcal{I}_{x,y,c} - \mathcal{I}_{x,y+1,c}|) \quad (2)$$

where $x, y$ are the pixel coordinates and $c$ refers to the color channels.

To jointly train $\mathcal{N}_{base}$ and $\mathcal{N}_{weight}$, we simply use pixel-wise L2 loss in the RGB color space as [18] by default:

$$\mathcal{L} = \|\mathcal{N}_{base}(\mathcal{N}_{weight}(\overrightarrow{\gamma}), \mathcal{I}, E) - f(\overrightarrow{\gamma}, \mathcal{I})\|^2 \quad (3)$$

### 3.2 Network Structure

As shown in Fig. 1, our *base* network $\mathcal{N}_{base}$ follows a similar network structure as [1]. We employ 20 convolutional layers with the same $3 \times 3$ kernel size, among which the intermediate 14 layers are formed as residual blocks. Except the last convolution layer, all the former convolutional layers are followed by an instance normalization [27] layer and a ReLU layer. To enlarge the receptive field of $\mathcal{N}_{base}$, the third convolution layer downsamples the dimension of feature maps by 1/2 using stride 2, and the third-to-last deconvolution layer (kernel size of $4 \times 4$) upsamples the downsampled feature maps to the original resolution symmetrically. In this way, the receptive field is effectively enlarged without losing too much image detail, and meanwhile the computation cost of intermediate layers is reduced. To further increase the receptive field, we also adopt the dilated convolution [28] as [18], more detailed network structure can be found in the supplementary material.

In this paper, the *weight learning* network $\mathcal{N}_{weight}$ simply consists of 20 fully connected (fc) layers by default. The $i_{th}$ fc layer is responsible to learn the weights $W_i$ for the $i_{th}$ convolutional layer, which can be written as following:

$$W_i = A_i\overrightarrow{\gamma} + B_i, \qquad \forall i \in \{1, 2, ..., 20\} \quad (4)$$

Where $A_i, B_i$ are the weight and bias of the $i_{th}$ fc layer. Assuming the parameter $\overrightarrow{\gamma}$ has a dimension of $m$ and $W_i$ has a dimension of $n_{wi}$. The dimension of $A_i$ and $B_i$ would be $n_{wi} \times m$ and $n_{wi}$ respectively.

Note in this paper, we don't intend to design an optimal network structure neither for the *base* network $\mathcal{N}_{base}$ nor the *weight learning* network $\mathcal{N}_{weight}$. On the contrary, we care more about whether it is feasible to learn the relationship between the weights of $\mathcal{N}_{base}$ and different parameter configurations $\overrightarrow{\gamma}$ even by such a simple *weight learning* network $\mathcal{N}_{weight}$.

### 3.3 Real-Time Extension

In the above default setting, the weights of all the convolution layers in $\mathcal{N}_{base}$ are learned by the *weight learning* network $\mathcal{N}_{weight}$ and would be dynamically changed with the input parameters. That is to say, if users want to do some parameter tuning for different visual effects, the whole network needs to be re-evaluated. However in order to obtain the best image quality, most current methods like [1], [17], [18], including our default design, often requires to run through a complex deep network. Though some of them can run in real time on powerful GPU devices, they are still very slow for CPU computation.

To tackle the efficiency issue, we further extend our framework from learning all the convolution weights to learning the weights of only one single layer. Since all the following layers behind this adjustable layer need to be re-evaluated when fed with varying input features, it is better to put this layer as deeper as possible in the *base* network. In this way, the computation of all the preceding layers can be shared by different parameters, and only the layers after this single adjustable layer need to be re-evaluated. This is of great practicability for many real scenarios.

Specifically, that is to say, only the weights $W_i$ of the $i$th layer is the function of input parameter $\overrightarrow{\gamma}$ as Equation (4) while all the remaining weights $W_k (k \neq i)$ are shared. During the runtime for parameter tuning, we only need to re-run the layers $k (k \geqslant i)$. Note that in our default network structure, $W_i$ can be either the weights of the convolutional layer or the scale and shift paramters in the following normalization layer. In this paper, by default, we choose the last instance normalization layer as the target layer and learn its scale and shift parameters, after which only one convolution layer needs to be run.

Though it seems more difficult for the network to adapt its behavior just with this single deep layer, we demonstrate its generality and effectiveness for different operators in the experiment

section. Such a network design is able to outperform real-time state-of-the-art approaches [4], [5] by a large margin. More comprehensive ablation study about the learned layer type (convolution or instance normalization) and position $k$ is conducted in the analysis section.

## 4 Experiments on the Proposed Framework

To demonstrate the ability of our proposed framework in incorporating parameterised image operators while maintaining their accuracy, we evaluate the proposed decoupled learning framework with different training configurations as shown from subsection 4.3 to 4.5. We leverage two representative types of image processing tasks: image filtering and image restoration. Within each of them, more than four popular operators are selected for detailed experiments. Below, we briefly introduce all the operators and their implementation details as follows.

### 4.1 Choice of Image Operators

**Image Filtering:** here we employ six popular image filters, denoted as $L_0$ [6], WLS [29], RTV [8], RGF [9], WMF [30] and LLF [31], which have been developed specifically for many different applications, such as edge-preserving image smoothing, texture removal, detail exaggeration, image abstraction, and image enhancement.

- $L_0$ **smooth** [6] - sharpening major image structures while eliminating a manageable degree of details by minimizing $L_0$ image gradients.
- **RTV** [8] - extracting structures from textures by optimizing the new inherent variation and relative total variation measures.
- **WLS** [29] - constructing edge-preserving multi-scale image decompositions for progressive coarsening of images.
- **RGF** [9] - removing textures by controlling detail smoothing under a scale measure.
- **WMF** [30] - an efficient 100+ times faster weighted median filter.
- **LLF** [31] - fast local Laplacian filters for tone mapping.

**Image Restoration:** The goal of image restoration is to recover a clear image from a corrupted image. In this paper we deal with four representative tasks in this venue: super resolution [32], denoising [33], deblocking [34] and derain [35], which have been studied with deep learning based approaches extensively. Except for the derain task, all the others are tested on various parameter settings that indicate the corruption level of the input image.

- **super resolution** - increasing the resolution or enhancing the lost details from a low-resolution blurry image, which is controlled by a downsampling scale with bicubic interpolation.
- **denoising** - restore the clear image from a noisy image, which is composed of Gaussian white noise controlled by the Gaussian standard deviation.
- **deblocking** - recover the image details from a compressed JPEG image differentiated by the image quality factor.
- **derain** - removing rain streaks from a captured rainy image.

### 4.2 Implementation Details

**Dataset:** We take use of the 17k natural images in the PASCAL VOC dataset as the clear images to synthesize the ground truth training samples. The PASCAL VOC images are picked from Flicker, and consists of a wide range of viewing conditions. To evaluate our performance, 100 images from the dataset are randomly picked as the test data for the image filtering task. While for the restoration tasks, we take the well-known benchmark for each specific task for testing, which is specifically BSD100 (super resolution), BSD68 (denoise), LIVE1 (deblock), RAIN12 (derain). For the filtering task, we filter the natural images with the aforementioned algorithms to produce ground truth labels. As for the image restoration tasks, the clear natural image is taken as the target image while the synthesized corrupted image is used as input.

**Parameter Sampling:** To make our network able to handle continuous parameters, we generate training image pairs with a much broader scope of parameter values rather than a single one. We uniformly sample parameters in either the logarithm or the linear space depending on the specific application. Regarding the case of logarithm space, let $l$ and $u$ be the lower bound and upper bound of the parameter, the parameters are sampled as follows:

$$y = e^x, \text{ where } x \in [\ln l, \ln u] \tag{5}$$

In other words, we first uniformly sample $x$ between $\ln l$ and $\ln u$, then map it back by the exponential function, similar to the one used in [18]. Note that if the upper bound $u$ is tens or even hundreds of times larger than the lower bound $l$, the parameters are sampled in the logarithm space to balance their magnitudes, otherwise they are sampled in the linear space.

### 4.3 Results on the single parameterized operator

**Image Filtering.** We first evaluate the performance of six image operators with various controllable parameters individually. We train one network for each parameter value ($\gamma$) in one operator, and also train a network jointly on continuous random values sampled from the operator's parameter range, which can be inferred from the $\gamma$ column in Table 1. The performance of the two networks is evaluated on the test dataset with PSNR and SSIM error metrics. Since our goal is to measure the performance difference between these two strategies, we directly compute the absolute difference of their errors and demonstrate the results in Table 1.

As can be seen from the table, though our proposed framework trained with numerous parameter settings lags a little behind the one trained on a single parameter value, their difference is very small especially for the visually more important error metric SSIM. For each image operator, previous methods usually requires to train separate networks for each parameter value, while our proposed approach only trains one single network jointly. Moreover, these image operators are dedicated to different image processing applications, the proposed framework is still able to learn all of them well, which verifies the versatility and robustness of our strategy.

Some visual results of our proposed framework are shown in Figure 2. As can be seen, our single network trained on continuous random parameter values is capable of predicting high-quality smooth images of various strengths.

**Image Restoration.** We then evaluate the proposed framework on three popular image restoration tasks as shown in Table 2, which perform essentially different from image filtering. Unlike the above operators which employ the filtered images as target output to learn, this task takes the clear images as the ground truth output while the corrupted images as input. That is to say, as for the former task, given an input image, our network learns to obtain

TABLE 1
Quantitative absolute difference between the network trained with a *single* parameter value and *numerous* random values for each image operator.

| | $L_0$ | | | | WLS | | | | RTV | | | | RGF | | | | WMF | | | | LLF | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| metric | $\gamma$ | single | nume. | diff | $\gamma$ | single | nume. | diff | $\gamma$ | single | nume. | diff | $\gamma$ | single | nume. | diff | $\gamma$ | single | nume. | diff | $\gamma$ | single | nume. | diff |
| PSNR | 0.002 | 40.69 | 39.46 | 1.23 | 0.100 | 44.00 | 42.12 | 1.88 | 0.002 | 41.11 | 40.66 | 0.45 | 1.00 | 41.77 | 37.03 | 4.74 | 1.00 | 39.06 | 36.79 | 2.27 | 2 | 38.00 | 37.83 | 0.17 |
| | 0.004 | 38.96 | 38.72 | 0.24 | 0.215 | 43.14 | 42.64 | 0.50 | 0.004 | 40.91 | 41.10 | 0.19 | 3.25 | 38.36 | 38.27 | 0.09 | 3.25 | 39.78 | 38.76 | 1.02 | 3 | 34.64 | 35.71 | 1.07 |
| | 0.020 | 36.07 | 35.71 | 0.36 | 1.000 | 41.93 | 41.63 | 0.30 | 0.010 | 40.50 | 41.07 | 0.57 | 5.50 | 38.11 | 38.35 | 0.24 | 5.50 | 39.94 | 38.53 | 1.41 | 5 | 32.34 | 32.29 | 0.05 |
| | 0.093 | 33.08 | 31.92 | 1.16 | 4.641 | 39.42 | 39.64 | 0.22 | 0.022 | 41.07 | 40.77 | 0.30 | 7.75 | 37.65 | 37.99 | 0.34 | 7.75 | 40.06 | 39.20 | 0.86 | 7 | 30.11 | 29.91 | 0.20 |
| | 0.200 | 31.75 | 30.43 | 1.32 | 10.00 | 39.13 | 38.51 | 0.62 | 0.050 | 40.73 | 39.18 | 1.55 | 10.0 | 37.52 | 37.08 | 0.44 | 10.0 | 39.49 | 38.72 | 0.77 | 8 | 29.53 | 28.95 | 0.58 |
| | ave. | 36.11 | 35.25 | **0.86** | ave. | 41.52 | 40.91 | **0.61** | ave. | 40.86 | 40.55 | **0.31** | ave. | 38.68 | 37.74 | **0.93** | ave. | 39.66 | 38.40 | **1.26** | ave. | 32.93 | 32.94 | **0.01** |
| SSIM | 0.002 | 0.989 | 0.988 | 0.001 | 0.100 | 0.994 | 0.993 | 0.001 | 0.002 | 0.987 | 0.988 | 0.001 | 1.00 | 0.994 | 0.981 | 0.013 | 1.00 | 0.985 | 0.972 | 0.013 | 2 | 0.992 | 0.992 | 0 |
| | 0.004 | 0.986 | 0.987 | 0.001 | 0.215 | 0.993 | 0.993 | 0 | 0.004 | 0.989 | 0.990 | 0.001 | 3.25 | 0.986 | 0.986 | 0 | 3.25 | 0.985 | 0.979 | 0.006 | 3 | 0.988 | 0.990 | 0.02 |
| | 0.020 | 0.982 | 0.981 | 0.001 | 1.000 | 0.992 | 0.991 | 0.001 | 0.010 | 0.990 | 0.991 | 0.001 | 5.50 | 0.985 | 0.986 | 0.001 | 5.50 | 0.986 | 0.981 | 0.005 | 5 | 0.983 | 0.984 | 0.01 |
| | 0.093 | 0.977 | 0.973 | 0.004 | 4.641 | 0.987 | 0.989 | 0.002 | 0.022 | 0.992 | 0.992 | 0 | 7.75 | 0.984 | 0.985 | 0.001 | 7.75 | 0.986 | 0.985 | 0.001 | 7 | 0.977 | 0.977 | 0 |
| | 0.200 | 0.973 | 0.968 | 0.005 | 10.00 | 0.986 | 0.987 | 0.001 | 0.050 | 0.992 | 0.990 | 0.002 | 10.0 | 0.984 | 0.982 | 0.002 | 10.0 | 0.986 | 0.984 | 0.002 | 8 | 0.976 | 0.974 | 0.02 |
| | ave. | 0.981 | 0.979 | **0.002** | ave. | 0.990 | 0.990 | **0** | ave. | 0.990 | 0.990 | **0** | ave. | 0.986 | 0.984 | **0.002** | ave. | 0.985 | 0.980 | **0.005** | ave. | 0.983 | 0.983 | **0** |

TABLE 2
Quantitative absolute difference in PSNR (above) and SSIM (bottom) between the network trained on a *single* parameter value and *numerous* random values on the three image restoration tasks. Their parameters specifically mean downsampling scale ($s$), Gaussian standard deviation ($\sigma$) and JPEG quality ($q$).

| Super Resolution | | | | Denoising | | | | Deblock | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s$ | single | nume. | diff | $\sigma$ | single | nume. | diff | $q$ | single | nume. | diff |
| 2 | 31.78 | 31.62 | 0.16 | 15 | 31.17 | 31.07 | 0.10 | 10 | 29.26 | 29.17 | 0.09 |
| 3 | 28.78 | 28.76 | 0.02 | 25 | 28.94 | 28.98 | 0.04 | 20 | 31.49 | 31.43 | 0.06 |
| 4 | 27.31 | 27.31 | 0 | 50 | 26.22 | 26.14 | 0.08 | | | | |
| ave. | 29.29 | 29.23 | **0.06** | ave. | 28.77 | 28.73 | **0.04** | ave. | 30.37 | 30.30 | **0.07** |
| 2 | 0.894 | 0.892 | 0.002 | 15 | 0.881 | 0.883 | 0.002 | 10 | 0.817 | 0.817 | 0 |
| 3 | 0.798 | 0.796 | 0.002 | 25 | 0.821 | 0.822 | 0.001 | 20 | 0.881 | 0.882 | 0.001 |
| 4 | 0.728 | 0.726 | 0.002 | 50 | 0.722 | 0.718 | 0.004 | | | | |
| ave. | 0.806 | 0.804 | **0.002** | ave. | 0.808 | 0.807 | **0.001** | ave. | 0.849 | 0.849 | **0** |

TABLE 3
Quantitative comparison between the network trained on a *single* parameter setting and *numerous* random settings under the condition of multiple input parameters. Their absolute difference is shown besides the value of *nume*. The results are tested by fixing one parameter while varying another.

| RTV ($\lambda$ = 0.01) | | | | RTV ($\sigma$ = 3) | | | |
|---|---|---|---|---|---|---|---|
| $\sigma$ | single | nume. | diff | $\lambda$ | single | nume. | diff |
| 2 | 40.53 | 40.39 | 0.14 | 0.002 | 41.11 | 40.17 | 0.94 |
| 3 | 39.52 | 40.76 | 1.24 | 0.004 | 40.91 | 40.78 | 0.13 |
| 4 | 41.19 | 41.06 | 0.13 | 0.010 | 40.50 | 40.76 | 0.26 |
| 5 | 41.29 | 41.26 | 0.03 | 0.022 | 41.07 | 40.45 | 0.62 |
| 6 | 41.81 | 41.19 | 0.62 | 0.050 | 40.73 | 38.52 | 2.21 |
| ave | 40.86 | 40.93 | **0.06** | ave | 40.86 | 40.14 | **0.72** |

different filtering effects, while regarding the latter one, our model learns to recover from different corrupted input images.

As shown in Table 2, our results trained jointly on continuous random parameter values also show no big difference from the one trained solely on an individual parameter value, which further validate our algorithm in a broader image processing literature. Some corresponding visual results of our proposed framework are shown in Figure 3.

### 4.4 Extension to multiple input parameters

Except for experimenting on a single input parameter, we also demonstrate the experiments on inputting multiple types of parameters, which is still very common for many image processing tasks.

In this section, we evaluate the performance on the famous texture removal tool RTV [8]. For this image filter, we leverage $\lambda$ which balances between the data prior term and smoothness term in its energy function as one parameter, and $\sigma$ which controls the spatial scale for computing the windowed variation and is even more effective in removing textures. To generate the training samples, we randomly sample these two parameters. Therefore, the input parameter $\overrightarrow{\gamma}$ of the *weight learning* network is a two-element vector $[\lambda, \sigma]$.

To evaluate the performance of the proposed framework on this two dimensional parameter space compared with the single parameter setting case, we sample a few parameters along one dimension while fixing another as shown in Table 3. We can see that for most of the 10 parameter settings, all achieve very close results to the one trained with an individual parameter setting. This verifies the effectiveness of the proposed network on this more difficult case.

### 4.5 Extension to joint training of multiple image operators

Intuitively, another challenging case for our proposed framework is to incorporate multiple distinct image operators into a single learned neural network, which is much harder to be trained due to their different implementation details and purposes. To explore the potential of our proposed neural network, we experiment by jointly training over (*i*). 6 filtering based operators, (*ii*). 4 image restoration operators or (*iii*). all the 10 different operators altogether. To generate training images of each image operator, we sample random parameter values continuously within its parameter range. Since there is no parameter tuning for the derain task, we leverage its default parameter setting for training.

The input to the *weight learning* network now takes two parameters, one indicates the specific image operator while the other is the random parameter values assigned to the specified filter. These 10 image operators are denoted simply by 10 discrete values that range from 0.1 to 1.0 in the input parameter vector.

TABLE 4
Numerical results (PSNR (above) and SSIM (bottom)) of our proposed framework jointly trained over different number of image operators (#operators). "6/4" refers to the results jointly trained over either the front 6 filtering based approaches or the last 4 restoration tasks. "10" is the results of jointly training all 10 tasks.

| #ope. | $L_0$ | WLS | RTV | RGF | WMF | LLF | SR | denoise | deblock | derain | ave. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 35.25 | 40.91 | 40.55 | 37.74 | 38.40 | 32.94 | 29.13 | 28.70 | 30.21 | 29.86 | **34.36** |
| 6/4 | 33.27 | 37.39 | 37.00 | 35.41 | 36.06 | 30.08 | 28.89 | 28.67 | 30.10 | 30.32 | **32.72** |
| 10 | 32.67 | 36.59 | 36.03 | 34.64 | 35.08 | 29.77 | 29.69 | 30.45 | 28.53 | 28.36 | **32.18** |
| 1 | 0.979 | 0.991 | 0.990 | 0.984 | 0.980 | 0.984 | 0.804 | 0.804 | 0.847 | 0.893 | **0.925** |
| 6/4 | 0.969 | 0.980 | 0.979 | 0.974 | 0.967 | 0.976 | 0.797 | 0.800 | 0.842 | 0.893 | **0.918** |
| 10 | 0.965 | 0.978 | 0.975 | 0.969 | 0.962 | 0.971 | 0.837 | 0.895 | 0.789 | 0.789 | **0.913** |

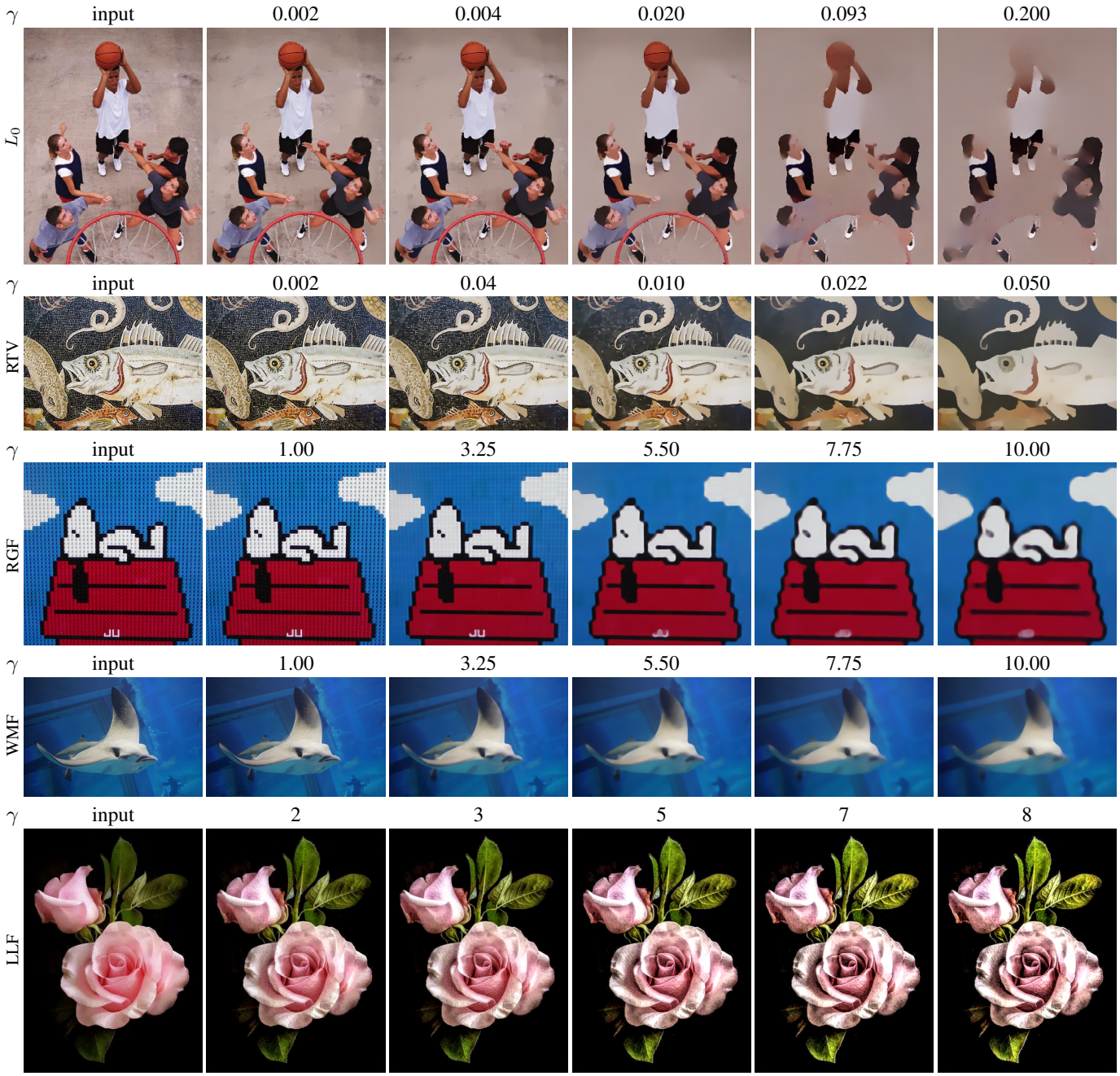



Fig. 2. Visual examples produced by our framework trained on continuous parameter settings of six image filters independently. Note all the smooth images for one filter are generated by a single network.

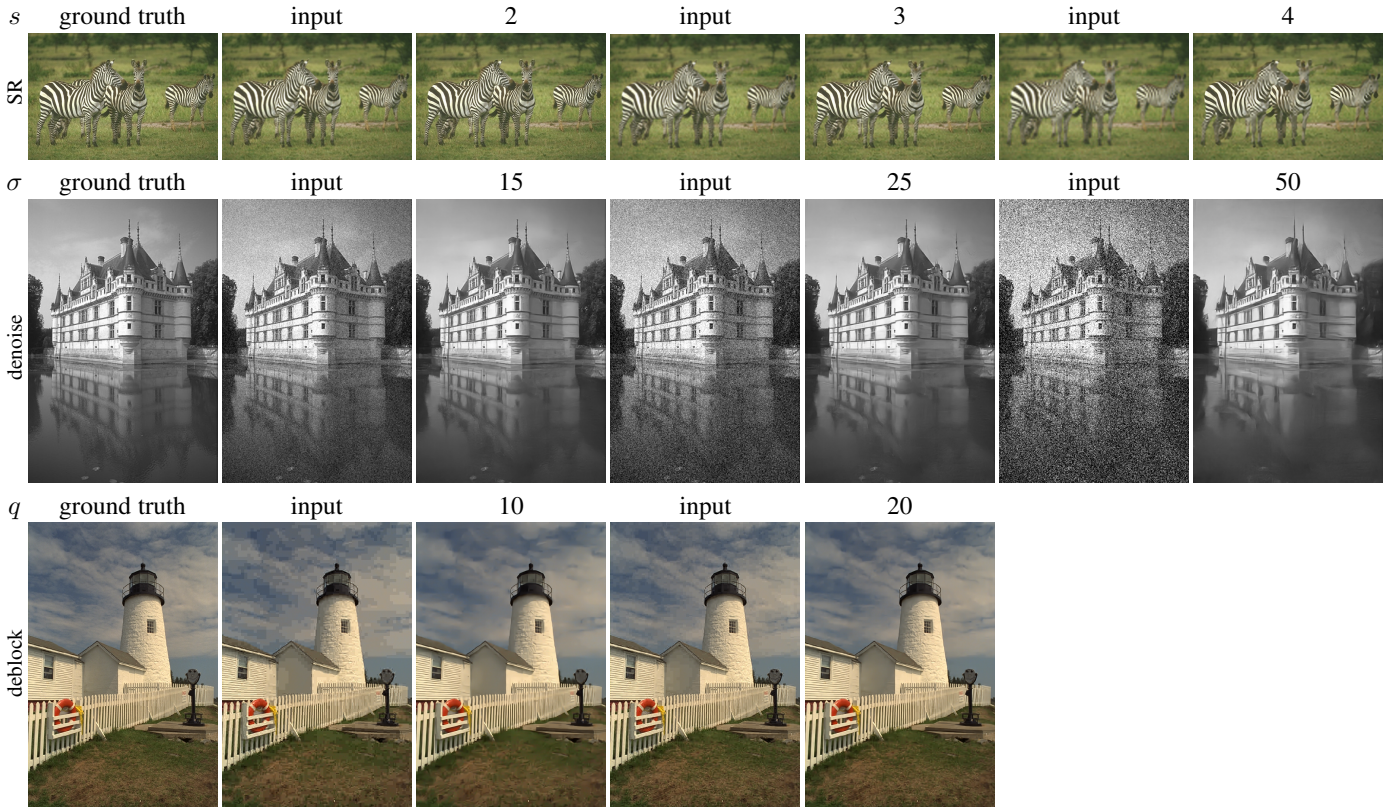



Fig. 3. Visual examples produced by our framework trained on continuous parameter settings of three image restoration tasks independently. Note all the smooth images for one filter are generated by a single network.

Since the absolute parameter range may differ a lot from operator to operator, for example, [2,4] for super resolution and [0.002,0.2] for $L_0$ filter, we rescale the parameters in all the operators into the same numerical range to enable consistent back-propagated gradient magnitude.

As shown in Table 4, training on each individual image operator achieves the highest numerical score (#ope.=1), which is averaged over multiple different parameter settings just like in previous tables. While jointly training over either 6 image filters or 4 restoration tasks (#ope.=6/4), even for the case where all 10 image operators are jointly trained (#ope.=10), their average performance degrades but still achieves close results to the best score. It means with the same network structure, our framework is able to incorporate all these different image operators together into a single network without losing much accuracy.

Note that for the image restoration tasks, it is more meaningful not to specify parameters since in real life, users usually do not know the corruption degree of the input image. Therefore, we disable specifying parameters for the four restoration operators in this experiment. Surprisingly, we do not observe much performance degradation with this modification. Though it degrades the necessity of learning continuous parameter settings for image restoration tasks, it still makes a lot of sense by jointly training multiple image operators.

### 4.6 Comparison with Baseline [18]

We compare our proposed framework with one naive approach that employs only the *base* network with additional input channels as [18], which indicates the parameter values and image operators separately. Each additional channel is occupied with a single value.

The results are shown in Table 5, which trains three different image operators including both image filtering and restoration tasks together. We can see that the baseline from [18] lags behind us on all the different parameter settings across all three image operators. The potential reason for this phenomenon could be that it is more difficult to learn unified convolution weights to be suitable for tasks different in both goals and implementations. By contrast, the convolutional weights of our *base* network are adaptively changed for different tasks and input parameters. Theoretically speaking, our learned network should have the capability to express the base network with more numerous possibilities.

TABLE 5
Numerical comparison (PSNR) between our proposed framework and the baseline [18] that are all jointly trained over three image operators.

| RTV | | | Super resolution | | | Denoising | | |
|---|---|---|---|---|---|---|---|---|
| $\lambda$ | ours | baseline | $s$ | ours | baseline | $\sigma$ | ours | baseline |
| 0.002 | **36.60** | 35.11 | 2 | **30.83** | 30.59 | 15 | **30.90** | 30.67 |
| 0.004 | **37.14** | 35.25 | 3 | **28.30** | 28.13 | 25 | **28.63** | 28.50 |
| 0.010 | **37.18** | 35.01 | 4 | **26.95** | 26.82 | 50 | **25.66** | 25.57 |
| 0.022 | **36.83** | 34.89 | | | | | | |
| 0.050 | **35.60** | 32.75 | | | | | | |
| ave. | **36.67** | 34.60 | ave. | **28.69** | 28.51 | ave. | **28.39** | 28.24 |

## 5 Experiments on the Real-time Extension

In this section, we evaluate the real-time extension of our proposed framework with many recent filtering and restoration approaches. Most compared approaches design some complex and advanced deep neural network to achieve the optimal performance for their

specific dedicated tasks. In order to balance between the performance and computation costs, we slightly modify the structure of $\mathcal{N}_{base}$ by utilizing the depth-wise convolution [36] and increasing the intermediate feature channels (64 to 128) in this section. We validate and study the effectiveness of the proposed framework, which not only runs in real time for parameter tuning on both GPU and CPU devices but also achieves state-of-the-art performance compared to many recent deep learning approaches.

### 5.1 Image filtering

We compare our proposed framework with two recent state-of-the-art approaches [4], [5], which share the closest spirit as ours in reproducing the image filters as well as possible while running in real-time by varying between different image operators.

To conduct on a broader scope of image operators like the other two approaches, besides the aforementioned six image filters, we further add four operators with more different visual effects for a fair comparison as follows.

- **LLF remapping** [31] - fast local Laplacian filters for tone mapping by leveraging a remapping function.
- **WLS enhancement** [29] - detail exaggeration implemented via multi-scale image decompositions.
- **Stylization** [31] - transfer the look of one photographers masterpiece onto another photo via fast local Laplacian filters.
- **Abstraction** [46] - pencil drawing by combining the tone and stroke structures, which complement each other in generating visually constrained results.

Note the six image filters mentioned in Section 4 contains controllable parameters that indicate different visual effects, while each of the above four operators represents a unique fixed visual effect. All these ten image operators are fed through the networks for joint training. The input parameters are sampled similarly as in section 4.5.

In Table 6, we demonstrate the numerical results of our proposed framework compared with the real-time competitors BGU [4] and DBL [5]. For a fair comparison, the evaluation is conducted on one parameter setting (the default one) for each image operator following the other two approaches. As can be seen from the numerical errors (PSNR and SSIM), our results achieve significantly better results. The average PSNR over 10 image operators are about 5dB larger than the second best competitor.

Note both BGU and DBL learn to reproduce one specific operator with the default parameter setting, while *our approach learns all the 10 image operators with their full range of parameter values jointly within one single network*, which is much more difficult and challenging.

Finally we evaluate our running time. Since the modified fast version of our proposed framework only requires to run the last convolutional layer to vary between different image operators, it becomes very efficient to run this model under either GPU or CPU devices. As implemented with multi-core CPUs (NNPACK) under the MXNet framework, it takes only 0.5 ms by varying the visual effects for an image of resolution 1024 x 1024. With a modern Nvidia TITAN Xp GPU, it also only takes 0.4 ms, which is fast enough for many real-life scenarios.

### 5.2 Image restoration

Furthermore we compare our framework on the four aforementioned image restoration tasks: super resolution, denoising, deblocking and derain with many recent restoration methods as shown from Table 7 to Table 10. Except for the derain task, all the others are tested on various parameter settings that indicate the corruption level of the input image. Though these existing algorithms are dedicated on each specific field with specifically designed algorithms or network structures, our proposed framework is still able to achieve very competitive results on all these tasks. Need to note that all these tasks are jointly trained with one single network by our approach.

Given each new example for the restoration tasks, it needs to rerun the whole network from the beginning. Therefore it's not a very valid case to enable real-time performance for practical usage. However, theoretically this experiment still demonstrates the extraordinary ability of deep networks to incorporate different restoration capabilities within a very limited parameterized operation (the last convolution layer).

## 6 Understanding and Analysis

To better understand the *base* network $\mathcal{N}_{base}$ and the *weight learning* network $\mathcal{N}_{weight}$, we analyze some meaningful behavior behind our framework in this section.

### 6.1 The effective receptive field

In neuroscience, the receptive field is the particular region of the sensory space in which a stimulus will modify the firing of one specific neuron. The large receptive field is also known to be important for modern convolutional networks. Different strategies are proposed to increase the receptive field, such as deeper network structure or dilated convolution. Though the theoretical receptive field of one network may be very large, the real effective receptive field may vary with different learning targets. So how is the effective receptive field of $\mathcal{N}_{base}$ changed with different parameters $\overrightarrow{\gamma}$ and $\mathcal{I}$ ? Here we use $L_0$ smoothing [26] as the default example operator.

In Fig. 4, we study the effective receptive field of a non-edge point, a moderate edge point, and a strong edge point with different smoothing parameters $\lambda$ respectively. To obtain the effective receptive field for a specific spatial point $p$, we first feed the input image into the network to get the smoothing result, then propagate the gradients back to the input while masking out the gradient of all points except $p$. Only the points whose gradient value is large than $0.025 * grad_{max}$ ($grad_{max}$ is the maximum gradient value of input gradient) are considered within the receptive field and marked as green in Fig. 4. From Fig. 4, we observe three important phenomena: 1) For a non-edge point, the larger the smoothing parameter $\lambda$ is, the larger the effective field is, and most effective points fall within the object boundary. 2) For a moderate edge point, its receptive field stays small until a relatively large smoothing parameter is used. 3) For a strong edge point, the effective receptive field is always small for all the different smoothing parameters. It means, on one hand, the *weight learning* network $\mathcal{N}_{weight}$ can dynamically change the receptive field of $\mathcal{N}_{base}$ based on different smoothing parameters. On the other hand, the *base* network $\mathcal{N}_{base}$ itself can also adaptively change its receptive field for different spatial points.

### 6.2 Investigation into the learned weight

In this subsection, we discuss about the relationship between different learned convolutional kernels, and alternatives of the learned parameter position and type to better understand our proposed framework.

TABLE 6
Quantitative comparison with state-of-the-art approaches in reproducing image operators.

| metric | method | $L_0$ | WLS | RTV | RGF | WMF | LLF | LLF remap | WLS enhance | Stylization | Abstraction | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PSNR | BGU | 31.76 | 27.03 | 26.15 | 22.71 | 21.27 | 26.97 | 33.05 | 26.93 | 14.31 | 16.11 | **24.62** |
| | DBL | 28.67 | 30.63 | 28.52 | 27.11 | 26.88 | 25.13 | 29.34 | 28.29 | 25.08 | 19.61 | **26.92** |
| | Ours | 31.81 | 36.59 | 34.28 | 33.29 | 34.00 | 29.74 | 32.41 | 34.40 | 26.66 | 25.61 | **31.87** |
| SSIM | BGU | 0.912 | 0.915 | 0.848 | 0.776 | 0.765 | 0.936 | 0.978 | 0.931 | 0.673 | 0.427 | **0.816** |
| | DBL | 0.852 | 0.890 | 0.826 | 0.805 | 0.786 | 0.899 | 0.945 | 0.944 | 0.887 | 0.502 | **0.833** |
| | Ours | 0.946 | 0.971 | 0.948 | 0.945 | 0.940 | 0.967 | 0.969 | 0.986 | 0.927 | 0.835 | **0.943** |

TABLE 7
Quantitative results (PSNR/SSIM) of the JPEG deblocking task on the LIVE1 benchmark.

| Quality | JPEG | ARCNN [34] | TNRD [37] | DnCNN [38] | MemNet [39] | Ours |
|---|---|---|---|---|---|---|
| 10 | 27.77/0.7730 | 28.96/0.8076 | 29.15/0.8111 | 29.19/0.8123 | 29.45/0.8193 | 29.31/0.8170 |
| 20 | 30.07/0.8512 | 31.29/0.8733 | 31.46/0.8769 | 31.59/0.8802 | 31.83/0.8846 | 31.60/0.8816 |

TABLE 8
Quantitative results (PSNR/SSIM) of the image super resolution task on the BSD100 benchmark.

| Scale | Bicubic | SRCNN [15] | VDSR [40] | DRCN [41] | DnCNN [38] | MemNet [39] | Ours |
|---|---|---|---|---|---|---|---|
| 2 | 29.56/0.8431 | 31.36/0.8879 | 31.90/0.8960 | 31.85/0.8942 | 31.90/0.8961 | 32.08/0.8978 | 31.67/0.8934 |
| 3 | 27.21/0.7385 | 28.41/0.7863 | 28.82/0.7976 | 28.80/0.7963 | 28.85/0.7981 | 28.96/0.8001 | 28.76/0.7969 |
| 4 | 25.96/0.6675 | 26.90/0.7101 | 27.29/0.7251 | 27.23/0.7233 | 27.29/0.7253 | 27.40/0.7281 | 27.29/0.7257 |

TABLE 9
Quantitative results (PSNR) of the image denoising task on the BSD68 benchmark.

| $\sigma$ | BM3D [42] | WNNM [43] | TNRD [37] | DCDP [44] | Ours |
|---|---|---|---|---|---|
| 15 | 31.07 | 31.37 | 31.42 | 31.63 | 31.48 |
| 25 | 28.57 | 28.83 | 28.92 | 29.15 | 29.11 |
| 50 | 25.62 | 25.87 | 25.97 | 26.19 | 26.22 |

TABLE 10
Quantitative results (PSNR) of the derain task on the RAIN12 benchmark.

| DerainNet [45] | DnCNN [38] | Ours |
|---|---|---|
| 28.94 | 30.90 | 30.08 |

#### 6.2.1 Relationship of the learned convolution kernels within a jointly trained network

When joint training for multiple parameterized operators, the learned convolution weights can be generally classified into two classes: kernels generated by different parameter values of a single image operator, and kernels generated by different image operators. We analyze both groups of kernels on the model trained on 10 image operators which is introduced in subsection 4.5. In this case, the input to the weight learning network takes two parameters, hence the learned convolution weights for a specific layer $i$ in the base network should be,

$$W_i = \gamma_1 A_{i1} + \gamma_2 A_{i2} + B_i \tag{6}$$

where $\gamma_1$ refers to the input parameter value of a specific operator, and $\gamma_2$ indicates the type of the operator, which is defined by ten discrete numbers that range from "0.1" to "1.0" for different operators separately. $A_{i1}$ and $A_{i2}$ are its corresponding weights in the fully connected layer. Therefore, for a single image operator, $\gamma_2 A_{i2} + B_i$ is a fixed value and the only modification to its different parameter values is $\gamma_1 A_{i1}$, which scales a high-dimension value. That is to say, each time when one adjusts the operator parameter by $\gamma_1$, the learned convolution weights are only shifted to some extent in a fixed high-dimensional direction. Similar analysis also applies to the transformation of different operators.

We visualize the learned convolution kernels via t-SNE in Figure 5. Each color indicates one image operator, and for each operator, we randomly generate 500 groups of convolution weights with different parameters. As can be seen, the distance of every two adjacent operator is almost the same, it shifts along the x dimension for a fixed distance. For a single filter, while adjusting the parameters continuously, the convolution weights shift along the y dimension. This figure just conforms to our analysis about the convolution weights in the high-dimensional space. It is very surprising that all different kinds of learned convolution weights can be related with a high-dimensional vector, and the transformation between them can be represented by a very simple linear function.

For better theoretical understanding, we link our proposed framework with evolutionary computing, where it is often difficult to directly get an optimal solution in a large search space. To help the searching process, some structure constraints are added for the target weight space, like Discrete Cosine Transform (DCT) used in [47]. Similarly, for our task, the weight space of the *base* network $\mathcal{N}_{base}$ is also very large, and there may exist many different parameter solutions (local minima) which can obtain the same/similar performance (analyzed in the following Section 6.2.2). When the *weight learning* network $\mathcal{N}_{weight}$ follows

(a) Input image (b) $\lambda = 0.01$ (c) $\lambda = 0.02$ (d) $\lambda = 0.03$ (e) $\lambda = 0.04$

Fig. 4. Effective receptive field of $L_0$ smoothing for different spatial positions and parameter $\lambda$. The top to bottom indicate the effective receptive field of a non-edge point, a moderate edge point, and a strong edge point.

TABLE 11
Comparison between the statistics of convolution kernels learned with random parameter values and a single parameter value. The numbers are generated based on the WLS filter while $\lambda$ equals to 10.

| layer index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| correlation | 0.005 | -0.001 | 0.008 | 0.004 | -0.005 | 0.007 | 0.008 | 0.010 | -0.002 | -0.012 |
| mean (nume.) | 0.219 | 0.066 | 0.004 | -0.069 | -0.227 | -0.183 | -0.214 | -0.131 | -0.152 | -0.141 |
| mean (fixed) | 0.542 | 0.801 | 0.604 | -0.571 | -2.090 | -0.685 | -0.520 | -1.638 | -1.604 | -1.326 |
| variance (nume.) | 15.514 | 12.166 | 15.511 | 18.231 | 17.676 | 20.408 | 16.578 | 19.420 | 16.447 | 18.292 |
| variance (fixed) | 542.28 | 373.70 | 490.53 | 419.55 | 482.71 | 559.00 | 532.14 | 505.67 | 471.50 | 437.69 |
| layer index | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| correlation | 0.011 | 0.184 | -0.014 | 0.001 | 0.012 | 0.024 | 0.011 | -0.001 | -0.071 | -0.017 |
| mean (nume.) | -0.296 | -0.168 | -0.084 | -0.310 | -0.137 | -0.201 | 0.003 | -0.239 | -0.407 | 0.634 |
| mean (fixed) | -1.117 | -1.134 | -2.417 | -1.437 | -1.043 | -1.977 | -0.596 | 1.898 | -3.203 | 3.358 |
| variance (nume.) | 17.704 | 20.141 | 18.767 | 21.080 | 28.461 | 23.795 | 19.475 | 14.592 | 13.703 | 4.220 |
| variance (fixed) | 549.51 | 500.61 | 585.75 | 507.40 | 792.44 | 588.06 | 581.66 | 501.08 | 487.62 | 140.33 |

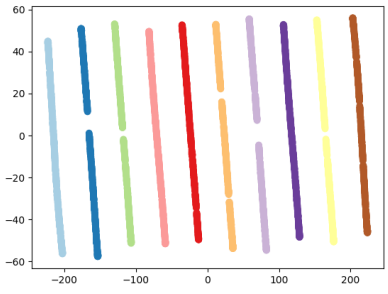



Fig. 5. T-SNE illustration of the learned weights of the 2nd convolution layer in the base network. The displayed convolution weights are generated by the jointly trained network with 10 image operators. Each color indicates one specific operator. We also observe similar visualized results for the other convolution layers.

the linear formulation, like DCT used in [47], it is an extra constraint, which constrains that the weight spaces of different parameters $\overrightarrow{\gamma}$ are related by the affine transform.

#### 6.2.2 *Difference of the learned convolution kernels between jointly trained network and solely trained network*

To analyse the difference of the convolution weights between networks jointly trained on numerous random parameter values ("nume.") and a single parameter value ("fixed"), we compute their correlation coefficient, individual mean and variance for each layer as shown in Table 11.

As can be seen, the correlation coefficient is almost 0 everywhere, which means there is no linear relationship between the two groups of convolution kernels. The absolute mean and variance of jointly trained network is also significantly smaller than that of the solely trained network. Therefore, in each way, their learned convolution weights are very different from each other, even if the learned smoothing effect is almost the same (PSNR/SSIM: 35.51dB/0.983 (nume.) *vs.* 35.83dB/0.982 (fixed)).

This simple experiment further verifies the huge solution space in the form of learned convolution kernels. Two exactly same results may be represented by very different convolution weights. The linear transformation in our proposed weight learning network actually connects all the different image operators and constrains

TABLE 12
Numerical analysis of learned parameter type and position. "norm" and "conv" indicates the learned parameter type in either the instance normalization layers or convolution layers. The number in the brackets behind the parameter type refer to the position where the learned layer is located, "1" for the 1st layer and "all" for all the layers.

| metric | method | $L_0$ | WLS | RTV | RGF | WMF | LLF | LLF remap | WLS enhance | Stylization | Abstraction | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PSNR | norm(1) | 31.06 | 34.86 | 33.75 | 32.87 | 33.49 | 29.22 | 32.21 | 32.59 | 27.26 | 24.81 | **31.21** |
| | norm(7) | 30.86 | 34.75 | 33.54 | 32.97 | 33.39 | 29.24 | 31.99 | 32.79 | 27.07 | 24.81 | **31.14** |
| | norm(14) | 30.64 | 34.31 | 33.06 | 32.42 | 33.21 | 29.15 | 31.80 | 32.53 | 27.05 | 24.83 | **30.90** |
| | norm(19) | 28.53 | 31.22 | 29.46 | 29.72 | 30.69 | 27.03 | 30.36 | 29.78 | 25.57 | 24.38 | **28.67** |
| | conv(19) | 30.57 | 34.17 | 32.45 | 31.84 | 32.96 | 28.80 | 32.07 | 31.95 | 27.46 | 26.30 | **30.85** |
| | conv channel1(19) | 29.54 | 32.80 | 31.14 | 30.94 | 31.97 | 28.09 | 31.53 | 31.39 | 27.07 | 25.89 | **30.03** |
| | conv channel2(19) | 24.87 | 26.63 | 24.86 | 26.17 | 27.48 | 23.26 | 24.86 | 25.29 | 21.82 | 24.45 | **24.96** |
| | norm(all) | 31.71 | 35.51 | 34.31 | 33.10 | 33.96 | 29.56 | 32.45 | 33.11 | 27.32 | 25.21 | **31.62** |
| | conv(all) | 31.64 | 35.02 | 33.67 | 32.81 | 33.97 | 29.59 | 32.52 | 32.69 | 28.12 | 26.35 | **31.63** |
| SSIM | norm(1) | 0.949 | 0.971 | 0.959 | 0.954 | 0.948 | 0.966 | 0.981 | 0.982 | 0.923 | 0.819 | **0.945** |
| | norm(7) | 0.945 | 0.969 | 0.958 | 0.955 | 0.948 | 0.967 | 0.980 | 0.982 | 0.917 | 0.812 | **0.943** |
| | norm(14) | 0.935 | 0.963 | 0.942 | 0.947 | 0.945 | 0.966 | 0.979 | 0.980 | 0.917 | 0.816 | **0.939** |
| | norm(19) | 0.871 | 0.890 | 0.809 | 0.881 | 0.876 | 0.947 | 0.970 | 0.965 | 0.889 | 0.796 | **0.889** |
| | conv(19) | 0.931 | 0.948 | 0.913 | 0.932 | 0.932 | 0.965 | 0.979 | 0.989 | 0.911 | 0.826 | **0.932** |
| | conv channel1(19) | 0.905 | 0.927 | 0.862 | 0.917 | 0.921 | 0.957 | 0.976 | 0.974 | 0.906 | 0.822 | **0.916** |
| | conv channel2(19) | 0.741 | 0.727 | 0.629 | 0.700 | 0.737 | 0.844 | 0.899 | 0.914 | 0.848 | 0.741 | **0.778** |
| | norm(all) | 0.954 | 0.972 | 0.960 | 0.956 | 0.953 | 0.970 | 0.982 | 0.984 | 0.924 | 0.823 | **0.947** |
| | conv(all) | 0.954 | 0.969 | 0.952 | 0.953 | 0.954 | 0.969 | 0.982 | 0.982 | 0.925 | 0.833 | **0.947** |

their learned convolution weights in a limited high dimensional space.

#### 6.2.3 Study of the learned parameter position and type

For the real-time extension, to conduct a comprehensive study of the effects of where and what the learned controllable parameters are, we start by three experimental settings, (i). learned instance normalization parameters at different network depth. (ii). learned different types of convolution parameters at a fixed network depth. (iii). learned parameters in either all the convolutional or instance normalization layers.

We compare the performance of different network settings in Table 12. These experiments are made by reproducing the 10 image filters with the base network described in Section 3.2.

**(i)**. At first, we adjust the position of learned controllable parameters with four discrete network depth. It ranges from the first (1) instance normalization layer to the last (19) layer (the 20th convolutional layer is not followed by instance normalization). From the table, we can see that as the learned parameters become deeper in the network, the performance degrades. It's a reasonable phenomenon, since the deeper the parameters are, the smaller the network capacity to differentiate various image operators is, and thus it's harder to incorporate all the operators in the network.

We can also observe that the average error (PSNR) over all the 10 image operators only decreases by about 2.5dB from the first layer to the last layer, however it significantly improves the running speed when varying between different image operators in the network.

**(ii)**. To explore the influence of other types of parameters in the network to separate these image operators, we replace the controllable parameters in the instance normalization layer with the ones in the convolutional layer.

Each convolutional layer contains a $M\times N\times k\times k$ weight tensor and a $M$-length bias vector. $M$ is the number of output feature channels, $N$ is the number of input feature channels, $k$ is the kernel size. $M$ equals to $N$ for the intermediate convolutional layers in our baseline network. Since the convolutional layer is followed by an instance normalization layer, the added bias in the convolutional layer will be balanced out by the following normalization operation and becomes useless. Therefore we experiment only by varying the parameters in the convolutional weights.

We study with three types of learned convolutional parameters, which are separately: "conv", all the weight kernels ($M\times N\times k\times k$); "conv channel1", one slice of weight kernels ($M\times 1\times k\times k$); "conv channel2", another slice of weight kernels ($1\times N\times k\times k$). The learned convolutional layer shares the same network depth as the last instance normalization layer.

As can be seen from the table, learning all the weight kernels in the convolutional layer (conv) achieves the highest performance, which is very reasonable. While learning all the convolutional kernels that are only related to one single input channel (conv channel1), the performance degrades by about 0.8dB. Learning all the parameters that are related to one single output channel (conv channel2) obtains the worst performance. Note even if "conv channel1" and "conv channel2" share the same number of controllable parameters, all the learned parameters in "conv channel2" only result in adjustment of a single channel of output feature map, which influence the following layers much less than "conv channel1" and hence achieve worse results.

Interestingly, by learning the convolutional parameters (conv and conv channel1) achieves better results than learning instance normalization parameters at the same network depth. This may be because the convolutional layer is ahead of the instance normalization, and the learned convolutional parameters (even $M\times k\times k$ for "conv channel1", $k$ equals to 3 in our case) is much more than the normalization's ($M\times 2$).

**(iii)**. To further explore the difference between learning convolution and normalization layers, we try to learn the parameters in either all the convolutional layers or all the instance normalization layers in the network, which should theoretically achieves the best performance of incorporating all the image operators.

As shown in the "conv(all)" and "norm(all)" rows, surprisingly they achieve almost the same performance on the either PSNR or SSIM error metric. This means it already touches the top

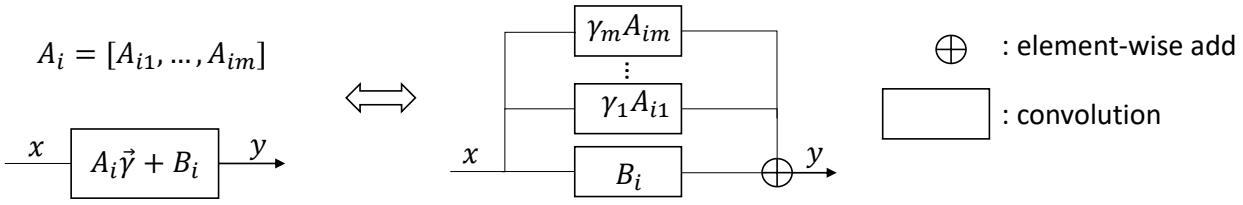



Fig. 6. Equivalent analysis of the connection between the *base* network $\mathcal{N}_{base}$ and the *weight learning* network $\mathcal{N}_{weight}$. One convolution layer whose weights are learnt by the fc layer is exactly equivalent to a multi-path convolution blocks.

TABLE 13
Quantitative evaluation of a few variants of the proposed network trained on the WLS filter [29]. We experiment separately by training only the base network on a fixed single parameter value ("single (base)"), extending the weight learning network to two or more fully connected layers trained on numerous random parameter values with ("nume. (fc2R)") and without ("nume. (fc2)") ReLU layers inbetween.

| $\lambda$ | single | | single (base) | | nume. | | nume. (fc2R) | | nume. (fc2) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM |
| 0.100 | 44.00 | 0.994 | 44.16 | 0.994 | 42.12 | 0.993 | 42.02 | 0.992 | 42.36 | 0.993 |
| 0.215 | 43.14 | 0.993 | 43.08 | 0.993 | 42.64 | 0.993 | 42.23 | 0.993 | 42.43 | 0.993 |
| 1.000 | 41.93 | 0.992 | 40.61 | 0.991 | 41.63 | 0.991 | 40.25 | 0.991 | 40.28 | 0.991 |
| 4.641 | 39.42 | 0.987 | 38.01 | 0.988 | 39.64 | 0.989 | 37.31 | 0.988 | 37.35 | 0.988 |
| 10.00 | 39.13 | 0.986 | 36.83 | 0.986 | 38.51 | 0.987 | 36.00 | 0.985 | 35.94 | 0.986 |
| ave | 41.52 | 0.990 | 40.54 | 0.990 | 40.91 | 0.990 | 39.56 | 0.990 | 39.67 | 0.990 |

TABLE 14
Quantitative evaluation of our proposed framework trained only with a set of fixed parameter values ("various (fixed)") on the $L_0$ smoother [6]. The parameters used for training our framework are taken from the 5 non-boldface parameters between [0.002, 0.2] in the table. The extra 4 parameters with boldface are only used in the test stage. The absolute difference between the network trained on a single parameter ("single") and various fixed parameters ("various (fixed)") is displayed in the bottom.

| | | 0.0020 | **0.0025** | **0.0033** | 0.0043 | 0.0200 | 0.0928 | **0.1200** | **0.1600** | 0.2000 | average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| single | PSNR | 40.69 | 40.19 | 39.77 | 38.96 | 36.07 | 33.08 | 31.78 | 31.13 | 31.75 | 35.94 |
| | SSIM | 0.989 | 0.987 | 0.986 | 0.986 | 0.982 | 0.977 | 0.973 | 0.972 | 0.973 | 0.981 |
| various (fixed) | PSNR | 39.61 | 39.33 | 38.95 | 38.51 | 35.37 | 31.80 | 31.40 | 30.69 | 30.54 | 35.13 |
| | SSIM | 0.988 | 0.987 | 0.986 | 0.986 | 0.979 | 0.972 | 0.972 | 0.971 | 0.970 | 0.979 |
| diff | PSNR | 1.08 | **0.86** | **0.82** | 0.45 | 0.7 | 1.28 | **0.38** | **0.44** | 1.21 | 0.81 |
| | SSIM | 0.001 | **0** | **0** | 0 | 0.003 | 0.005 | **0.001** | **0.001** | 0.003 | 0.002 |

performance limit of incorporating the 10 image operators with this neural network.

Note learning all the normalization parameters (norm(all)) only obtains a little improvement (0.41dB) than learning the one in the first layer (norm(1)), from which we can see it's more important which position the controllable parameters are instead of how many layers the parameters are located in.

### 6.3 Interpretation of the weight learning network

To help understand the connection between the *base* network $\mathcal{N}_{base}$ and the *weight learning* network $\mathcal{N}_{weight}$, we decompose the parameter vector $\overrightarrow{\gamma}$ and the weight matrix $A_i$ into independent elements $\gamma_1, ..., \gamma_m$ and $A_{i0}, ..., A_{im}$ respectively, then:

$$(A_i\overrightarrow{\gamma} + B_i) \otimes x = \sum_{k=1}^{m} \gamma_k A_{ik} \otimes x + B_i \otimes x \qquad (7)$$

where $\otimes$ denotes convolution operation, and $m$ is the dimension of $\overrightarrow{\gamma}$. In other words, the one convolution layer, whose weights are learned with one single fc layer, is exactly equivalent to a multi-path convolution block as shown in Fig. 7. Learning the weight and bias of the single fc layer is equivalent to learning the common basic convolution kernels $B_i, A_{i1}, A_{i2}, ..., A_{im}$ in the convolution block.

### 6.4 Analysis of more variants of our proposed network

In this subsection, we experiment with a few variants of our network to justify its effectiveness.

#### *6.4.1 Training the base network only*

Since training a fully convolutional network alone has been employed frequently by previous image processing papers [17], [1], [18], which presents a strong baseline, we experiment with this alternative ("nume. (base) ") which only leverages the *base* network by training it on one specific parameter configuration. As show in Table 13, our proposed framework ("single") achieves even better performance than the baseline under the PSNR error metric.

#### *6.4.2 Training with more fc layers*

We also try a deeper *weight learning* network with more fully connected layers. Here we simply add one more fully connected layer to the default *weight learning* network, and demonstrate the performance of its two variants with ("nume. (fc2R)") and without

("nume. (fc2)") ReLU between these two layers respectively. As shown in Table 13, they all achieve comparable performance with that of single layer ("nume.") used in our paper. The potential reason for this phenomenon is that this one-layer *weight learning* network is sufficient for adaptively learning various parameter settings, while adding more weights/complexity to the network does not contribute to the performance much.

### 6.5 Interpolation ability of the proposed framework on unseen input parameters.

Since the *weight learning* network contains a single fully connected layer with no non-linear activation layers, the predicted convolution weights should be a linear transformation of the input parameters. Given such a fact, any convolutional kernels of a specific parameter should be the linear interpolation of the other two. Hence we are curious about the interpolation ability of the proposed framework.

To verify such a property, we train the network using only a few fixed parameter values, which corresponds to the 5 non-boldface parameter values in Table 14. But in the test stage, we use another 4 parameter values (boldface in Table 14) that have not been seen by the network in the training stage but are between the lower and upper bound of its parameter range. As shown in Table 14, we can see that the network performs similarly to the one trained with only one parameter value, but more importantly for the interpolated boldface parameter values that the network does not recognize, it also surprisingly achieves very comparable results.

This means that a few parameter values are already sufficient for learning a good linear transformation in the *weight learning* network from input parameters to convolution weights. However, as in a real scenario, the number of such fixed training parameters is usually difficult to decide, due to many different parameter ranges of image operators. As a result, we choose to sample random parameter values instead of only a few of them for training in our paper.

## 7 Conclusion

In this paper, we propose the first decouple learning framework for parameterized image operators, where the weights of the task-oriented *base* network $\mathcal{N}_{base}$ are decoupled from the network structure and directly learned by another *weight learning* network $\mathcal{N}_{weight}$. These two networks can be easily end-to-end trained, and $\mathcal{N}_{weight}$ dynamically adjusts the weights of $\mathcal{N}_{base}$ for different parameters $\overrightarrow{\gamma}$ during the runtime. We show that the proposed framework can be applied to different parameterized image operators, such as image smoothing, denoising and super resolution, while obtaining comparable performance as the network trained for one specific parameter configuration. It also has the potential to jointly learn multiple different parameterized image operators within one single network.For real user scenarios, we further extend our framework to enable real-time parameter tuning, which obtains superior performance over state-of-the-art methods by a largin margin. To better understand the working principle, we also provide some valuable analysis and discussions, which may inspire more promising research in this direction. More theoretical analysis is worthy of further exploration in the future.

**Acknowledgement** This work was supported in part by: National 973 Program (2015CB352501), NSFC-ISF (61561146397), the Natural Science Foundation of China under Grant U1636201 and 61629301

## References

[1] 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.

[2] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1646–1654.

[3] 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.

[4] J. Chen, A. Adams, N. Wadhwa, and S. W. Hasinoff, "Bilateral guided upsampling," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 203, 2016.

[5] M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand, "Deep bilateral learning for real-time image enhancement," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 118, 2017.

[6] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via l 0 gradient minimization," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6. ACM, 2011, p. 174.

[7] L. Karacan, E. Erdem, and A. Erdem, "Structure-preserving image smoothing via region covariances," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 176, 2013.

[8] 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.

[9] Q. Zhang, X. Shen, L. Xu, and J. Jia, "Rolling guidance filter," in *European conference on computer vision*. Springer, 2014, pp. 815–830.

[10] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2. IEEE, 2005, pp. 60–65.

[11] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image processing*, vol. 15, no. 12, pp. 3736–3745, 2006.

[12] J. Yang, J. Wright, T. S. Huang, and Y. Ma, "Image super-resolution via sparse representation," *IEEE transactions on image processing*, vol. 19, no. 11, pp. 2861–2873, 2010.

[13] J. Sun, Z. Xu, and H.-Y. Shum, "Image super-resolution using gradient profile prior," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[14] M. E. Tipping and C. M. Bishop, "Bayesian image super-resolution," in *Advances in neural information processing systems*, 2003, pp. 1303–1310.

[15] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[16] V. Jain and S. Seung, "Natural image denoising with convolutional networks," in *Advances in Neural Information Processing Systems*, 2009, pp. 769–776.

[17] S. Liu, J. Pan, and M.-H. Yang, "Learning recursive filters for low-level vision via a hybrid neural network," in *European Conference on Computer Vision*. Springer, 2016, pp. 560–576.

[18] Q. Chen, J. Xu, and V. Koltun, "Fast image processing with fully-convolutional networks," in *IEEE International Conference on Computer Vision*, vol. 9, 2017.

[19] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Transactions on Graphics (ToG)*, vol. 26, no. 3, p. 96, 2007.

[20] K. He and J. Sun, "Fast guided filter. arxiv 2015," *arXiv preprint arXiv:1505.0099*.

[21] J. Schmidhuber, "Learning to control fast-weight memories: An alternative to dynamic recurrent networks," *Neural Computation*, vol. 4, no. 1, pp. 131–139, 1992.

[22] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, 2016, pp. 3981–3989.

[23] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *International Conference on Machine Learning*, 2017.

[24] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, and N. de Freitas, "Learning to learn for global optimization of black box functions," in *International Conference on Machine Learning*, 2017.

[25] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *ICLR*, 2018.

[26] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via l0 gradient minimization," *ACM Transactions on Graphics (SIGGRAPH Asia)*, 2011.

[27] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis," in *Proc. CVPR*, 2017.

[28] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *ICLR*, 2016.

[29] 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)*, vol. 27, no. 3. ACM, 2008, p. 67.

[30] 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*, 2014, pp. 2830–2837.

[31] M. Aubry, S. Paris, S. W. Hasinoff, J. Kautz, and F. Durand, "Fast local laplacian filters: Theory and applications," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 5, p. 167, 2014.

[32] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*. Springer, 2014, pp. 184–199.

[33] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in neural information processing systems*, 2016, pp. 2802–2810.

[34] C. Dong, Y. Deng, C. Change Loy, and X. Tang, "Compression artifacts reduction by a deep convolutional network," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 576–584.

[35] X. Fu, J. Huang, D. Zeng, Y. Huang, X. Ding, and J. Paisley, "Removing rain from single images via a deep detail network," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1715–1723.

[36] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CVPR*, 2017.

[37] Y. Chen and T. Pock, "Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1256–1272, 2017.

[38] 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.

[39] Y. Tai, J. Yang, X. Liu, and C. Xu, "Memnet: A persistent memory network for image restoration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4539–4547.

[40] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR Oral)*, June 2016.

[41] J. Kim, J. Kwon Lee, and K. Mu Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1637–1645.

[42] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Transactions on image processing*, vol. 16, no. 8, pp. 2080–2095, 2007.

[43] 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.

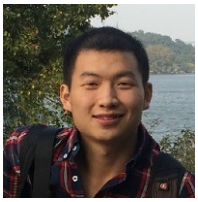
[44] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep cnn denoiser prior for image restoration," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2017.

[45] X. Fu, J. Huang, X. Ding, Y. Liao, and J. Paisley, "Clearing the skies: A deep network architecture for single-image rain removal," *IEEE Transactions on Image Processing*, vol. 26, no. 6, pp. 2944–2956, 2017.

[46] C. Lu, L. Xu, and J. Jia, "Combining sketch and tone for pencil drawing production," in *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*. Eurographics Association, 2012, pp. 65–73.

[47] J. Koutnik, F. Gomez, and J. Schmidhuber, "Evolving neural networks in compressed weight space," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 619–626.


**Qingnan Fan** is a Ph.D. student from Shandong University, China. He is currently a research intern in the Advanced Innovation Center for Future Visual Entertainment of Beijing Film Academy. His research interests mainly include real-time video segmentation, reflection removal, image smoothing, intrinsic image decomposition, and computational photography.

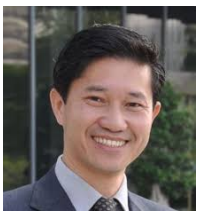


**Dongdong Chen** is a Ph.D. student from the University of Science and Technology of China. He is currently working at Microsoft Redmond as a Research Intern. He is also a joint phd between his university and Microsoft Asia. His research interests mainly include style transfer, image generation, low-level image processing and object detection.

**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. Now he is a Senior Research Manager in Microsoft Redmond. His research interests include computer vision, applied machine learning and computatational photography.

**Gang Hua** is a Principal Researcher/Research Manager in Microsoft Research. 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 130 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 IAPR Fellow, an ACM Distinguished Scientist, and a Senior Member of IEEE. His research focuses on artificial intelligence, computer vision, pattern recognition, machine learning, and robotics, with primary applications in the cloud and mobile intelligence domain.

**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 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.



**Baoquan Chen** is a Professor of Peking University, where he is the Executive Director of the Center on Frontiers of Computing Studies. Prior to the current post, he was the Dean of School of Computer Science and Technology at Shandong University, and the founding director of the Visual Computing Research Center, Shenzhen Institute of Advanced Technology (SIAT), Chinese Academy of Sciences (2008-2013), and a faculty member at Computer Science and Engineering at the University of Minnesota at Twin Cities (2000-2008). His research interests generally lie in computer graphics, visualization, and human-computer interaction, focusing specifically on large-scale city modeling, simulation and visualization.