

# Evaluating Parameterization Methods for Convolutional Neural Network (CNN)-Based Image Operators

Seung-Wook Kim, Sung-Jin Cho, Kwang-Hyun Uhm, Seo-Won Ji, Sang-Won Lee, and Sung-Jea Ko\*

School of Electrical Engineering, Korea University

Seoul, Korea

{swkim, sjcho, khuhm, swji, swlee}@dali.korea.ac.kr, sjko@korea.ac.kr

## Abstract

Recently, deep neural networks have been widely used to approximate or improve image operators. In general, an image operator has some hyper-parameters that change its operating configurations, e.g., the strength of smoothing, up-scale factors in super-resolution, or a type of image operator. To address varying parameter settings, an image operator taking such parameters as its input, namely a parameterized image operator, is an essential cue in image processing. Since many types of parameterization techniques exist, a comparative analysis is required in the context of image processing. In this paper, we therefore analytically explore the operation principles of these parameterization techniques and study their differences. In addition, performance comparisons between image operators parameterized by using these methods are assessed experimentally on common image processing tasks including image smoothing, denoising, deblocking, and super-resolution.

## 1. Introduction

Image operators are traditional research topics in multimedia processing and computer vision [2, 19, 20, 16, 18, 10]. As deep learning technology advances in recent years, various image operators using deep neural networks (DNNs) have been proposed to mimic conventional image operators or boost the performance of image processing tasks [6, 8, 17]. Most image operators have some hyper-parameters controlling its actions, e.g., the up-scale size in super-resolution and the strength of smoothing in the filtering process, as well as a choice of image operators.

A simple solution for dealing with such parameters is to train multiple networks separately, each of which is in charge of a specific parameter value and learns the mapping function with the parameter. However, this approach can be very inefficient because, for each parameter, a different

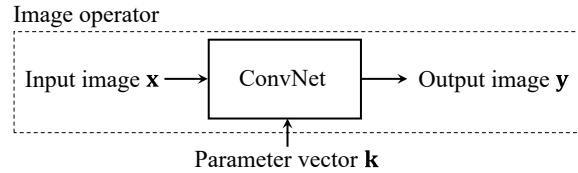


Figure 1. Illustration of parameterized image operators.

DNN model should be learned, i.e., the same number of models as the number of parameters is required. Thus, if a parameter spans too many values, this approach becomes infeasible. Various parameterization methods have been proposed to construct a feasible network model and control its built-in parameters [5, 3, 4, 12]. As conventional image operators employ many different parameterization methods, however, it is unclear what similarities such methods have and how they are different.

For more insight and effective implementation, this paper studies the properties of different parameterization methods. A taxonomy of the operation principles of each parameterization method is first presented. Then, we implement different parameterization methods by adopting a common network using residual blocks [7] for image processing. For experiments, we build a data set by combining images from public image databases, the Berkeley segmentation data set (BSDS) [1] and the Waterloo Exploration data set (WEDS) [11], and generate the images for each image processing task by using randomly selected parameter values. Then, we train the image operator networks using the generated data set and compare the network models with different parameterization methods. Based on extensive experimental results, our recommendations will be given regarding which parameterization method should be used for the given image operator.

This paper is organized as follows. In Section 2, a taxonomy of the parameterization methods for DNNs is presented. Further, their relation and operation principles are discussed. Implementation details and the experimental

\*Corresponding author

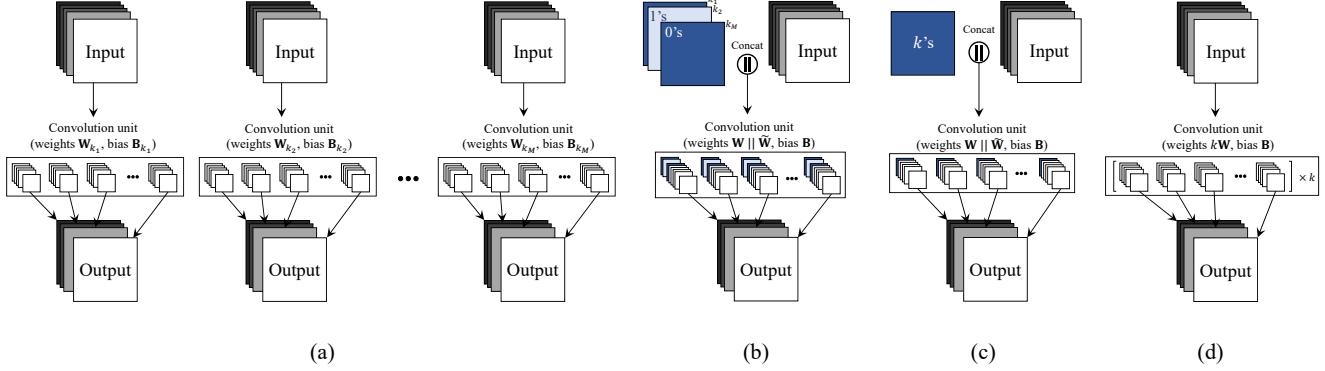


Figure 2. Different types of parameterized image operators. Type 1: the separated networks for each different parameter (a); Type 2: the single network with one-hot vector concatenation (b); Type 3: the single network with the parameter value concatenation(c); Type 4: the decoupled network (d).

setup are described in Section 3. Section 4 shows the extensive experimental results and Section 5 concludes this paper.

## 2. Parameterization methods

In this section, parameterized image operators proposed in the field of DNNs are presented and their operating characteristics are summarized. We will review four types of famous parameterization approaches, of which the overview is given in Figure 2

### 2.1. Type 1: Training separated networks

The simplest way to cope with varying parameter values of image operators is to train independent networks for different parameter values, as shown in Figure 2(a). In this type of networks, data sets consisting of a pair of inputs and labels for the parameter values are first generated and divided into data subsets according to values of the parameters. Then, each independent network is trained separately using the data subset for the corresponding parameters. For the sake of simplicity, we hereinafter assume a single-layer convolution unit to represent a convolutional network (ConvNet) and describe the case of the one-dimensional input and output. Without loss of generality, it is easy to extend the concept to the image domain. Formally, let an input image  $\mathbf{x} \in \mathbb{R}^d$  and a parameter  $k_m$  given, where the value of  $k_m$  represents encoded information, e.g., the up-scale size in super-resolution and the strength of smoothing in the filtering process, as well as a choice of image operators. Then, an output of the image operator for the channel  $c$  can be obtained as follows:

$$y_c = \langle \mathbf{W}_{k_m}^{(c)}, \mathbf{x} \rangle + b_{k_m}^{(c)}, \quad (1)$$

where  $\langle \cdot, \cdot \rangle$  is a function of inner product, and  $\mathbf{W}_{k_m}^{(c)} \in \mathbb{R}^d$  and  $b_{k_m}^{(c)} \in \mathbb{R}$  denote the weights and biases of the  $c$ th channel with respect to a parameter  $k_m$  for  $m \in \{1, 2, \dots, M\}$ , respectively. Since the networks are separately trained, each image operator works normally only when an input image produced by using parameters corresponding to the network is given. However, this approach becomes infeasible as a parameter spans many values, i.e., the value of  $M$  becomes very large, because the same number of image operators as the number of parameters should be trained.

### 2.2. Type 2: Concatenating one-hot vectors

Type 2 is also a common approach in which the input channels are increased and the information of the parameter [4, 12] is included. In this approach, a one-hot vector containing the parameter information is appended to the input of the image operator. Figure 2(b) illustrates an example. Let the concatenated input denoted as

$$\tilde{\mathbf{x}} = \mathbf{x} \parallel \mathbf{o}_{k_m}, \quad (2)$$

where  $\parallel$  is the concatenation operations of two features, and  $\mathbf{o}_{k_m} = [0, \dots, 0, 1, 0, \dots, 0]^T \in \{0, 1\}^M$  represents the one-hot vector, of which the  $k_m$ th channel value is 1 and otherwise 0. Note that, since the input channels are increased, the channel number of weights is accordingly increased. The output of an image operator can be obtained by

$$\begin{aligned} y_c &= \langle \mathbf{W}^{(c)} \parallel \tilde{\mathbf{W}}^{(c)}, \tilde{\mathbf{x}} \rangle + b^{(c)} \\ &= \langle \mathbf{W}^{(c)}, \mathbf{x} \rangle + \langle \tilde{\mathbf{W}}^{(c)}, \mathbf{o}_{k_m} \rangle + b^{(c)} \\ &= \langle \mathbf{W}^{(c)}, \mathbf{x} \rangle + \tilde{w}_{k_m}^{(c)} + b^{(c)} \\ &\triangleq \langle \mathbf{W}^{(c)}, \mathbf{x} \rangle + \tilde{b}_{k_m}^{(c)}, \end{aligned} \quad (3)$$

where  $\tilde{\mathbf{W}}^{(c)} = [\tilde{w}_{k_1}^{(c)}, \tilde{w}_{k_2}^{(c)}, \dots, \tilde{w}_{k_M}^{(c)}]$  is the appended weight as the number of input channels increases. From (3), we can find that the network output of Type 2 is controlled by the bias term corresponding to the parameter  $k_m$ . As compared with Type 1, Type 2 has a more efficient structure sharing the weight terms having a large volume for different parameters. However, Type 2 is still not a flexible structure because, as the covering range of the parameter increases, the size of an image operator also grows. If the parameter  $k_m$  is a continuous value, the network model is still infeasible.

### 2.3. Type 3: Concatenating parameter values

In [3], a parameterization method is introduced, in which a condition value is given to a network to produce a desired value under the given condition. As shown in Figure 2(c), in this type, the parameter value itself is first concatenated as follows:

$$\tilde{\mathbf{x}} = \mathbf{x} || k_m. \quad (4)$$

Then, by feeding  $\tilde{\mathbf{x}}$  into the image operator, the channel output is obtained by

$$\begin{aligned} y_c &= \langle \mathbf{W}^{(c)} || w^{(c)}, \tilde{\mathbf{x}} \rangle + b^{(c)} \\ &= \langle \mathbf{W}^{(c)}, \mathbf{x} \rangle + k_m w^{(c)} + b^{(c)} \\ &\triangleq \langle \mathbf{W}^{(c)}, \mathbf{x} \rangle + \tilde{b}_{k_m}^{(c)}. \end{aligned} \quad (5)$$

Note that, in (5),  $\tilde{b}_{k_m}^{(c)} = k_m w^{(c)} + b^{(c)}$ . As compared to (3), in Type 3, the separated bias term with respect to the parameter  $k_m$  is approximated as

$$\tilde{w}_{k_m}^{(c)} \approx k_m w^{(c)}. \quad (6)$$

By decomposing the bias term into the parameter value and its coefficient, Type 3 can efficiently conduct a parameterized image operation even if a value of the parameter is not quantized, *i.e.*, the network can deal with a continuous parameter value.

### 2.4. Type 4: Decoupled network

Recently, Fan *et al.* [5] proposed a network architecture more focusing on the parameterized image operation. They design another network which dynamically generates the weights for the convolution layers of an image operator with varying input parameters. Such design enables a parameterized image operator to be decoupled from parameter adaptation. In this model, the weight of a convolution layer is obtained by using a single fully-connected layer. Since the fully-connected layer takes a parameter  $k_m$  as its input, the weight for an image operator can be obtained by

$$\mathbf{W}_{k_m}^{(c)} = k_m \mathbf{W}_{\text{FC}}^{(c)}, \quad (7)$$

where  $\mathbf{W}_{\text{FC}}^{(c)}$  denotes the weight of the fully-connected layer (in fact, the fully-connected layer also has a bias term, but we omitted this term to simplify the analysis). Using (7), the channel output of a parameterized image operator is given by

$$\begin{aligned} y_c &= \langle \mathbf{W}_{k_m}^{(c)}, \mathbf{x} \rangle + b^{(c)} \\ &= \langle k_m \mathbf{W}_{\text{FC}}^{(c)}, \mathbf{x} \rangle + b^{(c)} \\ &= k_m \langle \mathbf{W}_{\text{FC}}^{(c)}, \mathbf{x} \rangle + b^{(c)}. \end{aligned} \quad (8)$$

As compared to (1), in Type 4, the separated convolutions with respect to the parameter  $k_m$  are approximated by scaling the channel output. In this case, the parameter  $k_m$  is used as a scale factor.

### 2.5. Discussion

As can be seen in the aforementioned review, Types 2-4 are an approximation of Type 1 to construct a single network model dealing with the multiple inputs of parameters. For different parameter values, Types 2 and 3 employ the same weights of the convolution layers and, on the other hand, tweak the operating configurations via bias values; on the contrary, Type 4 changes the configurations by scaling the weight values of the convolution layers. Let us consider that an activation function, *e.g.*, rectified linear unit (ReLU) [13], is applied after the parameterized convolution layer. In Types 2 and 3, the ReLU non-linearity discards the channel values if  $\langle \mathbf{W}^{(c)}, \mathbf{x} \rangle < -\tilde{b}_{k_m}^{(c)}$ , *i.e.*, the sparsity of the output feature maps is controlled by the input parameter values. In Type 4, since the bias term does not vary according to the parameters, the sparsity of the feature map is not influenced. Instead, the weight terms are scaled by the parameter values as in (8), and thus the magnitude of the feature map is changed by the parameter values (although Types 2 and 3 also can change the magnitude of the output by adding the bias terms, the effect of Type 4 is more dramatic). Intuitively, we can expect that scaling the magnitude of the feature maps would have an advantage in handling the intensity of an image operation. For switching the image operator through parameter values, however, it is not easy to say which type is better than the others.

## 3. Experimental setup

In this section, the experimental setup to evaluate the different parameterization methods for the image operator, Types 1-4, is presented. We first introduce the network architecture used to verify the performance of each image operators for the different parameterization methods. Next, the implementation details for the experiment are given. Finally, we describe the data sets used for evaluation and the evaluation criteria.

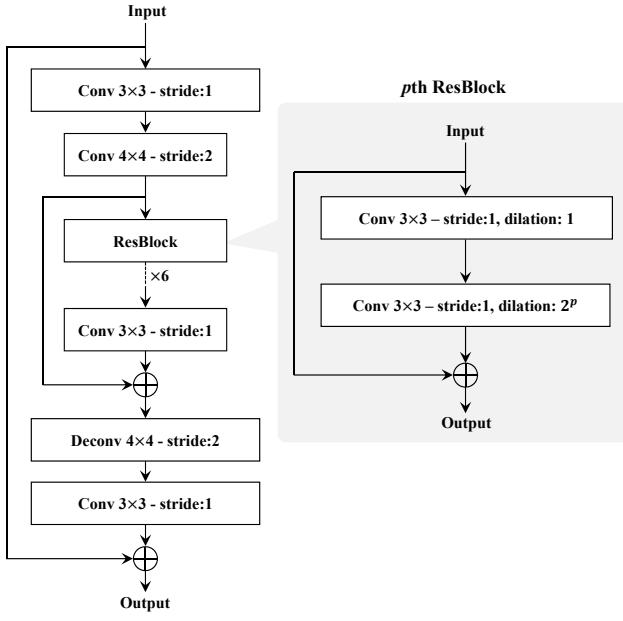


Figure 3. An overall network architecture composed of 17 convolution layers using simple basic residual blocks.

Image operators	PSNR	SSIM
$L_0$ smoothing [16]	35.33	0.9665
Pencil drawing [10]	30.65	0.8592
Denoising	35.92	0.9404
Super-resolution	34.94	0.8718
Deblocking	34.48	0.8932

Table 1. PSNR and SSIM results obtained by using the separated networks (Type 1).

### 3.1. Network architecture for an image operator

We empirically test the different parameterization methods, Types 1-4, using a simple network architecture inspired by CycleGAN [21] and StarGAN [4]. Figure 3 shows the overall network architecture of an image operator, which has a symmetric structure. The network is composed of one convolution layer for the low-level feature extraction, another convolution layer with a stride size of two for down-sampling, six residual blocks [7], one convolution layer for aggregating the features from the residual blocks, one deconvolution layer with a stride size of two for upsampling, and a final convolution layer for producing the results in the form of image. In the network architecture used, there are two residual paths to learn the residual information between the input and output images more effectively. As shown in Figure 3, a residual block consists of two  $3 \times 3$  convolution layers and a skip connection. Among the two convolution layers, the latter one has a dilation rate of  $2^p$  for  $p = 1, 2, \dots, 6$  to increase the receptive field of the im-

age operator. We used the channel number of 64 for all the convolution and deconvolution layers in the network model except for the last convolution layer. The ReLU [13] is used as an activation function.

## 3.2. Implementation details

### 3.2.1 Details

In parameterized image operators, two approaches are generally used, parameterizing only the first feature layer and parameterizing all network layers. In this paper, we tested both ways to verify which method is more effective for an image operator. Every feature layer was initialized using the Xavier method [14]. Since this paper does not focus on the performance improvement of image operators, we employed simple  $\ell_2$  loss between an output and its label to optimize the model parameters. The full source code is implemented on the Pytorch [15] framework and will be available online. All models were trained using the Adam [9] solver with  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$ . The mini-batch size was set to 12 for all experiments. We trained all network models for 50 epochs with an initial learning rate of  $10^{-2}$ , which is divided by 10 at 30 epochs and again at 40 epochs. Our experiments were conducted on a single NVIDIA Titan X GPU.

### 3.2.2 Image operators

To compare the parameterization methods on various types of image operators, we utilized two types of image processing tasks, image filtering and image restoration. For the image filtering task, we adopted  $L_0$  smoothing [16], relative total variation (RTV) [18], and pencil drawing [10]. For the image restoration task, we utilized denoising, super-resolution, and JPEG artifact deblocking.

### 3.3. Types of parameters

As mentioned earlier, parameters can change the operating configurations of an image operator such as the intensity of image processing. Also, the parameterization can be extended to handle multiple image operators if we allocate a specific parameter value for each image processing task. In Section 4, we will present the results of three sets of experiments. First, we build a network model for each single image operator and only parameterize the intensity of image processing (Subsection 4.1: A single operator with multiple parameters). Second, we build a single model, fix the intensity of image processing, and parameterize switching the image operators (Subsection 4.2: Multiple operators with a single parameter). Finally, we build a single model and parameterize both switching the image operators and varying the intensity of image operations (Subsection 4.3: Multiple operators with multiple parameters).

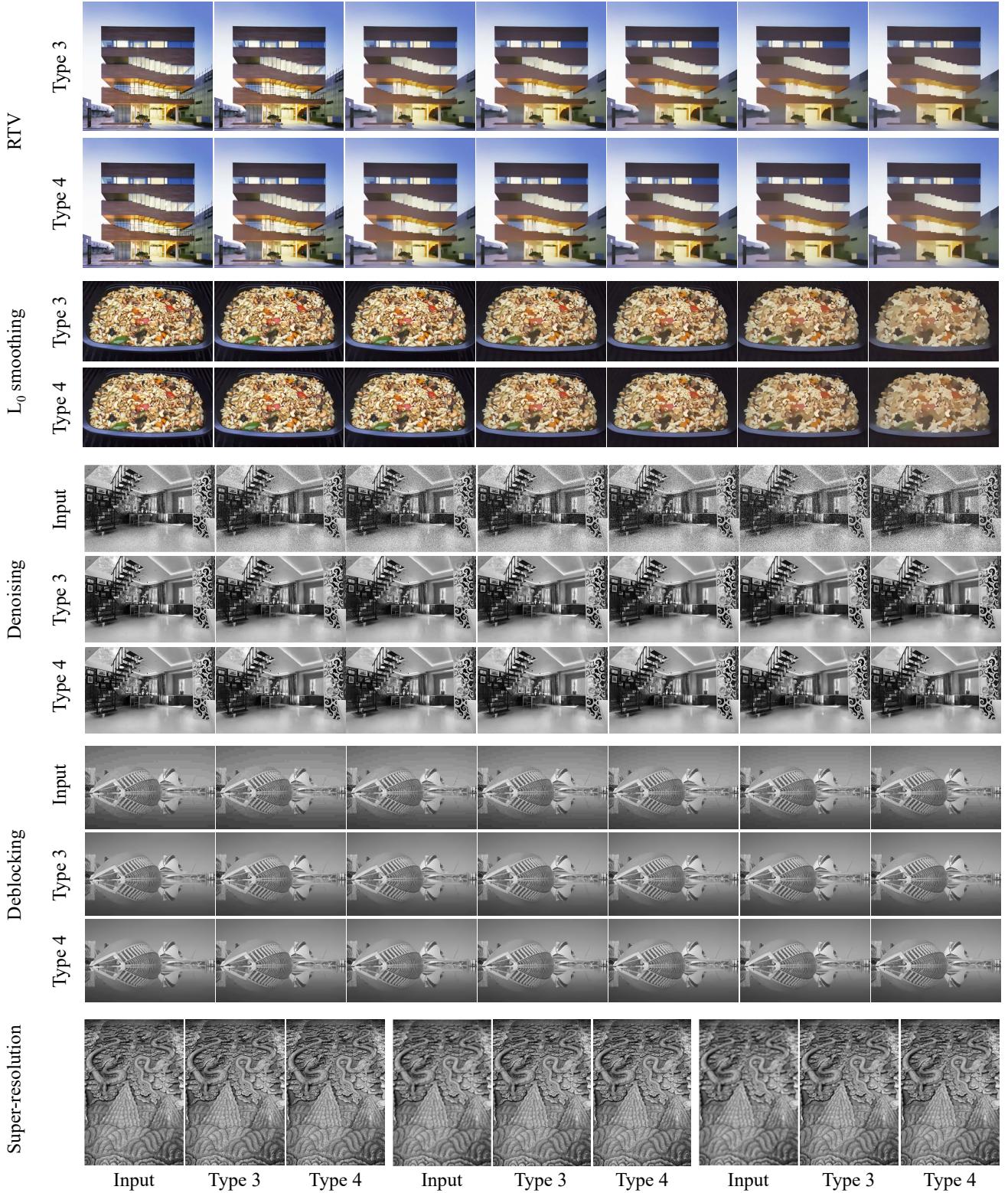


Figure 4. Resultant images of the image operators parameterized by Types 3 and 4. From left to right, the input parameter values increase. For the super-resolution task, the image operator restores the input images down-sampled with the scale factors of two, three, and four, respectively, from left to right. We only show the results of applying the parameterization to all layers due to the page limit.

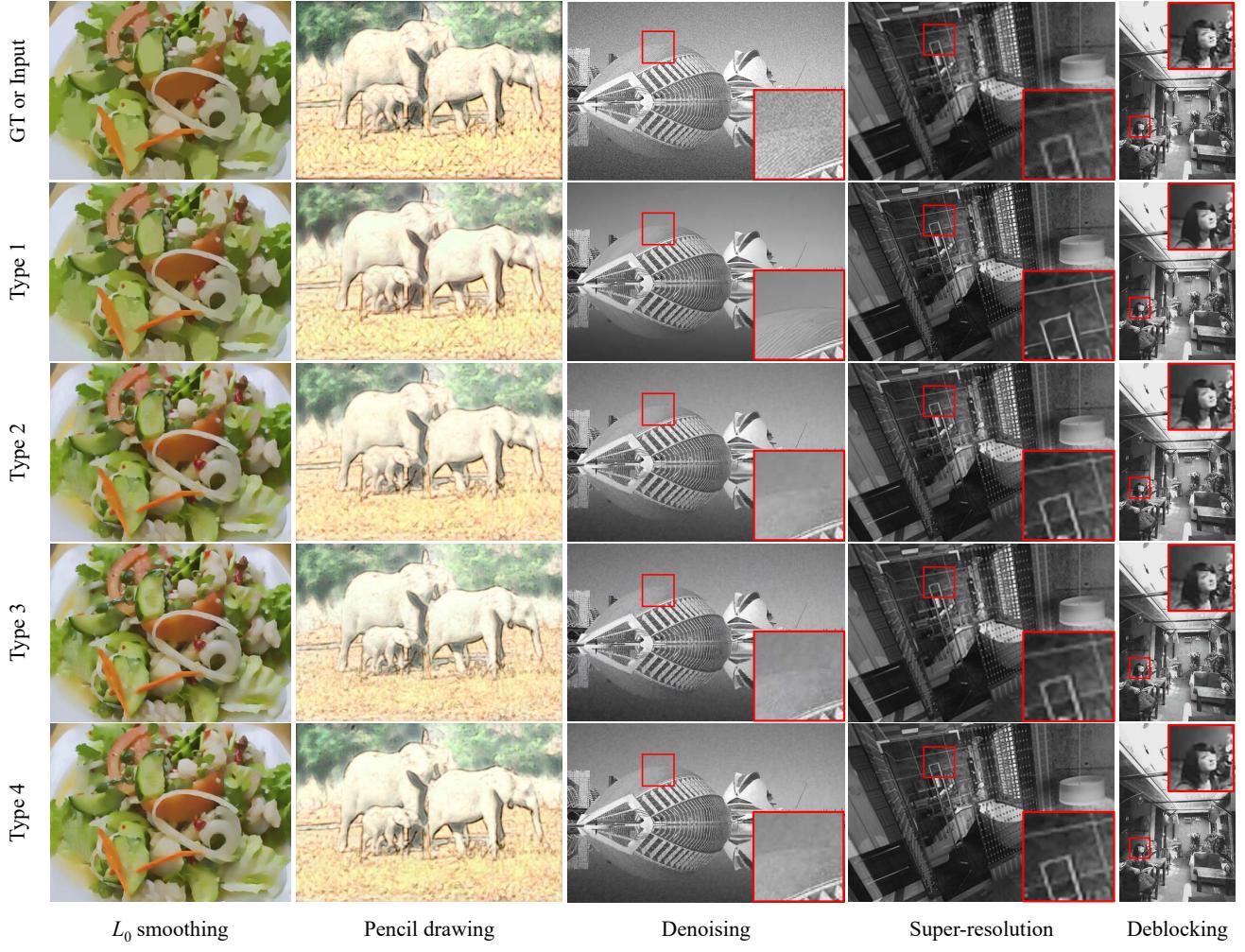


Figure 5. Resultant images of the image operators parameterized by Types 1-4. For  $L_0$  smoothing [16] and pencil drawing [10], the first row shows the ground truth labels for the filtered results, and for image restoration, it depicts the input to be recovered.

Image operators	PSNR				SSIM			
	1st layer		All layers		1st layer		All layers	
	Type 3	Type 4	Type 3	Type 4	Type 3	Type 4	Type 3	Type 4
RTV [18]	41.68	41.85	41.93	<b>42.17</b>	0.9866	0.9876	0.9878	<b>0.9883</b>
$L_0$ smoothing [16]	35.31	35.36	35.26	<b>35.89</b>	0.9618	0.9610	0.9598	<b>0.9672</b>
Denoising	32.56	32.61	32.58	<b>32.64</b>	0.8260	0.8287	0.8277	<b>0.8296</b>
Super-resolution	35.20	35.20	35.17	<b>35.26</b>	0.8753	0.8753	0.8749	<b>0.8767</b>
Deblocking	34.27	34.27	34.27	<b>34.28</b>	0.8830	0.8830	0.8828	<b>0.8831</b>

Table 2. PSNR and SSIM results obtained by using the networks parameterized by Type 3 and Type 4. In this experiment, each single operator with multiple parameters was learned. Parameterization methods applied only to the first layer and to all layers were both evaluated.

Image operators	PSNR						SSIM					
	1st layer			All layers			1st layer			All layers		
	Type 2	Type 3	Type 4	Type 2	Type 3	Type 4	Type 2	Type 3	Type 4	Type 2	Type 3	Type 4
$L_0$ smoothing [16]	31.57	31.16	31.59	<b>32.03</b>	30.90	31.69	0.9010	0.8903	0.9025	<b>0.9124</b>	0.8856	0.9078
Pencil drawing [10]	30.55	30.25	30.54	<b>30.63</b>	29.72	30.31	0.8365	0.8129	0.8270	<b>0.8439</b>	0.8017	0.8304
Denoising	33.20	32.18	33.24	<b>33.70</b>	31.68	33.38	0.8892	0.8390	0.8859	<b>0.8983</b>	0.8094	0.8918
Super-resolution	33.72	33.37	33.67	<b>33.90</b>	33.19	33.61	0.8456	0.8377	0.8458	<b>0.8528</b>	0.8316	0.8456
Deblocking	33.50	33.04	33.45	<b>33.69</b>	32.72	33.39	0.8754	0.8701	0.8739	<b>0.8776</b>	0.8690	0.8740

Table 3. PSNR and SSIM results obtained by using the networks parameterized by Type 2, Type 3, and Type 4. In this experiment, multiple operators with a single parameter were learned. Parameterization methods applied only to the first layer and to all layers were both evaluated.

Image operators	PSNR						SSIM					
	1st layer			All layers			1st layer			All layers		
	Type 3	Type 4	Type 3	Type 4	Type 3	Type 4	Type 3	Type 4	Type 3	Type 4	Type 3	Type 4
$L_0$ smoothing [16]	34.79	<b>35.60</b>	34.93	35.51	0.9551	<b>0.9635</b>	0.9552	0.9629				
Denoising	32.34	<b>32.48</b>	32.32	32.48	0.8260	0.8287	0.8277	<b>0.8296</b>				
Super-resolution	34.84	<b>35.04</b>	34.84	35.03	0.8678	<b>0.8714</b>	0.8681	0.8712				
Deblocking	34.01	34.10	34.04	<b>34.12</b>	0.8778	0.8793	0.8780	<b>0.8794</b>				

Table 4. PSNR and SSIM results obtained by using the networks parameterized by Type 3 and Type 4. In this experiment, Multipl operators with multiple parameters were learned. Parameterization methods applied only to the first layer and to all layers were both evaluated.

### 3.3.1 Parameter sampling

For each image operation task, parameter values that determine the intensity of image operation spans very different ranges. Thus, following the parameter sampling methods in [5], we uniformly sampled parameters in either the logarithm or the linear space according to the image operation tasks (Subsections 4.1 and 4.3). For the parameter of changing the image operators, we uniformly sampled values in the linear space within  $[0, 1]$  (Subsections 4.2 and 4.3). For the experiments using a single parameter value, we sampled a median parameter value in the sampling space of the corresponding image operation task unless otherwise noted.

### 3.3.2 Data sets

In the experiments, we make use of the natural images in the Berkeley segmentation data set (BSDS) [1], which contains 500 images for image segmentation, and the Waterloo Exploration data set (WEDS) [11], which contains 4,744 images for image quality assessment. To evaluate the performance of image operators, we randomly selected 100 images from the BSDS and 1,000 images from the WEDS and utilized the combined set of the selected images as a test set. On the other hand, we created a train set which is a combination of the remained 4,144 images. For each image filtering task, we generated seven random parameter values by employing the aforementioned parameter sampling method to produce the ground truth labels with the generated parameters. Similarly, for each image restoration task,

we constructed seven input images with the randomly generated parameters except the super-resolution task. That is, for each task, the train set of 29,008 images and the test set of 7,700 images were employed for our experiments. For super-resolution, we used three upscale factors of two, three, and four as a parameter value.

### 3.3.3 Evaluation criteria

We conducted four sets of experiments: 1) A single operator with a single parameter as a baseline; 2) A single operator with multiple parameters; 3) Multiple operators with a single parameter; 4) Multiple operators with multiple parameters. As described above, for the experiment with a single parameter, we used only the median value within the parameter range. The performance of the image operators was evaluated on the test set by using average peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) as image error metrics.

## 4. Experimental results

### 4.1. Experiment 1: A single operator with multiple parameters

In this experiment, we compare the performance between Type 3 and Type 4 for the image filtering tasks, RTV and  $L_0$  smoothing, and the image restoration tasks, Denoising, super-resolution, and deblocking. Type 2 is not considered in this case because it cannot be implemented with the parameter spanning a large number of values. Table 2

shows PSNR and SSIM results. For the image filtering task, Type 4 achieves the better results in terms of both PSNR and SSIM, but the improvement is not considerable. Since RTV is a relaxed filter of  $L_0$  smoothing, in both Types, PSNR and SSIM of RTV are much higher than those of  $L_0$  smoothing. Similar results can be seen for the image restoration tasks. For both tasks, the parameterization method applied to all the layers mostly shows better results than that applied to the first layer; however, the performance increase is not very significant. Figure 4 exhibits the results of the image operators parameterized by Types 3 and 4. As can be seen in Table 2, the parameterized image operators work well in various image processing tasks, and produce the results with similar image quality.

#### 4.2. Experiment 2: Multiple operators with a single parameter

In this experiment, we compare the performance between Types 2-4 for the image filtering tasks,  $L_0$  smoothing and pencil drawing, and the image restoration tasks, denoising, super-resolution, and deblocking. For the pencil drawing, we used the desired parameter recommended in [10]. As shown in Table 3, Type 2 achieves the best results in all image processing tasks. Even Type 2 applied to only the first layer achieves the comparable or higher results in terms of PSNR and SSIM as compared with the other types applied to all the layers. As shown in Tables 1 and 3, the separated networks for each task achieve a better performance than the other types which employ a single network. Types 3 and 4, which are the linear approximation of Type 2 with respect to the biases and that of Type 1 with respect to the weights, respectively, show somewhat the poor performance of learning a single network processing multiple image operations. Accordingly, similar results can be observed in Figure 5. In the five image processing tasks, Type 1 exhibits the best results. For a given parameter with a fixed value, image operators based on CNN work well despite using a very simple structure. As we discussed in Section 2, Types 2-4 are the approximated versions of Type 1, and thus the performance of Types 2-4 tends to be lower than Type 1. Among Types 2-4, Type 2 deals with the multiple image operators well as compared with Types 3 and 4.

#### 4.3. Experiment 3: Multiple operators with multiple parameters

In this experiment, we compare the performance between Type 3 and Type 4 for  $L_0$  smoothing, denoising, super-resolution, and deblocking tasks. Similar to the results in Table 2, the image operator parameterized by Type 4 shows the better PSNR and SSIM results than that parameterized by Type 3. For both types, parameterizing the first layer and all layers achieves similar PSNR and SSIM values. As comparing between Tables 2 and 4, the performance

is degraded, but the performance degradation is not very significant considering all the results of Table 4 is obtained by using only a single network.

### 5. Discussion and conclusion

As discussed in Section 2, common parameterization methods can be classified into two groups: one is to share the weight terms and control the bias terms according to parameter values; the other one is to handle the weight terms while fixing bias terms.

For a single network coping with a single operator with a single parameter, independent networks that are separately trained can be the best choice. However, in case of dealing with many parameter values, a single network is required for real-world applications. For a single operator with multiple parameters, as we expected, Type 4 achieves the better performance than Type 3, and for multiple operators with multiple parameters, similar results can be observed. However, the performance difference is not very significant. For multiple operators with a single parameter, Type 2 shows the best results, but there still exists great performance difference between Types 1 and 2. As comparing Types 2 and 3, we can also find that the performance difference is quite significant. To be short, for a single network generating images through very different image processing tasks, Type 2 can be the best choice; on the other hand, for a single network generating images with parameters handling the intensity of image processing, Type 4 will perform well, and Type 2 can be a good alternative as considering the computational complexity. In addition, applying the parameterization only to the first layer achieves satisfactory results in spite of much lower resource increases as compared with applying the parameterization to all layers.

In this paper, the operation properties of widely used parameterization methods for the image operator have been analytically and empirically studied. In future work, we will explore the combinations of different types considering their operation properties. In addition, we will attempt to extend the parameterization methods from linear approximation to the non-linear approach.

### Acknowledgments

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2014-3-00077, Development of global multi-target tracking and event prediction techniques based on real-time large-scale analysis)

### References

- [1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans.*

- Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011. 1, 7
- [2] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2005. 1
- [3] Q. Chen, J. Xu, and V. Koltun. Fast image processing with fully-convolutional networks. In *Proc. IEEE Int. Conf. Comput. Vis.*, 2017. 1, 3
- [4] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2018. 1, 2, 4
- [5] Q. Fan, D. Chen, L. Yuan, G. Hua, N. Yu, and B. Chen. Decouple learning for parameterized image operators. In *Proc. European Conf. Comput. Vis.*, 2018. 1, 3, 7
- [6] Q. Fan, J. Yang, G. Hua, B. Chen, and D. Wipf. A generic deep architecture for single image reflection removal and image smoothing. In *Proc. IEEE Int. Conf. Comput. Vis.*, 2017. 1
- [7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2016. 1, 4
- [8] J. Kim, J. Kwon Lee, and K. Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2016. 1
- [9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. Int. Conf. Learn. Representations*, 2015. 4
- [10] 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*, pages 65–73, 2012. 1, 4, 6, 7, 8
- [11] K. Ma, Z. Duanmu, Q. Wu, Z. Wang, H. Yong, H. Li, and L. Zhang. Waterloo Exploration Database: New challenges for image quality assessment models. *IEEE Trans. Image Process.*, 26(2):1004–1016, Feb. 2017. 1, 7
- [12] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 1, 2
- [13] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. Int. Conf. Mach. Learn.*, 2010. 3, 4
- [14] V. Nair and G. E. Hinton. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010. 4
- [15] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 4
- [16] L. Xu, C. Lu, Y. Xu, and J. Jia. Image smoothing via l0 gradient minimization. *ACM Transactions on Graphics (TOG)*, 30(6):174, 2011. 1, 4, 6, 7
- [17] L. Xu, J. Ren, Q. Yan, R. Liao, and J. Jia. Deep edge-aware filters. In *Proc. Int. Conf. Mach. Learn.*, 2015. 1
- [18] 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. 1, 4, 6
- [19] J. Yang, J. Wright, T. S. Huang, and Y. Ma. Image super-resolution via sparse representation. *IEEE Trans. Image Process.*, 19(11):2861–2873, 2010. 1
- [20] G. Zhai, W. Zhang, X. Yang, W. Lin, and Y. Xu. Efficient image deblocking based on postfiltering in shifted windows. 18(1):122–126, 2008. 1
- [21] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proc. IEEE Int. Conf. Comput. Vis.*, 2017. 4