Deformable Kernel Networks for Joint Image Filtering

Beomjun Kim, Jean Ponce, Bumsub Ham

Deformable Kernel Networks for Joint Image Filtering

Beomjun Kim, Jean Ponce, *Fellow, IEEE,* Bumsub Ham, *Member, IEEE*

Abstract—Joint image filters are used to transfer structural details from a guidance picture used as a prior to a target image, in tasks such as enhancing spatial resolution and suppressing noise. Previous methods based on convolutional neural networks (CNNs) combine nonlinear activations of spatially-invariant kernels to estimate structural details and regress the filtering result. In this paper, we instead learn explicitly sparse and spatially-variant kernels. We propose a CNN architecture and its efficient implementation, called the deformable kernel network (DKN), that outputs sets of neighbors and the corresponding weights adaptively for each pixel. The filtering result is then computed as a weighted average. We also propose a fast version of DKN that runs about four times faster for an image of size 640×480. We demonstrate the effectiveness and flexibility of our models on the tasks of depth map upsampling, saliency map upsampling, cross-modality image restoration, texture removal, and semantic segmentation. In particular, we show that the weighted averaging process with sparsely sampled 3×3 kernels outperforms the state of the art by a significant margin.

Index Terms—Joint filtering, convolutional neural networks, depth map upsampling, cross-modality image restoration, texture removal, semantic segmentation

1 Introduction

Image filtering with a guidance signal, a process called guided or joint filtering, has been used in a variety of computer vision and graphics tasks, including depth map upsampling [1], [2], [3], [4], [5], [6], cross-modality image restoration [7], [8], [9], texture removal [6], [10], [11], [12], scale-space filtering [6], dense correspondence [13], [14] and semantic segmentation [15]. For example, high-resolution color images can be used as guidance to enhance the spatial resolution of depth maps [4]. The basic idea behind joint image filtering is to transfer structural details from the guidance image to the target one, typically by estimating spatially-variant kernels from the guidance. Concretely, given the target image f and the guidance image g, the filtering output $\hat{f}$ at position $\mathbf{p} = (x, y)$ is expressed as a weighted average [4], [16]:

$$\hat{f}_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} W_{\mathbf{pq}}(f, g) f_{\mathbf{q}}, \tag{1}$$

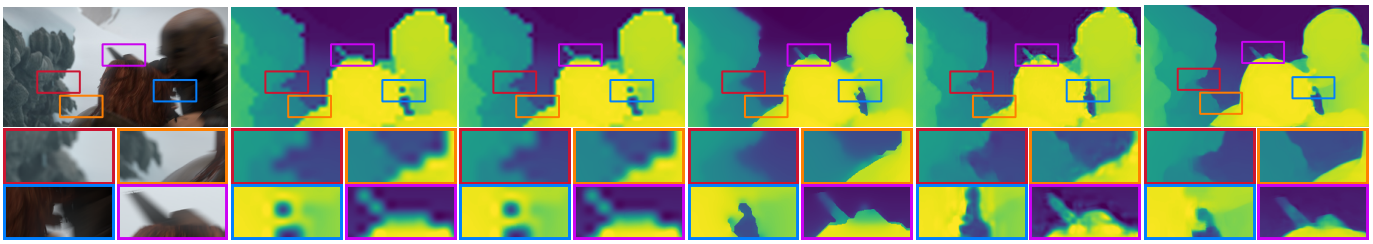
where we denote by $\mathcal{N}(\mathbf{p})$ a set of neighbors (defined on a discrete regular grid) near the position $\mathbf{p}$. The filter kernel W is a function of the guidance image g [2], [3], [4], [7], the target image f itself [11], [16], or both [5], [6], normalized so that

$$\sum_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} W_{\mathbf{pq}}(f, g) = 1. \tag{2}$$

Classical approaches to joint image filtering mainly focus on designing the filter kernels W and the set of neighbors $\mathcal{N}$ (i.e., sampling locations $\mathbf{q}$). They use hand-crafted kernels and sets of neighbors without learning. For example, the bilateral filter [16] uses spatially-variant Gaussian kernels to encode local structures from the guidance image. The guided filter [7] also leverages the local structure of the guidance image, but uses matting Laplacian kernels [19], enabling a constant processing time. These filters use regularly sampled neighbors for aggregating pixels, and do not handle inconsistent structures in the guidance and target images [6]. This causes texture-copying artifacts [3], especially in the case of data from different sensors. To address this problem, the SD filter [6] constructs spatially-variant kernels from both guidance and target images to exploit common structures, and formulates joint image filtering as an optimization problem. This type of approaches (e.g., [3], [10], [20]) computes a filtering output by optimizing an objective function that involves solving a large linear system. This is equivalent to filtering an image by an inverse matrix [7], whose rows correspond to a filter kernel, leveraging global structures in the guidance image. Optimization-based methods can be considered as implicit weighted-average filters. Learning-based approaches using convolutional neural networks (CNNs) [5], [21], [22] are also becoming increasingly popular. The networks are trained using large quantities of data, reflecting natural image priors and often outperforming traditional methods by large margins. These methods do not use a weighted averaging process with spatially-variant kernels as in (1). They combine instead nonlinear activations of spatially-invariant kernels learned from the networks. That is, they approximate spatially-variant kernels by mixing the activations of spatially-invariant ones nonlinearly (e.g., via the ReLU function [23]).

In this paper, we propose to exploit spatially-variant kernels explicitly to encode the structural details from both guidance and target images as in classical approaches, but learn the kernel weights in a completely data-driven way. We also learn the set of neighbors, building an adaptive and sparse neighborhood system for each pixel, which may be difficult to design by hand. To implement this idea, we propose a CNN architecture and its efficient implementation, called a *deformable kernel network* (DKN), for learning sampling locations of the neighboring pixels and their corresponding kernel weights at every pixel. We also propose a fast version of DKN (FDKN), achieving a $4\times$ speed-up compared to the plain the DKN for an image of size 640×480, while

- *Beomjun Kim and Bumsub Ham are with the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea. E-mail: beomjun.kim@yonsei.ac.kr; mimo@yonsei.ac.kr.*
- *Jean Ponce is with Inria and PSL Research University (CNRS/ENS/INRIA UMR 8548), Paris, France. E-mail: jean.ponce@inria.fr.*



(a) RGB image. (b) Depth image. (c) GF [7]. (d) SDF [6]. (e) DJFR [17]. (f) Ours (DKN).

Fig. 1. Qualitative comparison of the state of the art and our model on depth map upsampling ($16\times$). Given (a) a high-resolution color image and (b) a low-resolution depth image from the Sintel dataset [18], we upsample the depth image using (c) GF [7], (d) SDF [6], (e) DJFR [17] and (f) our method. The filtering results for GF and our model are obtained by the weighted average in (1). In this example, we use filter kernels W of size 3×3 in both methods. We can see that our method using sparsely sampled 3×3 kernels outperforms the state of the art including the optimization-based SDF method [6] and CNN-based one [17]. (Best viewed in color.)

retaining its superior performance. We show that the weighted averaging process using sparsely sampled 3×3 kernels is sufficient to obtain a new state of the art in a variety of applications, including depth map upsampling (Figs. 1 and 7), saliency map upsampling (Fig. 8), cross-modality image restoration (Fig. 9), texture removal (Fig. 10), and semantic segmentation (Fig. 11).

Contributions. The main contributions of this paper can be summarized as follows:

- We introduce a generic model for joint image filtering and its implementation, the DKN, that computes the set of neighbors and their corresponding weights adaptively for individual pixels (Section 3).
- We propose a fast version of DKN (FDKN) that runs about four times faster then the DKN while retaining its superior performance (Section 3).
- We achieve a new state of the art on several tasks, clearly demonstrating the advantage of our approach to learning both kernel weights and sampling locations (Section 4). We also provide an extensive experimental analysis with ablation studies to investigate the influence of all the components of our model including the size of the training dataset (Sections 5 and 6).

To encourage comparison and future work, our code and models are available at our project webpage: https://github.com/jun0kim/deformablekernelnetwork.

2 Related Work

Here we briefly describe the context of our approach, and review representative works related to ours.

2.1 Joint image filtering

We categorize joint image filtering into explicit/implicit weighted-average methods and learning-based ones. First, explicit joint filters compute the output at each pixel by a weighted average of neighboring pixels in the target image, where the weights are estimated from the guidance and/or target image [4], [7], [11]. The bilateral [16] and guided [7] filters are representative methods that have been successfully adapted to joint image filtering. They use hand-crafted kernels to transfer fine-grained structures from the guidance image. It is, however, difficult to manually adapt the kernels to new tasks, and these methods may transfer erroneous structures to the target image [5]. Second, implicit weighted-average methods formulate joint filtering as an optimization problem, and minimize an objective function that usually involves fidelity and regularization terms [2], [3], [6], [10], [20], [24]. The fidelity term encourages the filtering output to be close to the target image, and the regularization term, typically modeled using a weighted L2 norm [20], gives the output a structure similar to that of the guidance image. Although, unlike explicit ones, implicit joint filters exploit global structures in the guidance image, hand-crafted regularizers may not reflect structural priors in the guidance image. Moreover, optimizing the objective function involves solving a large linear system, which is time consuming, even with preconditioning [25] or multigrid methods [20]. Finally, learning-based methods can further be categorized into dictionary- and CNN-based approaches. Dictionary-based methods exploit the relationship between paired target patches (e.g., low- and high-resolution patches for upsampling), additionally coupled with the guidance image [26], [27]. In CNN-based methods [5], [21], [22], an encoder-decoder architecture is used to learn features from the target and guidance images, and the filtering output is then regressed directly from the network. Learning-based techniques require a large number of ground-truth images for training.

Our method borrows from both explicit weighted-average methods and CNN-based ones. Unlike existing explicit joint filters [4], [7], [11], that use hand-crafted kernels and neighbors defined on a fixed regular grid, we leverage CNNs to learn the set of neighbors and their corresponding weights adaptively. Our method differs from previous CNN-based ones [5], [21], [22] in that we learn spatially-variant kernels for each pixel to obtain filtering results as a weighted average.

2.2 Variants of the spatial transformer [28]

Recent works introduce more flexible and effective CNN architectures. Jaderberg *et al*. propose a novel learnable module, the spatial transformer [28], that outputs the parameters of the desired spatial transformation (e.g., affine or thin plate spline) given a feature map or an input image. The spatial transformer makes a standard CNN network for classification invariant to a set of geometric transformation, but it has a limited capability of handling local transformations. Choy *et al*. introduce a convolutional version of the spatial transformer [29]. They learn local transformation parameters for normalizing orientation and scale in feature matching. Most similar to ours are the dynamic filter network [30] for video prediction and the adaptive convolution network [31] for video frame interpolation, where a set of local transformation parameters is generated adaptively conditioned on the input image. Compared to these works, our network is more general in that it is not limited to learning spatially-variant kernels, but it also learns the sampling locations of neighbors. This allows us to achieve state-of-the-art results even with kernels of size 3×3 on several tasks. Our work is also related to the deformable convolutional network [32]. The basic idea of deformable convolutions is to add offsets to the

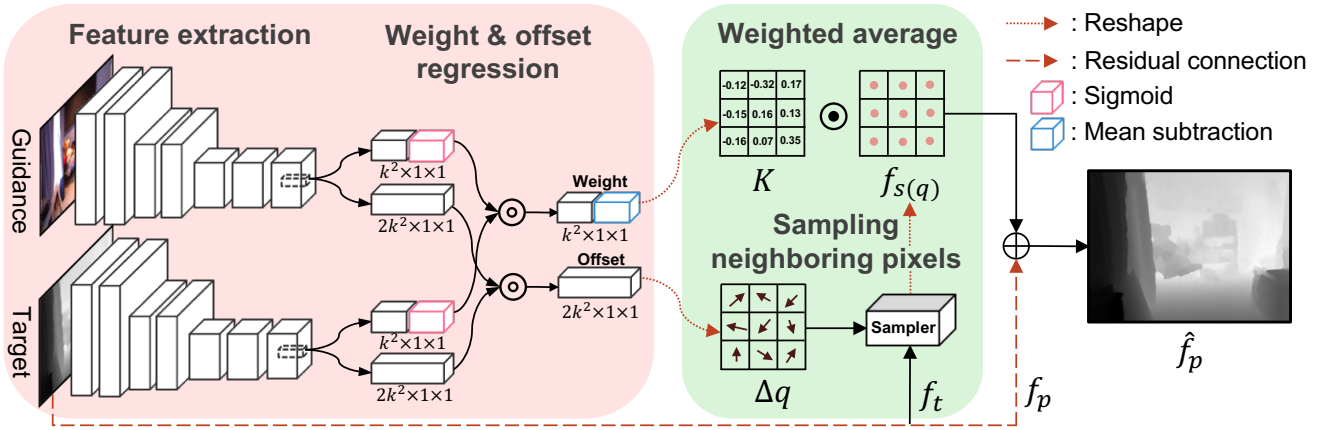


Fig. 2. The DKN architecture. We learn the kernel weights K and the spatial sampling offsets $\Delta\mathbf{q}$ from the feature maps of guidance and target images. To obtain the residual image $\hat{f}_\mathbf{p} - f_\mathbf{p}$, we then compute the weighted average with the kernel weights K and image values $f_{\mathbf{s(q)}}$ sampled at offset locations $\Delta\mathbf{q}$ from the neighbors $f_\mathbf{t}$. Finally, the result is combined with the target image $f_\mathbf{p}$ to obtain the filtering result $\hat{f}_\mathbf{p}$. Our model is fully convolutional and is learned end-to-end. We denote by $\circledcirc$ and $\odot$ element-wise multiplication and dot product, respectively. The reshaping operator and residual connection are drawn in dotted and dashed lines, respectively. See Table 1 for the detailed description of the network structure. (Best viewed in color.)

sampling locations defined on a regular grid in standard CNNs. The deformable convolutional network samples features directly from learned offsets, but shares the same weights for different sets of offsets as in standard CNNs. In contrast, we use spatially-variant weights for each sampling location. Another difference is that we use the learned offset explicitly to obtain the final result, while the deformable convolutional network uses it to compute intermediate feature maps.

3 Proposed Approach

In this section, we briefly describe our approach to learning both kernel weights and sampling locations for joint image filtering (Section 3.1), and present a concrete network architecture (Section 3.2) and its efficient implementation using a shift-and-stitch approach (Section 3.3). We then describe a fast version of DKN (Section 3.4).

3.1 Overview

Our network mainly consists of two parts (Fig. 2): We first learn spatially-variant kernel weights and spatial sampling offsets w.r.t the regular grid. To this end, a two-stream CNN [33], where each sub-network has the same structure (but different parameters), takes the guidance and target images to extract feature maps that are used to estimate the kernel weights and the offsets. We then compute a weighted average using the learned kernel weights and sampling locations computed from the offsets to obtain a residual image. Finally, the filtering result is obtained by combining the residuals with the target image. Our network is fully convolutional, does not require fixed-size input images, and it is trained end-to-end.

Weight and offset learning. It is hard to get supervision directly for kernel weights and offsets for individual pixels. We instead train our network with ground-truth target images and learn the kernel weights and offsets guided by an explicit weighted average. This is a kind of meta-supervision that specifies how the kernel weights and offsets behave. In particular, constraints on weight and offset regression (sigmoid and mean subtraction layers in Fig. 2) guide the learning process. For weight regression, we apply a sigmoid layer that makes all elements larger than 0. We then subtract the mean value from the output of the sigmoid layer so that the regressed weights should be similar to high-pass filter with kernel weights adding to 0. For offset regression, we do not apply the sigmoid layer since relative offsets (for x, y positions) from locations on a regular grid can have negative values.

Residual connection. The main reason behind using a residual connection is that the filtering result is largely correlated with the target image, and both share low-frequency content [17], [34], [35]. Focussing on learning the residuals also accelerates training speed while achieving better performance. Note that contrary to [17], [34], [35], we obtain the residuals by a weighted averaging process with the learned kernels, instead of estimating them directly from the network output. Empirically, the kernels learned with the residual connection have the same characteristics as the high-pass filters widely used to extract important structures (e.g., object boundaries) from images. Note also that we can train DKN without the residual connection. In this case, we compute the filtering result as the weighted average in (1).

3.2 DKN architecture

We design a fully convolutional network to learn the kernel weights and the sampling offsets for individual pixel. A detailed description of the network structure is shown in Table 1.

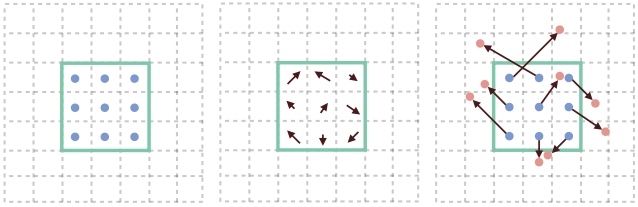
Feature extraction. We adapt a similar architecture in [31] for feature extraction that consists of 7 convolutional layers. We input the guidance and target images to each of sub-networks that gives a feature map of size $128 \times 1 \times 1$ for the receptive field of size 51×51. We use the ReLU [23] as an activation function and batch normalization [36] for regularization. In case of joint upsampling tasks (e.g., depth map upsampling), we input a high-resolution guidance image (e.g., a color image) and a low-resolution target image (e.g., a depth map) upsampled using bicubic interpolation.

Weight regression. For each sub-network, we add a 1×1 convolutional layer on top of the feature extraction layer. It gives a feature map of size $k^2 \times 1 \times 1$, where k is the size of the filter kernel, which is used to regress the kernel weights. To estimate the weights, we apply a sigmoid layer to each feature map of size $k^2 \times 1 \times 1$, and then combine the outputs by element-wise multiplication (see Fig. 2). We could use a softmax layer as in [31], but empirically find that it does not perform as well as the sigmoid layer. The softmax function encourages the estimated kernel to have only a few non-zero elements, which is not appropriate

TABLE 1
Network architecture details. "BN" and "Res." denote the batch normalization [36] and residual connection, respectively. We denote by "DownConv" convolution with stride 2. The inputs of our network are 3-channel guidance and 1-channel target images (denoted by D). For the model without the residual connection, we use an L1 normalization layer (denoted by "L1 norm.") instead of subtracting mean values for weight regression.

Feature extraction	
Type	Output
Input	$D \times 51 \times 51$
Conv(7×7)-BN-ReLU	$32 \times 45 \times 45$
DownConv(2×2)-ReLu	$32 \times 22 \times 22$
Conv(5×5)-BN-ReLU	$64 \times 18 \times 18$
DownConv(2×2)-ReLU	$64 \times 9 \times 9$
Conv(5×5)-BN-ReLU	$128 \times 5 \times 5$
Conv(3×3)-ReLU	$128 \times 3 \times 3$
Conv(3×3)-ReLU	$128 \times 1 \times 1$

Weight regression	
Type	Output
Conv(1×1)	$k^2 \times 1 \times 1$
Sigmoid	$k^2 \times 1 \times 1$
Mean subtraction or L1 norm. (w/o Res.)	$k^2 \times 1 \times 1$
Offset regression	
Type	Output
Conv(1×1)	$2k^2 \times 1 \times 1$



(a) (b) (c)

Fig. 3. Illustration of irregular sampling of neighboring pixels using offsets: (a) regular sampling $\mathbf{q}$ on discrete grid; (b) learned offsets $\Delta\mathbf{q}$; (c) deformable sampling locations $\mathbf{s}(\mathbf{q})$ with the offsets $\Delta\mathbf{q}$. The learned offsets are fractional and the corresponding pixel values are obtained by bilinear interpolation.

for image filtering. The estimated kernels should be similar to high-pass filters, with kernel weights adding to 0. To this end, we subtract the mean value from the combined output of size $k^2 \times 1 \times 1$. For our model without a residual connection, we apply instead L1 normalization to the output of size $k^2 \times 1 \times 1$. Since the sigmoid layer makes all elements in the combined output larger than 0, applying L1 normalization forces the kernel weights to add to 1 as in (2).

Offset regression. Similar to the weight regression case, we add a 1×1 convolutional layer on top of the feature extraction layer. The resulting two feature maps of size $2k^2 \times 1 \times 1$ are combined by element-wise multiplication. The final output contains relative offsets (for x, y positions) from locations on a regular grid. In our implementation, we use 3×3 kernels, but the filtering result is computed by aggregating 9 samples sparsely chosen from a much larger neighborhood. The two main reasons behind the use of small-size kernels are that (1) the size of the receptive field and the reliability of samples are much more important than the total number of samples aggregated, and (2) this enables an efficient implementation in terms of speed and memory. A similar finding is noted in [37], which shows that only high-confidence samples should be chosen when estimating foreground and background images in image matting. Note that offset regression is closely related to nonlocal means [38] in that both select which pixels to aggregate instead of immediate neighbors.

Weighted average. Given the learned kernel K and sampling offsets $\Delta\mathbf{q}$, we compute the residuals $\hat{f}_\mathbf{p} - f_\mathbf{p}$ as a weighted average:

$$\hat{f}_\mathbf{p} - f_\mathbf{p} = \sum_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} K_{\mathbf{ps}}(f, g) f_{\mathbf{s}(\mathbf{q})}, \tag{3}$$

where $\mathcal{N}(\mathbf{p})$ is a local 3×3 window centered at the location $\mathbf{p}$ on a regular grid (Fig. 3(a)). We denote by $\mathbf{s}(\mathbf{q})$ the sampling position computed from the offset $\Delta\mathbf{q}$ (Fig. 3(b)) of the location $\mathbf{q}$ as follows.

$$\mathbf{s}(\mathbf{q}) = \mathbf{q} + \Delta\mathbf{q}. \tag{4}$$

The sampling position $\mathbf{s}(\mathbf{q})$ predicted by the network is irregular and typically fractional (Fig. 3(c)). We use bilinear interpolation [28] to sample corresponding (sub) pixels $f_{\mathbf{s}(\mathbf{q})}$ as

$$f_{\mathbf{s}(\mathbf{q})} = \sum_{\mathbf{t} \in \mathcal{R}(\mathbf{s}(\mathbf{q}))} G(\mathbf{s}, \mathbf{t}) f_\mathbf{t}, \tag{5}$$

where $\mathcal{R}(\mathbf{s}(\mathbf{q}))$ enumerates all integer locations in a local 4-neighborhood system to the fractional position $\mathbf{s}(\mathbf{q})$, and G is a two-dimensional bilinear kernel. We split the kernel G into two one-dimensional ones [28], [32] as

$$G(\mathbf{s}, \mathbf{t}) = g(s_x, t_x) g(s_y, t_y), \tag{6}$$

where $g(a, b) = max(0, 1 - |a - b|)$. Note that the residual term in (3) is exactly the same as the explicit joint filters in (1), but we aggregate pixels from the sparsely chosen locations $\mathbf{s}(\mathbf{q})$ with the learned kernels K. Note that aggregating pixels from a fractional grid is not feasible in current joint filters including DJF [17].

When we do not use a residual connection, we compute the filtering result $\hat{f}_\mathbf{p}$ directly as a weighted average using the learned kernels and offsets:

$$\hat{f}_\mathbf{p} = \sum_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} K_{\mathbf{ps}}(f, g) f_{\mathbf{s}(\mathbf{q})}. \tag{7}$$

Loss. We train our model by minimizing the L_1 norm of the difference between the network output $\hat{f}$ and ground truth f^{gt} as follows.

$$L(f^{\text{gt}}, \hat{f}) = \sum_\mathbf{p} |f^{\text{gt}}_\mathbf{p} - \hat{f}_\mathbf{p}|_1. \tag{8}$$

3.2.1 *Efficient implementation*

We compute the filtering results for inputs of any size in a single forward pass, since our network is fully convolutional [39]. The output dimensions are, however, reduced by factor of 4 in each dimension due to the use of convolutions with multiple strides to extract features ("DownConv" in Table 1). A pixel-wise implementation (Fig. 2) prevents this problem, but it requires a total of N forward passes in case of an image of size $N(= HW)$ pixels, where H and W are hight and width, respectively.

We use instead the shift-and-stitch approach [31], [39] that stitches the network outputs from shifted versions of the input (Fig. 4). We can obtain the same result as the pixel-wise implementation in 16 forward passes. We first shift input images

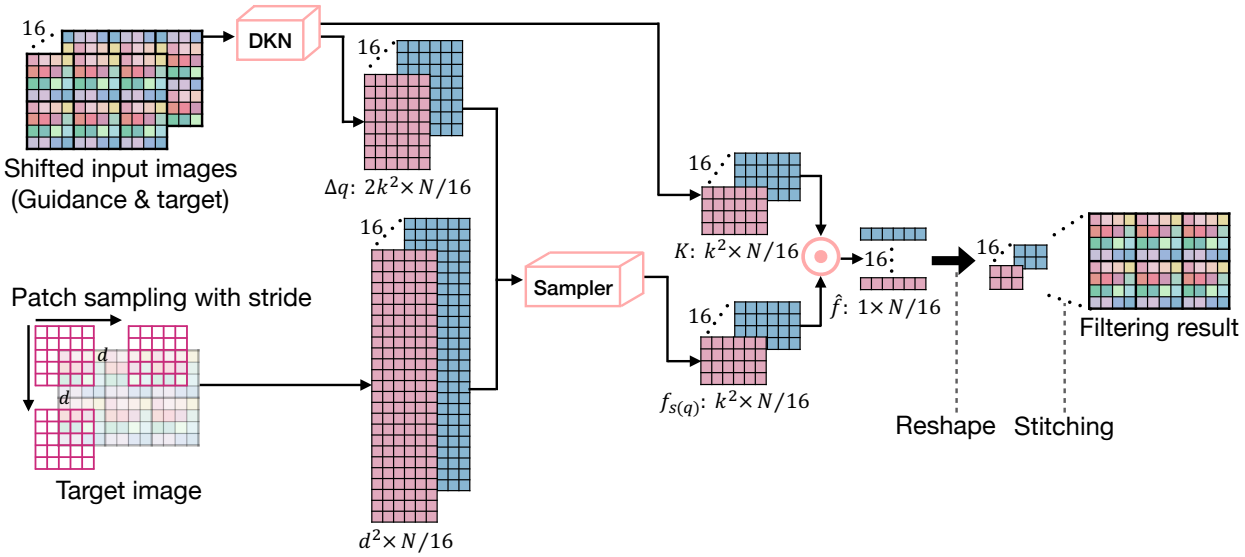


Fig. 4. Efficient implementation using a shift-and-stitch approach. We denote d by the maximum range of the sampling location $\mathbf{s}(\mathbf{q})$. We shift the input images and compute the filtering result for each shifted input. We then stitch them up to get a filtering result that has the same resolution as the inputs. Our approach makes it possible to reuse the storage for kernel weights, offsets, and resampled pixels. See text for details. (Best viewed in color.)

x pixels to the left and y pixels up, once for every (x, y) where $\{(x, y)|0 \leq x, y \leq 3\}$, and obtain total 16 shifted inputs. Each shifted input goes through the network that gives the kernel weights K and the offsets $\Delta\mathbf{q}$ of size $k^2 \times N/16$ and $2k^2 \times N/16$, respectively. The next step is to obtain image values $f_{\mathbf{s}(\mathbf{q})}$ using the sampling function $\mathbf{s}(\mathbf{q})$ from the target image. To this end, starting from every location (x, y) in the target image, we sample patches of size $d \times d$ with stride 4 in each dimension, each of which gives the output of size $d^2 \times N/16$. The patch size corresponds to the maximum range of the sampling position $\mathbf{s}(\mathbf{q})$. For an efficient implementation, we restrict the range (e.g., to 15×15 in our experiment). We then sample k^2 pixels using the sampling position $\mathbf{s}(\mathbf{q})$ from patches of size $d \times d$, obtaining $f_{\mathbf{s}(\mathbf{q})}$ of size $k^2 \times N/16$ for each shifted input. To compute a weighted average, we apply element-wise multiplication between the kernel weights K and the corresponding sampled pixels $f_{\mathbf{s}(\mathbf{q})}$ of size $k^2 \times N/16$ followed by column-wise summation, resulting in an output of size $1 \times N/16$. Finally, we stitch 16 outputs of size $1 \times N/16$ into a single one to get the final output. Note that one can stitch kernel weights and offsets first and then compute a weighted average, but this requires a large amount of memory. We stitch instead the outputs after the weighted average, and reuse the storage for kernel weights, offsets, and sampled pixels.

3.3 FDKN architecture

We now present a fast version of DKN (FDKN) that achieves a $4\times$ speed-up over the DKN for an image of size 640×480 while retaining the same level of quality (Fig. 5). The basic idea is to remove DownConv layers while maintaining the same receptive field size as the DKN, making it possible to obtain the filtering output in a single forward pass. Simply removing these layers leads to an increase in the total number of convolutions (see Section 6). We instead leverage resampling [40] (Fig. 6) that enables covering large receptive fields using small-size convolutions compared to the DKN. This also reduces the total number of network parameters to estimate. Particularly, we resample guidance and interpolated target images separately with stride 4 in each channel, and obtain resampled ones of size $H/4 \times W/4 \times 16C$ and $H/4 \times W/4 \times 16$, respectively, where C is the number of channels. This has an effect of enlarging receptive fields by factor of 4 in each dimension. We may use dilated convolutions [41] that support large receptive fields without loss of resolution, but empirically find that runtime gain is marginal (see Section 6). The FDKN takes as inputs resampled guidance and target images, and extracts features for weight and offset regression by 6 convolutional layers of size 3×3, which maintains a receptive field size comparable with the DKN. Different from the DKN, FDKN regresses kernel weights and offsets for all pixels simultaneously, allowing to obtain the filtering output in a single forward pass. After estimating weights and offset locations, we combine the residuals computed by a weighted average with the interpolated target image. We show in Table 2 the detailed description of the network structure.

4 Experiments

In this section we present a detailed analysis and evaluation of our approach. We describe implementation details including our experimental protocol and training/testing setups (Section 4.1). We then apply our models to the tasks of joint depth or saliency image upsampling (Section 4.2), cross-modality image restoration and texture removal[1] (Section 4.3) and semantic segmentation (Section 4.4), and compare them to the state of the art in each case. The results for all comparisons have been obtained from the source code provided by the authors.

4.1 Experimental details

4.1.1 *Joint image upsampling and noise removal*

Following the experimental protocol of [5], [17], we train our models on the tasks of depth map upsampling and depth noise removal using a large number of RGB/D image pairs, and test them on depth map upsampling and other joint filtering tasks (e.g., cross-modality image restoration) that require selectively transferring the structural details from the guidance image to the target one.

1. We only show qualitative results in this task since ground truth is not available.

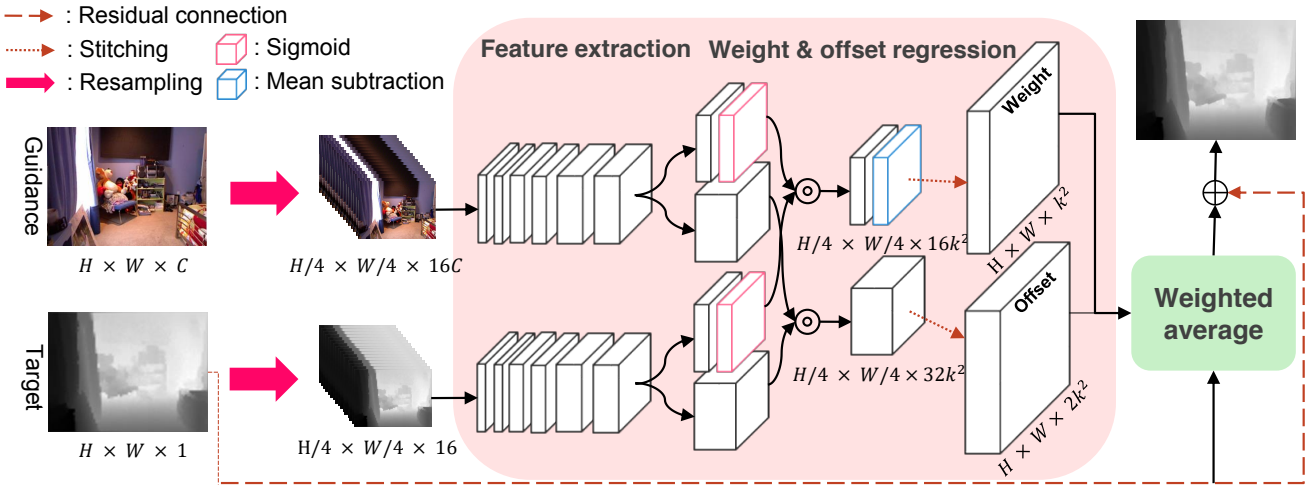


Fig. 5. The FDKN architecture. We resample a guidance image of size $H \times W \times C$ with stride 4 in each dimension (Fig. 6), where H, W and C are hight, width and the number of channels, respectively. This gives resampled guidance images of size $H/4 \times W/4 \times 16C$. The target image is also resampled to the size of $H/4 \times W/4 \times 16$. This allows the FDKN to maintain a receptive field size comparable with the DKN. The FDKN inputs resampled guidance and target images, and then computes the kernel weight and offset locations of size $H \times W \times k^2$ and $H \times W \times 2k^2$, respectively, making it possible to get filtering results in a single forward pass without loss of resolution. Finally, the residuals computed by the weighted average and the target image are combined to obtain the filtering result. See Table 2 for the detailed description of the network structure. (Best viewed in color.)

TABLE 2
FDKN architecture details. The inputs of FDKN are $16C$-channel guidance and 1-channel target images (denoted by D). Note that the receptive field of size 13×13 in the resampled images is comparable to that of size 51×51 in the DKN.

Feature extraction		Weight regression	
Type	Output	Type	Output
Input	$D \times 13 \times 13$	Conv(1×1)	$16k^2 \times 1 \times 1$
Conv(3×3)-BN-ReLU	$32 \times 11 \times 11$	Sigmoid	$16k^2 \times 1 \times 1$
Conv(3×3)-ReLU	$32 \times 9 \times 9$	Mean subtraction or	$16k^2 \times 1 \times 1$
Conv(3×3)-BN-ReLU	$64 \times 7 \times 7$	L1 norm. (w/o Res.)	
Conv(3×3)-ReLU	$64 \times 5 \times 5$	Offset regression	
Conv(3×3)-BN-ReLU	$128 \times 3 \times 3$	Type	Output
Conv(3×3)-ReLU	$128 \times 1 \times 1$	Conv(1×1)	$32k^2 \times 1 \times 1$

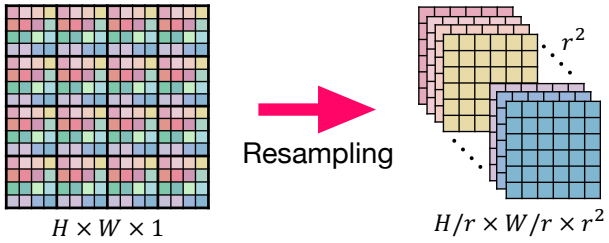


Fig. 6. Illustration of resampling. An image of size $H \times W \times 1$ is reshaped with stride r in each dimension, resulting a resampled one of size $H/r \times W/r \times r^2$.

Training. We sample 1,000 RGB/D image pairs of size 640×480 from the NYU v2 dataset [44]. We use the same image pairs as in [5], [17] to train the networks. We divide each image in the 1,000 training pairs into two halves of size 320×480 due to the lack of GPU memory, and use the corresponding 2,000 RGB/D image pairs as training samples. We train different models for joint image upsampling and other tasks. For joint image upsampling, the models are trained with a batch size of 1 for 40k iterations, giving roughly 20 epochs over the training data. We synthesize low-resolution depth images ($4\times$, $8\times$, $16\times$) from ground truth by bicubic downsampling. The models for depth noise removal are similarly trained but with 4k iterations under the guidance of high-resolution RGB images. Noisy depth images are synthesized by adding Gaussian noise with zero mean and variance of 0.005. We use the Adam optimizer [46] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. As learning rate we use 0.001 and divide it by 5 every 10k iterations. Regularization techniques such as weight decay, dropout, and data augmentation are not used since the training dataset is sufficiently large to train our models. All networks are trained end-to-end using `PyTorch` [47].

Testing. We can obtain filtering results of DKN for inputs of any size by stitching coarse outputs, which is far more efficient than a brute-force implementation. Filtering results for FDKN are obtained in a single pass. The inputs of our network are 3-channel guidance and 1-channel target images. In case of a 1-channel guidance image (e.g., RGB/NIR image restoration), we create a 3-channel image by duplicating the single channel three times. For multi-channel target images (e.g., in texture removal), we apply our model separately in each channel and combine the outputs.

4.1.2 *Semantic segmentation*

Following the experimental protocol of [48], we plug FDKN in DeepLab-v2 [49], which uses ResNet-101 [35] pretrained for ImageNet classification, as a part of CNNs for semantic

TABLE 3
Quantitive comparison with the state of the art on depth map upsampling in terms of average RMSE. Numbers in bold indicate the best performance and underscored ones are the second best.

Datasets	Middlebury [42]			Lu [43]			NYU v2 [44]			Sintel [18]		
Methods	4×	8×	16×	4×	8×	16×	4×	8×	16×	4×	8×	16×
Bicubic Int.	4.44	7.58	11.87	5.07	9.22	14.27	8.16	14.22	22.32	6.54	8.80	12.17
MRF [45]	4.26	7.43	11.80	4.90	9.03	14.19	7.84	13.98	22.20	8.81	11.77	15.75
GF [7]	4.01	7.22	11.70	4.87	8.85	14.09	7.32	13.62	22.03	6.10	8.22	11.22
JBU [4]	2.44	3.81	6.13	2.99	5.06	7.51	4.07	8.29	13.35	5.88	7.63	10.97
TGV [3]	3.39	5.41	12.03	4.48	7.58	17.46	6.98	11.23	28.13	32.01	36.78	43.89
Park [2]	2.82	4.08	7.26	4.09	6.19	10.14	5.21	9.56	18.10	9.28	12.22	16.51
SDF [6]	3.14	5.03	8.83	4.65	7.53	11.52	5.27	12.31	19.24	6.52	7.98	11.36
FBS [15]	2.58	4.19	7.30	3.03	5.77	8.48	4.29	8.94	14.59	11.96	12.29	13.08
DMSG [21]	1.88	3.50	6.28	2.30	4.14	7.22	3.02	5.40	9.17	5.32	7.23	10.11
DJF [5]	2.14	3.77	6.12	2.54	4.71	7.66	3.54	6.20	10.21	5.51	7.52	10.63
DJFR [17]	1.98	3.61	6.07	2.22	4.54	7.48	3.38	5.86	10.11	5.50	7.43	10.48
FDKN w/o Res.	1.12	2.23	4.52	0.85	2.19	5.15	1.88	3.67	7.13	3.38	5.02	7.74
FDKN	**1.08**	2.17	4.50	**0.82**	**2.10**	**5.05**	1.86	3.58	6.96	3.36	4.96	7.74
DKN w/o Res.	1.26	2.16	4.32	0.99	2.21	5.12	1.66	3.36	6.78	3.36	4.82	**7.48**
DKN	1.23	**2.12**	**4.24**	0.96	2.16	5.11	**1.62**	**3.26**	**6.51**	**3.30**	**4.77**	7.59

segmentation[2], instead of applying a fully connected conditional random field (CRF) [50] to refine segmentation results. That is, we integrate DeepLab-v2 and our model and train the whole network end-to-end, avoiding an offline post-processing using CRFs.

Training. We use the Pascal VOC 2012 dataset [51] that contains $1,464$, $1,449$, and $1,456$ images for training, validation and test, respectively. Following [48], [49], we augment the training dataset by the annotations provided by [52], resulting in $10,582$ images, and use $1,449$ images in the validation set for evaluation. We train the network using a softmax log loss with a batch size of 1 for 20k iterations. The SGD optimizer with momentum of 0.9 is used. As learning rate, we use the scheduling method of [49] with learning rate of 2.5×10^{-4} and 2.5×10^{-3} for DeepLab-v2 and FDKN, respectively.

Testing. We upsample 21-channel outputs (20 object classes and background) of DeepLab-v2 before a softmax layer using a high-resolution color image. We apply the FDKN separately in each channel.

4.2 Joint image upsampling experiments

We train different models to upsample depth images for scale factors $4\times$, $8\times$, $16\times$ with RGB/D image pairs from the NYU v2 dataset [44], and apply them to the tasks of depth map and saliency map upsampling. The inputs to our models are a high-resolution color image and a low-resolution depth image upsampled using bicubic interpolation.

Depth map upsampling. We test our models on depth map upsampling with the following four benchmark datasets. These datasets feature aligned color and depth images.

- Middlebury dataset [42], [53]: We use the 30 RGB/D image pairs from the 2001-2006 datasets provided by Lu [43].
- Lu dataset [43]: This provides 6 RGB/D image pairs acquired by the ASUS Xtion Pro camera [54].
- NYU v2 dataset [44]: It consists of 1,449 RGB/D image pairs captured with the Microsoft Kinect [55]. We exclude the 1,000 pairs used for training, and use the rest (449 pairs) for evaluation
- Sintel dataset [18]: This dataset provides 1,064 RGB/D image pairs created from an animated 3D movie. It contains realistic scenes including fog and motion blur. We use 864 pairs from a final-pass dataset for testing.

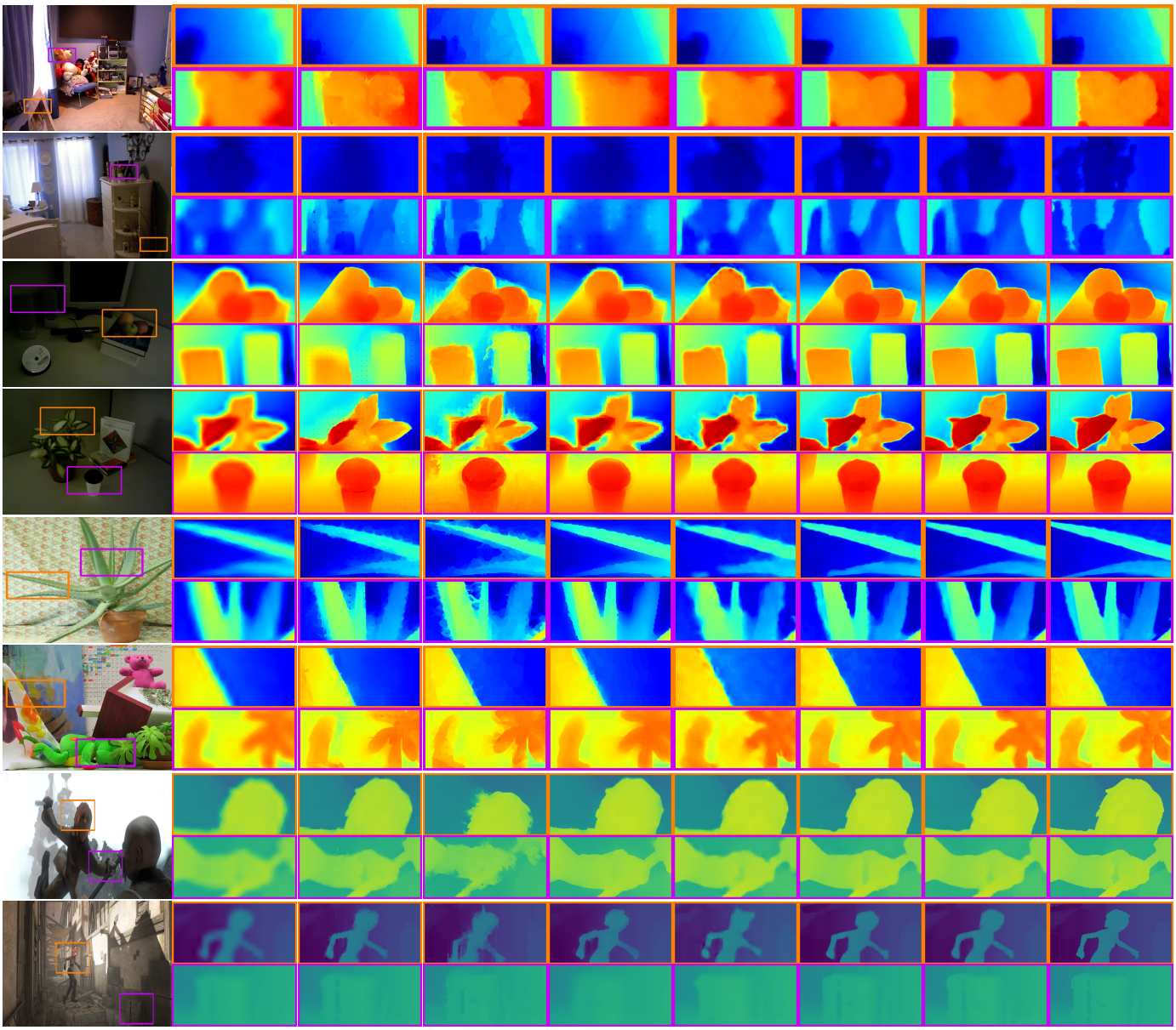
2. We use a PyTorch version available online: https://github.com/isht7/pytorch-deeplab-resnet

TABLE 4
Quantitative comparison on saliency map upsampling in terms of weighted F-scores [56]. We use 5,168 images from the DUT-OMRON dataset [57].

	Bicubic Int.	GF [7]	SDF [6]	DJFR [17]	DKN	FDKN
Weighted F-score	0.8860	0.8895	0.8851	0.9248	0.9440	**0.9606**

We compare our method with the state of the art in Table 3. It shows the average root mean squared errors (RMSE) between upsampling results and ground truth. The results of DJF [5] and its residual version (DJFR [17]) are obtained by the provided models trained with the NYU v2 dataset. DMSG [21] uses the Middlebury and Sintel datasets for training the network. For fair comparison of DMSG with other CNN-based methods including ours, we retrain the DMSG model using the same image pairs from the NYU v2 dataset as in [5], [17]. From this table, we can see that (1) our models outperform the state of the art including CNN-based methods [5], [17], [21] by significant margins in terms of RMSE, even without the residual connection (DKN w/o Res. and FDKN w/o Res.); (2) they perform well on both synthetic and real datasets (e.g., the Sintel and NYU v2 datasets), and generalize well for other images (e.g., on the Middlebury dataset) outside the training dataset; and (3) FDKN retains the superior performance of DKN, and even outperforms DKN for the Lu dataset. Figure 7 shows a visual comparison of the upsampled depth images ($8\times$). The better ability to extract common structures from the target and guidance images by our models here is clearly visible. Specifically, our results show a sharp depth transition without the texture-copying artifacts. In contrast, artifacts are clearly visible even in the results of DJFR, which tends to over-smooth the results and does not recover fine details. This confirms once more the advantage of using spatially-variant kernels and an adaptive neighborhood system in joint image filtering.

Saliency map upsampling. To evaluate the generalization ability of our models on other tasks, we apply them trained with the NYU v2 dataset to saliency map upsampling without fine-tuning. We downsample saliency maps ($\times 8$) in the DUT-OMRON



(a) RGB image. (b) GF [7]. (c) TGV [3]. (d) Park [2]. (e) SDF [6]. (f) DJFR [17]. (g) DKN. (h) FDKN. (i) Ground truth.

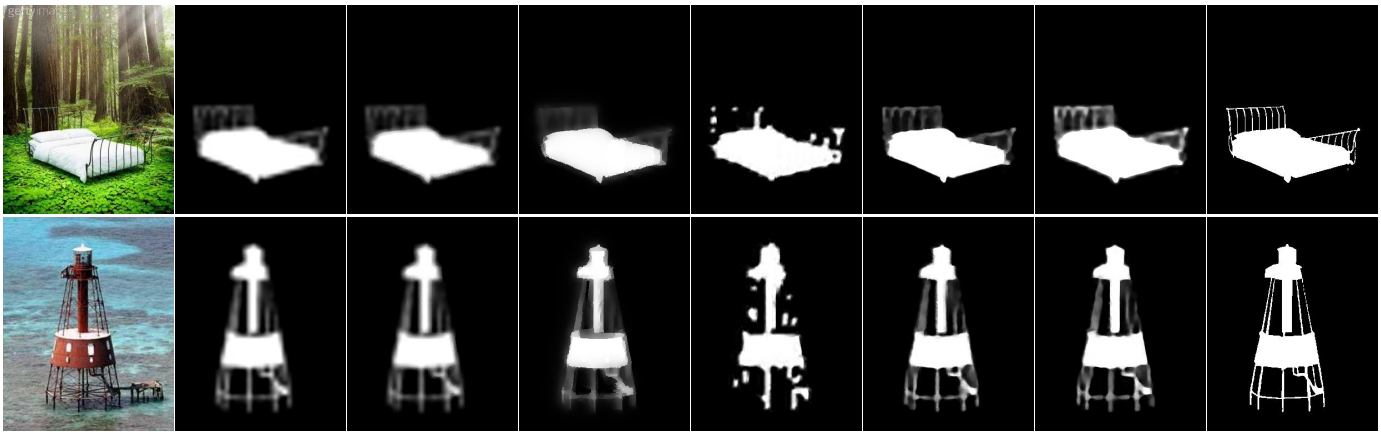
Fig. 7. Visual comparison of upsampled depth images ($8\times$). Top to bottom: Each two rows show upsampled images on the NYU v2 [44], Lu [43], Middlebury [42] and Sintel [18] dataset, respectively. Note that we train our models with the NYU v2 dataset, and do not fine-tune them to other datasets.

dataset [57], and then upsample them under the guidance of high-resolution color images. We show in Table 4 a comparison of weighted F-measure [56] between upsampled images and the ground truth. Figure 8 shows examples of the upsampling results by the state of the art and our models. The results show that our models outperform others including a CNN-based method [17].

4.3 Other joint filtering experiments

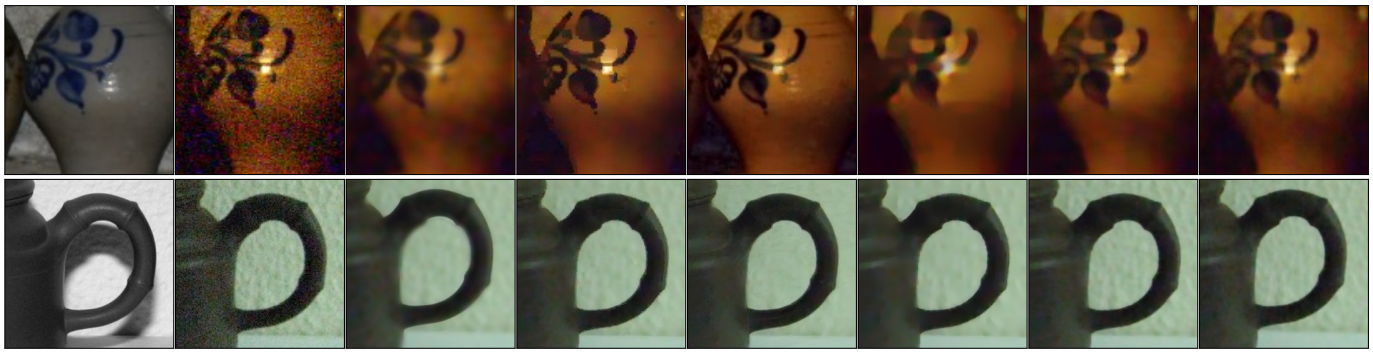
Following [5], [17], we use depth denoising as a proxy task for cross-modality image restoration and texture removal, since ground-truth datasets for these tasks are not available. We train the networks for denoising depth images with RGB/D image pairs from the NYU v2 dataset [44]. The models are then applied to the tasks of cross-modality image restoration and texture removal without fine-tuning. We do not use the residual connection for noise removal tasks, since we empirically find that it does not help in this case. For cross-modality image restoration and texture removal tasks, all previous works (e.g., [5], [7], [9], [17]) we are aware of for these tasks offer qualitative results only. For comparison, average RMSE for GF [7], Yan [9], SDF [6], DJF [5], and DKN on depth noise removal are 5.34, 12.53, 7.56, 2.63, and 2.46, respectively, in the test split of the NYU v2 dataset, showing that our model again outperforms the others.

Cross-modality image restoration. For flash/non-flash denoising, we set the flash and non-flash images as guidance and target ones, respectively. Similarly, we restore the color image guided by the flash NIR image in RGB/NIR denoising. Examples for flash/non-flash and RGB/NIR restoration are shown in Fig. 9. Qualitatively, our models outperform other state-of-the-art methods [5], [6], [7], and give comparable results to those of Yan [9] that is specially designed for this task. In particular, they preserve edges while smoothing noise without artifacts. This demonstrates that our models trained with RGB/D images can generalize well for others with different modalities.



(a) RGB image. (b) Bicubic Int. (c) GF [7]. (d) SDF [6]. (e) DJFR [17]. (f) DKN. (g) FDKN. (h) Ground truth.

Fig. 8. Visual comparison of saliency map upsampling ($8\times$) on the DUT-OMRON dataset [57].



(a) Guidance. (b) Target. (c) GF [7]. (d) SDF [6]. (e) Yan [9]. (f) DJF [5]. (g) DKN. (h) FDKN.

Fig. 9. Examples of cross-modality noise reduction for (top) flash/non-flash denoising and (bottom) RGB/NIR denoising. Our models preserve textures while smoothing noise. GF and DJF tend to over-smooth the textures. Artifacts are clearly visible in the results of SDF. The method of [9], specially designed for this task, gives the best results.

(a) Input image and snippets. (b) RGF [11]. (c) RTV [10]. (d) Cov [12]. (e) SDF [6]. (f) DJF [5]. (g) DKN. (h) FDKN.

Fig. 10. Visual comparison of texture removal for regular (top) and irregular (bottom) textures.

Texture removal. We set the textured image itself for guidance and target, and apply our models repeatedly to remove small-scale textures. We show examples in Fig. 10. Compared to the state of the art, our models remove textures without artifacts while maintaining other high-frequency structures such as image boundaries and corners. A plausible explanation of why networks trained for denoising depth images work for texture removal is that textures can be considered as patterned noise. Repeatedly applying our networks thus removes them. A similar finding can be found in [5]. We empirically find that 4 iterations are enough to get satisfactory results, and use the same number of iterations for all experiments.

4.4 Semantic segmentation experiments

CNNs commonly use max pooling and downsampling to achieve invariance, but this degrades localization accuracy especially at object boundaries [39]. DeepLab [49] overcomes this problem

(a) RGB image. (b) Baseline [49]. (c) DenseCRF [50]. (d) DGF [48]. (e) FDKN. (f) Ground truth.

Fig. 11. Visual comparison of semantic segmentation on the validation set of Pascal VOC 2012 benchmark [51]. Compared to the state of the art, our model shows better ability to improve the localization accuracy of object boundaries and refine incorrectly labeled segments.

TABLE 6
Average RMSE comparison (DKN/FDKN) of different components and size of kernels (from 3×3 to 25×25). From the third row, we can see that aggregating pixels from a 15×15 window is enough. We thus restrict the maximum range of offset locations to 15×15. For example, results for 7×7 in the forth row are computed using 49 pixels sparsely sampled from a 15×15 window. We omit the results for 15×15, 19×19 and 25×25 kernels, since they are equal to or beyond the maximum range of offset locations.

Weight learning		Offset learning		Res.	3×3	5×5	7×7	15×15	19×19	25×25
RGB	Depth	RGB	Depth							
✓					5.92 / 6.05	5.52 / 5.73	5.43 / 5.67	5.59 / 5.74	5.82 / 5.81	6.21 / 5.99
	✓				5.24 / 5.30	4.36 / 4.47	4.09 / 4.24	4.09 / 4.17	4.11 / 4.18	4.15 / 4.21
✓	✓				5.03 / 5.14	3.90 / 4.16	3.48 / 3.80	3.32 / 3.66	3.33 / 3.66	3.39 / 3.72
✓		✓			5.37 / 5.18	5.38 / 5.09	5.40 / 5.07	–	–	–
✓	✓	✓	✓		3.36 / 3.67	3.32 / 3.65	3.33 / 3.66	–	–	–
✓	✓	✓	✓	✓	3.26 / 3.58	3.21 / 3.53	**3.19** / **3.52**	–	–	–

TABLE 5
Quantitative comparison on semantic segmentation in terms of average IoU. We use 1,449 images from the validation set in the Pascal VOC 2012 benchmark [51].

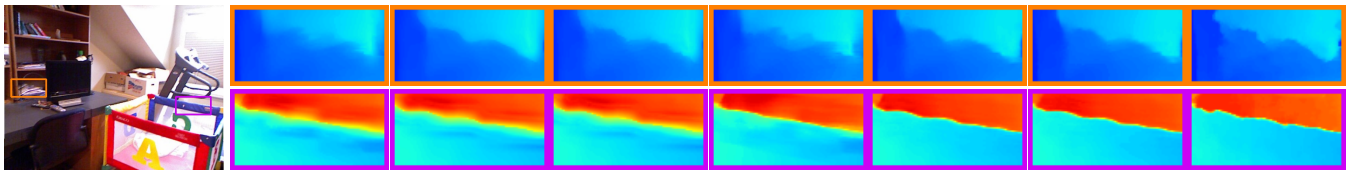
Methods	Baseline [49]	DenseCRF [50]	DGF [48]	FDKN
Mean IoU	70.69	71.98	72.96	**73.60**

using probabilistic graphical models. It applies a fully connected CRF [50] to the response of the final layer of a CNN. Zheng *et al*. [58] interpret CRFs as recurrent neural networks which are then plugged in as a part of a CNN, making it possible to train the whole network end-to-end. Recently, Wu *et al*. [48] have proposed a layer to integrate guided filtering [7] into CNNs. Instead of using CRFs [49], [50], [58] or guided image filtering [7], we apply the FDKN to the response of the final layer of DeepLab v2 [49], before CRFs. Note that FDKN is fully differentiable and the whole network is trained end-to-end. We show in Table 5 mean intersection-over-union (IoU) scores for the validation set in the Pascal VOC 2012 benchmark [51]. To the baseline method (DeepLab v2 w/o CRF [49]), we add CRF [50], guided filtering [48], or FDKN layers. This table shows that our model quantitatively yields better accuracy in terms of mean IoU than other state-of-the-art methods. Examples of semantic segmentation are shown in Fig. 11. Our model outperforms other methods qualitatively as well: It improves the localization of object boundaries (first row) and refines incorrect labels (second row).

5 Ablation Study

In this section, we conduct an ablation analysis on different components and parameters in our models. We report the results for depth map upsampling ($8\times$) on the NYU v2 dataset.

Network architecture. We show the average RMSE for five variants of our models in Table 6. The baseline model learns kernel weights from the guidance image only. The first row shows that this baseline already outperforms the state of the art (see Table 3). From the second row, we can see that our models trained using the target image only gives better results than the baseline, indicating that using the guidance image only is not enough to fully exploit common structures. The third row demonstrates that constructing kernels from both guidance and target images boosts performance. For example, the average RMSE of DKN decreases from 5.92 to 5.03 for the 3×3 kernel. We can also see that the effect of learning kernel weights and offsets from both inputs is significant, and combining all components including the residual connection gives the best results. For example, learning the offsets significantly boosts the performance of DKN from 5.03 to 3.36 for the 3×3 kernel, and the residual connection further decreases the average RMSE to 3.26. Figure 12 shows a visual comparison of using different networks for depth upsampling. Note that learning to predict the spatial offsets is important because (1) learning spatially-variant kernels for individual pixels would be very hard otherwise, unless using much larger kernels to achieve the same neighborhood size, which would lead to an inefficient implementation, and (2) contrary to current architectures including DJF [5] and DMSG [21], this allows sub-pixel information aggregation.



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

Fig. 12. Visual comparison of different networks for depth upsampling ($8\times$) using the kernel of size 3×3. (a) RGB image. Results for weight regression using (b) RGB, (c) depth and (d) RGB and depth images. Results for weight and offset regression using (e) RGB, (f) RGB and depth images, and (g) RGB and depth images with the residual connection. (h) Ground truth.

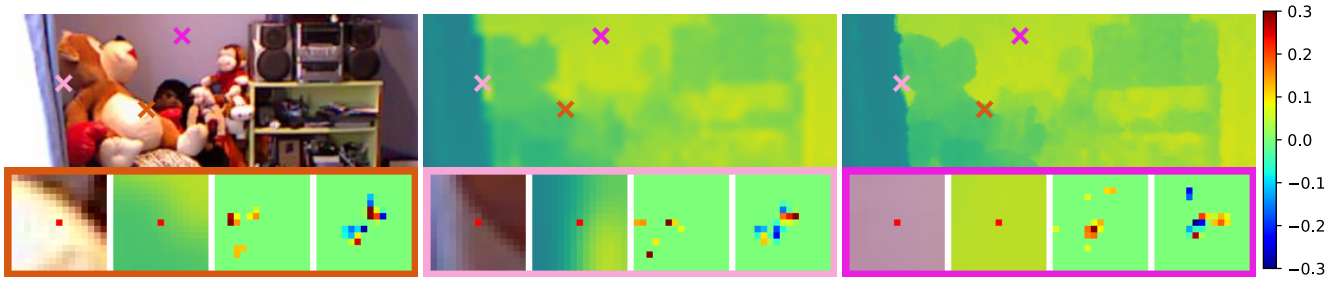


Fig. 13. Visualization of filter kernels. Top: (From left to right) a RGB image, a low-resolution depth image, and an upsampling result by DKN. Bottom: (From left to right) Snippets of RGB images, low-resolution depth images, and kernels learned w/o and w/ the residual connection. The center positions in the RGB and depth images are denoted by red dots. The kernel weights are plotted with a heat map. (Best viewed in color.)

TABLE 7
Quantitative comparison of using different number of channels for feature extraction (DKN/FDKN). We denote n_i $(i = 1, 2)$ by the number of channels in the final two layers for feature extraction. For each network, numbers in bold indicate the best performance and underscored ones are the second best.

n_1	128	256	256	256	256
n_2	128	128	256	512	1024
RMSE	3.26 / 3.58	3.22 / 3.51	3.20 / 3.49	3.17 / 3.46	**3.15** / **3.42**
Runtime (s)	**0.22** / **0.06**	0.25 / **0.06**	0.27 / **0.06**	0.32 / 0.07	0.44 / 0.07
Number of parameter (M)	**1.1** / **0.6**	1.7 / 1.1	2.3 / 1.7	3.5 / 2.9	5.8 / 5.9
Model size (MB)	**4.5** / **2.8**	6.7 / 4.5	9.0 / 7.3	14.0 / 13.0	23.0 / 24.2

TABLE 8
RMSE comparison of DKN by varying the number of training data on depth map upsampling ($8\times$).

Number of training data	10	50	100	200	500	700	1000
Middlebury [42]	2.88	2.35	2.24	2.14	2.13	2.10	2.12
Lu [43]	3.02	2.54	2.33	2.21	2.15	2.18	2.16
NYU v2 [44]	4.73	3.97	3.62	3.33	3.27	3.25	3.26
Sintel [18]	5.79	5.24	5.01	4.85	4.82	4.74	4.77

Kernel size. Table 6 also compares the performances of networks with different size of kernels. We enlarge the kernel size gradually from 3×3 to 25×25 and compute the average RMSE. From the third row, we observe that the performance improves until size of 15×15. Increasing size further does not give additional performance gain. This indicates that aggregating pixels from a 15×15 window is enough for the task. For offset learning, we restrict the maximum range of the sampling position to 15×15 for all experiments. That is, the filtering results from the third to last rows are computed by aggregating 9, 25 or 49 samples sparsely chosen from a 15×15 window. The last row of Table 6 suggests that our final models also benefit from using more samples. The RMSE for DKN decreases from 3.26 to 3.19 at the cost of additional runtime. For comparison, DKN with kernels of size 3×3, 5×5 and 7×7 take 0.22, 0.27 and 0.34 seconds, respectively, with a Nvidia GTX 1080Ti. A 3×3 size offers a good compromise in terms of RMSE and runtime and this is when we have used in all experiments.

Feature channels. In Table 7, we compare the effects of the number of feature channels in terms of RMSE, runtime, the number of network parameters, and model size. We use our DKN and FDKN models including the residual connection and a fixed size of 3×3 kernels. We vary the number of channels n_i $(i = 1, 2)$ in the final two layers for feature extraction (see Tables 1 and 2). The table shows that using more channels for feature extraction helps improve performance, but requires more runtime and a large number of parameters to be learned. For example, DKN takes twice more time for a (modest) 0.09 RMSE gain. Consequently, we choose the number of feature channels $n_1 = 128$ and $n_2 = 128$ for both models.

Size of training data. Current CNN-based approaches directly regress the filtering output, and are thus likely to overfit to the particular characteristics of training data. Contrary to this, a weighted average in our models smooths the output and acts as a regularizer, indicating that they are not seriously affected by the number of training samples. To demonstrate this, we compare the average RMSE performance in Table 8 by varying the size of the training data. We train the DKN for depth map upsampling ($8\times$) while gradually increasing the number of training samples from 10 to $1,000$ in the NYU v2 dataset. We test it in the same configuration as in Table 3. Table 8 shows that our model is robust to the size of training data and generalizes well for other images outside the training dataset. In particular, the DKN trained with only 10 images outperforms the state of the art by a significant margin for all test datasets (see Table 3).

6 Discussion

Kernel visualization. We show in Fig. 13 some examples of 3×3 filter kernels estimated by the DKN with/without the residual

TABLE 9
Runtime comparison for images of size 640×480 (NYU v2 dataset).

	MRF [45]	GF [7]	JBU [4]	TGV [3]	Park [2]	SDF [6]	FBS [15]	DMSG [21]	DJFR [17]	DKN	FDKN
Times (s)	0.69	0.14	0.31	33	18	25	0.37	0.05	**0.02**	0.22	0.06

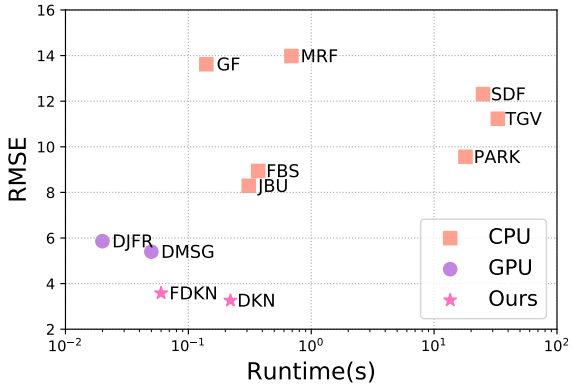


Fig. 14. Runtime and RMSE comparison to the state of the art. Our models show a good trade-off between rumtime and RMSE. They outperform other methods by significant margins in terms of RMSE and shows runtime comparable to DMSG.

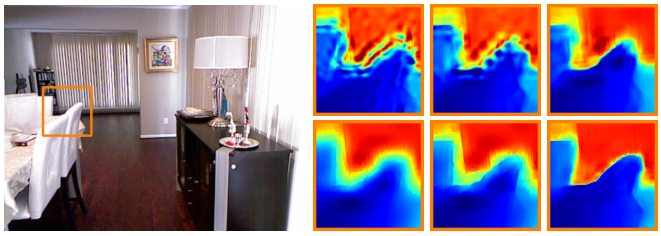
TABLE 10
RMSE comparison (DKN/DJFR [17]) of using different upscaling factors for training and testing on depth map upsampling.

Train/Test	4×	8×	16×
4×	**1.62**/ 3.34	6.70/10.21	11.24/19.75
8×	3.93/ 9.27	**3.26**/ 5.86	10.53/15.65
16×	9.04/19.12	8.61/13.43	**6.51**/10.11

connection. Although the sampling positions are fractional, we plot them on a discrete regular grid using bilinear interpolation for the purpose of visualization. Corresponding kernel weights are also interpolated. We observe three things: (1) The learned kernels are spatially adaptive and edge-aware. For example, the kernels learned without the residual connection aggregate depth values that are similar to that at the center position. (2) They can handle the case when the structures from the guidance and target images are not consistent as shown in the second example. (3) The kernels learned with the residual connection are orientation-selective and look like high-pass filters. For example, the kernels from the first and second examples can extract diagonal and vertical edges, respectively.

Runtime. Table 9 shows runtime comparisons on the same machine. We report the runtime for DMSG [21], DJFR [17] and our models with a Nvidia GTX 1080Ti and for other methods with an Intel i5 3.3 GHz CPU. Our current implementation for DKN takes on average 0.22 seconds for images of size 640×480. For comparison, the brute-force implementation takes 720 seconds, requiring about $3000\times$ more time. DKN is slower than DMSG [21] and DJFR [17], but yields a significantly better RMSE (Fig. 14 and Table 3). The faster overall speed comes from the use of spatially-invariant kernels in standard CNNs. FDKN runs about $4\times$ faster than the DKN and $3\times$ slower than DJFR, but with significantly higher accuracy.

DownConv for DKN. The DKN without "DownConv" layers can be implemented in a single forward pass, but requires more parameters (≈ 0.47M) to maintain the same receptive field size, with no runtime gain in experiments and a total number of convolutions increasing from 0.6M to 1M at each pixel. We have also tried dilated convolutions [59] that support large receptive fields without loss of resolution. When using the same receptive field size and learnable parameter numbers (≈ 0.5M), even though the runtime decreases from 0.22 to 0.16 seconds, the average RMSE for dilated convolutions increases from 3.26 to 4.30 for depth upsampling ($\times 8$) on the NYU v2 dataset, and the storage requirements are multiplied by 16. FDKN thus appears to be the preferable alternative.



(a) RGB image. (b) $\times 4$. (c) $\times 8$. (d) $\times 16$.

Fig. 15. Visual comparison of upsampled depth images for DJFR [17] (top) and DKN (bottom) when the scale factors for training ($\times 16$) and test ($\times 4, \times 8, \times 16$) are different.

Upscaling factors for training and testing. Table 10 compares the average RMSE between DKN and DJFR [17] on the NYU dataset [44], when the scale factors for training and test are different. It shows that the performance is degraded for both methods in this case. This may be handled by a scale augmentation technique during training [34]. Another observation is that our model gives a similar result when the scale factor for testing is smaller than that used for training. For scale factors of $\times 4$, $\times 8$ and $\times 16$, our model trained with factor of $\times 16$ gives the average RMSE of 9.04, 8.61 and 6.51, respectively, whereas DJFR gives 19.12, 13.43 and 10.11, respectively. This demonstrates that our model generalizes better over different scale factors. A visual comparison is shown in Fig. 15.

7 Conclusion

We have presented a CNN architecture for joint image filtering that is generic and applicable to a great variety of applications. Instead of regressing the filtering results directly from the network, we use spatially-variant weighted averages where the set of neighbors and the corresponding kernel weights are learned end-to-end in a dense and local manner. We have also presented an efficient implementation that gives much faster runtime than the brute-force one. A fast version further achieves an additional $4\times$ speed-up without much (if any) loss in performance. Our models generalize well to images that have different modalities from the training dataset, as demonstrated by our experiments. Finally, we have shown that the weighted averaging process with sparsely sampled 3×3 kernels is sufficient to set new state-of-the-art results on several tasks.

Acknowledgments

The authors would like to thank Yijun Li for helpful discussion. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2017R1C1B2005584), the Louis Vuitton/ENS chair on artificial intelligence and the NYU/Inria collaboration agreement.

REFERENCES

[1] Q. Yang, R. Yang, J. Davis, and D. Nistér, "Spatial-depth super resolution for range images," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007.

[2] J. Park, H. Kim, Y.-W. Tai, M. S. Brown, and I. Kweon, "High quality depth map upsampling for 3D-ToF cameras," in *in Proc. Int. Conf. Comput. Vis.*, 2011.

[3] D. Ferstl, C. Reinbacher, R. Ranftl, M. Rüther, and H. Bischof, "Image guided depth upsampling using anisotropic total generalized variation," in *in Proc. Int. Conf. Comput. Vis.*, 2013.

[4] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Trans. Graph.*, vol. 26, no. 3, p. 96, 2007.

[5] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep joint image filtering," in *in Proc. Eur. Conf. Comput. Vis.*, 2016.

[6] B. Ham, M. Cho, and J. Ponce, "Robust guided image filtering using nonconvex potentials," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 1, pp. 192–207, 2018.

[7] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 6, pp. 1397–1409, 2013.

[8] X. Shen, C. Zhou, L. Xu, and J. Jia, "Mutual-structure for joint filtering," in *in Proc. Int. Conf. Comput. Vis.*, 2015.

[9] Q. Yan, X. Shen, L. Xu, S. Zhuo, X. Zhang, L. Shen, and J. Jia, "Cross-field joint image restoration via scale map," in *in Proc. Int. Conf. Comput. Vis.*, 2013.

[10] L. Xu, Q. Yan, Y. Xia, and J. Jia, "Structure extraction from texture via relative total variation," *ACM Trans. Graph.*, vol. 31, no. 6, p. 139, 2012.

[11] Q. Zhang, X. Shen, L. Xu, and J. Jia, "Rolling guidance filter," in *in Proc. Eur. Conf. Comput. Vis.*, 2014.

[12] L. Karacan, E. Erdem, and A. Erdem, "Structure-preserving image smoothing via region covariances," *ACM Trans. Graph.*, vol. 32, no. 6, p. 176, 2013.

[13] B. Ham, M. Cho, C. Schmid, and J. Ponce, "Proposal flow," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016.

[14] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 2, pp. 504–511, 2013.

[15] J. T. Barron and B. Poole, "The fast bilateral solver," in *in Proc. Eur. Conf. Comput. Vis.*, 2016.

[16] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *in Proc. Int. Conf. Comput. Vis.*, 1998.

[17] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Joint image filtering with deep convolutional networks," *arXiv preprint arXiv:1710.04200*, 2017.

[18] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *in Proc. Eur. Conf. Comput. Vis.*, 2012.

[19] A. Levin, D. Lischinski, and Y. Weiss, "A closed-form solution to natural image matting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 2, pp. 228–242, 2008.

[20] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski, "Edge-preserving decompositions for multi-scale tone and detail manipulation," vol. 27, no. 3, p. 67, 2008.

[21] T.-W. Hui, C. C. Loy, and X. Tang, "Depth map super-resolution by deep multi-scale guidance," in *in Proc. Eur. Conf. Comput. Vis.*, 2016.

[22] L. Xu, J. Ren, Q. Yan, R. Liao, and J. Jia, "Deep edge-aware filters," in *in Proc. Int. Conf. Machine Learning*, 2015.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *in Adv. Neural Inf. Process. Syst.*, 2012.

[24] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via L0 gradient minimization," vol. 30, no. 6, p. 174, 2011.

[25] R. Szeliski, "Locally adapted hierarchical basis preconditioning," vol. 25, no. 3, pp. 1135–1143, 2006.

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

[27] D. Ferstl, M. Ruther, and H. Bischof, "Variational depth superresolution using example-based edge representations," in *in Proc. Int. Conf. Comput. Vis.*, 2015.

[28] M. Jaderberg, K. Simonyan, A. Zisserman *et al.*, "Spatial transformer networks," in *in Adv. Neural Inf. Process. Syst.*, 2015.

[29] C. B. Choy, J. Gwak, S. Savarese, and M. Chandraker, "Universal correspondence network," in *in Adv. Neural Inf. Process. Syst.*, 2016.

[30] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool, "Dynamic filter networks," in *in Adv. Neural Inf. Process. Syst.*, 2016.

[31] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive convolution," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017.

[32] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *in Proc. Int. Conf. Comput. Vis.*, 2017.

[33] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *in Adv. Neural Inf. Process. Syst.*, 2014.

[34] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *in Proc. Int. Conf. Machine Learning*, 2015.

[37] J. Wang and M. F. Cohen, "Optimized color sampling for robust matting," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007.

[38] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2005.

[39] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015.

[40] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016.

[41] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *in Proc. Int. Conf. Learning Representations*, 2016.

[42] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007.

[43] S. Lu, X. Ren, and F. Liu, "Depth enhancement via low-rank matrix completion," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014.

[44] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *in Proc. Eur. Conf. Comput. Vis.*, 2012.

[45] J. Diebel and S. Thrun, "An application of Markov random fields to range sensing," in *in Adv. Neural Inf. Process. Syst.*, 2006.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *in Proc. Int. Conf. Learning Representations*, 2015.

[47] 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," 2017.

[48] H. Wu, S. Zheng, J. Zhang, and K. Huang, "Fast end-to-end trainable guided filter," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018.

[49] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2018.

[50] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *in Adv. Neural Inf. Process. Syst.*, 2011.

[51] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, 2015.

[52] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *in Proc. Int. Conf. Comput. Vis.*, 2011.

[53] D. Scharstein and C. Pal, "Learning conditional random fields for stereo," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007.

[54] https://www.asus.com/ae-en/3D-Sensor/Xtion_PRO_LIVE/.

[55] Z. Zhang, "Microsoft Kinect sensor and its effect," *IEEE Trans. Multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[56] R. Margolin, L. Zelnik-Manor, and A. Tal, "How to evaluate foreground maps?" in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014.

[57] C. Yang, L. Zhang, H. Lu, X. Ruan, and M.-H. Yang, "Saliency detection via graph-based manifold ranking," in *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013.

[58] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr, "Conditional random fields as recurrent neural networks," in *in Proc. Int. Conf. Comput. Vis.*, 2015.

[59] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *in Proc. Int. Conf. Learning Representations*, 2016.

Beomjun Kim is a PhD student of Electrical and Electronic Engineering at Yonsei University in Seoul, Korea. He received the B.S. degrees in Computer Science from Yonsei University in 2016. He has received the

Global PhD Fellowship from the National Research Foundation of Korea (NRF) since 2018. His research interest include computer vision, deep learning, and joint filtering.

Jean Ponce is a research director at Inria and a visiting researcher at the NYU Center for Data Science, on leave from École Normale Supérieure (ENS) / PSL Research University, where he is a professor, and served as director of the computer science department from 2011 to 2017. Before joining ENS and Inria, Jean Ponce held positions at MIT, Stanford, and the University of Illinois at Urbana-Champaign, where he was a full professor until 2005. Jean Ponce is an IEEE Fellow and a Sr. member of the Institut Universitaire de France. He served as editor-in-chief for the International Journal of Computer Vision from 2003 to 2008, and chaired the IEEE Conference on Computer Vision and Pattern Recognition in 1997 and 2000, and the European Conference on Computer Vision in 2008. Jean Ponce is the recipient of two US patents, an ERC advanced grant, and the 2016 IEEE CVPR Longuet-Higgins prize. He is the author of "Computer Vision: A Modern Approach", a textbook translated in Chinese, Japanese, and Russian.

Bumsub Ham is an an Assistant Professor of Electrical and Electronic Engineering at Yonsei University in Seoul, Korea. He received the B.S. and Ph.D. degrees in Electrical and Electronic Engineering from Yonsei University in 2008 and 2013, respectively. From 2014 to 2016, he was Post-Doctoral Research Fellow with Willow Team of INRIA Rocquencourt, École Normale Supérieure de Paris, and Centre National de la Recherche Scientifique. His research interests include computer vision, computational photography, and machine learning, in particular, regularization and matching, both in theory and applications.