Deformable Kernel Networks for Joint Image Filtering

Beomjun Kim, Jean Ponce, Bumsub Ham

Deformable Kernel Networks for Joint Image Filtering

Beomjun Kim[1] Jean Ponce[2] Bumsub Ham[1,⋆]

[1]Yonsei University [2]DI-ENS / WILLOW

Abstract. Joint image filters transfer structural details from a guidance picture used as a prior to a target image, particularly for 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 learn instead sparse and spatially-variant kernels explicitly. We propose a CNN architecture, called a deformable kernel network (DKN), that outputs sets of neighbors and their corresponding weights adaptively for each pixel. The filtering result is then computed as a weighted average. We also propose an efficient implementation that runs about 3000× faster than a brute-force one for an image of size 640 × 480. We demonstrate the effectiveness and flexibility of our model on the tasks of depth map upsampling, saliency map upsampling, cross-modality image restoration, and texture removal. 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. Our code and models are available online: https://github.com/jun0kim/DeformableKernelNetwork

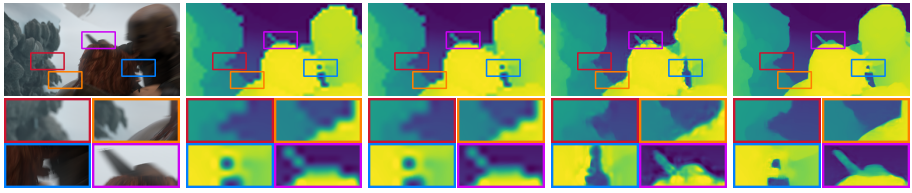
Keywords: Joint filtering, deep convolutional neural networks, depth map upsampling

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]:

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

⋆ corresponding author
work in progress



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

Fig. 1. Comparison of the state of the art and our model on depth map upsampling (16×). 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 (b) GF [7], (c) DJFR [17] and (d) 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. (Best viewed in color.)

Here, 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 in constant time. These filters use locally and 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 the kernel W from both images f and g 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 solving a linear system. This is equivalent to implicitly 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, making it possible to reflect 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). CNN-based methods combine instead nonlinear activations of spatially-invariant kernels learned from the networks. That is, they approximates 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, 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 an efficient implementation that is about 3000× faster at test time than a brute-force one for images of size 640×480. We show that the weighted averaging process using sparsely sampled 3×3 kernels is sufficient to reach state-of-the-art results in a variety of applications, including depth map upsampling, saliency map upsampling, cross-modality image restoration and texture removal (Fig. 1). Our code and models are available online:https://github.com/jun0kim/DeformableKernelNetwork.

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

- We introduce a generic and efficient model for joint image filtering, the DKN, that computes the set of neighbors and their corresponding weights adaptively for individual pixels.
- We propose its efficient implementation using a shift-and-stitch approach [24,25].
- 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.

2 Related work

Here we briefly describe representative approaches related to our work.

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 filter [16] and guided filter [7] 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 difficult to manually design 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 [2,3,6,10,20,26]. 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 [27] or multigrid methods [20]. Finally, learning-based methods can further be categorized into dictionary-based and CNN-based

approaches. Dictionary-based methods exploit the relationship between paired low-resolution and high-resolution target patches, additionally coupled with the guidance image [28,29]. 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 methods 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.

Variants of the spatial transformer [30]. Recent works introduce more flexible and effective CNN architectures. Jaderberg *et al.* propose a novel learnable module, the spatial transformer [30], that outputs the parameters of the desired spatial transformation (e.g., affine and 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 [31]. They learn local transformation parameters for normalizing orientation and scale in feature matching. Most similar to ours are the dynamic filter network [32] for video prediction and the adaptive convolution network [25] 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 [33]. The basic idea of deformable convolutions is to add offsets to the sampling locations defined on a regular grid in the standard convolution. The deformable convolutional network samples features directly from learned offsets, but shares the same weights for different sets of offsets as in standard CNNs. Unlike this, 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

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 [34], 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 to

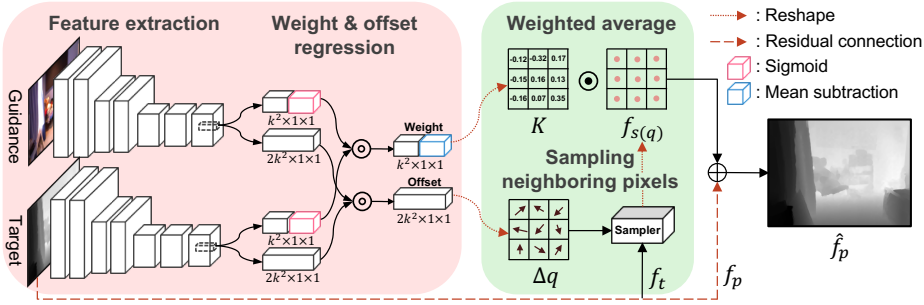


Fig. 2. The DKN architecture. Our model learns the kernel weights K and the spatial sampling offsets $\Delta\mathbf{q}$ from the feature maps of guidance and target images. To obtain the residual image $\hat{f}_\mathbf{p} - f_\mathbf{p}$, it then computes the weighted average with the kernel weights 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 $\odot$ and $\cdot$ by 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.)

obtain a residual image. The sampling locations are computed from the offsets. 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.

The main reasons behind using a residual connection are that the filtering result is largely correlated with the target image, and both share low-frequency content [17,35,36]. Focussing on learning the residuals also accelerates training speed while achieving better performance. Note that contrary to [17,35,36], 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 the DKN without the residual connection. In this case, we compute the filtering result directly as the weighted average in (1). In the following sections, we describe each component of our model and its efficient implementation.

3.2 Network architecture

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

Feature extraction. We input the guidance and target images to each of sub-networks that consist of 7 convolutional layers. The sub-network gives a feature map of size $128\times1\times1$ for the receptive field of size 51×51. We use the ReLU [23] as an activation function and batch normalization [37] for regularization.

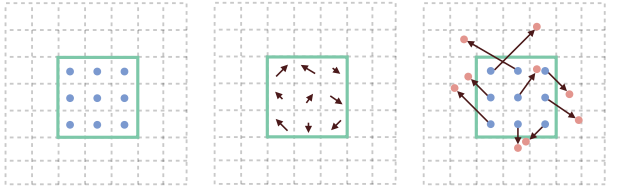
Table 1. Network architecture details. "BN" and "Res." denote the batch normalization [37] and residual connection, respectively. We denote by "DownConv" convolution with stride 2. The inputs of our network are 3-channel guidance and 1-channel target images (denoted by C). For the model without the residual connection, we use an L1 normalization layer (denoted by "L1 norm.") instead of subtracting mean values for weight regression.

Feature extraction		Weight regression	
Type	Output	Type	Output
Input	$C \times 51 \times 51$	Conv(1×1)	$k^2 \times 1 \times 1$
Conv(7×7)-BN-ReLU	$32 \times 45 \times 45$	Sigmoid	$k^2 \times 1 \times 1$
DownConv(2×2)	$32 \times 22 \times 22$	Mean subtraction or	$k^2 \times 1 \times 1$
Conv(5×5)-BN-ReLU	$64 \times 18 \times 18$	L1 norm. (w/o Res.)	
DownConv(2×2)-ReLU	$64 \times 9 \times 9$	Offset regression	
Conv(5×5)-BN-ReLU	$128 \times 5 \times 5$	Type	Output
Conv(3×3)-ReLU	$128 \times 3 \times 3$	Conv(1×1)	$2k^2 \times 1 \times 1$
Conv(3×3)-ReLU	$128 \times 1 \times 1$		

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 [25], 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 for image filtering. The estimated kernels should be similar to high-pass filters, with the 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 residual connection, we apply instead L1 normalization to the output of size $k^2 \times 1 \times 1$[1], forcing 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. That is, the filtering result is computed by aggregating 9 samples sparsely chosen from a large 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 has been observed in [38], 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 [39] in that both select which pixels to aggregate instead of the immediate ones.

[1] All elements in the combined output are larger than 0.



(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.

Filtering. 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 followed by adding the target image:

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

Here, we define $\mathcal{N}(\mathbf{p})$ as 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 [30] 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 [30,33] 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.

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 L_1 norm of the difference between the network output $\hat{f}$ and ground truth f^{gt} as follows.

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

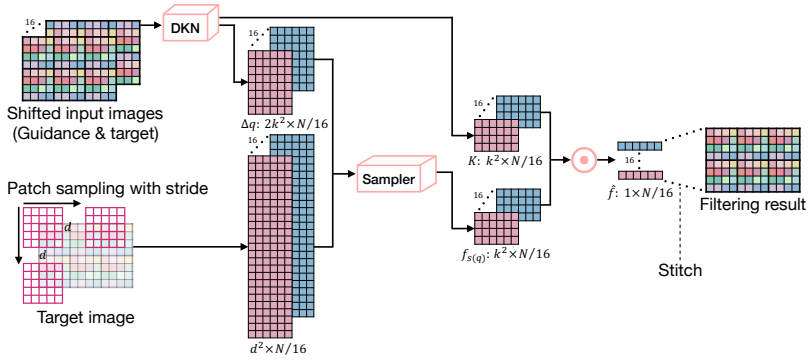


Fig. 4. Efficient implementation using a shift-and-stitch approach. We shift the input images and compute the results through the network. We then stitch them up to get a filtering result that has the same resolution as the inputs. Our approach makes it possible to reuse the storage for kernel weights,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

3.3 Efficient implementation

We compute the filtering results for inputs of any size in a single forward pass, since our network is fully convolutional [24]. 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 pixels.

We use instead the shift-and-stitch approach [24,25] 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 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. A 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 size of patches 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 by the sampling position $\mathbf{s}(\mathbf{q})$ from the 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. 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 resampled pixels.

4 Experiments

We apply our model to the tasks of joint image upsampling (depth map upsampling and saliency map upsampling) and noise removal (cross-modality image restoration and texture removal), and compare it to the state of the art in each case. For noise removal, we can only show qualitative comparisons as quantitative ground truth is not available. The results for the comparison have been obtained from the source code provided by the authors. Our code and models will be made publicly available at the time of publication. More results can be found in the supplementary material.

4.1 Implementation details

Following the experimental protocol in [5,17], we train our models using a large number of RGB/D image pairs, and test them to 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.

Training. We sample 1,000 RGB/D image pairs of size 640×480 from the NYU v2 dataset [40]. 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 total 2,000 RGB/D image pairs as training samples. We train different models for joint image upsampling and noise removal. 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 noise removal are similarly trained but with 4k iterations. Noisy depth images are synthesized by adding the Gaussian noise with zero mean and variance of 0.005. We use the Adam optimizer [41] 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 as the training dataset is sufficiently large to train our models. All networks are trained end-to-end using `PyTorch` [42].

Testing. We can obtain filtering results for inputs of any size by stitching coarse outputs, which is far more efficient than a brute-force implementation. 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 a multi-channel target image (e.g., texture removal), we apply our model separately in each channel and combine the outputs.

4.2 Joint image upsampling

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 [40], 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.

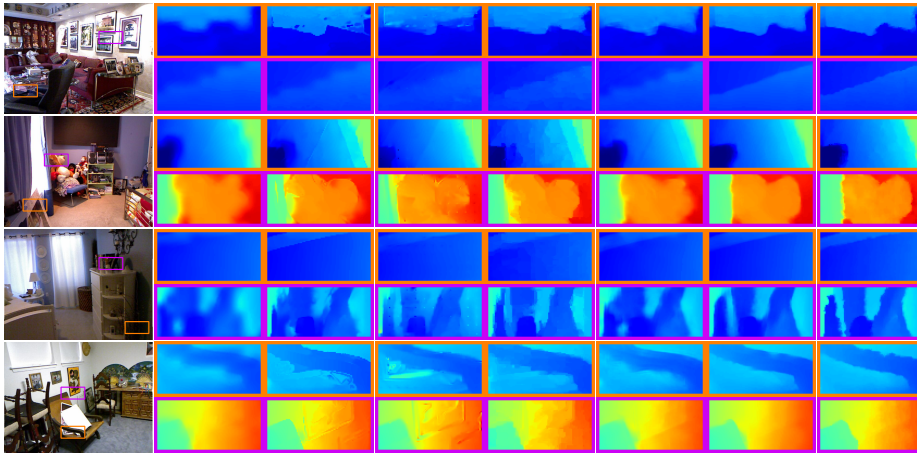
Table 2. Quantitive comparison with the state of the art on depth map upsampling in terms of average RMSE. All numbers except for the Sintel dataset are taken from [5,17]. Numbers in bold indicate the best performance and underscored ones are the second best.

Datasets	Middlebury [43]			Lu [44]			NYU v2 [40]			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
Ours 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	**7.47**
Ours	**1.23**	**2.12**	**4.24**	**0.96**	**2.16**	**5.11**	**1.62**	**3.26**	**6.51**	**3.30**	**4.77**	7.58

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 [43,46]: We use the 30 RGB/D image pairs from 2001-2006 datasets provided by Lu [44].
- Lu dataset [44]: This provides 6 RGB/D image pairs acquired by the ASUS Xtion Pro camera [47].
- NYU v2 dataset [40]: It consists of 1,449 RGB/D image pairs captured with the Microsoft Kinect [48]. 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.

We compare our method with the state of the art in Table 2. It shows the average root mean squared errors (RMSE) between upsampling results and ground truth. All numbers except those for the Sintel dataset are taken from [5,17]. For the Sintel dataset, the results of DJF [5] and its residual version (DJFR [17]) are obtained by the provided models trained with the NYU v2 dataset. 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 (Ours w/o Res.), and (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., in the Middlebury dataset) outside the training dataset. The direct comparison of DMSG [21] with DJF, DJFR, and



RGB image. GF [7]. JBU [4]. TGV [3]. Park [2]. DJFR [17]. Ours. GT.

Fig. 5. Visual comparison of upsampled depth images (8×) on the NYU v2 dataset.

Table 3. Quantitative comparison on saliency map upsampling in terms of weighted F-score. We use 5,168 images from the DUT-OMRON dataset [49].

	HR	Bicubic Int.	GF [7]	SDF [6]	DJFR [17]	Ours
Weighted F-score	0.386	0.371	0.370	0.371	<u>0.378</u>	**0.380**

ours may not be fair, since DMSG uses the Middlebury and Sintel datasets for training the network. Nevertheless, our models perform better for scale factors ×8 and ×16 in the Middlebury dataset. DMSG [21] does not generalize for the real dataset.

Figure 5 shows a visual comparison of the upsampled depth images (8×) on the NYU v2 dataset. The better ability to extract common structures from the target and guidance images by our model here is clearly visible. Specifically, our results show a sharp depth transition without the texture-copying artifacts. In contrast, the artifacts are clearly visible even from the results of DJFR. DJFR tends to over-smooth the results and does not recover fine details. This verifies once more the advantage of using spatially-variant kernels and an adaptive neighborhood system in joint image filtering.

Saliency map upsampling. We set the saliency map computed by the manifold method [49] on the downsampled (8×) image in the DUT-OMRON dataset [49] as target. The low-resolution saliency map is then upsampled under the guidance of the original image. We show in Table 3 a comparison of weighted F-scores [50] between upsampled saliency maps and the ground truth. Figure 6 shows examples of the upsampling results by the state of the art and our model. The results show that our model outperforms others including a CNN-based one [17].

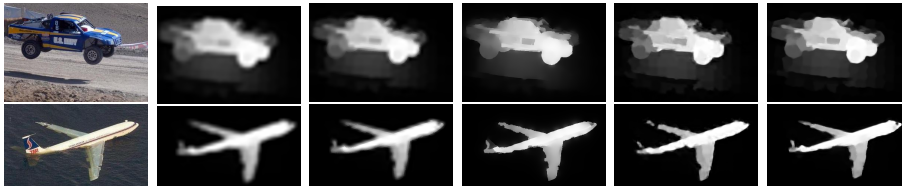


Fig. 6. Visual comparison of saliency map upsampling (8×) on the DUT-OMRON dataset [49].

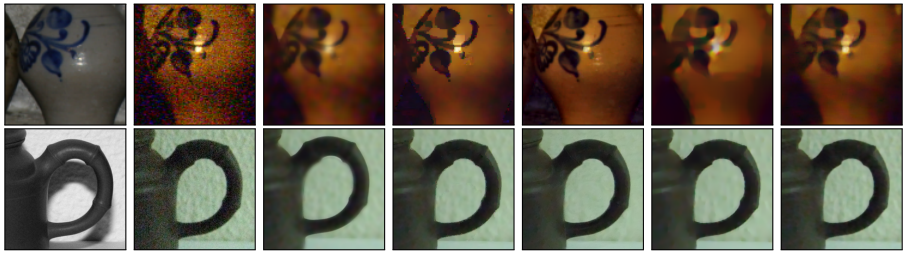


Fig. 7. Examples of cross-modality noise reduction for (top) flash/non-flash denoising and (bottom) RGB/NIR denoising.

4.3 Noise removal

We train a model for denoising depth images with RGB/D image pairs from the NYU v2 dataset [40]. We do not use the residual connection, since we empirically find that it does not help in this case. The model is applied to the tasks of cross-modality image restoration and texture removal without fine-tuning.

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 on flash/non-flash and RGB/NIR restoration are shown in Fig. 7 qualitatively. Our model outperforms 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, it can preserve edges while smoothing noise without artifacts. This demonstrates that our model trained with RGB/D images can generalize well for other ones that have different modalities.

Texture removal. We set the textured image itself as guidance and target ones, and apply our model repeatedly to remove small-scale textures. We show examples of texture removal in Fig. 8. Compared to the state of the art, our model removes textures without artifacts while maintaining other high-frequency structures such as image boundaries and corners.

4.4 Ablation study

In this section, we conduct an ablation analysis on different components and parameters in our model. We report the results for depth map upsampling (8×)

Fig. 8. Visual comparison of texture removal for regular (top) and irregular (bottom) textures.

Table 4. RMSE comparison of different networks.

Weight learning		Offset learning		Res.	3×3	5×5	7×7	15×15	19×19	25×25
RGB	Depth	RGB	Depth							
✓					5.92	5.52	5.43	5.59	5.82	6.21
	✓				5.24	4.36	4.09	4.09	4.11	4.15
✓	✓				5.03	3.90	3.48	3.32	3.33	3.39
✓		✓			5.37	5.38	5.40	–	–	–
✓	✓	✓	✓		3.36	3.32	3.33	–	–	–
✓	✓	✓	✓	✓	3.26	3.21	**3.19**	–	–	–

on the NYU v2 dataset. An ablation study on the number of feature channels can be found in the supplementary material.

Network architecture. We show the average RMSE for five variants of our model in Table 4. The baseline model learns kernel weights from the guidance image only. The first row shows that our baseline already outperforms the state of the art (Table 2). From the second row, we can see that our model trained without using the guidance image gives comparable results to the state of the art. We can also see that, for a fixed size of the kernel, 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.

Kernel size. Table 4 compares the performance for different size of kernels. We enlarge the size of the kernels 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. We restrict the maximum range of the sampling position to 15×15 for all experiments. The last row of Table 4 suggests that our final model also benefits from increasing the kernel size. The RMSE decreases from 3.26 to 3.19 at the cost of taking more runtime. For comparison, our models with kernels of size

Table 5. RMSE comparison (ours/DJFR [17]) of using different scale 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

3×3, 5×5 and 7×7 take 0.22, 0.27 and 0.34 seconds, respectively, with a Nvidia GTX 1080Ti. Accordingly, we use the kernel of size 3×3 to all experiments. This could be a good trade-off between the performance and efficiency.

4.5 Discussion

Runtime. With a Nvidia GTX 1080Ti, our current implementation takes on average 0.22 second for images of size 640×480. For comparison, the brute-force implementation takes 720 seconds, requiring about 3000× more time than the efficient version. The average runtime of DJFR [17] is about 0.02 seconds on the same configuration. The faster overall speed comes from the use of spatially-invariant kernels in standard CNNs.

Limitations. Table 5 compares the average RMSE between our model and DJFR [17] on the NYU dataset [40], when the scale factors for training and test are different. It shows that the performance is degraded for both methods if the factors for training/test are different. This may be handled by a scale augmentation technique during training [35]. 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 ×4, ×8 and ×16, our model trained with factor of ×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.

5 Conclusions

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. Our model generalizes well for 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.

Acknowledgement

The authors would like to thank Yijun Li for helpful discussions.

References

1. Yang, Q., Yang, R., Davis, J., Nistér, D.: Spatial-depth super resolution for range images. In: CVPR. (2007)
2. Park, J., Kim, H., Tai, Y.W., Brown, M.S., Kweon, I.: High quality depth map upsampling for 3D-ToF cameras. In: ICCV. (2011)
3. Ferstl, D., Reinbacher, C., Ranftl, R., Rüther, M., Bischof, H.: Image guided depth upsampling using anisotropic total generalized variation. In: ICCV. (2013)
4. Kopf, J., Cohen, M.F., Lischinski, D., Uyttendaele, M.: Joint bilateral upsampling. In: SIGGRAPH. (2007)
5. Li, Y., Huang, J.B., Ahuja, N., Yang, M.H.: Deep joint image filtering. In: ECCV. (2016)
6. Ham, B., Cho, M., Ponce, J.: Robust guided image filtering using nonconvex potentials. IEEE Trans. PAMI **40**(1) (2018) 192–207
7. He, K., Sun, J., Tang, X.: Guided image filtering. IEEE Trans. PAMI **35**(6) (2013) 1397–1409
8. Shen, X., Zhou, C., Xu, L., Jia, J.: Mutual-structure for joint filtering. In: ICCV. (2015)
9. Yan, Q., Shen, X., Xu, L., Zhuo, S., Zhang, X., Shen, L., Jia, J.: Cross-field joint image restoration via scale map. In: ICCV. (2013)
10. Xu, L., Yan, Q., Xia, Y., Jia, J.: Structure extraction from texture via relative total variation. In: SIGGRAPH Asia. (2012)
11. Zhang, Q., Shen, X., Xu, L., Jia, J.: Rolling guidance filter. In: ECCV. (2014)
12. Karacan, L., Erdem, E., Erdem, A.: Structure-preserving image smoothing via region covariances. In: SIGGRAPH Asia
13. Ham, B., Cho, M., Schmid, C., Ponce, J.: Proposal flow. In: CVPR. (2016)
14. Hosni, A., Rhemann, C., Bleyer, M., Rother, C., Gelautz, M.: Fast cost-volume filtering for visual correspondence and beyond. IEEE Trans. PAMI **35**(2) (2013) 504–511
15. Barron, J.T., Poole, B.: The fast bilateral solver. In: ECCV. (2016)
16. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: ICCV. (1998)
17. Li, Y., Huang, J.B., Ahuja, N., Yang, M.H.: Joint image filtering with deep convolutional networks. arXiv preprint arXiv:1710.04200 (2017)
18. Butler, D.J., Wulff, J., Stanley, G.B., Black, M.J.: A naturalistic open source movie for optical flow evaluation. In: ECCV. (2012)
19. Levin, A., Lischinski, D., Weiss, Y.: A closed-form solution to natural image matting. IEEE Trans. PAMI **30**(2) (2008) 228–242
20. Farbman, Z., Fattal, R., Lischinski, D., Szeliski, R.: Edge-preserving decompositions for multi-scale tone and detail manipulation. In: SIGGRAPH. (2008)
21. Hui, T.W., Loy, C.C., Tang, X.: Depth map super-resolution by deep multi-scale guidance. In: ECCV. (2016)
22. Xu, L., Ren, J., Yan, Q., Liao, R., Jia, J.: Deep edge-aware filters. In: ICML. (2015)
23. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: NIPS. (2012)
24. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR. (2015)
25. Niklaus, S., Mai, L., Liu, F.: Video frame interpolation via adaptive convolution. In: CVPR. (2017)

26. Xu, L., Lu, C., Xu, Y., Jia, J.: Image smoothing via L0 gradient minimization. In: SIGGRAPH Asia. (2011)
27. Szeliski, R.: Locally adapted hierarchical basis preconditioning. In: SIGGRAPH. (2006)
28. Yang, J., Wright, J., Huang, T.S., Ma, Y.: Image super-resolution via sparse representation. IEEE Trans. Image Process. **19**(11) (2010) 2861–2873
29. Ferstl, D., Ruther, M., Bischof, H.: Variational depth superresolution using example-based edge representations. In: ICCV. (2015)
30. Jaderberg, M., Simonyan, K., Zisserman, A., et al.: Spatial transformer networks. In: NIPS. (2015)
31. Choy, C.B., Gwak, J., Savarese, S., Chandraker, M.: Universal correspondence network. In: NIPS. (2016)
32. Jia, X., De Brabandere, B., Tuytelaars, T., Gool, L.V.: Dynamic filter networks. In: NIPS. (2016)
33. Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y.: Deformable convolutional networks. In: ICCV. (2017)
34. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: NIPS. (2014)
35. Kim, J., Kwon Lee, J., Mu Lee, K.: Accurate image super-resolution using very deep convolutional networks. In: CVPR. (2016)
36. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
37. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML. (2015)
38. Wang, J., Cohen, M.F.: Optimized color sampling for robust matting. In: CVPR. (2007)
39. Buades, A., Coll, B., Morel, J.M.: A non-local algorithm for image denoising. In: CVPR. (2005)
40. Silberman, N., Hoiem, D., Kohli, P., Fergus, R.: Indoor segmentation and support inference from rgbd images. In: ECCV. (2012)
41. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR. (2015)
42. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. (2017)
43. Hirschmuller, H., Scharstein, D.: Evaluation of cost functions for stereo matching. In: CVPR. (2007)
44. Lu, S., Ren, X., Liu, F.: Depth enhancement via low-rank matrix completion. In: CVPR. (2014)
45. Diebel, J., Thrun, S.: An application of Markov random fields to range sensing. In: NIPS. (2006)
46. Scharstein, D., Pal, C.: Learning conditional random fields for stereo. In: CVPR. (2007)
47. https://www.asus.com/ae-en/3D-Sensor/Xtion_PRO_LIVE/
48. Zhang, Z.: Microsoft Kinect sensor and its effect. IEEE Trans. Multimedia **19**(2) (2012) 4–10
49. Yang, C., Zhang, L., Lu, H., Ruan, X., Yang, M.H.: Saliency detection via graph-based manifold ranking. In: CVPR. (2013)
50. Margolin, R., Zelnik-Manor, L., Tal, A.: How to evaluate foreground maps? In: CVPR. (2014)