Deformable Kernel Networks for Joint Image Filtering

Beomjun Kim, Jean Ponce, Bumsub Ham

International Journal of Computer Vision
https://doi.org/10.1007/s11263-018-1074-6

Deformable Kernel Networks for Joint Image Filtering

Beomjun Kim[1] · **Jean Ponce**[2] · **Bumsub Ham**[1]

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 seventeen 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 in all cases.

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

Beomjun Kim
E-mail: beomjun.kim@yonsei.ac.kr

Jean Ponce
E-mail: jean.ponce@inria.fr

Bumsub Ham (Corresponding author)
E-mail: bumsub.ham@yonsei.ac.kr

[1] School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea.
[2] Inria and DI-ENS, Département d'Informatique de l'ENS, CNRS, PSL University, Paris, France.

1 Introduction

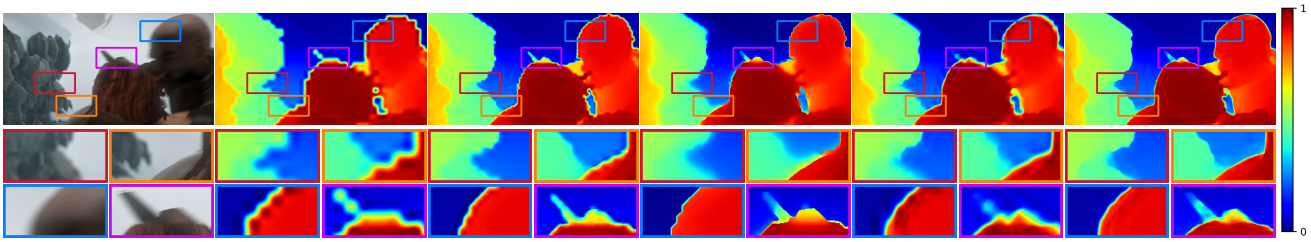
Image filtering with a guidance signal, a process called guided or joint filtering, has been used in a variety of computer vision and graphics tasks, including depth map upsampling (Yang et al. 2007; Park et al. 2011; Ferstl et al. 2013; Kopf et al. 2007; Li et al. 2016; Ham et al. 2018), cross-modality image restoration (He et al. 2013; Shen et al. 2015; Yan et al. 2013), texture removal (Ham et al. 2018; Xu et al. 2012; Zhang et al. 2014; Karacan et al. 2013), scale-space filtering (Ham et al. 2018), dense correspondence (Ham et al. 2016; Hosni et al. 2013) and semantic segmentation (Barron and Poole 2016). For example, high-resolution color images can be used as guidance to enhance the spatial resolution of depth maps (Kopf et al. 2007). 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 (He et al. 2013; Kopf et al. 2007; Tomasi and Manduchi 1998):

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

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 (Park et al. 2011; Ferstl et al. 2013; Kopf et al. 2007; He et al. 2013), the target image f itself (Zhang et al. 2014; Tomasi and Manduchi 1998), or both (Li et al. 2016; Ham et al. 2018), 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



(a) RGB image. (b) Depth image. (c) GF. (d) SDF. (e) DJFR. (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 (Butler et al. 2012), we upsample the depth image using (c) GF (He et al. 2013), (d) SDF (Ham et al. 2018), (e) DJFR (Li et al. 2019) and (f) our method. The filtering results for GF and our model are obtained by the weighted average in (1). We use filter kernels W of size 3×3 and 17×17 in our model and GF, respectively. We can see that our method using sparsely sampled 3×3 kernels outperforms GF and even the state of the art including the optimization-based SDF method (Ham et al. 2018) and CNN-based one (Li et al. 2019). Note that applying GF with 3×3 kernels does not recover fine details. (Best viewed in color.)

kernels and sets of neighbors without learning (He et al. 2013; Kopf et al. 2007; Tomasi and Manduchi 1998). For example, the bilateral filter (Tomasi and Manduchi 1998) uses spatially-variant Gaussian kernels to encode local structures from the guidance image. The guided filter (He et al. 2013) also leverages the local structure of the guidance image, but uses matting Laplacian kernels (Levin et al. 2008), 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 (Ham et al. 2018). This causes texture-copying artifacts, especially in the case of data from different sensors (Ferstl et al. 2013). To address this problem, the SD filter (Ham et al. 2018) constructs spatially-variant kernels from both guidance and target images to exploit common structures, and formulates joint image filtering as an optimization problem. The DG filter (Gu et al. 2017) uses a task-driven learning method to obtain the optimized guidance images tailored to depth upsampling. This type of approaches (e.g., (Ferstl et al. 2013; Xu et al. 2012; Farbman et al. 2008)) 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 (He et al. 2013), 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) (Li et al. 2016; Hui et al. 2016; Xu et al. 2015) 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 (Krizhevsky et al. 2012)).

In this paper, we revisit the guided weighted average framework in (1) for joint image filtering. We argue that leveraging spatially-invariant kernels in CNN-based methods (Li et al. 2016; Hui et al. 2016; Xu et al. 2015) is limited to encoding structural details from guidance and target images that typically change with image location. We exploit instead spatially-variant kernels explicitly, as in the classical approaches using the weighted average, but learn the kernel weights (W) and the set of neighbors ($\mathcal{N}$) in a completely data-driven way, 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 $17\times$ speed-up compared to the plain DKN for an image of size 640×480, while 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 6), saliency map upsampling (Fig. 10), cross-modality image restoration (Fig. 11), texture removal (Fig. 12), and semantic segmentation (Fig. 13).

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

- We introduce a novel variant of the classical guided weighted averaging process 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 seventeen times faster than 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 to investigate the influence of all the components and parameters of our model (Section 5).

To encourage comparison and future work, our code and models are available at our project webpage[1].

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 (Kopf et al. 2007; He et al. 2013; Zhang et al. 2014). The bilateral (Tomasi and Manduchi 1998) and guided (He et al. 2013) 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 (Li et al. 2016).

Second, implicit weighted-average methods formulate joint filtering as an optimization problem, and minimize an objective function that usually involves fidelity and regularization terms (Park et al. 2011; Ferstl et al. 2013; Ham et al. 2018; Xu et al. 2012; Farbman et al. 2008; Xu et al. 2011). 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 (Farbman et al. 2008), gives the output having 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 (Szeliski 2006) or multigrid methods (Farbman et al. 2008).

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 (Yang et al. 2010; Ferstl et al. 2015). In CNN-based methods (Hui et al. 2016; Li et al. 2016, 2019), 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. Other methods (Riegler et al. 2016a,b) integrate a variational optimization into CNNs by unrolling the optimization steps of the primal-dual algorithm, which requires two stages in training and a number of iterations in testing. Similar to implicit weighted-average methods, they use hand-crafted regularizers, which may not capture structural priors.

Our method borrows from both explicit weighted-average methods and CNN-based ones. Unlike existing explicit weighted-average methods (He et al. 2013; Kopf et al. 2007), that use hand-crafted kernels and neighbors defined on a fixed regular grid, we leverage CNNs to learn the set of sparsely chosen neighbors and their corresponding weights adaptively. Our method differs from previous CNN-based ones (Hui et al. 2016; Li et al. 2016, 2019) in that we learn sparse and spatially-variant kernels for each pixel to obtain upsampling results as a weighted average. The bucketing stretch in single image super-resolution (Getreuer et al. 2018; Romano et al. 2017) can be seen as a non-learning-based approach to filter selection. It assigns a single filter by solving a least-squares problem for a set of similar patches (buckets). In contrast, our model *learns* different filters using CNNs even for similar RGB patches, since we learn them from a set of multi-modal images (i.e., pairs of RGB/D images).

2.2 Variants of the spatial transformer

Recent works introduce more flexible and effective CNN architectures. Jaderberg et al. propose a novel learnable module, the spatial transformer (Jaderberg et al. 2015), 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 (Choy et al. 2016). They learn local transformation parameters for normalizing orientation and scale in feature matching.

Most similar to ours are the dynamic filter network (Jia et al. 2016) and its variants (the adaptive convolution network (Niklaus et al. 2017) and the kernel prediction networks (Bako et al. 2017; Mildenhall et al. 2018; Vogels et al. 2018)), where a set of local transformation parameters is generated adaptively conditioned on the input image. The main differences between our model and these works are three-fold. First, our network is not limited to learning spatially-variant kernels, but it also learns the sampling lo-

[1] https://cvlab.yonsei.ac.kr/projects/dkn

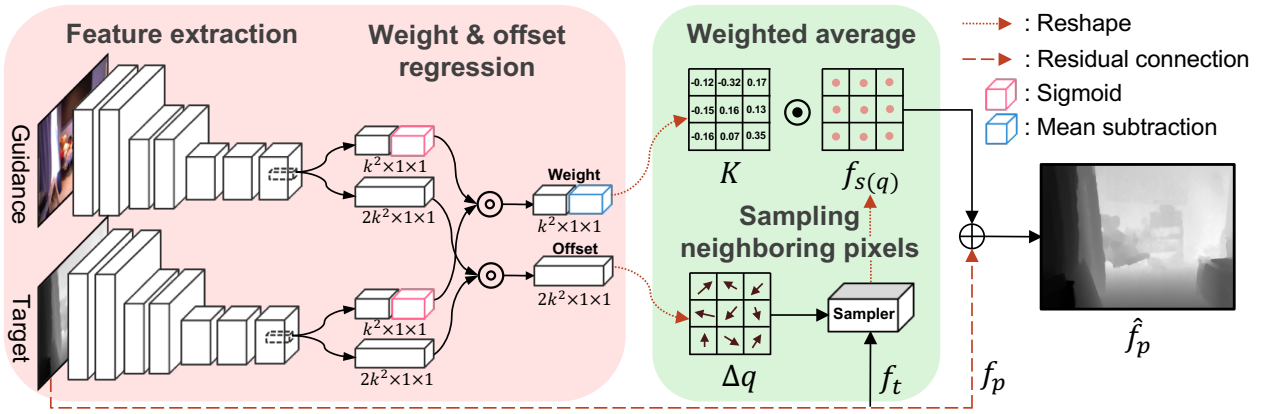


Fig. 2 The DKN architecture. We learn the kernel weights K and the spatial sampling offsets $\Delta\mathbf{q}$ from the feature maps of guidance and target images. To obtain the residual image $\hat{f}_\mathbf{p} - f_\mathbf{p}$, we then compute the weighted average with the kernel weights K and image 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.)

cations of neighbors. This allows us to aggregate sparse but highly related samples only, enabling an efficient implementation in terms of speed and memory and achieving state-of-the-art results even with kernels of size 3×3 on several tasks. For comparison, the adaptive convolution and kernel prediction networks require much larger neighbors (e.g., 21×21 in (Bako et al. 2017; Vogels et al. 2018), 41×41 in (Niklaus et al. 2017), and $8\times5\times5$ in (Mildenhall et al. 2018)). As will be seen in our experiments, learning sampling locations of neighbors clearly boosts the performance significantly compared to learning kernel weights only. Second, our model is generic and generalizes well to other tasks. It can handle multi-modal images (e.g., RGB/D images in depth map upsampling and an RGB image/a cost volume storing the costs for choosing labels in semantic segmentation), and thus is applicable to various tasks including depth and saliency map upsampling, cross-modality image restoration, texture removal, and semantic segmentation. In contrast, the adaptive convolution network is specialized to video frame interpolation, and kernel prediction networks are applicable to denoising Monte Carlo renderings (Bako et al. 2017; Vogels et al. 2018) or burst denoising (Mildenhall et al. 2018) only. Finally, our model learns spatially-variant kernels to compute residual images, not a final output as in (Bako et al. 2017; Jia et al. 2016; Mildenhall et al. 2018; Niklaus et al. 2017; Vogels et al. 2018), with constraints on weight regression. This allows the use of residual connections for adaptive convolution and kernel prediction networks, and achieves better results.

Our work is also related to the deformable convolutional network (Dai et al. 2017). The basic idea of deformable convolutions is to add offsets to the 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 and its efficient implementation using a shift-and-stitch approach (Section 3.2). We then describe a fast version of DKN (Section 3.3).

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. Motivated by the SD filter (Ham et al. 2018) and DJF (Li et al. 2016), we extract features from individual guidance and target images. Specifically, we use a two-stream CNN (Simonyan and Zisserman 2014), where each sub-network is in charge of one of the two images, with different feature maps used to estimate the corresponding kernel weights and 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. Dual supervisory information for the weights and offsets is typically not available. We

Table 1 Network architecture details. "BN" and "Res." denote the batch normalization (Ioffe and Szegedy 2015) 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. We apply a zero padding technique to input images to handle boundary pixels.

Feature extraction		Weight regression	
Type	Output	Type	Output
Input	$D \times 51 \times 51$	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)-ReLu	$32 \times 22 \times 22$	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	$128 \times 1 \times 1$		

learn instead these parameters by minimizing directly the discrepancy between the output of the network and a reference image (e.g., a ground-truth depth map for depth upsampling). In particular, constraints on weight and offset regression (sigmoid and mean subtraction layers in Fig. 2) specify how the kernel weights and offsets behave and guide the learning process. For weight regression, we apply a sigmoid layer that makes all elements larger than 0 and smaller than 1. We then subtract the mean value from the output of the sigmoid layer. This makes the sum of kernel weights to be 0, so that the regressed weights act similar to a high-pass filter. 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 (Li et al. 2019; Kim et al. 2016; He et al. 2016; Zhang et al. 2017). Focussing on learning the residuals also accelerates training speed while achieving better performance (Kim et al. 2016). Note that contrary to (Li et al. 2019; Kim et al. 2016; He et al. 2016; Zhang et al. 2017), we obtain the residuals by a weighted averaging process with the learned kernels, instead of regressing 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).

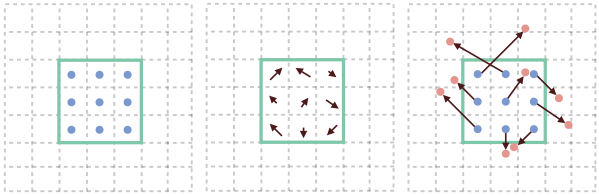
3.2 DKN architecture

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

Feature extraction. We adapt the architecture of (Niklaus et al. 2017) for feature extraction, with 7 convolutional layers. We input the guidance and target images to each of the sub-networks, which gives a feature map of size $128 \times 1 \times 1$ for a receptive field of size 51×51. We use the ReLU (Krizhevsky et al. 2012) as an activation function. Batch normalization (Ioffe and Szegedy 2015) is used for speeding up training and 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 (Bako et al. 2017; Niklaus et al. 2017; Vogels et al. 2018), 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 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



(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.

samples sparsely chosen from a much larger neighborhood. The two main reasons behind the use of small-size kernels are as follows: (1) This enables an efficient implementation in terms of speed and memory. (2) The reliability of samples is more important than the total number of samples aggregated. A similar finding is noted in (Wang and Cohen 2007), which shows that only high-confidence samples should be chosen when estimating foreground and background images in image matting. Offset regression is closely related to non-local means (Buades et al. 2005) in that both aggregate pixels not located at immediate neighbors. Note that learning offsets can be seen as estimating correspondences *within* input images. The sampling positions computed by the offsets point to pixels whose intensity values are similar to that of the center of the kernel. That is, our model with filter kernels of size $k \times k$ computes k^2 matching points for each pixel. The kernel weights tell us how much information is aggregated from these matching points, indicating that the weights correspond to matching confidence. Note also that we use a larger receptive field than the filter kernel. This better handles the aperture problem occurring in finding the matching points by considering contextual information (Niklaus et al. 2017).

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 (Jaderberg et al. 2015) 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 sampling kernel. Following (Dai et al. 2017; Jaderberg et al. 2015), we use a two-dimensional bilinear kernel, and split it into two one-dimensional ones 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 (Li et al. 2019).

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.

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

Testing. Two principles have guided the design of our learning architecture: (1) Points from a large receptive field in the original guidance and target images should be used to compute the weighted averages associated with the value of the upsampled depth map at each one of its pixels; and (2) inference should be fast. The second principle is rather self-evident. We believe that the first one is also rather intuitive, and it is justified empirically by the ablation study presented later. In fine, it is also the basis for our approach, since our network learns where, and how to sample a small number of points in a large receptive field.

A reasonable compromise between receptive field size and speed is to use one or several convolutional layers with a multi-pixel stride, which enlarges the image area pixels are drawn from without increasing the number of weights in the network. This is the approach we have followed in our base architecture, DKN, with two stride-2 "DownConv" layers. The price to pay is a loss in spatial resolution for the final feature map, with only $1/16$ of the total number N of pixels in the input images. One could of course give as input to our network the receptive fields associated with all N of the original guidance and target image pixels, at the cost of N forward passes during inference. DKN implements a much more efficient method where 16 shifted copies of the two images are used in turn as input to the network, and the corresponding network outputs are then stitched together in a single image, at the cost of only 16 forward passes. The details of this *shift-and-stitch* approach (Long et al. 2015; Niklaus et al. 2017) can be found in Appendix A.

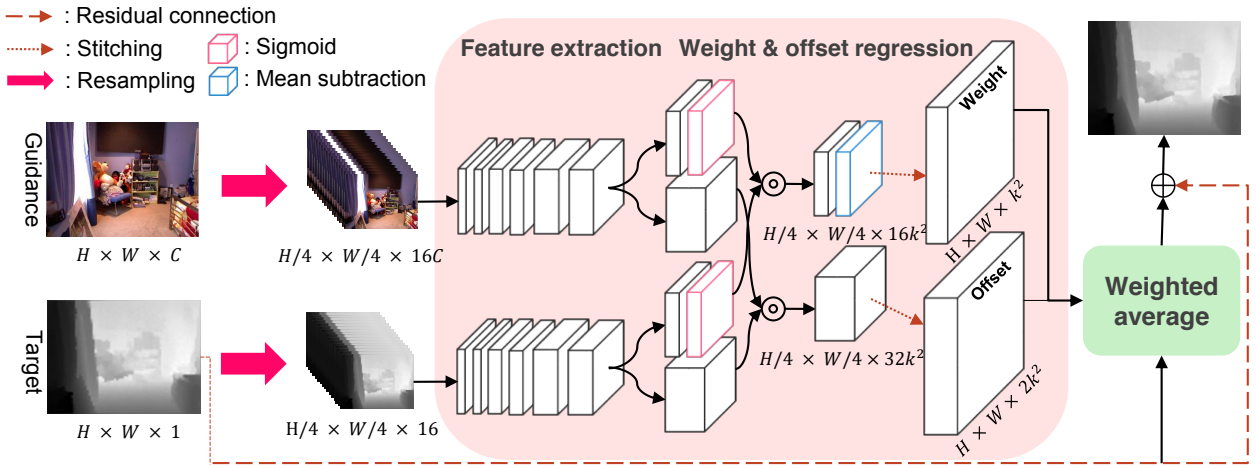


Fig. 4 The FDKN architecture. We resample a guidance image of size $H \times W \times C$ with stride 4 in each dimension (Fig. 5), where H, W and C are height, width and the number of channels, respectively. This gives resampled guidance images of size $H/4 \times W/4 \times 16C$. The target image is 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 16-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. We apply a zero padding technique to input images to handle boundary pixels.

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	
Conv(3×3)-BN-ReLU	$64 \times 7 \times 7$	L1 norm. (w/o Res.)	$16k^2 \times 1 \times 1$
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)	$32k^2 \times 1 \times 1$

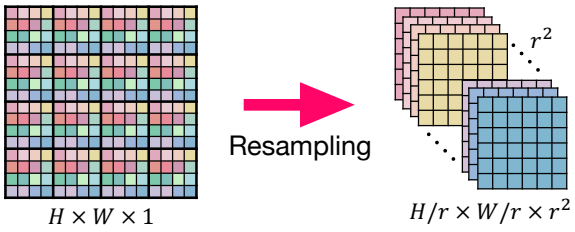


Fig. 5 Illustration of resampling. An image of size $H \times W \times 1$ is reshaped with stride r in each dimension, resulting a resampled one of size $H/r \times W/r \times r^2$.

3.3 FDKN architecture

We now present a fast version of DKN (FDKN) that achieves a $17\times$ speed-up over the DKN for an image of size 640×480 while retaining the same level of quality (Fig. 4). 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 5). A more efficient alternative to DKN is to split the input images into the same 16 subsampled and shifted parts as before, but this time *stack* them into new target and guidance images (Fig. 5), with 16 channels for the former, and $16C$ channels for the latter, e.g., $C = 3$ when the RGB image is used. The effective receptive field for FDKN is comparable to that of DKN[2], but FDKN involves much fewer parameters because of the reduced input image resolution and the shared weights across channels. The individual channels are then recomposed into the final upsampled image (Shi et al. 2016), at the cost of only one forward pass. Specifically, we use a series of 6 convolutional layers of size 3×3 for feature extraction. For weight and offset regression, we apply a 1×1 convolution on top of the feature extraction layers similar to DKN, but using more network parameters. For example, FDKN and DKN compute feature maps of size $16k^2 \times 1 \times 1$ and $k^2 \times 1 \times 1$,

[2] We could also use dilated convolutions (Yu and Koltun 2016) that support large receptive fields without loss of resolution, but empirically find that runtime gain is marginal (see Section 5).

Table 3 Quantitative 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. Following (Li et al. 2016, 2019), the average RMSE are measured in centimeter for the NYU v2 dataset (Silberman et al. 2012). For other datasets, we compute RMSE with upsampled depth maps scaled to the range $[0, 255]$. †: Our models trained with the depth map only without any guidance.

Datasets	Middlebury			Lu			NYU v2			Sintel		
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 (Diebel and Thrun 2006)	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 (He et al. 2013)	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 (Kopf et al. 2007)	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 (Ferstl et al. 2013)	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 (Park et al. 2011)	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 (Ham et al. 2018)	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 (Barron and Poole 2016)	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 (Hui et al. 2016)	1.88	3.45	6.28	2.30	4.17	7.22	3.02	5.38	9.17	5.32	7.24	10.11
DJF (Li et al. 2016)	1.68	3.24	5.62	1.65	3.96	6.75	2.80	5.33	9.46	4.58	6.57	9.38
DG (Gu et al. 2017)	1.97	4.16	5.27	2.06	4.19	6.90	3.68	5.78	10.08	4.11	6.61	9.24
DJFR (Li et al. 2019)	1.32	3.19	5.57	1.15	3.57	6.77	2.38	4.94	9.18	3.97	6.32	9.19
PAC (Su et al. 2019)	1.32	2.62	4.58	1.20	2.33	5.19	1.89	3.33	6.78	3.38	4.89	7.65
FDKN†	**1.07**	2.23	5.09	0.85	1.90	5.33	2.05	4.10	8.10	3.31	5.08	8.51
DKN†	1.12	2.13	5.00	0.90	**1.83**	**4.99**	2.11	4.00	8.24	3.40	4.90	8.18
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
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**
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	1.23	**2.12**	**4.24**	0.96	2.16	5.11	**1.62**	**3.26**	**6.51**	**3.30**	**4.77**	7.59
DMSG-NN (Hui et al. 2016)	2.11	3.74	6.03	2.48	4.74	7.51	3.37	6.20	10.05	5.49	7.48	10.52
DJF-NN (Li et al. 2016)	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-NN (Li et al. 2019)	1.98	3.61	6.07	2.22	4.54	7.48	3.38	5.86	10.11	5.50	7.43	10.48
PAC-NN (Su et al. 2019)	**1.91**	3.20	5.60	2.48	4.37	6.60	2.82	5.01	8.64	5.41	6.98	9.64
FDKN†-NN	2.89	3.92	6.75	2.85	4.64	7.62	3.64	5.43	8.96	5.74	7.31	10.31
DKN†-NN	2.94	4.14	8.12	2.88	5.13	8.24	3.67	6.68	12.15	5.78	7.59	10.95
FDKN w/o Res.-NN	2.11	3.63	6.29	2.50	4.52	7.37	2.65	5.02	8.69	5.48	7.27	10.22
DKN w/o Res.-NN	1.95	3.18	5.50	2.37	4.17	6.37	2.48	4.78	8.52	**5.29**	6.95	**9.53**
FDKN-NN	2.21	3.64	6.15	2.64	4.55	7.20	2.62	4.99	8.67	5.33	7.25	9.86
DKN-NN	1.93	**3.17**	**5.49**	**2.35**	**4.16**	**6.33**	**2.46**	**4.76**	**8.50**	**5.29**	**6.92**	9.56

respectively, for weight regression, from each feature of size $128 \times 1 \times 1$. This allows FDKN to estimate kernel weights and offsets for all pixels simultaneously. In practice, FDKN gives a 17 times speed-up over DKN. Because it involves fewer parameters (0.6M vs. 1.1M for DKN), one might expect somewhat degraded results. Our experiments demonstrate that FDKN remains in the ballpark of that of DKN, still significantly better than competing approaches, and in one case even overperforming DKN. We show in Table 2 the detailed description of the network structure.

4 Experiments

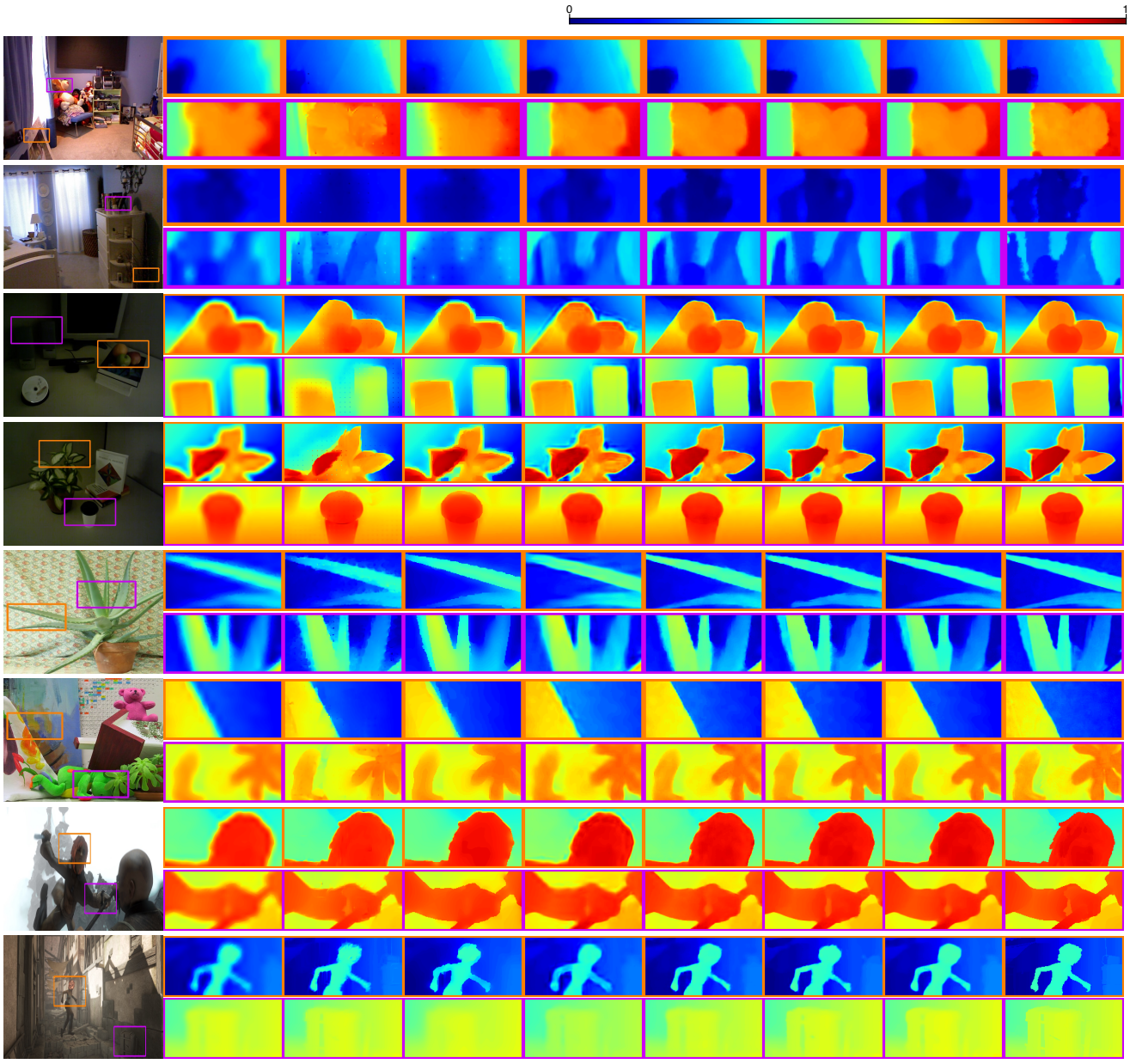
In this section we present a detailed analysis and evaluation of our approach. We apply our models to the tasks of joint image upsampling (Section 4.1), cross-modality image restoration and texture removal (Section 4.2) and semantic segmentation (Section 4.3), and compare them to the state of the art in each case. 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. The results for comparisons have been obtained from the source code provided by the authors.

4.1 Joint image upsampling experiments

Following the experimental protocol of (Li et al. 2016, 2019), we train our models on the task of depth map upsampling using a large number of RGB/D image pairs, and test them on the tasks of (noisy) depth map upsampling, and saliency image upsampling that require selectively transferring the structural details from the guidance image to the target one.

4.1.1 Experimental details

We sample 1,000 RGB/D image pairs of size 640×480 from the NYU v2 dataset (Silberman et al. 2012). We use the same



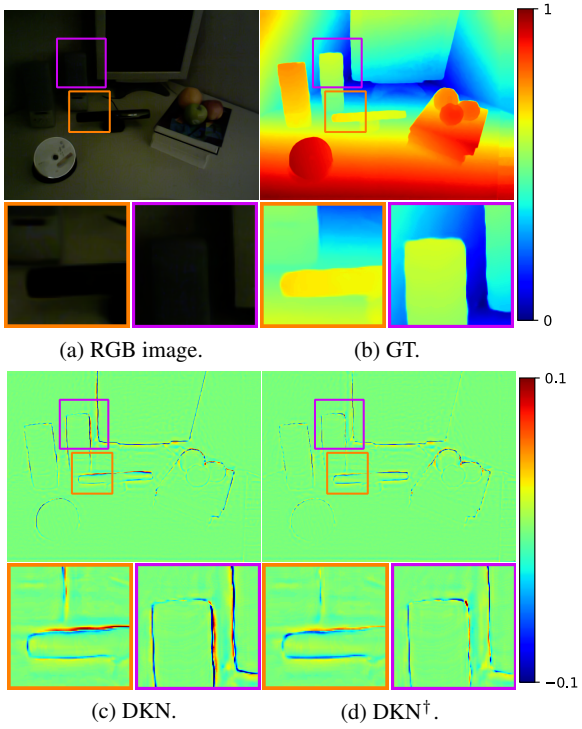
(a) RGB image. (b) GF. (c) TGV. (d) SDF. (e) DJFR. (f) PAC. (g) DKN. (h) FDKN. (i) Ground truth.

Fig. 6 Visual comparison of upsampled depth images (8×): (a) an RGB image, (b) GF (He et al. 2013), (c) TGV (Ferstl et al. 2013), (d) SDF (Ham et al. 2018), (e) DJFR (Li et al. 2019), (f) PAC (Su et al. 2019), (g) DKN, (h) FDKN, and (i) ground truth. Top to bottom: Each two rows show upsampled images on the NYU v2 (Silberman et al. 2012), Lu (Lu et al. 2014), Middlebury (Hirschmuller and Scharstein 2007) and Sintel (Butler et al. 2012) datasets, respectively. Note that we train our models with the NYU v2 dataset, and do not fine-tune them to other datasets.

image pairs as in (Li et al. 2016, 2019) to train the networks. Following (Hui et al. 2016; Gu et al. 2017), we synthesize low-resolution depth images (4×, 8×, 16×) from ground truth by bicubic downsampling. We train different models to upsample depth map with a batch size of 1 for 40k iterations, giving roughly 20 epochs over the training data. We use the Adam optimizer (Kingma and Ba 2015) 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. Data augmentation and regularization techniques such as weight decay and dropout (Krizhevsky et al. 2012) are not used, since 1,000 RGB/D image pairs from the NYU dataset have proven to be sufficient to train our models. All networks are trained end-to-end using `PyTorch` (Paszke et al. 2017).

4.1.2 Results

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. The inputs to our models are a high-resolution color image and a low-resolution depth image upsampled using bicubic interpolation.



(a) RGB image. (b) GT.

(c) DKN. (d) DKN†.

Fig. 7 An example from the Lu dataset (Lu et al. 2014) where the RGB guidance image does not help. From left to right: (a) an RGB image, (b) ground-truth depth, and results from (c) DKN and (d) DKN†, rendered as difference maps between upsampling results and ground truth. In this example, the poor contrast of color edges in dark regions hampers the ability of the RGB guidance to assist upsampling, and DKN† performs better than DKN.

- Middlebury dataset (Hirschmuller and Scharstein 2007; Scharstein and Pal 2007): We use the 30 RGB/D image pairs from the 2001-2006 datasets provided by Lu (Lu et al. 2014).
- Lu dataset (Lu et al. 2014): This provides 6 RGB/D image pairs acquired by the ASUS Xtion Pro camera.
- NYU v2 dataset (Silberman et al. 2012): It consists of 1,449 RGB/D image pairs captured with the Microsoft Kinect (Zhang 2012) using structured light, similar to the Middlebury dataset. We exclude the 1,000 pairs used for training, and use the rest (449 pairs) for evaluation.
- Sintel dataset (Butler et al. 2012): 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[3].

We compare our method with the state of the art in Table 3. It shows the average RMSE between upsampling results and ground truth. DJF (Li et al. 2016), DJFR (Li et al. 2019), and PAC (Su et al. 2019) use nearest-neighbor downsampling to simulate low-resolution depth images, which causes aliasing artifacts. To avoid this problem, DMSG (Hui et al. 2016) and DG (Gu et al. 2017) exploit a bicubic downsampling technique. For fair comparison, we report in Table 3 the results obtained by all methods, DJF, DJFR, PAC, DMSG, DG using the exact same experimental setting as that used for our method as described in Section 4.1.1, including using the same training data and the same bicubic downsampling technique[4]. All other numbers except those for the Sintel dataset are taken from (Li et al. 2016, 2019). From this table, we observe four things: (1) Our models outperform the state of the art including CNN-based methods (Hui et al. 2016; Li et al. 2016, 2019; Su et al. 2019) by significant margins in terms of RMSE, even without the residual connection (DKN w/o Res. and FDKN w/o Res.). For example, DKN decreases the average RMSE by 32% (4×), 34% (8×) and 29% (16×) compared to DJFR. (2) Our models trained *without the guidance of color images* (DKN† and FDKN†), using the depth map only, also outperform the state of the art. (3) We can clearly see that our models perform well on both synthetic and real datasets (e.g., the Sintel and NYU v2 datasets), and generalize well to other images (e.g., on the Middlebury dataset) outside the training dataset. Note that we train our models with the NYU v2 dataset, and do not fine-tune them to other datasets. (4) FDKN retains the superior performance of DKN.

Figure 6 shows a visual comparison of the upsampled depth images (8×). 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 the weighted average with spatially-variant kernels and an adaptive neighborhood system in joint image filtering.

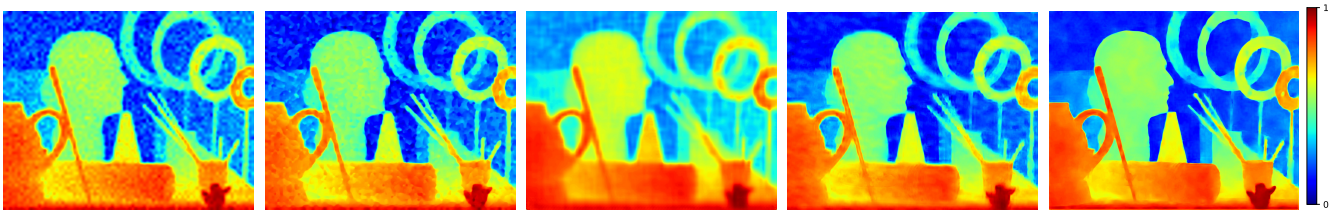
We show in Fig. 7 a failure example on the Lu dataset (Lu et al. 2014). As the color images in the dataset are captured in low-light conditions, the color boundaries become less reliable, and our model trained without the color image guidance, DKN†, outperforms DKN. Specifically, the percentage of the image that DKN† outperforms DKN (8×) is as follows: 43% (13/30), 100% (6/6), 2% (10/449), and 27% (238/864) for the Middlebury, Lu, NYU v2, and Sintel datasets, respectively. This suggests that using guidance images does not *always* give better results. *In general*, RGB guidance images provide structural information (e.g., gradients along occluding edges) useful in depth map upsampling, but they may also contain unrelated details

[3] We discard the 200 pairs that provide RGB images only.

[4] We have also performed the same comparison using neareast-neighbor downsampling for all methods, except for DG, as it does not provide a source code for training. The results are omitted since they all demonstrate worse performance in that setting.

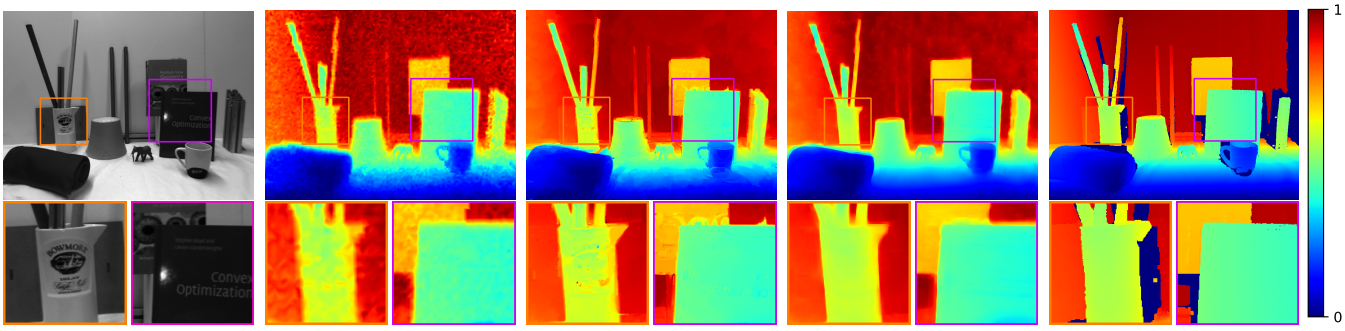
Table 4 Quantitative comparison with the state of the art on noisy depth map upsampling in terms of average RMSE on the noisy Middlebury dataset (Park et al. 2011). †: The model trained with a synthetic dataset.

	Art			Books			Moebius		
Methods	4×	8×	16×	4×	8×	16×	4×	8×	16×
Bicubic	6.07	7.27	9.59	5.15	5.45	5.97	5.51	5.68	6.11
DMSG (Hui et al. 2016)	6.19	7.26	9.53	5.38	5.18	5.20	5.48	5.06	5.36
DJFR (Li et al. 2019)	4.25	6.43	9.05	2.20	3.35	4.94	2.39	3.51	4.56
PAC (Su et al. 2019)	5.34	7.69	10.66	2.11	3.12	4.60	2.21	3.38	4.72
PDN† (Riegler et al. 2016a)	3.11	4.48	7.35	1.56	2.24	3.46	1.68	2.48	3.62
FDKN w/o Res.	3.24	4.57	7.67	1.55	2.18	3.32	1.76	2.57	3.92
FDKN	3.14	4.47	7.61	1.49	2.13	3.40	1.70	2.53	3.91
DKN w/o Res.	**2.98**	4.25	7.78	1.47	2.16	3.62	1.64	2.43	4.03
DKN	3.01	**4.14**	**7.01**	**1.44**	**2.10**	**3.09**	**1.63**	**2.39**	**3.55**



(a) Input. (b) DMSG. (c) PAC. (d) DJFR. (e) DKN.

Fig. 8 Visual comparison of noisy depth map upsampling (8×) on the art sequence in the noisy Middlebury dataset (Park et al. 2011): (a) Input, (b) DMSG (Hui et al. 2016), (c) PAC (Su et al. 2019), (d) DJFR (Li et al. 2019), and (e) DKN.



(a) Intensity. (b) LR ToF. (c) TGV. (d) DKN. (e) GT.

Fig. 9 Visual comparison of noisy depth map upsampling on the books sequence in the ToFMark dataset (Ferstl et al. 2013): (a) Intensity, (b) LR ToF, (c) TGV (Ferstl et al. 2013), (d) DKN, and (e) GT.

Table 5 Quantitative comparison of noisy depth map upsampling in terms of average RMSE on the ToFMark dataset (Ferstl et al. 2013). †: The model trained with a synthetic dataset.

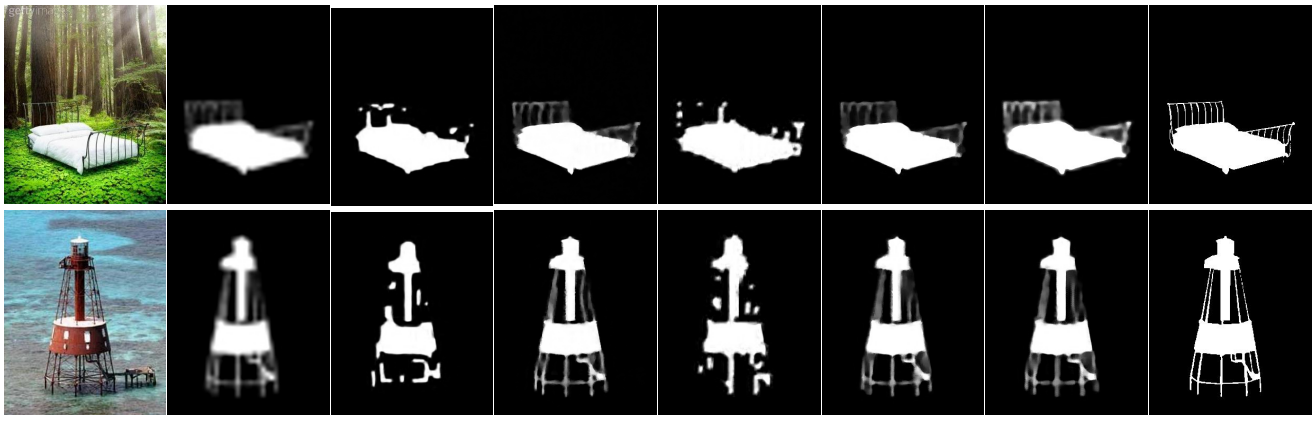
Methods	Books	Devil	Shark
NN	30.46	27.53	38.21
Bilinear	29.11	25.34	36.34
JBU (Kopf et al. 2007)	27.82	25.30	34.79
GF (He et al. 2013)	27.11	23.45	33.26
TGV (Ferstl et al. 2013)	24.00	23.19	29.89
PDN† (Riegler et al. 2016a)	23.74	20.47	28.81
DKN w/o Res.	**23.45**	**19.97**	**27.91**

Table 6 Quantitative comparison on saliency map upsampling in terms of weighted F-scores (Margolin et al. 2014). We use 5,168 images from the DUT-OMRON dataset (Yang et al. 2013).

Methods	Weighted-Fscore
Bicubic Int.	0.886
GF (He et al. 2013)	0.890
SDF (Ham et al. 2018)	0.885
DMSG (Hui et al. 2016)	0.938
DJFR (Li et al. 2019)	0.925
PAC (Su et al. 2019)	0.943
DKN	0.944
FDKN	**0.961**

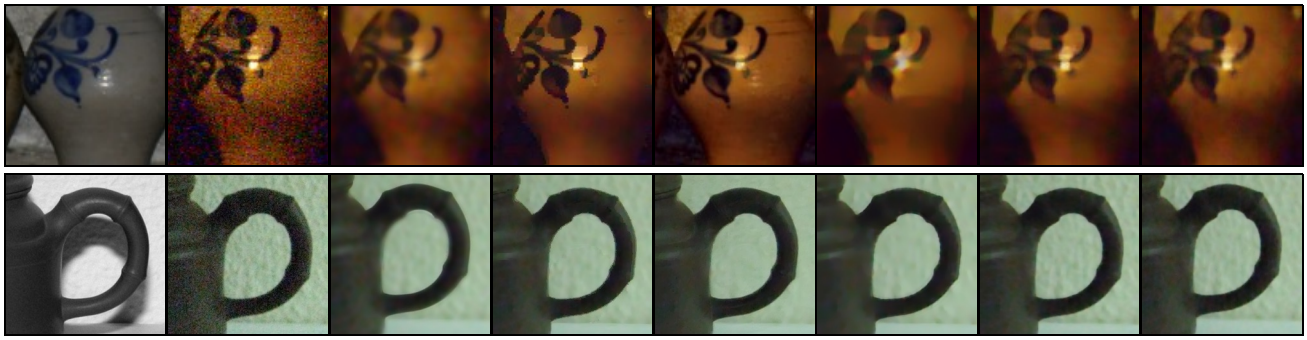
(e.g., gradients along texture edges) that at time degrade performance.

Noisy depth map upsampling. To show the robustness of our models on noisy data, we train our models using pairs of noisy low-resolution/ground-truth depth images in the NYU



(a) RGB image. (b) Bicubic Int. (c) DMSG. (d) PAC. (e) DJFR. (f) DKN. (g) FDKN. (h) Ground truth.

Fig. 10 Visual comparison of saliency map upsampling (8×) on the DUT-OMRON dataset (Yang et al. 2013): (a) an RGB image, (b) Bicubic Int., (c) DMSG (Hui et al. 2016), (d) PAC (Su et al. 2019), (e) DJFR (Li et al. 2019), (f) DKN, (g) FDKN, and (h) ground truth.



(a) Guidance. (b) Target. (c) GF. (d) SDF. (e) Yan. (f) DJF. (g) DKN. (h) FDKN.

Fig. 11 Examples of cross-modality noise reduction for (top) flash/non-flash denoising and (bottom) RGB/NIR denoising: (a-b) Guidance and target images, (c) GF (He et al. 2013), (d) SDF (Ham et al. 2018), (e) Yan (Yan et al. 2013), (f) DJF (Li et al. 2016), (g) DKN, and (h) FDKN. 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 (Yan et al. 2013), specially designed for this task, gives the best results.

v2 (Silberman et al. 2012) dataset. We simulate noisy low-resolution depth images following the protocol of (Riegler et al. 2016a). The other experimental settings are the same as in Section 4.1.1. We compare our models with the state of the art in Table 4 including CNN-based methods (Hui et al. 2016; Li et al. 2019; Su et al. 2019; Riegler et al. 2016a) on the noisy Middlebury dataset. To obtain the results of DMSG (Hui et al. 2016), DJFR (Li et al. 2019), and PAC (Su et al. 2019), we retrain the models using the same image pairs as ours. The results of PDN are taken from (Riegler et al. 2016a)[5]. In Table 5, we compare our model with other methods (Kopf et al. 2007; He et al. 2013; Ferstl et al. 2013; Riegler et al. 2016a) on the ToFMark dataset, where all numbers are taken from (Riegler et al. 2016a). From these tables, we can see that our models outperform the state of the art, demonstrating that they are quite effective to handle synthetic and real noisy data, even better than PDN (Riegler et al. 2016a) that requires many iterations to compute the primal-dual algorithm in testing. Figures 8 and 9 show visual comparisons on the noisy Middlebury and ToFmark datasets, respectively. We can see that our models suppress the noise, while preserving sharp depth boundaries (e.g., thin sticks and small holes in Fig. 8) and being robust to texture-copying artifacts (e.g., texts in a vase and books in Fig. 9).

Saliency map upsampling. To evaluate the generalization ability of our models for depth map upsampling on other tasks, we apply them trained with the NYU v2 dataset to saliency map upsampling without fine-tuning. We downsample saliency maps (×8) in the DUT-OMRON dataset (Yang et al. 2013), and then upsample them under the guidance of high-resolution color images. We show in Table 6 a comparison of weighted F-measure (Margolin et al. 2014) between upsampled images and the ground truth. Figure 10 shows examples of the upsampling results by the state

[5] PDN uses synthetic depth maps created by a 3D renderer. The authors provide the source code online but despite our best efforts, we have not been able to retrain the corresponding models in the same setting as ours. We thus simply indicate the original results from (Riegler et al. 2016a)

of the art and our models. The results show that our models outperform others including CNN-based methods (Li et al. 2019; Hui et al. 2016; Su et al. 2019).

4.2 Cross-modality image restoration and texture removal experiments

4.2.1 Experimental details

Following (Li et al. 2016, 2019), 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 our models for denoising depth images with RGB/D image pairs from the NYU v2 dataset (Silberman et al. 2012). The models for depth noise removal are similarly trained to those for joint image upsampling in Section 4.1.1 under the guidance of high-resolution RGB images but with 4k iterations. Noisy depth images are synthesized by adding Gaussian noise with zero mean and variance of 0.005. The models are then applied to the tasks of cross-modality image restoration and texture removal without fine-tuning. For comparison, average RMSE for GF (He et al. 2013), Yan (Yan et al. 2013), SDF (Ham et al. 2018), DJF (Li et al. 2016), 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. We do not use the residual connection for noise removal tasks, since we empirically find that it does not help in this case. We only show qualitative results in these tasks, since ground truth is not available. All previous works (e.g., (Li et al. 2016; Yan et al. 2013; Li et al. 2019; He et al. 2013)) we are aware of for these tasks offer qualitative results only.

4.2.2 Results

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. 11. Qualitatively, our models outperform other state-of-the-art methods (Li et al. 2016; Ham et al. 2018; He et al. 2013). For example, GF (He et al. 2013) using guidance images only cannot deal with gradient reversal in flash NIR images, resulting in smoothed edges. SDF (Ham et al. 2018) and DJF (Li et al. 2016) use both guidance and target images. However, SDF tends to enhance edges, giving over-sharpened results, while DJF overly smooths images. In contrast, our models 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. The method of (Yan et al. 2013), specially designed for this task, gives the best results.

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. 12. Our models outperform DJF (Li et al. 2016) trained for denoising depth images, suggesting that a generalization ability is better. In particular, they remove textures without artifacts while maintaining other high-frequency structures such as image boundaries and corners. Our models give results comparable to other methods including RTV (Xu et al. 2012) and Cov (Karacan et al. 2013) specially designed for texture removal. 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 (Li et al. 2016). We empirically find that 4 iterations are enough to get satisfactory results, and use the same number of iterations for all experiments.

4.3 Semantic segmentation experiments

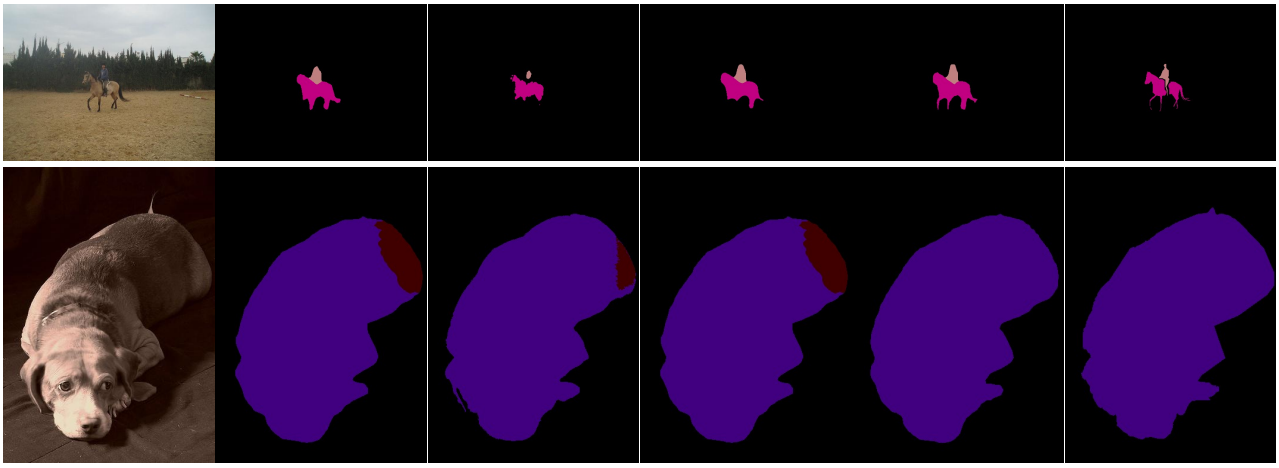
CNNs commonly use max pooling and downsampling to achieve invariance, but this degrades localization accuracy especially at object boundaries (Long et al. 2015). DeepLab (Chen et al. 2018) overcomes this problem using probabilistic graphical models. It applies a fully connected CRF (Krähenbühl and Koltun 2011) to the response of the final layer of a CNN. Zheng et al. interpret CRFs as recurrent neural networks which are then plugged into as a part of a CNN (Zheng et al. 2015), making it possible to train the whole network end-to-end. Recently, Wu et al. have proposed a layer to integrate guided filtering (He et al. 2013) into CNNs (Wu et al. 2018). Instead of using CRFs (Chen et al. 2018; Krähenbühl and Koltun 2011; Zheng et al. 2015) or guided image filtering (He et al. 2013), we apply the FDKN to the response of the final layer of DeepLab v2 (Chen et al. 2018) for semantic segmentation.

4.3.1 Experimental details

Following the experimental protocol of (Wu et al. 2018), we plug FDKN into DeepLab-v2 (Chen et al. 2018), which uses ResNet-101 (He et al. 2016) pretrained for ImageNet classification, as a part of CNNs for semantic segmentation, instead of applying a fully connected conditional random field (CRF) (Krähenbühl and Koltun 2011) 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. We use the Pascal VOC 2012 dataset (Everingham et al. 2015) that contains $1,464$, $1,449$, and $1,456$ images for training, validation and test, respectively. Following (Chen et al. 2018; Wu et al. 2018), we augment the training dataset by the annotations provided by (Hariharan et al. 2011), 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 (Chen et al. 2018) with learning rate of 2.5×10^{-4}

(a) Input image. (b) RGF. (c) RTV. (d) Cov. (e) SDF. (f) DJF. (g) DKN. (h) FDKN.

Fig. 12 Visual comparison of texture removal for regular (top) and irregular (bottom) textures: (a) an input image, (b) RGF (Zhang et al. 2014), (c) RTV (Xu et al. 2012), (d) Cov (Karacan et al. 2013), (e) SDF (Ham et al. 2018), (f) DJF (Li et al. 2016), (g) DKN, and (h) FDKN.



(a) RGB image. (b) Baseline. (c) DenseCRF. (d) DGF. (e) FDKN. (f) Ground truth.

Fig. 13 Visual comparison of semantic segmentation on the validation set of Pascal VOC 2012 benchmark (Everingham et al. 2015): (a) an RGB image, (b) baseline (Chen et al. 2018), (c) denseCRF (Krähenbühl and Koltun 2011), (d) DGF (Wu et al. 2018), (e) FDKN, and (f) ground truth. 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.

and 2.5×10^{-3} for DeepLab-v2 and FDKN, respectively. 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.

4.3.2 Results

We show in Table 7 mean intersection-over-union (IoU) scores for the validation set in the Pascal VOC 2012 benchmark (Everingham et al. 2015). To the baseline method (DeepLab v2 w/o CRF (Chen et al. 2018)), we add CRF (Krähenbühl and Koltun 2011), guided filtering (Wu et al. 2018), or FDKN layers. This table shows that our model quantitatively yields better accuracy in terms of mean

Table 7 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 (Everingham et al. 2015).

Methods	Mean IoU
Baseline (Chen et al. 2018)	70.69
DenseCRF (Krähenbühl and Koltun 2011)	71.98
DGF (Wu et al. 2018)	72.96
FDKN	**73.60**

IoU than other state-of-the-art methods. Examples of semantic segmentation are shown in Fig. 13. 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 in our models, and show the effects of different parameters for depth map upsampling ($8\times$) on the NYU v2 dataset. We also discuss other issues including kernel visualization, runtime, training loss, and upscaling factors for training and testing.

Network architecture. We show the average RMSE for six variants of our models in Table 8. The baseline models learn kernel weights from the guidance images only. From the second row, we can see that our models trained using the target images only give better results than the baseline, indicating that using the guidance images 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. The fourth and fifth rows show that learning the offsets significantly boosts the performance of our models. The average RMSE of DKN trained using the guidance or target images only decreases from 5.92 to 5.37 and from 5.24 to 4.06, respectively, for the 3×3 kernel. The last two rows demonstrate 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. Figure 14 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 (Li et al. 2016), PAC (Su et al. 2019) and DMSG (Hui et al. 2016), this allows sub-pixel information aggregation.

Our models use a two-stream network to extract feature maps from the guidance and target images. We can regress the weights and offsets with concatenated guidance and target images passed to a single network. This model, however, gives worse errors than our two-stream DKN. In particular, the RMSE increases from 3.26 to 3.50 for filter kernels of size 3×3. This suggests that the different feature maps from the two-stream architecture are better to estimate the kernel weights and offsets. A similar finding is noted in DJF (Li et al. 2016).

Our models use sigmoid and mean subtraction layers for weight regression. We could use a softmax layer that makes all elements larger than 0 and smaller 1 as in the sigmoid layer. The sigmoid and mean subtraction layer can be replaced with a single linear one. We compare in Table 9 the performance of DKN and FDKN with softmax or sigmoid layers, and when using mean subtraction or not. This demonstrates that 1) the softmax layer does not perform as well as the sigmoid layer, and 2) constraints on weight regression using sigmoid and mean subtraction layers give better results.

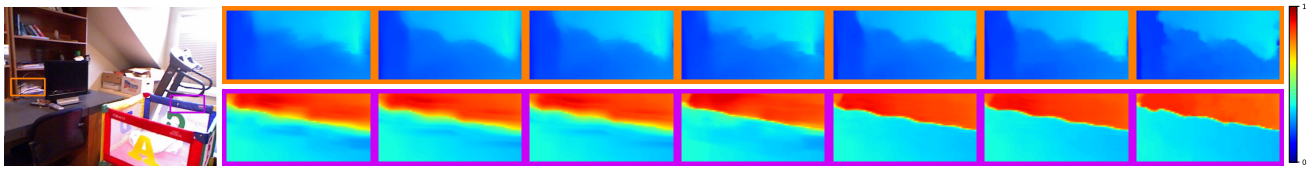
Kernel size. Table 8 also compares the performances of networks with different size of kernels. We enlarge the kernel size gradually from 3×3 to 25×25 and compute the average RMSE. From the third row, we observe that the performance improves until size of 15×15. Increasing size further does not give 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 8 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 0.17, 0.18 and 0.19 seconds, respectively, with a Nvidia Titan XP. A 3×3 size offers a good compromise in terms of RMSE and runtime and this is what we have used in all experiments.

Feature channels. In Table 10, 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) 0.11 RMSE gain. Consequently, we choose the number of feature channels $n_1 = 128$ and $n_2 = 128$ for both models.

DownConv for DKN. We empirically find that extracting features from large receptive fields is important to incorporate context for weight and offset learning. For example, reducing the size from 51×51 to 23×23 causes an increase of the average RMSE from 3.26 to 5.00 for the 3×3 kernel. The DKN without DownConv layers can be implemented in a single forward pass, but requires more parameters (1.6M vs. 1.1M for DKN) to maintain the same receptive field size, with a total number of convolutions increasing from 0.6M to 1M at each pixel. We may use dilated convolutions (Yu and Koltun 2016) that support large receptive fields without loss of resolution. When using the same receptive field size

Table 8 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. For each network, numbers in bold indicate the best performance and underscored ones are the second best.

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	–	–	–
	✓		✓		4.06 / 4.13	4.09 / 4.13	4.13 / 4.14	–	–	–
✓	✓	✓	✓		3.36 / 3.67	3.32 / 3.65	3.33 / 3.66	–	–	–
✓	✓	✓	✓	✓	3.26 / 3.58	3.21 / 3.53	**3.19** / **3.52**	–	–	–



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

Fig. 14 Visual comparison of different networks (DKN) for depth upsampling ($8\times$) using the kernel of size 3×3. (a) an 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.

Table 9 Quantitative comparison of different components for weight regression for filter kernels of size 3×3.

Activation function		Mean subtraction	DKN	FDKN
Softmax	Sigmoid			
			3.74	3.98
	✓		5.72	5.89
✓		✓	3.49	3.65
	✓	✓	**3.26**	**3.58**

as 51×51, the average RMSE for dilated convolutions increases from 3.26 to 4.30 for the 3×3 kernel. We can also use deconvolutional layers, similar to the U-Net architecture (Ronneberger et al. 2015) for upsampling. They, however, produce checkerboard artifacts (Odena et al. 2016), which may degrade the performance. The resampling technique (Fig. 5) thus appears to be the preferable alternative.

Training loss. Average RMSEs for L1, L2 and L1+perceptual losses are 3.58, 3.69, and 3.66, respectively, for the task of depth upsampling ($8\times$) on the NYU v2 dataset. The L1 loss is robust to outliers, preserving depth boundaries better than the L2 one. The perceptual loss, typically using a network (Simonyan and Zisserman 2015) pretrained for the ImageNet classification, does not provide a gain on the RMSE performance.

Runtime. Our network consists of two parts: 1) feature extraction and weight & offset regression, and 2) weighted average (Fig. 2). The second part takes less than 0.001s for all models. The first part for DKN, DKN†, FDKN, FDKN† takes 0.165s, 0.086s, 0.011s, 0.01s respectively for images

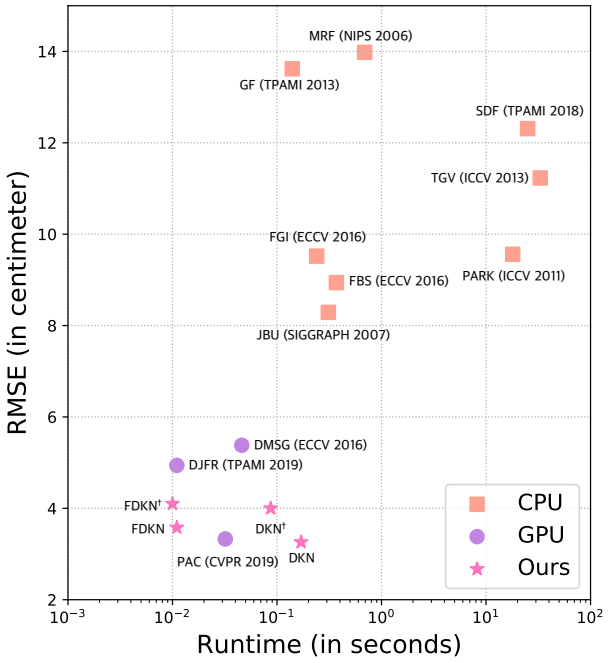


Fig. 15 Runtime and root mean squared errors (RMSE) comparison of upsampled depth maps ($8\times$) on the NYU v2 (Silberman et al. 2012) dataset. †: Our models trained with the depth map only without any guidance.

Table 10 Quantitative comparison of using different number of channels for feature extraction. The results of DKN and FDKN are separated by "/". We denote by n_i $(i = 1, 2)$ 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)	**0.17** / **0.01**	0.20 / **0.01**	0.22 / **0.01**	0.27 / 0.02	0.36 / 0.02
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 11 Runtime comparison for images of size 640×480 on the NYU v2 (Silberman et al. 2012) dataset. †: Our models trained with the depth map only without any guidance.

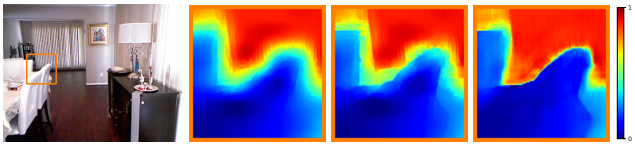
Methods	GPU Times(s)	CPU Times (s)
MRF (Diebel and Thrun 2006)	–	0.69
GF (He et al. 2013)	–	**0.14**
JBU (Kopf et al. 2007)	–	0.31
TGV (Ferstl et al. 2013)	–	33
Park (Park et al. 2011)	–	17
SDF (Ham et al. 2018)	–	25
FBS (Barron and Poole 2016)	–	0.37
DMSG (Hui et al. 2016)	0.04	1.4
DJFR (Li et al. 2019)	**0.01**	1.3
PAC (Su et al. 2019)	0.03	1.4
DKN†	0.09	3.2
FDKN†	**0.01**	0.9
DKN	0.17	5.4
FDKN	**0.01**	1.0

Table 12 RMSE comparison of using different upscaling factors for training and testing on depth map upsampling.

Train/Test	4×	8×	16×
4×	**1.62**	6.70	11.24
8×	3.93	**3.26**	10.53
16×	9.04	8.61	**6.51**

of size 640×480. For comparison with other methods, table 11 shows runtime on the same machine. We report the GPU runtime for DMSG (Hui et al. 2016), DJFR (Li et al. 2019), PAC (Su et al. 2019), and our models with a Nvidia Titan XP. DKN is slower than DMSG (Hui et al. 2016), PAC (Su et al. 2019) and DJFR (Li et al. 2019), but yields a significantly better RMSE (Fig. 15 and Table 3). FDKN runs about $17\times$ faster than the DKN, as fast as DJFR, but with significantly higher accuracy. We also report the CPU runtime with an Intel i5 3.3 GHz, demonstrating that FDKN is as fast or faster than other CNN-based methods, even on CPUs.

Upscaling factors for training and testing. Table 12 compares the average RMSE on the NYU dataset (Silberman et al. 2012), when the scale factors for training and test are different. It shows that the performance is degraded. This may be handled by a scale augmentation technique during training (Kim et al. 2016). A visual comparison is shown in Fig. 16.



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

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

Table 13 RMSE comparison by varying the number of training data on depth map upsampling $(8\times)$.

Datasets	Methods	10	50	100	200	500	700	1000
NYU v2	DMSG	7.40	6.32	5.97	5.64	5.41	5.35	5.38
	FDKN	5.01	4.28	3.98	3.77	3.61	3.60	3.58
	DKN	4.73	3.97	3.62	3.33	3.27	3.25	3.26
Sintel	DMSG	8.62	8.08	7.65	7.46	7.27	7.21	7.24
	FDKN	6.04	5.37	5.16	5.05	4.98	4.98	4.96
	DKN	5.79	5.24	5.01	4.85	4.82	4.74	4.77

Kernel prediction vs. direct regression. Our model has several advantages over current CNN-based approaches that directly regress the filtering output. First, the direct regression may overfit the particular characteristics of training data, especially when the number of training samples is small. In contrast, weighted averaging smooths the output and acts as a regularizer, suggesting that our model is not seriously affected by the number of training samples. To demonstrate this, we evaluate the average RMSE performance in Table 13, when varying the size of the training data. We train the DKN, FDKN and DMSG (Hui et al. 2016), where the filtering output is directly regressed from input images, for depth map upsampling $(8\times)$ while gradually increasing the number of training samples from 10 to $1,000$ in the NYU v2 dataset (Silberman et al. 2012). We test them in the same configuration as in Table 3.

Table 13 shows that our models are more robust to the size of training data and generalize better to other images (e.g., on the Sintel dataset (Butler et al. 2012)) outside the training dataset than the direct regression approach,

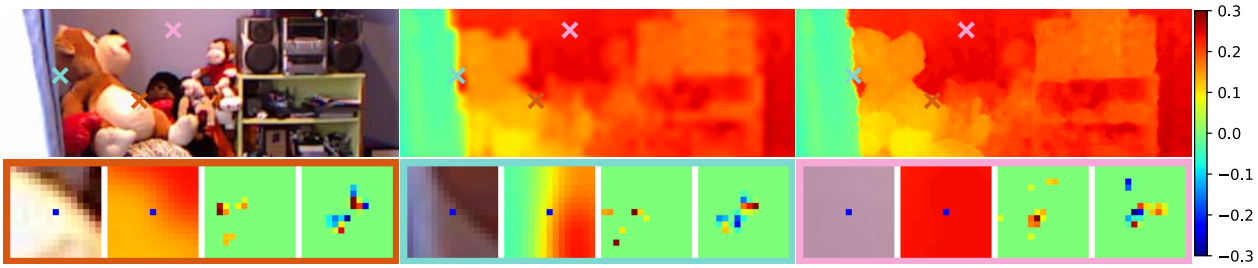


Fig. 17 Visualization of filter kernels. Top: (From left to right) an RGB image, a low-resolution depth image, and an upsampling result by DKN. Bottom: (From left to right) Snippets of RGB images, low-resolution depth images, and kernels learned w/o and w/ the residual connection. The center positions in the RGB and depth images are denoted by blue dots. The kernel weights are plotted with a heat map. (Best viewed in color.)

even with more learnable parameters (1.1M for DKN and 0.6M for FDKN vs. 0.43M for DMSG (Hui et al. 2016)). 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). Second, the kernels learned by direct regression are defined implicitly and hard to visualize. In contrast, our method learns sparse kernels (i.e., *where* to aggregate) explicitly. We can interpret and visualize why kernels learned by our model give smooth results while preserving edges (Fig. 17), and this also gives a clue for tuning hyper-parameters. For examples, we can reduce the maximum range of offset locations (i.e., the size of the filter kernel) and the number of weights (i.e., the total number of samples to aggregate), when the weights are concentrated on central parts of the kernels, and a few of them are highly confident, respectively. Note that sparsely aggregating subpixel information is not feasible for direct regression approaches (e.g., DMSG (Hui et al. 2016) and DJF (Li et al. 2016)). Finally, our model can be applied to any tasks requiring an explicit weighted averaging processing beyond (joint) image filtering, as confirmed for the task of semantic segmentation in Section 4.3.

Kernel visualization. We show in Fig. 17 some examples of 3×3 filter kernels estimated by the DKN with/without the residual 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. Note that nearby pixels have lower weights than non-neighboring ones especially at depth boundaries, as they are blurred in the low-resolution depth image. This suggests that structural details are also related to further away pixels. A similar finding is noted in nonlocal means (Buades et al. 2005). (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.

6 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 $17\times$ 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. In future work, we will explore network architectures for sparse-to-dense interpolation such as depth completion (Tang et al. 2019) and optical flow propagation (Revaud et al. 2015).

A Appendix: Efficient implementation

We use the shift-and-stitch approach (Long et al. 2015; Niklaus et al. 2017) that stitches the network outputs from shifted versions of the input (Fig. 18). 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 a total of 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 $d \times d$,

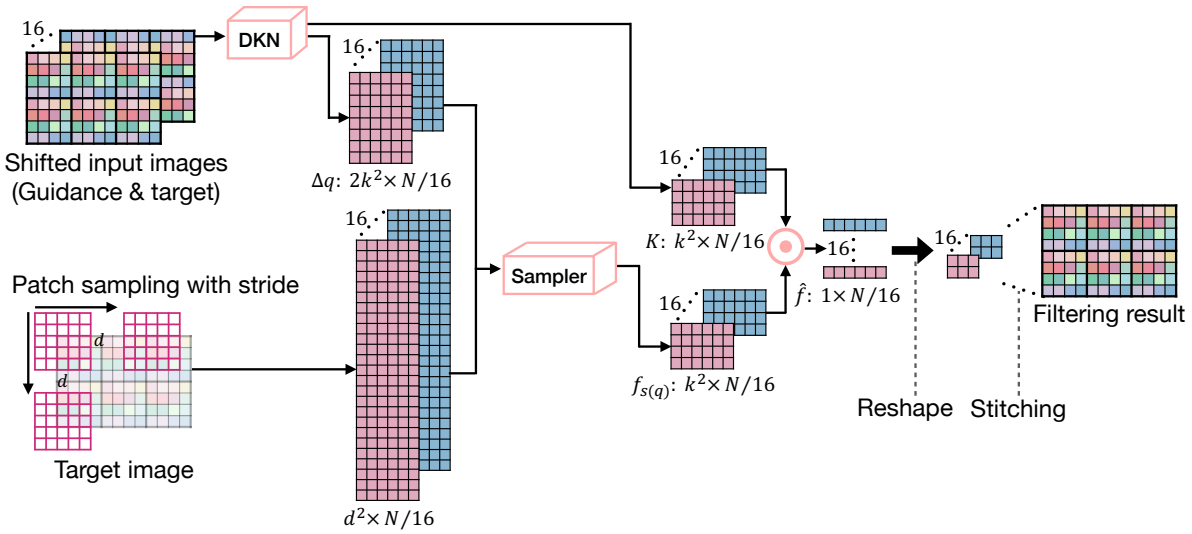


Fig. 18 Efficient implementation using a shift-and-stitch approach. We denote d by the maximum range of the sampling location $\mathbf{s}(\mathbf{q})$. We shift the input images and compute the filtering result for each shifted input. We then stitch them up to get a filtering result that has the same 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.)

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.

Acknowledgements The authors would like to thank Yijun Li for helpful discussion. This work was supported in part by Samsung Research Funding & Incubation Center for Future Technology (SRFC-IT1802-06), the Louis Vuitton/ENS chair on artificial intelligence, the Inria/NYU collaboration agreement, and the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

References

Bako S, Vogels T, McWilliams B, Meyer M, Novák J, Harvill A, Sen P, Derose T, Rousselle F (2017) Kernel-predicting convolutional networks for denoising Monte Carlo renderings. ACM Trans Graph 36(4):97

Barron JT, Poole B (2016) The fast bilateral solver. In: Proc. Eur. Conf. Comput. Vis.

Buades A, Coll B, Morel JM (2005) A non-local algorithm for image denoising. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.

Butler DJ, Wulff J, Stanley GB, Black MJ (2012) A naturalistic open source movie for optical flow evaluation. In: Proc. Eur. Conf. Comput. Vis.

Chen LC, Papandreou G, Kokkinos I, Murphy K, Yuille AL (2018) Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE Trans Pattern Anal Mach Intell 40(4):834–848

Choy CB, Gwak J, Savarese S, Chandraker M (2016) Universal correspondence network. In: Adv. Neural Inf. Process. Syst.

Dai J, Qi H, Xiong Y, Li Y, Zhang G, Hu H, Wei Y (2017) Deformable convolutional networks. In: Proc. Int. Conf. Comput. Vis.

Diebel J, Thrun S (2006) An application of Markov random fields to range sensing. In: Adv. Neural Inf. Process. Syst.

Everingham M, Eslami SA, Van Gool L, Williams CK, Winn J, Zisserman A (2015) The pascal visual object classes challenge: A retrospective. Int J Comput Vis 111(1):98–136

Farbman Z, Fattal R, Lischinski D, Szeliski R (2008) Edge-preserving decompositions for multi-scale tone and detail manipulation 27(3):67

Ferstl D, Reinbacher C, Ranftl R, Rüther M, Bischof H (2013) Image guided depth upsampling using anisotropic total generalized variation. In: Proc. Int. Conf. Comput. Vis.

Ferstl D, Rüther M, Bischof H (2015) Variational depth superresolution using example-based edge representations. In: Proc. Int. Conf. Comput. Vis.

Getreuer P, Garcia-Dorado I, Isidoro J, Choi S, Ong F, Milanfar P (2018) Blade: Filter learning for general purpose computational photography. In: Proc. IEEE Conf. Computational Photography

Gu S, Zuo W, Guo S, Chen Y, Chen C, Zhang L (2017) Learning dynamic guidance for depth image enhancement. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.

Ham B, Cho M, Schmid C, Ponce J (2016) Proposal flow. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.

Ham B, Cho M, Ponce J (2018) Robust guided image filtering using nonconvex potentials. IEEE Trans Pattern Anal Mach Intell 40(1):192–207

Hariharan B, Arbelaez P, Bourdev L, Maji S, Malik J (2011) Semantic contours from inverse detectors. In: Proc. Int. Conf. Comput. Vis.

He K, Sun J, Tang X (2013) Guided image filtering. IEEE Trans Pattern Anal Mach Intell 35(6):1397–1409

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.

Hirschmuller H, Scharstein D (2007) Evaluation of cost functions for stereo matching. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.

Hosni A, Rhemann C, Bleyer M, Rother C, Gelautz M (2013) Fast cost-volume filtering for visual correspondence and beyond. IEEE

Trans Pattern Anal Mach Intell 35(2):504–511
Hui TW, Loy CC, Tang X (2016) Depth map super-resolution by deep multi-scale guidance. In: Proc. Eur. Conf. Comput. Vis.
Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proc. Int. Conf. Machine Learning
Jaderberg M, Simonyan K, Zisserman A, et al. (2015) Spatial transformer networks. In: Adv. Neural Inf. Process. Syst.
Jia X, De Brabandere B, Tuytelaars T, Gool LV (2016) Dynamic filter networks. In: Adv. Neural Inf. Process. Syst.
Karacan L, Erdem E, Erdem A (2013) Structure-preserving image smoothing via region covariances. ACM Trans Graph 32(6):176
Kim J, Kwon Lee J, Mu Lee K (2016) Accurate image super-resolution using very deep convolutional networks. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Proc. Int. Conf. Learning Representations
Kopf J, Cohen MF, Lischinski D, Uyttendaele M (2007) Joint bilateral upsampling. ACM Trans Graph 26(3):96
Krähenbühl P, Koltun V (2011) Efficient inference in fully connected crfs with gaussian edge potentials. In: Adv. Neural Inf. Process. Syst.
Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Adv. Neural Inf. Process. Syst.
Levin A, Lischinski D, Weiss Y (2008) A closed-form solution to natural image matting. IEEE Trans Pattern Anal Mach Intell 30(2):228–242
Li Y, Huang JB, Ahuja N, Yang MH (2016) Deep joint image filtering. In: Proc. Eur. Conf. Comput. Vis.
Li Y, Huang JB, Ahuja N, Yang MH (2019) Joint image filtering with deep convolutional networks. IEEE Trans Pattern Anal Mach Intell 41(8):1909–1923
Long J, Shelhamer E, Darrell T (2015) Fully convolutional networks for semantic segmentation. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Lu S, Ren X, Liu F (2014) Depth enhancement via low-rank matrix completion. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Margolin R, Zelnik-Manor L, Tal A (2014) How to evaluate foreground maps? In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Mildenhall B, Barron JT, Chen J, Sharlet D, Ng R, Carroll R (2018) Burst denoising with kernel prediction networks. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Niklaus S, Mai L, Liu F (2017) Video frame interpolation via adaptive convolution. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Odena A, Dumoulin V, Olah C (2016) Deconvolution and checkerboard artifacts. Distill 1(10)
Park J, Kim H, Tai YW, Brown MS, Kweon I (2011) High quality depth map upsampling for 3D-ToF cameras. In: Proc. Int. Conf. Comput. Vis.
Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in PyTorch. In: NIPS-W
Revaud J, Weinzaepfel P, Harchaoui Z, Schmid C (2015) EpicFlow: Edge-preserving interpolation of correspondences for optical flow. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., pp 1164–1172
Riegler G, Ferstl D, Rüther M, Horst B (2016a) A deep primal-dual network for guided depth super-resolution. In: Proc. British Machine Vision Conference
Riegler G, Rüther M, Horst B (2016b) ATGV-Net: Accurate depth super-resolution. In: Proc. Eur. Conf. Comput. Vis.
Romano Y, Isidoro J, Milanfar P (2017) RAISR: Rapid and accurate image super resolution. IEEE Trans Computational Imaging 3(1):110–125
Ronneberger O, Fischer P, Brox T (2015) U-net: Convolutional networks for biomedical image segmentation. In: Proc. Intl. Conf. on Medical image computing and computer-assisted intervention
Scharstein D, Pal C (2007) Learning conditional random fields for stereo. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Shen X, Zhou C, Xu L, Jia J (2015) Mutual-structure for joint filtering. In: Proc. Int. Conf. Comput. Vis.
Shi W, Caballero J, Huszár F, Totz J, Aitken AP, Bishop R, Rueckert D, Wang Z (2016) Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Silberman N, Hoiem D, Kohli P, Fergus R (2012) Indoor segmentation and support inference from rgbd images. In: Proc. Eur. Conf. Comput. Vis.
Simonyan K, Zisserman A (2014) Two-stream convolutional networks for action recognition in videos. In: Adv. Neural Inf. Process. Syst.
Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: Proc. Int. Conf. Learning Representations
Su H, Jampani V, Sun D, Gallo O, Learned-Miller E, Kautz J (2019) Pixel-adaptive convolutional neural networks. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Szeliski R (2006) Locally adapted hierarchical basis preconditioning 25(3):1135–1143
Tang J, Tian FP, Feng W, Li J, Tan P (2019) Learning guided convolutional network for depth completion. arXiv preprint arXiv:190801238
Tomasi C, Manduchi R (1998) Bilateral filtering for gray and color images. In: Proc. Int. Conf. Comput. Vis.
Vogels T, Rousselle F, McWilliams B, Röthlin G, Harvill A, Adler D, Meyer M, Novák J (2018) Denoising with kernel prediction and asymmetric loss functions. ACM Trans Graph 37(4):124
Wang J, Cohen MF (2007) Optimized color sampling for robust matting. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Wu H, Zheng S, Zhang J, Huang K (2018) Fast end-to-end trainable guided filter. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Xu L, Lu C, Xu Y, Jia J (2011) Image smoothing via L0 gradient minimization 30(6):174
Xu L, Yan Q, Xia Y, Jia J (2012) Structure extraction from texture via relative total variation. ACM Trans Graph 31(6):139
Xu L, Ren J, Yan Q, Liao R, Jia J (2015) Deep edge-aware filters. In: Proc. Int. Conf. Machine Learning
Yan Q, Shen X, Xu L, Zhuo S, Zhang X, Shen L, Jia J (2013) Cross-field joint image restoration via scale map. In: Proc. Int. Conf. Comput. Vis.
Yang C, Zhang L, Lu H, Ruan X, Yang MH (2013) Saliency detection via graph-based manifold ranking. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Yang J, Wright J, Huang TS, Ma Y (2010) Image super-resolution via sparse representation. IEEE Trans Image Process 19(11):2861–2873
Yang Q, Yang R, Davis J, Nistér D (2007) Spatial-depth super resolution for range images. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.
Yu F, Koltun V (2016) Multi-scale context aggregation by dilated convolutions. In: Proc. Int. Conf. Learning Representations
Zhang K, Zuo W, Chen Y, Meng D, Zhang L (2017) Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. IEEE Trans Image Process 26(7):3142–3155
Zhang Q, Shen X, Xu L, Jia J (2014) Rolling guidance filter. In: Proc. Eur. Conf. Comput. Vis.
Zhang Z (2012) Microsoft Kinect sensor and its effect. IEEE Trans Multimedia 19(2):4–10
Zheng S, Jayasumana S, Romera-Paredes B, Vineet V, Su Z, Du D, Huang C, Torr PH (2015) Conditional random fields as recurrent neural networks. In: Proc. Int. Conf. Comput. Vis.