Fast End-to-End Trainable Guided Filter

Huikai Wu[1,2]   Shuai Zheng[3] *   Junge Zhang[1,2]   Kaiqi Huang[1,2,4]

[1] CRIPAC & NLPR, Institute of Automation, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] eBay Research
[4] CAS Center for Excellence in Brain Science and Intelligence Technology, Beijing, China

[1]{huikai.wu, jgzhang, kqhuang}@nlpr.ia.ac.cn, [3]shuzheng@ebay.com

Abstract

Image processing and pixel-wise dense prediction have been advanced by harnessing the capabilities of deep learning. One central issue of deep learning is the limited capacity to handle joint upsampling. We present a deep learning building block for joint upsampling, namely guided filtering layer. This layer aims at efficiently generating the high-resolution output given the corresponding low-resolution one and a high-resolution guidance map. The proposed layer is composed of a guided filter, which is reformulated as a fully differentiable block. To this end, we show that a guided filter can be expressed as a group of spatial varying linear transformation matrices. This layer could be integrated with the convolutional neural networks (CNNs) and jointly optimized through end-to-end training. To further take advantage of end-to-end training, we plug in a trainable transformation function that generates task-specific guidance maps. By integrating the CNNs and the proposed layer, we form deep guided filtering networks. The proposed networks are evaluated on five advanced image processing tasks. Experiments on MIT-Adobe FiveK Dataset demonstrate that the proposed approach runs 10-100× faster and achieves the state-of-the-art performance. We also show that the proposed guided filtering layer helps to improve the performance of multiple pixel-wise dense prediction tasks. The code is available at https://github.com/wuhuikai/DeepGuidedFilter.

1. Introduction

Research in image processing and pixel-wise dense prediction enables a variety of applications. For example, image processing techniques enable smooth an image while preserving the edges [39, 41, 43], enhance the details of an image [13, 36], transfer style from a reference image [3, 4],

Figure 1: **Example results of the deep guided filtering networks**. The top row shows the input images, and the bottom row includes the results in full-resolution. We show the results of image retouching, multiscale detail manipulation, non-local dehazing, saliency object detection, and depth estimation from a single image in each column. Best viewed in color.

dehaze the photos [5, 20, 14, 15], and even retouch the images for beautification [6]. In Computer Vision, the pixel-wise dense prediction techniques not only address the problem of segmenting an image into semantic parts [22, 35, 11], but also help to estimate depth from a single image [34], and detect the most salient object in an image [32, 29]. Some of the approaches [18, 10, 9] in these applications have been further advanced by deep learning techniques. One of the central issues is the limited capacity of deep learning techniques to handle the joint up-sampling.

The computational demand and running time of an algorithm are critically important to the user experience in these applications. However, existing techniques usually take large amount of computational cost, especially on high-resolution images. In image processing tasks, one well-known approach to accelerating an algorithm follows a coarse-to-fine fashion, which firstly down-samples the im-

age, executes the algorithm at low-resolution, and then up-samples the result back to the original resolution. However, the main challenge is that how to restore the low-resolution output into the original resolution with high quality.

Similar challenge also occurs in computer vision tasks, such as depth estimation, saliency detection and semantic segmentation. For these tasks, convolutional neural networks (CNNs) usually take a high-resolution image as input and generate a low-resolution output, which often have blurred boundaries [33]. How to combine the edges in high-resolution input with low-resolution output to get the high-resolution output remains an active research area.

In image processing literature, such a problem is usually formulated as joint upsampling and solved with bilateral filter [27], which requires large amount of computation resources. In computer vision tasks, a fully-connected conditional random field is usually applied [8, 44, 38], which relies on bilateral filter and runs slowly. Alternatively, we propose to address this problem by reformulating the guided filter [21], which has been shown better performance regarding the trade-off between speed and accuracy.

However, by directly applying guided filter as post-processing operation, the network trained with low-resolution targets cannot cooperate well with guided filter and the restored high-resolution output usually has low quality. We attribute this as the lack of direct supervision from high-resolution targets. Inspired by the prior work [44], we formulate the guided filter as an fully differentiable module, which could be jointly trained with the entire network and benefit from end-to-end training.

In this paper, we propose a novel deep learning layer for joint up-sampling, namely guided filtering layer. Concretely, we formulate the guided filter into a fully differentiable module, which is expressed as a computational graph consisting of spatially varying linear transformation matrices. A learnable transformation function is also imported into the proposed module, which could generate task-specific guidance map from high-resolution input images and facilitate better restoration by taking advantage of end-to-end training. The proposed layer could be integrated with a predefined network without extra efforts, and all the parameters could be learned in a data-driven manner through fully end-to-end training. We note such a network as Deep Guided Filtering Network (DGF).

The advantages of learning guided filtering layer with CNNs in a data-driven manner are multifolds. First, the entire network is supervised by the high-resolution targets, which forces guided filtering layer to cooperate with the rest CNNs, generating outputs with higher quality. Second, by end-to-end training, the guided filtering layer could generalize well to different tasks by generating task-orient guidance map. Fig 1 demonstrates the high quality results obtained by the proposed layer, ranging from image processing tasks

to computer vision tasks. Experiments further show that our method could achieve the state-of-the-art performance in both quality and speed for image processing tasks.

The main contribution is that we develop an end-to-end trainable guided filtering layer and extend it with a learnable guidance map. When combining with CNNs, the proposed layer could significantly improve the state-of-the-art results in multiple image processing tasks, and runs 10-100× faster than the alternatives. Extra experiments show that our approach could generalize well to dense prediction tasks in computer vision and achieve significant improvements.

2. Related Work

The most prominent contemporary works are along the direction of joint upsampling. Prior works along this line aim to generate the high-resolution output given the low-resolution one and a high-resolution guidance map. Many algorithms have been developed to tackle such problem, among which the closest works to ours are Joint Bilateral Upsampling [27], Bilateral Guided Upsampling [7] and Deep Bilateral Learning [18].

Joint bilateral upsampling [27] applies a bilateral filter [37] to the high-resolution guidance map and obtain a piecewise-smoothing high-resolution output. Such an operation requires large amount of computation resources, though many methods are presented to accelerate bilateral filter [2, 1, 16].

Bilateral Guided Upsampling [7] fits an image operator with a grid of local affine models on the low-resolution input/output pair firstly. The high-resolution output is then generated by applying the local affine model to the high-resolution input image. Compared to our method, this method serves as post-processing operation, while ours could be jointly trained with the entire network.

Deep Bilateral Learning [18] integrates bilateral filter with convolution neural networks, which could be learned through end-to-end training. However, the method requires producing affine coefficients before obtaining outputs, which lacks direct supervision from the targets. For computer vision tasks, the number of affine coefficients is usually very large, which becomes the performance and speed bottlenecks.

Alternatively, Xu *et al.* [40], Liu *et al.* [31] and Yan *et al.* [42] deploy neural networks to generate high-resolution output directly, accelerating the operation by dedicated designed network architectures. Xu *et al.* [40] used deep networks to approximate a variety of edge-preserving filters with a gradient domain training procedure, while Liu *et al.* [31] combined a convolutional network and a set of recurrent networks to approximate various image filters. Yan *et al.* [42] use a fully-connected network that operates on each pixel separately with hand-crafted features as input.

Chen *et al.* [10] propose context aggregation networks

to accelerate a wide variety of image processing operators, which performed superior to the prior works [40, 31, 7, 26, 24], achieving the best results regarding speed and accuracy. Our approach is complementary to this method and other similar approaches, which could deliver comparable or better results and runs 10-100× faster.

Compared to the related works, the proposed guided filtering layer could be end-to-end trained with the entire network and generalize well across different tasks ranging from image processing to computer vision, while achieving the state-of-the-art performance on both quality and speed.

3. Guided Filtering Layer

3.1. Problem Formulation

Given an image I_h and the corresponding low-resolution output O_l, we seek a full-resolution output O_h, which is visually similar to O_l while preserving the edges and details from I_h. To obtain O_h, we formulate the problem as joint upsampling, which is introduced by Johannes *et al.* [27] and solved with bilateral filters, requiring large amount of computation resources. Alternatively, we adopt the guided filter proposed by He *et al.* [21] as a fast joint upsampling solver and formulate it into an end-to-end trainable module.

Algorithm 1: Gradients for Guided Filtering Layer

Input : Low-resolution image I_l
High-resolution image I_h
Low-resolution output O_l
Derivative for high-resolution output ∂O_h
Output: Gradients for all the inputs

1. $\partial b_l = \partial O_h \cdot \nabla_{b_l} f_\uparrow, \ \partial A_l = \partial O_h * G_h \cdot \nabla_{A_l} f_\uparrow - \partial b_l * \bar{G}_l$
2. $\partial \Sigma_{G_l O_l} = \partial A_l / (\Sigma_{G_l} + \epsilon)$

   $\partial \Sigma_{G_l} = -\partial A_l * \Sigma_{G_l O_l} / (\Sigma_{G_l} + \epsilon)^2$
3. $\partial \bar{O}_l = \partial b_l - \partial \Sigma_{G_l O_l} * \bar{G}_l$

   $\partial O_l = \partial \Sigma_{G_l O_l} \cdot \nabla_{G_l * O_l} f_\mu * G_l + \partial \bar{O}_l \cdot \nabla_{O_l} f_\mu$
4. $\partial \bar{G}_l = -\partial b_l * A_l - \partial \Sigma_{G_l O_l} * \bar{O}_l - 2\partial \Sigma_{G_l} * \bar{G}_l$

   $\partial G_l = \partial \Sigma_{G_l O_l} \cdot \nabla_{G_l * O_l} f_\mu * O_l$
   $\qquad + \ 2\partial \Sigma_{G_l} \cdot \nabla_{G_l * G_l} f_\mu * G_l + \partial \bar{G}_l \cdot \nabla_{G_l} f_\mu$
5. $\partial I_l = \partial G_l \cdot \nabla_{I_l} F, \ \partial I_h = \partial O_h * A_h \cdot \nabla_{I_h} F$

3.2. Guided Filter Revisited

For joint upsampling, the guided filter takes a low-resolution image I_l, the corresponding high-resolution image I_h, and a low-resolution output O_l as inputs, producing the high-resolution output O_h. Concretely, A_l and b_l are first computed by minimizing the reconstruction error between I_l and O_l, which subjects to a local linear model [21] such that

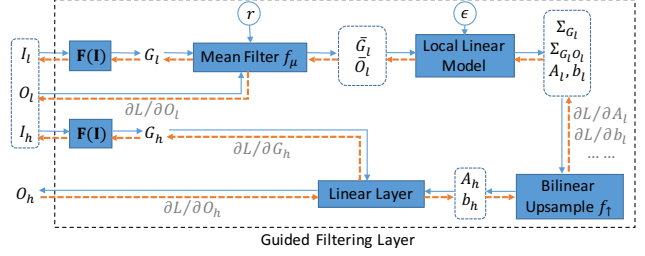$$O_l^i = a_l^k I_l^i + b_l^k, \forall i \in \omega_k, \tag{1}$$

Figure 2: **Computation Graph of Guided Filtering Layer.** Guided filtering layer takes low-resolution image I_l, high-resolution image I_h and low-resolution output O_l as inputs, generating the high-resolution output O_h. Compared to guided filter [21], the proposed layer is fully differentiable as shown by dotted lines. Besides, we also plug in $F(I)$ for learning task-specific guidance map.

where i is the index of a pixel, and k is the index of a local square window ω with radius r. A_h and b_h are then produced by upsampling A_l and b_l. The high-resolution output O_h is finally generated by a linear transformation [21], which is defined as

$$O_h = A_h * I_h + b_h, \tag{2}$$

where $*$ is element-wise multiplication.

3.3. Guided Filter as a Trainable Layer

The low-resolution output O_l is usually generated by a transformation function $f(x)$, which could be modeled with CNNs and learned with strong supervision signals. However, when we apply the guided filter as post-processing operation directly, the obtained results are not always pleasing, especially for tasks that are not scale-invariant. We attribute this to the lack of jointly training and direct supervision from high-resolution targets.

To take the benefits of end-to-end learning, we formulate guided filter as a fully differentiable layer, which could be trained with $f(x)$ jointly from scratch. The computation graph of guided filtering layer is shown in Figure 2. A_l and b_l are computed with mean filter f_μ and local linear model given I_l and O_l. A_h and b_h are then generated by applying bilinear upsampling operator $f_\uparrow$. O_h is finally generated with a linear layer taking A_h, b_h and I_h as inputs. r is the radius of f_μ and ϵ is the regularization term, which we set to 1 and $1e$-8 by default.

The equations for propagating the gradients through guided filtering layer are shown in Algorithm 1, while the corresponding forward algorithm is presented in the supplementary material. By modeling each operator into a differentiable function, the gradient of O_h could back-propagate to O_l, I_l as well as I_h through the computation graph, which

facilitates training $f(x)$ with direct guidance from the high-resolution targets. With the direct supervision signal, $f(x)$ learns to generate more suitable O_l for guided filtering layer to restore.

3.4. Learn to Generate Guidance Map

In Section 3.2 and 3.3, input images and outputs are assumed to have single channel. However, I_h, I_l, O_l and O_h often have more than one channel in real cases, noted n_I and n_O respectively. When $n_I \neq n_O$, we usually apply a transformation function $F(I)$ to I_h and I_l, generating guidance maps G_h and G_l with channel size equal to n_O. Even when $n_I = n_O$, a better guidance map than I_h and I_l is required. Traditionally, the transformation function is usually manually defined for different tasks, which takes lots efforts to design. By formulating the guided filter as a trainable layer, we could learn $F(I)$ by end-to-end training, and generate more suitable, task-specific guidance maps.

Concretely, we develop a fully-convolutional neural network serving as $F(I)$, which is composed of two convolution layers, between which are an adaptive normalization layer [10] and a leaky ReLU layer. We set the kernel size of both convolution layers to 1×1, and the channel size of the first convolution layer to 64 by default.

The complete computation graph is shown in Figure 2, which is both fully differentiable and could be adapted to specific tasks by jointly optimizing its parameters with the base neural network.

4. Deep Guided Filtering Network

We now integrate the proposed layer into CNNs following the coarse-to-fine manner, we name the whole network Deep Guided Filtering Network (DGF). The proposed framework aims for generating full-resolution, edge-preserving outputs with low computational cost, which would be widely applied to various dense prediction tasks, ranging from image processing to computer vision.

As shown in Figure 3, we first downsample the original input image I_h, obtaining the low-resolution image I_l. Then, a convolutional neural network $C_l(I_l)$ is applied, generating the corresponding low-resolution output O_l. The full-resolution output O_h is finally generated by the proposed guided filtering layer, taking I_l, I_h and O_l as inputs. The entire framework is end-to-end trainable, which could be learned from scratch.

To verify the effectiveness of the proposed guided filtering layer together with the framework DGF, we take a series of experiments on several dense prediction tasks, which could be divided into image processing tasks and computer vision tasks respectively. For image processing tasks, we take our experiments on L_0 smoothing [39], multiscale detail manipulation [13], photographic style transfer from a
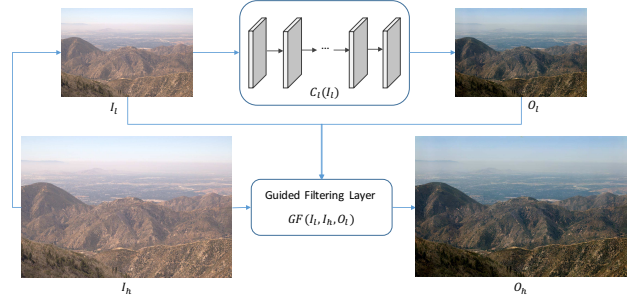

Figure 3: **Framework overview of deep guided filtering network.** Given an image I_h, we first produce I_l with downsampling operator. Then, we generate the corresponding low-resolution output O_l with a convolutional neural network $C_l(I_l)$. I_l, I_h and O_l are fed into the proposed guided filtering layer $GF(I_l, I_h, O_l)$ to generate the full-resolution output O_h finally. Best viewed in color.

reference image [3], non-local dehazing [5] and image retouching learning from human annotations [6]. For computer vision tasks, we select three tasks ranging from low-level vision to high level-vision, namely depth estimation from a single image [34], saliency object detection [32] and semantic segmentation [22]. The detail configurations for the two groups of tasks will be described in the following parts.

4.1. Low-resolution Networks

In our framework, $C_l(I_l)$ serves as the main component for learning to transform the input images I_l to outputs O_l. Most of the fully convolutional networks could be plugged into our framework to obtain huge acceleration while remaining similar quality performance. However, for a fair comparison, we use the networks from baseline methods as $C_l(I_l)$.

The architecture of $C_l(I_l)$ varies according to the characteristics of different tasks. Concretely, for image processing tasks, we deploy a unified network as $C_l(I_l)$ across different tasks. Because all the tasks share similar instincts despite the produced results are visually different. Particularly, we adopt the architecture of Context Aggregation Network (CAN) proposed by Chen *et al.* [10] as our default architecture. The main difference is that the resolution of input images I_l and outputs O_l is much smaller, which saves lots of computation resources.

For computer vision tasks, we use MonoDepth [19] for depth estimation, while using DSS [23] for saliency detection and Deeplab [9] for segmentation.

4.2. Integrate Guided Filtering Layer

We first present a preliminary framework, named DGF_s. In DGF_s, $C_l(I_l)$ is first trained to generate the low-

resolution output O_l, and then the full-resolution output O_h is produced with the guided filter as post-processing operation without any training. To benefit from end-to-end training, we plug the proposed guided filtering layer into $C_l(I_l)$, namely DGF_b. DGF_b learns to generate the high-resolution output O_l directly with the strong supervision signal from the high-resolution targets.

In both DGF_s and DGF_b, $F(I)$ is an identity function when the input images and outputs have the same number of channels. When they have different channel size, $F(I)$ serves as a transformer, turning I into a single channel grey image. To learn to generate task-orient guidance maps without manually designing $F(I)$, we further integrate the proposed $F(I)$ in Section 3.4, which composes into the final version of deep guided filtering network, *i.e.* DGF. DGF is not only end-to-end trainable but also fit different tasks better with the learnable $F(I)$.

4.3. Objective Function

Our model is trained end-to-end by the objective function defined at the full-resolution level. Given the full-resolution output O_h and the corresponding target T_h, the objective function is noted as $L(O_h, T_h)$. The concrete formulation varies with different tasks. For tasks in image processing, we use $l2$ loss, following the convention of previous works [10, 18]. As for tasks in computer vision, the loss proposed by the baseline model is deployed.

5. Applications and Experiments

5.1. Image Processing Tasks

Experimental Setup   We test the proposed guided filtering layer with our framework DGF on five different image processing operators: L_0 smoothing, non-local dehazing, multiscale detail manipulation, photographic style transfer and image retouching learning from human annotations. The details of each operator are described in the supplementary material.

Our experiments are taken on MIT-Adobe FiveK Dataset [6], which contains 5,000 high-resolution photographs covering a broad range of scenes, subjects, and lighting conditions. We use the default 2.5K/2.5K training/test split. To obtain the target images T_h for each task, we apply the corresponding operator on the entire dataset with the official implementation. For image retouching, we use the annotations from expert A as targets.

To learn a specific operator, we first train the network on the training set for 150 epochs, with the training data resized to 512**s**[1]. To improve the generalization ability of images across resolutions, we further train the network for 30 epochs, with training data randomly resized to a specific

---
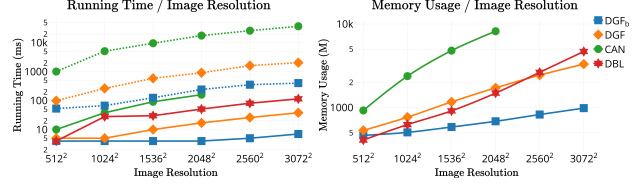[1]512**s** means the short side of an image is resized to 512 without changing the aspect ratio.

Figure 4: **Comparison on Speed and Memory Usage.** The performance on GPU devices is shown by solid lines, while dotted lines represent CPU devices.

resolution between 512**s** and 1672s. Despite the different resolution of I_h, we downsample it to 64**s** as I_l by default. We use Adam as our optimizer, with learning rate set to 0.0001 and batch size set to 1.

Our primary baseline is Deep Bilateral Learning (DBL) [18], which shares similar architecture to ours and achieves a good balance between quality and speed. Another strong baseline is CAN, which achieved the state-of-the-art performance while runs reasonably fast. CAN also serve as $C_l(I_l)$ in our framework. To ensure a fair comparison, we train the models using the official implementations and training procedures for both methods.

Running Time and Memory Usage   For image processing operators, it's crucial to run in real-time with limited memory usage. The running time and memory usage of our method together with the baseline methods are shown in Figure 4, which are measured on a workstation with Intel E5-2650 2.20GHz CPU and Nvidia Titan X (Pascal) GPU.

On GPU devices (shown by solid lines), DGF_b takes less than 10ms to process images with resolution ranging from 512^2 to 3072^2. The improved version DGF is slightly slower, but it still runs in real-time on images with resolution 3072^2. Compared to our method, CAN and DBL runs much slower across different resolutions. For images in 2048^2, DGF_b and DGF take 4ms and 17ms respectively, while CAN takes 160ms which are more than $40\times$ and $9\times$ slower than our methods. DBL takes 51ms to run, which is slightly faster than CAN but more than $10\times$ slower than DGF_b. The advantage of our method in speed is even more significant as the resolution grows. On CPU devices (shown by dotted lines), both DGF_b and DGF run more than $10\times$ faster than CAN.

The analysis running time is also reported. For I_h with $h \times w \times n_I$ and O_h with $h \times w \times n_O$. The running time complexity of DGF_b, DGF and DBL are $\mathcal{O}(n_O \times h \times w)$, $\mathcal{O}((n_I + n_O) \times h \times w)$ and $\mathcal{O}(n_I \times n_O \times h \times w)$ respectively.

Besides speed, our method also takes less memory space than both baseline methods on GPU devices. CAN is memory inefficient, which takes nearly 10G GPU memory space to process an image with resolution 2048^2. DGF takes a

Method	L_0 smoothing [39]			Detail Manipulation [13]			Style Transfer [3]			Non-local Dehazing [5]			Image Retouching [6]		
	MSE	PSNR	SSIM	MSE	PSNR	SSIM	MSE	PSNR	SSIM	MSE	PSNR	SSIM	MSE	PSNR	SSIM
Input	73	29.61	0.796	443	22.12	0.789	3534	13.28	0.521	2081	16.95	0.684	1507	18.44	0.727
CAN	**27**	**35.05**	**0.970**	**9**	**38.97**	**0.986**	519	21.31	**0.870**	355	24.47	0.862	964	20.43	0.744
DBL	39	32.35	0.896	75	29.84	0.924	354	23.32	0.834	502	23.27	0.852	1056	20.21	0.748
DJF	90	29.40	0.937	100	28.99	0.927	383	22.73	0.856	649	21.04	0.724	1216	18.89	0.702
DGF_s	36	32.90	0.914	94	29.03	0.903	340	23.12	0.733	355	24.17	0.862	961	20.54	0.753
DGF_b	34	33.05	0.911	79	29.86	0.905	375	22.65	0.737	**337**	**25.32**	0.883	**931**	**20.70**	**0.757**
DGF	34	33.13	0.917	71	30.78	0.931	**224**	**25.15**	0.867	338	25.09	**0.885**	949	20.46	0.744

Table 1: **Quantitative Comparison.** We evaluate our methods DGF_s, DGF_b and DGF as well as three baseline methods CAN, DBL and DJF on the test set of MIT-Adobe FiveK dataset. Five different operators are tested with three metrics, namely MSE, PSNR and SSIM. Our methods achieves the state-of-the-art performance on three tasks and obtain comparable results on the other two tasks.

similar amount of memory space to DBL but grows slower as the resolution increases. DGF_b is the most memory efficient method, which takes less than 1G memory even on images with resolution 3072^2.

Quantitative and Qualitative Comparison Our method achieves comparable or even better performance than the baselines, as shown in Table 1. To evaluate the performance, we run each method on the test set of MIT-Adobe FiveK dataset, with images re-scaled to 1024**s**. Mean Square Error (MSE, the lower, the better) serves as the evaluation metric. We also use PSNR and SSIM as our metrics (the higher, the better) because the alternative metric MSE is known to have limited correlation with perceptual image fidelity [10].

Our method achieves the state-of-the-art performance on style transfer, non-local dehazing and image retouching, while obtaining comparable results on L_0 smoothing and multi-scale detail manipulation. Our method improves over DBL across all the five tasks under all three metrics. For the style transfer task, DGF achieves 25.15 dB in PSNR, which improves over CAN by 3.84 dB and DBL by 1.83 dB.

However, our method and DBL are not performing as well as CAN on the tasks of L_0 smoothing and multi-scale detail manipulation. We attribute this to the nature of the methods following coarse-to-fine fashion, which could not perform well when the action of the operator at high-resolution cannot be recovered from the low-resolution output. This is one limitation of our method, and we would like to improve it in the future work.

The qualitative results are shown in Figure 6 as well as in the supplementary material.

The Role of Guided Filtering Layer To verify the effectiveness of the proposed guided filtering layer, we replace it with DJF [30], which is a complex CNN designed for joint image upsampling and achieves the state-of-the-art performance among many filters. The results in Table 1 show that our method outperforms DJF in all tasks. Our method also
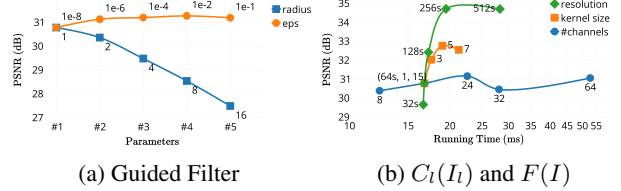

(a) Guided Filter

(b) $C_l(I_l)$ and $F(I)$

Figure 5: **Ablation study results.** a) reports the performance in PSNR with different r and ϵ. b) demonstrates the change of performance and running time with different input resolution for $C_l(I_l)$, different kernel size and channel size for the first layer of $F(I)$.

runs much faster, which takes $9\times$ less time than DJF on images with resolution 1024^2 (5ms v.s. 46ms).

The role of end-to-end learning is demonstrated by the performance of DGF_s and DGF_b in Table 1. By jointly training, DGF_b achieves better performance on most tasks. For non-local dehazing, DGF_b improves 1.15 dB in PSNR, while it improves 0.83 dB for detail manipulation.

By adding learnable $F(I)$, we could gain significant improvements in several tasks, especially in tasks that are resolution-dependent. Table 1 shows that DGF increases PSNR by 2.5 dB compared to DGF_b for style transfer. It also obtains 0.92 dB improvement on detail manipulation.

Ablation Study We take a series of experiments on detail manipulation to validate the function of hyper-parameters in the proposed guided filtering layer.

We first evaluate the role of radius r and regularization term ϵ. As shown in Figure 5a, the performance drops quickly as r grows, and our default setting ($r = 1$) obtains the best PSNR score. ϵ has limited effect on DGF. By increasing it from 1e-8 (default setting) to 1e-2, we could only improve PSNR by 0.49 dB.

The role of $C_l(I_l)$ are then explored by setting I_l to different resolution, ranging from 32**s** to 512**s**. As shown in

| (a) Input | (b) Ours (DGF) | (c) CAN | (d) DBL |

| (e) Ground Truth | (f) Error map of (b) | (g) Error map of (c) | (h) Error map of (d) |

Non-local Dehazing [5]

| (a) Input | (b) Ours (DGF) | (c) CAN | (d) DBL |

| (e) Ground Truth | (f) Error map of (b) | (g) Error map of (c) | (h) Error map of (d) |

Style Transfer [3]

| (a) Input | (b) Ours (DGF) | (c) CAN | (d) DBL |

| (e) Ground Truth | (f) Error map of (b) | (g) Error map of (c) | (h) Error map of (d) |

Multi-scale Detail Manipulation [13]

| (a) Input | (b) Ours (DGF) | (c) CAN | (d) DBL |

| (e) Ground Truth | (f) Error map of (b) | (g) Error map of (c) | (h) Error map of (d) |

L_0 Smoothing [39]

| (a) Input | (b) Ground Truth | (c) Ours (DGF) | (d) Baseline |

Saliency Object Detection [32]

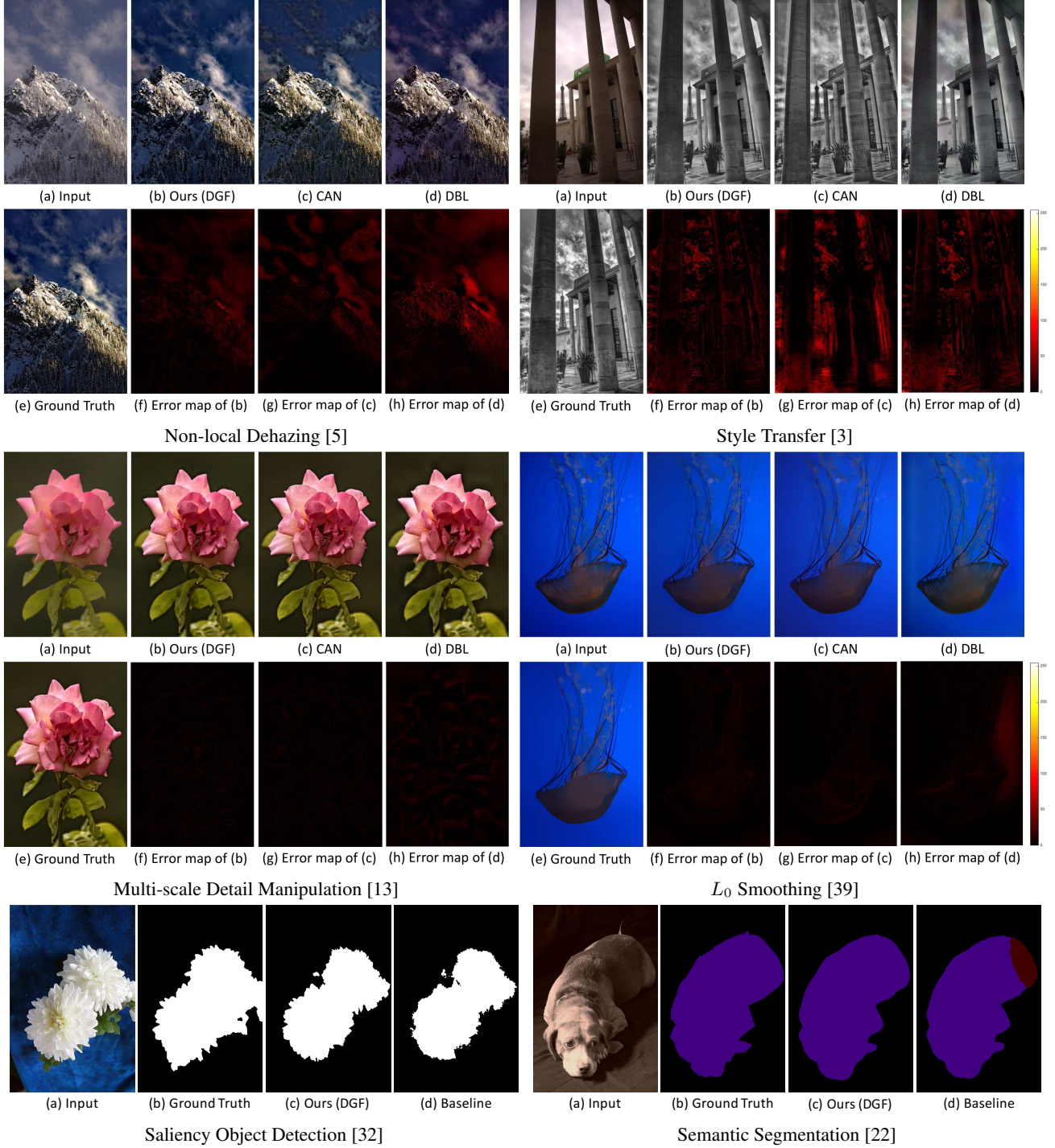| (a) Input | (b) Ground Truth | (c) Ours (DGF) | (d) Baseline |

Semantic Segmentation [22]

Figure 6: **Qualitative results of four tasks in image processing and two tasks in computer vision.** Best viewed in color.

Figure 5b, by changing the resolution of I_l from 64**s** to 256**s**, DGF improves PSNR by 3.93 dB with 2.7 ms more running time.

We also validate the function of $F(I)$ by varying the kernel size and channel size of its first convolution layer.

Figure 5b shows that channel size has the huge impact on running time, but no significant improvement could be obtained. However, by changing the kernel size from 1 (default setting) to 5, we could achieve better performance by 1.98 dB, with another 2.22 ms time cost, which also vali-

Resolution	768s	1024s	1280s	1536s	1792s	2048s
PSNR	29.11	30.78	30.25	30.51	30.65	30.55

Table 2: **Cross resolution generalization of DGF on detail manipulation.**

Method	Depth Prediction		Saliency	Segmentation
	rms	log10	F_β	Mean IOU
Baseline	6.081	0.216	90.61%	71.79%
DenseCRF	-	-	**91.87%**	72.69%
DGF$_s$	5.904	0.211	91.29%	71.72%
DGF	**5.887**	**0.209**	91.75%	**73.58%**

Table 3: **Quantitative Comparison.** Our method DGF achieves significant improvement compared to both baseline method and DGF_s across all there tasks, while it's comparable to DenseCRF.

dates the importance of $F(I)$.

Cross Resolution Generalization In the main experiment, we evaluate our method with 1024**s** images. We now test how DGF generalizes across different resolutions. We first resize the test images into different resolution, ranging from 768**s** to 2048**s**, and then generate the corresponding ground truth as described in the main experiment. The PSNR scores across resolutions tell the stability of our method, as shown in Table 2. Our method could perform equally well on 2048**s** images as the 1024**s** ones, despite the max resolution used in training is 1672**s**.

5.2. Computer Vision Tasks

Experiment Setup To demonstrate the generalization of our method, we apply DGF to three dense prediction tasks, ranging from low-level vision to high-level vision, namely depth estimation, saliency detection and segmentation.

For computer vision tasks, $C_l(I_l)$ takes I_h as inputs to generate O_l. The guided filtering layer is also adapted accordingly to work well.

KITTI [17], MSRA-B [25] and PASCAL VOC 2012 [12] are used for training and testing respectively. The details of the three tasks as well as the training and testing procedure will be discussed in the supplementary material.

Main Results We compare the performance between DGF$_s$, DGF and $C_l(I_l)$ (noted as Baseline), as shown in Table 3. For depth estimation, DGF$_s$ obtain 0.177 improvements in rms (the lower, the better) over the baseline. By end-to-end training and adding the learnable guidance map, we achieve 5.887 in rms, which improves 0.194 points. Similar results are also observed in saliency detection and semantic segmentation. F_β (higher is better) increases from

90.61% to 91.29% for saliency detection by applying the guided filter layer. By changing DGF$_s$ to DGF, F_β further improves to 91.75%. For segmentation, DGF obtains 73.58% in mean IOU (the higher, the better), which increases 1.79 points compared to the baseline method.

We also compare with DenseCRF [28], which is a commonly used filter for saliency detection and semantic segmentation. Experiments show that our method is comparable to DenseCRF in saliency detection, while obtains better performance in semantic segmentation. Besides, the proposed layer performs at least $10\times$ faster than DenseCRF. Averagely, our approach takes 34ms to process a 512^2 image, while Dense-CRF takes 432 ms on CPU.

Figure 6 visualizes the comparison results. The results obtained by our approach are better in preserving edges and details [2].

6. Conclusion

We present a novel deep learning building block, namely guided filtering layer. This new building block aims at efficiently generating high-resolution output with low-resolution one and a high-resolution guidance map. By formulating the guided filter into a fully differentiable module, deep-learning-based image processing and dense prediction approaches would benefit from end-to-end training and generate high-quality results. We further extend the proposed layer with a learnable transformation function, which makes it generalize well to different tasks by producing task-specific guidance maps. We integrate the guided filtering layer with CNNs and evaluate it on five image processing tasks and three computer vision tasks. Experiments show that the proposed layer could achieve the state-of-the-art performance while taking 10-$100\times$ less computationally time cost. We also conduct a comprehensive ablation study, which demonstrates the contribution of each component as well as each hyper-parameters. Although we observe the proposed approach obtains the significant improvements, it is noticeable that guided filter layer is not in an optimal architecture. We believe this is because the guided filter is a resolution-dependent operator. In the future, we plan to develop a better architecture to tackle this issue.

---

[2]More qualitative results are shown in the supplementary material.

References

[1] A. Adams, J. Baek, and A. Davis. Fast high-dimensional filtering using the permutohedral lattice. In *Eurographics*, 2009.

[2] A. Adams, N. Gelfand, J. Dolson, and M. Levoy. Gaussian kd-trees for fast high-dimensional filtering. *ACM TOG*, 28:1–21:12, 2009.

[3] M. Aubry, S. Paris, S. W. Hasinoff, J. Kautz, and F. Durand. Fast local laplacian filters: theory and applications. *ACM TOG*, 35:1–14, 2014.

[4] S. Bae and S. P. F. Durand. Two-scale tone management for photographic look. *ACM TOG*, 2006.

[5] D. Berman, T. Treibitz, and S. Avidan. Non-local image dehazing. In *CVPR*, 2016.

[6] V. Bychkovsky, S. Paris, E. Chan, and F. Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *CVPR*, 2011.

[7] J. Chen, A. Adams, N. Wadhwa, and S. W. Hasinoff. Bilateral guided upsampling. *ACM TOG*, 35(6):203, 2016.

[8] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015.

[9] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2016.

[10] Q. Chen, J. Xu, and V. Koltun. Fast image processing with fully-convolutional networks. In *ICCV*, 2017.

[11] Y. Cheng, R. Cai, Z. Li, X. Zhao, and K. Huang. Locality-sensitive deconvolution networks with gated fusion for rgb-d indoor semantic segmentation. In *CVPR*, 2017.

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.

[13] Z. Farbman, R. Fattal, and D. Lischinski. Edge-preserving decomposition for multi-scale tone and detail manipulation. *ACM TOG*, 27, 2008.

[14] R. Fattal. Single image dehazing. *ACM TOG*, 27, 2008.

[15] R. Fattal. Dehazing using color-lines. *ACM TOG*, 34, 2014.

[16] E. S. L. Gastal and M. M. Oliveira. Adaptive manifolds for real-time high-dimensional filtering. *ACM TOG*, 31(4):33, 2012.

[17] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[18] M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand. Deep bilateral learning for real-time image enhancement. *ACM TOG*, 36(4):118, 2017.

[19] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, 2017.

[20] K. He, J. Sun, and X. Tang. Single image haze removal using dark channel prior. *IEEE TPAMI*, 33, 2011.

[21] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE TPAMI*, 35(6):1397–1409, 2013.

[22] X. He, R. S. Zemel, and M. . Carreira-Perpinan. Multiscale conditional random fields for image labeling. In *CVPR*, 2004.

[23] Q. Hou, M.-M. Cheng, X. Hu, A. Borji, Z. Tu, and P. Torr. Deeply supervised salient object detection with short connections. In *CVPR*, 2017.

[24] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[25] H. Jiang, J. Wang, Z. Yuan, Y. Wu, N. Zheng, and S. Li. Salient object detection: A discriminative regional feature integration approach. In *CVPR*, 2013.

[26] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

[27] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele. Joint bilateral upsampling. *ACM TOG*, 26(3), 2007.

[28] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *NIPS*, 2011.

[29] G. Li and Y. Yu. Visual saliency based on multiscale deep features. In *CVPR*, 2015.

[30] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep joint image filtering. In *ECCV*, 2016.

[31] S. Liu, J. Pan, and M.-H. Yang. Learning recursive filters for low-level vision via a hybrid neural network. In *ECCV*, 2016.

[32] T. Liu, J. Sun, N.-N. Zheng, X. Tang, and H.-Y. Shum. Learning to detect a salient object. In *CVPR*, 2007.

[33] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[34] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *NIPS*, 2005.

[35] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *ECCV*, 2006.

[36] K. Subr, C. Soler, and F. Durand. Edge-preserving multiscale image decomposition based on local extrema. *ACM TOG*, 28, 2009.

[37] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV*, 1998.

[38] R. Vemulapalli, O. Tuzel, M.-Y. Liu, and R. Chellapa. Gaussian conditional random field network for semantic segmentation. In *CVPR*, 2016.

[39] L. Xu, C. Lu, Y. Xu, and J. Jia. Image smoothing via l0 gradient minimization. *ACM TOG*, 30, 2011.

[40] L. Xu, J. Ren, Q. Yan, R. Liao, and J. Jia. Deep edge-aware filters. In *ICML*, 2015.

[41] L. Xu, Q. Yan, Y. Xia, and J. Jia. Structure extraction from texture via relative total variation. *ACM TOG*, 31, 2012.

[42] Z. Yan, H. Zhang, B. Wang, S. Paris, and Y. Yu. Automatic photo adjustment using deep neural networks. *ACM TOG*, 35(2):11, 2016.

[43] Q. Zhang, X. Shen, L. Xu, and J. Jia. Rolling guidance filter. In *ECCV*, 2014.

[44] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015.