Non-blind Deblurring: Handling Kernel Uncertainty with CNNs Supplementary Material

Subeesh Vasu[1], Venkatesh Reddy Maligireddy[2], A. N. Rajagopalan[3]
Indian Institute of Technology Madras
subeeshvasu@gmail.com[1], venkateshmalgireddy@gmail.com[2] raju@ee.iitm.ac.in[3]

In this supplementary material, we begin by providing additional implementation details (Section 5) for training our network and run-time comparisons (Section 8) for all methods. This is followed by visualization of synthetic noisy kernels generated using our proposed scheme (Section 7.1), a discussion on noise-specific training experiments (Section 9.1) that we have conducted to assess the performance of the learned network with respect to variation in kernel noise level, and details on the network architectures (Section 4.2) that we experimented with (before converging to the one used in the main paper). This is followed by qualitative comparisons on our proposed improvisations (Section 9) for kernel noise reduction. Finally, we discuss the generalizability of our trained network followed by additional quantitative and qualitative comparisons.

S1. Implementation Details and Run-time

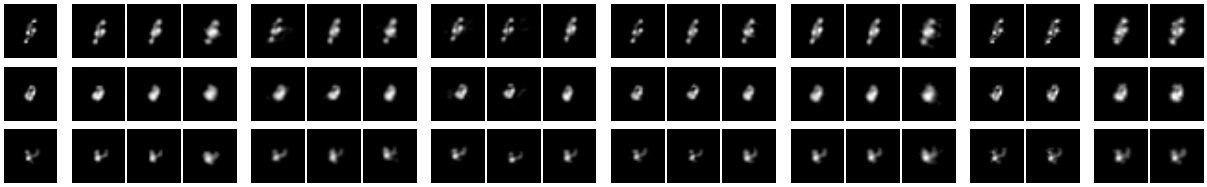
For training with real kernels, we use 400 clean images, and 1.6K real noisy kernels (formed from 4 GT kernels as discussed in Section 5) to generate 10K restored images. The set of kernels and images used for generation of test data is kept different from the ones used for generation of training data. For the generation of training as well as test data, we use ground truth kernels to form the corresponding blurred images, and the noisy kernel estimates to obtain image estimates. To accommodate the effects of image noise, while generating the training data we added AWG noise (by varying noise levels up to 2%) to the blurry images. To form the training data corresponding to synthetic noisy kernels, we used 10K blurry images formed using 10K GT kernels obtained from the synthetic kernel generation scheme in [1]. We have then used our proposed noisy kernel generation scheme to generate corresponding 10K synthetic noisy kernels and image estimates. To form training data corresponding to a mixture of synthetic and real kernels, we used 5K images each from both the real kernel based training data as well as synthetic kernel-based training data. This ensures that the performance variation across different categories is a true reflection of the differences in kernels used for training data. For all the cases, we used about 1.6 million patches for training. Patches close to the boundary were excluded since they can be information-deficient. Training of our network was done using gray scale images. To obtain results for color images, following others, we recombined the restored results of each channel. We trained our network on Nvidia Titan-X GPU using Torch. Table S1 lists the run-time. For the methods in [7, 11, 18, 25] we report the run-time using MATLAB on Intel core i5 CPU. For [9], we report the run-time on Nvidia Titan-X GPU for their TensorFlow implementation. For our proposed approach, run-time for the first NBD unit was computed on Intel core i5 CPU (for fairness in comparison with [7]) and the time for the CNN unit was computed using Nvidia Titan-X GPU. As is evident, the run-time for our approach is comparable to most of the works and is significantly less as compared to the state-of-the-art [25].

S2. Visualization of synthetic noisy kernels

As mentioned in our main paper, the noise behavior in kernel estimates primarily depends on the type of kernel prior employed. Typical kernel priors such as L_2 or L_1 norm on kernel intensity and/or gradients tends to deliver smoothly varying kernel estimate, while suppressing isolated noises which might get generated otherwise. Fig. S1 illustrates the closeness between some of the real kernels returned by blind-deblurring methods and our synthetically generated kernels corresponding to different parameter settings. From Figs. S1(a-f), we note that the noise pattern consists of smooth variations (rather than abrupt transitions), and is mainly distributed around the neighborhoods of the spatial locations where the GT kernel is non-zero. The estimated real kernels (Figs. S1(b-f)) appear to be low-pass filtered versions of the original kernel (Fig. S1(a)). As is evident from Figs. S1(g-h), the synthetic noisy kernels generated using our proposed scheme also mimic this behavior. To further verify the validity of synthetic noisy kernels we trained our network using "noisy kernels generated by directly adding AWGN to GT kernels", "kernels ob-

Table S1. Run-time (in seconds) for NBD methods

Image size	[7]	[11]	[5]	[3]	[6]	[18]	[9]	[25]	Ours
255 × 255	0.36	3.32	20.89	52.25	11.91	0.20	0.31	115.1	2.31
868 × 612	0.76	25.05	132.5	305.32	78.18	1.06	1.23	697.6	7.60



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

Figure S1. (a) Ground truth blur kernels from dataset in [13]. Real kernels recovered by BD methods in (b) [4] (c [2] (d) [13] (e) [19], and (f) [16]. Synthetically generated noisy kernels corresponding to (g) $\lambda_n = 6$, $v_g = 0.5$, and (h) $\lambda_n = 9$, $v_g = 0.7$.

tained using proposed scheme", and "real kernels" (the underlying GT kernels were kept the same in all three cases). The PSNR gain (as discussed in Section 7.1) obtained using these three different networks was found to be 0.19, 0.74, and 0.89, underscoring the merit of proposed kernel generation.

S3. Noise level-specific training

To analyze the impact of the level of kernel noise in the training data on network performance, we divided the kernels used for training data generation into three categories: low (N_1, average error-ratio < 1.2), high (N_2, $1.2 \leq$ average error-ratio ≤ 1.6), and mixed ($N_1 + N_2$) noise types. To divide the kernels into different categories we have used error-ratio computed in the following fashion. For each kernel estimate, we used a test image and corresponding GT kernels to generate the blurry images for all the kernels under consideration. These images are then deblurred using [7] (with $\lambda = 2e3$) and the noisy kernel estimates. We repeat this process for a number of test images and compute the average of the error ratio [12] for all the test images. Although the error-ratio gives a measure of kernel noise normalized according to the blur extent [12], we empirically observed that it still depends on the contents of the image. By using the same set of test images for all the kernels, we removed the dependency on image content which in turn yields reliable decisions on noise levels. We use multiple test images to remove the bias of the threshold on the contents of the test image. In our experiments, we used all the images from set14 dataset of [22] as test images. We treated kernels with error-ratio above 1.6 (emprically found) as outliers. We trained the network on the above-mentioned three different categories of noisy kernels. In a similar way to the generation of kernels for training we have also divided the testing kernels into three categories low (N_1'), high (N_2'), and mixed ($N_1' + N_2'$) noise types.

Fig. S2 illustrates performance differences of the network on low (N_1'), high (N_2'), and mixed ($N_1' + N_2'$) noise categories of the test data. As is evident, when we restrict the noise variations, performance for that specific noise level improves but suffers at other noise levels. The performance of the network trained with $N_1 + N_2$ yields the best results when we tested on the whole data ($N_1' + N_2'$) in a noise-level independent fashion. While this is true, if we have prior knowledge of the noise level in the kernel, we could selectively choose the best performing network for each scenario. We call this as noise-level-dependent testing and denote it by $N_1||N_2$. For such a case, we can further improve performance on the whole data ($N_1' + N_2'$) as shown in Fig. S2. Our noise-level-dependent testing is useful in scenarios where we can assess kernel noise level. Deciphering the noise level of estimated kernel is a research problem in itself. One way to exploit the advantage of noise-level-dependent testing is to restore images with network for different noise levels and choose the one that yields the best score based on some nonreference quality metric (such metrics can be found in [10]) as the optimal restoration result.

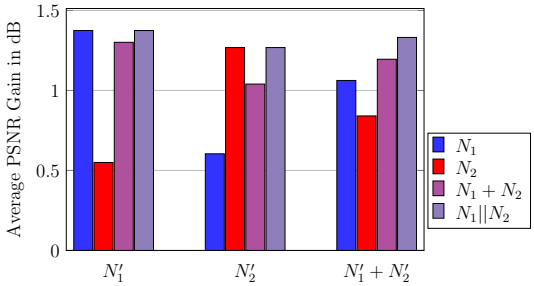


Figure S2. Network performance against noise level.

S4. Analyzing Performance of Different Network Structures

Before arriving at our final architecture, we had attempted different variations, details of which are provided here. The structure is described in the format 'number

of filters (filter size)' for each layer. In the following description, the layers inside curly braces represent feature extraction units for a single input, C refers to 64 filters, C_q refers to $64q$ filters, and A_n represent the n^{th} network structure.

A_1) 2 layers of feature extraction for each input (the network used in the main paper). Overall network structure: $3 \times \{C(3 \times 3) - C_2(3 \times 3)\} - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_2(3 \times 3) - C_1(3 \times 3) - 1(3 \times 3)$

A_2) 3 layers of feature extraction for each input: $3 \times \{C_1(3 \times 3) - C_2(3 \times 3) - C_2(3 \times 3)\} - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_2(3 \times 3) - C_1(3 \times 3) - 1(3 \times 3)$

A_3) 1 layer of feature extraction for each input: $3 \times \{C_2(3 \times 3)\} - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_2(3 \times 3) - C_1(3 \times 3) - 1(3 \times 3)$

A_4) No individual feature extraction unit: $C_3(3 \times 3) - C_6(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_2(3 \times 3) - C_1(3 \times 3) - 1(3 \times 3)$

A_5) 2 layers of feature extraction for each input, with larger filter size at input layer: $3 \times \{C_1(5 \times 5) - C_2(3 \times 3)\} - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_2(3 \times 3) - C_1(3 \times 3) - 1(3 \times 3)$

A_6) 2 layers of feature extraction for each input, with larger filter size at output layer: $3 \times \{C_1(3 \times 3) - C_2(3 \times 3)\} - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_2(3 \times 3) - C_1(3 \times 3) - 1(5 \times 5)$

A_7) 2 layers of feature extraction for each input, with one layer removed from the middle: $3 \times \{C_1(3 \times 3) - C_2(3 \times 3)\} - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_2(3 \times 3) - C_1(3 \times 3) - 1(3 \times 3)$

A_8) 2 layers of feature extraction for each input, with one layer added at the middle: $3 \times \{C_1(3 \times 3) - C_2(3 \times 3)\} - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_8(3 \times 3) - C_2(3 \times 3) - C_1(3 \times 3) - 1(3 \times 3)$

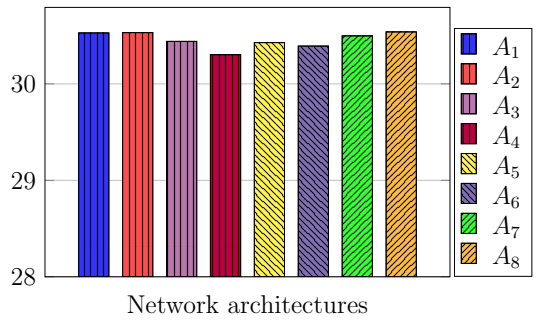


Figure S3. Performance of different network architectures.

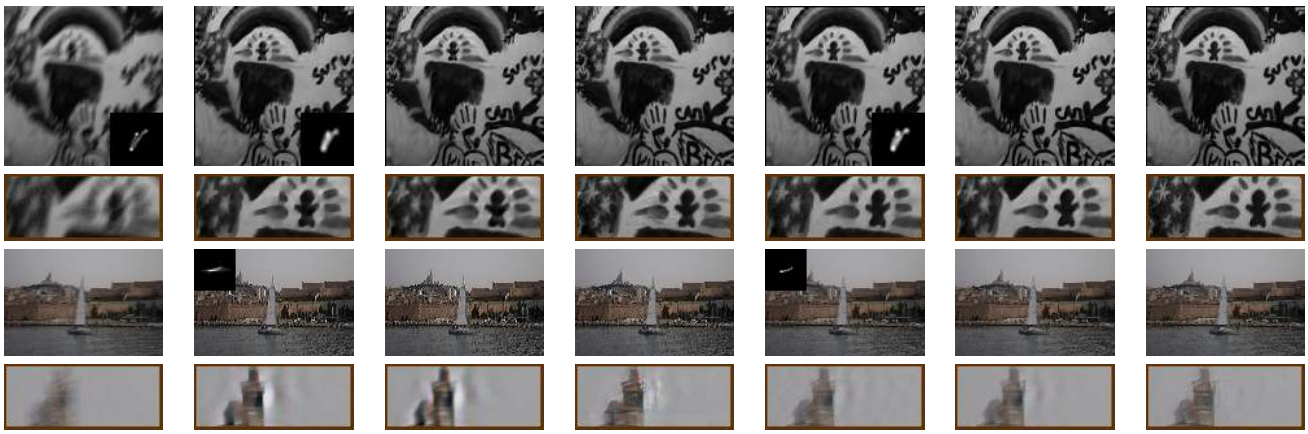
A network with no individual feature extraction unit (A_4) will discriminate the features from multiple inputs at the first layer itself. For networks with individual feature extraction units, the feature discrimination starts from the junction of all individual feature extraction units. Fig. S3 displays the PSNR values of test data for each of these networks. As can be observed, the inclusion of feature extraction units ($A_1 - A_3$) for each input resulted in improvement in PSNR. However, the performance improvement begins to narrow down when we use more than 2 feature extraction layers. To explore the scope for performance improvement by using filters of larger spatial extent, we increased the filter size at input and output layers, respectively ($A_5 - A_6$). However, for both cases, increasing the filter size leads to minor reduction in performance. The total number of layers used had a direct bearing on the performance of the network. To display this sensitivity, we compared the performance of two networks obtained by removing and adding one layer each, with respect to our final network ($A_7 - A_8$). As is evident, the performance is less for the network that has one layer less. Although there exists minor improvement in PSNR for network with more layers, we chose to proceed with the architecture proposed in the main paper as a judicious trade-off between computational complexity and performance. Scope exists to improve the performance with additional layers.

S5. Effect of loss function

To test the difference in performance with respect to loss function, we trained our final network with both L_2 and L_1 loss. While the results from both networks were visually comparable, the PSNR and SSIM values of the outputs were slightly better for the network trained with L_2 loss.

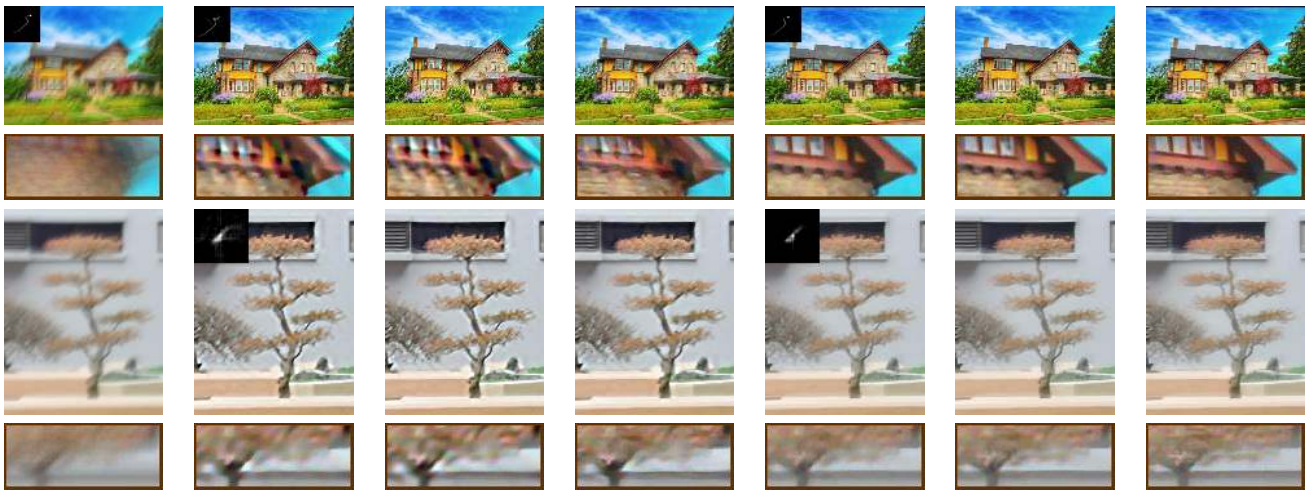
S6. Qualitative Comparisons on improvisations

Here we will provide qualitative comparisons on the two improvisations which we proposed in the main paper. Examples on the effect of kernel refinement is shown in Fig. S4. When the kernel noise is high, although our proposed method (Fig. S4 (d)) performs much better than the competing methods (Fig. S4 (b,c)), the restoration quality has scope for improvement. Using our iterative kernel refinement scheme, we could further reduce the noise level in the kernel (Fig. S4 (e)) which in turn leads to a high-quality restoration result. It is also evident from the final restoration results that our NBD method performs remarkably well (Fig. S4 (g)) post refinement of noisy kernel. Fig. S5 illustrates the impact of plugging-in our NBD module into [21]. While the BD method in [21] gets trapped in a local minimum (Fig. S5 (b)) from successive accumulation of residual artifacts, our proposed BD approach '[21] + $\text{CNN}_{2/3}$' (whose latent image estimation arm is driven by our NBD method) is successful in removing the artifacts thus leading to good quality kernel estimate (Fig. S5 (e)) as well as final restored image (Fig. S5 (g)). As is evident from Figs. S5 (b-d), our NBD approach gives rise to better restoration



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

Figure S4. Examples on kernel refinement through iterative restoration. 'Rows 1-2: Synthetic example from [13] (for a kernel estimate from [2]), Rows 3-4: Real example from [10] (for a kernel estimate from [15])'. (a) Blurred image. Restored images from initial kernel estimate using (b) [7] (and corresponding kernel estimate), (c) [25], and (d) proposed approach ($\text{CNN}_{2/3}$). Restored images from refined kernel (obtained via proposed iterative kernel refinement scheme) using (e) [7] (and corresponding kernel estimate), (f) [25], and (g) proposed NBD method ($\text{CNN}_{2/3}$).



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

Figure S5. Examples on adaptation to blind deblurring. 'Rows 1-2: Synthetic example, Rows 3-4: Real example' from [10]. (a) Blurred image. Restored images obtained by using kernel estimate from [21] and NBD method of (b) [7] (and corresponding kernel estimate), (c) [25], and (d) proposed approach ($\text{CNN}_{2/3}$). Restored images obtained by using kernel estimate from proposed BD method '[21] + $\text{CNN}_{2/3}$' and NBD method of (e) [7] (and corresponding kernel estimate), (f) [25], and (g) proposed NBD method ($\text{CNN}_{2/3}$).

results (as compared to other NBD methods) even for high noisy kernels, but the restoration quality is still limited by the noise level in the input kernel. However, the noise level reduction achieved through '[21] + $\text{CNN}_{2/3}$' leads to superior quality of restoration.

Table S2. Average PSNR on dataset of [13]

BD method $\rightarrow$	[4]	[2]	[13]
$\text{L}_1 \rightarrow \text{CNN}_{2/3}$	30.23	30.46	30.65
$\text{CNN}_{2/3}$	30.40	30.62	30.87
[25] $\rightarrow \text{CNN}_{2/3}$	30.69	30.83	31.02
average	28.74	29.36	29.35
median	28.91	29.34	29.47

S7. Generalizability to other NBD methods

Note that, our core idea is 'if regions with fewer artifacts and high-frequency details can be provided as inputs, then it is possible to train a CNN to yield high-quality restoration'. Since, generation of such inputs is possible with any prior-driven NBD, our approach can be expected to work well with initialization from other NBD methods too. However, we need to pick different λ values if we shift to a different NBD. To pick the λ values corresponding to a new NBD

Table S3. Performance comparison on dataset of [19] and [10]

NBD method	BD method for kernel estimation						
	PSNR/SSIM for images from [19]			SSIM/IFC for images from [10]			
	[2]	[20]	[14]	[20]	[21]	[19]	[17]
$L_1 \rightarrow CNN_{2/3}$	29.46/0.87	32.49/0.90	30.54/0.84	0.77/2.53	0.76/2.28	0.75/2.29	0.74/2.34
$CNN_{2/3}$	29.54/0.88	32.60/0.91	30.69/0.85	0.78/2.58	0.76/2.30	0.75/2.31	0.75/2.41
[25] $\rightarrow CNN_{2/3}$	29.61/0.88	32.75/0.91	30.81/0.85	0.79/2.62	0.77/2.36	0.75/2.34	0.75/2.43
average	28.99/0.84	30.72/0.86	29.58/0.82	0.74/2.46	0.72/2.23	0.72/2.14	0.71/2.19
median	28.61/0.83	30.46/0.86	29.35/0.81	0.73/2.42	0.73/2.25	0.71/2.11	0.71/2.17

Table S4. Average PSNR on dataset of [13], for GT kernels

NBD method $\rightarrow$	[5]	[6]	[3]	[9]	[11]	[7]	[18]	[25]	$CNN_{2/3}$
PSNR	32.95	33.98	33.92	35.02	33.77	33.93	33.41	34.69	35.26

method one can use the criteria mentioned in Section 6 (i.e., visual inspection and observation on average error, Fig. 1 (n)). Also, the behavior of L_2 prior is significantly different as compared to others, whereas $L_{2/3}$ and [25] have similar behavior. Since our approach already works well for two fundamentally different priors, it stands to reason that our approach should be applicable to other NBDs.

Now we will explore the possibility of using outputs of other NBD methods as inputs to the network trained with [7] (i.e., $CNN_{2/3}$). We used $CNN_{2/3}$ but initialized with outputs from [25] (we call it as [25] $\rightarrow CNN_{2/3}$) and L_1 norm based restoration ($L_1 \rightarrow CNN_{2/3}$). The optimal inputs used for [25] $\rightarrow CNN_{2/3}$ are λ_p = 64e3, 64e4, and 64e5 (refer Fig. 1), whereas we used λ_1 = 1e2, 1e3, and 1e4 for $L_1 \rightarrow CNN_{2/3}$. Qualitative performance comparisons for [25] $\rightarrow CNN_{2/3}$ and $L_1 \rightarrow CNN_{2/3}$ can be found in Figs. S12-S27 and Figs. S6-S11 respectively. Tables S2,S3 contains the reproduced values of comparisons in Tables 1,2 for [25] $\rightarrow CNN_{2/3}$ and $L_1 \rightarrow CNN_{2/3}$. As revealed by our quantitative and qualitative evaluations, initialization using [25] can indeed produce better restoration quality, whereas initialization using L_1 gives rise to results marginally inferior to $CNN_{2/3}$. This also indicates that the features learned by our network do not over-fit to [7] but instead help to distinguish between natural image contents and noise artifacts. Thus potential certainly exists to directly use $CNN_{2/3}$ in conjunction with other NBD methods without the need for retraining. At the same time, as indicated by the performance differences between $L_1 \rightarrow CNN_{2/3}$, $CNN_{2/3}$, and [25] $\rightarrow CNN_{2/3}$, the restoration performance of the network do depends on the quality of inputs, underscoring the fact that there exists scope to further improve the performance by adopting NBD approaches with better priors.

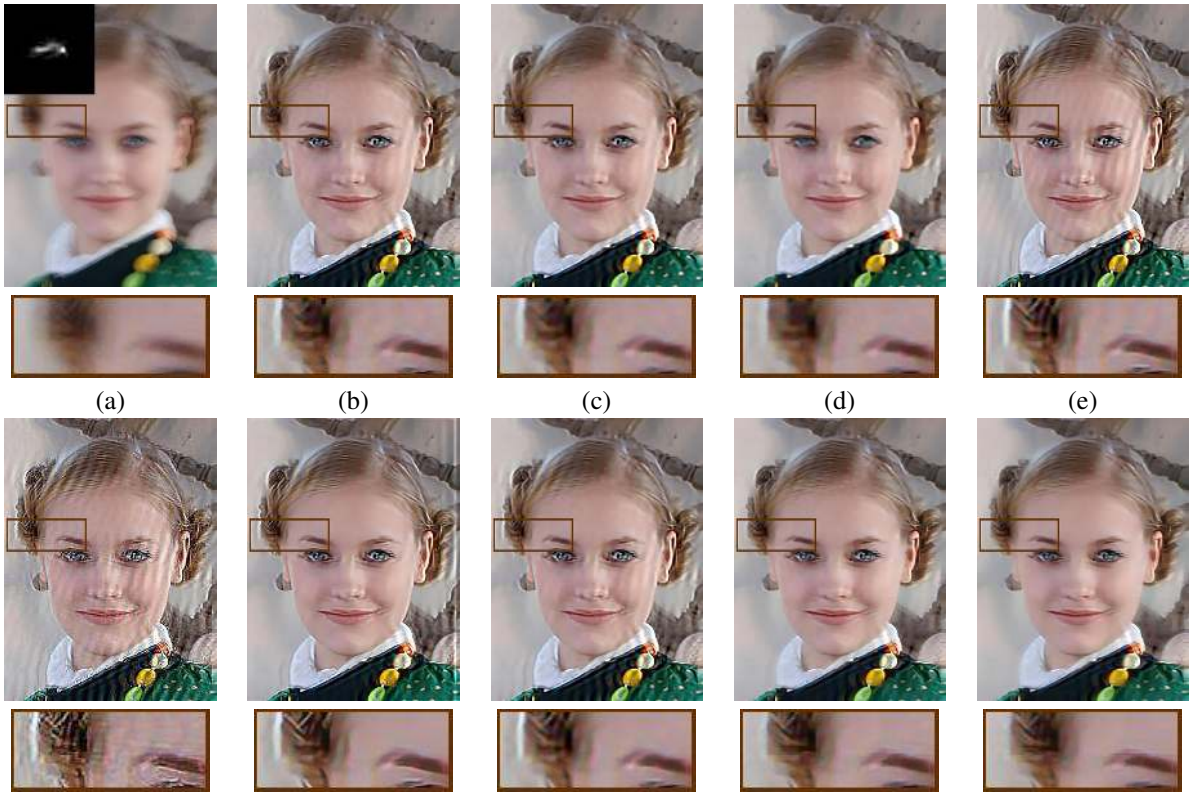
S8. Additional quantitative comparisons

With GT kernels - To analyze the performance when the kernel is quite accurate to begin with, we compare the PSNR values for restored images (using GT kernels) on the dataset of [13]. As revealed in Table S4, our approach is able to maintain the performance improvement even when GT kernel is available.

Simple filtering operations on the network inputs - To give more intuition on the difficulty in extracting a sharp image from the 3 network inputs, we have attempted to restore the images by applying simple schemes such as average and median filtering. The outputs of average and median filtering can be served as the baselines and can therefore help us to understand the restoration quality that can be achieved by such naive approaches. Tables S2,S3 contains the reproduced values of comparisons in Tables 1,2 for the outputs of average and median filtering of network inputs. As is evident, such naive schemes do not lead to satisfactory performance improvement.

S9. Additional qualitative comparisons

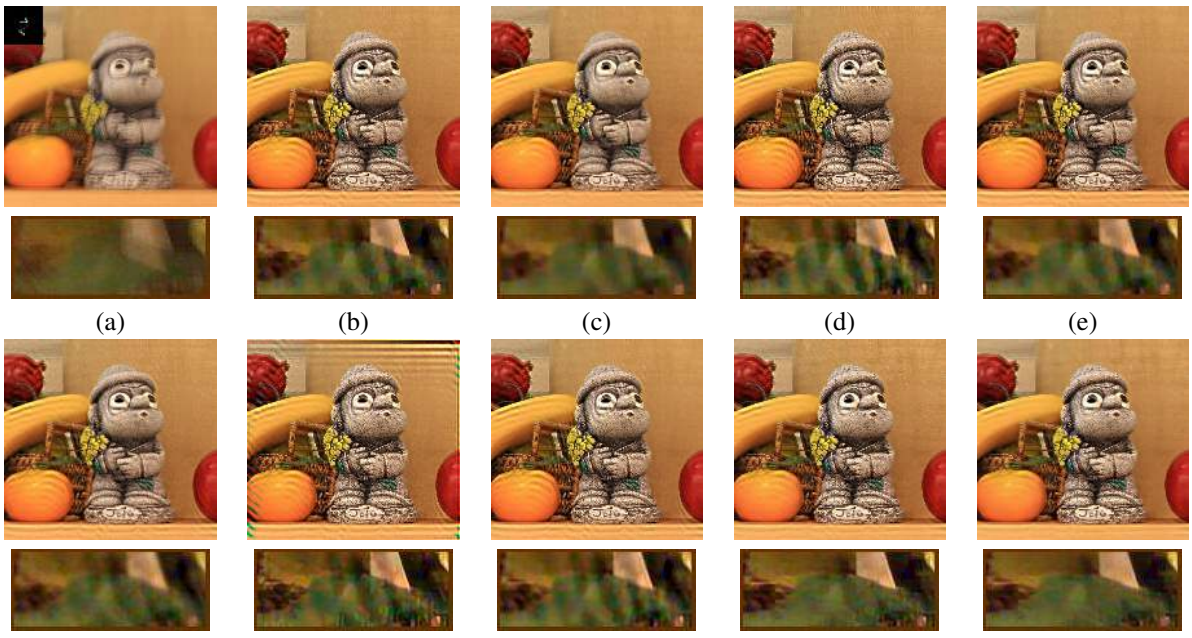
In this section, we provide visual comparisons for a variety of examples from the datasets in [12, 19, 10]. The results shown here include synthetic and real images of a wide range of scenes, and kernel estimates from various blind deblurring methods. The qualitative comparison that we present here is mainly to illustrate the wide-scope of our proposed approach for NBD ($CNN_{2/3}$) in handling different kinds of scene content, kernel noise, deconvolution artifacts etc. We compare the results of the proposed approach with all the inputs of our network to visualize the variations in the types of image artifacts that we can handle. Visual comparisons with respect to blurry image and kernel estimates are provided to have an understanding of the level of blur, and the nature of kernel noise that our approach can handle, respectively. We also provide comparisons with respect to ground truth image to illustrate the closeness of our result (to ground truth) and the efficacy in preserving details. In all the examples, the enhanced detail preservation that can be achieved through our approach as compared to other non-blind deblurring methods is clearly evident. We also provide some very challenging scenarios (Figs. S48, S49, S50, S51) for which the estimated kernels are significantly noisy. Although we are unable to remove the artifacts completely for such cases, the results of our approach are significantly better than those of competing methods.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

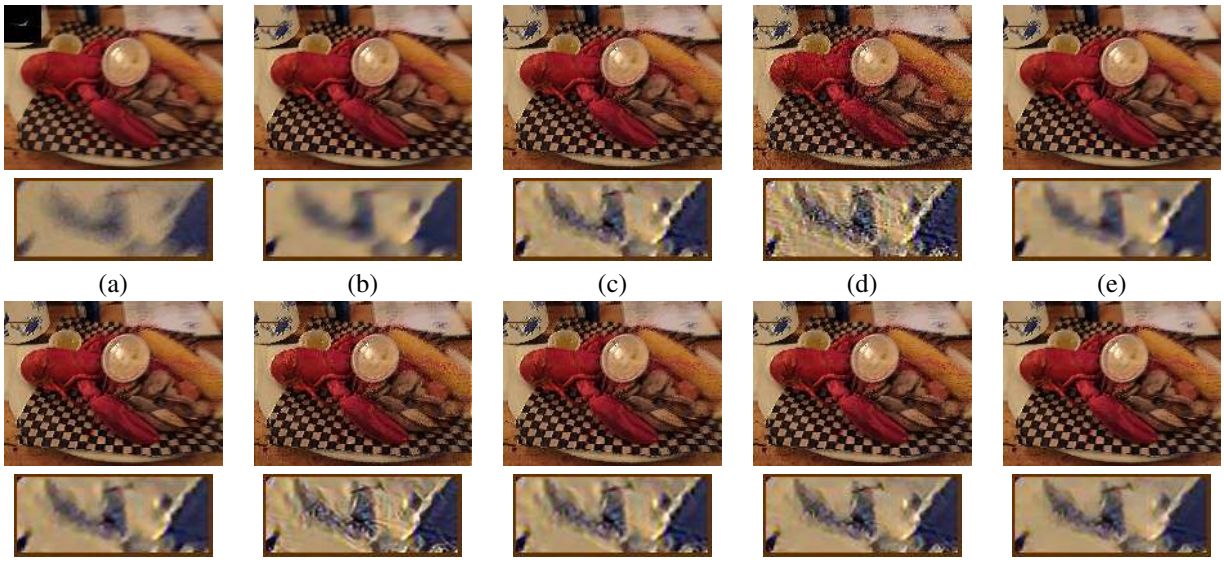
Figure S6. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [15]. (a) Input blurred image and kernel estimate. Restored image using (b) [7], (c) [11], (d) [5], (e) [3], (f) [6], (g) [9], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) $\text{L}_1 \rightarrow \text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

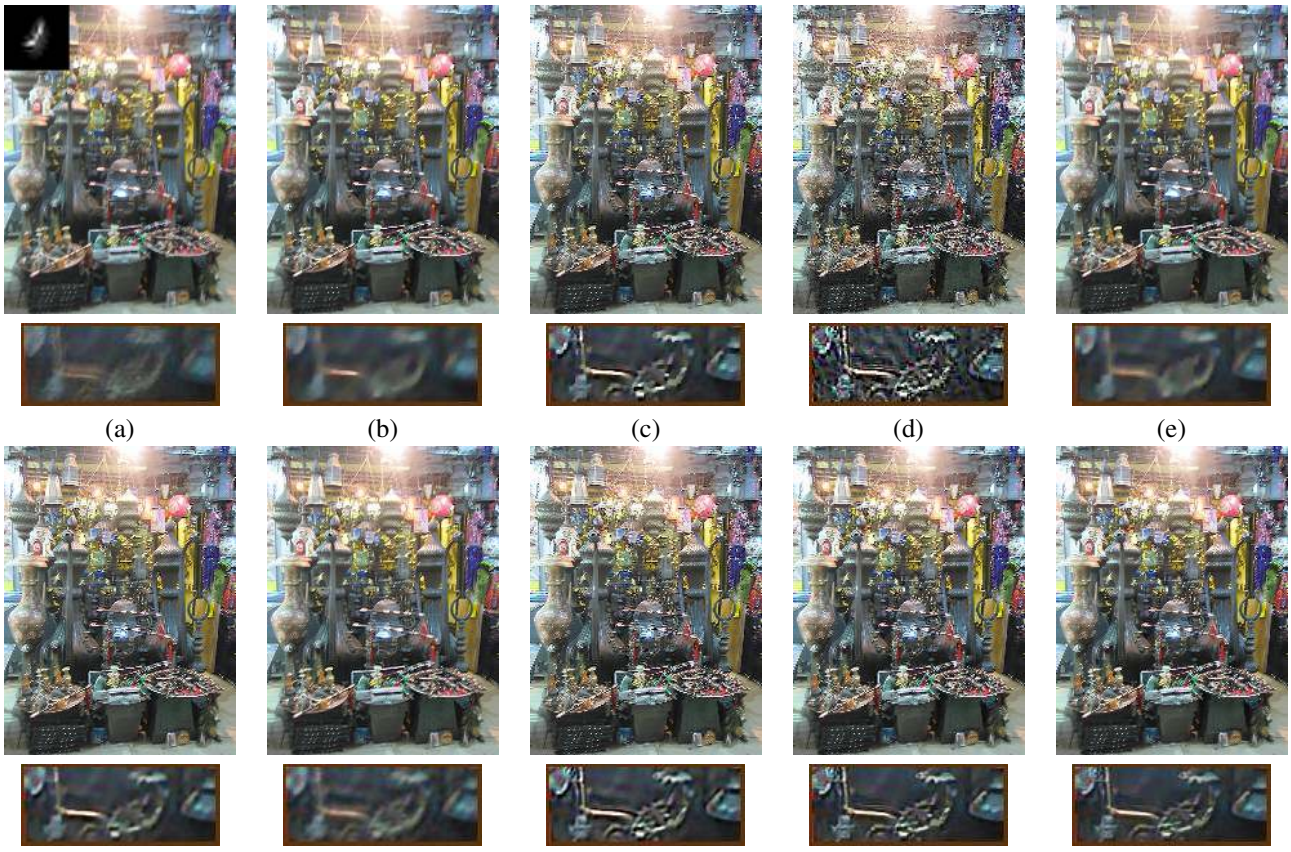
Figure S7. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [2]. (a) Input blurred image and kernel estimate. Restored image using (b) [7], (c) [5], (d) [3], (e) [11], (f) [18], (g) [9], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) $\text{L}_1 \rightarrow \text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

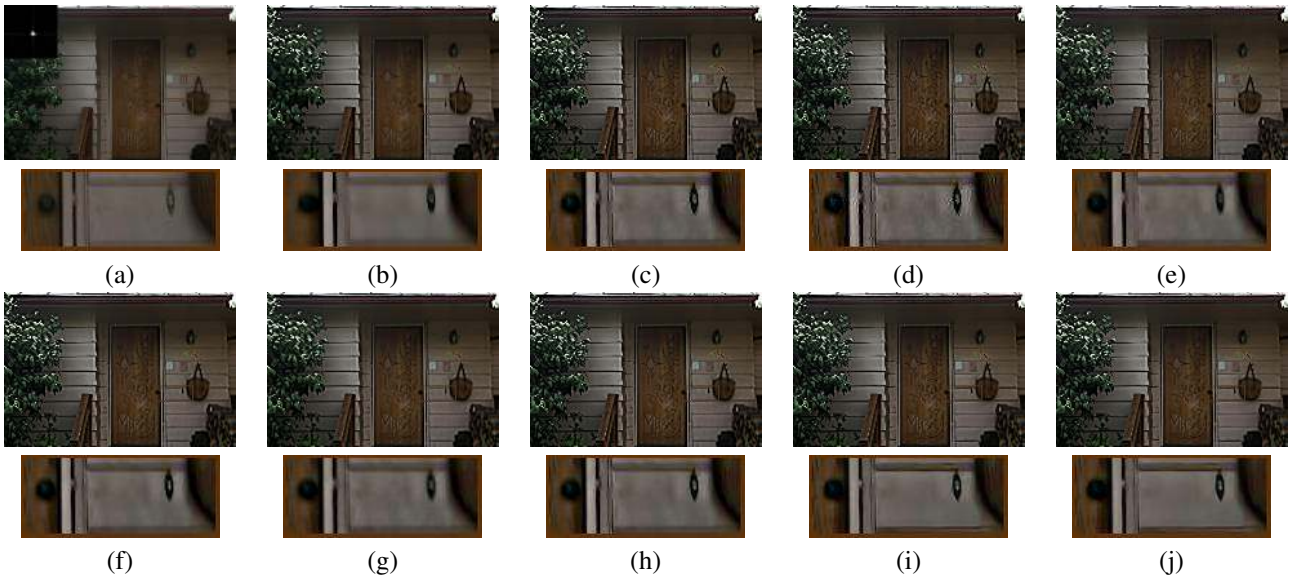
Figure S8. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [15]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5] (f) [11], (g) [9], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) $\text{L}_1 \rightarrow \text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

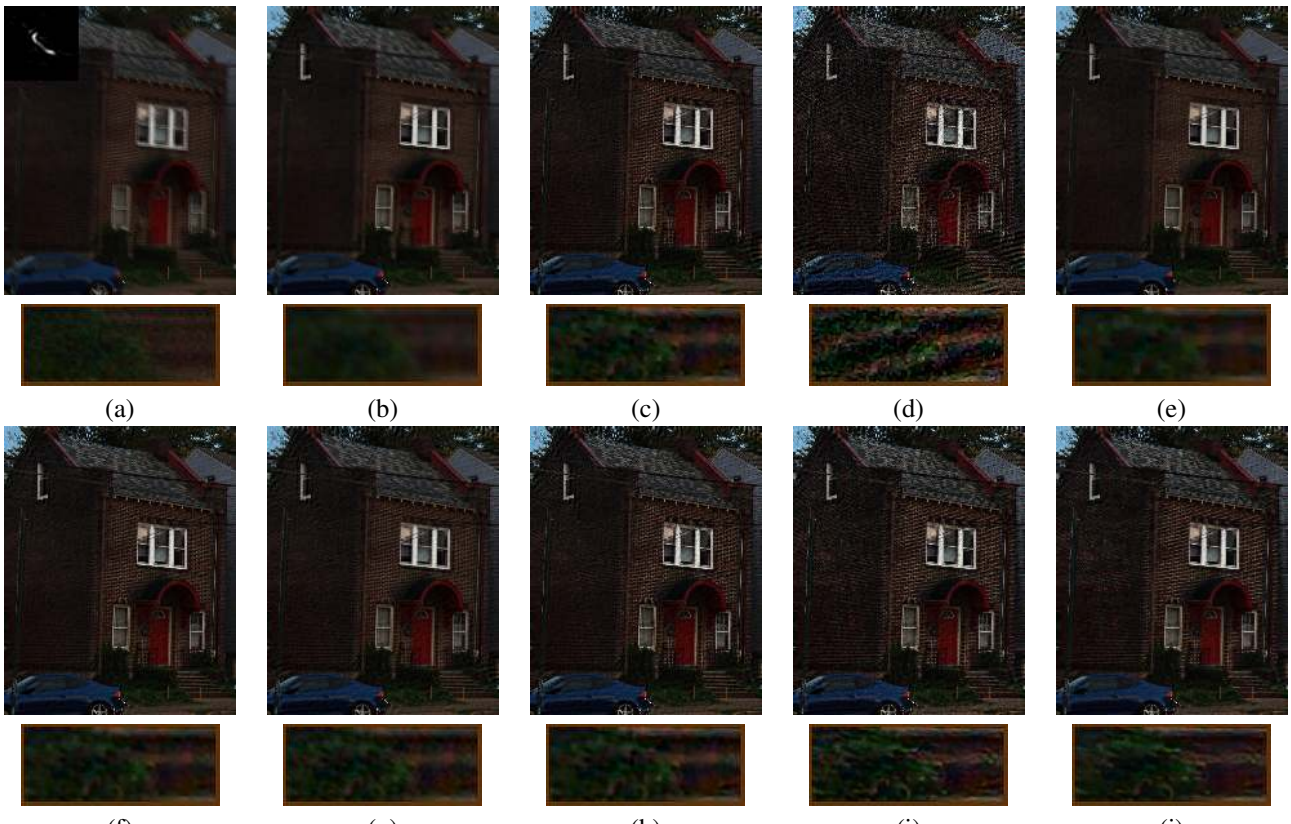
Figure S9. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [15]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5] (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) $\text{L}_1 \rightarrow \text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

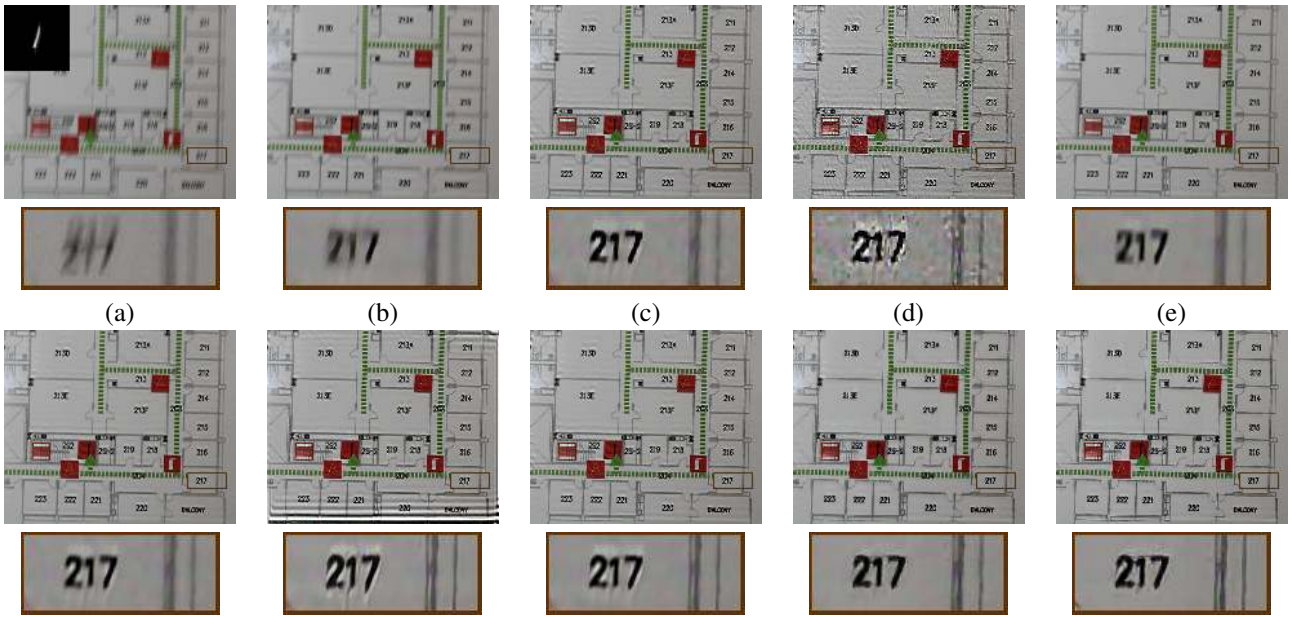
Figure S10. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [17]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5] (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) $\text{L}_1 \rightarrow \text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

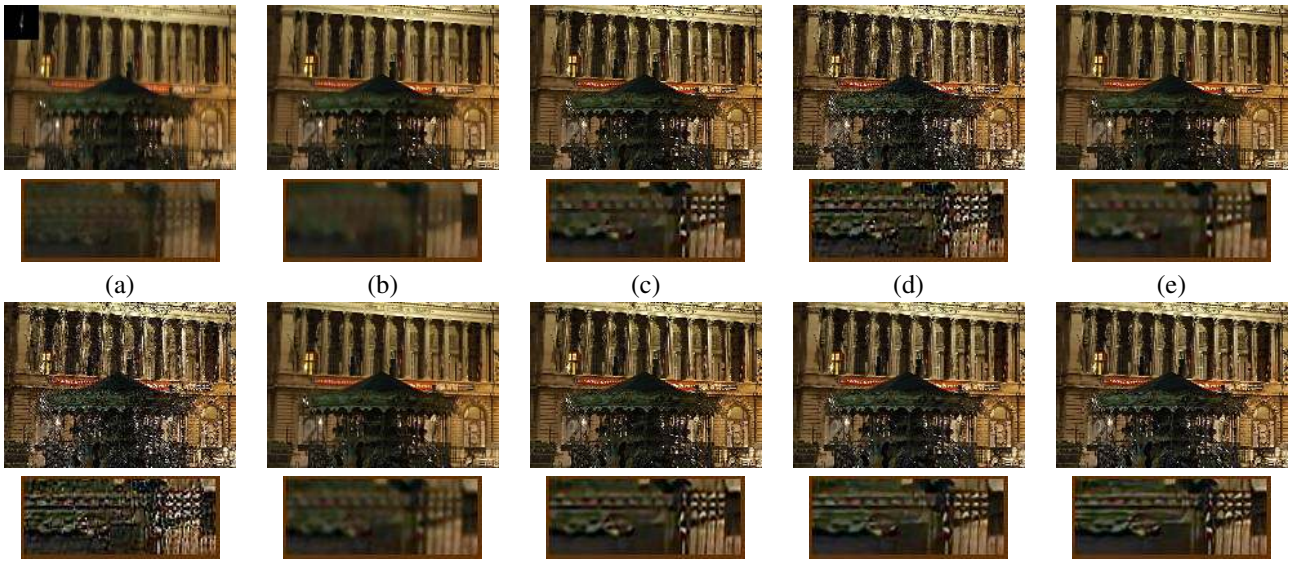
Figure S11. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [20]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5] (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) $\text{L}_1 \rightarrow \text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure S12. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [15]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [9], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

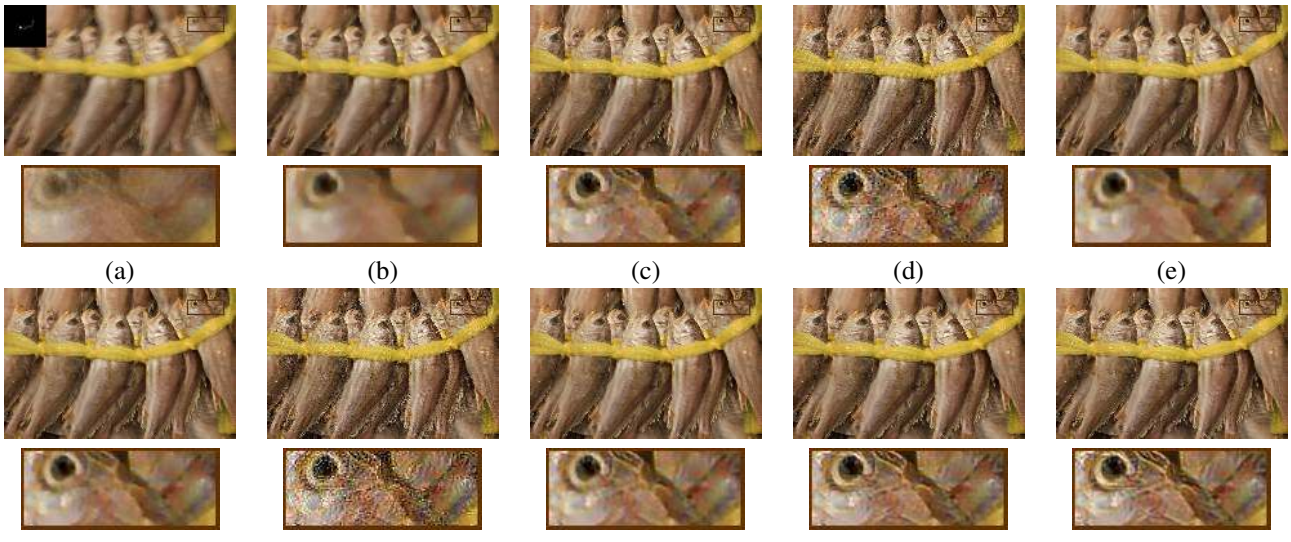
(f) (g) (h) (i) (j)

Figure S13. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [15]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [11], (f) [6], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.

(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

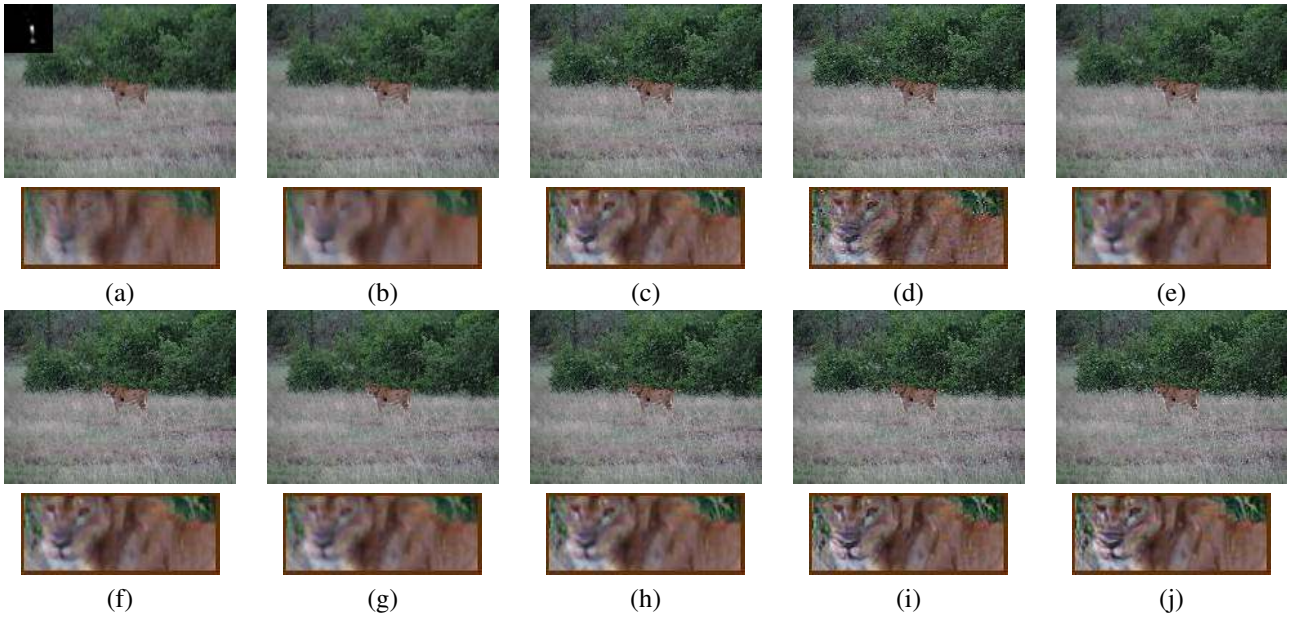
Figure S14. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [20]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

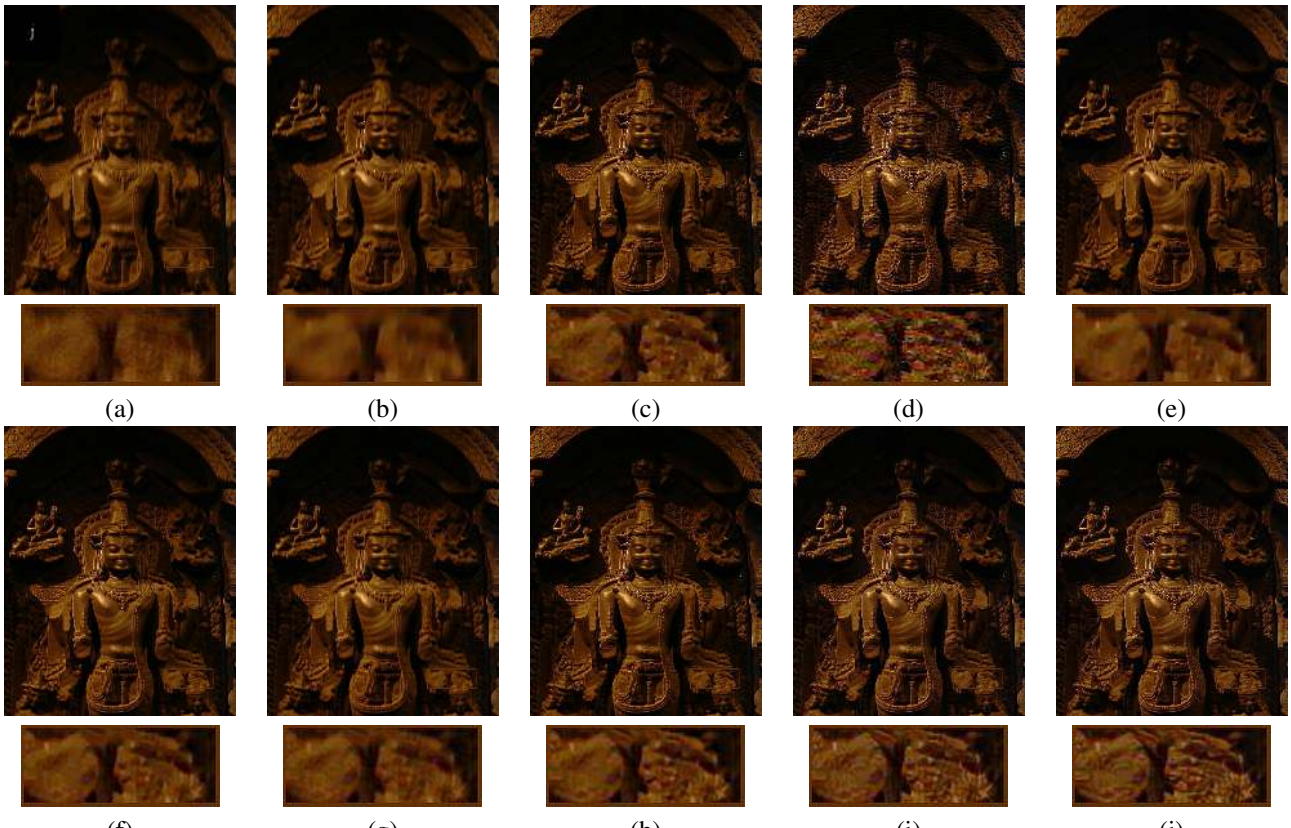
Figure S15. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [2]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [6], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

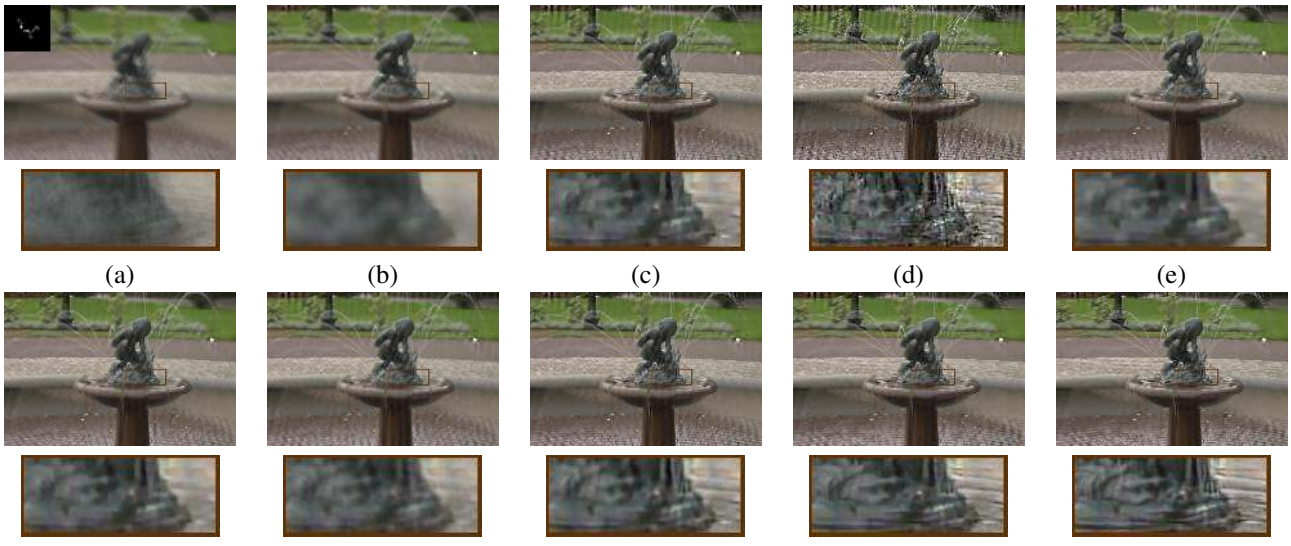
Figure S16. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [15]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

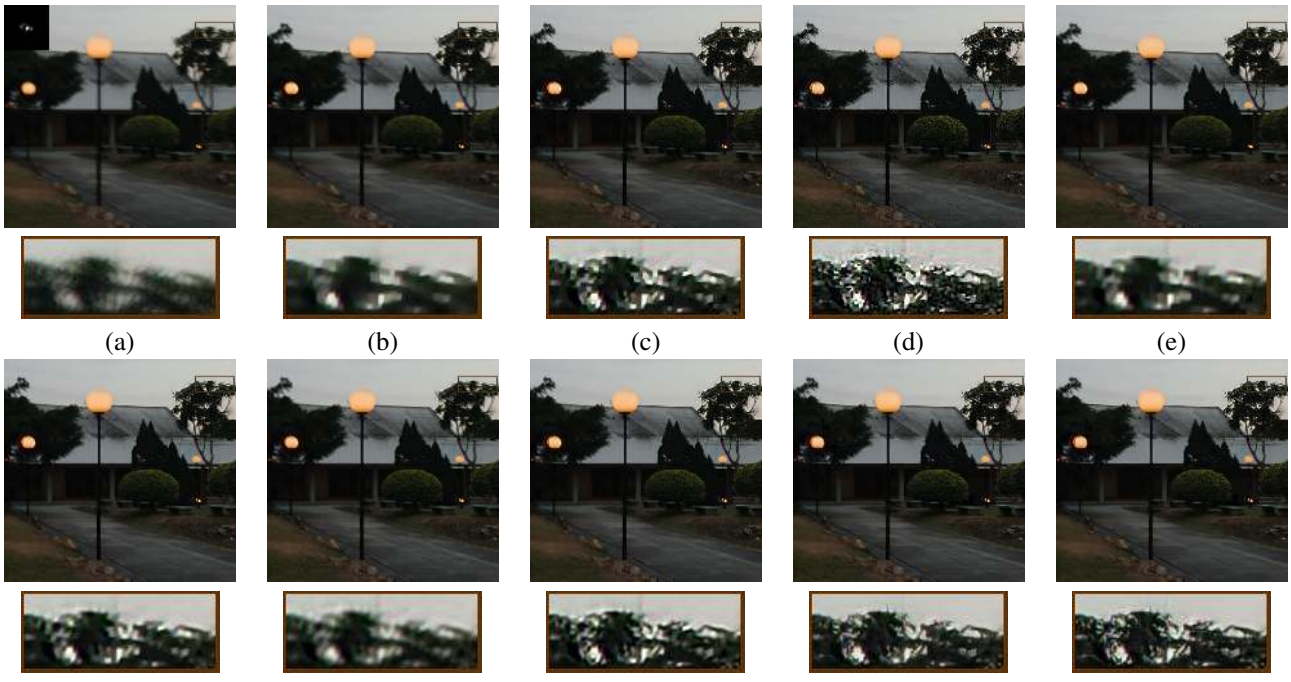
Figure S17. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [21]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

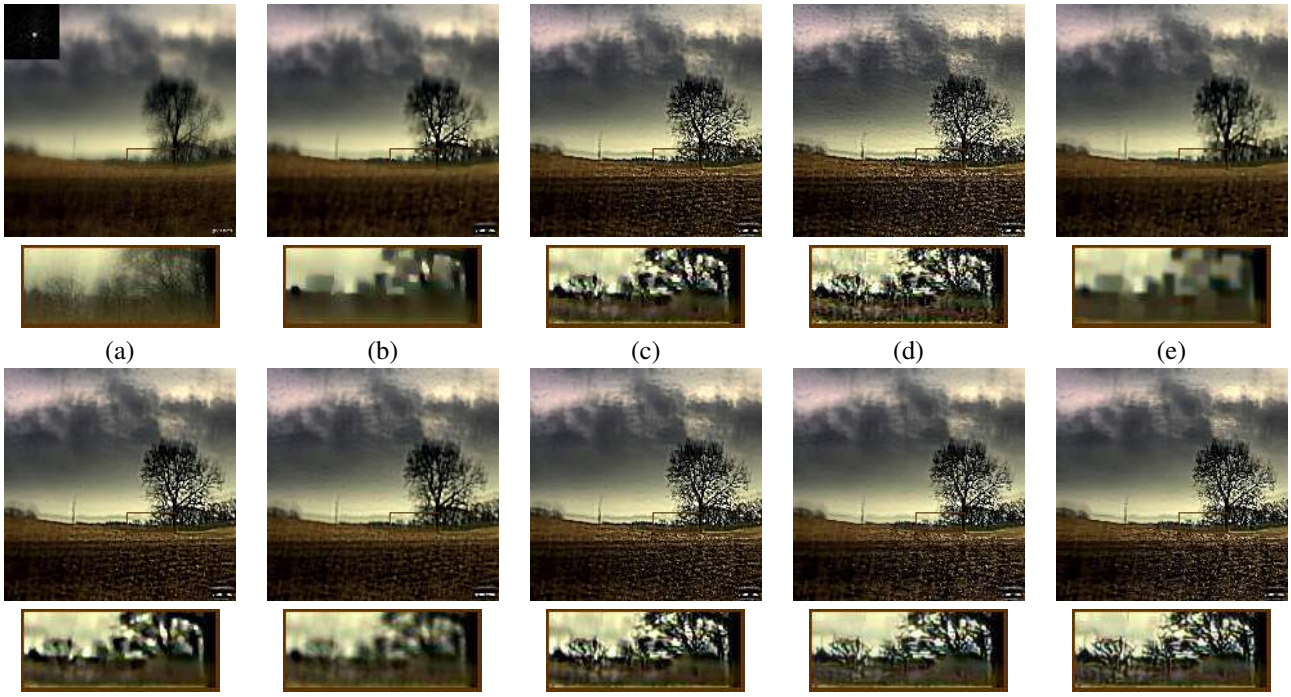
Figure S18. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [19]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$ and (c) $\lambda = 2e3$. Results from (d) [6], (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

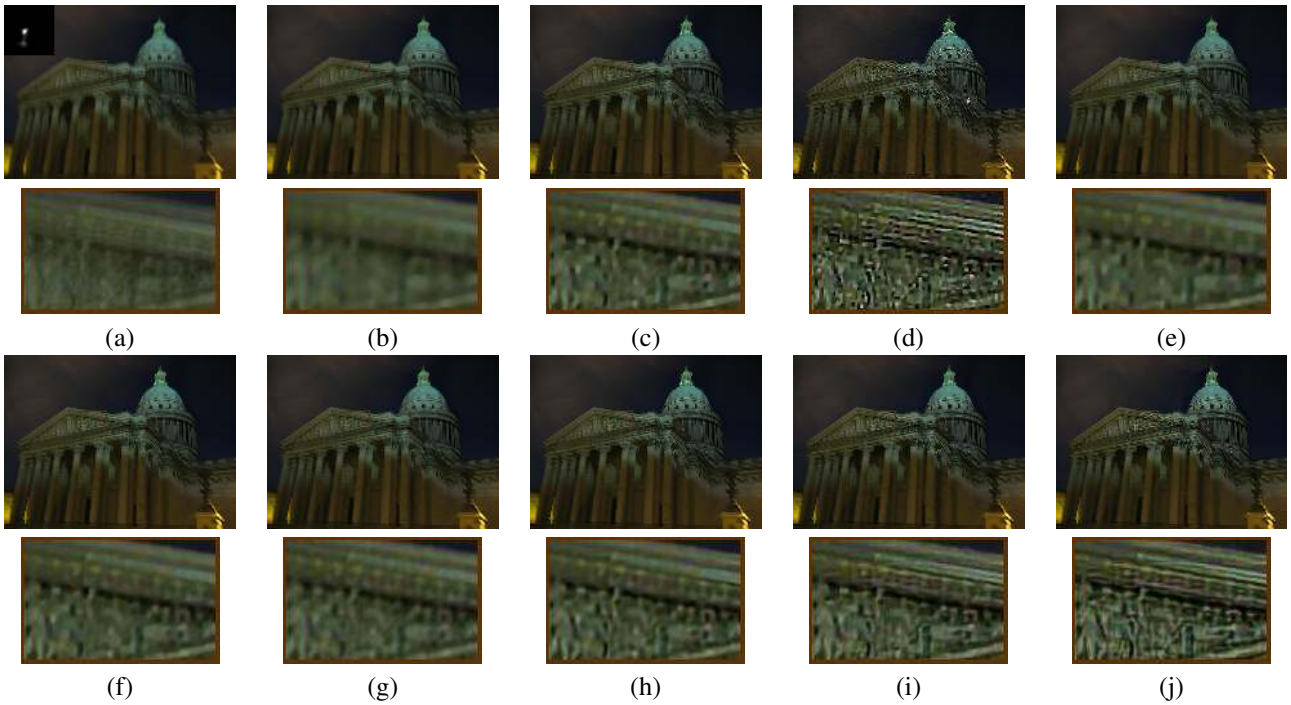
Figure S19. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [20]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

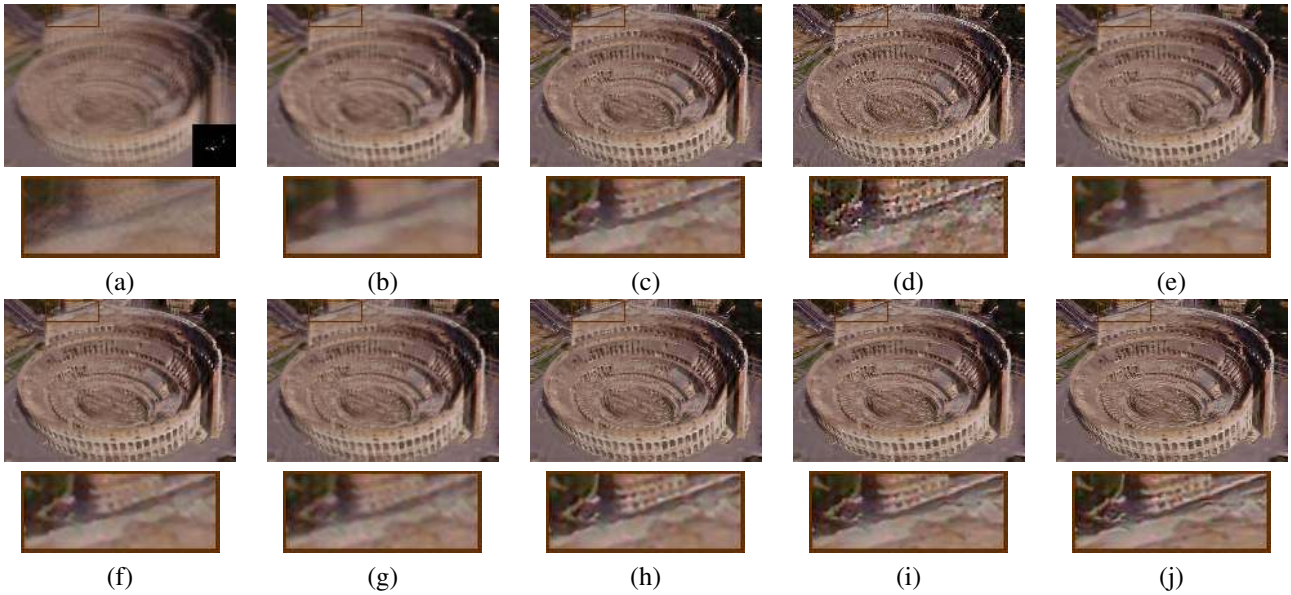
Figure S20. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [23]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

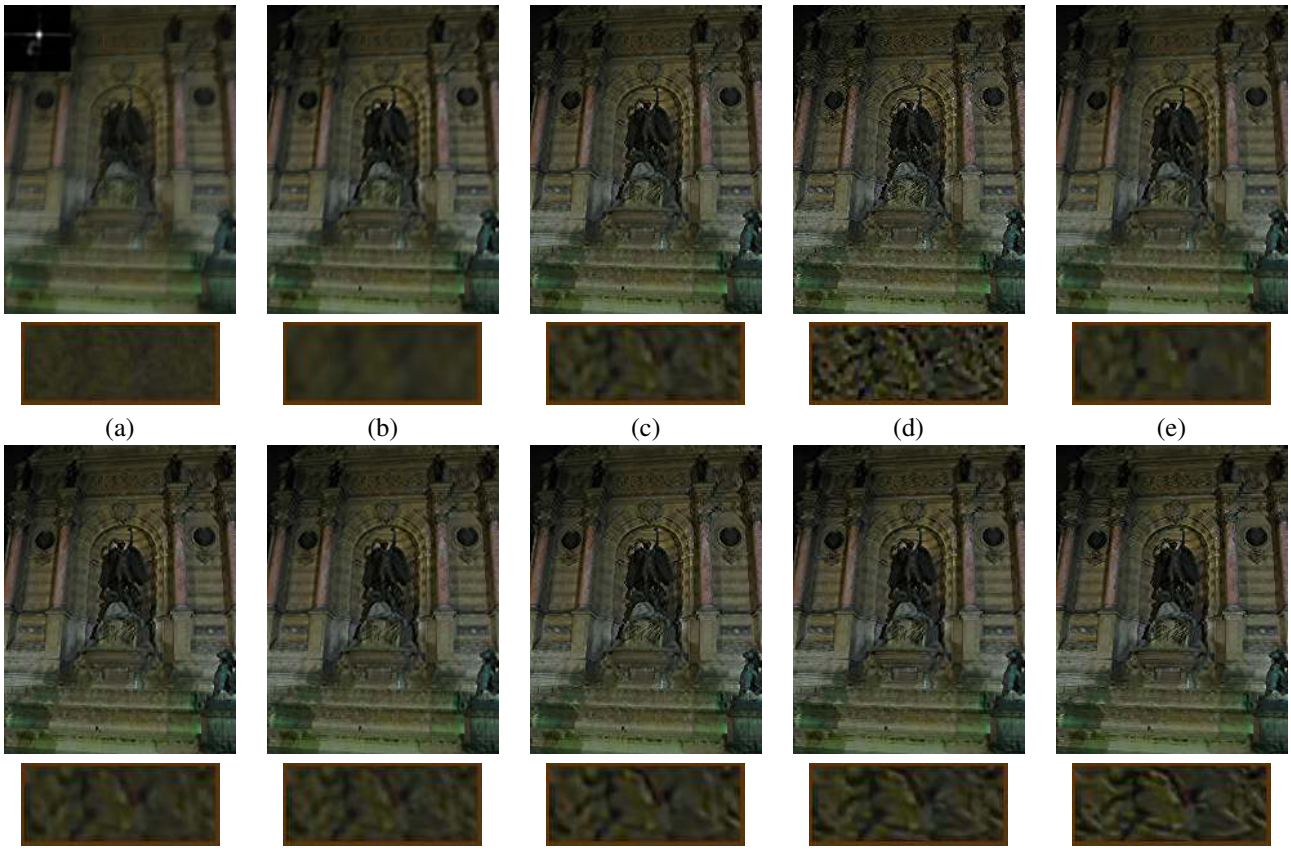
Figure S21. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [24]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure S22. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [21]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure S23. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [8]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.

Figure S24. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [13]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.

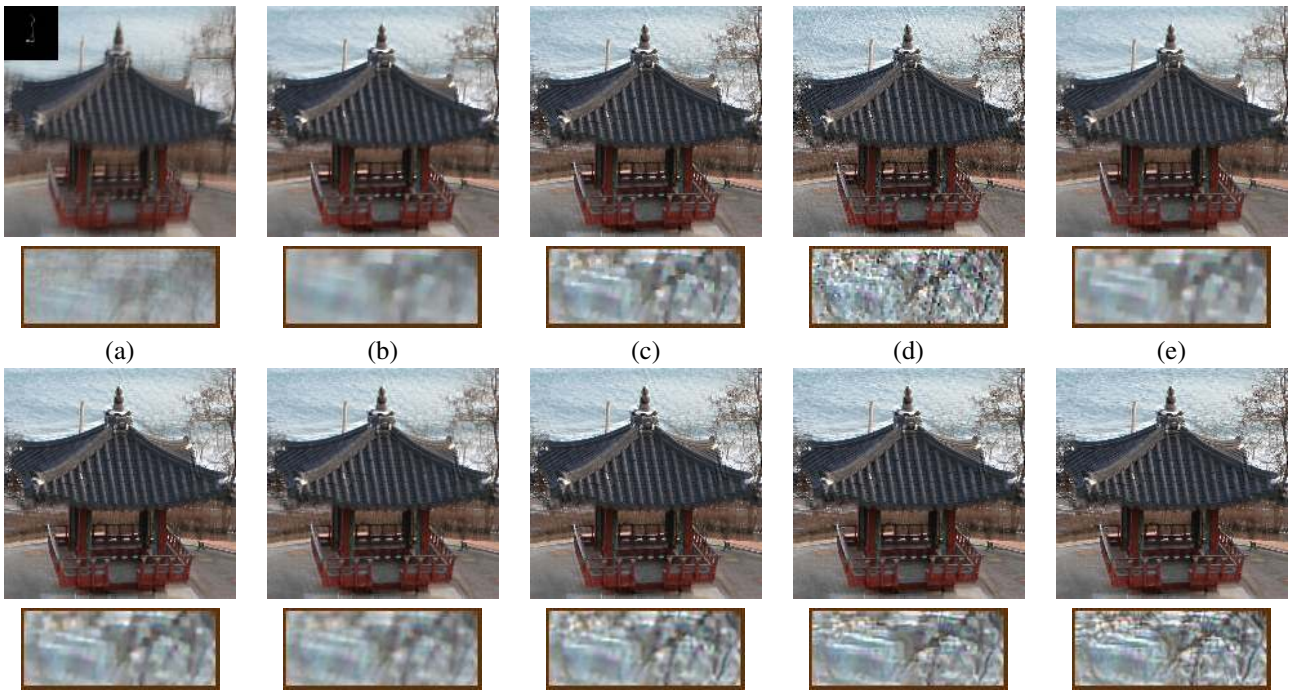
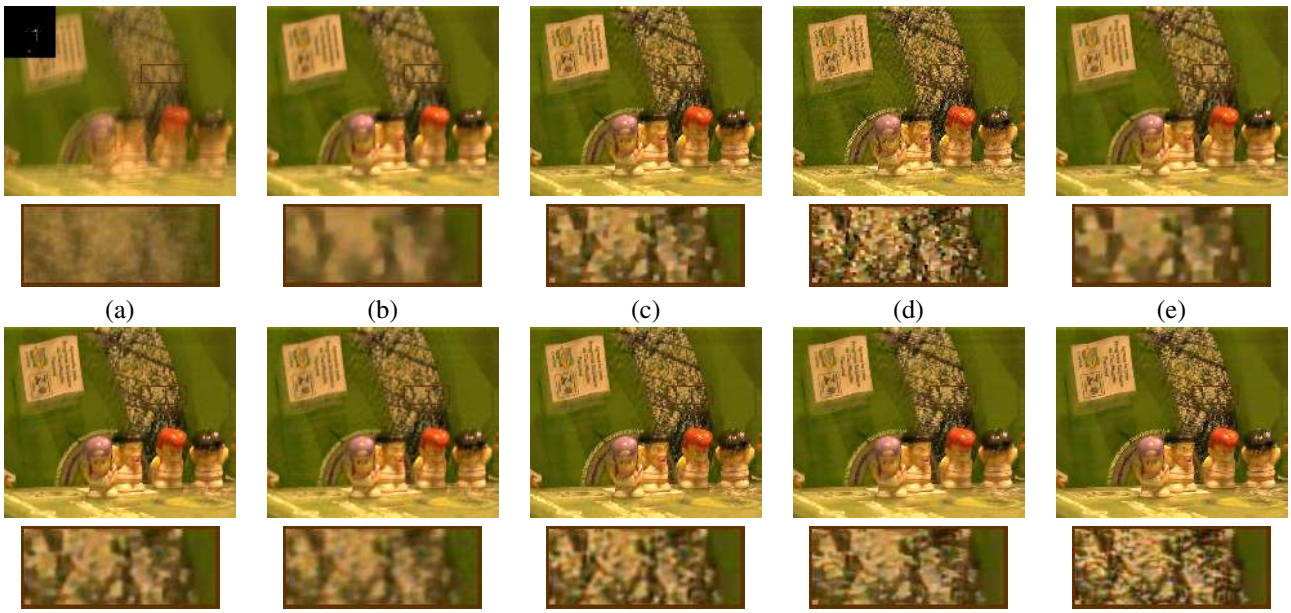


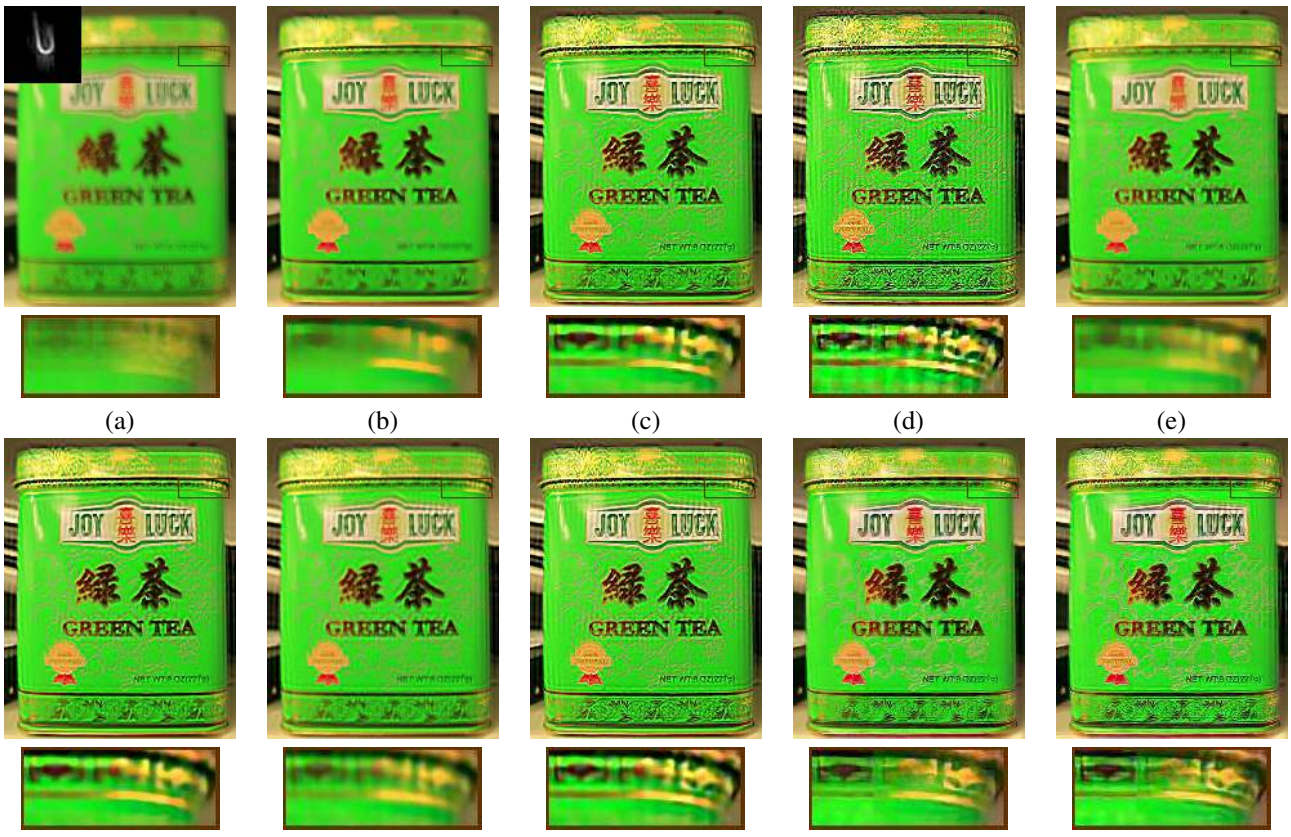
Figure S25. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [19]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5], (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

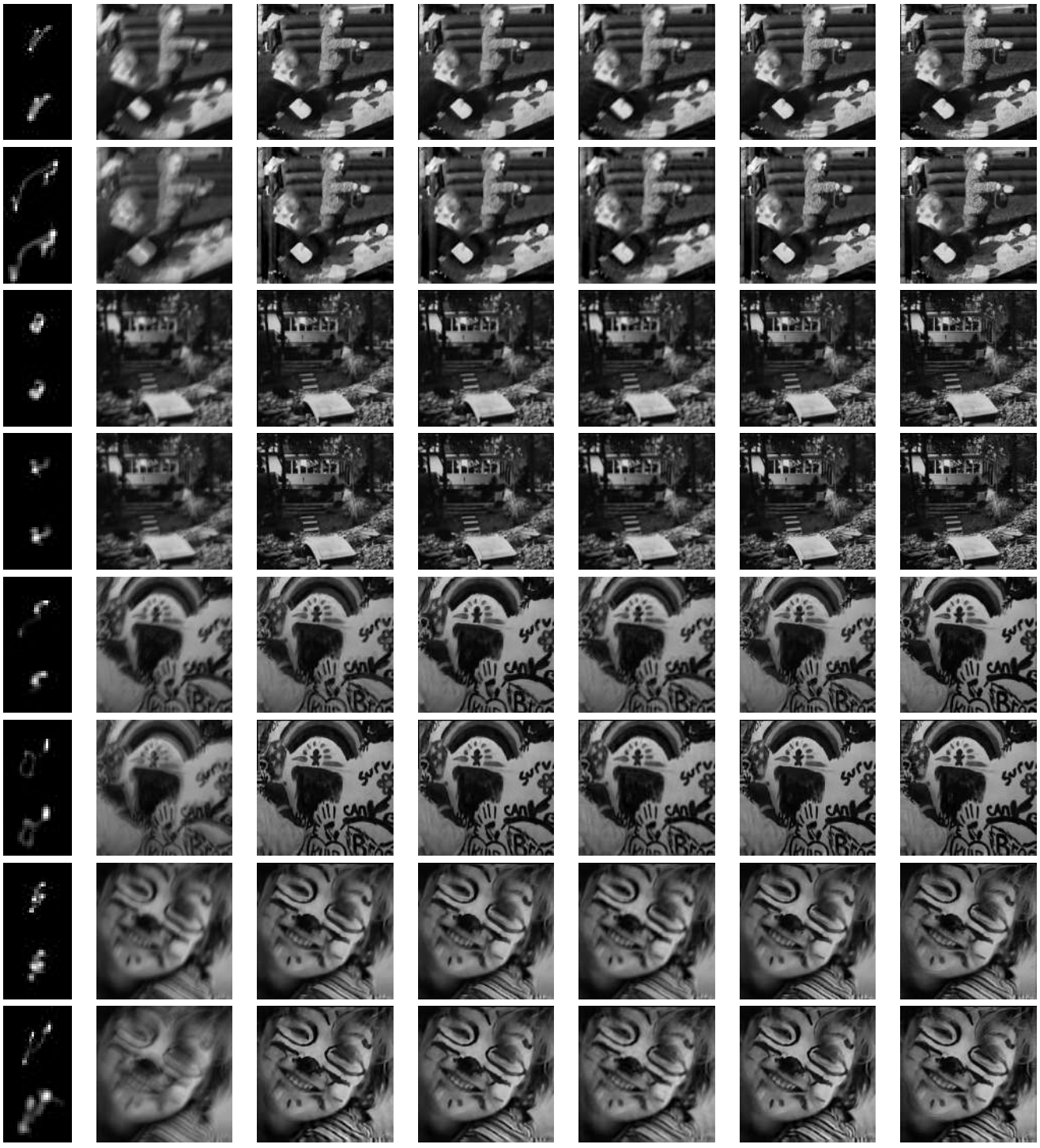
Figure S26. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [13]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5] (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure S27. Real example: Image from dataset in [10], for a noisy kernel estimate returned by [15]. (a) Input blurred image and kernel estimate. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (e) [5] (f) [11], (g) [18], (h) [25], (i) proposed approach ($\text{CNN}_{2/3}$), and (j) [25] $\rightarrow$ $\text{CNN}_{2/3}$.



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

Figure S28. Examples from [13] dataset, for kernel estimates returned by [19, 16]. (a) Ground truth (top) and estimated kernels (bottom). (b) Input blurred image. Restored images using (c) [7] (d) [11], (e) [18], (f) [25], and (g) proposed approach ($\text{CNN}_{2/3}$).

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

Figure S29. Examples from [13] dataset, for kernel estimates returned by [19]. Restored image using [7] with (a) $\lambda = 2e2$, (b) $\lambda = 2e3$, and (c) $\lambda = 2e4$. Results from (d) [11], (e) [18], (f) [25], and (g) proposed approach ($\text{CNN}_{2/3}$).

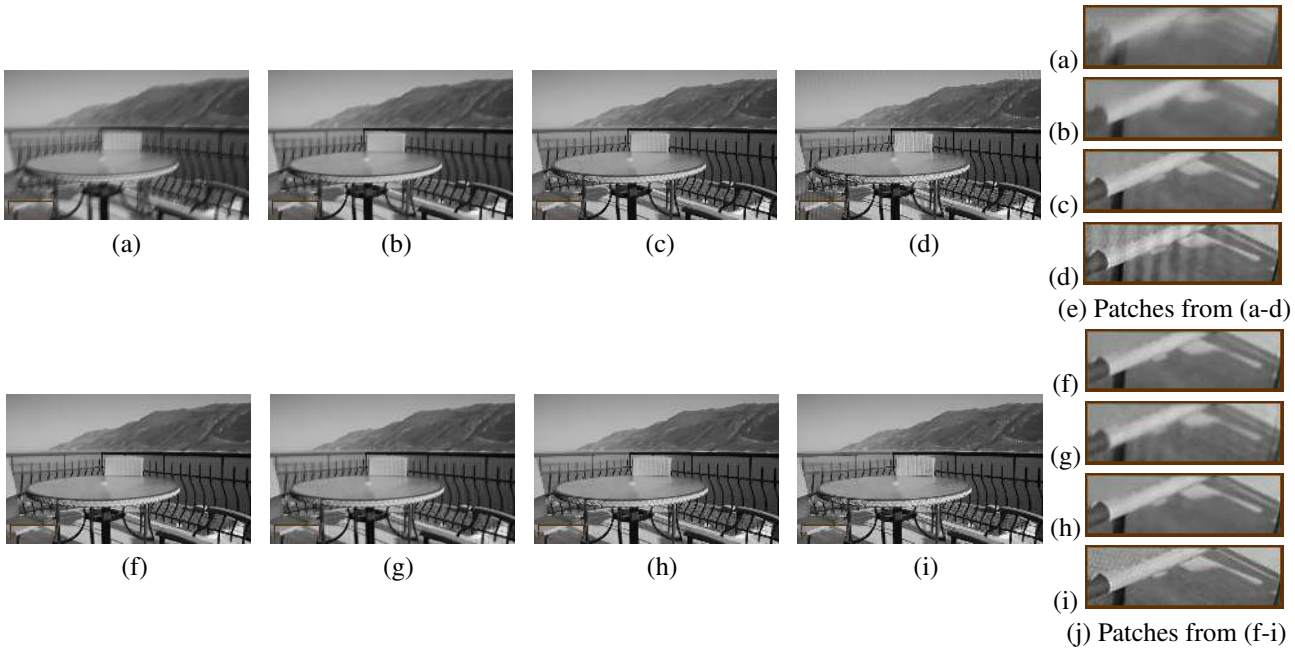


Figure S30. Image from [19] dataset, for a noisy kernel estimate returned by [20]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

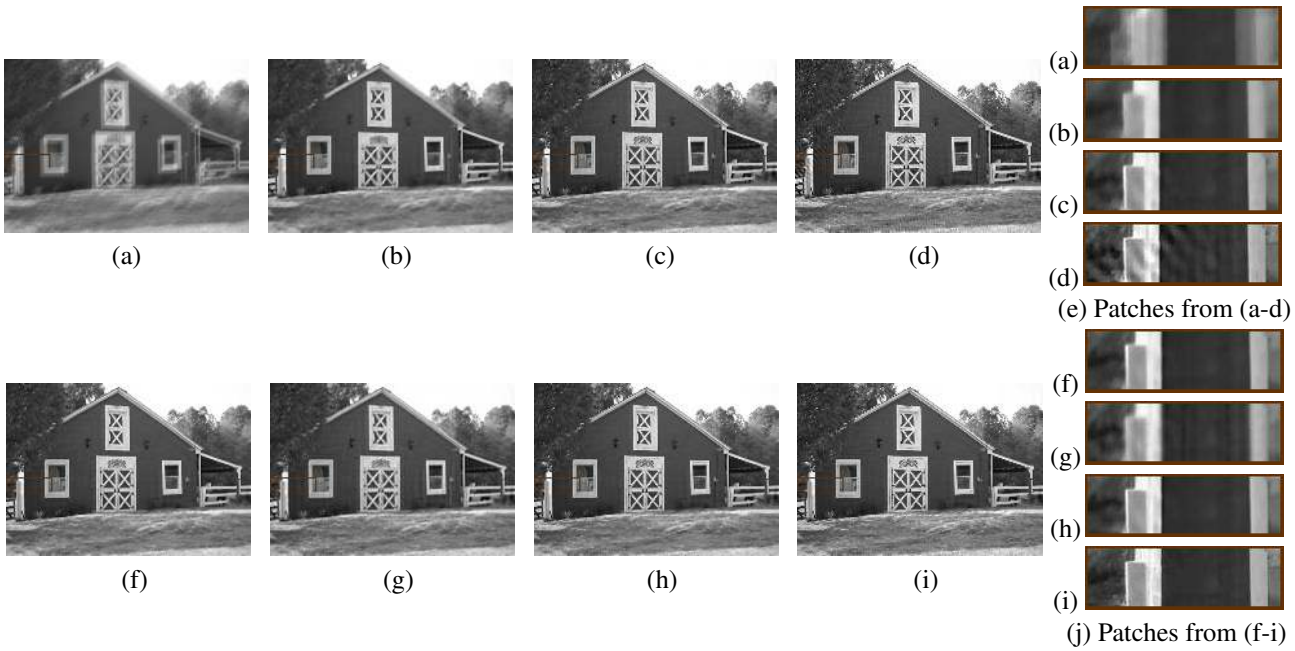


Figure S31. Image from [19] dataset, for a noisy kernel estimate returned by [20]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

Figure S32. Image from [19] dataset, for a noisy kernel estimate returned by [2]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

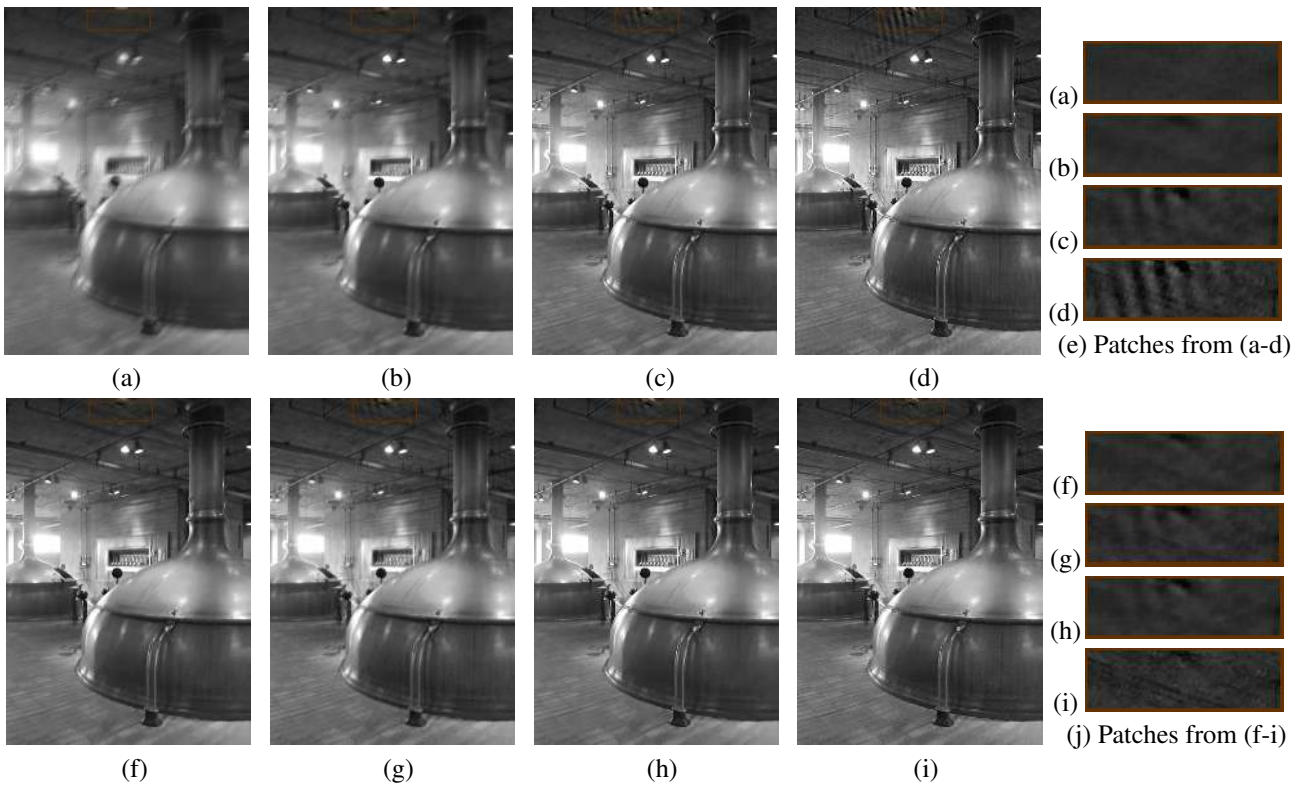


Figure S33. Image from [19] dataset, for a noisy kernel estimate returned by [20]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

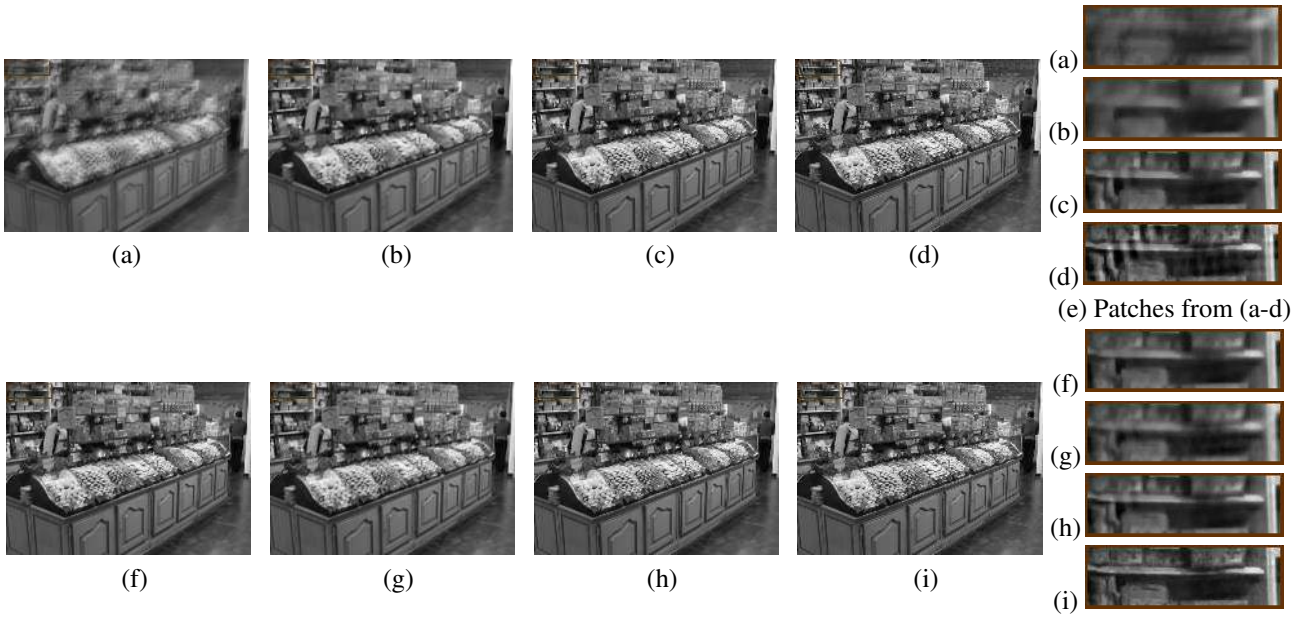


Figure S34. Image from [19] dataset, for a noisy kernel estimate returned by [2]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

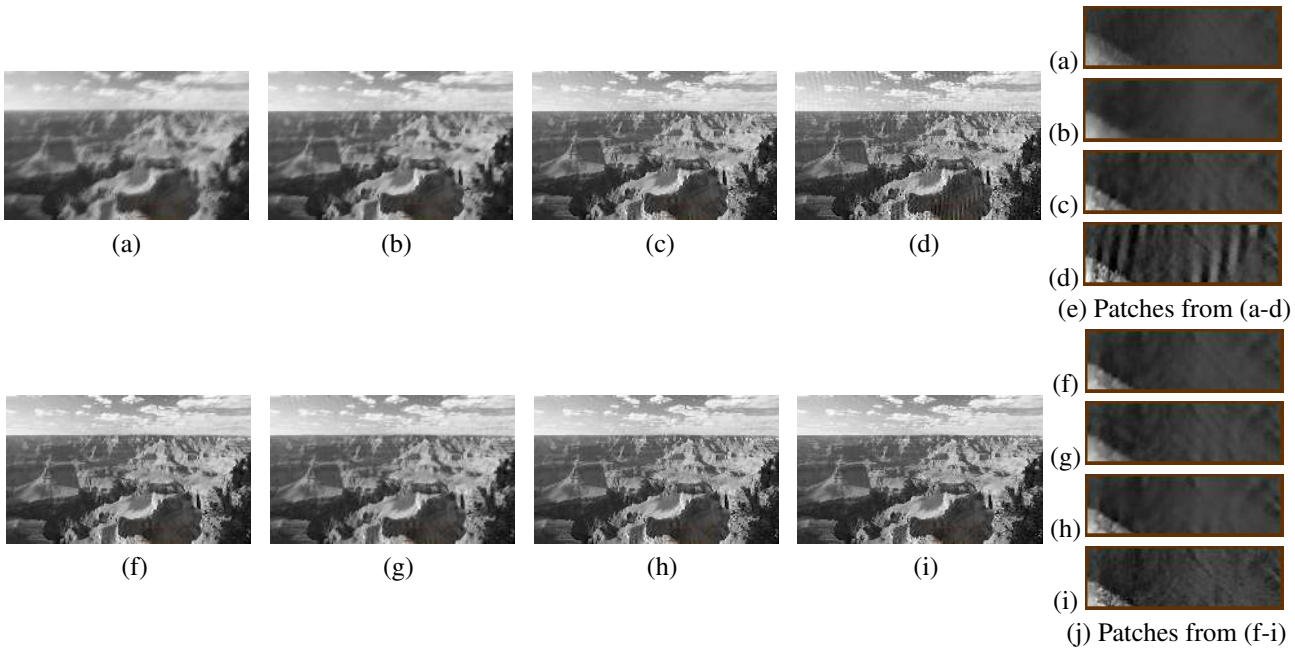


Figure S35. Image from [19] dataset, for a noisy kernel estimate returned by [2]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

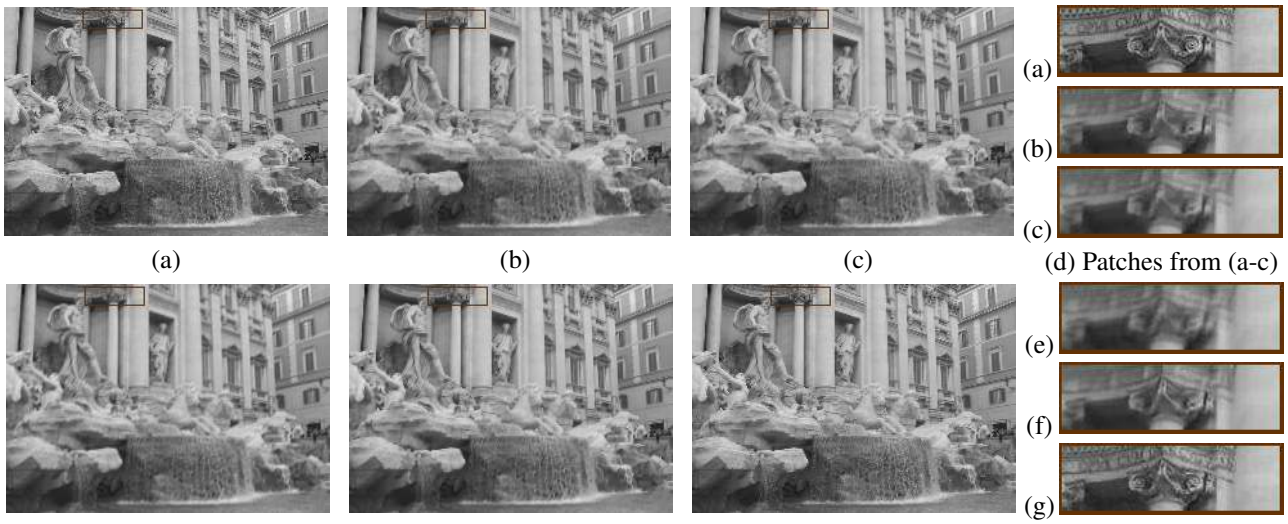


Figure S36. Image from [19] dataset, for a noisy kernel estimate returned by [14]. (a) Ground truth image. Restored image using (b) [7], (c) [11], (e) [18], (f) [25], and (g) proposed approach ($CNN_{2/3}$).

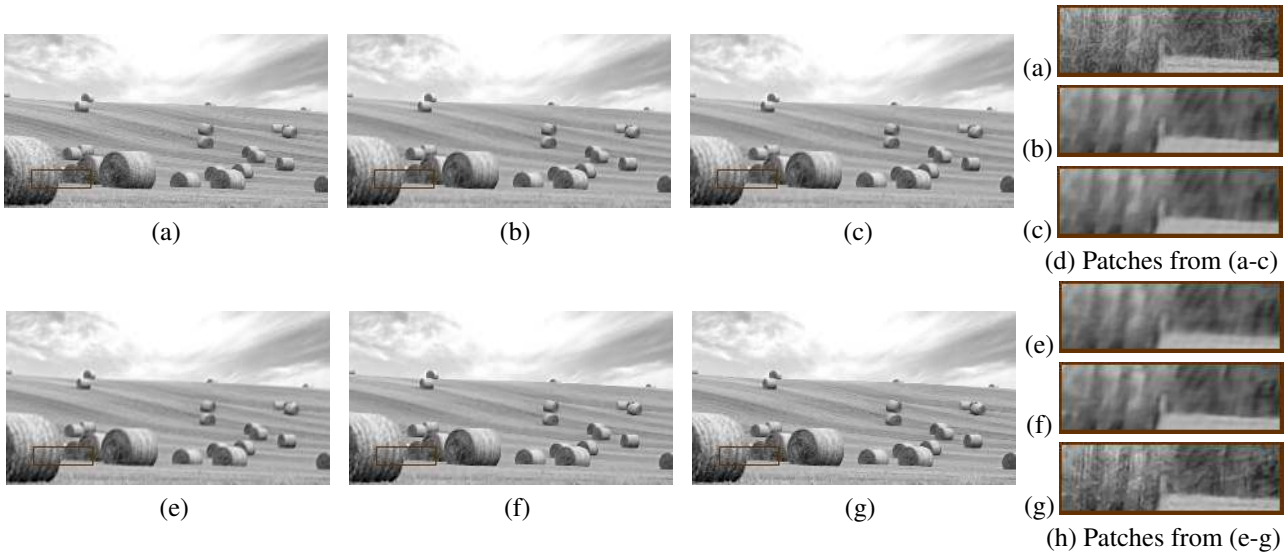
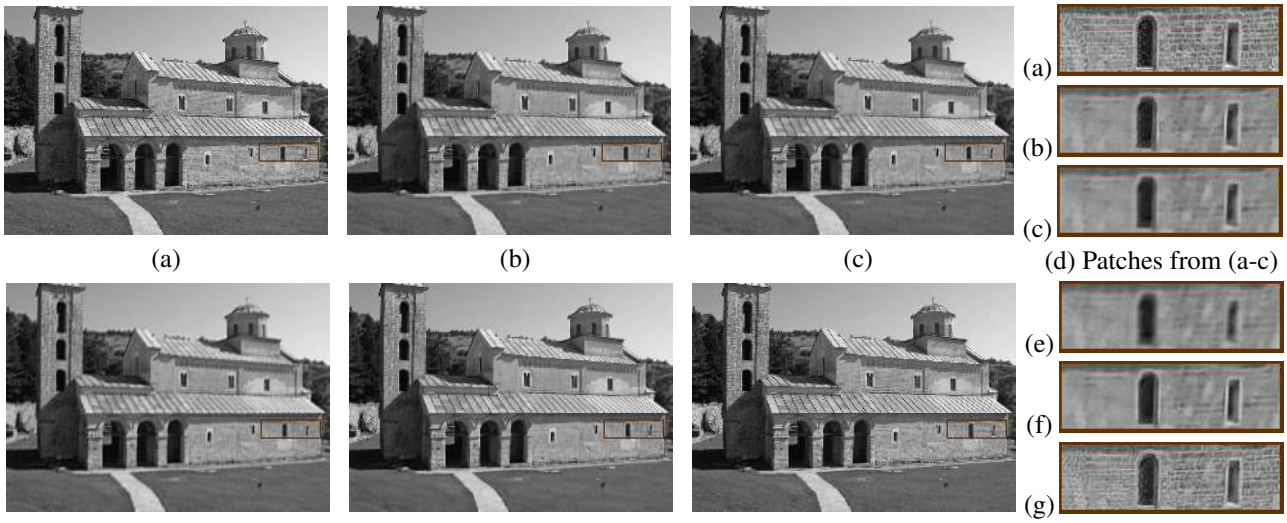


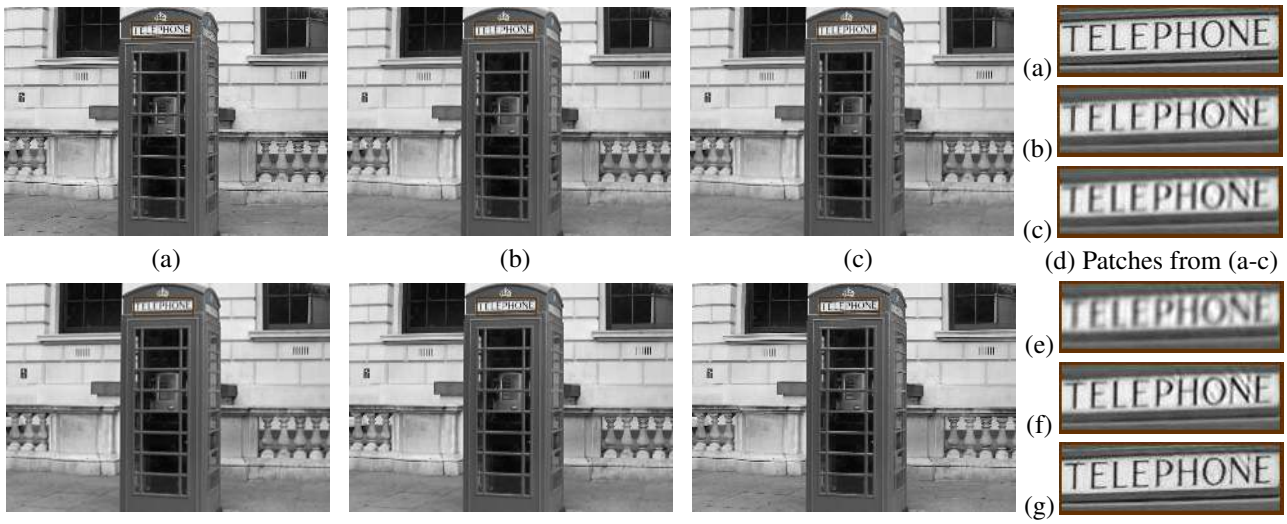
Figure S37. Image from [19] dataset, for a noisy kernel estimate returned by [14]. (a) Ground truth image. Restored image using (b) [7], (c) [11], (e) [18], (f) [25], and (g) proposed approach ($CNN_{2/3}$).



(a) (b) (c) (d) Patches from (a-c)

(e) (f) (g) (h) Patches from (e-g)

Figure S38. Image from [19] dataset, for a noisy kernel estimate returned by [14]. (a) Ground truth image. Restored image using (b) [7], (c) [11], (e) [18], (f) [25], and (g) proposed approach ($CNN_{2/3}$).



(a) (b) (c) (d) Patches from (a-c)

(e) (f) (g) (h) Patches from (e-g)

Figure S39. Image from [19] dataset, for a noisy kernel estimate returned by [14]. (a) Ground truth image. Restored image using (b) [7], (c) [11], (e) [18], (f) [25], and (g) proposed approach ($CNN_{2/3}$).

Figure S40. Image from [19] dataset, for a noisy kernel estimate returned by [14]. (a) Ground truth image. Restored image using (b) [7], (c) [11], (e) [18], (f) [25], and (g) proposed approach ($CNN_{2/3}$).

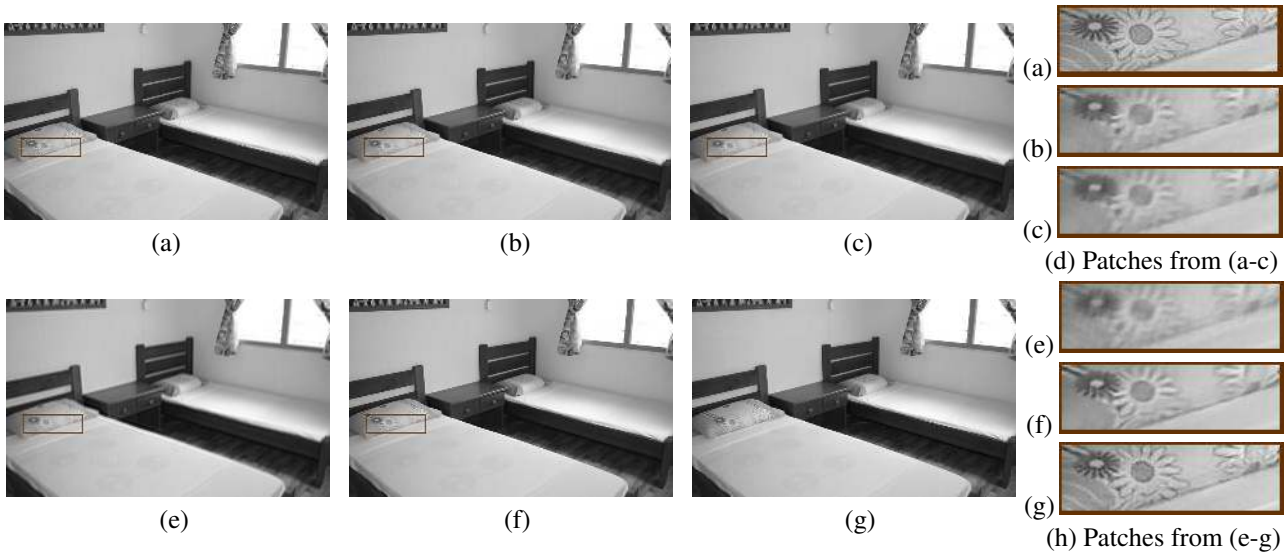


Figure S41. Image from [19] dataset, for a noisy kernel estimate returned by [14]. (a) Ground truth image. Restored image using (b) [7], (c) [11], (e) [18], (f) [25], and (g) proposed approach ($CNN_{2/3}$).

Figure S42. Image from [10] dataset, for a noisy kernel estimate returned by [20]. (a) Ground truth (top) and estimated kernels (bottom). (b) Input blurred image. Restored images using (c) [7] (d) [11], (f) [18], (g) [25], and (h) proposed approach ($\text{CNN}_{2/3}$).

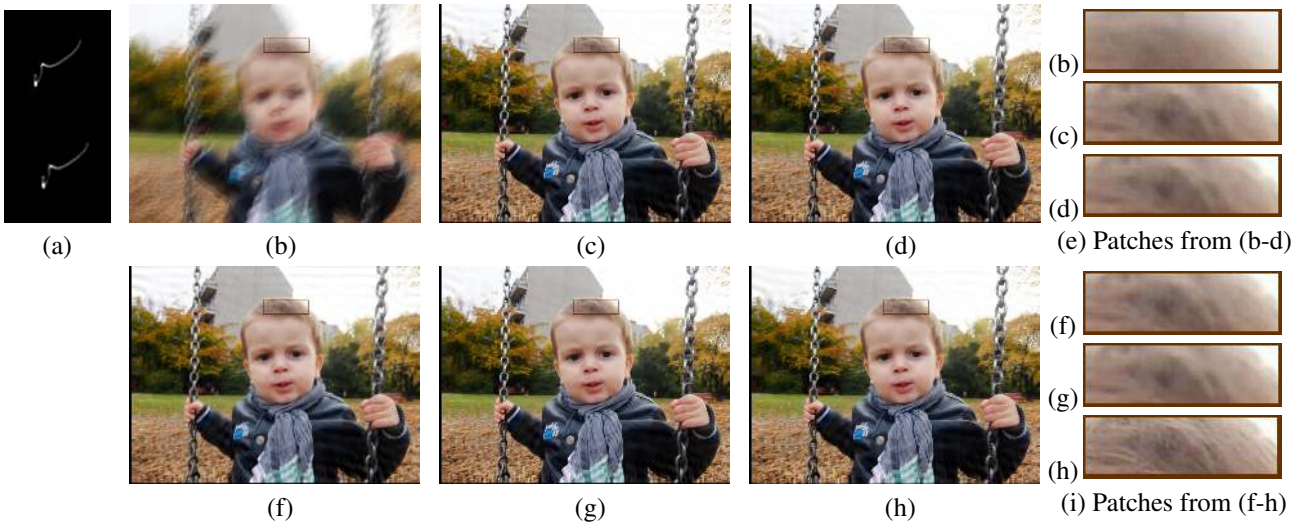


Figure S43. Image from [10] dataset, for a noisy kernel estimate returned by [19]. (a) Ground truth (top) and estimated kernels (bottom). (b) Input blurred image. Restored images using (c) [7] (d) [11], (f) [18], (g) [25], and (h) proposed approach ($\text{CNN}_{2/3}$).

Figure S44. Image from [10] dataset, for a noisy kernel estimate returned by [20]. (a) Ground truth (top) and estimated kernels (bottom). (b) Input blurred image. Restored images using (c) [7] (d) [11], (f) [18], (g) [25], and (h) proposed approach ($\text{CNN}_{2/3}$).

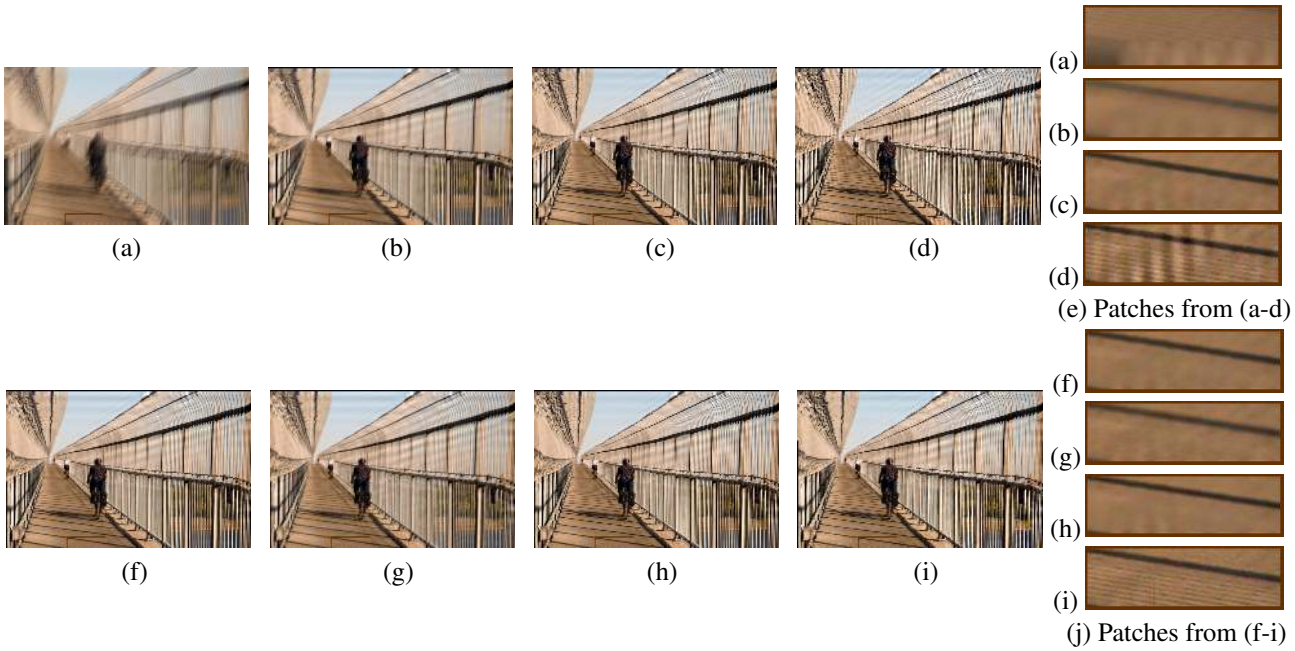


Figure S45. Image from [10] dataset, for a noisy kernel estimate returned by [20]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

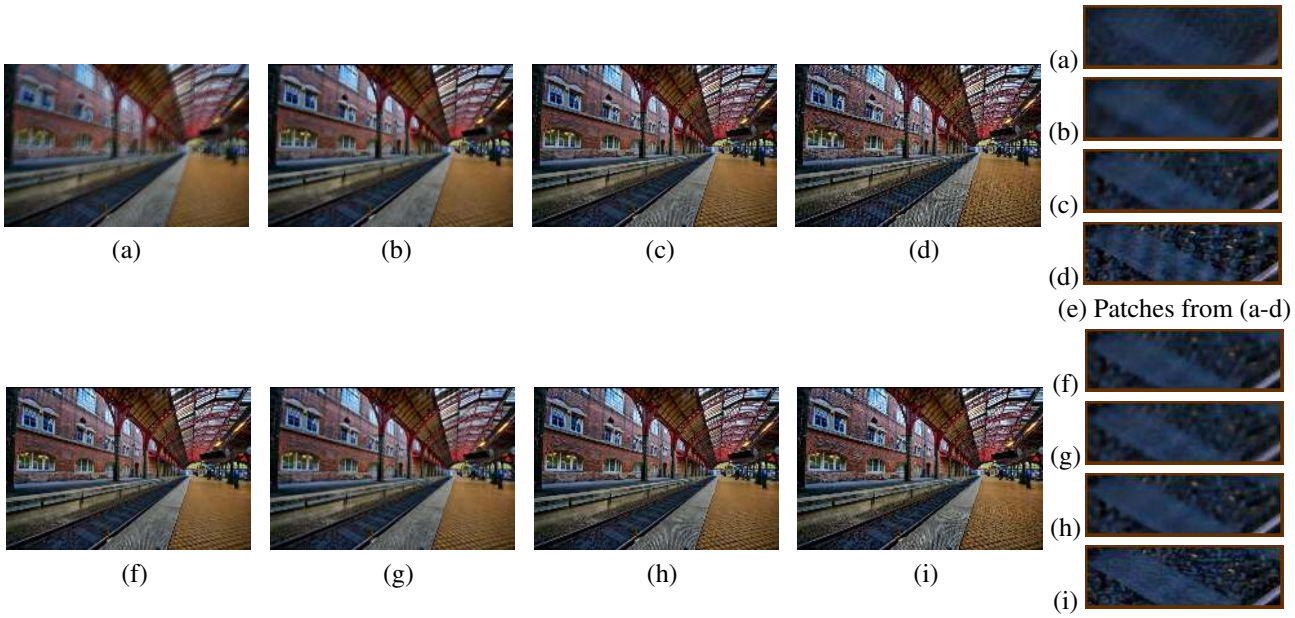


Figure S46. Image from [10] dataset, for a noisy kernel estimate returned by [20]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

Figure S47. Image from [10] dataset, for a noisy kernel estimate returned by [20]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

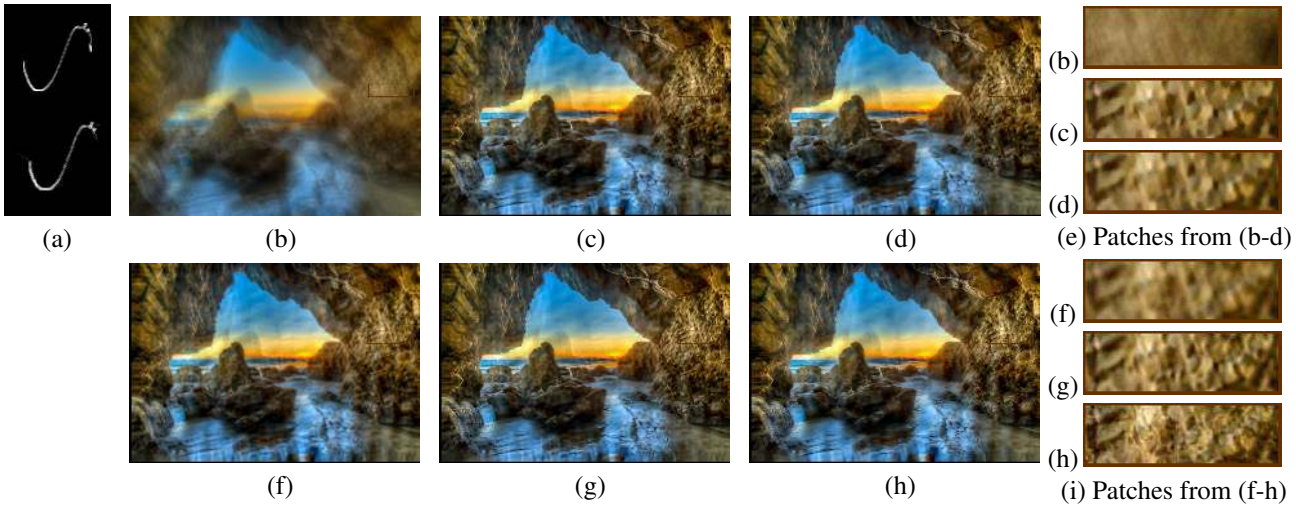


Figure S48. Worst case scenario example 1: Image from dataset in [10], for a severely noisy kernel estimate returned by [19]. (a) Ground truth (top) and estimated kernels (bottom). (b) Input blurred image. Restored images using (c) [7] (d) [11], (f) [18], (g) [25], and (h) proposed approach ($\text{CNN}_{2/3}$).

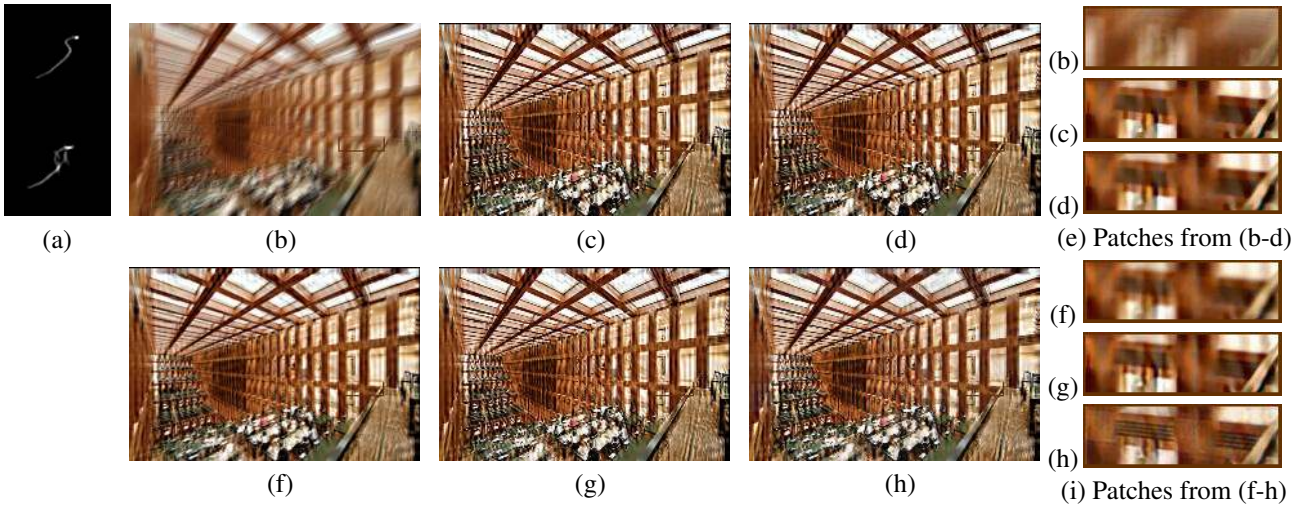


Figure S49. Worst case scenario example 2: Image from dataset in [10], for a severely noisy kernel estimate returned by [20]. (a) Ground truth (top) and estimated kernels (bottom). (b) Input blurred image. Restored images using (c) [7] (d) [11], (f) [18], (g) [25], and (h) proposed approach ($\text{CNN}_{2/3}$).

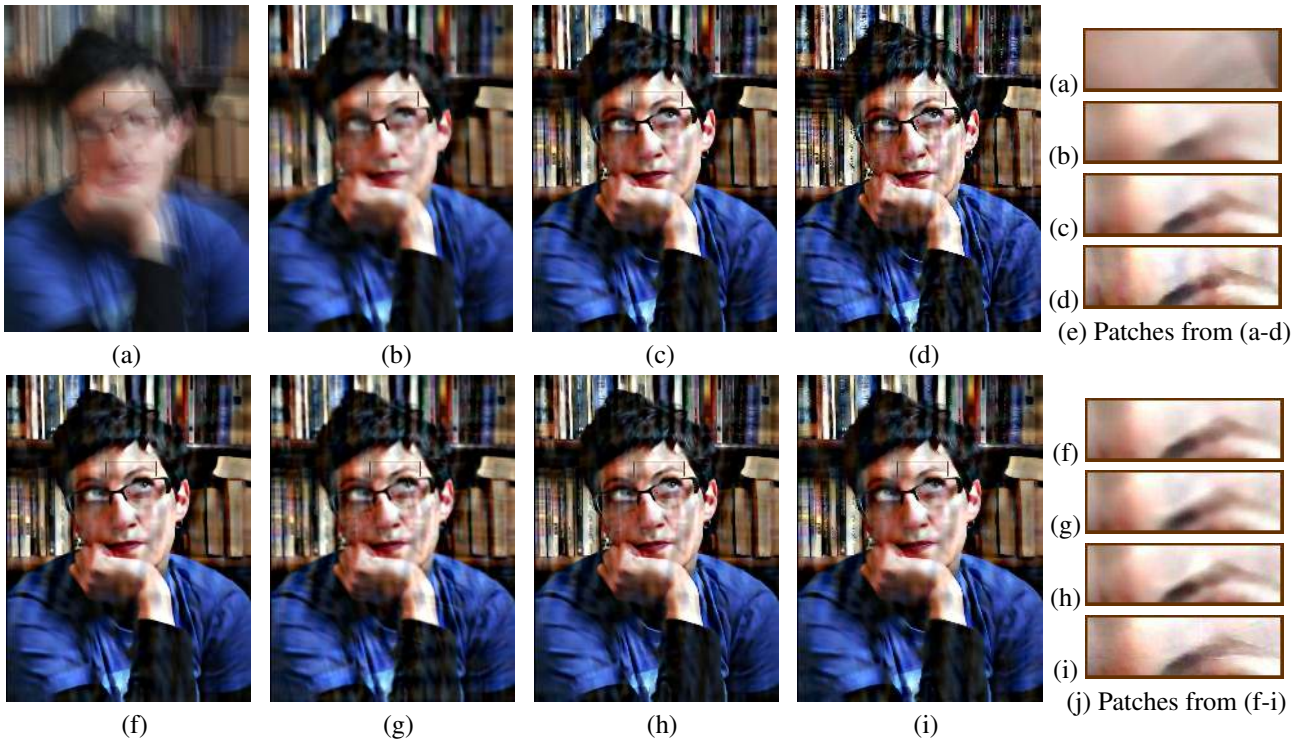


Figure S50. Worst case scenario example 3: Image from dataset in [10], for a severely noisy kernel estimate returned by [20]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

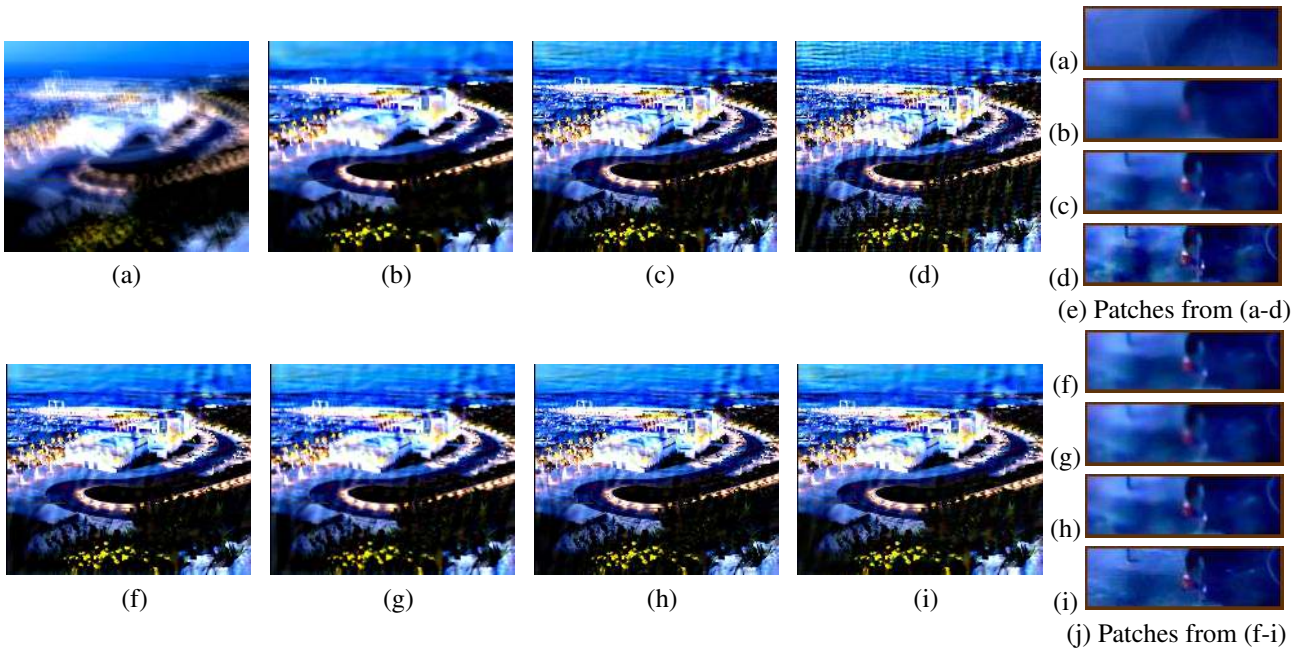


Figure S51. Worst case scenario example 4: Image from dataset in [10], for a severely noisy kernel estimate returned by [19]. (a) Input blurred image. Restored image using [7] with (b) $\lambda = 2e2$, (c) $\lambda = 2e3$, and (d) $\lambda = 2e4$. Results from (f) [11], (g) [18], (h) [25], and (i) proposed approach ($\text{CNN}_{2/3}$).

References

[1] A. Chakrabarti. A neural approach to blind motion deblurring. In *ECCV*, pages 221–235. Springer, 2016. 1

[2] S. Cho and S. Lee. Fast motion deblurring. In *TOG*, volume 28, page 145. ACM, 2009. 2, 4, 5, 6, 10, 20, 21

[3] A. Danielyan, V. Katkovnik, and K. Egiazarian. Bm3d frames and variational image deblurring. *TIP*, 21(4):1715–1728, 2012. 2, 5, 6

[4] R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis, and W. T. Freeman. Removing camera shake from a single photograph. In *TOG*, volume 25, pages 787–794. ACM, 2006. 2, 4

[5] H. Ji and K. Wang. Robust image deblurring with an inaccurate blur kernel. *TIP*, 21(4):1624–1634, 2012. 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

[6] A. Kheradmand and P. Milanfar. A general framework for regularized, similarity-based image restoration. *TIP*, 23(12):5136–5151, 2014. 2, 5, 6, 9, 10, 12

[7] D. Krishnan and R. Fergus. Fast image deconvolution using hyper-laplacian priors. In *NIPS*, pages 1033–1041, 2009. 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29

[8] D. Krishnan, T. Tay, and R. Fergus. Blind deconvolution using a normalized sparsity measure. In *CVPR*, pages 233–240. IEEE, 2011. 14

[9] J. Kruse, C. Rother, and U. Schmidt. Learning to push the limits of efficient fft-based image deconvolution. In *ICCV*, Oct 2017. 1, 2, 5, 6, 7, 9

[10] W.-S. Lai, J.-B. Huang, Z. Hu, N. Ahuja, and M.-H. Yang. A comparative study for single image blind deblurring. In *CVPR*, June 2016. 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 25, 26, 27, 28, 29

[11] A. Levin, R. Fergus, F. Durand, and W. T. Freeman. Image and depth from a conventional camera with a coded aperture. *TOG*, 26(3):70, 2007. 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29

[12] A. Levin, Y. Weiss, F. Durand, and W. T. Freeman. Understanding and evaluating blind deconvolution algorithms. In *CVPR*, pages 1964–1971. IEEE, 2009. 2, 5

[13] A. Levin, Y. Weiss, F. Durand, and W. T. Freeman. Efficient marginal likelihood optimization in blind deconvolution. In *CVPR*, pages 2657–2664. IEEE, 2011. 2, 4, 5, 15, 16, 17, 18

[14] T. Michaeli and M. Irani. Blind deblurring using internal patch recurrence. In *ECCV*, pages 783–798. Springer, 2014. 5, 22, 23, 24

[15] J. Pan, Z. Hu, Z. Su, and M.-H. Yang. Deblurring text images via l0-regularized intensity and gradient prior. In *CVPR*, pages 2901–2908, 2014. 4, 6, 7, 9, 11, 16

[16] J. Pan, D. Sun, H. Pfister, and M.-H. Yang. Blind image deblurring using dark channel prior. In *CVPR*, June 2016. 2, 17

[17] D. Perrone and P. Favaro. Total variation blind deconvolution: The devil is in the details. In *CVPR*, pages 2909–2916, 2014. 5, 8

[18] U. Schmidt and S. Roth. Shrinkage fields for effective image restoration. In *CVPR*, pages 2774–2781, 2014. 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29

[19] L. Sun, S. Cho, J. Wang, and J. Hays. Edge-based blur kernel estimation using patch priors. In *ICCP*, pages 1–8. IEEE, 2013. 2, 5, 12, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 28, 29

[20] L. Xu and J. Jia. Two-phase kernel estimation for robust motion deblurring. In *ECCV*, pages 157–170. Springer, 2010. 5, 8, 10, 12, 19, 20, 25, 26, 27, 28, 29

[21] L. Xu, S. Zheng, and J. Jia. Unnatural l0 sparse representation for natural image deblurring. In *CVPR*, pages 1107–1114, 2013. 3, 4, 5, 11, 14

[22] R. Zeyde, M. Elad, and M. Protter. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*, pages 711–730. Springer, 2010. 2

[23] H. Zhang, D. Wipf, and Y. Zhang. Multi-image blind deblurring using a coupled adaptive sparse prior. In *CVPR*, pages 1051–1058, 2013. 13

[24] L. Zhong, S. Cho, D. Metaxas, S. Paris, and J. Wang. Handling noise in single image deblurring using directional filters. In *CVPR*, pages 612–619, 2013. 13

[25] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *ICCV*, pages 479–486. IEEE, 2011. 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29