Supplementary file to "Deep Convolutional Dictionary Learning for Image Denoising"

Hongyi Zheng Hongwei Yong Lei Zhang

In this supplementary file, we provide the derivation for the closed-form solutions of coefficients $\mathbf{X}$ and dictionaries $\mathbf{D}$ in DCDicL, and more visual comparisons of the denoising results obtained by different methods. Specifically:

- Section 1 presents the derivation of closed-form solution of coefficients $\mathbf{X}$.
- Section 2 presents the derivation of closed-form solution of dictionaries $\mathbf{D}$.
- Section 3 shows more visual comparisons of the denoising results on grayscale image denoising and color image denoising.
- Section 4 shows more visual comparisons between DCDicL-U and DCDicL.
- Section 5 shows more visualization of learned dictionaries on different input images.

1 Solving X

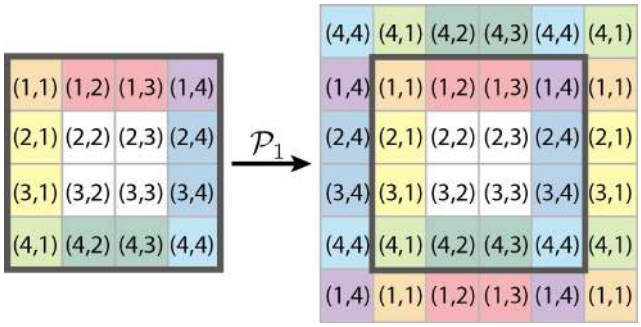


Figure S1: Illustration of circular padding with width 1, denoted by $\mathcal{P}_1$. (x,y) means the pixel at the x^{th} row and the y^{th} column of the image. As can be seen, circular padding repeats the image circularly around it.

To solve $\mathbf{X}'$ in Eq. (10a) of the main paper, we actually need to solve the following problem:

$$\text{argmin}_{\mathbf{X}^\star} \frac{1}{2}\left\|\mathbf{D}\circledast\mathcal{P}_{\frac{k-1}{2}}(\mathbf{X}^\star)-\mathbf{Y}\right\|_2^2+\frac{\alpha_{\mathbf{X}}}{2}\|\mathbf{X}^\star-\mathbf{X}\|_2^2 \tag{S1}$$

where $\mathbf{D}\in\mathbb{R}^{C\times k\times k}$, $\mathbf{X}\in\mathbb{R}^{C\times h\times w}$, $\mathbf{Y}\in\mathbb{R}^{h\times w}$, and $\mathcal{P}_a$ is circular padding operator with width a. (We omit the padding operation in the main paper for the convenience of expression.) Fig. S1 illustrates the circular padding of an image with width 1. According to [1], Eq. S1 can be efficiently solved using Fast Fourier Transform (FFT). Denote by $\mathcal{F}(\cdot)$ the 2D FFT, and let $\boldsymbol{\mathcal{D}}=\mathcal{F}(\mathbf{D})$, $\boldsymbol{\mathcal{X}}^\star=\mathcal{F}(\mathbf{X}^\star)$, $\boldsymbol{\mathcal{Y}}=\mathcal{F}(\mathbf{Y})$ and $\boldsymbol{\mathcal{X}}=\mathcal{F}(\mathbf{X})$. Note that $\boldsymbol{\mathcal{D}}\in\mathbb{R}^{C\times h\times w}$ after FFT. Then solving Eq. S1 is equivalent to solving the following objective function:

$$\text{argmin}_{\boldsymbol{\mathcal{X}}^\star} \frac{1}{2}\|\boldsymbol{\mathcal{D}}\odot\boldsymbol{\mathcal{X}}^\star-\boldsymbol{\mathcal{Y}}\|_2^2+\frac{\alpha_{\mathbf{X}}}{2}\|\boldsymbol{\mathcal{X}}^\star-\boldsymbol{\mathcal{X}}\|_2^2 \tag{S2}$$

Note that the equivalence between Eq. S1 and Eq. S2 can only perfectly hold when the circular padding is adopted in Eq. S1, since Fourier Transform assumes periodic signal in the spatial domain.

Taking the derivative of Eq. S2 w.r.t. $\boldsymbol{\mathcal{X}}^\star$ and letting the derivative be zero, we have:

$$\boldsymbol{\mathcal{D}}\circ\left(\left(\bar{\boldsymbol{\mathcal{D}}}\odot\boldsymbol{\mathcal{X}}^\star\right)-\boldsymbol{\mathcal{Y}}\right)\uparrow_C+\alpha_{\mathbf{X}}(\boldsymbol{\mathcal{X}}^\star-\boldsymbol{\mathcal{X}})=0 \tag{S3}$$

where $\bar{\boldsymbol{\mathcal{D}}}$ denotes complex conjugate of $\boldsymbol{\mathcal{D}}$, $\circ$ is the Hadamard product, $\mathbf{A}\odot\mathbf{B}=\sum_{c=1}^{C}\mathbf{A}_c\circ\mathbf{B}_c$, $\mathbf{A}\uparrow_C$ expands the channel dimension of $\mathbf{A}$ to C.

Let us only consider one pixel in the image, then we have:

$$\boldsymbol{\mathcal{D}}_{i,j}\circ\left(\left(\bar{\boldsymbol{\mathcal{D}}}_{i,j}\odot\boldsymbol{\mathcal{X}}^\star_{i,j}\right)-\mathcal{Y}_{i,j}\right)\uparrow_C+\alpha_{\mathbf{X}_{i,j}}\left(\boldsymbol{\mathcal{X}}^\star_{i,j}-\boldsymbol{\mathcal{X}}_{i,j}\right)=0 \tag{S4}$$

where $\boldsymbol{\mathcal{A}}_{ij}\in\mathbb{R}^C$ is a strip vector of tensor $\boldsymbol{\mathcal{A}}$ at position $\{i,j\}$, $\mathcal{Y}_{i,j}$ is a scalar of tensor $\boldsymbol{\mathcal{Y}}$ at position $\{i,j\}$, $i=1,...h$, and $j=1,...w$. Since $\mathbf{a}\odot\mathbf{b}=\mathbf{a}^T\mathbf{b}$ and $\mathbf{a}\circ(d)\uparrow_C=\mathbf{a}d$, we may rewrite Eq. S4 into matrix multiplication form and get the closed-form solution of $\boldsymbol{\mathcal{X}}^\star_{ij}$:

$$\boldsymbol{\mathcal{X}}^\star_{ij}=\left(\boldsymbol{\mathcal{D}}_{ij}\bar{\boldsymbol{\mathcal{D}}}^T_{ij}+\alpha_{\mathbf{X}}\mathbf{I}_C\right)^{-1}(\boldsymbol{\mathcal{D}}_{ij}\mathcal{Y}_{ij}+\alpha_{\mathbf{X}}\boldsymbol{\mathcal{X}}_{ij}) \tag{S5}$$

With the Sherman-Morrison formula [2], we may eliminate the time-consuming matrix inverse operation as following:

$$\boldsymbol{\mathcal{X}}^\star_{ij}=\frac{1}{\alpha_{\mathbf{X}}}\left(\mathbf{I}_C-\frac{\boldsymbol{\mathcal{D}}_{ij}\bar{\boldsymbol{\mathcal{D}}}^T_{ij}}{\alpha_{\mathbf{X}}+\bar{\boldsymbol{\mathcal{D}}}^T_{ij}\boldsymbol{\mathcal{D}}_{ij}}\right)(\boldsymbol{\mathcal{D}}_{ij}\mathcal{Y}_{ij}+\alpha_{\mathbf{X}}\boldsymbol{\mathcal{X}}_{ij}) \tag{S6}$$

Finally, we can write Eq. S6 to Hadamard product form and solve the $h\times w$ independent linear systems in parallel to get the closed-form solution of $\mathbf{X}'$:

$$\mathbf{X}'=\frac{1}{\alpha_{\mathbf{X}}}\mathcal{F}^{-1}\left\{\boldsymbol{\mathcal{Z}}-\boldsymbol{\mathcal{D}}\circ\left(\frac{(\bar{\boldsymbol{\mathcal{D}}}\odot\boldsymbol{\mathcal{Z}})}{\alpha_{\mathbf{X}}+(\bar{\boldsymbol{\mathcal{D}}}\odot\boldsymbol{\mathcal{D}})}\uparrow_C\right)\right\} \tag{S7}$$

where $\boldsymbol{\mathcal{Z}}=\boldsymbol{\mathcal{D}}\circ(\boldsymbol{\mathcal{Y}}\uparrow_C)+\alpha_{\mathbf{X}}\boldsymbol{\mathcal{X}}$, $\mathcal{F}^{-1}(\cdot)$ denotes the inverse FFT, $\div$ is the elementwise division.

2 Solving D

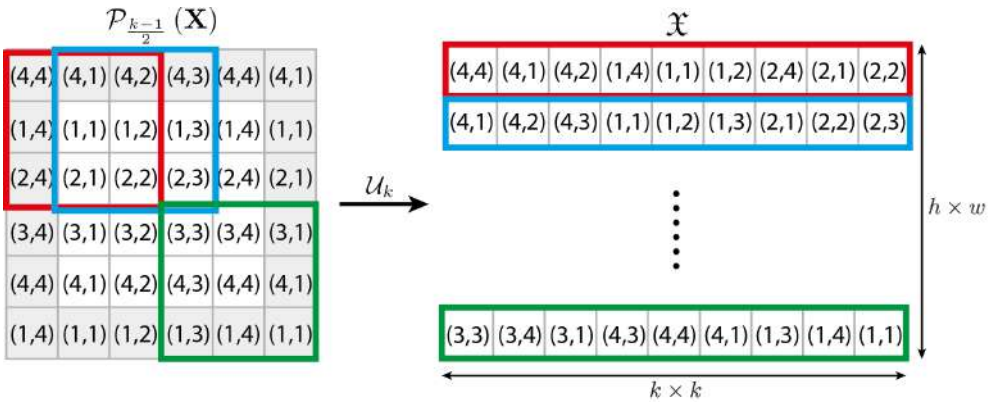


Figure S2: Illustration of unfolding $\mathcal{U}_k$ on an image (after circular padding with $\mathcal{P}_1$) with $k=3$.

In this section, we explain how to compute the closed-form solution of $\mathbf{D}'$ in Eq. (10c) of the main paper. We actually need to solve the following problem:

$$\text{argmin}_{\mathbf{D}^\star}\frac{1}{2}\left\|\mathbf{D}^\star\circledast\mathcal{P}_{\frac{k-1}{2}}(\mathbf{X})-\mathbf{Y}\right\|_2^2+\frac{\alpha_{\mathbf{D}}}{2}\|\mathbf{D}^\star-\mathbf{D}\|_2^2 \tag{S8}$$

where $\mathbf{D}\in\mathbb{R}^{C\times k\times k}$, $\mathbf{X}\in\mathbb{R}^{C\times h\times w}$, and $\mathbf{Y}\in\mathbb{R}^{h\times w}$. As in practice k is much smaller than h and w, the system is overdetermined and can be solved by least squares method. For utilizing the modern least square solvers, we need to unfold $\mathbf{X}$ and $\mathbf{Y}$ into matrix form. Denote by $\mathcal{U}_a$ the unfolding operator with kernel size a, and let $\mathfrak{X}=\mathcal{U}_k\left(\mathcal{P}_{\frac{k-1}{2}}(\mathbf{X})\right)$ and $\mathfrak{Y}=\mathcal{U}_k(\mathbf{Y})$. The unfolding process is illustrated in Fig. S2. Then the objective function in Eq. S8 becomes:

$$\text{argmin}_{\mathbf{d}^\star}\ \frac{1}{2}\|\mathfrak{X}\mathbf{d}-\mathfrak{Y}\|_2^2+\frac{\alpha_{\mathbf{D}}}{2}\|\mathbf{d}^\star-\mathbf{d}\|_2^2 \tag{S9}$$

where $\mathbf{d}^\star=\text{vec}(\mathbf{D}^\star)$, $\mathbf{d}=\text{vec}(\mathbf{D})$, and $\text{vec}(\cdot)$ is the vectorization operator.

By taking the derivative of Eq. S9 function w.r.t. $\mathbf{d}^\star$ and letting the derivative be zero, we have:

$$\mathfrak{X}^T\mathfrak{X}\mathbf{d}^\star-\mathfrak{X}^T\mathfrak{Y}+\alpha_{\mathbf{D}}(\mathbf{d}^\star-\mathbf{d})=0 \tag{S10}$$

By rearranging Eq. S10, we have the following closed-form solution:

$$\mathbf{D}'=\text{vec}^{-1}\left\{\left(\mathfrak{X}^T\mathfrak{X}+\alpha_{\mathbf{D}}\mathbf{I}\right)^{-1}\left(\mathfrak{X}^T\mathfrak{Y}+\alpha_{\mathbf{D}}\mathbf{d}\right)\right\} \tag{S11}$$

where $\text{vec}^{-1}(\cdot)$ reverses the vectorization. Eq. (S11) can be efficiently solved by modern least square solvers such as LU solver provided by PyTorch. However, $\mathfrak{X}$ has a size of $\mathbb{R}^{C\times h\times w\times k\times k}$, which is much larger than $\mathbf{X}\in\mathbb{R}^{C\times h\times w}$ and would consume much memory. Note that $\mathfrak{Y}$ can be regarded as a special case of $\mathfrak{X}$, where $C=1$. Hence, we focus on $\mathfrak{X}$ and $\mathfrak{X}^T\mathfrak{X}$ in the following

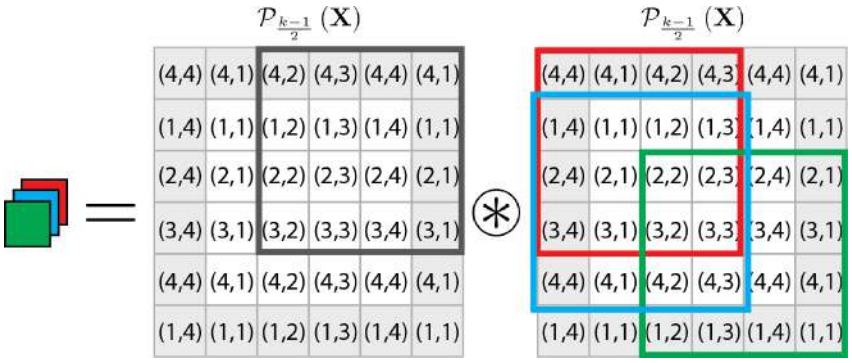


Figure S3: Illustration of $\mathcal{P}_{\frac{k-1}{2}}(\mathbf{X}) \circledast \mathcal{P}_{\frac{k-1}{2}}(\mathbf{X})$.

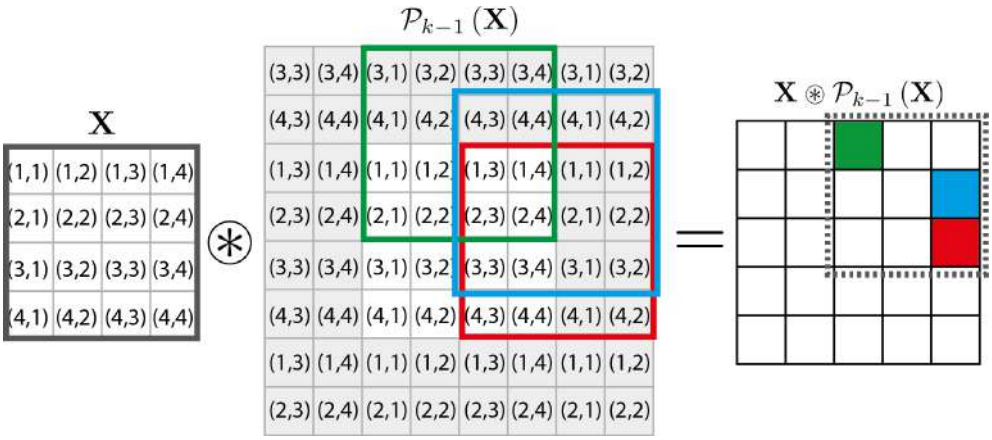


Figure S4: Illustration of $\mathbf{X} \circledast \mathcal{P}_{k-1}(\mathbf{X})$

development. It can be seen that $\mathfrak{X}^T\mathfrak{X}$ has a size of $\mathbb{R}^{C\times k\times k\times k\times k}$. Since $k \ll h$ and $k \ll w$, it would save a lot of memory if we could directly calculate $\mathfrak{X}^T\mathfrak{X}$ from $\mathbf{X}$ without explicitly storing $\mathfrak{X}$.

The elements (x,y) in $\mathfrak{X}^T\mathfrak{X}$ can be calculated through convolving the x^{th} and the y^{th} unfolding patches in $\mathcal{P}_{\frac{k-1}{2}}(\mathbf{X})$ with unfolding kernel of size $2(k-2)$, as illustrated in Fig. S3. However, each element in $\mathfrak{X}^T\mathfrak{X}$ corresponds to a unique pair of unfolding patches in $\mathcal{P}_{\frac{k-1}{2}}(\mathbf{X})$, and such a calculation can not fully utilize the parallel processing capabilities of deep learning framework.

Fortunately, we can also get the elements through convolution between a universal kernel $\mathbf{X}$ with an enlarged feature map $\mathcal{P}_{k-1}(\mathbf{X})$, as shown in Fig. S4. The elements in the x^{th} row of $\mathfrak{X}^T\mathfrak{X}$ will be in the x^{th} unfolding patch of $\mathbf{X} \circledast \mathcal{P}_{k-1}(\mathbf{X})$ with unfolding kernel of size k. Finally, the complete process of calculating $\mathfrak{X}^T\mathfrak{X}$ is as follows:

$$\mathfrak{X}^T\mathfrak{X} = \mathcal{R}(\mathcal{U}_k(\mathbf{X} \circledast \mathcal{P}_{k-1}(\mathbf{X}))) \tag{S12}$$

where $\mathcal{R}(\mathbf{A})$ reverses each row of matrix $\mathbf{A}$. Note that when $C>1$, $\circledast$ in Eq. S12 becomes the 3D convolutional operator. The process in Eq. S12 is also illustrated in Fig. S5.

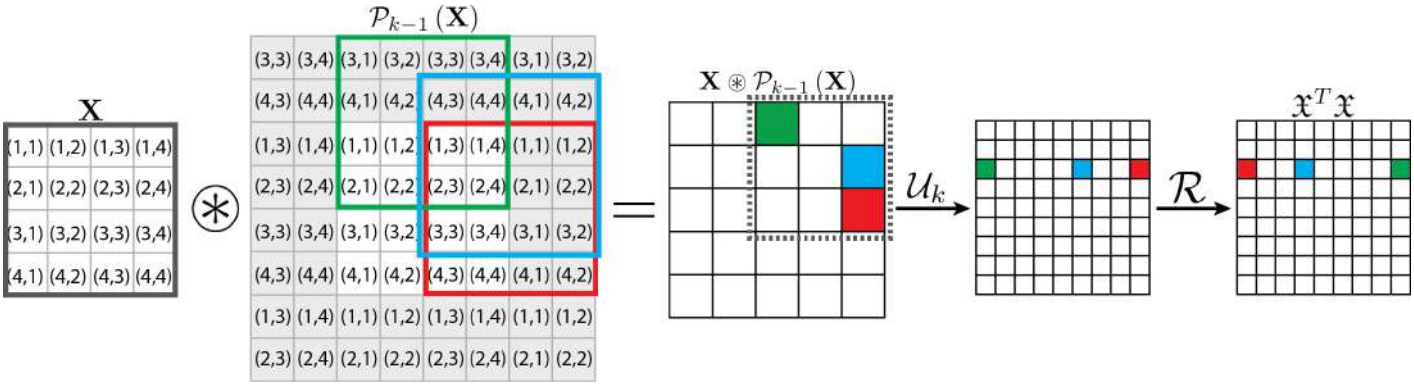
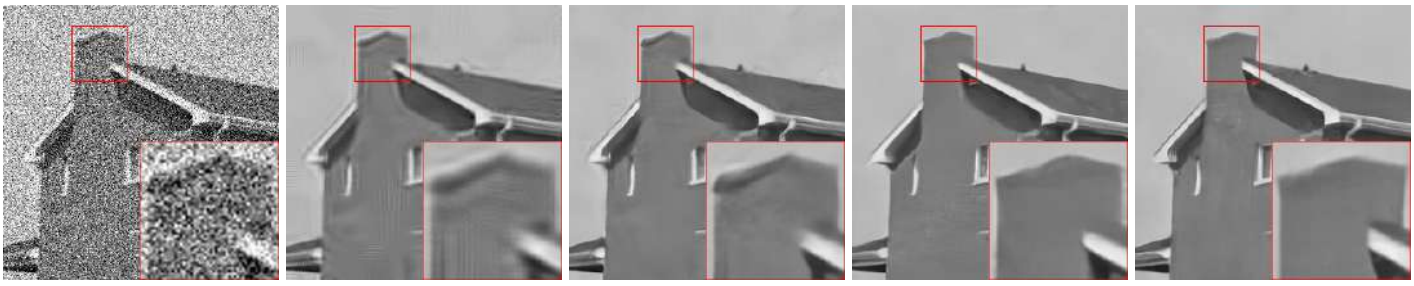
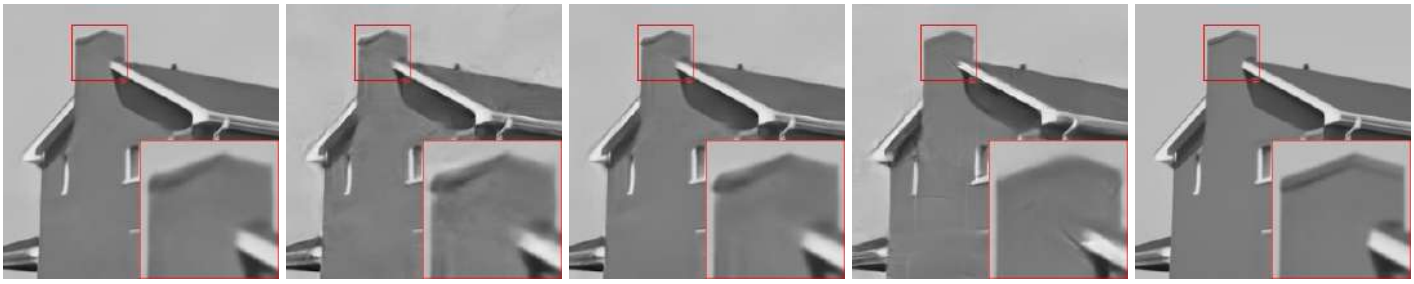


Figure S5: Illustration of the complete process for calculating $\mathfrak{X}^T\mathfrak{X}$.

3 Image Denoising Results



Noisy input ($\sigma=50$) BM3D (29.02dB/78.76) DnCNN (29.96dB/81.85) N3Net (30.50dB/84.15) NLRN (30.62dB/83.17)



RNAN (31.02dB/83.72) IRCNN (29.90dB/81.59) FFDNet (30.36dB/82.73) CSCNet (30.22dB/82.34) DCDicL (31.47dB/84.50)

Figure S6: Denoising results on image 02 in Set12.

Noisy input ($\sigma=50$) BM3D (27.06dB/79.42) DnCNN (26.15dB/76.82) N3Net (27.01dB/80.81) NLRN (27.76dB/82.25)

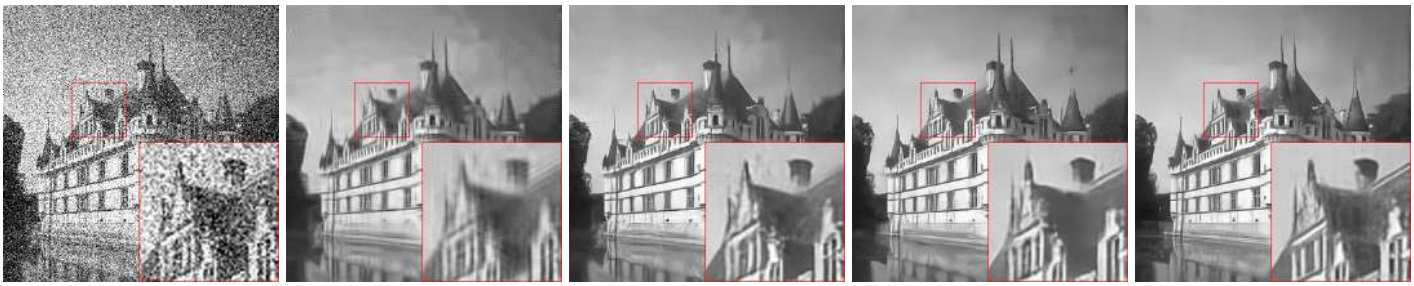
RNAN (27.65dB/82.05) IRCNN (26.17dB/77.04) FFDNet (26.41dB/77.92) CSCNet (25.77dB/75.21) DCDicL (28.47dB/85.27)

Figure S7: Denoising results on image 09 in Set12.

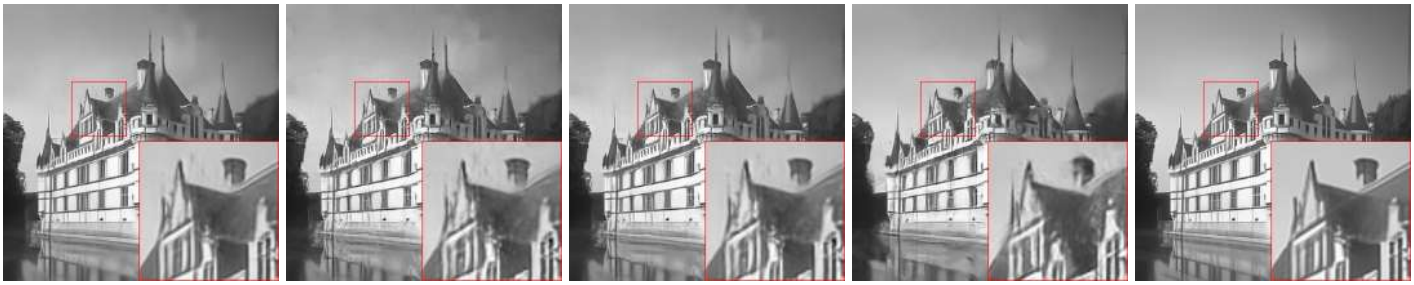
Noisy input ($\sigma=50$) BM3D (23.62dB/68.91) DnCNN (25.53dB/76.89) N3Net (25.90dB/80.29) NLRN (26.17dB/79.91)

RNAN (26.21dB/80.29) IRCNN (25.55dB/76.92) FFDNet (25.74dB/78.11) CSCNet (25.61dB/77.98) DCDicL (26.60dB/82.72)

Figure S8: Denoising results on image test011 in BSD68.

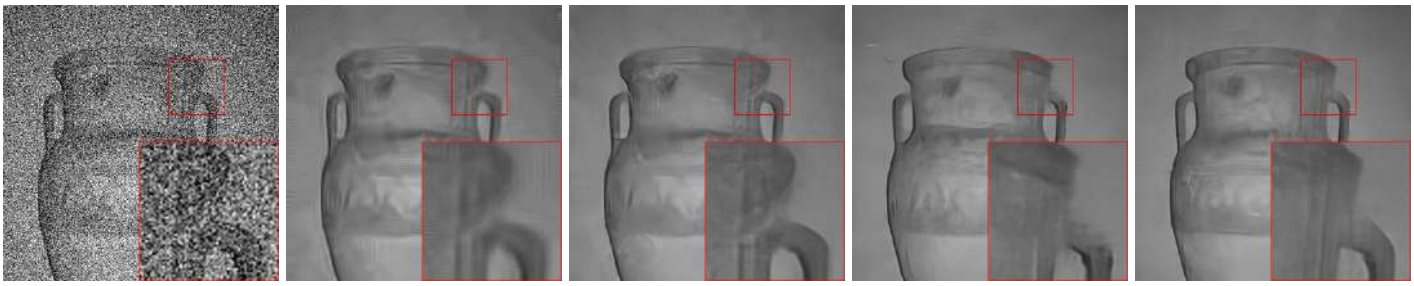


Noisy input ($\sigma=50$) BM3D (25.63dB/80.48) DnCNN (26.90dB/79.38) N3Net (27.14dB/81.18) NLRN (27.27dB/80.54)



RNAN (27.29dB/80.98) IRCNN (26.85dB/79.08) FFDNet (26.92dB/79.96) CSCNet (26.85dB/79.52) DCDicL (27.54dB/82.07)

Figure S9: Denoising results on image test003 in BSD68.



Noisy input ($\sigma=50$) BM3D (30.39dB/76.85) DnCNN (31.09dB/80.15) N3Net (31.29dB/81.69) NLRN (31.42dB/81.09)

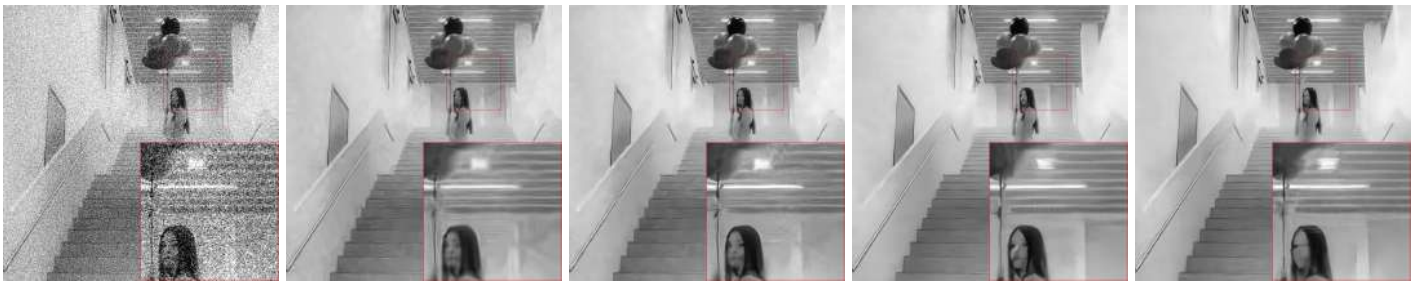
RNAN (31.43dB/81.09) IRCNN (30.99dB/79.67) FFDNet (31.26dB/80.84) CSCNet (31.29dB/80.89) DCDicL (31.97dB/83.40)

Figure S10: Denoising results on image test045 in BSD68.

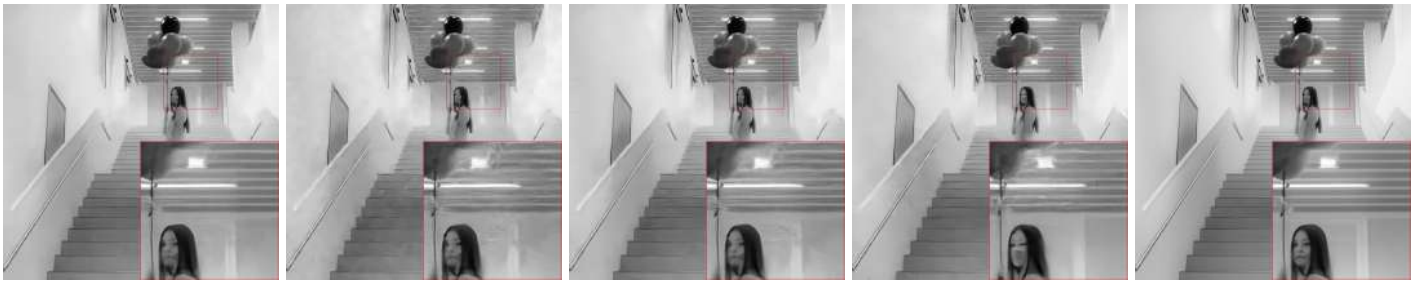
Noisy input ($\sigma=50$) BM3D (22.62dB/59.59) DnCNN (23.91dB/66.83) N3Net (24.15dB/69.38) NLRN (24.33dB/68.92)

RNAN (24.24dB/58.41) IRCNN (23.91dB/66.96) FFDNet (23.98dB/67.29) CSCNet (23.99dB/67.35) DCDicL (24.47dB/70.17)

Figure S11: Denoising results on image test022 in BSD68.

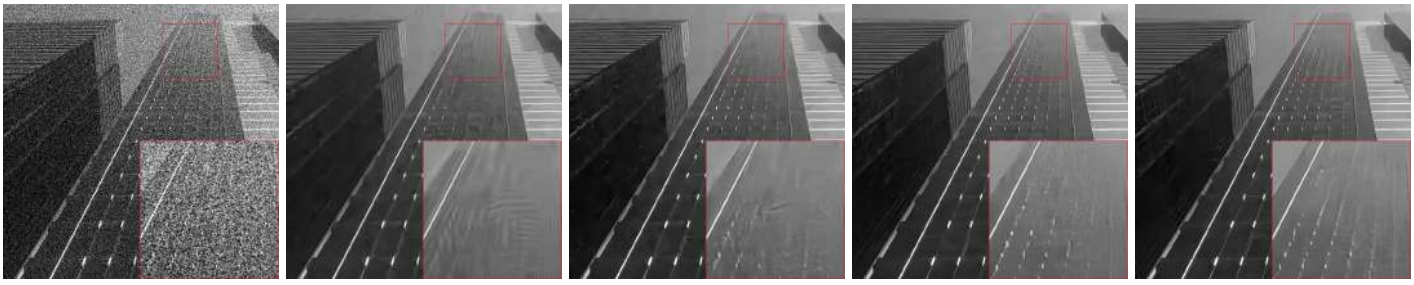


Noisy input ($\sigma=50$) BM3D (29.69dB/84.74) DnCNN (31.68dB/87.31) N3Net (32.91dB/90.53) NLRN (33.02dB/90.16)

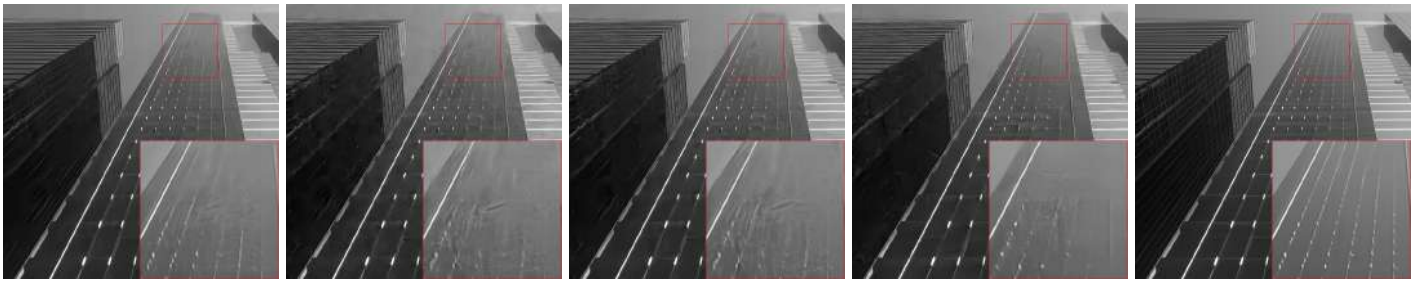


RNAN (33.72dB/91.74) IRCNN (31.54dB/86.83) FFDNet (32.40dB/89.24) CSCNet (32.62dB/90.20) DCDicL (34.92dB/93.51)

Figure S12: Denoising results on image 009 in Urban100.

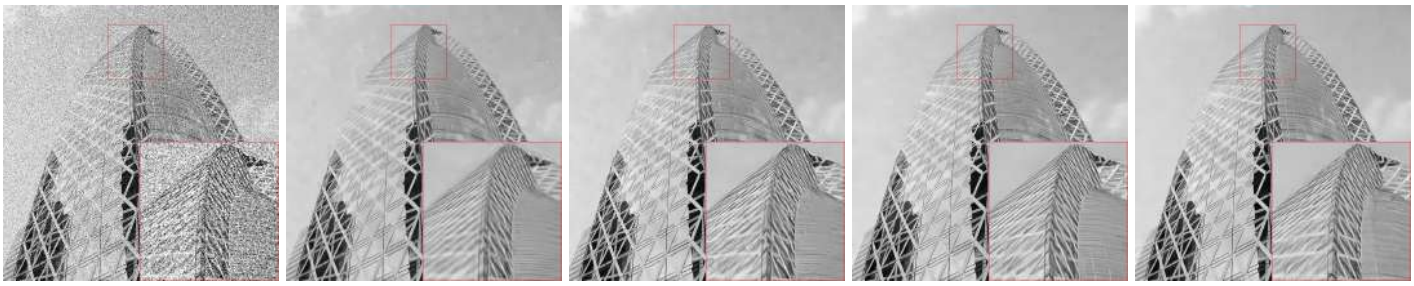


Noisy input ($\sigma=50$) BM3D (25.58dB/71.80) DnCNN (27.35dB/76.46) N3Net (27.94dB/79.25) NLRN (28.40dB/79.92)

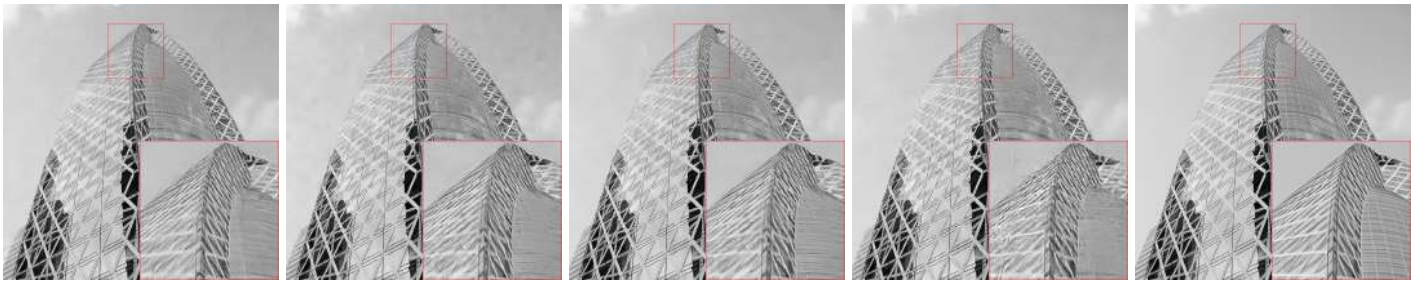


RNAN (28.81dB/81.43) IRCNN (27.30dB/76.21) FFDNet (27.50dB/77.36) CSCNet (27.36dB/77.44) DCDicL (30.03dB/86.80)

Figure S13: Denoising results on image 033 in Urban100.



Noisy input ($\sigma=50$) BM3D (22.82dB/74.67) DnCNN (24.79dB/82.22) N3Net (25.20dB/84.34) NLRN (26.19dB/86.64)



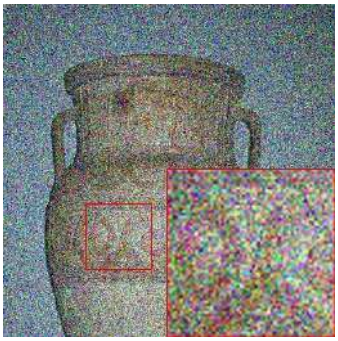
RNAN (26.19dB/86.64) IRCNN (24.61dB/81.72) FFDNet (24.80dB/82.79) CSCNet (24.48dB/81.53) DCDicL (27.41dB/90.52)

Figure S14: Denoising results on image 039 in BSD68.

Noisy input ($\sigma=50$) BM3D (21.85dB/64.69) DnCNN (22.32dB/72.02) N3Net (22.58dB/74.34) NLRN (23.03dB/75.73)

RNAN (23.03dB/75.89) IRCNN (22.35dB/72.58) FFDNet (22.40dB/72.85) CSCNet (22.22dB/72.00) DCDicL (23.43dB/79.23)

Figure S15: Denoising results on image 073 in Urban100.



Noisy input ($\sigma=50$)

RNAN (32.91dB/85.53)

BRDNet (32.65dB/84.80)

FFDNet (32.47dB/84.25)

IRCNN (32.20dB/83.83)

DCDicL (33.51dB/87.33)

Figure S16: Denoising results on image 227092 in CBSD68.

Noisy input ($\sigma=50$)

RNAN (27.16dB/78.26)

BRDNet (26.94dB/77.10)

FFDNet (26.82dB/76.46)

IRCNN (26.79dB/77.29)

DCDicL (27.51dB/79.88)

Figure S17: Denoising results on image 216081 in CBSD68.

Noisy input ($\sigma=50$)

RNAN (30.58dB/79.53)

BRDNet (30.33dB/78.80)

FFDNet (30.06dB/77.77)

IRCNN (29.87dB/77.65)

DCDicL (31.15dB/81.35)

Figure S18: Denoising results on image kodim04 in Kodak24.

Noisy input ($\sigma=50$)

RNAN (31.68dB/85.57)

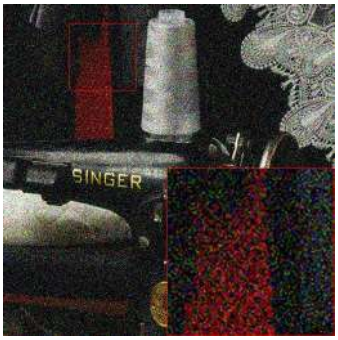
BRDNet (31.32dB/84.47)

FFDNet (31.01dB/83.89)

IRCNN (30.73dB/83.12)

DCDicL (32.30dB/87.30)

Figure S19: Denoising results on image kodim10 in Kodak24.



Noisy input ($\sigma=50$)

RNAN (30.68dB/86.43)

BRDNet (30.27dB/85.89)

FFDNet (30.16dB/82.92)

IRCNN (30.07dB/83.66)

DCDicL (31.41dB/89.53)

Figure S20: Denoising results on image 8 in McMaster.

Noisy input ($\sigma=50$)

RNAN (31.55dB/89.39)

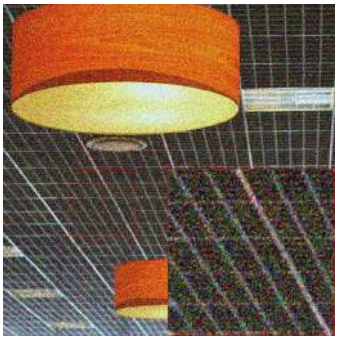
BRDNet (31.20dB/88.44)

FFDNet (30.85dB/88.06)

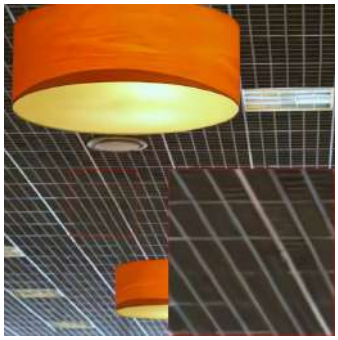
IRCNN (30.44dB/86.48)

DCDicL (32.25dB/90.83)

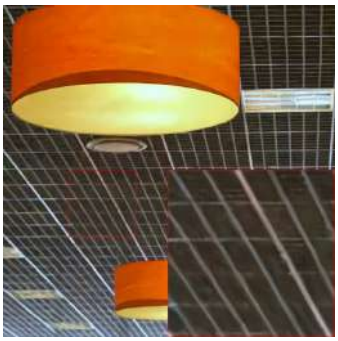
Figure S21: Denoising results on image 12 in McMaster.



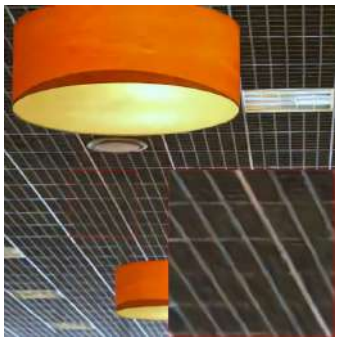
Noisy input ($\sigma=50$)



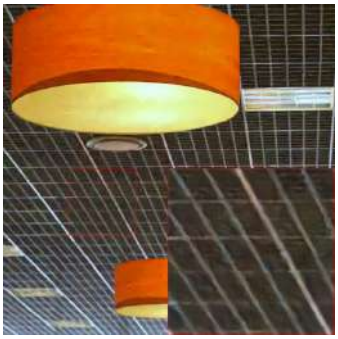
RNAN (31.41dB/90.59)



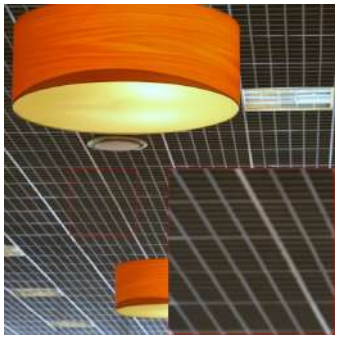
BRDNet (30.35dB/88.00)



FFDNet (29.91dB/86.66)

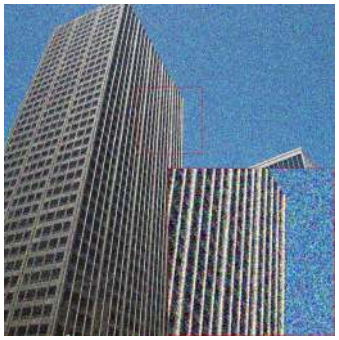


IRCNN (35.57dB/85.95)



DCDicL (33.63dB/94.76)

Figure S22: Denoising results on image 044 in Urban100.



Noisy input ($\sigma=50$)

RNAN (30.50dB/94.54)

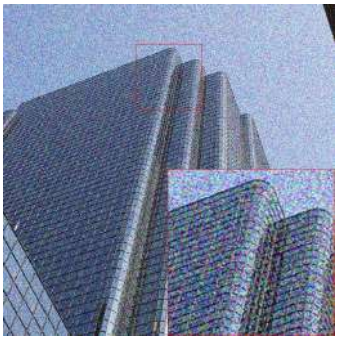
BRDNet (29.62dB/93.40)

FFDNet (28.83dB/92.57)

IRCNN (28.35dB/91.38)

DCDicL (31.87dB/95.95)

Figure S23: Denoising results on image 096 in Urban100.



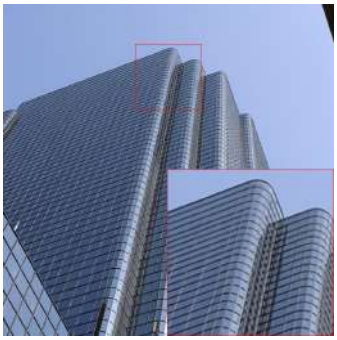
Noisy input ($\sigma=50$)

RNAN (27.82dB/90.75)

BRDNet (27.26dB/89.50)

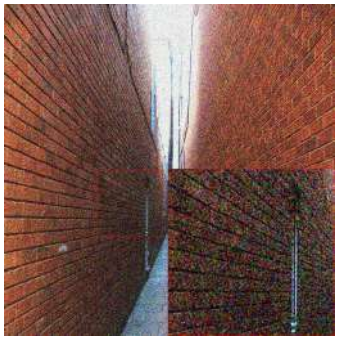
FFDNet (26.78dB/88.17)

IRCNN (26.51dB/87.26)



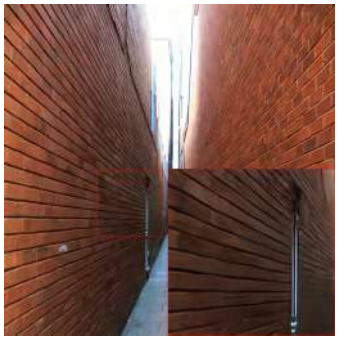
DCDicL (29.16dB/93.32)

Figure S24: Denoising results on image 074 in Urban100.



Noisy input ($\sigma=50$)

RNAN (27.87dB/73.33)



BRDNet (27.55dB/71.52)

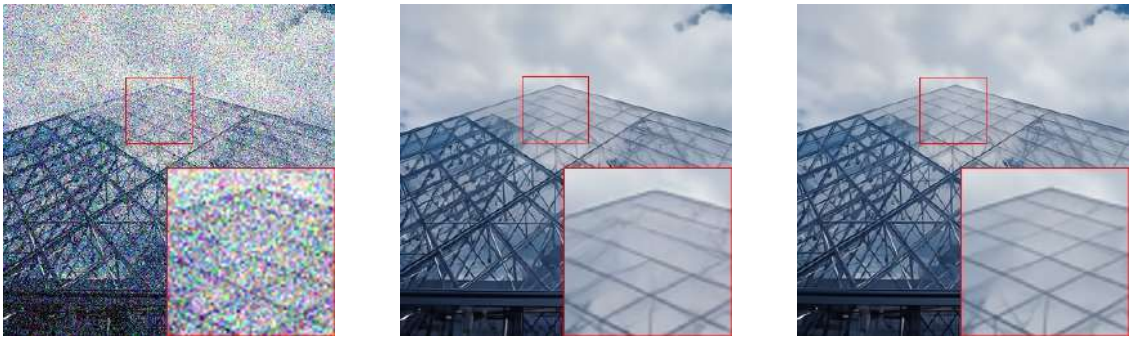
FFDNet (27.21dB/69.30)

IRCNN (26.95dB/69.43)

DCDicL (28.39dB/75.75)

Figure S25: Denoising results on image 038 in Urban100.

4 Results of DCDicL-U and DCDicL



Noisy input ($\sigma=50$) DCDicL-U (27.40dB/84.50) DCDicL (27.87dB/86.04)

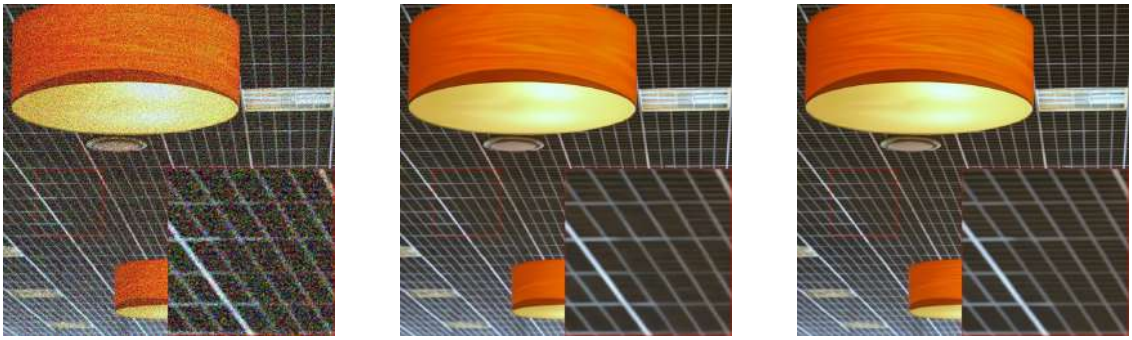
Figure S26: Denoising results on image 023061 in CBSD68.

Noisy input ($\sigma=50$) DCDicL-U (30.89dB/80.64) DCDicL (31.15dB/81.35)

Figure S27: Denoising results on image kodim04 in Kodak24.

Noisy input ($\sigma=50$) DCDicL-U (31.05dB/88.55) DCDicL (31.41dB/89.53)

Figure S28: Denoising results on image 8 in McMaster.



Noisy input ($\sigma=50$) DCDicL-U (32.44dB/92.87) DCDicL (33.63dB/94.76)

Figure S29: Denoising results on image 044 in Urban100.

Noisy input ($\sigma=50$) DCDicL-U (31.27dB/88.66) DCDicL (31.77dB/89.94)

Figure S30: Denoising results on image 057 in Urban100.

Noisy input ($\sigma=50$) DCDicL-U (32.01dB/91.10) DCDicL (32.21dB/91.89)

Figure S31: Denoising results on image 086 in Urban100.

5 Visualization of Adaptive Dictionaries

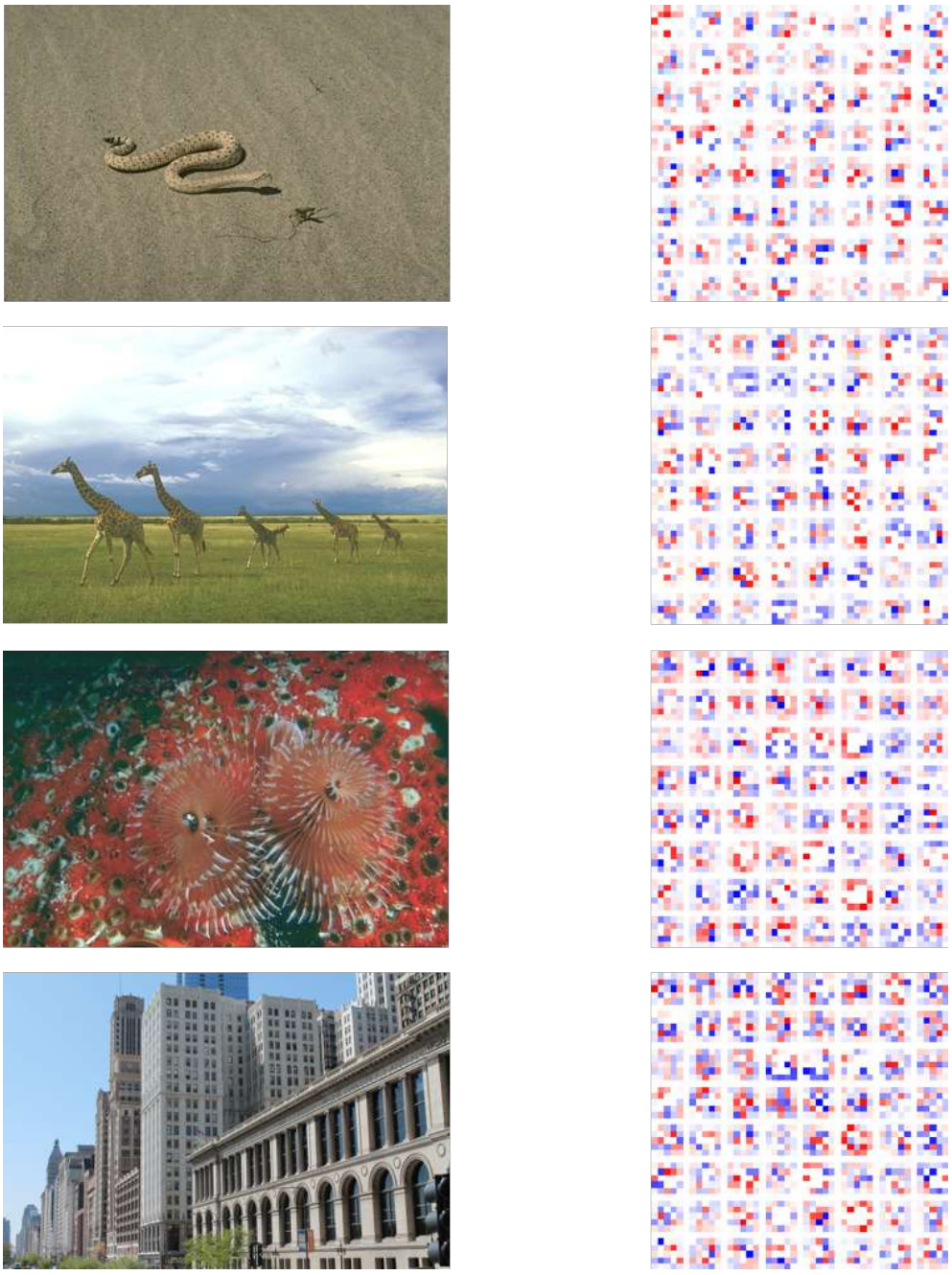


Figure S32: Adaptive dictionaries learned on different images.

References

[1] Hilton Bristow, Anders Eriksson, and Simon Lucey. Fast convolutional sparse coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 391–398, 2013. 1

[2] Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950. 2