Generative Status Estimation and Information Decoupling for Image Rain Removal – Supplementary Material –

Di Lin\textsuperscript{1,1}, Xin Wang\textsuperscript{2,1}, Jia Shen\textsuperscript{1}, Renjie Zhang\textsuperscript{2}, Ruoran Liu\textsuperscript{1}, Miaohui Wang\textsuperscript{3}, Wuyuan Xie\textsuperscript{3}, Qing Guo\textsuperscript{4} and Ping Li\textsuperscript{2,*}

\textsuperscript{1}Tianjin University, China
\textsuperscript{2}The Hong Kong Polytechnic University, Hong Kong
\textsuperscript{3}Shenzhen University, China
\textsuperscript{4}Center for Frontier AI Research, A*STAR, Singapore

\begin{itemize}
\item p.li@polyu.edu.hk
\end{itemize}

1 Experimental Details

We construct SEIDNet based on PyTorch\textsuperscript{1}. There are 26 convolutional layers for extracting the visual feature map from the rainy image. The feature masking contains two convolutional layers. It computes the rain (or object) feature map. The encoder/condition/decoder branch of CVAE (\(V_c\) or \(V_e\)) consists of 16 convolutional layers. There is a pair of batch normalization and ReLU layers between the adjacent convolutional layers. The size of kernels in each convolutional layer is \(3 \times 3\). \(V_{id}\) generates \(3 \times 3\) kernel for deraining each pixel. There are 132 convolutional layers in SEIDNet.

We use Adam solver to optimize the parameters of SEIDNet. We set the initial learning rate to \(5e^{-5}\), which is decayed linearly. We train the network for 5,000 epoches on the dataset, where we randomly

\begin{itemize}
\item \textsuperscript{1}https://pytorch.org/
\end{itemize}


Figure 1: Different strategies of using the status estimation and information decoupling.

We use Adam solver to optimize the parameters of SEIDNet. We set the initial learning rate to \(5e^{-5}\), which is decayed linearly. We train the network for 5,000 epoches on the dataset, where we randomly
select 10 images for constructing each mini-batch. The spatial resolution of the image is $256 \times 256$. We use a GeForce RTX 3090 GPUs for training. Given the testing architecture of SEIDNet, we generate 8 kernel maps, which are aggregated for the deraining on the rainy image.

<table>
<thead>
<tr>
<th>SE</th>
<th>ID</th>
<th>Memory</th>
<th>Time</th>
<th>PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>D</td>
<td>1.0</td>
<td>0.0515</td>
<td>29.43</td>
<td>0.8809</td>
</tr>
<tr>
<td>G</td>
<td>D</td>
<td>1.2</td>
<td>0.0643</td>
<td>29.62</td>
<td>0.8849</td>
</tr>
<tr>
<td>D</td>
<td>G</td>
<td>1.9</td>
<td>0.0704</td>
<td>31.42</td>
<td>0.9172</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>2.1</td>
<td>0.0832</td>
<td>33.22</td>
<td>0.9327</td>
</tr>
</tbody>
</table>

Table 1: We refer to SE, ID, Memory and Time as the status estimation, information decoupling, GPU memory (GB) and testing time (second). D and G indicate the discriminative and generative networks, respectively. The performances are reported on the test set of Rain100H.

2 Ablation Study of SEIDNet

Analysis of Network Components The status estimation and information decoupling of SEIDNet are the core components for computing the pixel-wise statuses and kernels. These components are based on the generative CVAEs. To evaluate the generative power for rain removal, we experiment with using the discriminative convolutional network for the status estimation and information decoupling. We report the results in Table 1 (also see Table 1 of the main paper). In this supplementary material, we illustrate the architectures in Figure 1(a–d).

Various Combinations of Networks In Table 2 (also see Table 2 of the main paper), we compare different strategies of using the discriminative and generative networks for deraining. Similar to the full model of SEIDNet (see Figure 2(a)), we use 132 convolutional layers to construct a single discriminative network for predicting the pixel-wise kernels. The architecture of the discriminative network is illustrated in Figure 2(b).

We combine the discriminative and generative networks, by averaging the kernel maps produced by the discriminative and generative networks. The combined networks are illustrated in Figure 2(c). The trivial combination increases the network parameters but degrades the performances.

Different Ways of Using CVAEs SEIDNet has a pair of CVAEs that model the factorized distributions of the status and the kernel (see Figure 3(c)). We compare SEIDNet with the alternative methods, which use a CVAE without distribution factorization. We list the results in Table 3 (also see Table 3 of the main paper).

First, we experiment with the single CVAE that only takes the visual feature map of the rainy image as the condition. This method directly generates the kernel maps (see Figure 3(a)).

Second, we use the single CVAE to generate the status and kernel maps. Again, this CVAE takes the visual feature map of the rainy image as the only condition.

Table 2: Network means the discriminative (abbreviated as D) or generative (abbreviated as G) network that estimates the kernels for rain removal. The performances are reported on the test set of Rain100H.
Table 3: We refer to $K$ and $R$ as the kernel and status maps. We use different factorizations of probability and report the performances on the test set of Rain100H.

The status and kernel maps are generated by the separate decoder branches. The single CVAE only depends on the training loss of the status map, for implicitly guiding the generation of the kernel map (see Figure 3(b)).

Visualization of Status Maps  We use the normal distribution to generate an array of latent variable maps $\{Z^m | m = 1, ..., N\}$. The decoder of $\mathcal{V}_{se}$ uses feature map $F$, latent variable maps $Z^m$ and $(\mu_f, \sigma_f)$ to generate the status map $\{R^m | m = 1, ..., N\}$. In Figure 6, we compare the visualized status maps, which are estimated from the testing images.

In each status map, a higher (or lower) value means a higher (or lower) probability of predicting the pixel as the rain (or object). For visualization, the pixel in blue (or white) has a higher (or lower) value. In each row of Figure 6, we zoom in some of the regions on the status maps (see the regions in the red rectangles) that are estimated from the identical image. These regions contain different status values for capturing the confusing appearance of rain and object.

Visualization of Kernel Maps  The decoder of $\mathcal{V}_{id}$ uses $F, R^m, Z^m$ and $(\mu_m, \sigma_m)$ to generate the kernel map $\{K^m | m = 1, ..., N\}$. Given various status maps generated by $\mathcal{V}_{se}$, the decoder of $\mathcal{V}_{id}$
Figure 5: (a) We choose the chosen regions in each input image. (b) We show a status map for each input image. (c) For each chosen region, we use t-SNE to embed different counterparts of statuses into the 2D space. Here, each scatter point corresponds to a counterpart of statuses in a chosen region with the same color. (d) We show a kernel map for each input image. (e) For each chosen region, we use t-SNE to embed different counterparts of kernels into the 2D space. Here, each scatter point corresponds to a counterpart of kernels in a chosen region with the same color. (f) The final results achieved by SEIDNet.

Figure 6: Visualization of status maps.

can generate various kernel maps for the same image. In Figure 7, we compare the visualized kernel maps $K^m$, which are estimated from the testing images.

In each row of Figure 7, the sample regions on the kernel maps (see the red rectangles) are estimated from the identical image. Note that these regions correspond to different kernels for deraining. As evidenced in Figure 4 (also see Figure 4 of the main paper), increasing the number of the estimated kernel maps helps to improve the deraining performance.
Table 4: We compare SEIDNet with state-of-the-art methods on the test sets of Rain13K. The performances are reported in terms of PSNR and SSIM.

<table>
<thead>
<tr>
<th>Method</th>
<th>Rain100H</th>
<th>Rain100L</th>
<th>Test100</th>
<th>Test1200</th>
<th>Test2800</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>PReNet [2]</td>
<td>27.02</td>
<td>0.8655</td>
<td>32.61</td>
<td>0.9513</td>
<td>31.54</td>
<td>0.9136</td>
</tr>
<tr>
<td>JORDER [3]</td>
<td>27.43</td>
<td>0.8677</td>
<td>32.42</td>
<td>0.9476</td>
<td>31.44</td>
<td>0.9110</td>
</tr>
<tr>
<td>SPANet [4]</td>
<td>26.88</td>
<td>0.8536</td>
<td>31.26</td>
<td>0.9247</td>
<td>29.93</td>
<td>0.8928</td>
</tr>
<tr>
<td>RCDNet [5]</td>
<td>30.17</td>
<td>0.8876</td>
<td>35.06</td>
<td>0.9603</td>
<td>31.68</td>
<td>0.9294</td>
</tr>
<tr>
<td>CVID [6]</td>
<td>26.25</td>
<td>0.8444</td>
<td>30.53</td>
<td>0.9025</td>
<td>23.23</td>
<td>0.7824</td>
</tr>
<tr>
<td>MPR [7]</td>
<td>30.47</td>
<td>0.8926</td>
<td>36.45</td>
<td>0.9669</td>
<td>30.29</td>
<td>0.9139</td>
</tr>
<tr>
<td>EfDeRain [8]</td>
<td>30.44</td>
<td>0.8954</td>
<td>35.45</td>
<td>0.9645</td>
<td>27.67</td>
<td>0.8874</td>
</tr>
<tr>
<td>SEIDNet</td>
<td>30.56</td>
<td>0.8956</td>
<td>35.37</td>
<td>0.9621</td>
<td>24.87</td>
<td>0.8349</td>
</tr>
<tr>
<td></td>
<td>31.18</td>
<td>0.8993</td>
<td>36.83</td>
<td>0.9657</td>
<td>30.29</td>
<td>0.9148</td>
</tr>
</tbody>
</table>

Analysis on the Correlation between Statuses and Kernels

To analyze the correlation between the status and kernel maps produced by SEIDNet, we employ t-SNE [1] to embed the status and kernel maps into the 2D space for visual analysis.

As shown in Figure 5(a), we choose the regions from the input images. In the chosen regions of each input image, the statuses of rains and objects are very similar (see Figure 5(b)). Thus, for the chosen region of each image, their different counterparts of statuses, which are sampled from the status space, are embedded closely into the 2D space (see Figure 5(c)). Intuitively, the overlapping of statuses in the 2D space likely let deraining kernels for the chosen regions be similar, thus leading to the failure in the deraining on these regions.

Note that the status estimation and information decoupling of SEIDNet are powerful. Though the statuses of the chosen regions are similar, multiple counterparts of status maps still provide the useful information, which helps the information decoupling to estimate the reliable kernels for different regions (see the kernel map in Figure 5(d)). As shown in Figure 5(e), we embed different counterparts of kernels of each chosen region into the 2D space, where the kernels of different regions are relatively far from each other. It helps to yield the satisfactory results in Figure 5(f).
Figure 8: Visual results on the image deraining task. We zoom in the image regions (see the blue rectangles) to compare deraining performance of different methods.

![Image Ground Truth](image1.png)

![Input](image2.png)

![RCDNet](image3.png)

![MPR](image4.png)

![SPDNet](image5.png)

![SEIDNet](image6.png)

![Ground Truth](image7.png)

Table 5: We compare SEIDNet with state-of-the-art methods on the test sets of Snow100K. The performances are reported in terms of PSNR and SSIM.

<table>
<thead>
<tr>
<th>Method</th>
<th>Snow100K-S</th>
<th>Snow100K-M</th>
<th>Snow100K-L</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR</td>
<td>SSIM</td>
<td>PSNR</td>
<td>SSIM</td>
</tr>
<tr>
<td>DuRN-S-P [12]</td>
<td>32.27</td>
<td>0.9497</td>
<td>30.92</td>
<td>0.9398</td>
</tr>
<tr>
<td>Composition GAN [13]</td>
<td>30.43</td>
<td>0.9612</td>
<td>31.21</td>
<td>0.9431</td>
</tr>
<tr>
<td>DesnowNet [14]</td>
<td>32.33</td>
<td>0.9500</td>
<td>30.87</td>
<td>0.9409</td>
</tr>
<tr>
<td>DS-GAN [15]</td>
<td>33.43</td>
<td>0.9641</td>
<td>31.88</td>
<td>0.9570</td>
</tr>
<tr>
<td>HECWNet [16]</td>
<td>33.21</td>
<td>0.9623</td>
<td>32.38</td>
<td>0.9541</td>
</tr>
<tr>
<td>DDMSNet [17]</td>
<td>34.34</td>
<td>0.9445</td>
<td>32.89</td>
<td>0.9330</td>
</tr>
<tr>
<td>RFMPRaLSGAN [18]</td>
<td>33.68</td>
<td>0.9690</td>
<td>30.47</td>
<td>0.9500</td>
</tr>
<tr>
<td>RSRNet [19]</td>
<td>31.54</td>
<td>0.9519</td>
<td>30.52</td>
<td>0.9444</td>
</tr>
<tr>
<td>SEIDNet</td>
<td>35.01</td>
<td>0.9765</td>
<td>33.45</td>
<td>0.9711</td>
</tr>
</tbody>
</table>

Table 6: We compare SEIDNet with other methods on ITS&OTS. The results are listed in terms of PSNR and SSIM.

<table>
<thead>
<tr>
<th>Method</th>
<th>Grid-Net</th>
<th>MSBDN</th>
<th>FFA-Net</th>
<th>AECR-Net</th>
<th>D-Former</th>
<th>SEIDNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR</td>
<td>SSIM</td>
<td>PSNR</td>
<td>SSIM</td>
<td>PSNR</td>
<td>SSIM</td>
</tr>
<tr>
<td>Grid-Net</td>
<td>32.16</td>
<td>0.9836</td>
<td>30.86</td>
<td>0.9820</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSBDN</td>
<td>33.67</td>
<td>0.9850</td>
<td>33.48</td>
<td>0.9820</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFA-Net</td>
<td>36.39</td>
<td>0.9886</td>
<td>33.57</td>
<td>0.9840</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AECR-Net</td>
<td>37.17</td>
<td>0.9901</td>
<td>33.84</td>
<td>0.9837</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D-Former</td>
<td>40.05</td>
<td>0.9960</td>
<td>34.95</td>
<td>0.9840</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEIDNet</td>
<td>40.62</td>
<td>0.9968</td>
<td>35.72</td>
<td>0.9951</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3 Supplementary Results on Rain Removal

In Table 4, we report the performances on the separate test sets (i.e., the test sets of Test100, Test1200 [10], Test2800 [11], Rain100H, and Rain100L) of Rain13K. We also compare the performances of different methods. Each method is trained on the unified training set of Rain13K and tested on the separate test sets. Again, SEIDNet outperforms other methods, showing a strong generalization across different datasets.

In Figures 8 and 9, we provide more visual results on the task of rain removal.
Figure 9: Visual results on real-world dataset. We zoom in the image regions (see the blue rectangles) to compare deraining performance of different methods.

4 Extensive Comparison on Different Tasks

Note that SEIDNet can be extended to various image restoration tasks. To evaluate the generalization of SEIDNet on different tasks, we experiment with using SEIDNet to address the removal of snow, haze, and shadow. We report the performances of different methods in Tables 5, 6, 7, and 8 (also see Tables 7–10 of the main paper). In Figures 14, 15, and 16, we compare the visual results of different methods on the Snow100K, ITS&OTS, ISTD, and ISTD+ datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>Shadow</th>
<th>Non-Shadow</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGAN [26]</td>
<td>9.21</td>
<td>6.27</td>
<td>6.63</td>
</tr>
<tr>
<td>DSC [27]</td>
<td>9.22</td>
<td>6.39</td>
<td>6.67</td>
</tr>
<tr>
<td>DADNet [29]</td>
<td>8.65</td>
<td>6.17</td>
<td>6.57</td>
</tr>
<tr>
<td>CANet [31]</td>
<td>8.86</td>
<td>6.07</td>
<td>6.15</td>
</tr>
<tr>
<td>AEFNet [32]</td>
<td>7.77</td>
<td>5.56</td>
<td>5.92</td>
</tr>
<tr>
<td>CRFormer 33</td>
<td>7.32</td>
<td>5.82</td>
<td>6.07</td>
</tr>
<tr>
<td>SEIDNet</td>
<td>7.47</td>
<td>5.08</td>
<td>5.47</td>
</tr>
</tbody>
</table>

Table 7: We compare SEIDNet with other methods on the test set of ISTD. The performances are reported in term of RMSE.

<table>
<thead>
<tr>
<th>Method</th>
<th>Shadow</th>
<th>Non-Shadow</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST-CGAN [34]</td>
<td>13.4</td>
<td>7.7</td>
<td>8.7</td>
</tr>
<tr>
<td>DeshadowNet [35]</td>
<td>15.9</td>
<td>6.0</td>
<td>7.6</td>
</tr>
<tr>
<td>Mask-GAN [25]</td>
<td>12.4</td>
<td>4.0</td>
<td>5.3</td>
</tr>
<tr>
<td>SP+M-Net [36]</td>
<td>9.7</td>
<td>3.0</td>
<td>4.0</td>
</tr>
<tr>
<td>PMDNet [37]</td>
<td>9.7</td>
<td>3.0</td>
<td>4.0</td>
</tr>
<tr>
<td>AEFNet [32]</td>
<td>6.5</td>
<td>3.8</td>
<td>4.2</td>
</tr>
<tr>
<td>CRFormer 33</td>
<td>5.9</td>
<td>2.9</td>
<td>3.4</td>
</tr>
<tr>
<td>SEIDNet</td>
<td>6.4</td>
<td>3.4</td>
<td>3.9</td>
</tr>
</tbody>
</table>

Table 8: We compare SEIDNet with other methods on the test set of ISTD+. The performances are reported in term of RMSE.
5 Limitation

Quality of Kernels from Feature Masking  As evidenced in Table 2, the trivial combination of the discriminative and generative networks increases the network parameters but degrades the performances. It demonstrates that the inaccurate kernels, which are produced by the discriminative network, can harm the performance of the generative network. Though SEIDNet independently generates the kernels for rain removal, it still needs to learn from the kernels that are computed by the discriminative network (i.e., the feature masking in Figure 10) as a reference during training.

Figure 10: The training architecture of SEIDNet. The architecture has the (a) status estimation and (b) information decoupling. We use (c) the CVAE for learning the status space, (d) the feature masking for yielding the rain and object feature maps, and (e) the CVAE for learning the kernel space.

In this section, we evaluate the impact of the kernels, which are produced by the discriminative network, on the deraining performance of SEIDNet. This is done by changing the number of layers of the feature masking. We choose the layer number from the set \{12, 15, 18, 21, 24, 27\}, where the feature masking has 27 convolutional layers in the full model. By decreasing the layer number, we reduce the computational overheads (i.e., GPU memory and floating point of operations (FLOPs)), as reported in Figure 11(a–b). On the other hand, fewer layers degrade the deraining performances (see PSNR and SSIM in Figure 11(c–d)). We conjecture that fewer layers weaken the learning capability of the feature masking. In this case, the feature masking likely produces the problematic kernels, which are embedded into the kernel space by CVAE. It misleads the construction of the kernel space, where the problematic kernels are likely sampled for rain removal. We show the visual results of...
using different numbers of layers for the feature masking in Figure 12, where the failure cases are yielded by using fewer layers for computing the reference kernels.

The limitation of our methods motivates the future investigation of how to use CVAE to learn more effective kernels, without depending on the extra and sensitive network for computing the reference kernels.

**Errors on Some Bright Objects**  It should be noted that bright objects are extremely similar to the appearances of the rain streaks. The examples of the confusing bright objects and rain streaks can be found in Figure 6: (1) the top-right water region in the first row; (2) the second-left person in white cloth in the fourth row; (3) the bright regions of the wheels in the last row. There are pixels of these bright objects misunderstood as rain streaks, as illustrated in the corresponding status maps. Yet, we use multiple status maps, which provide more differentiable information for separating the bright objects from the rain streaks. Thus, the final deraining results are reasonable. In future work, we plan to further improve the deraining results of the bright objects, while relying on fewer status maps for saving computation.

6  **Negative Societal Impacts**

Our approach can be broadly applied in many scenarios (e.g., autonomous vehicles and video surveillance). One should be cautious of the problematic results, which may give rise to the infringement of privacy or economic interest.

7  **Analysis of the Average Kernels**

We sample multiple kernels, which are averaged for processing each pixel in the image. Here, we justify the effectiveness of the average kernels in terms of processing the confusing rain streaks and object textures.

Figure 13: The difference of average kernels of the confusing patch pairs, with different. We change the number of sampled kernels (1, 2, 4, 8, 16, 32) for computing the average kernel.

We manually select 100 pairs of image patches from Rain100L and Rain1400 (see Figure 13(a)). Each pair of image patches contain the rain streaks (top row) and the object textures (bottom row),
respectively. Each pair of rain and object patches are visually similar. The typical discriminative networks (i.e., EfDeRain, SPDNet, and MPR) compute similar kernels for the confusing rain and object, thus yielding unsatisfactory results in these confusing patches. In contrast, we sample more kernels for each image patch, where the sampled kernels are averaged. We compute the difference (L1 distance) between the average kernels of each pair of confusing rain and object patches. We accumulate and average the differences of all pairs. We change the number of sampled kernels for computing the average kernel. In Figure 13(b), we report the difference of average kernels of the confusing patch pairs. With more sampled kernels, the average kernels are more specific for processing the confusing rain and object patches.

8 Code Segment

We release the implementation of SEIDNet via https://github.com/wxxx1025/SEIDNet.

9 Traing and testing pseudo-code of SEIDNet

Algorithm 1: Training pseudo-code of SEIDNet

1: epoch = 1;
2: while epoch $\leq$ max_epoch do
3: Input rainy image $I$, object layer $O$ for estimating status map $R$ via Eq. (3);
4: Extract feature map $F$ from given rainy image $I$;
5: Pass feature map $F$ and status map $R$ to CVAE $\mathcal{V}_{sc}$;
6: Estimate mean value maps and standard deviation maps: $[\mu_r, \sigma_r] \leftarrow \text{encoder}([F, R])$,
   $[\mu_f, \sigma_f] \leftarrow \text{condition}(F)$ in Eq. (4);
7: Input $F$, $Z$ and $(\mu_r, \sigma_r)$ to the decoder of $\mathcal{V}_{sc}$ to generate $R'$ via Eq. (4);
8: Calculate status estimation loss $L_{se}$ via Eq. (5);
9: Estimate kernel maps $K_r$ and $K_o$ from $F$, $R$ via Eq. (6);
10: Estimate kernel map: $K \leftarrow R \odot K_r + (1 - R) \odot K_o$ in Eq. (7);
11: Pass $K$, $F$ and $R$ to CVAE $\mathcal{V}_{id}$;
12: Calculate mean value maps and standard deviation maps $[\mu_k, \sigma_k] \leftarrow \text{encoder}([K, F, R])$,
   $[\mu_c, \sigma_c] \leftarrow \text{condition}([F, R])$ in Eq. (8);
13: Generate kernel map from kernel space: $K' \leftarrow \text{decoder}([F, R, \mu_k + \sigma_k \odot Z])$ in Eq. (8);
14: Calculate information decoupling loss $L_{id}$ via Eq. (9);
15: Employ $K'$ and $I$ to estimate object layer $O'$ via Eq. (10);
16: Calculate deraining loss $L_{de}$ via Eq. (10);
17: Calculate overall loss $L$ via Eq. (11);
18: Update network weights;
19: if epoch $\geq lr_{decrease}_{epoch}$ then
20:   Adjust learning rate;
21: end if
22: epoch $\leftarrow$ epoch + 1;
23: end while
Algorithm 2: Testing pseudo-code of SEIDNet

```plaintext
1: \textit{index} = 1
2: \textbf{while} \textit{index} \leq \text{len(test\_dataset)} \textbf{do}
3: \hspace{1em} Extract feature map $F$ from given rainy image $I$;
4: \hspace{1em} Pass feature map $F$ to CVAE $V_{\text{ae}}$;
5: \hspace{1em} Calculate mean value map and standard deviation map: $[\mu_f, \sigma_f] \leftarrow \text{condition}(F)$ in Eq. (12);
6: \hspace{1em} \textbf{while} $m \leq N$ \textbf{do}
7: \hspace{2em} Generate status map from constructed status space: $R^m \leftarrow \text{decoder}(F, \mu_f + \sigma_f \odot Z^m)$ in Eq. (12);
8: \hspace{2em} Pass $F$, $R^m$ and $Z^m$ to CVAE $V_{\text{id}}$;
9: \hspace{2em} Calculate mean value map and standard deviation map: $\mu_c^m, \sigma_c^m \leftarrow \text{condition}(F, R^m)$ in Eq. (12);
10: \hspace{2em} Generate kernel map from constructed kernel space: $K^m \leftarrow \text{decoder}(F, R^m, \mu_c^m + \sigma_c^m \odot Z^m)$ in Eq. (12);
11: \hspace{1em} $m \leftarrow m + 1$
12: \hspace{1em} \textbf{end while}
13: \hspace{1em} Estimate $K^u$ via Eq. (13)
14: \hspace{1em} $O = K^u \odot I$
15: \hspace{1em} $\textit{index} \leftarrow \textit{index} + 1$
16: \textbf{end while}
```

References


Figure 14: Visual results on the image desnow task. We zoom in the image regions (see the blue rectangles) to compare desnow performance of different methods.


Figure 15: Visual results on the image dehaze task. We zoom in the image regions (see the blue rectangles) to compare dehaze performance of different methods.

Figure 16: Visual results on the image deshadow task. We zoom in the image regions (see the blue rectangles) to compare deshadow performance of different methods.


