Supplementary Material of "Semi-Supervised Video Deraining with Dynamic Rain Generator"

Zongsheng Yue[1], Jianwen Xie[2], Qian Zhao[1], Deyu Meng[1,3,*]
[1]Xi'an Jiaotong University, Xi'an, China
[2]Baidu Research, Bellevue, USA
[3]The Macau University of Science and Technology, Macau, China
zsyzam@gmail.com, jianwen@ucla.edu, timmy.zhaoqian@gmail.com, dymeng@mail.xjtu.edu.cn
https://github.com/zsyOAOA/S2VD

Abstract

In this supplementary material, we provide more details on how to apply the proposed dynamic generator to imitate any pre-given video purely containing rain layer and the corresponding rain generation experiments to evaluate its effectiveness. Besides, more analysis about the model capacity and running time of our proposed video method are presented.

1. Dynamic rain generator

In the Sec. 3.1 of the main text, we design the following dynamic rain generator, i.e.,

$$\boldsymbol{s}_t = F(\boldsymbol{s}_{t-1}, \boldsymbol{z}_t; \boldsymbol{\alpha}), \tag{1}$$
$$\mathcal{R}_t = H(\boldsymbol{s}_t; \boldsymbol{\beta}), \tag{2}$$

where

$$\boldsymbol{z}_t \sim \mathcal{N}(0, \boldsymbol{I}),\ \boldsymbol{s}_0 \sim \mathcal{N}(0, \boldsymbol{I}). \tag{3}$$

The detailed explanations about Eqs. (1)-(3) can be seen in the main text. By denoting $\boldsymbol{z} = \{\boldsymbol{z}_t\}_{t=1}^n$ and $\boldsymbol{\theta} = \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$, Eqs. (1) and (2) can be simply written together as follows,

$$\mathcal{R} = G(\boldsymbol{s}_0, \boldsymbol{z}; \boldsymbol{\theta}). \tag{4}$$

1.1. Maximum likelihood estimation

Given any observed video $\mathcal{R}^o$ purely containing rain layer, we assume that it is generated by the aforementioned generator with an additional residual term $\mathcal{E}$, i.e.,

$$\begin{aligned} \mathcal{R}^o &= G(\boldsymbol{s}_0, \boldsymbol{z}; \boldsymbol{\theta}) + \mathcal{E}, \\ \mathcal{E}_{ijt} &\sim \mathcal{N}(0, \sigma^2), \end{aligned} \tag{5}$$

*Corresponding author.

Algorithm 1 Inference and learning procedure for the dynamic rain generator

```
Input: Observe data $\mathcal{R}^o$, number of Langevin steps $l$.
Output: the generator parameters $\boldsymbol{\theta}$.
1: Initialize $\boldsymbol{\theta}$.
2: while not converged do
3:     E-Step: Run $l$ steps of Langevin dynamics to sample
       $\boldsymbol{z}$ following Eq. (7).
4:     M-Step: Update $\boldsymbol{\theta}$ by Eq. (10).
5: end while
```

where $\mathcal{E}_{ijt}$ denotes the element with index (i, j, t) in $\mathcal{E}$.

According to Eq. (5), our goal turns to maximize the log-likelihood $p(\mathcal{R}^o; \boldsymbol{\theta})$ w.r.t. the parameters $\boldsymbol{\theta}$, i.e.,

$$\begin{aligned} \max_{\boldsymbol{\theta}} \log p(\mathcal{R}^o; \boldsymbol{\theta}) &= \log \int p(\mathcal{R}^o | \boldsymbol{z}) p(\boldsymbol{z}) \,\mathrm{d}\boldsymbol{z} \\ &= \log \int \mathcal{N}\left(G(\boldsymbol{s}_0, \boldsymbol{z}; \boldsymbol{\theta}), \sigma^2 \boldsymbol{I}\right) p(\boldsymbol{z}) \,\mathrm{d}\boldsymbol{z} \\ &\triangleq \mathcal{L}(\mathcal{R}^o; \boldsymbol{\theta}), \end{aligned} \tag{6}$$

where $p(\boldsymbol{z})$ is defined in Eq. (3).

1.2. Inference and learning algorithm

Inspired by the technology of alternative back-propagation through time [6], a Monte Carlo EM [2] algorithm is designed to solve the problem of Eq. (6), which consists of two alternative steps, i.e., one expectation step and one maximization step. The expectation step aims to sample latent variable $\boldsymbol{z}$ from its posterior $p(\boldsymbol{z}|\mathcal{R}^o)$, and the next maximization step updates the parameters $\boldsymbol{\theta}$ based on current sampled $\boldsymbol{z}$.

E-Step: Let $\boldsymbol{\theta}^{\text{old}}$ and $p_{\text{old}}(\boldsymbol{z}|\mathcal{R}^o)$ denote the current parameters $\boldsymbol{\theta}$ and the posterior under them, we can sample $\boldsymbol{z}$ from

$p_{\text{old}}(\boldsymbol{z}|\mathcal{R}^o)$ using the Langevin dynamic [3]:

$$\begin{aligned}\boldsymbol{z}^{(\tau+1)} &= \boldsymbol{z}^{(\tau)} + \frac{\delta^2}{2}\left[\frac{\partial}{\partial \boldsymbol{z}}\log p_{\text{old}}(\boldsymbol{z}|\mathcal{R}^o)\right]\Bigg|_{\boldsymbol{z}=\boldsymbol{z}^{(\tau)}} + \delta\boldsymbol{\xi}^{(\tau)} \\ &= \boldsymbol{z}^{(\tau)} - \frac{\delta^2}{2}\left[\frac{\partial}{\partial \boldsymbol{z}} g(\boldsymbol{z})\right]\Bigg|_{\boldsymbol{z}=\boldsymbol{z}^{(\tau)}} + \delta\boldsymbol{\xi}^{(\tau)}, \end{aligned} \tag{7}$$

where

$$g(\boldsymbol{z}) = \frac{1}{2\sigma^2}\left\|\mathcal{R}^o - G\left(\boldsymbol{z}, \boldsymbol{s}_0; \boldsymbol{\theta}^{\text{old}}\right)\right\|_2 + \frac{1}{2}\|\boldsymbol{z}\|_2, \tag{8}$$

τ indexs the time step for Langevin dynamics, δ denotes the step size. And $\boldsymbol{\xi}^{(\tau)}$ is the Gaussian white noise, which is added to prevent trapping into local modes. A key point in Eq. (7) is $\frac{\partial}{\partial \boldsymbol{z}}\log p_{\text{old}}(\boldsymbol{z}|\mathcal{R}^o) = \frac{\partial}{\partial \boldsymbol{z}}\log p_{\text{old}}(\mathcal{R}^o, \boldsymbol{z})$, and the right term can be easily calculated.

In practice, for the purpose of avoiding high computational cost of MCMC, Eq. (7) starts from the previous updated results of $\boldsymbol{z}$. As for the initialized state vector $\boldsymbol{s}_0$ of Eq. (5), we also sample it together with $\boldsymbol{z}$ using the Langevin dynamics.

M-Step: Denote the sampled latent variable in E-Step as $\tilde{\boldsymbol{z}}$, M-Step aims to maximize the approximate upper bound w.r.t. parameters $\boldsymbol{\theta}$ as follows:

$$\begin{aligned}\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}) &= \int p_{\text{old}}(\boldsymbol{z}|\mathcal{R})\log p(\mathcal{R}^o, \boldsymbol{z}; \boldsymbol{\theta})\,\mathrm{d}\boldsymbol{z} \\ &\approx \log p(\mathcal{R}^o, \tilde{\boldsymbol{z}}; \boldsymbol{\theta}).\end{aligned} \tag{9}$$

Equivalently, Eq. (9) can be further rewritten as the following minimization problem, i.e.,

$$\min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\left\|\mathcal{R}^o - G\left(\tilde{\boldsymbol{z}}, \boldsymbol{s}_0; \boldsymbol{\theta}\right)\right\|_2.$$

Naturally, we can update $\boldsymbol{\theta}$ by gradient descent based on the back-propagation (BP) algorithm [5] as follows,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\frac{\partial}{\partial \boldsymbol{\theta}}\hat{\mathcal{L}}(\boldsymbol{\theta}), \tag{10}$$

where η denotes the step size. And the detailed steps of our entire algorithm are listed in Algorithm 1.

2. Rain Generation Experiments

2.1. Evaluation on Rain Generation task

Given any video purely containing rain layer, our proposed dynamic generator is able to learn the regular generative rules underlying it as shown in Sec. 1 of this supplementary material. With the trained generator, we can not only recover the original rain video but also generate many other ones. To verify this point, one rain layer video synthesized by commercial Adobe After Effects[1] is downloaded from YouTube as source videos. We learn a generator for this video, and the visualization of recovered rain videos by our generator in different iterations are displayed in Fig. 1. It can be seen that our generator is able to quickly recover the rain layers only with 20 iterations, which indicates its powerful capabilities in this task.

[1]https://www.adobe.com/products/aftereffects.html

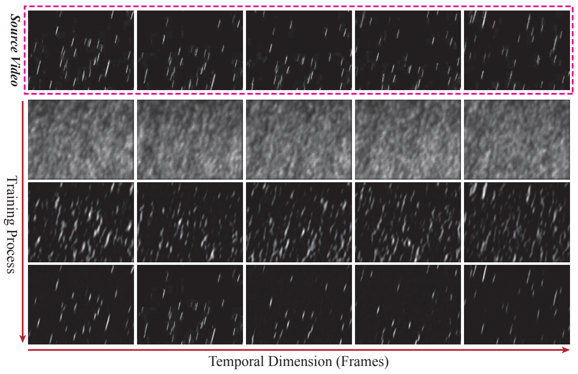

Figure 1. Illustation of the source and recovered rain videos. From top to bottom: the 1st row is the source rain video, and the 2-4th rows are the recovered ones by our generator after 3, 10 and 20 iterations. From left to right: 5 adjacent frames in each video.

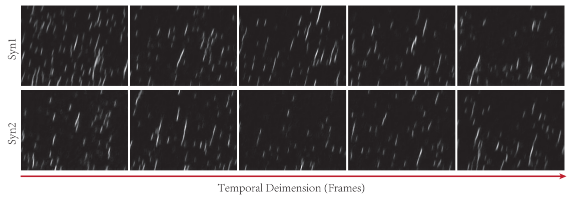

Figure 2. Two typical examples of generated rain videos by our dynamic generator, and each row corresponds to one group of different initializations for $\boldsymbol{s}_0$ and $\{\boldsymbol{z}_t\}$.

Additionally, two synthesized rain videos by the learned generator are shown in Fig. 2. To generate such videos, we only need to randomly initialize the state variable $\boldsymbol{s}_0$ and the innovation vectors $\{\boldsymbol{z}_t\}$ in Eq. (1) from Gaussian distribution, and then follow Eqs. (1) and (2) to mimic each image frame one by one. The vivid rain videos shown in Fig. 2 indicate that our generator indeed captures the general generative laws behind the source video. Therefore, it is rational to fit the rain layers by such dynamic generator in our proposed deraining model in Sec. 3.1 of the main text.

2.2. User Study

While we have displayed the synthesized rain layers by our dynamic generator in Fig. 2, now a user study is further conducted to quantitatively evaluate their realism. Three current widely-used benchmark data sets are considered as compared methods, including *RainSynComplex25* [4], *RainSynLigh25* [4] and *NTURain* [1]. The rain layers of them are obtained as follows,

$$\mathcal{R} = \max_c(\mathcal{Y} - \mathcal{X}), \tag{11}$$

Table 1. Quantitative results of user study experiments on different rain video clips.

Metrics	Methods				
	RainSynComplex25	*RainSynLigh25*	*NTURain*	Adobe	Ours
Rating↑	1.30 ± 0.66	2.72 ± 0.98	3.49 ± 1.02	4.02 ± 0.84	4.04 ± 0.82
Realism↑	26.04	54.34	69.81	80.38	80.75

Table 2. The number of model parameters (K) and running time (s) of different methods.

Metrics	Methods				
	DDN	PReNet	SpacCNN	SLDNet	S2VD
# Parameters (K)	57	169	477	166302	525
Time (s)	0.035	0.187	3.632	2.268	0.032

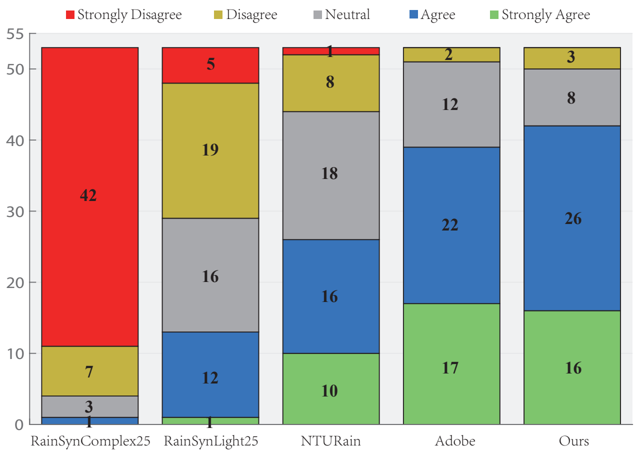

Figure 3. User study of rain realism. The y-axis represents the ratings to the statement *Rain in videos looks realistic*. Our generated rain layers are even a little better than that of the commercial Adobe After Effects.

where $\mathcal{Y}$ and $\mathcal{X}$ denote the rainy video and the corresponding clean video, respectively, $\max_c(\cdot)$ is the element-wise maximized operation along the channel dimension (RGB). Besides, since our generator is trained on one source video downloaded from YouTube, which are synthesized by the commercial Adobe After Effects[2], thus we also regard it as one compared method.

For visual comparisons, some typical image frames of these four rain videos and the synthesized rain video by our generator are shown in Fig. 4. To manually evaluate their realism, we present these five video clips in a random order, each with 100 continuous frames, to 53 recruited participants, and then ask each of them to rate how real every video is, using a 5-point Likert scale. Therefore, we finally get 53 ratings for each video clips as shown in Fig. 3. And the corresponding mean ratings and realism scores are list in Table 1, in which the realism scores are calculated by converting the ratings to range [0,1]. It can be easily seen that the video clips generated by our proposed generator achieve the best results, even a little better than that of the commercial Adobe After Effects software, which indicates the effectiveness of such dynamic rain generator. Therefore, it is rational to use this generator to fit the rain layers in our proposed video deraining method of the main text.

[2] `https://www.adobe.com/products/aftereffects.html`

3. Model capacity and running time of S2VD

In this part, we compare the model capacity (number of model parameters) and running time of different deep learning (DL)-based methods. For the number of parameters of the proposed S2VD, we only consider the parameters in the derainer $f(\cdot; W)$, since only the derainer is necessary in the testing phase after training. The running time evaluation was performed on a computer with 6-cores Inter(R) Core(TM) i7-8700K CPU (3.3GHz) and a Nvidia GTX 1080Ti GPU. Specifically, it is tested on a rainy video that contains 60 image frames with spatial size 480×640, and the average time on each frame is regarded as the running time for each method. And the time for data transfer between CPU and GPU is not counted during calculation.

The detailed results is listed in Table 2. It can be easily observed that: 1) On the whole, the video deraining methods have more parameters than the single image deraining methods. Comparing with two state-of-the-art video deraining methods, the number of parameters of our S2VD is very close to SpacCNN and about 300 times smaller than SLDNet. 2) As for the running time, our S2VD is at least 70 times faster than SLDNet and SpacCNN, which is mainly because of the simple architecture of the derainer network. No matter considering the model capacity or running time, S2VD is very competitive for real applications.

References

[1] Jie Chen, Cheen-Hau Tan, Junhui Hou, Lap-Pui Chau, and He Li. Robust video content alignment and compensation for rain removal in a cnn framework. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6286–6295, 2018. 2

[2] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. 1

[3] Paul Langevin. On the theory of brownian motion. 1983. 2

[4] Jiaying Liu, Wenhan Yang, Shuai Yang, and Zongming Guo. Erase or fill? deep joint recurrent rain removal and reconstruction in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3233–3242, 2018. 2

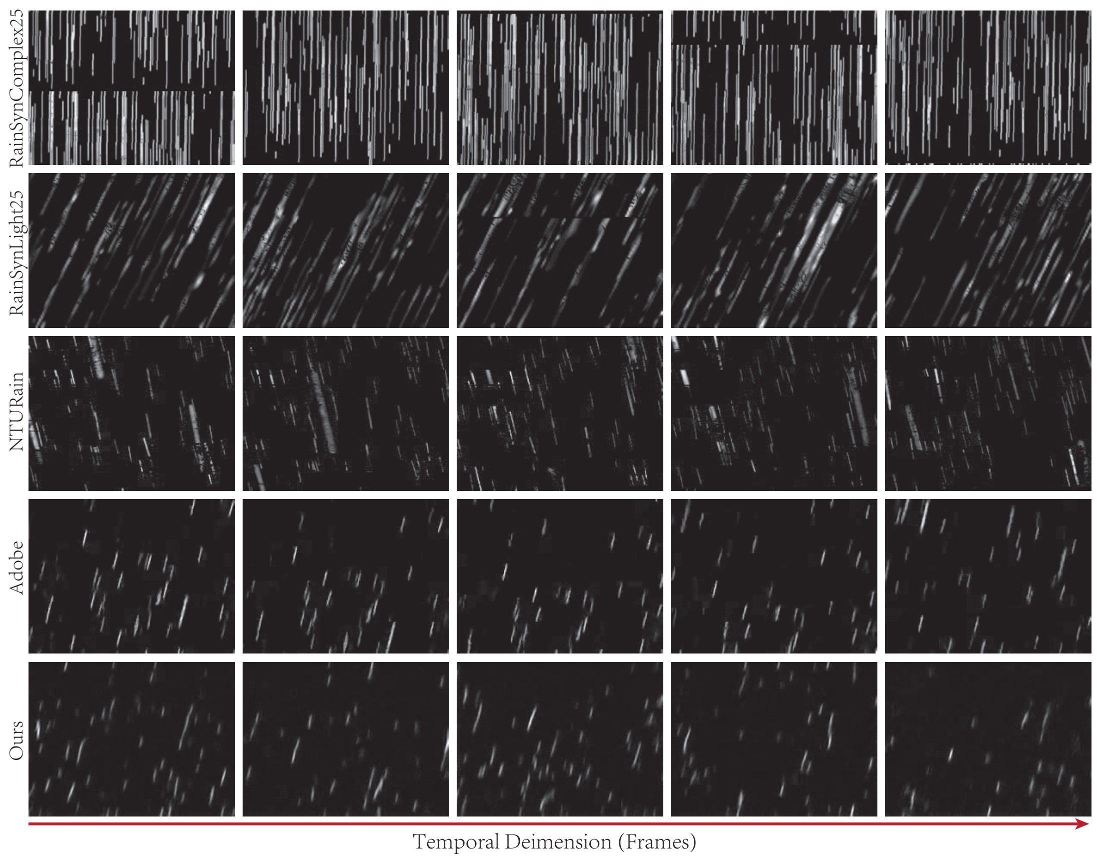

Figure 4. Visualization of the rain layers in different data sets. From top to bottom: the rain layers contained in *RainSynComplex25*, *RainSynLigh25* and *NTURain*, the rain layers generated by Adobe After Effects and our trained generator.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 2

[6] Jianwen Xie, Ruiqi Gao, Zilong Zheng, Song-Chun Zhu, and Ying Nian Wu. Learning dynamic generator model by alternating back-propagation through time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5498–5507, 2019. 1