Lightweight Pyramid Networks for Image Deraining

Xueyang Fu, Borong Liang, Yue Huang, Xinghao Ding, and John Paisley

***Abstract*— Existing deep convolutional neural networks (CNNs) have found major success in image deraining, but at the expense of an enormous number of parameters. This limits their potential applications, e.g., in mobile devices. In this paper, we propose a lightweight pyramid networt (LPNet) for single-image deraining. Instead of designing a complex network structure, we use domain-specific knowledge to simplify the learning process. In particular, we find that by introducing the mature Gaussian–Laplacian image pyramid decomposition technology to the neural network, the learning problem at each pyramid level is greatly simplified and can be handled by a relatively shallow network with few parameters. We adopt recursive and residual network structures to build the proposed LPNet, which has less than 8K parameters while still achieving the state-of-the-art performance on rain removal. We also discuss the potential value of LPNet for other low- and high-level vision tasks.**

***Index Terms*— Deep convolutional neural network (CNN), image pyramid, lightweight networks, rain removal, residual learning.**

I. Introduction

RAIN impacts not only human visual perception but also computer vision systems, such as self-driving vehicles and surveillance systems. Due to the effects of light refraction and scattering, objects in an image are easily blurred and blocked by individual rain streaks. When facing heavy rainy conditions, this problem becomes more severe due to the increased density of rain streaks. Since most of the existing computer vision algorithms are designed based on the assumption of clear inputs, their performance is easily degraded by rainy weather. Thus, designing effective and efficient algorithms for rain streak removal is a significant problem with many downstream uses. Fig. 1 shows an example of our proposed lightweight pyramid network (LPNet) and the corresponding parameter savings over the next best lightweight network.

(a) (b)

Fig. 1. Deraining example of our LPNet for single-image deraining. The whole network only contains **7548** parameters, a 50% decrease on the best lightweight comparison. (a) Rainy image. (b) Our result.

Depending on the input data, rain removal algorithms can be categorized into video-based and single-image-based methods.

A. Video-Based Methods

We first briefly review the rain removal methods in a video, which was the major focus in the early stages of this problem. These methods use both spatial and temporal information from video. The first study on video deraining removed rain from a static background using average intensities from the neighboring frames [1]. Other methods focus on deraining in the Fourier domain [2], using Gaussian mixture models (GMMs) [3], low-rank approximations [4], and via matrix completions [5]. Ren *et al.* [6] divide rain streaks into sparse ones and dense ones, then a matrix decomposition based algorithm is proposed for deraining. More recently, Wei *et al.* [7] propose a patch-based mixture of Gaussians for rain removal in video. Although these methods work well, they require temporal content of video. In this paper, we, instead, focus on the single-image deraining problem.

B. Single-Image Methods

Since information is drastically reduced in individual images, single-image deraining is a much more difficult problem. Methods for addressing this problem employ kernels [8], low-rank approximations [4], [9], and dictionary learning [10]–[13]. In [8], rain streaks are detected and removed by using kernel regression and a non-local mean filtering. Kang *et al.* [10] decompose a rainy image into its low- and high-frequency components. The high-frequency part is processed to extract and remove rain streaks by using sparse-coding-based dictionary learning. In [11], a self-learning method is proposed to automatically distinguish rain streaks from the

Manuscript received May 2, 2018; revised November 7, 2018 and February 21, 2019; accepted June 28, 2019. Date of publication July 22, 2019; date of current version June 2, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61571382, Grant 81671766, Grant 61571005, Grant 81671674, Grant 61671309, and Grant U1605252, in part by the Fundamental Research Funds for the Central Universities under Grant 20720160075 and Grant 20720180059, in part by the CCF-Tencent open fund, and in part by the Natural Science Foundation of Fujian Province of China under Grant 2017J01126. *(Corresponding author: Xinghao Ding.)*

X. Fu is with the Fujian Key Laboratory of Sensing and Computing for Smart City, School of Information Science and Engineering, Xiamen University, Xiamen 361005, China, and also with the School of Information Science and Technology, University of Science and Technology of China, Hefei 230026, China.

B. Liang, Y. Huang, and X. Ding are with the Fujian Key Laboratory of Sensing and Computing for Smart City, School of Information Science and Engineering, Xiamen University, Xiamen 361005, China (e-mail: dxh@xmu.edu.cn).

J. Paisley is with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA, and also with the Data Science Institute, Columbia University, New York, NY 10027 USA.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2926481

high-frequency part. A discriminative sparse coding method is proposed in [12]. By forcing the coefficient vector of rain layer to be sparse, the objective function is solved to separate background and rain streaks. Other methods utilize mixture models [14] and local gradients [15] to model and then remove rain streaks. By utilizing GMMs, Li *et al.* [14] explore patch-based priors for both the clean and rain layers. The GMM prior for background layers is learned from natural images, while that for rain streaks layers is learned from rainy images. In [15], three new priors are defined by exploring local image gradients. The priors are used to model the objective function which is solved by using alternating direction method of multipliers (ADMMs).

Deep learning has also been introduced for this problem. Learning-based methods and convolutional neural networks (CNN) have proven useful for a variety of high-level vision tasks [16]–[23], as well as various image processing problems [24]–[28]. In [29], a related work based on deep learning is introduced to remove static raindrops and dirt spots from pictures taken through windows. Our previous CNN-based method for removing dynamic rain streaks is introduced in [30]. Here, the authors build a relative shallow network with three layers to extract features of rain streaks from the high-frequency content of a rainy image. Based on the introduction of an effective strategy for training very deep networks [20], two deeper networks are proposed based on image residuals [31] and multi-scale information [32], [33]. Zhang *et al.* [34] utilize the generative adversarial framework to further enhance the textures and improve the visual quality of derained results. Recently, in [35], a density-aware multi-stream densely connected CNN is proposed for joint rain density estimation and deraining. This method can automatically generate a rain density label, which is further utilized to guide rain streaks removal.

1) Our Contributions: Although very deep networks achieve excellent performance on single-image deraining, a main drawback that potentially limits their application in mobile devices, automatic driving, and other computer vision tasks is their huge number of parameters. As a networks become deeper, more storage space is required [36]. To address this issue, we propose a LPNet, which contains fewer than 8K parameters, with the single image rain removal problem in mind. Instead of designing a complex network structure, we use domain-specific knowledge to simplify the learning process. Specifically, we first adopt Laplacian pyramids to decompose a degraded/rainy image into different levels. Then we use recursive and residual networks to build a subnetwork for each level to reconstruct Gaussian pyramids of derained images. A specific loss function is selected for training each subnetwork according to its own physical characteristics and the whole training is performed in a multi-task supervision. The final recovered image is the bottom level of the reconstructed Gaussian pyramid.

The main feature of our LPNet approach is to use the mature Gaussian–Laplacian image pyramid technique [37] to transform one hard problem into several easier subproblems. In other words, since the Laplacian pyramid contains different levels that can differentiate large scale edges from small scale details, one can design simple and lightweight subnetwork to handle each level in a divide-and-conquer way. The contributions of our paper are summarized as follows.

1) We show how to utilize domain-specific knowledge to drive deep learning for the tough single-image deraining problem. By combining the classical Gaussian–Laplacian pyramid technique with CNN, a simple network structure with few parameters and relative shallow depth is sufficient for excellent performance. To our knowledge, the resulting easy-to-implement network is far more lightweight (in terms of parameters) among deep networks with good deraining performance.
2) Due to the multi-scale decomposition of the Gaussian–Laplacian pyramid, the spatial scales of all pyramid levels are constrained. Therefore, the LPNet we train on a limited-sized rain streaks can also be adapted to other sizes that have never been seen before. In other words, although LPNet is trained on synthetic data by necessity, it still generalizes well to real-world images.
3) We discuss how LPNet can be applied to other fundamental low- and high-level vision tasks in image processing. We also show how LPNet can improve downstream applications such as object recognition.

II. Lightweight Pyramid Network for Deraining

In Fig. 2, we show our proposed LPNet for single-image deraining. To summarize at a high level, we first decompose a rainy image into a Laplacian pyramid and build a subnetwork for each pyramid level. Then, each subnetwork is trained with its own loss function according to the specific physical characteristics of the data at that level. The network outputs a Gaussian pyramid of the derained image. The final derained result is the bottom level of the Gaussian pyramid.

A. Motivation

Since rain streaks are blended with object edges and the background scene, it is hard to directly learn the deraining function in the image domain [30]. To simplify the problem, it is natural to train a network on the high-frequency information in images, which primarily contain rain streaks and edges without background interference. Based on this motivation, Fu *et al.* [30], [31] use the guided filter [38] to obtain the high-frequency component of an image as the input to a deep network, which is then derained and fused back with the low-resolution information of the same image. However, these two methods fail when very thick rain streaks cannot be extracted by the guided filter. Inspired by this decomposition idea, we instead build a lightweight pyramid of networks to instead simplify the learning processing and reduce the number of necessary parameters as a result.

B. Stage 1: Laplacian Pyramid

We first decompose a rainy image $\mathbf{X}$ into its Laplacian pyramid, which is a set of images L with N levels

$$L_n(\mathbf{X}) = G_n(\mathbf{X}) - \text{upsample}(G_{n+1}(\mathbf{X})) \quad (1)$$

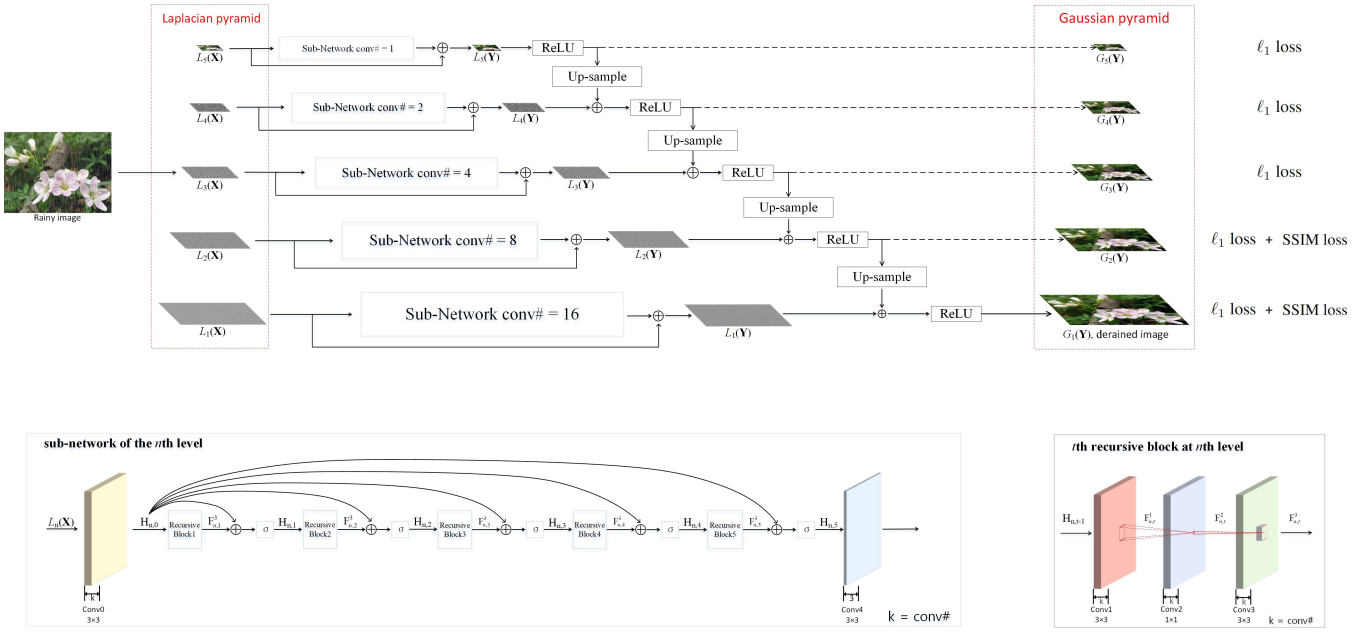


Fig. 2. Proposed structure of our deep lightweight pyramid of networks based on Gaussian–Laplacian image pyramids. The bottom level of the reconstructed Gaussian pyramid is the final derained image. All notations correspond to (1)–(8).

where G_n is the Gaussian pyramid, $n = 1, \ldots, N-1$. The function $G_n(\mathbf{X})$ is computed by downsampling $G_{n-1}(\mathbf{X})$ using a Gaussian kernel, with $G_1(\mathbf{X}) = \mathbf{X}$ and $L_N(\mathbf{X}) = G_N(\mathbf{X})$.

The reasons we choose the classical Laplacian pyramid to decompose the rainy image are fourfold.

1) The background scene can be fully extracted at the top level of L_n while the other levels contain rain streaks and details at different spatial scales. Thus, the rain interference is removed and each subnetwork only needs to deal with high-frequency components at a single scale.
2) This decomposition strategy will allow the network to take advantage of the sparsity at each level, which motivates many other deraining methods [8], [11], [30], to simplify the learning problem. However, unlike previous deraining methods that use a single-scale decomposition, LPNet performs a multi-scale decomposition using Laplacian pyramids.
3) As shown in Fig. 3, compared with the image domain, deep learning at each pyramid level is more like an identity mapping (e.g., the top row is more similar to the middle row, as evident in the bottom row) which is known to be the situation where residual learning (ResNet) excels [20].
4) The Laplacian pyramid is a mature algorithm with low computation cost. Most calculations are based on convolutions (Gaussian filtering) which can be easily embedded into existing systems with GPU acceleration.

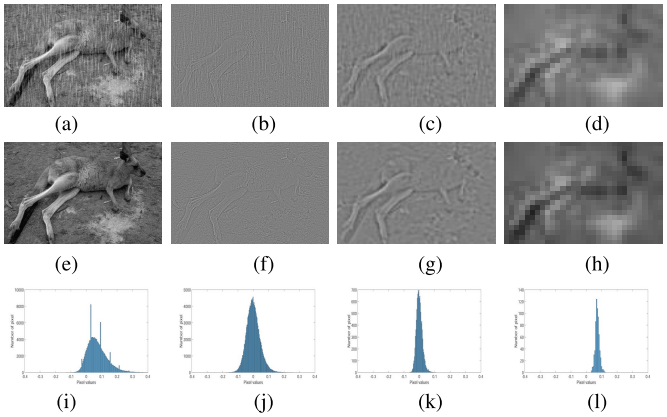


Fig. 3. Example of Laplacian pyramid. We show three levels here. The third and fifth levels are increased in size for better visualization. Bottom row: histogram of the residual to demonstrate the increased sparsity over the image domain. (a) Rainy image $\mathbf{X}$. (b) $L_1(\mathbf{X})$. (c) $L_3(\mathbf{X})$. (d) $L_5(\mathbf{X})$. (e) Clean image $\mathbf{Y}$. (f) $L_1(\mathbf{Y})$. (g) $L_3(\mathbf{Y})$. (h) $L_5(\mathbf{Y})$. (i) (a)–(e). (j) (b)–(f). (k) (c)–(g). (l) (d)–(h).

C. Stage 2: Subnetwork Structure

After decomposing $\mathbf{X}$ into different pyramid levels, we build a set of subnetworks independently for each level to predict a corresponding clean Gaussian pyramid $G(\mathbf{Y})$. All the subnetworks have the same network structure with different numbers of kernels. We adopt residual learning [20] for each network structure and recursive blocks [39] to reduce parameters. The subnetwork structure can be expressed as follows.

1) *Feature Extraction:* The first layer extracts features from the nth input level

$$\mathbf{H}_{n,0} = \sigma\left(\mathbf{W}_n^0 * L_n(\mathbf{X}) + \mathbf{b}_n^0\right) \tag{2}$$

where $\mathbf{H}$ indexes the feature map, $*$ is the convolution operation, $\mathbf{W}$ are weights and $\mathbf{b}$ are biases. σ is an activation function for nonlinearity.
2) *Recursive Block:* To reduce the number of parameters, we build intermediate inference layers in a

recursive fashion. The basic idea is to share parameters among recursive blocks. Motivated by our experiments, we adopt three convolutional operations in each recursive block. Calculations in the tth recursive block are

$$\mathbf{F}_{n,t}^1 = \sigma\left(\mathbf{W}_n^1 * \mathbf{H}_{n,t-1} + \mathbf{b}_n^1\right) \tag{3}$$

$$\mathbf{F}_{n,t}^2 = \sigma\left(\mathbf{W}_n^2 * \mathbf{F}_{n,t}^1 + \mathbf{b}_n^2\right) \tag{4}$$

$$\mathbf{F}_{n,t}^3 = \mathbf{W}_n^3 * \mathbf{F}_{n,t}^2 + \mathbf{b}_n^3 \tag{5}$$

where $\mathbf{F}^{\{1,2,3\}}$ are intermediate features in the recursive block, $\mathbf{W}^{\{1,2,3\}}$ and $\mathbf{b}^{\{1,2,3\}}$ are shared parameters among T recursive blocks and $t = 1, \ldots, T$. To help propagate information and back-propagate gradients, the output feature map $\mathbf{H}_{n,t}$ of the tth recursive block is calculated by adding $\mathbf{H}_{n,0}$

$$\mathbf{H}_{n,t} = \sigma\left(\mathbf{F}_{n,t}^3 + \mathbf{H}_{n,0}\right). \tag{6}$$

3) *Gaussian Pyramid Reconstruction:* To obtain the output level of the pyramid, the reconstruction layer is expressed as

$$L_n(\mathbf{Y}) = \left(\mathbf{W}_n^4 * \mathbf{H}_{n,T} + \mathbf{b}_n^4\right) + L_n(\mathbf{X}). \tag{7}$$

After obtaining the output of the Laplacian pyramid $L(\mathbf{Y})$, the corresponding Gaussian pyramid of the derained image can be reconstructed by

$$\begin{aligned} G_N(\mathbf{Y}) &= \max(0, L_N(\mathbf{Y})) \\ G_n(\mathbf{Y}) &= \max(0, L_n(\mathbf{Y}) + \text{upsample}(G_{n+1}(\mathbf{Y}))) \end{aligned} \tag{8}$$

where $n = 1, \ldots, N-1$. Since each level of a Gaussian pyramid should equal or lager than 0, we use $x = \max(0, x)$, which is actually the rectified linear units (ReLUs) operation [16], to simply correct the outputs. The final derained image is the bottom level of the Gaussian pyramid, i.e., $G_1(\mathbf{Y})$.

Denton *et al.* [40], Ghiasi and Fowlkes [41], Lai *et al.* [42], and Shen *et al.* [43] build similar networks based on the image pyramid, which are the most related to our own work. However, these papers apply similar structures to other tasks such as image generation, segmentation, or super-resolution using different network approaches on the pyramid. On the other hand, our LPNet aims to design a lightweight model and focuses on problem simplification from the perspective of signal analysis. The Gaussian–Laplacian pyramid is utilized to constrain and simplify the problem. Moreover, above-mentioned methods design the network structure in a pyramid fashion to obtain multi-scale feature maps. On the other hand, our LPNet directly decomposes the input image by using exact Laplacian pyramid algorithm to shrink all pyramid bands.

D. Loss Function

Given a training set $\{\mathbf{X}^i, \mathbf{Y}_{GT}^i\}_{i=1}^M$, where M is the number of training data and $\mathbf{Y}_{GT}$ is the ground truth, the most widely used loss function for training a network is mean squared error (MSE). However, MSE usually generates over-smoothed results due to the squared penalty that works poorly at edges in an image. Thus, for each subnetwork, we adopt different loss functions and minimize their combination. Following [44], we choose ℓ_1 and Structural SIMilarity index (SSIM) [45] as our loss functions. Specifically, as shown in Fig. 3, since finer details and rain streaks exist in lower pyramid levels we use SSIM loss to train the corresponding subnetworks for better preserving high-frequency information. On the contrary, larger structures and smooth background areas exist in higher pyramid levels. Thus, we use the ℓ_1 loss to update the corresponding network parameters there. The overall loss function is

$$\mathcal{L} = \frac{1}{M}\sum_{i=1}^{M}\left\{\sum_{n=1}^{N}\mathcal{L}^{\ell_1}\left(G_n(\mathbf{Y}^i), G_n\left(\mathbf{Y}_{GT}^i\right)\right) + \sum_{n=1}^{2}\mathcal{L}^{\text{SSIM}}\left(G_n(\mathbf{Y}^i), G_n\left(\mathbf{Y}_{GT}^i\right)\right)\right\} \tag{9}$$

where $\mathcal{L}^{\text{SSIM}}$ is the SSIM loss and $\mathcal{L}^{\ell_1}$ is the ℓ_1 loss. In this paper, we set the pyramid level $N = 5$ based on our experiments. We use SSIM loss for levels $\{1, 2\}$ and ℓ_1 loss for all levels.

According to our network design, the direct input of the subnetwork is a Laplacian pyramid. Each input of subnetwork, except the top pyramid level, is high-frequency parts. According to (7), the direct output of each subnetwork is also high-frequency parts. Thus, even though we give the same weights for ℓ_1 and SSIM loss, the sparse constraint of high-frequency information is implicitly embedded in each subnetwork, which makes the subnetwork tend to focus on high-frequency parts. Moreover, to make the proposed LPNet to be a unified network, we connect all outputs and reconstruct the Gaussian pyramid, which contains low-frequency parts at each level. At the back-propagation step, the gradients of lower pyramid level can flow to higher levels, which helps updating the parameters. Thus, by using ℓ_1 and SSIM losses, our LPNet can achieve the state-of-the-art deraining performances.

E. Removing Batch Normalization

As one of the most effective way to alleviate the internal co-variate shift, batch normalization (BN) [46] is widely adopted before the nonlinearity in each layer in existing deep learning based methods. However, we argue that by introducing image pyramid technology, BN can be removed to improve the flexibility of networks [47]. This is because BN constrains the feature maps to obey a Gaussian distribution. While during our experiments, we found that distributions of lower Laplacian pyramid levels of both clean and rainy images are sparse. To demonstrate this viewpoint, in Fig. 4, we show the histogram distributions of each Laplacian pyramid level from 200 clean and light rainy training image pairs from [32]. As can be seen, compared to the image domain shown in Fig. 4(a), distributions of lower pyramid levels, i.e., Fig. 4(c)–(f), are more sparse and do not obey Gaussian distribution. This implies that we do not need BN to further constrain the feature maps since the mapping problem already becomes easy to handle. Moreover, removing BN can

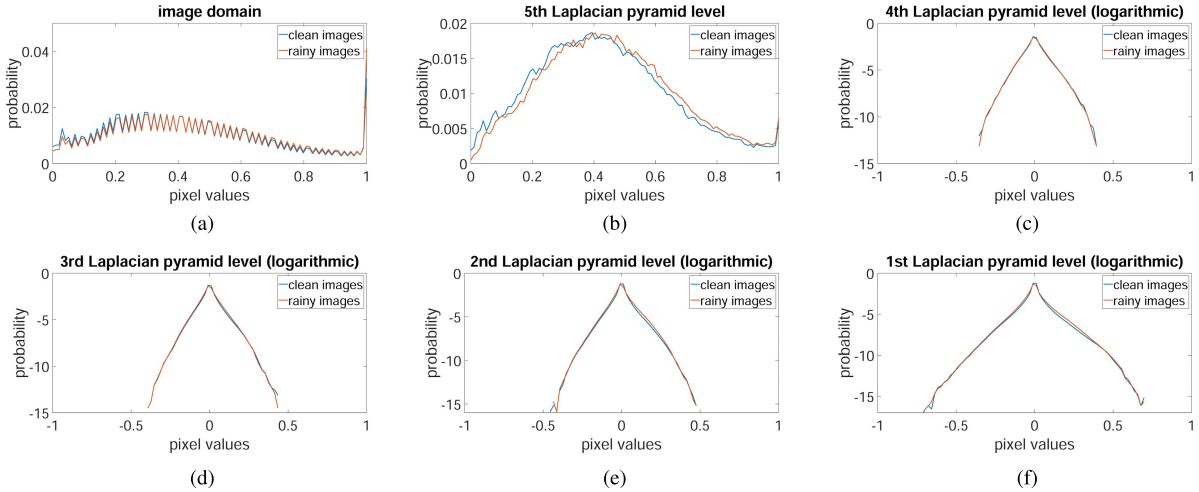


Fig. 4. Statistical histogram distributions of 200 clean and rainy pairs from [32]. To highlight the tail error, (c)–(f) are logarithmic transformed. (a) Image domain. (b) Fifth level. (c) Fourth level. (d) Third level. (e) Second level. (f) First level.

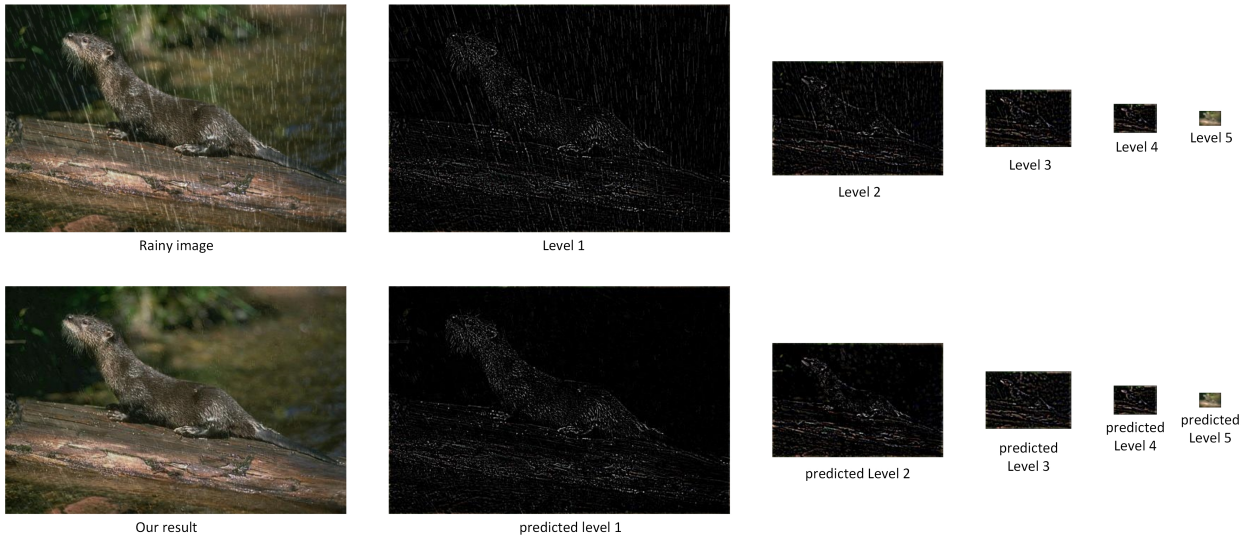


Fig. 5. One example of intermediate results predicted by our LPNet.

sufficiently reduce GPU memory usage since the BN layers consume the same amount of memory as the preceding convolutional layers. Based on the above observation and analysis, we remove BN layers from our network to improve flexibility and reduce parameter numbers and computing resource.

F. Parameter Settings

We decompose an RGB image into a five-level Laplacian pyramid by using a fixed smoothing kernel $[0.0625, 0.25, 0.375, 0.25, 0.0625]$, which is also used to reconstruct the Gaussian pyramid. In our network architecture, each subnetwork has the same structure with a different numbers of kernels. The kernel sizes for $\mathbf{W}^{\{0,1,3\}}$ are 3×3. For $\mathbf{W}^{\{2\}}$, the kernel size is 1×1 to further increase nonlinearity and reduce parameters. For the reconstruction layer $\mathbf{W}^{\{4\}}$, we also set the kernel size as 1×1 since this layer is used to merge feature maps into RGB images. The number of recursive blocks is $T = 5$ for each subnetwork. For the activation function σ, we use the leaky ReLUs (LReLUs) [48] with a negative slope of 0.2.

Moreover, as shown in the last row of Fig. 3, higher levels are closer to an identity mapping since rain streaks only remain in lower levels. This means that, for higher levels, fewer parameters are required for learning a good network. Thus, from low to high levels, we set the kernel numbers to $16, 8, 4, 2$, and 1, respectively. Since the top level is a tiny and smoothed version of image and rain streaks remain in high-frequency parts, the function of top level subnetwork is more like a simple global contrast adjustment. Thus, we set the kernel numbers to 1 kernel for the top level. As shown in Fig. 2, by connecting the upsampled version of the output from the higher level, the direct prediction of all subnetworks is actually the clean Laplacian pyramid. We show the intermediate results predicted by each subnetwork in Fig. 5. It is clear that rain streaks remain in lower levels while higher levels are almost the same. This demonstrates that our diminishing parameter setting is reasonable.

To each subnetwork, the equation of calculating parameter number is the same. Assuming the number of kernels at each layer is M and the channel of the input is C. Thus, the parameter number of first layer and last layer are $(C \times 3 \times 3 \times M + M) = 9 \times M \times C + M$ and $(M \times 1 \times 1 \times C + C) = M \times C + C$, respectively. Since the recursive blocks share the parameters, the number of parameters of all blocks is actually equal to the number of one block. Thus, the parameter number of each block is $(M \times 3 \times 3 \times M + M) + (M \times 1 \times 1 \times M + M) + (M \times 3 \times 3 \times M + M) = 19 \times M \times M + 3 \times M$. Thus, the parameter number of each subnetwork is $19 \times M \times M + 10 \times M \times C + 4 \times M + C$. According to our default configuration, $C = 3$ and $M = \{16, 8, 4, 2, 1\}$. Thus, the total number of trainable parameters is $(5411 + 1491 + 443 + 147 + 56) = 7548$, far fewer than the hundreds of thousands often encountered in deep learning.

G. Training Details

We use synthetic rainy images from [32] as our training data. This data set contains 1800 images with heavy rain and 200 images with light rain. We randomly generate three million 80×80 clean/rainy patch pairs for training. We use TensorFlow [49] to train LPNet using the Adam solver [50] with a minibatch size of 10. We set the learning rate as 0.001. The whole network is trained in a end-to-end fashion.

III. Experiments

We compare our LPNet with six state-of-the-art deraining methods: the GMM of [14], a CNN baseline super-resolution convolutional neural network (SRCNN) [25], the deep detail networks (DDNs) of [31], an advanced network called convolution in convolution (CIC) [21], a convolutional neural pyramid (CNP) network [43] for image processing, and joint rain detection and removal (JORDER) [32]. For fair comparison, all CNN based methods are retrained on the same training data sets.

A. Synthetic Data

Three synthetic data sets are chosen for comparison. Two of them are from [32] and each one contains 100 images. One is synthesized with heavy rain called *Rain100H* and the other one is with light rain called *Rain100L*. The third data set called *Rain12* is from [14] which contains 12 synthetic images. All testing results shown are not included in the training data. Following [32], for each CNN method, we train two models, one is for heavy rain and the other is for light rain. The model trained on the light rainy data set is used to test *Rain12*.

Figs. 6–9 show visual results from each data set. As can be seen, GMM [14] fails to remove rain streaks from heavy rainy images. SRCNN [25] and DDN [31] are able to remove the rain streaks while also tending to generate obvious artifacts. Our LPNet has comparable visual results with JORDER and outperforms other methods. The reason is twofold: first, due to the multi-scale pyramid decomposition, each subnetwork only needs to deal with specific components at that scale. This enables each subnetwork to extract more accurate features for more accurate image reconstruction. Second, both SRCNN and DDN use MSE as the loss function to guide the network

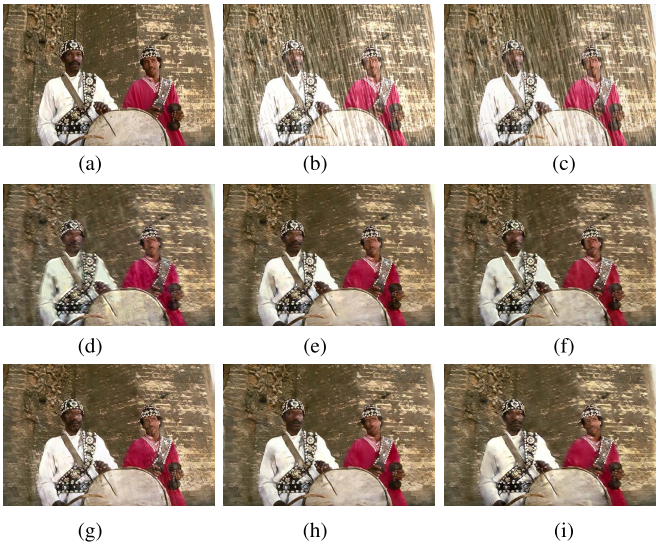


Fig. 6. One synthetic image from *Rain100H* [32]. (a) Ground Truth. (b) Rainy image. (c) GMM. (d) SRCNN. (e) DDN. (f) CIC. (g) CNP. (h) JORDER. (i) Our LPNet.

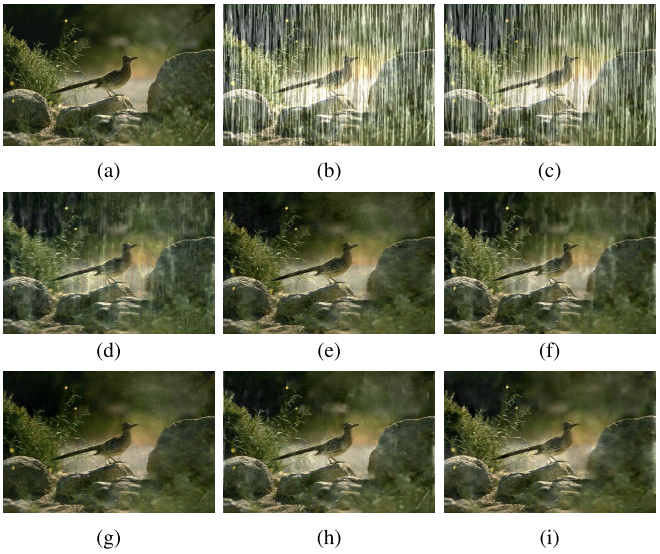


Fig. 7. One synthetic image from *Rain100H* [32]. (a) Ground Truth. (b) Rainy image. (c) GMM. (d) SRCNN. (e) DDN. (f) CIC. (g) CNP. (h) JORDER. (i) Our LPNet.

training, while our LPNet is trained by using a combined ℓ_1 and SSIM loss. This can focus more attention to small errors caused by rain streaks. Therefore, although all methods are based on the CNN structure, these additional aspects of LPNet makes it better able to perform the image restoration.

We also adopt peak signal-to-noise ratio (PSNR) and SSIM [45] to perform quantitative evaluations in Table I. Our method has comparable SSIM values with JORDER while outperforming other methods, in agreement with the visual results. Although our result has a lower PSNR value than JORDER method, the visual quality is comparable. This is because PSNR is calculated based on the MSE, which measures global pixel errors without considering local image characters. Moreover, as shown in Table I our LPNet contains far fewer parameters. Compared with the state-of-the-art methods CNP [43] and JORDER [32], our LPNet achieves comparable results while the parameter amount is reduced by 98.77% and

TABLE I
AVERAGE SSIM AND PSNR VALUES ON SYNTHESIZED IMAGES

	GMM [14]		SRCNN [25]		DDN [31]	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Rain100L	0.86 ± 0.06	28.7 ± 3.2	0.91 ± 0.04	29.4 ± 2.2	0.96 ± 0.02	34.6 ± 3.1
Rain100H	0.43 ± 0.14	15.0 ± 3.5	0.70 ± 0.08	22.8 ± 2.5	0.81 ± 0.07	**26.9** ± 2.7
Rain12	0.91 ± 0.04	32.0 ± 2.3	0.92 ± 0.03	31.9 ± 1.9	0.92 ± 0.04	34.4 ± 3.1
Parameters # / reduction	-		20,099 / -62.45%		57,369 / -86.84%	

	CIC [21]		CNP [43]		JORDER [32]		Our LPNet	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Rain100L	0.96 ± 0.02	33.6 ± 3.3	0.95 ± 0.02	33.1 ± 2.6	**0.97** ± 0.01	**36.6** ± 3.1	0.96 ± 0.02	33.4 ± 2.9
Rain100H	0.80 ± 0.07	23.8 ± 2.5	**0.83** ± 0.07	23.9 ± 3.1	**0.83** ± 0.07	26.5 ± 2.9	0.82 ± 0.06	23.4 ± 2.0
Rain12	0.95 ± 0.02	35.3 ± 3.3	**0.95** ± 0.02	35.3 ± 2.8	**0.95** ± 0.03	**35.9** ± 3.7	**0.95** ± 0.03	34.7 ± 3.1
Parameters # / reduction	15,287 / -50.62%		613,819 / -98.77%		369,792 / -97.96%		**7,548**	

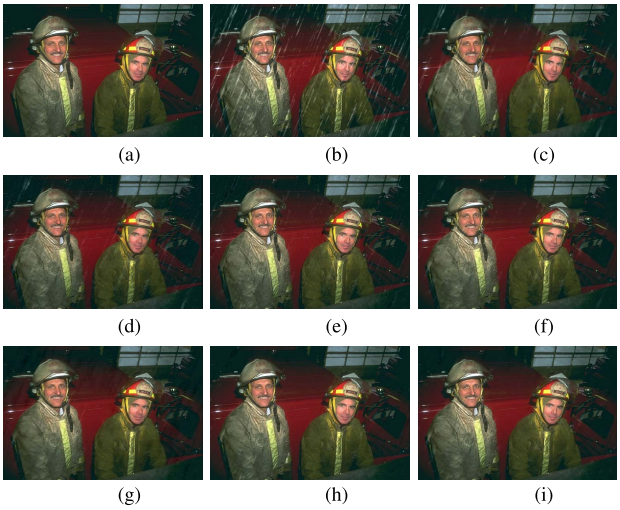


Fig. 8. One synthetic image from *Rain100L* [32]. (a) Ground Truth. (b) Rainy images. (c) GMM. (d) SRCNN. (e) DDN. (f) CIC. (g) CNP. (h) JORDER. (i) Our LPNet.

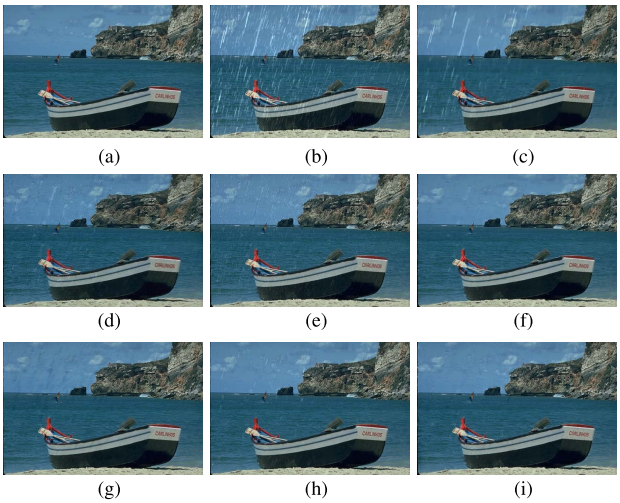


Fig. 9. One synthetic image from *Rain12* [14]. (a) Ground Truth. (b) Rainy images. (c) GMM. (d) SRCNN. (e) DDN. (f) CIC. (g) CNP. (h) JORDER. (i) Our LPNet.

97.96%, respectively. This makes our LPNet more suitable for storage, e.g., in mobile devices. In Fig. 10, we also show error bars of SSIM and PSNR results to provide a more intuitive comparison.

B. Real-World Data

Since one real-world rainy image may contain different scales of rain streaks, the degradations of real-world rain scenarios are very complex. In this section, we show that the LPNet learned on synthetic training data still performs well on real-world data. Figs. 11–13 show three visual results on real-world images. As can be seen, LPNet generates consistently promising derained results on images with different kinds of rain streaks. This is because our LPNet utilizes Laplacian pyramid algorithm to decompose rainy images. After decomposition, all pyramid bands are shrink which helps to deal with real-world scenarios.

Since no ground truth exists, we construct an independent user study to provide realistic feedback and quantify the subjective evaluation. We collect 300 real-world rainy images from the Internet as a new data set.[1] We use the compared five methods to generate derained results and randomly order the outputs, as well as the original rainy image, and display them on a screen. We then separately asked 20 participants to rank each image from 1 to 5 subjectively according to quality, with the instructions being that visible rain streaks should decrease the quality and clarity should increase quality (1 represents the worst quality and 5 represents the best quality). In Fig. 14, we show the scatter plot of the rainy inputs versus derained user scores. This small-scale experiment gives additional support that our LPnet improves the deraining on real-world images.

Moreover, when dealing with dense rain, LPNet trained on images with heavy rain has a dehazing effect as shown in Fig. 15, which can further improve the visual quality. This is because the highest level subnetwork (low-pass component) can adjust image contrast. Although dehazing is not the main focus of this paper, we believe that LPNet can be easily modified for joint deraining and dehazing.

C. Running Time and Convergence

To demonstrate the efficiency of LPNet, we show the average running time for a test image in Table II. Three different image sizes are chosen and each one is tested over 100 images. The GMM is implemented on CPUs according to the provided code, while other deep CNN-based methods are tested on both

[1] Code and data set: https://xueyangfu.github.io/projects/LPNet.html.

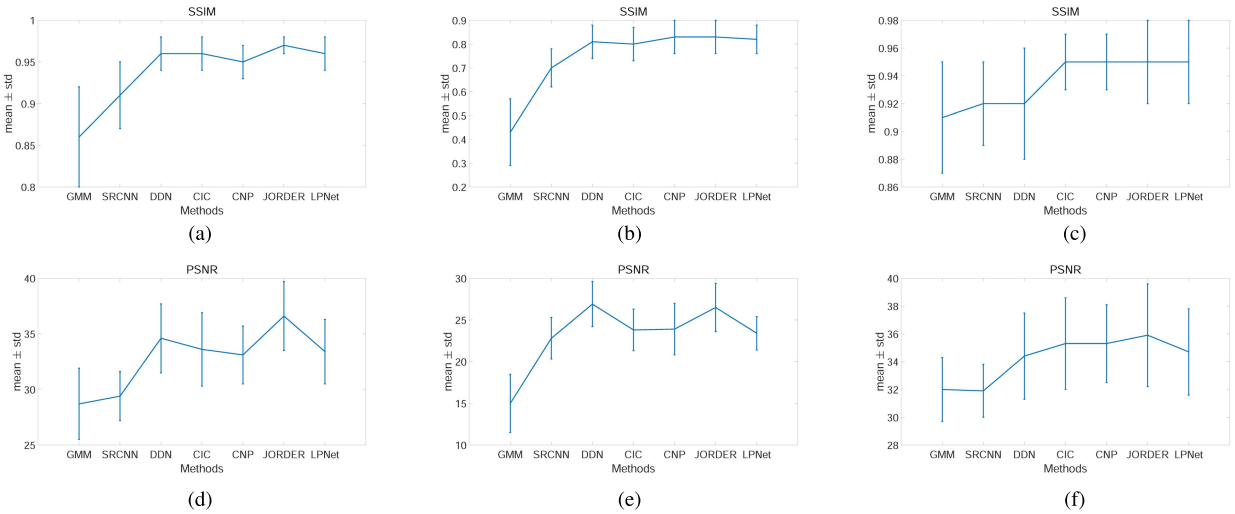


Fig. 10. Error bars of quantitative metrics. Top row: SSIM results. Bottom row: PSNR results. (a) *Rain100L* data set. (b) *Rain100H* data set. (c) *Rain12* data set. (d) *Rain100L* data set. (e) *Rain100H* data set. (f) *Rain12* data set.

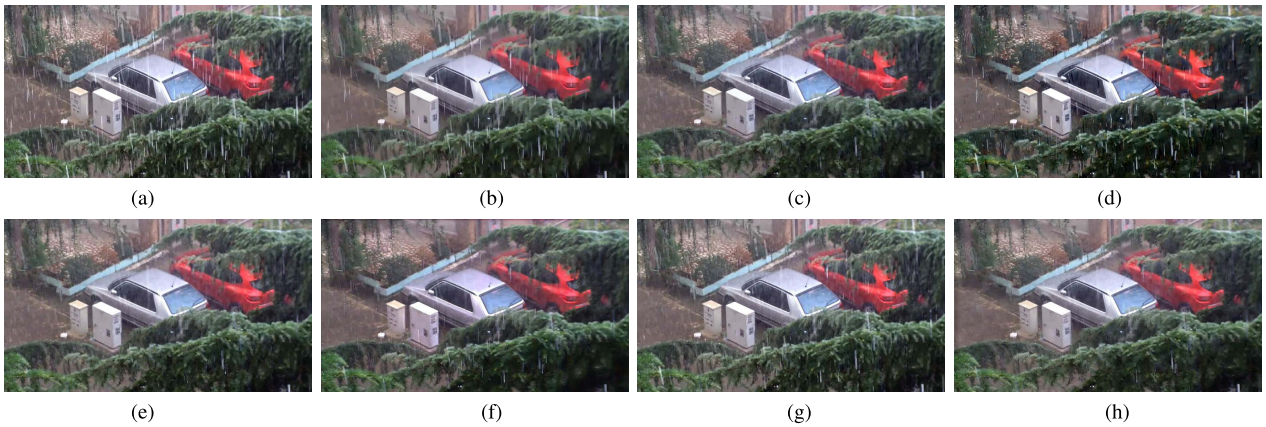


Fig. 11. One result on real-world rainy images. (a) Rainy images. (b) GMM. (c) SRCNN. (d) DDN. (e) CIC. (f) CNP. (g) JORDER. (h) Our LPNet.

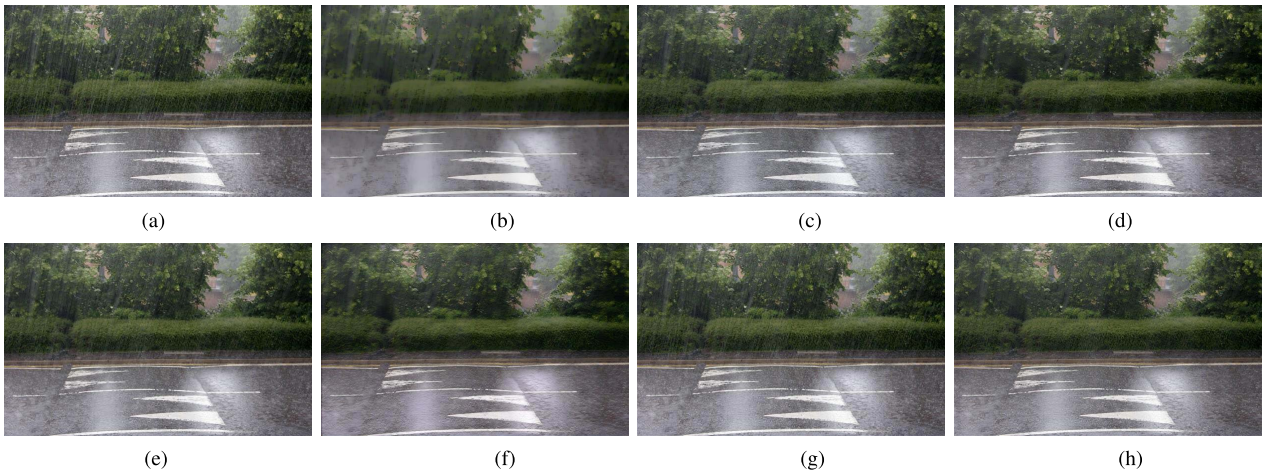


Fig. 12. One result on real-world rainy images. (a) Rainy images. (b) GMM. (c) SRCNN. (d) DDN. (e) CIC. (f) CNP. (g) JORDER. (h) Our LPNet.

CPU and GPU. All experiments are performed on a server with Intel(R) Xeon(R) CPU E5-2683, 64 GB RAM and NVIDIA GTX 1080. The GMM has the slowest running time since complicated inference is required to process each new image. Our method has a comparable and even faster computational time on both CPU and GPU compared with other deep models.

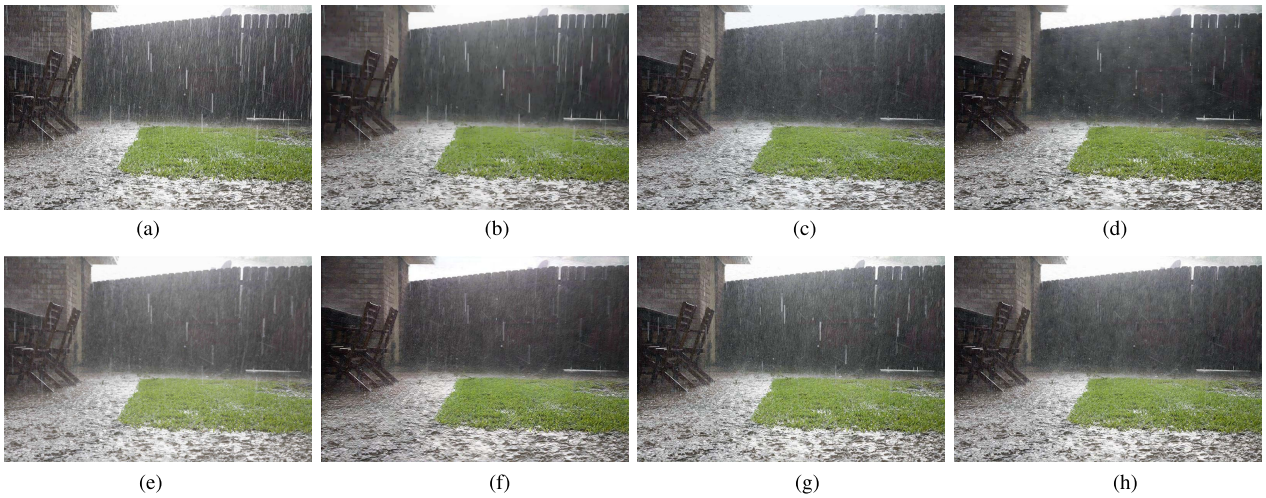


Fig. 13. One result on real-world rainy images. (a) Rainy images. (b) GMM. (c) SRCNN. (d) DDN. (e) CIC. (f) CNP. (g) JORDER. (h) Our LPNet.

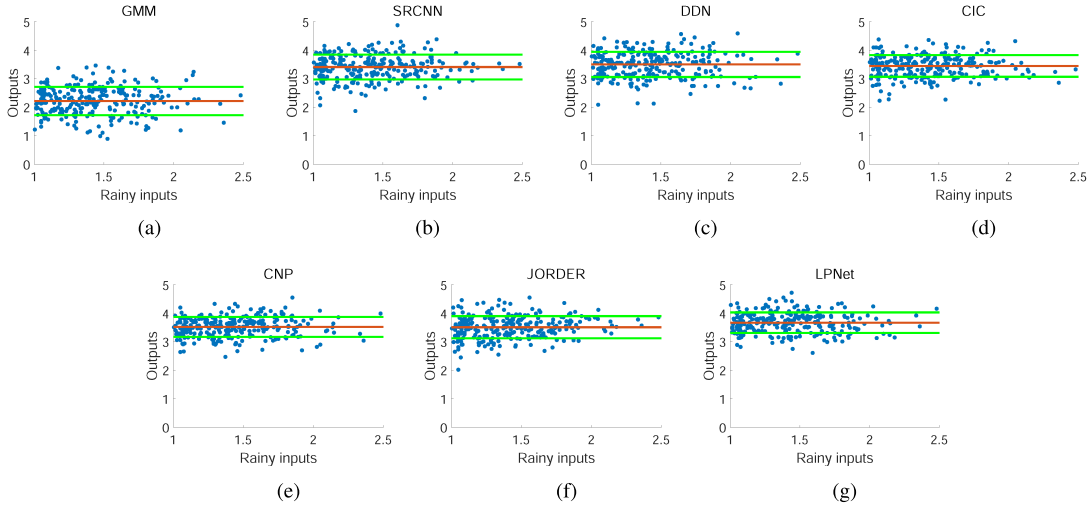


Fig. 14. Scatter plots of the rainy inputs versus derained user scores. Red line: mean value. Green line: standard deviation. (a) mean = 2.22, std = 0.50. (b) mean = 3.41, std = 0.44. (c) mean = 3.50, std = 0.44. (d) mean = 3.45, std = 0.38. (e) mean = 3.52, std = 0.35. (f) mean = 3.51, std = 0.39. (g) mean = 3.67, std = 0.36.

TABLE II
COMPARISON OF RUNNING TIME (IN SECONDS)

	GMM [14]		SRCNN [25]		DDN [31]		CIC [21]		CNP [43]		JORDER [32]		Our LPNet	
Image size	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
500 × 500	1.99×10^3	-	0.25	0.03	1.51	0.16	1.01	0.12	1.19	0.21	2.95×10^2	0.18	0.67	0.12
750 × 750	3.09×10^3	-	0.58	0.09	3.33	0.22	2.58	0.18	2.09	0.15	5.98×10^2	0.36	1.49	0.16
1024 × 1024	6.52×10^3	-	1.07	0.11	5.40	0.32	3.47	0.29	4.71	0.25	1.20×10^3	0.82	2.46	0.20

This is because LPNet uses relatively shallow networks for each level, so requires fewer convolutions.

We also show the average training loss as a function of training epoch in Fig. 16. We observe that LPNet converges quickly on training with both light and heavy rainy data sets. Since heavy rain streaks are harder to handle, as shown in Fig. 6, the training error of heavy rain streaks has a vibration.

D. Ablation Study

In this section, we discuss different configurations to study their impact on performance.

1) Increasing Kernel Number: We have conducted an experiment on the *Rain100H* data set with increased kernel parameters, i.e., 16 feature maps for all convolution layers at each subnetwork. The results are shown in Table III. As can be seen, the SSIM evaluation is better than JORDER and PSNR value is also improved. We believe that the performance can be further improved by using more kernels. However, increasing the kernel number requires more storage and computing resources. Fig. 17 shows one example by using different kernel numbers. As can be seen, the visual quality is almost the same. Thus, we use our diminishing kernel parameter setting to achieve the balance between effectiveness and efficiency.

(a) (b)

Fig. 15. Example of dehazing effect. Our LPNet trained on the heavy rainy data set can further improve image contrast. (a) Light rainy model. (b) Heavy rainy model.

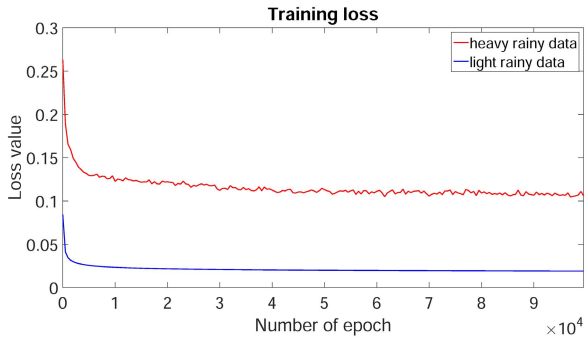


Fig. 16. Convergence on different training data sets.

TABLE III
SSIM AND PSNR COMPARISON FOR DIFFERENT KERNEL PARAMETERS

	Our LPNet (default)		Our LPNet (increasing)	
	SSIM	PSNR	SSIM	PSNR
Rain100H	0.82	23.43	0.84	24.09
Parameters #	7,548		27,055	

(a) (b) (c)

Fig. 17. One example by using different kernel numbers. (a) Rainy image. (b) Default numbers. (c) Sixteen feature maps.

2) Pyramid Levels: Intuitively, if we decompose the image into pyramid with more levels, the performance should improve. We train three models by using different pyramid levels: 3, 5, and 6. The average SSIM and PSNR results on *Rain100H* data set are shown in Table IV. As can be seen, better performance can be achieved by increasing the levels from 3 to 5. However, keeping increasing the levels brings only limited improvement. This is because after generating five pyramid levels, the input image can already be finely decomposed, and the learning problem has been well simplified. Continuously increasing the number of pyramid levels can only provide limited constraints on the learning problem. Thus, we decompose the input images into five pyramid levels as the default setting.

3) Recursive Block Number: We also test the performance of using different numbers of recursive block at each pyramid level. We train and test on three networks with recursive block numbers 3, 5, and 7. As shown in Table V, a significant improvement can be achieved by increasing recursive block numbers from 3 to 5. However, from our experiments we find that keep increasing the recursive block numbers brings limited improvement. This is because the learning process is simplified by incorporating our domain-specific knowledge. At each level, the nonlinear mapping function constructed by five recursive blocks can well solve the learning problem. Thus, we choose five recursive blocks as the default setting.

TABLE IV
SSIM AND PSNR COMPARISON FOR DIFFERENT PYRAMID LEVELS

Pyramid #	3		5 (default)		6	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Rain100H	0.815	23.25	0.821	23.43	0.824	23.57

TABLE V
SSIM AND PSNR COMPARISON FOR DIFFERENT NUMBERS OF RECURSIVE BLOCK

Block #	3		5 (default)		7	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Rain100H	0.812	21.05	0.821	23.43	0.827	23.62

Fig. 18. Training curves with and without skip connections.

TABLE VI
SSIM AND PSNR COMPARISON FOR DIFFERENT LOSS FUNCTIONS

Loss	MSE		SSIM + ℓ_1 (default)		MSE + SSIM + ℓ_1	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Rain100H	0.78	24.63	0.82	23.43	0.81	23.89

4) Skip Connections: Although Laplacian pyramid images introduce sparsity in each level to simply the mapping problem, it is still essential to add skip connection in each subnetwork. We adopt skip connection for two reasons. First, image information may be lost during feed-forward convolutional operations, using skip connection helps to propagate information flow and improve the deraining performance. Second, using skip connection helps to back-propagate gradient, which can accelerate the training procedure, when updating parameters. In Fig. 18, we show the training curves on the heavy rainy data set with and without all skip connections. As can be seen, using skip connection can bring a faster convergence rate and lower training loss.

TABLE VII
SSIM AND PSNR COMPARISON FOR NONRAINY IMAGES

	GMM [14]	SRCNN [25]	DDN [31]	CIC [21]	CNP [43]	JORDER [32]	Our LPNet
PSNR	31.03	33.17	31.88	39.33	37.58	41.05	40.83
SSIM	0.915	0.975	0.980	0.990	0.990	0.996	0.994

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

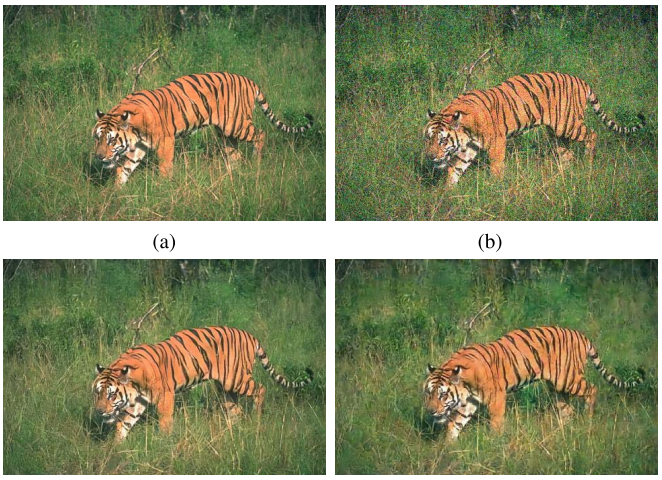
Fig. 19. Deraining example by using different loss functions. Using SSIM + ℓ_1 loss generates a more sharpen result as shown in (c). While shown in (d), a similar result with (c) is generated by using all the three losses. (a) Rainy image. (b) MSE loss. (c) SSIM + ℓ_1 loss. (d) MSE + SSIM + ℓ_1 loss.

5) Loss Function: We use SSIM as a part of loss function (9) for two main reasons. First, SSIM is calculated based on local image characteristics, e.g., local contrast, luminance, and details, which are also the characteristics of rain streaks. Thus, using SSIM as the loss function is appropriate to guide the network training. Second, the human visual system is also sensitive to local image characteristics. SSIM has been motivated as generating more visually pleasing results, unlike PSNR. It has, therefore, become a more prominent measure in the image processing community. We also use ℓ_1 loss because ℓ_1 does not overpenalize larger errors and, thus, can preserve structures and edges. On the contrary, the widely used MSE loss (which PSNR is based on) often generates oversmoothed results because it penalizes larger errors and tolerates small errors. Therefore, MSE struggles to preserve underlying structures in the image compared with ℓ_1. Fig. 19 shows three results generated by using our combined loss (9) and MSE loss, respectively. As can be seen, using our combined loss (9) can preserve more details. Moreover, using all the three losses generates a similar result with (9). The SSIM and PSNR comparison is also shown in Table VI. Thus, we choose (9) as the default loss function.

6) Non-Rainy Images: Intuitively, a deraining algorithm should be able to distinguish rain streaks from object details. In other words, if the input is a nonrainy image, the output should not be seriously distorted. To demonstrate that our LPNet can isolate rain streaks, we test 500 clean images from BSD500 data set [51]. Fig. 20 shows one visual comparison, where we see that the output of LPNet is almost the same as the clean image. As can be seen in Table VII, LPNet has comparable results with JORDER in this case, and outperforms other methods. This experiment indicates that our multi-scale decomposition strategy is helpful for capturing and distinguishing rain streaks.

(a) (b)

Fig. 20. One example of directly testing on nonrainy image. (a) Nonrainy input/PSNR, SSIM. (b) Our LPNet / 45.87, 0.998.



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

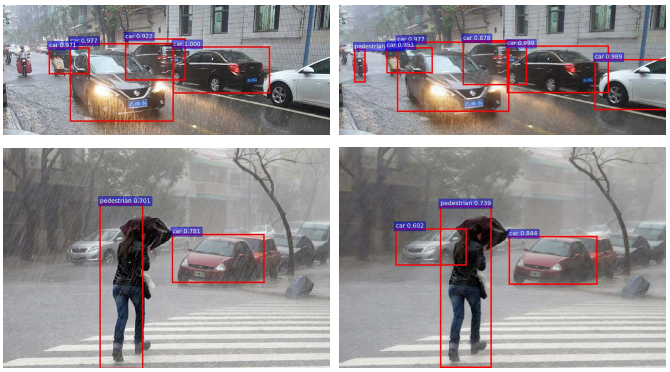
Fig. 21. Visual comparison on Gaussian denoising, our LPNet generates a comparable visual result with fewer parameters. (a) Ground Truth. (b) Noisy image, $\sigma = 30$. (c) DnCNN [52]. (d) Our result.

E. Other Applications

1) Other Image Processing Tasks: Since both Laplacian pyramids and CNNs are fundamental and general image processing technologies, our network design has potential value for other low-level vision tasks. Fig. 21 shows the experimental result on Gaussian denoising compared with one popular deep CNN based method, which is designed for general image restoration, named DnCNN [52]. We use the BSD500 data set provided by [51] and test on noise levels 30 and 50. Fig. 21 shows one visual result at noise level 30. As can be seen, our LPNet can well handle the image denoising task since the desired image is also corrupted by high frequency content. We also show the quantitative results in Table VIII. Compared to DnCNN, our LPNet achieves

TABLE VIII
SSIM AND PSNR COMPARISON ON GAUSSIAN DENOISING

	DnCNN [52]		Our LPNet	
	SSIM	PSNR	SSIM	PSNR
$\sigma = 30$	0.82	30.44	0.81	29.63
$\sigma = 50$	0.76	26.37	0.74	24.59
Parameter #	559,233		7,548	



(a) (b)

Fig. 22. Example of joint deraining and object detection on a real-world image. We use Faster R-CNN [53] to detect objects. (a) Direct detection. (b) Deraining + detection.

comparable SSIM and PSNR values with fewer parameters. This test demonstrates that LPNet can generalize to similar image restoration problems.

2) Preprocessing for High-Level Vision Tasks: Due to the lightweight architecture, our LPNet can potentially be efficiently incorporated into other high-level vision systems. For example, we study the problem of object detection in rainy environments. Since rain steaks can blur and block objects, the performance of object detection will degrade in rainy weather. Fig. 22 shows a visual result of object detection by combining with the popular Faster R-CNN model [53]. It is obviously that rain streaks can degrade the performance of Faster R-CNN, i.e., by missing detections and producing low recognition confidence. On the other hand, after deraining by LPNet, the detection performance has a notable improvement over the naive Faster-RCNN.

In addition, due to the lightweight architecture, using LPNet with Faster R-CNN does not significantly increase the complexity. To process a color image with size of 1024×1024, the running time is 3.7 s for Faster R-CNN, and 4.0 s for LPNet + Faster R-CNN.

IV. CONCLUSION

In this paper, we have introduced a lightweight deep network that is based on the classical Gaussian–Laplacian pyramid for single image deraining. Our LPNet contains several subnetworks and inputs the Laplacian pyramid to predict the clean Gaussian pyramid. By using the pyramid to simplify the learning problem and adopting recursive blocks to share parameters, LPNet has fewer than 8K parameters while still achieving good performance. Moreover, due to the generality and lightweight architecture, our LPNet has potential values for other low- and high-level vision tasks.

By utilizing Gaussian–Laplacian pyramid, the spatial constraint on each pyramid level is particularly effective to separate high-frequency parts into different scales. We have verified that this decomposition can capture a narrower distribution to describe both rain streaks and objects' details at different scales. Thus, our LPNet can generalizes well to real-world images since the spatial scale of each pyramid level is well constrained. Without the constraint of pyramid decomposition, the image deraining problem becomes tougher since the change of image contents and rain streaks are almost infinite. Our method is not without limitations. Issues such as the optimal size of kernel, how to take rain orientations into consideration, how to well distinguish rain streaks from object edges, and how to handle heavy rain artifacts, are still open problems. Moreover, other advanced deep learning methods, such as generative adversarial networks (GANs), can also be utilized to improve the deraining performance.

Moreover, it should be noticed that fewer parameters do not equal to fewer floating-point operations or higher inference efficiency. The goal of this work is to design a lightweight model for the tough image deraining problem. For practical applications, the computational time can be further improved by combining our network with more efficient network architectures, such as MobileNet [54] and ShuffleNet [55]. We will incorporate above issues into our future work.

REFERENCES

[1] K. Garg and S. K. Nayar, "Detection and removal of rain from videos," in *Proc. CVPR*, Jun./Sep. 2004, p. 1.
[2] P. C. Barnum, S. Narasimhan, and T. Kanade, "Analysis of rain and snow in frequency space," *Int. J. Comput. Vis.*, vol. 86, nos. 2–3, pp. 256–274, Jan. 2010.
[3] J. Bossu, N. Hautière, and J.-P. Tarel, "Rain or snow detection in image sequences through use of a histogram of orientation of streaks," *Int. J. Comput. Vis.*, vol. 93, no. 3, pp. 348–367, 2011.
[4] Y.-L. Chen and C.-T. Hsu, "A generalized low-rank appearance model for spatio-temporally correlated rain streaks," in *Proc. ICCV*, Dec. 2013, pp. 1968–1975.
[5] J.-H. Kim, J.-Y. Sim, and C.-S. Kim, "Video deraining and desnowing using temporal correlation and low-rank matrix completion," *IEEE Trans. Image Process.*, vol. 24, no. 9, pp. 2658–2670, Sep. 2015.
[6] W. Ren, J. Tian, Z. Han, A. Chan, and Y. Tang, "Video desnowing and deraining based on matrix decomposition," in *Proc. ICCV*, Jul. 2017, pp. 4210–4219.
[7] W. Wei, L. Yi, Q. Xie, Q. Zhao, D. Meng, and Z. Xu, "Should we encode rain streaks in video as deterministic or stochastic?" in *Proc. ICCV*, Oct. 2017, pp. 2516–2525.
[8] J.-H. Kim, C. Lee, J.-Y. Sim, and C.-S. Kim, "Single-image deraining using an adaptive nonlocal means filter," in *Proc. IEEE ICIP*, Sep. 2013, pp. 914–917.
[9] Y. Chang, L. Yan, and S. Zhong, "Transformed low-rank model for line pattern noise removal," in *Proc. ICCV*, Oct. 2017, pp. 1726–1734.
[10] L.-W. Kang, C.-W. Lin, and Y.-H. Fu, "Automatic single-image-based rain streaks removal via image decomposition," *IEEE Trans. Image Process.*, vol. 21, no. 4, pp. 1742–1755, Apr. 2012.
[11] D.-A. Huang, L.-W. Kang, Y.-C. F. Wang, and C.-W. Lin, "Self-learning based image decomposition with applications to single image denoising," *IEEE Trans. Multimedia*, vol. 16, no. 1, pp. 83–93, Jan. 2014.
[12] Y. Luo, Y. Xu, and H. Ji, "Removing rain from a single image via discriminative sparse coding," in *Proc. ICCV*, Dec. 2015, pp. 3397–3405.
[13] Y. Wang, S. Liu, C. Chen, and B. Zeng, "A hierarchical approach for rain or snow removing in a single color image," *IEEE Trans. Image Process.*, vol. 26, no. 8, pp. 3936–3950, Aug. 2017.

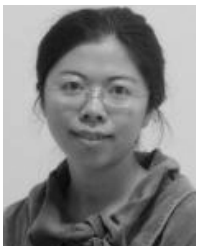
[14] Y. Li, R. T. Tan, X. Guo, J. Lu, and M. S. Brown, "Rain streak removal using layer priors," in *Proc. CVPR*, Jun. 2016, pp. 2736–2744.
[15] L. Zhu, C.-W. Fu, D. Lischinski, and P.-A. Heng, "Joint bi-layer optimization for single-image rain streak removal," in *Proc. ICCV*, Oct. 2017, pp. 2526–2534.
[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1097–1105.
[17] H. Cecotti, M. P. Eckstein, and B. Giesbrecht, "Single-trial classification of event-related potentials in rapid serial visual presentation tasks using supervised spatial filtering," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 11, pp. 2030–2042, Nov. 2014.
[18] T. Chen, L. Lin, L. Liu, X. Luo, and X. Li, "DISC: Deep image saliency computing via progressive representation learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 6, pp. 1135–1149, Jun. 2016.
[19] M. Gong, J. Zhao, J. Liu, Q. Miao, and L. Jiao, "Change detection in synthetic aperture radar images based on deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 1, pp. 125–138, Jan. 2015.
[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, Jun. 2016, pp. 770–778.
[21] Y. Pang, M. Sun, X. Jiang, and X. Li, "Convolution in convolution for network in network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 1587–1597, May 2018.
[22] T. Li, B. Ni, M. Xu, M. Wang, Q. Gao, and S. Yan, "Data-driven affective filtering for images and videos," *IEEE Trans. Cybern.*, vol. 45, no. 10, pp. 2336–2349, Oct. 2015.
[23] X. Cao, F. Zhou, L. Xu, D. Meng, Z. Xu, and J. Paisley, "Hyperspectral image classification with Markov random fields and a convolutional neural network," *IEEE Trans. Image Process.*, vol. 27, no. 5, pp. 2354–2367, May 2018.
[24] W. Hou, X. Gao, D. Tao, and X. Li, "Blind image quality assessment via deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 6, pp. 1275–1286, Jun. 2015.
[25] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 295–307, Feb. 2015.
[26] Y. Tai, J. Yang, X. Liu, and C. Xu, "MemNet: A persistent memory network for image restoration," in *Proc. ICCV*, Oct. 2017, pp. 4539–4547.
[27] X. Hu, G. Feng, S. Duan, and L. Liu, "A memristive multilayer cellular neural network with applications to image processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 8, pp. 1889–1901, Aug. 2017.
[28] R. Dian, S. Li, A. Guo, and L. Fang, "Deep hyperspectral image sharpening," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5345–5355, Nov. 2018.
[29] D. Eigen, D. Krishnan, and R. Fergus, "Restoring an image taken through a window covered with dirt or rain," in *Proc. ICCV*, Dec. 2013, pp. 633–640.
[30] X. Fu, J. Huang, X. Ding, Y. Liao, and J. Paisley, "Clearing the skies: A deep network architecture for single-image rain removal," *IEEE Trans. Image Process.*, vol. 26, no. 6, pp. 2944–2956, Jun. 2017.
[31] X. Fu, J. Huang, D. Zeng, Y. Huang, X. Ding, and J. Paisley, "Removing rain from single images via a deep detail network," in *Proc. CVPR*, Jul. 2017, pp. 3855–3863.
[32] W. Yang, R. T. Tan, J. Feng, J. Liu, Z. Guo, and S. Yan, "Deep joint rain detection and removal from a single image," in *Proc. CVPR*, Jul. 2017, pp. 1357–1366.
[33] W. Yang, R. T. Tan, J. Feng, J. Liu, S. Yan, and Z. Guo, "Joint rain detection and removal from a single image with contextualized deep networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, to be published. doi: 10.1109/TPAMI.2019.2895793.
[34] H. Zhang, V. Sindagi, and V. M. Patel, "Image de-raining using a conditional generative adversarial network," *IEEE Trans. Circuits Syst. Video Technol.*, to be published. doi: 10.1109/TCSVT.2019.2920407.
[35] H. Zhang and V. M. Patel, "Density-aware single image de-raining using a multi-stream dense network," in *Proc. CVPR*, Jun. 2018, pp. 695–704.
[36] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. ICLR*, 2016, pp. 1–14.
[37] P. J. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Trans. Commun.*, vol. COM-31, no. 4, pp. 532–540, Apr. 1983.
[38] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 6, pp. 1397–1409, Jun. 2013.
[39] Y. Tai, J. Yang, and X. Liu, "Image super-resolution via deep recursive residual network," in *Proc. CVPR*, Jul. 2017, pp. 3147–3155.
[40] E. L. Denton, S. Chintala, and R. Fergus, "Deep generative image models using a Laplacian pyramid of adversarial networks," in *Proc. NIPS*, 2015, pp. 1486–1494.
[41] G. Ghiasi and C. C. Fowlkes, "Laplacian pyramid reconstruction and refinement for semantic segmentation," in *Proc. ECCV*, 2016, pp. 519–534.
[42] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep Laplacian pyramid networks for fast and accurate super-resolution," in *Proc. CVPR*, Jul. 2017, pp. 624–632.
[43] X. Shen, Y.-C. Chen, X. Tao, and J. Jia, "Convolutional neural pyramid for image processing," 2017, *arXiv:1704.02071*. [Online]. Available: https://arxiv.org/abs/1704.02071
[44] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Trans. Comput. Imag.*, vol. 3, no. 1, pp. 47–57, Mar. 2017.
[45] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.
[46] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, 2015, pp. 1–11.
[47] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. CVPR Workshops*, Jul. 2017, pp. 136–144.
[48] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, 2013, pp. 1–6.
[49] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 1–21.
[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2014, pp. 1–15.
[51] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 3, no. 5, pp. 898–916, May 2011.
[52] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian Denoiser: Residual learning of deep CNN for image denoising," *IEEE Trans. Image Process.*, vol. 26, no. 7, pp. 3142–3155, Jul. 2017.
[53] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. NIPS*, 2015, pp. 91–99.
[54] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861
[55] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. CVPR*, Jun. 2018, pp. 6848–6856.

Xueyang Fu received the Ph.D. degree in signal and information processing from Xiamen University, Xiamen, China, in 2018.

From 2016 to 2017, he was a Visiting Student with Columbia University, New York, NY, USA, sponsored by the China Scholarship Council. He is currently an Associate Researcher with the Department of Automation, University of Science and Technology of China, Hefei, China. His current research interests include machine learning and image processing.

Borong Liang received the B.S. degree from Wuhan University, Wuhan, China, in 2016. He is currently pursuing the master's degree with the Department of Communication Engineering, School of Information Science and Engineering, Xiamen University, Xiamen, China.

His current research interests include machine learning and image processing.



Yue Huang received the B.S. degree from Xiamen University, Xiamen, China, in 2005, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2010.

From 2015 to 2016, she was a Visiting Scholar with Carnegie Mellon University, Pittsburgh, PA, USA. She is currently an Associate Professor with the Department of Communication Engineering, School of Information Science and Engineering, Xiamen University. Her current research interests include machine learning and image processing.

Xinghao Ding was born in Hefei, China, in 1977. He received the B.S. and Ph.D. degrees from the Department of Precision Instruments, Hefei University of Technology, Hefei, in 1998 and 2003, respectively.

From 2009 to 2011, he was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA. Since 2011, he has been a Professor with the School of Information Science and Engineering, Xiamen University, Xiamen, China. His current research interests include machine learning, representation learning, medical image analysis, and computer vision.

John Paisley received the B.S., M.S., and Ph.D. degrees in electrical engineering from Duke University, Durham, NC, USA.

He was a Post-Doctoral Researcher with the Computer Science Department, University of California at Berkeley, Berkeley, CA, USA, and with Computer Science Department, Princeton University, Princeton, NJ, USA. He is currently an Associate Professor with the Department of Electrical Engineering, Columbia University, New York, NY, USA, where he also a member of the Data Science Institute. His current research is machine learning, focusing on models and inference techniques for text and image processing applications.