Supplementary Material for FBI-Denoiser: Fast Blind Image Denoiser for Poisson-Gaussian Noise

Jaeseok Byun[1]*, Sungmin Cha[1]*, and Taesup Moon[2]†
[1]Department of ECE, Sungkyunkwan University, Suwon, Korea
[2]Department of ECE, Seoul National University, Seoul, Korea
{wotjr3868, csm9593}@skku.edu, tsmoon@snu.ac.kr

1. Implementation details

1.1. Software Platform / hyperparameters for training

All experiments were conducted by eight-core Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz, 128GB RAM and an NVIDIA Titan XP. Learning rages of 0.001 and 0.0001 were used for training FBI-Net and PGE-Net, respectively. We used Adam [8] optimizer and drop the learning rate by half every ten epochs. Additionally, we used Pytorch [10] with CUDA 10.2 to implement FBI-Net and PGE-Net, with mini-batch size as 1. For the Gaussian noise level estimation [5], the patch size is set to eight.

1.2. Gaussian noise estimation [5] (Tensorized)

[5] is a PCA-based Gaussian noise level estimation algorithm, developed with an intuition that image patches generated from a clean image often lie in a low-dimensional subspace. Thus, their core idea is to find redundant dimensions of a noisy image for capturing noise component by carefully eliminating the principal dimensions of a image. Namely, they start removing eigenvalues of a covariance matrix of noisy patches from the largest value, and stop at the right time.

As described in [5, Algorithm 1], it is mainly composed of three steps. First, decompose a single noisy image $\boldsymbol{Y} \in \mathbb{R}^{N\times N}$ to generate vectorized patches with size d^2. In second step, calculate the eigenvalues $\boldsymbol{S} = \{\lambda_i\}_{i=1}^{d^2}$ of the covariance matrix of patches. If eigenvalues are sorted in ascending order, *i.e.*, $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{d^2}$, $\boldsymbol{S}$ can be represented as $\boldsymbol{S}_n \cup \boldsymbol{S}_p$ where $\boldsymbol{S}_n = \{\lambda_i\}_{i=1}^{m}$ indicates a set of eigenvalues for redundant dimensions and $\boldsymbol{S}_p = \{\lambda_i\}_{i=m+1}^{d^2}$ indicates a set of eigenvalues for the principal dimensions. Then, the last step of [5] is carefully finding a point that divides $\boldsymbol{S}_n$ and $\boldsymbol{S}_p$, and then calculating the noise variance with $\boldsymbol{S}_n$. For this step, [5] proposed effective *iterative* dimension selection procedure that stops iterations where mean and median of subset $\boldsymbol{S}_n$ are equal. They theoretically verified that this stopping point is a right point to separate $\boldsymbol{S}_n$ and $\boldsymbol{S}_p$. More details about algorithm and theoretical justifications can be found in [5, Section 2]. The original implementation is initialized with $\boldsymbol{S}_p = \varnothing$ and $\boldsymbol{S}_n = \boldsymbol{S}$. By denoting the mean and median value of the subset $\boldsymbol{S}_n$ as γ and ψ respectively, the difference between γ and ψ is iteratively calculated for

*Equal contribution.
†Corresponding author (E-mail: tsmoon@snu.ac.kr)

finding outliers in the subset $\boldsymbol{S}_n$. If $\gamma \neq \psi$, take out the largest value in the subset $\boldsymbol{S}_n$ and put it into the subset $\boldsymbol{S}_p$. This procedure is iterated until $\gamma = \psi$. Then, the final result of noise estimation is obtained by $\sqrt{\gamma}$.

Algorithm 1 Gaussian Noise estimation [5] (Tensorized)

Require: A noisy image $\boldsymbol{Y} \in \mathbb{R}^{N \times N}$, Patch size d

▷ **/* Step1: Decompose image */**

Decompose a noisy image $\boldsymbol{Y}$ into vectorized multiple patches $\{\boldsymbol{y}_t\}_{t=1}^{s}$ with size d^2 where $s = (N - d + 1)^2$

$\boldsymbol{u} \leftarrow \frac{1}{s}\sum_{t=1}^{s}(\boldsymbol{y}_t)$, $\Sigma_y \leftarrow \frac{1}{s}\sum_{t=1}^{s}(\boldsymbol{y}_t - \boldsymbol{u})(\boldsymbol{y}_t - \boldsymbol{u})^T$

▷ **/* Step2: Calculate the eigenvalues */**

Calculate eigenvalues λ_i of Σ_y with ascending order $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{d^2}$

$\boldsymbol{\Lambda} \in \mathbb{R}^{d^2 \times d^2} \leftarrow$ a diagonal matrix where $\Lambda_{ii} = \lambda_i$

▷ **/* Step3: Finding σ without iterative procedure */**

/* Generate two lower triangular matrices */

$\boldsymbol{A}_\Lambda \leftarrow \boldsymbol{A}_1 \cdot \boldsymbol{\Lambda}$ (dot product)

Calculate a mean vector $\boldsymbol{\tau} \in \mathbb{R}^{d^2}$
where $\tau_i = \frac{1}{k_i}\sum_{j=1}^{d^2}(A_\Lambda)_{ij}$ and k_i is the number of nonzero elements in $(\boldsymbol{A}_\Lambda)_i$

$\boldsymbol{A}_\tau \leftarrow \text{Diag}(\boldsymbol{\tau}) \cdot \boldsymbol{A}_1$ (dot product)

/* Identifying the median value with the masking scheme */

Generate masking matrix $\boldsymbol{M}_{big} \in \mathbb{R}^{d^2 \times d^2}$ and $\boldsymbol{M}_{small} \in \mathbb{R}^{d^2 \times d^2}$
where $(M_{big})_{ij} = \mathbb{1}\{(A_\Lambda)_{ij} > (A_\tau)_{ij}\}$ and $(M_{small})_{ij} = \mathbb{1}\{(A_\Lambda)_{ij} < (A_\tau)_{ij}\}$

Generate counting vectors $\boldsymbol{c}_{big} \in \mathbb{R}^{d^2}$ and $\boldsymbol{c}_{small} \in \mathbb{R}^{d^2}$
where $(c_{big})_i = \sum_{j=1}^{d^2}(M_{big})_{ij}$ and $(c_{small})_i = \sum_{j=1}^{d^2}(M_{small})_{ij}$

Generate a masking vector $\boldsymbol{m} \in \mathbb{R}^{d^2}$ where $m_i = \mathbb{1}\{(c_{big})_i == (c_{small})_i\}$.

$\sigma \leftarrow \sqrt{\max(\boldsymbol{m} \odot \boldsymbol{\tau})}$

return σ

As we mentioned in Section 4.1 (manuscript), we replaced this *iterative* procedure in [5] with tensor operations. It is straight-forward to change operations of first and second step to tensor operations, but the last step is not. Thus, we mainly describe the last step. Before describing the tensorized version of [5], we introduce a few more notations. Let's define a lower triangular matrix as $\boldsymbol{A} \in \mathbb{R}^{d^2 \times d^2}$ and a specific lower triangular matrix which has 1 as all nonzero elements as $\boldsymbol{A}_1 \in \mathbb{R}^{d^2 \times d^2}$. Moreover, define $\boldsymbol{M} \in \mathbb{R}^{d^2 \times d^2}$ as masking matrix and $\boldsymbol{m} \in \mathbb{R}^{d^2}$ as masking vector such that $M_{ij} \in \{0, 1\}$ and $m_i \in \{0, 1\}$. $\text{Diag}(\boldsymbol{a})$ is the $d^2 \times d^2$ diagonal matrix whose diagonal elements are the elements of the vector $\boldsymbol{a} \in \mathbb{R}^{d^2}$ and $\odot$ denotes element-wise multiplication.

As can be seen in Algorithm 1, instead of *iterative* procedure for calculating the mean value γ and median value ψ, we generate two lower triangular matrices which can compute all possible mean values and identify whether those mean values are median or not *at once*. For identifying each mean value is a median or not, every possible mean values should be calculated and compared with eigenvalues. Namely, the individual averages of the eigenvalues of each candidate of $\boldsymbol{S}_n$ are computed first, and then each mean value is compared with the eigenvalues of that candidate. Thus, firstly, we make a low triangular matrix $\boldsymbol{A}_\Lambda$ which consists of candidates for $\boldsymbol{S}_n$. For this, $\boldsymbol{A}_\Lambda$ is calculated by the dot product of A_1 and eigenvalue matrix Λ. Considering only the non-zero element, the first row represents the smallest $\boldsymbol{S}_n$ containing only λ_1, and the last row represents the largest $\boldsymbol{S}_n$, *i.e*, $\boldsymbol{S}$. Secondly, we calculate a mean vector $\boldsymbol{\tau}$ and generate a low triangular matrix $\boldsymbol{A}_\tau$ for comparing with $\boldsymbol{A}_\Lambda$. By using $\boldsymbol{A}_\Lambda$, a mean vector $\boldsymbol{\tau}$ consisting of every possible mean values is calculated. Then, $\boldsymbol{A}_\tau$ is calculated by the dot product of $Diag(\boldsymbol{\tau})$ and A_1. Note that the non-zero elements in each row of $\boldsymbol{A}_\tau$ all have the same value, and the single non-zero element in i-th row of $\boldsymbol{A}_\tau$ is the average value of the i-th row of $\boldsymbol{A}_\tau$. Namely, the non-zero elements of the i-th row of $\boldsymbol{A}_\tau$ are composed of i-repeats of the average of i-th row of $\boldsymbol{A}_\tau$.

Now, all possible mean values can be compared with corresponding eigenvalues at once. To identify the median value, two masking matrices $\boldsymbol{M}_{big}$ and $\boldsymbol{M}_{small}$ are generated, derived from conditions that the inequality for the above two lower triangular matrices are opposite. By comparing two counting vectors, the final masking vector $\boldsymbol{m}$ is calculated. Since we assume the eigenvalues are sorted with ascending order, the maximum value of $\boldsymbol{m} \odot \boldsymbol{\tau}$ is the final estimates for the noise variance.

2. Detailed Experimental Results

2.1. Toy example for verifying the Section 5.2 (manuscript)

As we discussed in Section 5.2 (manuscript), whereas the accuracy of PGE-Net for estimating σ is not high, the GAT+BM3D or BP-AIDE framework with the GAT-transformed images (using underestimated $\hat{\sigma}$ of PGE-Net) still shows great blind denoising performance. This somewhat counter-intuitive can be explained by our empirical verification; We observed that the underestimation of PGE-Net for σ is not a major issue in achieving that GAT-transformed image (with the estimated parameters) has the stabilized noise variance close to 1. To verify this, consider the simple toy example shown in Figure 1. We generated four different homogeneous image patches of size 256×256 with four different pixel values $(0.2, 0.7, 0.8, 0.3)$, respectively, and corrupted them with a Poisson-Gaussian noise ($\alpha = 0.1$, $\sigma = 0.02$) (Fig. 1a). Note that due to the source-dependence of the noise, the four image patches have different noise levels as shown in Fig. 1b. Now, Fig. 1c shows the GAT-transformed images with the estimated noise parameters from the PGE-Net ($(\hat{\alpha}, \hat{\sigma})$ for each image is given in Fig.1c), and Fig. 1d shows the corresponding noise of the GAT-transformed images (obtained by subtracting the mean of each image). Notice that while $\hat{\sigma}$ of PGE-Net is indeed much smaller than the true σ, as in Table 3 (manuscript), Fig. 1d shows that the source-dependent noise is almost removed after GAT, and the variance of the remaining noise gets *very close* to 1. Note PGE-Net is trained *exactly* with this objective (Eq.(8) in manuscript), and we believe this example clearly shows why running the BP-AIDE framework with the GAT-transformed images (using underestimated $\hat{\sigma}$ of PGE-Net) still shows great blind denoising performance.

2.2. Comparison with supervised trained methods in SIDD / DND

In this section, we additionally compare the performance of FBI-Denoiser with supervised trained models such as UPI [4] and CycleISP [12]. As we mentioned in Introduction (manuscript), CycleISP and UPI modeled in-camera processing pipeline (ISP) and achieved the state-of-the-art performance in

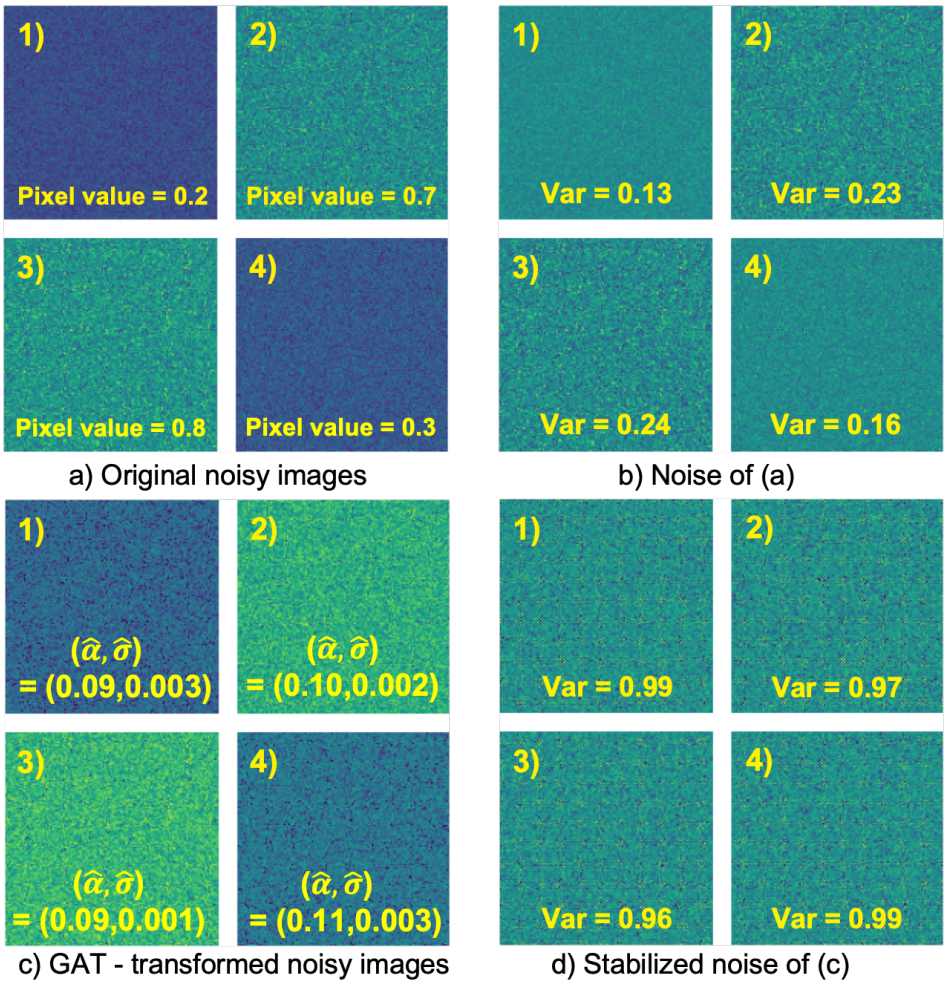

Figure 1. A toy example for Section 5.2 (manuscript)

SIDD and DND dataset by generating noisy and clean pairs of raw-RGB or sRGB images from clean synthetic sRGB images. Their approaches are trained in a supervised way with pairs of generated pairs. Moreover, they used an additional input channel which is composed of per-pixel estimates of the noise standard-deviation for a noisy image. Note the information about Poisson-Gaussian noise parameters is required for getting these per-pixel estimates. However, they assumed that the information about noise parameters of test noisy images is given in both training and evaluation phase which is unrealistic in the real-world application. In evaluation, they used true noise parameters which are given by DND and SIDD websites [1, 2]. Moreover, in training, as can be seen in [4, Section 3.1], they specify the sampling range of noise parameters (α, σ) based on the maximum and minimum values of the noise parameters of test noisy images, and further define more specific sampling ranges by performing regression fitting with the noise parameters of the test noisy images. Namely, they used the additional information about noise characteristics in both training and evaluation phase.

Thus, we recalculated their results in more realistic settings which the information about noise characteristics is not given in training or evaluation. Additionally, we measured the performance of

Table 1. PSNR(dB)/SSIM on SIDD and DND dataset. Red and blue denote the highest and second highest results, respectively, among models except supervised trained models.

Type	Algorithm	SIDD				DND			
		Raw		sRGB		Raw		sRGB	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
unsup	GAT+BM3D	48.52	**0.9800**	34.61	**0.8789**	47.53	0.9761	37.98	0.9203
	N2V	46.30	0.9760	32.85	0.8470	45.41	0.9688	35.82	0.9022
	D-BSN	37.16	0.8390	24.07	0.4999	39.63	0.8642	30.23	0.7095
	BP-AIDE (DND)	-	-	-	-	47.60	0.9732	38.60	0.9259
	FBI-D (DND)	-	-	-	-	47.53	0.9706	38.56	0.9185
	BP-AIDE (SIDD)	**50.45**	**0.9900**	**37.91**	**0.9420**	**47.75**	**0.9770**	**38.79**	**0.9446**
	FBI-D (SIDD)	**50.57**	**0.9900**	**38.07**	**0.9420**	**48.02**	**0.9787**	**38.98**	**0.9451**
sup	UPI (25k, blind) with Liu [9]	47.89	0.9860	36.59	0.9280	46.96	0.9756	38.34	0.9450
	UPI (25k) with Liu [9]	48.95	0.9870	37.15	0.9330	47.58	0.9756	38.92	0.9413
	MCU-Net	48.80	0.9900	36.54	0.8750	-	-	-	-
	UPI (1M)	-	-	-	-	48.89	0.9820	40.17	0.9620
	CycleISP (1M)	52.41	0.9930	39.47	0.9180	49.13	0.9830	40.50	0.9660

them when the number of training pairs is limited to 25000 (originally, 1 million) which is the same number with training patches in FBI-Denoiser. Note that the training code of CycleISP is not available, we only measured the performance of UPI in realistic settings, thus, we mainly analyzed the results of UPI. Since the main difference between CycleISP and UPI lies in the detailed implementation of data conversion step from sRGB to raw-RGB, we believe that results of CycleISP which is trained in a realistic setting will have a similar tendency to the remeasured results of UPI.

The first realistic setting "UPI (25k) with Liu [9]" denotes the result of UPI which is trained with the original sampling procedure of UPI, but evaluated with estimated noise parameters from Liu [9]. The more realistic setting "UPI (25k, blind) with Liu [9]" denotes the result of UPI which is trained with a blind uniform sampling procedure ($\alpha \in [0, 0.16^2], \sigma \in [0, 0.06]$) as in [7], and evaluated with estimated noise parameters from Liu [9]. Note that the inference time for noise estimation is additionally needed for UPI in these realistic settings.

The results of MCU-Net, UPI (1M), and CycleISP were taken from their original papers [3, 4, 12] since their training takes too long. Note the result of UPI which is trained with 1 million noisy images in SIDD is not reported since [4] only reports the DND evaluation results. As can be seen in Table 1, in realistic settings, the performance of UPI is significantly degraded. As a result, FBI-Denoiser outperforms not only unsupervised trained methods but also supervised trained methods including UPI (in realistic settings) and MCU-Net. Note that MCU-Net is also a strong baseline that ranked high in the NTIRE 2020 Challenge on Real Image Denoising - Track1: rawRGB. These results imply that FBI-Denoiser which is trained solely based on single noisy images is enough to compete with supervised trained methods, and given sufficient noisy images, the performance of FBI-Denoiser could be further improved.

2.3. Experiments on the weak noise cases

Although we have shown that our FBI-Denoiser obtains the state-of-the-art denoising performance in real-world noisy benchmark datasets which contain a mixture of various noise levels, the performance of FBI-Denoiser is relatively low in weak noise cases as can be seen (0.01, 0.0002) case of Fivek dataset in Table 5 (manuscript) and this performance degradation becomes severe as the signal to noise ratio (SNR) increases. Note this performance degradation is caused by the original loss function (Eq. 7 in manuscript) of BP-AIDE, not the PGE-Net and FBI-net we proposed. As we described in Section 4.2 (manuscript), we followed the same procedure of BP-AIDE which uses the loss function (Eq. 7 in manuscript) for training a denoiser. FC-AIDE and BP-AIDE achieve higher

performance than N2V by using additional information, the noisy pixel Z_i, as can be seen in their pixelwise affine denoiser and loss function (Eq. 5,7 in manuscript), but, it's not always good to use whole Z_i. If the noise intensity is too strong, it may be better not to use Z_i for training since Z_i is are too noisy. Thus, we believe that the process which decides how much Z_i will be utilized based on the noise intensity would be needed to get better performance. But, BP-AIDE and our FBI-Denoiser set the range of slope coefficients $a_1(w;\cdot)$ as a single fixed range for all noisy images. This range is set between 0 and a specific maximum value as can be seen in Section 5.1 (manuscript). Although they show great performance in real-world noise benchmarks with only this fixed range of $a_1(w;\cdot)$, their performance is limited in some noise cases due to the fixed range of slope coefficients $a_1(w;\cdot)$. Thus, if a loss function that adaptively designates the range of slope coefficient $a_1(w;\cdot)$ according to noise intensity is devised, we expect that overall performance in real-world noisy benchmark datasets would be further enhanced. We left this as our future work.

2.4. Decision rules for the failure cases of Liu [9] and Foi [6] in BP-AIDE

As mentioned in Section 5.2 (manuscript), BP-AIDE has to set up rules for handling the extremely small values of $\hat{\alpha}$ which are the failure cases of Liu [9] and Foi [6]. Two decision rules were suggested in their source code; First, if $\hat{\alpha} < 1E-7$, set $\hat{\alpha}$ as 1 in both training and evaluation. Second, if $\hat{\alpha} < 1E-7$, exclude its noisy image from the training. The best results among them are reported in all experiments. Note that the performance deviation of BP-AIDE is relatively high due to the estimation failure and its handling.

2.5. Inference time of noise estimation

We additionally measured the inference time according to the different size of patches. As seen in Table 2, the inference time of Foi [6] and Liu [9] increases as the patch size increases. In particular, the inference time of Liu [9] drastically increased when the patch size is doubled. On the other hand, PGE-Net keeps the inference time almost constant. Thus, we verified that the efficiency of PGE-Net outperforms other algorithms in all various sized patches.

Table 2. Experimental results of inference time of noise estimation

Dataset	Patch size	Foi [6]	Liu [9]	PGE-Net
Fivek	256x256	0.69	0.72	0.0017
	512x512	2.5	1.8	0.0020
	1024x1024	2.83	6.88	0.0023

2.6. Network architecture of PGE-Net

Table 3 compares the results of GAT+BM3D using estimated noise parameters from two PGE-Nets which have the different number of layers. The experimental setting is same as ablation studies on PGE-Net in Section 5.4 (manuscript), which uses the uniformly sampled noise parameters ($\alpha \in [0, 0.16^2], \sigma \in [0, 0.06]$). As can be seen in Table 3, the difference in performance between 3-layer and 1-layer is *very* small. Thus, it is highly likely that there will be a simpler and more efficient network architecture than PGE-Net, which simply adopted three layer U-Net, but we did not attempt to find the optimized architecture of PGE-Net and left it as our future work.

Table 3. Network architecture of PGE-Net

Algorithms	**PGE-Net (3 layer)**	**PGE-Net (1 layer)**
GAT+BM3D PSNR/SSIM	**38.72 / 0.9350**	38.61 / 0.9340

2.7. Visualization Results on DND

We visualized an additional benchmark image in DND [11]. We clearly observe that FBI-Denoiser and BP-AIDE achieve a superior performance in terms of both PSNR and SSIM compared with other baselines. However, note FBI-Denoiser restored the detailed texture better than BP-AIDE with a faster reference time.

Figure 2. Visualization results of DND

References

[1] Dnd Benchmark Website. https://noise.visinf.tu-darmstadt.de/. 4

[2] Sidd Benchmark Website. https://www.eecs.yorku.ca/~kamel/sidd/. 4

[3] Long Bao, Zengli Yang, Shuangquan Wang, Dongwoon Bai, and Jungwon Lee. Real image denoising based on multi-scale residual dense block and cascaded u-net with block-connection. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 448–449, 2020. 5

[4] Tim Brooks, Ben Mildenhall, Tianfan Xue, Jiawen Chen, Dillon Sharlet, and Jonathan T Barron. Unprocessing images for learned raw denoising. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11028–11037, 2019. 3, 4, 5

[5] Guangyong Chen, Fengyuan Zhu, and Pheng Ann Heng. An efficient statistical method for image noise level estimation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 477–485, 2015. 1, 2

[6] Alessandro Foi, Mejdi Trimeche, Vladimir Katkovnik, and Karen Egiazarian. Practical poissonian-gaussian noise modeling and fitting for single-image raw-data. *IEEE Transactions on Image Processing*, 17(10):1737–1754, 2008. 6

[7] Shi Guo, Zifei Yan, Kai Zhang, Wangmeng Zuo, and Lei Zhang. Toward convolutional blind denoising of real photographs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1712–1722, 2019. 5

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations (ICLR), year = 2015*. 1

[9] Xinhao Liu, Masayuki Tanaka, and Masatoshi Okutomi. Practical signal-dependent noise parameter estimation from a single noisy image. *IEEE Transactions on Image Processing*, 23(10):4361–4371, 2014. 5, 6

[10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 1

[11] Tobias Plotz and Stefan Roth. Benchmarking denoising algorithms with real photographs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1586–1595, 2017. 7

[12] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. Cycleisp: Real image restoration via improved data synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2696–2705, 2020. 3, 5