Synthesizing Camera Noise using Generative Adversarial Networks

Bernardo Henz, Eduardo S. L. Gastal, and Manuel M. Oliveira

APPENDIX A
IMPLEMENTATION DETAILS

Our GAN implementation was inspired by Zhu et al.'s work [1]. This includes the use of residual blocks [2], instance normalization [3], and PatchGANs in the discriminators [4]. However, we made some important modifications, especially on the generators architecture, for improving the handling of high frequencies. The final architecture can be found in Appendix B. For training, we used the Adam optimizer [5] with default parameters and learning rate of 0.0002, linearly decaying to zero in the course of 2 epochs (totaling more than 800k iteration updates). We used batch size of 1. Specifically for training the discriminator, we followed the idea of Shrivastava et al. [6] of updating the discriminator using a history of generated images.

APPENDIX B
NETWORK ARCHITECTURES

Generators' architecture. The architecture of our F and G generators were designed to better handle high-frequency content. Let `Ck-f` denote a $k \times k$ 2D convolution layer with f filters, and let `Res-k` represent a residual block consisting of a Reflection padding, Conv layer, Reflection padding and another Conv layer, in this order, where both Conv layers use 3×3 kernels and k filters. `Ref` represents a Reflection padding layer, and `BN` is a batch-normalization layer. All Conv layers are followed by an Instance-Normalization layer and a ReLU. The architecture of the generators is as follows: `Ref, C7-64, Res-64, Res-64, Res-64, Res-64, Res-64, Ref, C7-64, BN` (Fig. 4, top).

Discriminators' architecture. We have used the same architecture as Zhu et al. [1] for the discriminators D_X and D_Y. Following the same notation of names defined above, and considering that for the discriminator all Conv layers are followed by Instance-Normalization and leaky-ReLU (with decay of 0.2), the discriminator architecture consists of: `C4-64, C4-128, C4-256, C4-512` (Fig. 4, bottom).

• *Bernardo Henz, Eduardo S. L. Gastal and Manuel M. Oliveira are with UFRGS. E-mails: bhenz, eslgastal, oliveira@inf.ufrgs.br.*

Manuscript received December 13, 2019.

TABLE 1
Chosen σ for different pair of domains (ISO levels).

Mapping	Chosen σ
$100, 400$	1.5
$100, 800$	2.5
$100, 1600$	3.5
$100, 3200$	4.5
$200, 1600$	2.5

APPENDIX C
CHOOSING σ FOR LOW-FREQUENCY LOSS

For extracting the low-frequency content used for computing the low-frequency-consistency loss term, we perform a Gaussian Blur with a pre-defined σ on each patch. The noise found in higher ISO levels (normally encoded in the higher frequencies) has greater intensity compared to lower ISO levels. Thus, we use different σ values for different mappings (e.g. ISO 100→1600 vs. ISO 100→3200). But rather than choosing the values empirically, we pick σ based on the dataset containing scenes captured on each ISO level (Section 5.2). Thus, for each pair of domains (e.g., 100→1600, 200→1600), we linearly search for the σ value that satisfies the following inequality:

$$\mathrm{MSE}(\mathbb{G}_\sigma(x), \mathbb{G}_\sigma(y)) < 0.005, \tag{1}$$

where MSE is the mean-squared error between the blurred versions of x (image belonging to domain X) and y (belonging to domain Y), and $\mathbb{G}$ is a Gaussian blur implemented as a convolution by a 21×21 kernel with the appropriate σ (measured in pixels). We choose a low-threshold in Eq. (1) (0.005) to enforce the blurred versions to be at least similar. The σ values chosen for each mapping of the experiments can be found in Table 1.

APPENDIX D
EVALUATION NETWORK ARCHITECTURES

All classifiers, for natural versus artificial noise (Section 6.2), for identifying the ISO level of a given patch (Section 5.2), and for noise-model classification (Section 6.3) share similar architectures. All networks take as input a 128×128 RGB patch, and the patch's Fourier amplitude coefficients for each channel in log-space and in their shifted version (*i.e.*, with the DC term translated to the center of the matrix

representation). Thus, each classifier takes a $128 \times 128 \times 6$ input, and outputs one (for the case of natural noise classification), six (for the case of ISO level identification), or seven probabilities (for discriminating among noise models).

Let `Ck-f` denote a $k \times k$ Conv layer with `f` filters and stride $= k/2 + 1$, followed by a ReLU; and `M2` a 2×2 max-pooling layer. Also, let `Fl` denote a flattening layer, `FC-n` a fully-connected layer with n hidden units, also followed by a ReLU. Let `Dp-d` be a dropout layer with d representing the fraction (between $[0, 1]$) of units that will be randomly set to 0, `Sig` the sigmoid activation function, and `Soft` the Softmax activation function. The architecture for the natural-noise classifier is as follows: `C5-256, C5-256, C5-256, M2, C3-256, C3-256, C3-256, M2, C3-256, C3-256, M2, C3-256, M2, Fl, FC-128, Dp-0.2, FC-64, Dp-0.1, FC-1, Sig`.

The architecture of the ISO-level classifier is similar, replacing the last two layers with: `FC-6, Soft`.

The architecture of the noise-model classifier replaces the last two layers with: `FC-7, Soft`.

The natural-noise classifier uses the *binary cross-entropy* as loss function, while the ISO-level identifier and noise-model classifier use the *categorical cross-entropy*. All network training used the Adam optimizer with default parameters and learning rate of 0.001, batch size of 12, and trained for 2000 iterations. We also employed data augmentation: horizontal and vertical flips, random crops, and random 90° rotations.

APPENDIX E
COMPUTATION OF KL DIVERGENCE AND KS-VALUE

We use the following Python methods for computing the KL divergence and KS-value for our experiments, where p and q are the normalized histograms of noise being compared:

```
import numpy as np

def kl_div(p, q):
    idx = (p > 0) & (q > 0)
    p = p[idx]
    q = q[idx]
    return np.sum(p * np.log(p/q))

def ks_value(p, q):
    cum_p = np.cumsum(p)
    cum_q = np.cumsum(q)
    return np.max(np.abs(cum_p - cum_q))
```

Each noise histogram is computed from pixels obtained by the subtraction of the clean image from the noisy one, which extracts only the (per channel) additive noise component. Input images are stored with 8-bits per channel and as such the noise components are in the range $[-255, 255]$, but mostly concentrated around zero. Histograms are computed with the numpy.histogram function with 50 bins evenly distributed in $[-50, 50]$, and two additional bins for the extremes: "bins = np.concatenate(([-256], np.arange(-50, 50, 2), [256]), axis=0) - 0.1". The bins' intervals are shifted by -0.1 to avoid quantization artifacts. Finally, as mentioned in Section 6.4, we compute these metrics for each color channel, which are then averaged together.

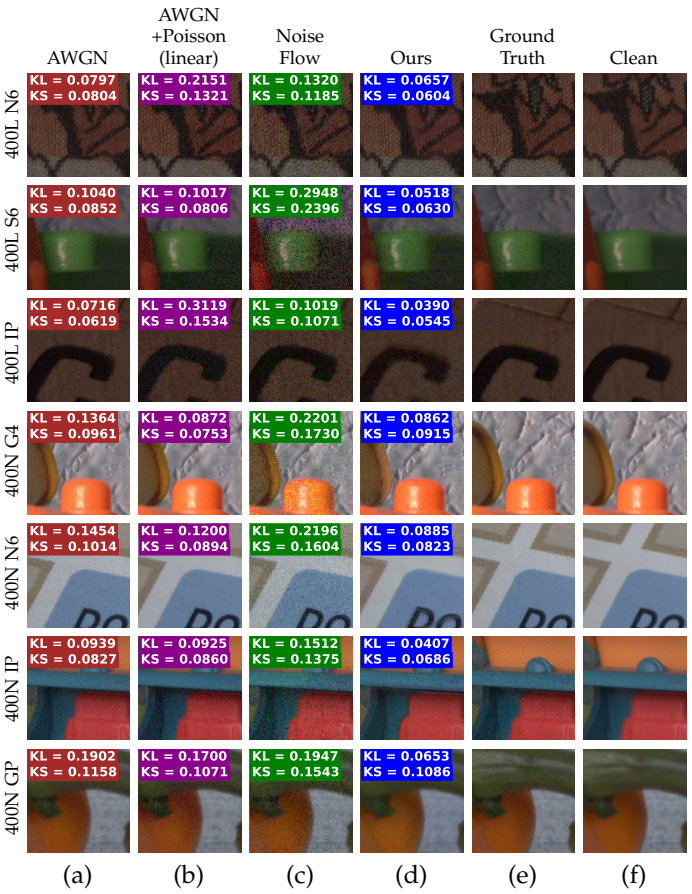

Fig. 1. Comparison among noise models for synthesizing noise for ISO 400. (a) Patch corrupted by AWGN (using average std computed from the paired dataset, in this case $\sigma_{400} = 0.0420$); (b) patch corrupted by AWGN+Poisson in linear space; (c) patch corrupted by Noise Flow (applied in raw space); (d) patch corrupted by our GAN_{SIDD}; (e) the noisy patch; and (f) the clean patch.

APPENDIX F
COMPUTATION OF KL DIVERGENCE AND KS-VALUE

Table 2 compares the KS divergence and KL-values over all population patches (see Section 6.4) for the following noise models: AWGN and AWGN+Poisson (both in sRGB and linear space), Noise Flow, and GAN_{SIDD}.

APPENDIX G
ADDITIONAL COMPARISONS

Here we show more comparisons among different noise models. Figs. 1 to 4 compare closeups of photographs captured by several cameras and lighting conditions for ISO 400, ISO 800, ISO 1600, and ISO 3200, respectively. Note how the results produced by our models consistently achieve better (lower) KL and KS scores, and look more similar to the corresponding ground truth images. For better visualization, we opt to show only the most competitive noise models: AWGN, AWGN+Poisson (in linear space), Noise Flow, and ours.

APPENDIX H
RESIDUAL COMPARISON

Fig. 5 exhibit only the synthesized noise (residual), i.e., the difference between the noisy and clean versions. For better

TABLE 2
KL divergences and KS values for different noise models, ISO values, and lighting conditions ([L]ow and [N]ormal light brightness levels). Best (lower) values are highlighted in bold. These values were measured over the whole population of image patches. Notice how our method achieves better values for these metrics across all but one case.

		ISO 400		ISO 800		ISO 1600		ISO 3200	
		L	N	L	N	L	N	L	N
KL divergence	AWGN	0.0910	0.1063	0.0938	0.1108	0.1780	0.1478	0.0765	0.0796
	AWGN (linear)	0.3204	0.1358	0.2053	0.1009	0.0731	0.1785	0.0580	0.0374
	AWGN+Poisson	0.0559	0.0912	0.0938	0.1110	0.1782	0.1478	0.0764	0.0807
	AWGN+Poisson (linear)	0.1358	0.0781	0.2053	0.1010	0.0732	0.1786	0.0784	0.0383
	NoiseFlow	0.1052	0.1557	0.0593	0.1140	0.0517	0.0634	0.0801	0.0520
	GAN_{SIDD}	**0.0091**	**0.0323**	**0.0104**	**0.0249**	**0.0228**	**0.0129**	**0.0339**	**0.0188**
KS value	AWGN	0.0693	0.0787	0.0842	0.0909	0.1096	0.1237	0.1100	0.0896
	AWGN (linear)	0.1700	0.1000	0.1521	0.0726	0.0937	0.1277	0.0784	0.0629
	AWGN+Poisson	0.1806	0.0693	0.0842	0.0910	0.1096	0.1236	0.1100	0.0895
	AWGN+Poisson (linear)	0.1000	0.0668	0.1520	0.0726	0.0937	0.1277	0.0784	**0.0628**
	NoiseFlow	0.1138	0.1422	0.0900	0.1057	0.0929	0.0821	0.1270	0.0925
	GAN_{SIDD}	**0.0333**	**0.0564**	**0.0404**	**0.0550**	**0.0600**	**0.0643**	**0.0761**	0.0677

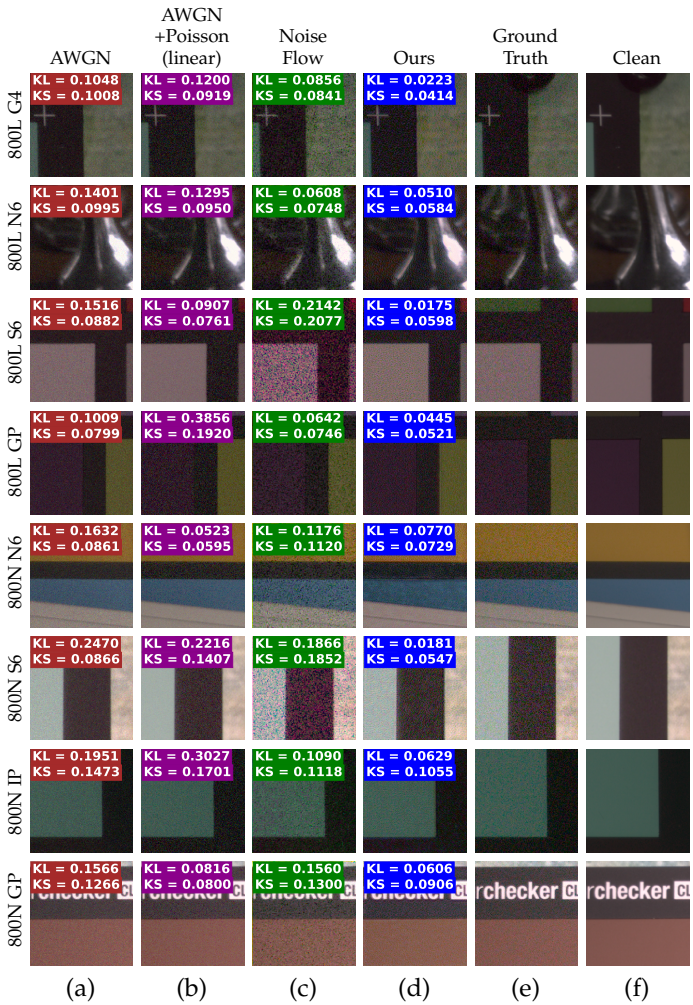

Fig. 2. Comparison among noise models for synthesizing noise for ISO 800. (a) Patch corrupted by AWGN (using average std computed from the paired dataset, in this case $\sigma_{800} = 0.0536$); (b) patch corrupted by AWGN+Poisson in linear space; (c) patch corrupted by Noise Flow (applied in raw space); (d) patch corrupted by our GAN_{SIDD}; (e) the noisy patch; and (f) the clean patch.

visualization, we take the absolute value and multiply it by 3. Notice how our method better mimics camera noise, specially regarding the size of noise grain and color distribution (check residual images).

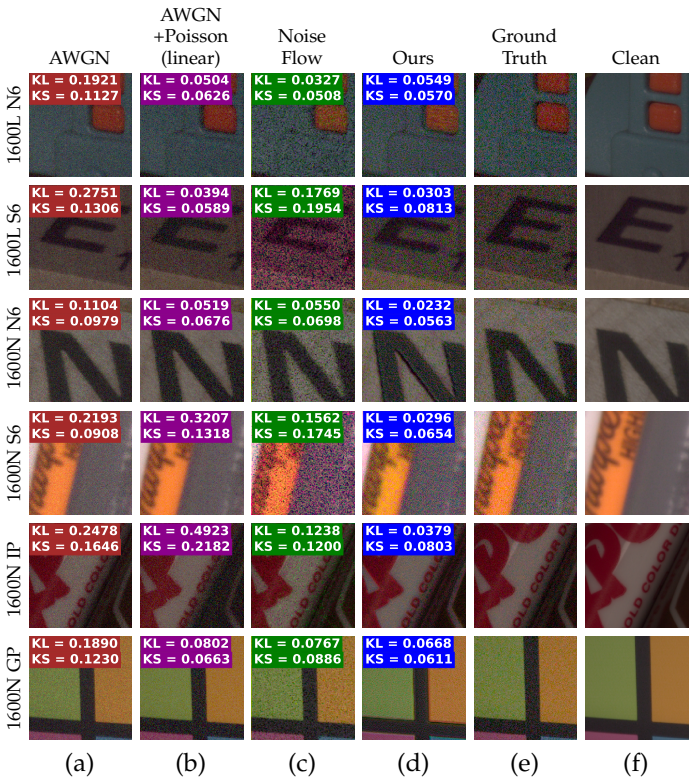

Fig. 3. Comparison among noise models for synthesizing noise for ISO 1600. (a) Patch corrupted by AWGN (using average std computed from the paired dataset, in this case $\sigma_{1600} = 0.0714$); (b) patch corrupted by AWGN+Poisson in linear space; (c) patch corrupted by Noise Flow (applied in raw space); (d) patch corrupted by our GAN_{SIDD}; (e) the noisy patch; and (f) the clean patch.

APPENDIX I
DATA AUGMENTATION ON DENOISER TRAINING

For the DnCNN experiments described in Section 7, besides using random 90^o rotations, we also augmented the training dataset using patches defined as

$$\alpha\, x_{noisy} + (1 - \alpha)\, y_{clean},$$

where x_{noisy} is a noisy patch synthesized by the noise model, y_{clean} is the corresponding clean image (noise free), and α is an interpolation parameter whose value is randomly selected from a uniform distribution in $[0.7, 1]$ (values chosen empirically). Despite its simplicity, such strategy

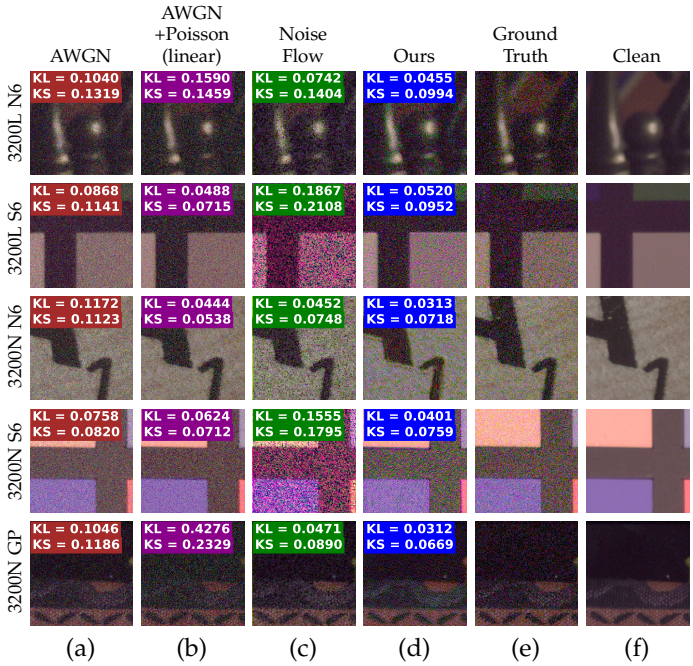

Fig. 4. Comparison among noise models for synthesizing noise for ISO 3200. (a) Patch corrupted by AWGN (using average std computed from the paired dataset, in this case $\sigma_{3200} = 0.1352$); (b) patch corrupted by AWGN+Poisson in linear space; (c) patch corrupted by Noise Flow (applied in raw space); (d) patch corrupted by our GAN_{SIDD}; (e) the noisy patch; and (f) the clean patch.

helped to improve the denoiser's PSNR (around 2 dB) by delaying overfitting (our optimizer was able to train an additional epoch when using such augmentation).

REFERENCES

[1] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *IEEE ICCV*, 2017, pp. 2242–2251.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.

[3] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *CoRR*, vol. abs/1607.08022, 2016.

[4] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CVPR*, 2017.

[5] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 12 2014.

[6] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *IEEE CVPR*, 2017.

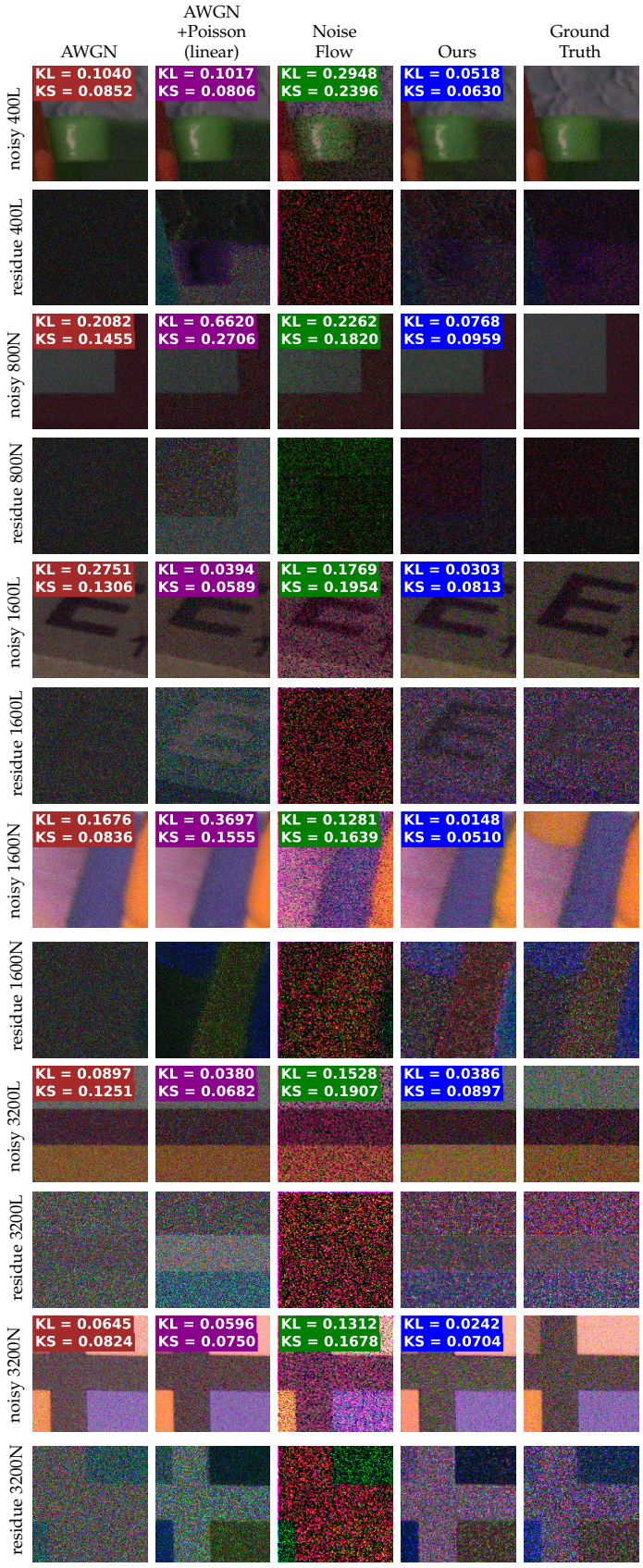

Fig. 5. Comparison of synthesized noisy images and corresponding noise (residual) produced by the various methods. For better visualization, contrast of the residual images has been enhanced by factor of $3\times$. Our results better mimic the noise found in digital photographs (note noise grain and color distribution).