Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings

STEVE BAKO*, University of California, Santa Barbara
THIJS VOGELS*, ETH Zürich & Disney Research
BRIAN MCWILLIAMS, Disney Research
MARK MEYER, Pixar Animation Studios
JAN NOVÁK, Disney Research
ALEX HARVILL, Pixar Animation Studios
PRADEEP SEN, University of California, Santa Barbara
TONY DEROSE, Pixar Animation Studios
FABRICE ROUSSELLE, Disney Research

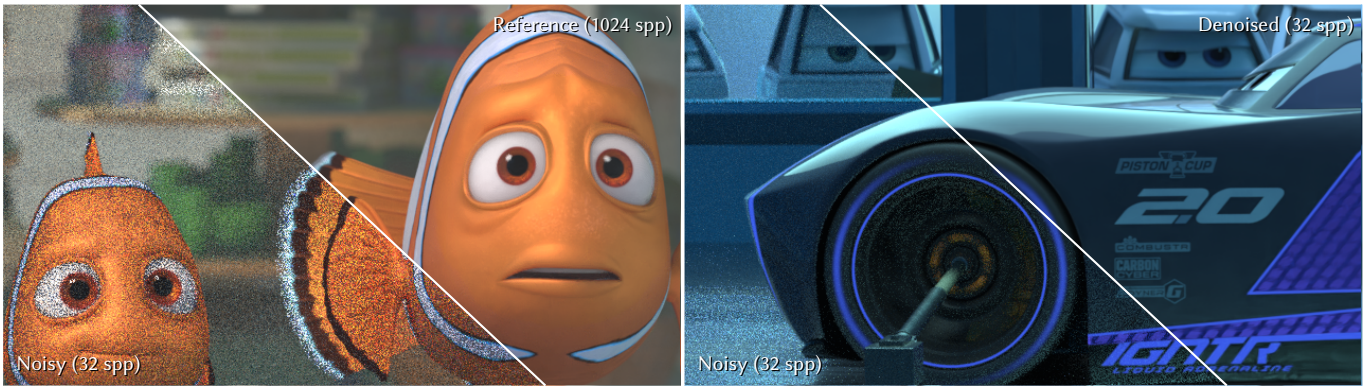


Fig. 1. We introduce a deep learning approach for denoising Monte Carlo-rendered images that produces high-quality results suitable for production. We train a convolutional neural network to learn the complex relationship between noisy and reference data across a large set of frames with varying distributed effects from the film *Finding Dory* (left). The trained network can then be applied to denoise new images from other films with significantly different style and content, such as *Cars 3* (right), with production-quality results.

Regression-based algorithms have shown to be good at denoising Monte Carlo (MC) renderings by leveraging its inexpensive by-products (e.g., feature buffers). However, when using higher-order models to handle complex cases, these techniques often overfit to noise in the input. For this reason, supervised learning methods have been proposed that train on a large collection of reference examples, but they use explicit filters that limit their denoising ability. To address these problems, we propose a novel, supervised learning approach that allows the filtering kernel to be more complex and general by leveraging a deep convolutional neural network (CNN) architecture. In one embodiment of our framework, the CNN directly predicts the final denoised pixel value as a highly non-linear combination of the input features. In a second approach, we introduce a novel, kernel-prediction network which uses the CNN to estimate the local weighting kernels used to compute each denoised pixel from its neighbors. We train and evaluate our networks on production data and observe improvements over state-of-the-art MC denoisers, showing that our methods generalize well to a variety of scenes. We conclude by analyzing various components of our architecture and identify areas of further research in deep learning for MC denoising.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; *Rendering*; Ray tracing;

Additional Key Words and Phrases: Monte Carlo rendering, Monte Carlo denoising, global illumination

ACM Reference format:
Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. 2017. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Trans. Graph.* 36, 4, Article 97 (July 2017), 14 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073708

1 INTRODUCTION

In recent years, physically-based image synthesis has become widespread in feature animation and visual effects [Keller et al. 2015].

*Joint first authors

DOI: http://dx.doi.org/10.1145/3072959.3073708

Fueled by the desire to produce photorealistic imagery, many production studios have switched their rendering algorithms from REYES-style micropolygon architectures [Cook et al. 1987] to physically-based Monte Carlo (MC) path tracing [Kajiya 1986]. While MC rendering algorithms can satisfy strict quality requirements, they do so at an immense computational cost and with convergence characteristics that require long rendering times for noise-free images, especially for scenes with complex light transport.

Fortunately, recent postprocess, image-space, general MC denoising algorithms have demonstrated it is possible to achieve high-quality results at considerably reduced sampling rates (see Zwicker et al. [2015] and Sen et al. [2015] for an overview), and commercial renderers are now incorporating these techniques. For example, Chaos Group's VRay renderer, the Corona renderer, and Pixar's RenderMan now ship with integrated denoisers. Moreover, many production houses are developing their own internal solutions [Goddard 2014] or using third-party tools (e.g., the Altus denoiser).

Although a wide variety of image-space MC denoising approaches have been proposed, most state-of-the-art techniques use a regression framework [Moon et al. 2014; Bitterli et al. 2016]. Improvements have been achieved thanks to more robust distance metrics, higher order regression models, and diverse auxiliary buffers tailored to specific light transport components. These advances, however, have come at the cost of ever-increasing complexity, while offering progressively diminishing returns. This is partially because higher-order regression models are prone to overfitting to the noisy input.

To circumvent the noise-fitting problem, Kalantari et al. [2015] recently proposed an MC denoiser based on supervised learning that is trained with a set of examples of noisy inputs and the corresponding reference outputs. However, this approach used a relatively simple multi-layer perceptron (MLP) for the learning model and was trained on a small number of scenes. More importantly, their approach hardcoded the filter to either be a joint bilateral or joint non-local means, which limited the flexibility of their system.

To address these shortcomings, in this paper we propose a novel, supervised learning framework that allows for more complex and general filtering kernels by leveraging deep convolutional neural networks (CNNs). The ever-increasing amount of production data offers the large and diverse dataset required for training a deep CNN to learn the complex mapping between a large collection of noisy inputs and corresponding references. The advantage is that CNNs are able to learn powerful, non-linear models for such a mapping by leveraging information from the entire set of training images, not just a single input as in many of the previous approaches. Moreover, once trained, CNNs are fast to evaluate and do not require manual tuning or parameter tweaking. Finally, such a system can more robustly cope with noisy renderings to generate high-quality results on a variety of MC effects without overfitting.

Although our approach could be used for other applications of physically-based image synthesis, in this work we focus on high-quality denoising of static images for production environments. Specifically, our contributions are as follows:

- Our main contribution is the first deep learning solution for denoising MC renderings which was trained and evaluated on actual production data. Our architecture performs on par or better than existing state-of-the-art denoising methods.
- Inspired by the standard approach of estimating a pixel value as a weighted average of its noisy neighborhood, we propose a novel kernel-prediction CNN architecture that computes the locally optimal neighborhood weights. This provides regularization for a better training convergence rate and facilitates use in production environments.
- Finally, we explore and analyze the various processing and design decisions of our system, including our two-network framework for denoising diffuse and specular components of the image separately, and a simple normalization procedure that significantly improves our approach (as well as previous methods) for images with high dynamic range.

2 PREVIOUS WORK

Both MC denoising and deep learning have been the focus of extensive research, the scope of which is too large to be covered in this paper. Therefore, for MC denoising, we will restrict ourselves to the most directly related of the *a posteriori* methods, which treat the renderer as a black box. For a more complete overview, we refer readers to the review by Zwicker et al. [2015]. For deep learning, we will focus on convolutional neural networks [LeCun et al. 2015].

2.1 Image-space General Monte Carlo Denoising

We begin by discussing image-space denoising methods that filter the noise from general distributed Monte Carlo effects (e.g., depth of field, motion blur, glossy reflections, and global illumination). The most successful state-of-the-art methods build on the idea of using generic non-linear image-space filters [Rushmeier and Ward 1994] and auxiliary feature buffers as a guide to improve the robustness of the filtering process [McCool 1999]. A key development introduced by Sen and Darabi [2012] was to leverage *noisy* auxiliary buffers in a joint bilateral filtering scheme, where the bandwidths of the various auxiliary features are derived from the sample statistics. Li et al. [2012] later proposed to estimate the filter error with the SURE metric [Stein 1981] to set the filter bandwidths, while Moon et al. [2014] used asymptotic bias analysis to do so. In our system, the training procedure implicitly learns the appropriate weighting of the various auxiliary buffers.

A particularly successful application of these ideas was to use the non-local means filter of Buades et al. [2005] in a joint filtering scheme [Rousselle et al. 2013; Moon et al. 2013; Zimmer et al. 2015]. The enduring appeal of the non-local means filter for denoising MC renderings is largely due to its versatility. Indeed, more powerful image-space filters, such as BM3D [Dabov et al. 2006], have seen less use for MC denoising with some notable exceptions [Kalantari and Sen 2013]. This is due to the fact that they have not yet been successfully extended to leverage auxiliary buffers, a key component of current state-of-the-art methods. In our work, we propose to use machine learning instead of a fixed filter, which not only has been shown to perform on par with state-of-the-art image filters [Burger et al. 2012], but also allows us to feed our network with auxiliary buffers and leverage the robustness they provide.

Recently, it was shown that joint filtering methods, such as those cited above, can be interpreted as linear regressions using a zero-order model, and that, more generally, most state-of-the-art MC denoising techniques are based on a linear regression using a zero-

or first-order model [Moon et al. 2014; Bitterli et al. 2016]. Methods leveraging a first-order model have proved to be very useful for MC denoising [Bauszat et al. 2011; Moon et al. 2014; Bitterli et al. 2016], and while higher-order models have also been explored [Moon et al. 2016], it must be done carefully to prevent overfitting to the input noise. In contrast, the deep CNN used in our system can offer powerful non-linear mappings, without overfitting, by learning the complex relationship between noisy and reference data across a large training set.

Recently, Kalantari et al. [2015] proposed a learning-based filtering approach, which is closely related to our own work. However, their network uses a fixed filter as a back-end, and therefore inherits its limitations. In contrast, we propose a solution that implicitly learns the filter itself and therefore produces better results.

Finally, there is concurrent work by Chakravarty et al. [2017] that also applies deep learning to denoise Monte Carlo renderings, but it targets different applications than ours focusing more on interactive renderings with low sample counts instead of high-end, production-quality renderings. To facilitate comparisons between the two approaches, we both compare to a previous baseline method in our respective papers (see Sec. 6).

2.2 Convolutional Neural Networks

In recent years, convolutional neural networks (CNNs) have emerged as a ubiquitous model in machine learning, achieving state-of-the-art performance in a diverse range of tasks such as image classification [He et al. 2016], speech processing [Oord et al. 2016], and many others. CNNs have also been used a great deal for a variety of low-level, image-processing tasks. In particular, several works have considered the problem of natural image denoising [Xie et al. 2012; Zhang et al. 2016; Gharbi et al. 2016] and the highly related problem of image super-resolution [Yang et al. 2016].

However, a naïve application of a convolutional network to MC denoising exposes a wide range of issues that is handled in our framework. First, training a network to compute a denoised color from only a raw, noisy color buffer causes overblurring since the network cannot distinguish between scene noise and scene detail. Moreover, since the rendered images have high dynamic range, direct training can cause unstable weights (e.g., extremely large or small values) that cause bright ringing and color artifacts in the final image. By preprocessing our features as well as exploiting the diffuse/specular decomposition, we are able to preserve important detail while denoising the image. Furthermore, we introduce the novel kernel prediction architecture (Sec. 4.1) to keep training tractable/stable. In Sec. 7, we motivate and explore how these design decisions affect the performance of our system.

3 THEORETICAL BACKGROUND

Before introducing our proposed denoising framework, we first define our notation and present the interpretation of denoising as a supervised learning problem. To begin, the samples output by a typical MC renderer can be averaged down into a vector of per-pixel data, $\mathbf{x}_p = \{\mathbf{c}_p, \mathbf{f}_p\}$, where $\mathbf{x}_p \in \mathbb{R}^{3+D}$. Here, $\mathbf{c}_p$ represents the RGB color channels and $\mathbf{f}_p$ is a set of D auxiliary features (e.g., surface normals, depth, albedo, and their corresponding variances).

The goal of MC denoising is to obtain a filtered estimate $\widehat{\mathbf{c}}_p$ that is as close as possible to a *ground truth* result $\overline{\mathbf{c}}_p$ that would be obtained as the number of samples goes to infinity. This estimate is usually computed by operating on a block $\mathbf{X}_p$ of per-pixel vectors around the neighborhood $\mathcal{N}(p)$ to produce the filtered output at pixel p. Given a denoising function $g(\mathbf{X}_p; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$, the *ideal* denoising parameters at every pixel can be written as:

$$\widehat{\boldsymbol{\theta}}_p = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \ell(\overline{\mathbf{c}}_p, g(\mathbf{X}_p; \boldsymbol{\theta})), \tag{1}$$

where the denoised value is $\widehat{\mathbf{c}}_p = g(\mathbf{X}_p; \widehat{\boldsymbol{\theta}}_p)$ and $\ell(\overline{\mathbf{c}}, \widehat{\mathbf{c}})$ is a loss function between the ground truth value, $\overline{\mathbf{c}}$, and the denoised value.

Clearly, optimizing Eq. 1 is impossible since ground truth values $\overline{\mathbf{c}}$ are not available at run time. Instead, most MC denoising algorithms estimate the denoised color at a pixel by replacing $g(\mathbf{X}_p; \boldsymbol{\theta})$ with $\boldsymbol{\theta}^\top \phi(\mathbf{x}_q)$, where function $\phi : \mathbb{R}^{3+D} \to \mathbb{R}^M$ is a (possibly non-linear) feature transformation with parameters $\boldsymbol{\theta}$. They then solve the following *weighted* least-squares regression on the color values, $\mathbf{c}_q$, around the neighborhood, $q \in \mathcal{N}(p)$:

$$\widehat{\boldsymbol{\theta}}_p = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{q \in \mathcal{N}(p)} \left(\mathbf{c}_q - \boldsymbol{\theta}^\top \phi(\mathbf{x}_q)\right)^2 \omega(\mathbf{x}_p, \mathbf{x}_q), \tag{2}$$

where the final denoised pixel value is computed as $\widehat{\mathbf{c}}_p = \widehat{\boldsymbol{\theta}}_p^\top \phi(\mathbf{x}_p)$. In this case, the regression kernel $\omega(\mathbf{x}_p, \mathbf{x}_q)$ helps to ignore values that are corrupted by noise, e.g., by changing the feature bandwidths in a joint bilateral filter [Sen and Darabi 2012]. Note that ω could potentially also operate on patches, rather than single pixels, as in the case of a joint non-local means filter.

As observed previously [Moon et al. 2014; Bitterli et al. 2016], some of the previous methods can be classified as zero-order methods with $\phi_0(\mathbf{x}_q) = 1$ [Sen and Darabi 2012; Rousselle et al. 2013], first-order methods with $\phi_1(\mathbf{x}_q) = [1; \mathbf{x}_q]$ [Moon et al. 2014], or higher-order methods [Moon et al. 2016] where $\phi_m(\mathbf{x}_q)$ enumerates all the polynomial terms of $\mathbf{x}_q$ up to degree m (see Bitterli et al. [2016] for a detailed discussion).

With this formulation in mind, the limitations of these individual approaches can be understood in terms of bias-variance tradeoff [Friedman et al. 2001]. Zero-order methods are equivalent to using an explicit function such as a joint bilateral [Li et al. 2012] or non-local means filter [Rousselle et al. 2012]. These represent a restrictive class of functions that trade reduction in variance for a high modeling bias. Although a well-chosen weighting kernel, ω, can yield good performance [Rousselle et al. 2013; Kalantari et al. 2015], such approaches are fundamentally limited by their explicit filters. In this work, we seek to remove this limitation by making the filter kernel more flexible and powerful.

Furthermore, using a first- or higher-order regression increases the complexity of the function, but is prone to overfitting as $\widehat{\boldsymbol{\theta}}_p$ is estimated locally using only a *single* image and can easily fit to the noise. To address this problem, Kalantari et al. [2015] proposed to take a *supervised learning* approach to estimate g using a dataset $\mathcal{D}$ of N example pairs of noisy image patches and their corresponding reference color information, $\mathcal{D} = \{(\mathbf{X}_1, \overline{\mathbf{c}}_1), \ldots, (\mathbf{X}_N, \overline{\mathbf{c}}_N)\}$, where $\overline{\mathbf{c}}_i$ corresponds to the reference color at the center of patch $\mathbf{X}_i$ located at pixel i of one of the many input images. Here, the goal is to find parameters of the denoising function, g, that minimize

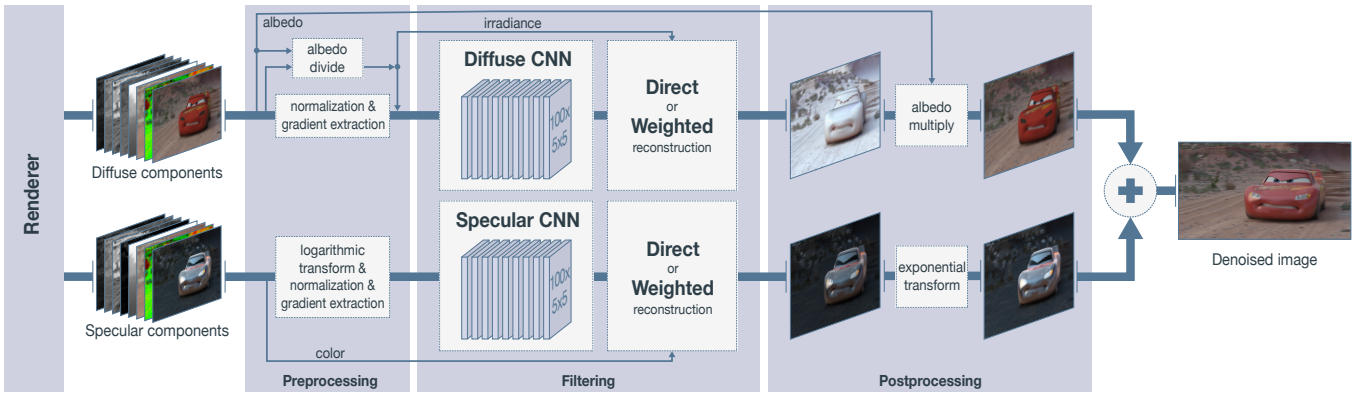


Fig. 2. An overview of our general framework. We start by preprocessing diffuse and specular data coming from the rendering system independently, and then feed the information to two separate networks which denoise the diffuse and specular illumination, respectively. The output from each network undergoes reconstruction and postprocessing before being combined to obtain the final, denoised image.

the average loss with respect to the reference values across all the patches in $\mathcal{D}$:

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \ell(\bar{\mathbf{c}}_i, g(\mathbf{X}_i; \boldsymbol{\theta})), \tag{3}$$

In this case, the parameters, $\boldsymbol{\theta}$, are optimized with respect to all the reference examples, not the noisy information as in Eq. 2. If $\widehat{\boldsymbol{\theta}}$ is estimated on a large and representative training data set, then it can adapt to a wide variety of noise and scene characteristics.

However, the approach of Kalantari et al. [2015] has several limitations, the most important of which is that the function $g(\mathbf{X}_i; \boldsymbol{\theta})$ was hardcoded to be either a joint bilateral or joint non-local means filter with bandwidths provided by a multi-layer perceptron (MLP) with trained weights, $\boldsymbol{\theta}$. Because the filter was fixed, the resulting system lacked the flexibility to handle the wide range of Monte Carlo noise that can be encountered in production environments.

To address this limitation, we consider extending the supervised learning approach to handle significantly more complex functions for g, which results in more flexibility while still avoiding overfitting. Thus, we can reduce modeling bias while simultaneously ensuring the variance of the estimator is kept under control for a suitably large N. This enables the resulting denoiser to generalize well to images not used during training.

To do this, we observe that there are three issues inherent to the supervised learning framework that must be considered to develop a better MC denoising system:

- **(i)** The function, g, must be flexible enough to capture the complex relationship between input data and reference colors for a wide range of scenarios. In the following section, we describe how we model g using *deep convolutional networks*.
- **(ii)** The choice of loss function, ℓ, is critical. Ideally, the loss must capture perceptually important differences between the estimated and reference color. However, it must also be easy to evaluate and optimize. We use the absolute value loss function, ℓ_1, (Sec. 5) and explore its benefits in Sec. 7.
- **(iii)** In order for our model to be deep yet avoid overfitting, we require a large training dataset, $\mathcal{D}$. Since we require reference images rendered at high sample counts, obtaining a large data set is extremely computationally expensive. Furthermore, in order to generalize well, the network needs examples that are representative of the various effects to be denoised. We describe our data in Sec. 5.

4 DEEP CONVOLUTIONAL DENOISING

In this section, we describe our approach to model the denoising function g in Eq. (3) with a deep convolutional neural network (CNN). Since each layer of a CNN applies multiple spatial kernels with learnable weights that are shared over the entire image space, they are naturally suited for the denoising task and have indeed been previously used for traditional image denoising [Xie et al. 2012]. Furthermore, by joining many such layers together with activation functions, CNNs are able to learn highly nonlinear functions of the input features, which are important for obtaining high-quality outputs. Fig. 2 illustrates our entire denoising pipeline. We first focus on the filtering core of the denoiser—the network architecture and the reconstruction filter—and later describe data decomposition and preprocessing that are specific to the problem of MC denoising.

4.1 Network Architecture

We use deep fully convolutional networks with no fully-connected layers to keep the number of parameters reasonably low. This reduces the danger of overfitting and speeds up both training and inference. Stacking many convolutional layers together effectively increases the size of the input receptive field to capture more context and long-range dependencies [Simonyan and Zisserman 2014].

In each layer l, the network applies a linear convolution to the output of the previous layer, adds a constant bias, and then applies an element-wise nonlinear transformation $f^l(\cdot)$, also known as the *activation function*, to produce output $\mathbf{z}^l = f^l\left(\mathbf{W}^l * \mathbf{z}^{l-1} + \mathbf{b}^l\right)$. Here, $\mathbf{W}^l$ and $\mathbf{b}^l$ are tensors of weights and biases (the weights in $\mathbf{W}$ are shared appropriately to represent linear convolution kernels), and $\mathbf{z}^{l-1}$ is the output of the previous layer. For the first layer, we set $\mathbf{z}^0 = \mathbf{X}_p$, which provides the block of per-pixel vectors around pixel p as input to our CNN.

For all layers, we use rectified linear unit (ReLU) activations, $f^l(a) = \max(0, a)$, except for the last layer, L, where $f^L(a) = a$

(i.e., the identity function). Despite their C_1 discontinuity, ReLUs have been shown to achieve state-of-the-art performance in many tasks and are known to encourage the (non-convex) optimization procedure to find better local minima [Balduzzi et al. 2016].

The weights and biases $\boldsymbol{\theta} = \{(\mathbf{W}^1, \mathbf{b}^1), \ldots, (\mathbf{W}^L, \mathbf{b}^L)\}$, represent the trainable parameters of g for our L-layer CNN. The dimensions of the weights in each layer, which are fixed before training, are described in Sec. 5.2.

4.2 Reconstruction Methods

In our system, the function g outputs denoised color values using one of two possible architectures: a *direct-prediction* convolutional network (DPCN) or a novel *kernel-prediction* convolutional network (KPCN). We now describe each one in turn.

Direct Prediction Convolutional Network (DPCN). Producing the denoised image using *direct prediction* is straightforward. We simply choose the size of the final layer of the network to ensure that for each pixel, p, the corresponding element of the network output, $\mathbf{z}_p^L \in \mathbb{R}^3$ is the denoised color:

$$\widehat{\mathbf{c}}_p = g_{\text{direct}}(\mathbf{X}_p; \boldsymbol{\theta}) = \mathbf{z}_p^L.$$

Direct prediction achieves good results. However, we found that the unconstrained nature and complexity of the problem makes optimization difficult. The magnitude and variance of the stochastic gradients computed during training can be large, which slows convergence. For example, in order to obtain good performance, the DPCN architecture required over a week of training.

Kernel Prediction Convolutional Network (KPCN). Instead of directly outputting a denoised pixel, $\widehat{\mathbf{c}}_p$, the final layer of the network outputs a *kernel* of scalar weights that is applied to the noisy neighborhood of p to produce $\widehat{\mathbf{c}}_p$. Letting $\mathcal{N}(p)$ be the $k \times k$ neighborhood centered around pixel p, the dimensions of the final layer are chosen so that the output is $\mathbf{z}_p^L \in \mathbb{R}^{k \times k}$. Note that the kernel size k is specified before training along with the other network hyperparameters (e.g., layer size, CNN kernel size, and so on) and the same weights are applied to each RGB color channel.

Defining $[\mathbf{z}_p^L]_q$ as the q-th entry in the vector obtained by flattening $\mathbf{z}_p^L$, we compute the final, normalized kernel weights as

$$w_{pq} = \frac{\exp([\mathbf{z}_p^L]_q)}{\sum_{q' \in \mathcal{N}(p)} \exp([\mathbf{z}_p^L]_{q'})},$$

and the denoised pixel color as

$$\widehat{\mathbf{c}}_p = g_{\text{weighted}}(\mathbf{X}_p; \boldsymbol{\theta}) = \sum_{q \in \mathcal{N}(p)} \mathbf{c}_q w_{pq}.$$

The kernel weights can be interpreted as including a softmax activation function on the network outputs in the final layer over the entire neighborhood. This enforces that $0 \le w_{pq} \le 1,\ \forall q \in \mathcal{N}(p)$ and $\sum_{q \in \mathcal{N}(p)} w_{pq} = 1$. Doing this has three specific benefits:

(i) It ensures that the final color estimate always lies within the convex hull of the respective neighborhood of the input image. This vastly reduces the search space of output values as compared to the direct-prediction method and avoids potential artifacts (e.g., color shifts).

(ii) It ensures the gradients of the error with respect to the kernel weights are well behaved, which prevents large oscillatory changes to the network parameters caused by the high dynamic range of the input. Intuitively, the weights need only encode the relative importance of the neighborhood; the network does not need to learn the absolute scale. In general, scale-reparameterization schemes have recently proven to be crucial for obtaining low-variance gradients and speeding up convergence [Salimans and Kingma 2016].

(iii) It could potentially be used for denoising across layers of a given frame, a common case in production, by applying the same reconstruction weights to each component.

We analyze the behavior of both of our proposed architectures in Sec. 7, observing that both converge to a similar overall error, but at different speeds. For example, with our training data, the weighted kernel prediction converges roughly 5-6× faster than the direct reconstruction. Due to its faster convergence, we use the KPCN architecture for all results and analysis, unless otherwise noted.

4.3 Diffuse/Specular Decomposition

Denoising the color output of a MC renderer in a single filtering operation may be prone to overblurring (see Sec. 7). This is because the various components of the image have different noise characteristics and spatial structure, which often leads to conflicting denoising constraints. We mitigate this issue by decomposing the image into diffuse and specular components as in Zimmer et al. [2015]. These components are then independently preprocessed, filtered, and postprocessed, before recombining them to obtain the final image, as illustrated in Figure 2.

Diffuse-component Preprocessing. The diffuse color—the outgoing radiance due to diffuse reflection—is well behaved and typically has small ranges. Thus, training the diffuse CNN is stable and the resulting network yields good performance without color preprocessing. However, in practice, we factor out the noisy albedo produced by the renderer in the preprocessing step, to have the CNN use the effective irradiance [Zimmer et al. 2015], $\widetilde{\mathbf{c}}_{\text{diffuse}} = \mathbf{c}_{\text{diffuse}} \oslash (\mathbf{f}_{\text{albedo}} + \epsilon)$, where $\oslash$ is an element-wise (Hadamard) division and $\epsilon = 0.00316$ in our implementation. This allows for larger filtering kernels, since the irradiance buffer is smoother. Our postprocessing step inverts this procedure (i.e., multiplies back the albedo), thereby restoring all texture detail.

Specular-component Preprocessing. Denoising the specular color is a challenging problem due to the high dynamic range of specular and glossy reflections; the values in one image can span several orders of magnitude. The large variations and arbitrary correlations in the input make the iterative optimization process highly unstable. We thus apply a log transform to each color channel of the input image yielding $\widetilde{\mathbf{c}}_{\text{specular}} = \log(1 + \mathbf{c}_{\text{specular}})$, which significantly reduces the range of color values. This transformation greatly improves results and avoids artifacts in regions with high dynamic range (see Sec. 7).

After the two components have been denoised separately, we apply the inverse of the preprocessing transform to the reconstructed output of each network and compute the final denoised image,

$$\widehat{\mathbf{c}} = (\mathbf{f}_{\text{albedo}} + \epsilon) \odot \widehat{\mathbf{c}}_{\text{diffuse}} + \exp(\widehat{\mathbf{c}}_{\text{specular}}) - 1, \tag{4}$$

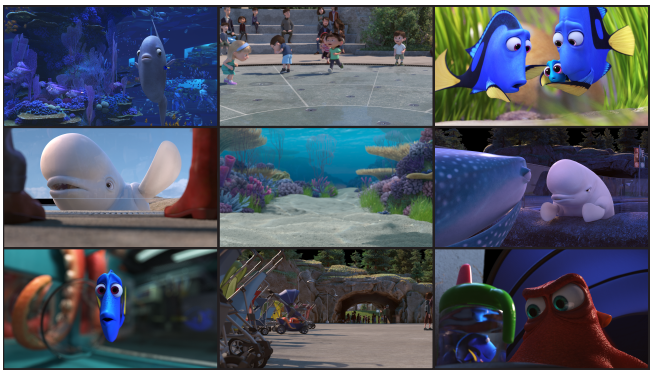


Fig. 3. Example reference images from our 600 frame training set, sampled from the full *Finding Dory* film.

where $\odot$ is an element-wise (Hadamard) product. To train our system, we pre-train the specular and diffuse networks separately on the specular and diffuse references, respectively. Afterwards, we apply Eq. 4 and fine-tune the complete framework by minimizing the error of the final image for additional iterations. This allows us to recover missing detail and obtain sharper results.

5 EXPERIMENTAL SETUP

5.1 Data

Training a deep neural network requires obtaining a large and representative dataset in order to learn the complex relationship between input and output while avoiding overfitting. For our training set, we used 600 representative frames sampled from the entire movie *Finding Dory* generated using RenderMan's path-tracer (Fig. 3).

Meanwhile, our test set consists of 25 diverse frames from the films *Cars 3* and *Coco*, and contain effects such as motion blur, depth of field, glossy reflections, and global illumination. These films significantly differ in style and content, testing how our system generalizes to new inputs. For example, the test set includes mostly outdoor scenes with a wide-range of color palettes that are very different from *Finding Dory*.

The reference images for training were rendered with 1024 samples per pixel (spp). Although we considered removing the residual noise from these using standard MC denoisers, we found that the CNNs performed better when trained on images with *uncorrelated* residual noise rather than correlated errors and artifacts introduced by the additional denoising step. Therefore, we used reference images that, although still contained a small amount of visible noise, were converged enough to properly train with.

To evaluate our proposed approach, we trained, validated, and tested on inputs rendered at a fixed 128 spp (for production-level quality) and 32 spp (for pre-visualization). For each scene, the renderer outputs the diffuse and specular RGB color buffers, $\mathbf{c}_{\text{diffuse}}$ and $\mathbf{c}_{\text{specular}}$, the corresponding per-pixel, color variances, $\sigma^2_{\text{diffuse}}$ and $\sigma^2_{\text{specular}}$, the feature buffers, $\mathbf{f}$, consisting of surface normals (3 channels), albedo (3 channels), depth (1 channel), and the corresponding per-pixel feature variances, ${\sigma_{\mathbf{f}}}^2$. In our implementation, we convert variances of three channels to a single channel by computing its luminance. Thus, we have 2 channels for the color variance (for diffuse and specular) and 3 channels for the feature variance.

As is commonly done in machine learning, we process some of the raw data to provide the network with more useful features that facilitate learning and convergence. First, since the depth values can have arbitrary ranges, we linearly scale it to the range $[0, 1]$ for each frame. We also preprocess the color buffers as described previously in Sec. 4.3 to get $\widetilde{\mathbf{c}}_{\text{diffuse}}$ and $\widetilde{\mathbf{c}}_{\text{specular}}$. Finally, we take the gradients in both x and y directions, G_x and G_y, for all buffers, as we found these highlight important details that facilitate training.

Since we preprocess the color buffers, we must apply an appropriate transformation to their variances to make them valid. In general, if we apply a transformation, h, to a random variable, X, we can approximate the corresponding transformation on its second moment using a Taylor series expansion: $\sigma_{h(X)} \approx (h'(\mu_X))^2\sigma^2_X$, where μ_X and σ^2_X are the mean and variance of X, respectively, and h' is the derivative with respect to X. Thus, for the diffuse and specular components, the modified variance is given by:

$$\begin{aligned}(\widetilde{\sigma}_{\text{diffuse}})^2 &\approx \sigma^2_{\text{diffuse}} \oslash (\mathbf{f}_{\text{albedo}} + \epsilon)^2,\\ (\widetilde{\sigma}_{\text{specular}})^2 &\approx \sigma^2_{\text{specular}} \oslash (\widetilde{\mathbf{c}}_{\text{specular}})^2.\end{aligned} \tag{5}$$

After this processing, we construct our network input as:

$$\mathbf{x} = \{\widetilde{\mathbf{c}}, G_x(\{\widetilde{\mathbf{c}}, \mathbf{f}\}), G_y(\{\widetilde{\mathbf{c}}, \mathbf{f}\}), \widetilde{\sigma}^2, {\sigma_{\mathbf{f}}}^2\},$$

where $\widetilde{\mathbf{c}}$ and $\widetilde{\sigma}^2$ are either diffuse or specular.

After processing the data at each pixel, we split the images into 65×65 patches that are sampled, shuffled, and used to train the network. Although uniform sampling could be used to select the patches from each frame, we found that this was suboptimal as the network would be frequently shown simple cases containing smooth regions that are straightforward to denoise. Instead, we wanted the network to be exposed to and learn how to handle difficult cases.

To do this, we use the following sampling strategy, inspired by Gharbi et al. [2016], to get 400 patches for each 1920×1080 frame. We start with dart throwing to find candidate patches, which we then prune using a PDF based on the variance of the noisy color buffer and the shading normals. Using the color ensures that we target regions that have lots of noise, detail, or texture, while using the normal buffer provides examples with geometric complexity. Finally, to ensure that we provide a proper balance between the easy and hard cases and avoid biasing the network, we automatically accept a patch after it has been rejected a certain number of times.

5.2 Training

We use eight hidden layers (i.e., nine total convolutions, so $L = 9$) with 100 kernels of 5×5 in each layer for each network. For KPCN, we used an output kernel with size $k = 21$. Weights for both the 128 and 32 spp networks were initialized using the Xavier method [Glorot and Bengio 2010]. Specifically, we generate random values from a uniform distribution with a variance determined by the number of nodes between layers.

The specular and diffuse networks are trained independently using the ℓ_1 (absolute value) error metric. We observed that this loss function offered the best perceptual quality while still being fast to compute and optimize (see Sec. 7 for additional justification). The loss for the diffuse network is computed between the reconstructed irradiance (i.e., before multiplying with the albedo) and the

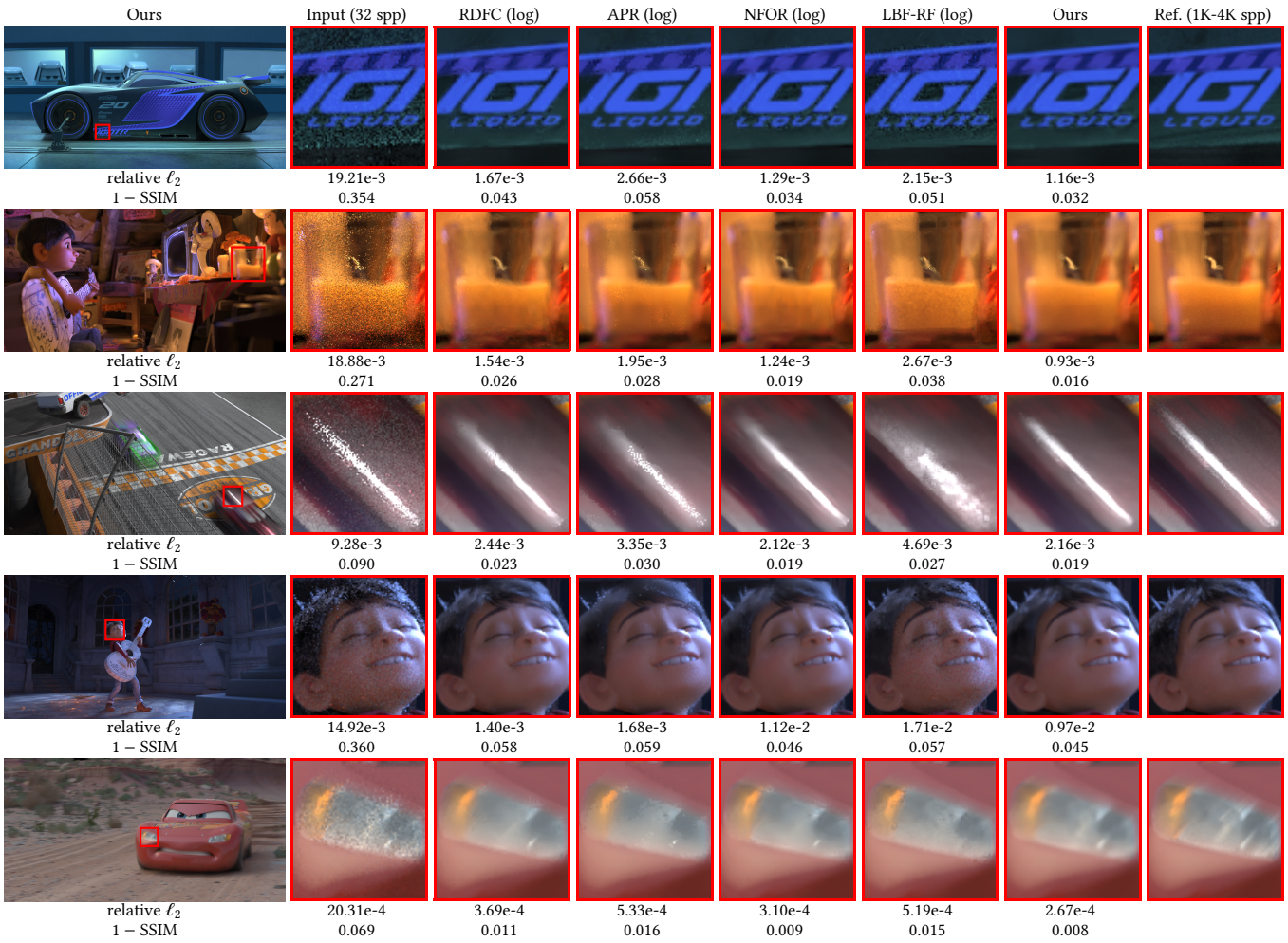


Fig. 4. We demonstrate favorable results relative to state-of-the-art denoisers on 32 spp production-quality data, often removing more noise while still keeping detail and better preserving highlights. Please see the supplemental material for comparisons with 128 spp data typically used in the final stages of production. Note that the LBF results shown are run with modifications that can cause suboptimal performance (see text).

albedo-factorized reference image. The loss for the specular CNNs is computed in the log domain.

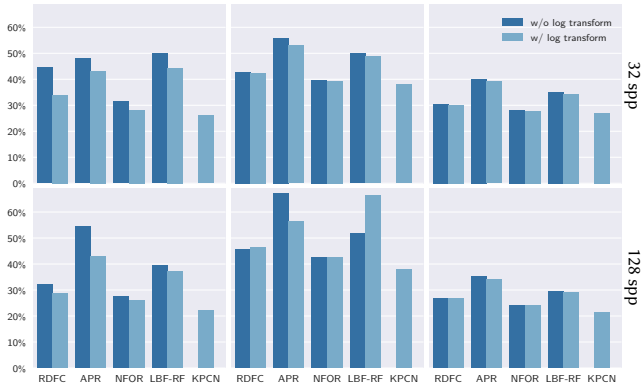
The networks were optimized using the ADAM [Kingma and Ba 2014] optimizer in TensorFlow [Abadi et al. 2015] with a learning rate of 10^{-5} and mini-batches of size 5. Each network is pre-trained for approximately 750K iterations over the course of ~1.5 days on an Nvidia Quadro M6000 GPU. Afterwards, the system is combined and fine-tuned (Sec. 4.3) for another ~0.5 days or 250K iterations.

6 RESULTS

To evaluate our method, we compare our results to a range of state-of-the-art methods: RDFC [Rousselle et al. 2013], APR [Moon et al. 2016], NFOR [Bitterli et al. 2016], and LBF [Kalantari et al. 2015]. In the supplemental, we also compare against the RenderMan denoiser, which was used during the production of the films in the training/test sets. We use four metrics to evaluate the results: ℓ_1, relative ℓ_1, relative ℓ_2 [Rousselle et al. 2011], and Structural Similarity Index (SSIM) [Wang et al. 2004] (see supplemental for a description of how these are computed). For conciseness, we report only relative ℓ_2 and SSIM in the paper, as they are the most commonly used. See our supplemental material for full resolution results at 16, 32, and 128 samples per pixel (spp), all metrics with heat maps, and a web-based interactive viewer that allows for inspection of the results.[1]

All denoisers are given the same inputs: the color buffer and the albedo, normal, and depth buffers corresponding to the first ray intersection. Note that we save the feature buffers at the first diffuse intersection in order to handle specular regions with little useful information (e.g., glass). Previous methods gave better results when run with some of our preprocessing steps, so we report them like this in the paper. In particular, we applied all methods on top of our diffuse/specular decomposition, including the albedo divide for the diffuse component and the log transform of the specular component. Interestingly, the log transform often significantly increased the robustness of these denoisers and resulted in much fewer halo

[1] Supplemental materials can be found here: https://doi.org/10.7919/F4057CVT.



(a) relative ℓ_2 (b) ℓ_1 (c) 1−SSIM

Fig. 5. Average performance of RDFC, APR, NFOR, LBF-RF, and our KPCN across test scenes for 32 spp (top) and 128 spp (bottom) inputs. The values are relative to the noisy input and expressed as percentages (%); lower is better. The dark-colored bars show the performance of prior art with decomposition, irradiance factorization, but *without* log-transforming the specular component. The light-colored bars show performance *with* the log transform. For increased robustness, the relative ℓ_2 error was computed as a trimmed mean, removing 0.01% of the best and the worst pixels per image.

artifacts (see our supplemental material for results using the raw specular component).

For all the denoisers, we multiply in the albedo buffer extracted from a separate, higher sampling rate pass to obtain the final image. In practice, this noise-free albedo could be generated from either a fast high-sample count render that ignores illumination calculations or alternatively from a separate denoising process (e.g., prefiltering). Furthermore, for all methods, we currently ignore the alpha channel during the filtering process, so to generate the final image, we simply use the original alpha and zero out the appropriate regions to avoid color bleeding. Finally, for the production data we used, RenderMan has been configured to send out 8 shadow rays at the first bounce of each sample to get a better estimate of the direct illumination.

Our noisy renderings use correlated samples because of low discrepancy sampling, so we cannot directly estimate an accurate variance of the per-pixel sample mean. Instead, we instrumented RenderMan to output the two-buffer variance used in previous works [Rousselle et al. 2012] to properly evaluate RDFC, NFOR, and APR on our *test* data. Note that the training/test data for our system has the raw sample variances directly from the renderer, rather than the two-buffer variances used in the aforementioned methods.

All methods used the default settings suggested by the authors, except for LBF, where we trained the network on our own data using a joint non-local means filter back-end and the MLP architecture described in the original paper. Since our *training* dataset does not have the two-buffer variance expected by LBF, their system cannot pre-filter the features. Thus, for fairer comparisons, we substitute the pre-filtered features with the relatively noise-free ones of the reference image and denote it as LBF-RF (for reference features).

However, there are still some distinct differences from the original implementation that cause LBF to run suboptimally. First, our dataset does not provide some of the primary features expected by LBF, namely the secondary albedo and direct visibility, which are useful guiding features for the filter. To compensate for this missing data, we instead replace the LBF secondary features corresponding to these two primary features with features calculated from the noisy color buffers. However, as observed in their paper, using such buffers leads to overfitting and residual noise. These issues are further exacerbated by substituting the noisy sample mean variance into the joint non-local means filter instead of the filtered two-buffer variance expected by LBF. As a result, the LBF results shown here tend to leave excessive residual noise.

As described in Sec. 5, we trained our CNN on 600 frames from the film *Finding Dory*, all rendered at a uniform sampling rate of 32 and 128 spp with references at 1024 spp. We trained two networks, one for each sampling rate, and applied them to the test data with the corresponding sampling rate. In Fig. 4, we show a subset of results from our test set containing 25 frames from the films *Cars 3* and *Coco* on 32 spp data (see supplemental for all results at both sampling rates).

Overall, we perform as well or better than state-of-the-art techniques both perceptually and quantitatively. For example, rows 1, 4, and 5 of Fig. 4 show how previous methods have residual noise in the car decals, child's face, and car headlight, respectively, while our approach removes the noise and still preserves detail. Furthermore, our approach generates a smooth result on the glass of row 2 and keeps the energy of the strong specular highlight in row 3. Meanwhile, the other approaches tend to introduce filter artifacts and lose energy in bright regions.

Figure 5 shows a comparison of the average performance of each method across all test scenes with respect to each error metric for both 32 and 128 spp. We observe that our network consistently improves over state of the art across all error metrics shown. In Fig. 6, we demonstrate the flexibility of our method by processing inputs at 16 spp with our network trained on 32 spp data. As shown, despite being trained on a higher sampling rate, our network is able to successfully extrapolate to this data while still improving on the state-of-the-art methods. In particular, the previous approaches tend to leave excessive residual noise relative to our approach along the edges of the cables.

To facilitate future comparisons and demonstrate our network's ability to perform well on noisier data from a different rendering system, we provide results in Fig. 7 on publicly available Tungsten scenes [Bitterli 2016] and compare our approach to a baseline method, NFOR [Bitterli et al. 2016]. In particular, the results show slight residual noise in the NFOR result even at 128 spp, while our approach more closely resembles the reference. A similar figure in concurrent work [Chaitanya et al. 2017] allows readers to see the relative improvements over the baseline, facilitating comparisons of these two systems.

Note that to produce these results, we trained our system on a set of Tungsten training scenes (see Sec. 7 for results with our original training). Specifically, we took 8 Tungsten scenes not in our test set and randomly modified them in various ways, including swapping materials, camera parameters, and environment maps to generate 1484 unique training scenes. Please see the supplemental for a list of the original Tungsten scenes used to generate the training set.

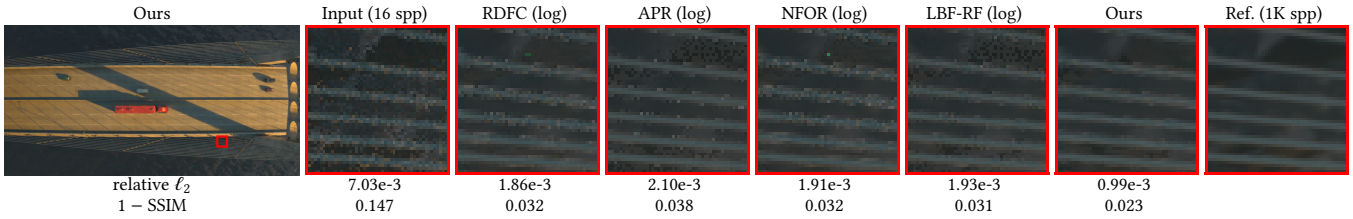


Fig. 6. Our network trained on 32 spp data and tested on 16 spp data still performs well relative to other approaches. This demonstrates that our technique can successfully extrapolate to other sampling rates. See supplemental for additional results at 16 spp.

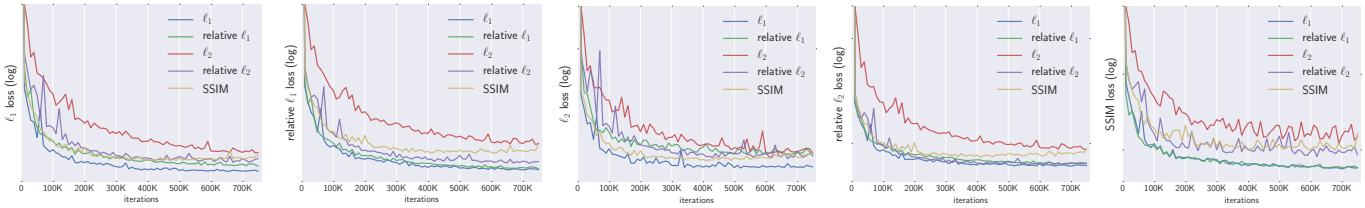
Fig. 7. We retrained our network on data rendered with the Tungsten path tracer and compared with a baseline approach (NFOR) on scenes from Bitterli et al. [2016] using the publicly available lighting and camera parameters. See the concurrent work of Chaitanya et al. [2017] for a similar figure.

In terms of timing, for an HD image of 1920×1080, our network takes about 12 seconds to evaluate and output a full denoised image. For comparison, the timings for the other GPU-based approaches are approximately 10 seconds for RDFC, 10-20 seconds for APR, and 20 seconds for LBF. The CPU version of NFOR takes 4-6 minutes. It is worth noting that these images take about 100 core hours to render at 128 spp, so no additional samples can be rendered in the time it takes to evaluate any of the denoisers.

7 ANALYSIS

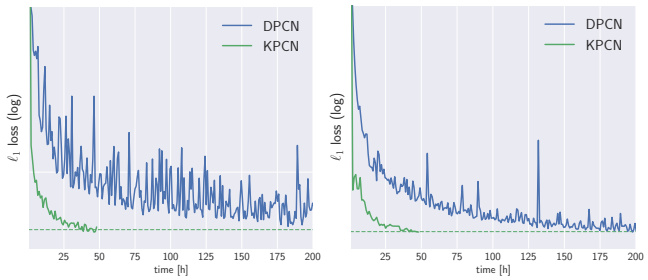
In this section, we analyze the various design choices made in our network architecture using hold-out frames from *Finding Dory* and test frames from *Cars 3*. We begin by examining the choice of loss function, a crucial aspect of our design as it determines what the network deems important. For MC denoising, we ideally want a loss function that reflects the perceptual quality of the image relative to the reference. To evaluate the behavior of various error metrics, we optimize the network with each and evaluate their performance on held-out training data from *Finding Dory* and validation data from *Cars 3*. We evaluate five common metrics: ℓ_1, relative ℓ_1, ℓ_2, relative ℓ_2, and SSIM, when optimizing for each in turn. Fig. 8 shows that the network trained with the ℓ_1 metric consistently has the lowest error across all five metrics for both datasets. Due to this robustness, we chose the ℓ_1 error metric for our system.

It is interesting to note that sometimes the network optimized on a given error is not always the best performing one. For example, the



(a) ℓ_1 (b) relative ℓ_1 (c) ℓ_2 (d) relative ℓ_2 (e) SSIM

Fig. 8. Here we show convergence plots of networks optimized with common error metrics evaluated on hold-out data from *Finding Dory*. For example, (a) shows the ℓ_1 error of the dataset using networks trained on ℓ_1, relative ℓ_1, ℓ_2, relative ℓ_2, and SSIM. The network trained with ℓ_1 consistently has the best performance across all the error metrics tested. This behavior carries over to our validation set of *Cars 3* images (see supplemental materials).



(a) Diffuse (b) Specular

Fig. 9. Comparison of optimization speed between the DPCN and KPCN architectures. Although both approaches converge to a similar error on the *Cars 3* validation set, the KPCN system converges 5–6× faster.

network trained on ℓ_1 error performs better on ℓ_2 than the network optimized on ℓ_2. One possible reason for this is that ℓ_2 is sensitive to outliers, such as fireflies, or extremely bright specular highlights that significantly contribute to the error. Trying to compensate for these regions will sacrifice performance elsewhere, while networks trained on different losses are more robust to outliers.

Figure 9 compares the validation loss between the DPCN and KPCN reconstruction schemes as a function of hours trained for both the specular and diffuse networks. We stop training the KPCN after 50 hours and show the average loss during the last 10% of training with the horizontal, dashed line. We observed that the convergence of the DPCN is slower with considerably higher variance, on average requiring 5-6× longer to reach the same loss value. Therefore, by imposing reasonable constraints on the network output, we can greatly speed up training without sacrificing average performance.

Since there has been previous work in using machine learning for natural image denoising, we evaluated the performance of naïvely applying a CNN to the problem of MC denoising. Specifically, we train on the raw color buffer (without decomposition or the albedo divide) and directly output the denoised color. [2] As shown in Fig. 10, such a network produces overblurred results since it has no features/information to allow it to distinguish between scene noise and detail. Furthermore, since the input and output have high dynamic range, it cannot properly handle bright regions and causes ringing and color artifacts around highlights. Moreover, working in the HDR domain causes instability in the network weights making it difficult to train properly.

Next, we evaluate the effect of the various additions to our framework that alleviate the aforementioned issues of a vanilla CNN. First, we explored the effect of including extra features as input. One significant advantage over deep networks used in the denoising of photographs is that we can utilize additional information output by the rendering system including shading normals, depth, and albedo. Thus, we trained our architecture with and without our additional features (Sec. 5). The network trained only on the color buffer cannot differentiate between scene detail and noise, so it overblurs compared to our full approach (see Fig. 11).

We found that training with high dynamic range data introduced many issues. Namely, the wide range of values for both the inputs and outputs created instability in the weights and made training difficult. Fig. 12 shows how using the log transform of the color buffer and its corresponding transformed variance (Eq. 5) reduces artifacts in bright regions. Interestingly, we found that working in the log domain had benefits for previous denoising techniques as well, reducing halos and ringing issues (see the supplemental for results of previous approaches with and without the log transform).

Both the diffuse/specular decomposition and albedo factorization also improve our method significantly. The decomposition allows the networks to separately handle the fundamentally different diffuse and specular noise. Furthermore, by dividing out the albedo from the diffuse illumination and thereby denoising the effective irradiance, we can preserve texture details more easily. We retrained our system without the albedo divide and observed overblurring. For example, Fig. 13 shows how the decals on the car become overblurred and illegible without the albedo divide. Moreover, if we perform the albedo divide without the decomposition, the network preserves detail but has clear artifacts in specular regions. In this experiment, we still perform the log transform to handle the high dynamic range.

Figure 14 further demonstrates the ability of our network to generalize to new scenes with different artistic styles than are present in our training set. This is a frame from the photorealistic short film *Piper* denoised by our network without additional training or modification (i.e., trained only on *Finding Dory*). This suggests that the network is not overfitting to a specific style, film, or noise pattern and instead learns a robust relationship between input and output enabling good performance on a wide variety of data.

There are various inherent limitations of our learning-based approach, however. First, our results can lose scene detail that is not properly captured by our input features and that is not present in our training set. For example, in the top row of Fig. 15, we show how the lines on the jumbo screen are removed because they are not in the auxiliary features and the network mistakes them for

[2] We use the same hyperparameters as reported for our final architecture: 8 hidden layers of 5×5×100.

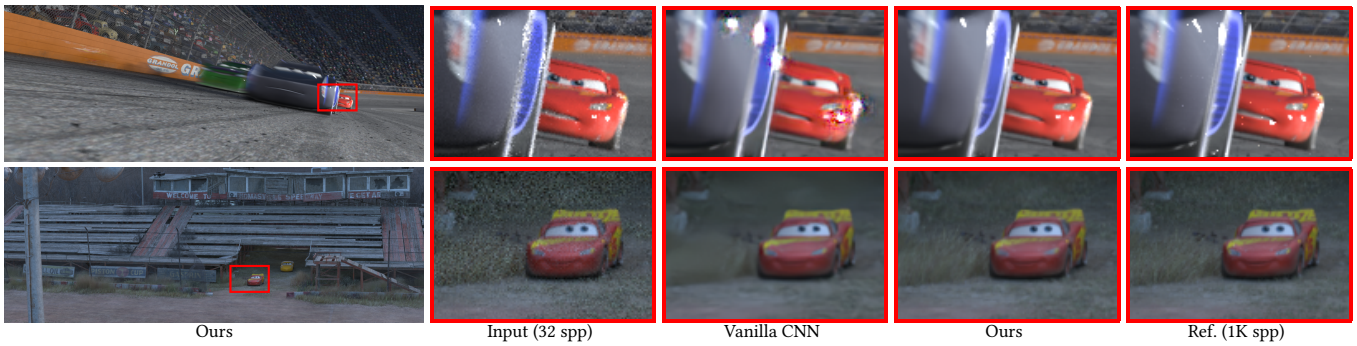


Fig. 10. We naively apply a CNN for MC denoising using only the unprocessed color buffer as input and directly outputting the denoised image. The high dynamic range data creates color artifacts around highlights (top row), while the missing additional features results in overblurring of detail (bottom row).

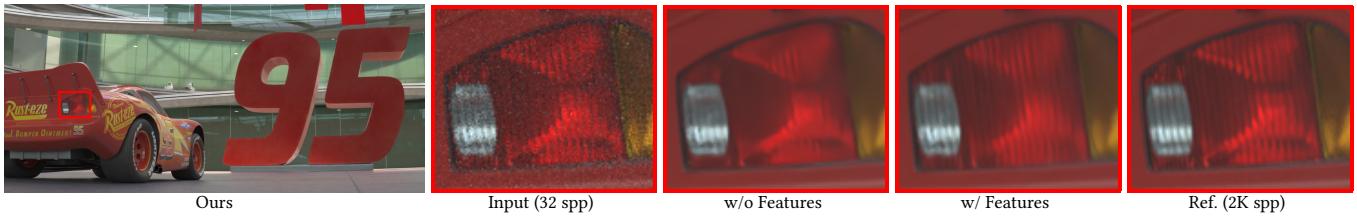


Fig. 11. When training using only the diffuse/specular color buffers without additional features, the network overblurs detail.

Fig. 12. When we train with high dynamic range images, we observe artifacts in regions with large-valued specular highlights. Our full approach with the log and corresponding transformed variance handles these difficult cases better.

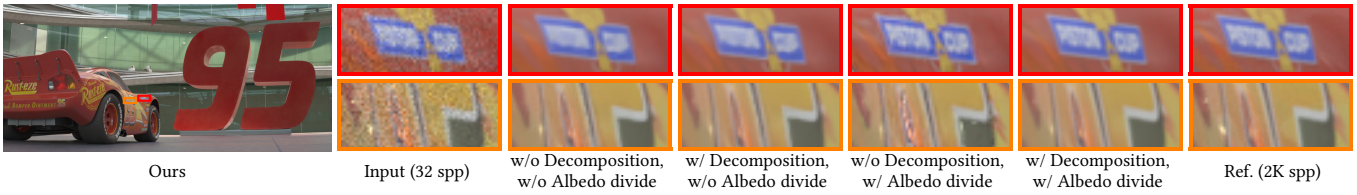


Fig. 13. Retraining our network without the diffuse/specular decomposition or albedo factorization results in overblurred textures, such as these illegible car decals. Using the decomposition without the albedo divide continues to overblur (top row). On the other hand, doing the albedo divide without the decomposition creates artifacts in specular regions (bottom row). Our full approach preserves the text clearly and closely resembles the reference.

scene noise. Also, since such patches were not present in the training dataset, the network cannot resolve them using only the color buffer. However, this could be potentially alleviated by additional training on similar examples. Likewise, examples of all distributed effects from the test set should be shown during training, otherwise the network cannot properly denoise them. For example, volumetric effects with lots of fine detail, such as fire or smoke, that were not in the training set are typically overblurred by our system (second row of Fig. 15).

Another limitation occurs when applying our method to a different rendering system than the one it was trained on. The third row of Fig. 15 shows the results of using the network trained with *Finding Dory* data from RenderMan on test data from the Tungsten renderer. Although both renderers output the same features, there are inevitable differences (e.g., dynamic range and noise levels) that can cause artifacts. These issues largely disappear when training on the Tungsten data, although our approach still generates artifacts when the input has severe noise, such as with the 32 spp scene shown in the last row of Fig. 15.

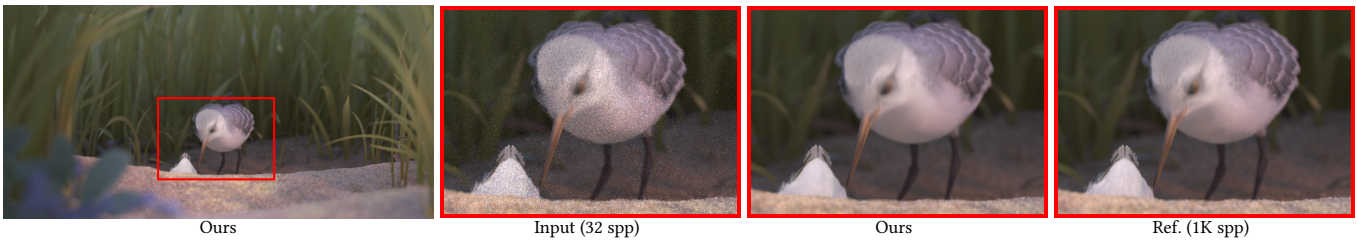


Fig. 14. We demonstrate how our network is able to denoise a photorealistic frame from the short film *Piper*, which significantly differs from the training data, *Finding Dory*. Note that even at low sampling rates, our network generalizes well and produces high-quality denoised results.

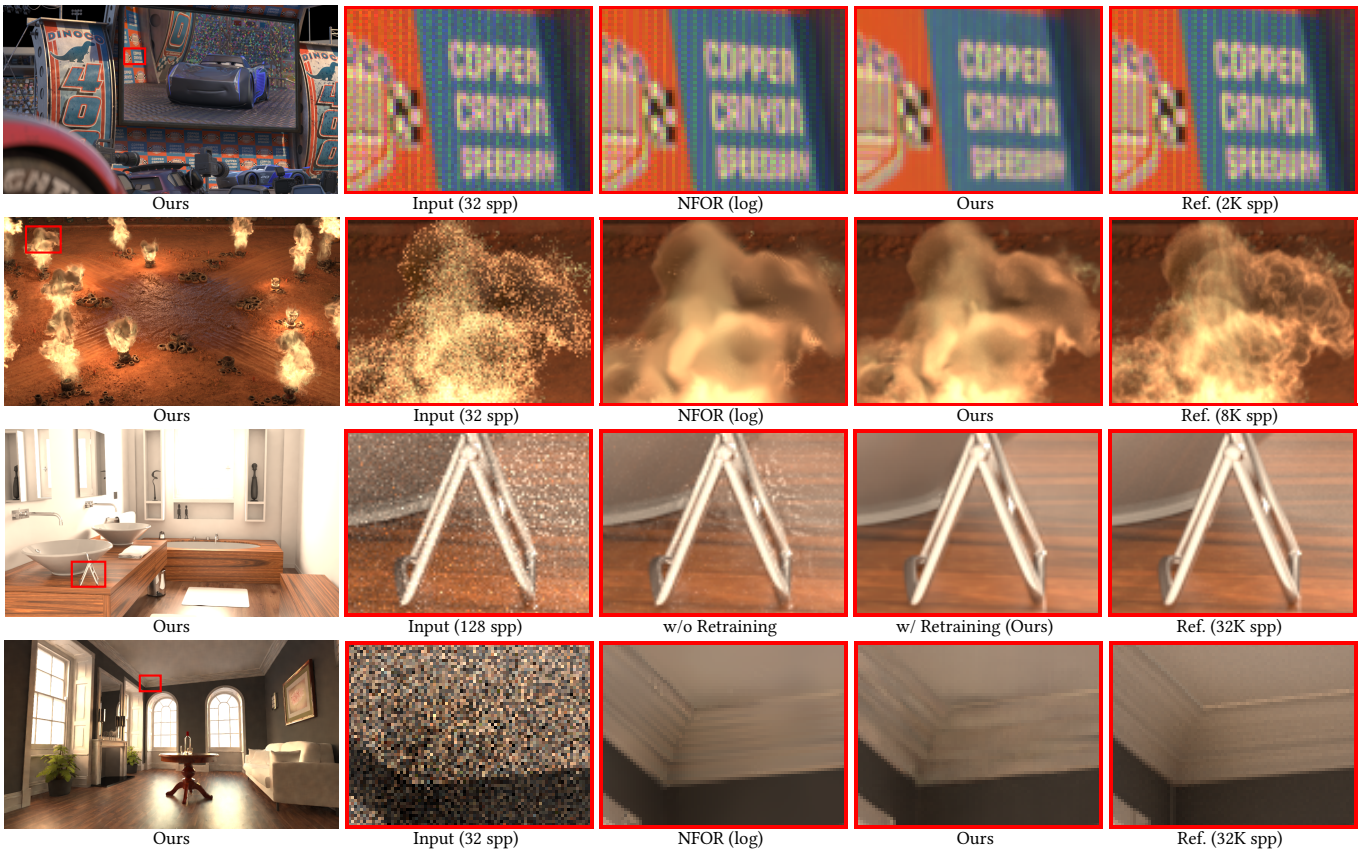


Fig. 15. We demonstrate various limitations of our approach. When the input features fail to capture important scene detail, the network will mistake it for noise and try to remove it (top row). Examples of fire were not used in training, so our method tends to overblur these cases (second row). Applying a network trained on data from a different rendering system will cause artifacts due to inherent differences in noise levels, ranges, and sampling strategies. The results are significantly improved if the network is instead trained on data from the new rendering system (third row). However, even when trained on this data, the network struggles with extremely noisy inputs (bottom row).

8 FUTURE WORK AND CONCLUSIONS

Although we have demonstrated a robust, learning-based MC denoising algorithm in this paper, there are many design decisions that could be explored more extensively to further improve performance. To facilitate this exploration and enable others to run our system on publicly available Tungsten scenes, we will release the code and trained weights to the community.

The first potential topic to investigate is the choice of error metric. Often, perceptually important features are not captured by any of the standard loss metrics which also behave quite differently from each other. We see notable examples of this in Sec. 6 and Sec. 7. This poses an especially important problem during training. A more thorough investigation of perceptual loss functions is required, which would improve both network training *and* lead to a more principled perceptual evaluation of results.

Furthermore, we presented a simple sampling approach for selecting important patches from each image used in training. Although this helped performance, our approach is far from optimal. One can imagine using other features and metrics to better sample patches

and allow the network to converge faster or even learn more complicated relationships.

Our network's hyperparameters are also not optimal. We explored various layer numbers/sizes and kernel sizes to find settings that work well, but a more thorough search through the parameter space could reveal better ones. Different architectures and concepts might also yield improved performance. We explored the use of recurrent and residual connections [Yang et al. 2016; He et al. 2016], but found little benefit. However, these could be potentially useful tools to explore much deeper networks that improve performance yet keep the number of model parameters tractable. Moreover, *generative* models, such as variational autoencoders [Kingma and Welling 2013], and generative adversarial networks have shown great promise for natural image super-resolution and denoising [Ledig et al. 2016]. Although scaling to high-resolution images presents a large computational hurdle for these methods, it would be an interesting avenue for future research.

Finally, we demonstrated results for denoising only a single image at a time, but it would be useful to handle animated sequences as well. This extension is non-trivial and involves further exploration of the architecture and design to be able to preserve temporal coherency across neighboring denoised frames. For example, the concurrent work of Chakravarty et al. [2017] focuses on denoising sequences at interactive rates.

In summary, we have presented the first successful step towards practically using deep convolutional networks for denoising Monte Carlo rendered images in production environments. Specifically, we demonstrated that a deep learning approach can recognize the fundamental, underlying relationship between the noisy and reference data without overfitting, all while still being able to withstand the strict production demands on quality. Although it uses a relatively straightforward architecture, our solution is fast, robust, stable to train/evaluate, and it performs favorably with respect to state-of-the-art denoising algorithms.

9 ACKNOWLEDGMENTS

We gratefully thank John Halstead for generating the *Finding Dory* training data and Andreas Krause for helpful discussions. We also thank the following Blendswap artists for creating the scenes in both Fig. 7 and the training set: Jay-Artist, Mareck, MrChimp2313, nacimus, NovaZeeke, SlykDrako, thecali, and Wig42. This work was partially funded by National Science Foundation grants #13-21168 and #16-19376.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, , and others. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

David Balduzzi, Brian McWilliams, and Tony Butler-Yeoman. 2016. Neural Taylor Approximations: Convergence and Exploration in Rectifier Networks. *arXiv preprint arXiv:1611.02345* (2016).

Pablo Bauszat, Martin Eisemann, and Marcus Magnor. 2011. Guided Image Filtering for Interactive High-quality Global Illumination. *Computer Graphics Forum* 30, 4 (2011), 1361–1368.

Benedikt Bitterli. 2016. Rendering Resources. (2016). https://benedikt-bitterli.me/resources/.

Benedikt Bitterli, Fabrice Rousselle, Bochang Moon, José A. Iglesias-Guitián, David Adler, Kenny Mitchell, Wojciech Jarosz, and Jan Novák. 2016. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. *Computer Graphics Forum* 35, 4 (2016), 107–117.

Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. 2005. A Review of Image Denoising Algorithms, with a New One. *Multiscale Modeling & Simulation* 4, 2 (2005), 490–530.

H. C. Burger, C. J. Schuler, and S. Harmeling. 2012. Image Denoising: Can Plain Neural Networks Compete with BM3D?. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2392–2399.

Chakravarty R. A. Chaitanya, Anton Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Noisy Monte Carlo Image Sequences using a Recurrent Autoencoder. *ACM Trans. Graph. (Proc. SIGGRAPH)* (2017).

Robert L. Cook, Loren Carpenter, and Edwin Catmull. 1987. The Reyes Image Rendering Architecture. *SIGGRAPH Comput. Graph.* 21, 4 (Aug. 1987), 95–102.

Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. 2006. Image Denoising with Block-Matching and 3D Filtering. (2006).

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics Springer, Berlin.

Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. 2016. Deep Joint Demosaicking and Denoising. *ACM Trans. Graph.* 35, 6, Article 191 (Nov. 2016), 12 pages.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International conference on artificial intelligence and statistics*. 249–256.

Luke Goddard. 2014. Silencing the Noise on Elysium. In *ACM SIGGRAPH 2014 Talks (SIGGRAPH '14)*. ACM, New York, NY, USA, Article 38, 1 pages.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. http://arxiv.org/abs/1512.03385

James T. Kajiya. 1986. The Rendering Equation. *SIGGRAPH Comput. Graph.* 20, 4 (Aug. 1986), 143–150.

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. 34, 4, Article 122 (July 2015), 12 pages.

Nima Khademi Kalantari and Pradeep Sen. 2013. Removing the Noise in Monte Carlo Rendering with General Image Denoising Algorithms. 32, 2pt1 (2013), 93–102.

A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. Christensen, J. Hanika, C. Eisenacher, and G. Nichols. 2015. The Path Tracing Revolution in the Movie Industry. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH '15)*. ACM, New York, NY, USA, Article 24, 7 pages.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980

Diederik P Kingma and Max Welling. 2013. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521 (2015), 436–444.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and others. 2016. Photo-Realistic Single Image Super-Resolution using a Generative Adversarial Network. *arXiv preprint arXiv:1609.04802* (2016).

Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012. SURE-based Optimization for Adaptive Sampling and Reconstruction. *ACM Trans. Graph.* 31, 6, Article 194 (Nov. 2012), 9 pages.

Michael D. McCool. 1999. Anisotropic Diffusion for Monte Carlo Noise Reduction. *ACM Transactions on Graphics* 18, 2 (April 1999), 171–194.

Bochang Moon, Nathan Carr, and Sung-Eui Yoon. 2014. Adaptive Rendering Based on Weighted Local Regression. *ACM Trans. Graph.* 33, 5 (Sept. 2014), 170:1–170:14.

Bochang Moon, Jong Yun Jun, JongHyeob Lee, Kunho Kim, Toshiya Hachisuka, and Sung-Eui Yoon. 2013. Robust Image Denoising Using a Virtual Flash Image for Monte Carlo Ray Tracing. *Computer Graphics Forum* 32, 1 (2013), 139–151.

Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. 2016. Adaptive Polynomial Rendering. *To appear in ACM Trans. Graph. (Proc. SIGGRAPH)* (2016), 10.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A Generative Model for Raw Audio. *arXiv preprint arXiv:1609.03499* (2016).

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2011. Adaptive Sampling and Reconstruction using Greedy Error Minimization. *ACM Trans. Graph.* 30, 6, Article 159 (Dec. 2011), 12 pages.

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2012. Adaptive Rendering with Non-local Means Filtering. 31, 6, Article 195 (Nov. 2012), 11 pages.

Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. 2013. Robust Denoising using Feature and Color Information. *Computer Graphics Forum* 32, 7 (2013), 121–130.

Holly E. Rushmeier and Gregory J. Ward. 1994. Energy Preserving Non-Linear Filters. In *Proc. 21st annual Conf. on Computer graphics and interactive techniques (SIGGRAPH '94)*. ACM, 131–138.

Tim Salimans and Diederik P Kingma. 2016. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *Adv in Neural Information Processing Systems (NIPS)*.

Pradeep Sen and Soheil Darabi. 2012. On Filtering the Noise from the Random Parameters in Monte Carlo Rendering. *ACM Transactions on Graphics* 31, 3, Article 18 (June 2012), 15 pages.

Pradeep Sen, Matthias Zwicker, Fabrice Rousselle, Sung-Eui Yoon, and Nima Khademi Kalantari. 2015. Denoising Your Monte Carlo Renders: Recent Advances in Image-space Adaptive Sampling and Reconstruction. In *ACM SIGGRAPH 2015 Courses*. ACM, 11.

Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556* (2014).

Charles M. Stein. 1981. Estimation of the Mean of a Multivariate Normal Distribution. *The Annals of Statistics* 9, 6 (1981), 1135–1151. http://www.jstor.org/stable/2240405

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image Quality Assessment: from Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (April 2004), 600–612.

Junyuan Xie, Linli Xu, and Enhong Chen. 2012. Image Denoising and Inpainting with Deep Neural Networks. In *Advances in Neural Information Processing Systems*. 341–349.

Wenhan Yang, Jiashi Feng, Jianchao Yang, Fang Zhao, Jiaying Liu, Zongming Guo, and Shuicheng Yan. 2016. Deep Edge Guided Recurrent Residual Learning for Image Super-Resolution. *CoRR* abs/1604.08671 (2016). http://arxiv.org/abs/1604.08671

Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. 2016. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *arXiv preprint arXiv:1608.03981* (2016).

Henning Zimmer, Fabrice Rousselle, Wenzel Jakob, Oliver Wang, David Adler, Wojciech Jarosz, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. 2015. Path-space Motion Estimation and Decomposition for Robust Animation Filtering. *Computer Graphics Forum* 34, 4 (2015), 131–142.

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and Sung-Eui Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. 34, 2 (May 2015), 667–681.