Denoising with Kernel Prediction and Asymmetric Loss Functions

THIJS VOGELS, Disney Research
FABRICE ROUSSELLE, Disney Research
BRIAN MCWILLIAMS, Disney Research
GERHARD RÖTHLIN, Disney Research
ALEX HARVILL, Pixar Animation Studios
DAVID ADLER, Walt Disney Animation Studios
MARK MEYER, Pixar Animation Studios
JAN NOVÁK, Disney Research

Fig. 1. We introduce a novel, modular architecture based on kernel-predicting neural networks that performs multi-scale, temporal denoising of rendered sequences. Our network can be retrained using a small amount of data and performs robustly with artist-control over the variance-bias tradeoff on novel data, such as these two examples from our test set. © Disney / Pixar, © Disney.

We present a modular convolutional architecture for denoising rendered images. We expand on the capabilities of kernel-predicting networks by combining them with a number of task-specific modules, and optimizing the assembly using an asymmetric loss. The source-aware encoder—the first module in the assembly—extracts low-level features and embeds them into a common feature space, enabling quick adaptation of a trained network to novel data. The spatial and temporal modules extract abstract, high-level features for kernel-based reconstruction, which is performed at three different spatial scales to reduce low-frequency artifacts. The complete network is trained using a class of asymmetric loss functions that are designed to preserve details and provide the user with a direct control over the variance-bias trade-off during inference. We also propose an error-predicting module for inferring reconstruction error maps that can used to drive adaptive sampling. Finally, we present a theoretical analysis of convergence rates of kernel-predicting architectures, shedding light on why kernel prediction performs better than synthesizing the colors directly, complementing the empirical evidence presented in this and previous works. We demonstrate that our networks attain results that compare favorably to state-of-the-art methods in terms of detail preservation, low-frequency noise removal, and temporal stability on a variety of production and academic data sets.

0730-0301/2018/8-ART124
https://doi.org/10.1145/3197517.3201388

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

Additional Key Words and Phrases: Monte Carlo denoising, neural networks, kernel prediction, asymmetric loss, temporal, multi-scale

ACM Reference Format:
Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Trans. Graph.* 37, 4, Article 124 (August 2018), 15 pages. https://doi.org/10.1145/3197517.3201388

1 INTRODUCTION

Monte Carlo (MC) rendering is used ubiquitously in computer animation and visual effect productions [Keller et al. 2015]. Despite continuously increasing computational power, the cost of constructing light paths—the core component of image synthesis—remains a limiting practical constraint that leads to noise. Among the many strategies explored to reduce Monte Carlo noise, image-space denoising has emerged as a particularly attractive solution due to its effectiveness and ease of integration into rendering pipelines.

Until recently, the best-performing MC denoisers were hand-designed and based on linear regression models [Zwicker et al. 2015]. In publications from last year, however, Bako et al. [2017] and Chaitanya et al. [2017] demonstrated that solutions employing convolutional neural networks (CNN) can outperform the best zero- and first-order regression models under specific circumstances. Despite

this, the previous generation of hand-designed models are still used extensively in commercial rendering systems (e.g. RenderMan, VRay, and Corona). Furthermore, there are several well-known issues with neural networks—in particular with regards to data efficiency during training and domain adaptation during inference—which limit their broad application. We present several novel architectural extensions and a class of asymmetric loss functions that overcome these limitations and allow trading variance for bias at run-time.

Data-efficiency of deep learning remains an open challenge with larger networks requiring enormous training data sets to produce good results. This poses a particular problem for denoising since generating ground-truth renders to be used as targets for prediction in the supervised-learning framework is extremely computationally expensive. This issue impacts several areas including training, adaptation to data from different sources, and temporal denoising. We propose several solutions to overcome this problem.

First, our denoiser is based on the recently presented kernel-predicting (KPCN) architecture [Bako et al. 2017; Vogels 2016]. Intuitively, kernel prediction trades a larger inductive bias for lower-variance estimates resulting in faster and more stable training than direct prediction. In this work we provide theoretical reasoning for why KPCN converges faster. Specifically, we show that in the convex case, optimizing the kernel prediction problem using gradient descent is equivalent to performing mirror descent [Beck and Teboulle 2003] which enjoys an up-to-exponentially faster convergence speed than standard gradient descent.

Second, to integrate data from different sources (e.g. different renderers and auxiliary buffer sets), we propose *source-aware encoders* that extract low-level features particular to each data source. This allows the network to leverage data from multiple renderers during training by embedding different datasets into a common feature space. Furthermore, it enables a pre-trained network to be quickly adapted to a new data source with few training examples, but at the same time avoiding catastrophic forgetting [Kirkpatrick et al. 2017], which can result from naive fine-tuning.

Third, we propose an extension to the temporal domain—necessary for processing animated sequences—that requires less ground-truth data than previous approaches. Chaitanya et al. [2017] propose a recurrent model which they train using a converged reference image for each frame in the sequence. We present an alternative scheme that crucially does not require reference images for each input in the sequence. Instead, we combine feature representations from individual frames using a lightweight, temporal extension to KPCN. Our approach amortizes the cost of denoising each frame across multiple sliding temporal windows yet produces temporally stable animations of higher quality than a single-frame KPCN.

We incorporate these developments in a modular, *multi-scale* architecture that operates on a mip-map pyramid to reduce low-frequency noise. We employ a lightweight scale-compositing module trained to combine spatial scales such that blotches and ringing artifacts are prevented. We also propose a dedicated *error-predicting* module that approximates the reconstruction error. This enables adaptive sampling by iteratively executing the error prediction during rendering and distributing the samples according to the predicted error. We demonstrate that acknowledging the strengths and weaknesses of the denoiser in this way yields better results than adaptive sampling driven by the variance of rendered outputs.

Finally, we provide a mechanism for a user control over the trade-off between variance and bias. We propose *asymmetric loss functions* that magnify gradients during back-propagation when the result deviates strongly from the input. The asymmetry is varied during training and linked to an input parameter of the denoiser that provides the user with direct control over the trade-off between residual noise and loss of detail due to blurring or other artifacts—a crucial feature for production scenarios.

The remainder of the paper is organized as follows: In Section 2, we briefly review modern regression- and neural-network-based approaches to denoising. Our modular denoising architecture is described in Section 3, which introduces the source-aware encoder and a novel approach to temporal and multi-scale denoising. The class of asymmetric loss functions for incorporating user-controllable behavior is described in Section 4. Our approach to adaptive sampling is discussed in Section 5. In Section 7, we evaluate the performance of our denoiser on a mixture of production and freely-available scenes against state-of-the-art approaches and carefully study our design choices. Finally, we provide a theoretical analysis explaining the superior convergence of kernel prediction.

2 RELATED WORK

A large body of research has been devoted to MC denoising. We refer the reader to the review by Zwicker et al. [2015] and focus here on the most relevant and recent developments, and related work employing CNNs [LeCun et al. 2015].

2.1 Image-space Methods

Early work by Rushmeier and Ward [1994] pioneered the idea of denoising Monte Carlo renderings. McCool [1999] proposed the idea of using auxiliary buffers to improve the robustness of the denoising procedure, which was adopted in most current methods based on linear regression. These can be categorized depending on the polynomial order of the underlying model. Central to zero-order methods is the idea of extracting a weighting kernel, and computing the denoised result as a linear combination of kernel-weighted noisy pixels. Most approaches use joint bilateral [Sen and Darabi 2012] or non-local, patch-based schemes [Buades et al. 2005; Kalantari et al. 2015; Moon et al. 2013; Rousselle et al. 2013] to compute kernel weights. Several works have also explored the possibility of employing first-order [Bauszat et al. 2011; Bitterli et al. 2016; Moon et al. 2014] and even higher-order [Moon et al. 2016] models. These represent the current state of the art in the context of linear-regression denoising.

Temporal Stability. Several of the aforementioned approaches extend well to the temporal domain. Methods based on patch-based metrics [Bitterli et al. 2016; Buades et al. 2008; Kalantari et al. 2015; Rousselle et al. 2012, 2013] do so by considering spatial neighborhoods in temporally near frames. The main drawback of these methods is that the weighting of neighbor contributions is unreliable if the center frame has a bad signal-to-noise-ratio—even if the neighbors are noise-free.

Our approach is largely similar to the patch-based methods, but we provide the denoising network with all temporal information to ensure the extracted kernels are not biased towards the center frame. Inspired by the work of Delbracio et al. [2014], we also operate on multiple scales to further reduce low-frequency artifacts. However, in contrast we recombine the scales adaptively using a trained scale-compositing module.

Error Estimation and Adaptive Sampling. Techniques for adaptive rendering initially focused on the sampling problem alone, using the contrast measure of extrema sample values [Mitchell 1987] or by modeling the human visual system [Bolin and Meyer 1998; Ramasubramanian et al. 1999] as a proxy for perceptual significance. Later works considered the combination of adaptive sampling and reconstruction, either using the sample-based contrast measure of noisy inputs [Hachisuka et al. 2008; Overbeck et al. 2009], or by using the reconstruction Mean relative Squared Error (MrSE). This enabled a tight coupling of the sampling and reconstruction steps for linear filters [Moon et al. 2014; Rousselle et al. 2011], non-linear filters with known derivatives [Li et al. 2012], or arbitrary filters using pairs of independent noisy renderings [Bitterli et al. 2016; Rousselle et al. 2012]. Our work also leverages an estimation of the reconstruction error to distribute samples, but we employ a neural network taking only the noisy and denoised images as input, allowing for robust loss estimates with no constraints on the filter, or reliance on pairs of independent renderings. Unlike recent work, we use the Symmetric Mean Absolute Percentage Error (SMAPE) of the reconstruction to distribute samples, instead of the MrSE, which leads to more robust reconstructions according to the perceptual SSIM metric [Wang et al. 2004].

2.2 Denoising Neural Networks

Neural networks have been successfully applied to a vast number of problems. Here we review applications to denoising. Jain and Seung [2008] employed four-layer CNNs and Burger et al. [2012] used fully-connected multi-layer perceptrons to denoise natural images corrupted with Gaussian noise; the latter demonstrating comparable performance to BM3D [Dabov et al. 2006].

Several works proposed alternatives to predicting the denoised colors directly. Kalantari et al. [2015] used a multi-layer perceptron to predict the optimal parameters of cross-bilateral and non-local-means denoisers for filtering MC renderings. Bako et al. [2017] demonstrated that a CNN can be trained to achieve state-of-the-art results via a kernel-based reconstruction [Jia et al. 2016; Vogels 2016], similar in spirit to zero-order regression models. Kernel-predicting networks have also been recently utilized for burst-denoising of natural images [Mildenhall et al. 2017] and more widely for temporal upsampling of video sequences [Niklaus et al. 2017].

We complement previous works utilizing a kernel-predicting convolutional network (KPCN) in numerous ways. First, we provide theoretical evidence supporting the earlier empirical claims that KPCN trains faster than a direct-predicting network. Second, we extend the concept over a temporal window predicting kernels for an entire sequence of subsequent animation frames.

With regards to temporal denoising, Chaitanya et al. [2017] propose a convolutional architecture which enforces temporal stability via recurrent connections that provide a mechanism to incorporate information from past frames in a sequence. We argue that our approach is better suited for applications where future frames are also available—such as movie rendering—supplementing our claim with empirical evidence.

3 MODULAR ARCHITECTURE

In this section, we first formalize the denoising operation as a supervised learning problem and introduce necessary notation. As a solution to the problem we propose a neural network with a modular architecture based on kernel-predicting CNNs. The modules are designed to deliver a solution with specific properties such as temporal stability, and handling of diverse sets of inputs.

Problem Statement. Denoising Monte Carlo renderings can be formally described as a mapping g of an input tuple $\mathbf{x} = \{\mathbf{c}, \mathbf{f}\}$ to an estimate $\mathbf{d}$ of the ground-truth color $\mathbf{r}$. For a given pixel p, the noisy values in the input tuple are obtained from a renderer as average RGB color $\mathbf{c}_p$ and (optional) auxiliary buffers $\mathbf{f}_p$, e.g. color variance, surface normal, or albedo, over multiple samples contributing to p.

Similar to previous works [Bako et al. 2017; Chaitanya et al. 2017], we use a convolutional neural network parameterized by a set of weights $\boldsymbol{\theta}$ to represent g. The optimal parameters $\widehat{\boldsymbol{\theta}}$ are estimated via supervised learning that utilizes a large dataset with N example pairs of noisy inputs and corresponding ground-truth color images, $\mathcal{D}_N = \{(\mathbf{x}^1, \mathbf{r}^1), \ldots, (\mathbf{x}^N, \mathbf{r}^N)\}$. The learning aims to minimize the average distance between the ground-truth $\mathbf{r}$ and the denoised image $\mathbf{d} = g(\mathbf{x}; \boldsymbol{\theta})$ via a loss function ℓ:

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \ell(\mathbf{r}^n, g(\mathbf{x}^n; \boldsymbol{\theta})). \tag{1}$$

In practice, we use the *symmetric mean absolute percentage error*[1] (SMAPE) for ℓ:

$$\ell(\mathbf{r}, \mathbf{d}) = \frac{1}{3|\mathcal{I}|} \sum_{p \in \mathcal{I}} \sum_{c \in C} \frac{|\mathbf{d}_{p,c} - \mathbf{r}_{p,c}|}{|\mathbf{d}_{p,c}| + |\mathbf{r}_{i,c}| + \varepsilon}. \tag{2}$$

Here $\mathcal{I}$ is the domain of pixels in the image, C are the three color channels, and we use $\varepsilon = 10^{-2}$. The choice for SMAPE is motivated by it's stable behavior in HDR images.

In what follows, we describe a modular design that reuses trained components in different networks. The architecture facilitates debugging and permits constructing large networks that would be difficult to train all together due to large memory requirements.

3.1 Single-frame Denoiser

The core of our denoiser is a kernel-predicting CNN [Bako et al. 2017; Vogels 2016] designed to denoise a single frame; see Figure 2 for illustration. In contrast to Bako et al. [2017], who use a vanilla 9-layer CNN, we employ *residual blocks* [He et al. 2016] consisting of two layers bypassed by a skip connection; see Figure 2 (right). The skip connection enables chaining many such blocks without optimization instabilities. With up to 48 layers, we obtain significantly increased performance as demonstrated in Section 7.

[1] Here "symmetric" refers to the equivalent role of $\mathbf{d}$ and $\mathbf{r}$ in the equation. This concept is orthogonal to the concept of "asymmetric loss functions" introduced in Section 4.

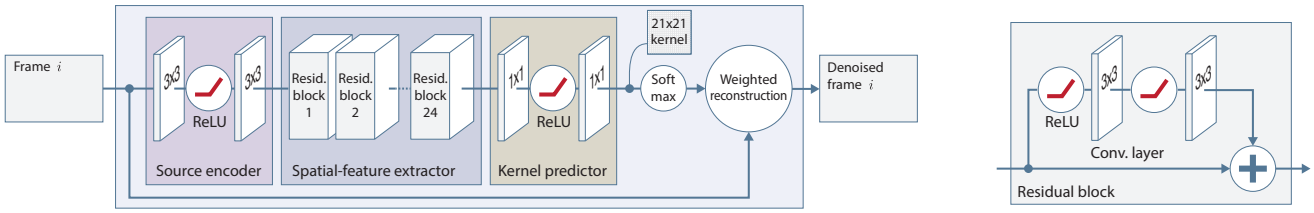


Fig. 2. Our single-frame denoiser module consists of a source-aware encoder followed by an extractor of spatial features, which are fed into a KPCN kernel-predicting module. The scalar kernels are normalized using the softmax function and applied to the noisy input to obtain the denoised image. The spatial-feature extractor consists of a number of residual blocks (right), each consisting of two 3×3 convolutional layers bypassed by a skip connection.

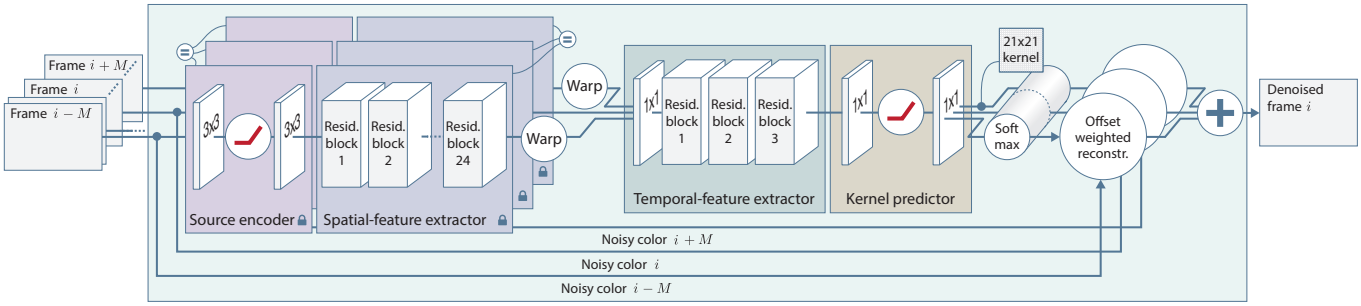


Fig. 3. Our temporal denoiser module first extracts spatial features from individual frames, which are then warped using motion vectors to match the time of the center frame. All features are concatenated and fed into an extractor of temporal features, which are used to predict a kernel for each frame in the temporal window. The kernels are jointly normalized to sum up to 1 and applied to the input colors to produce the denoised version of the center frame. We use a pre-trained, single-frame source-aware encoder and spatial-feature extractor with the same parameters $\widehat{\boldsymbol{\theta}}$ for all frames (symbolized by "=") that are "locked" during the training of temporal-feature extractors and kernel predictors.

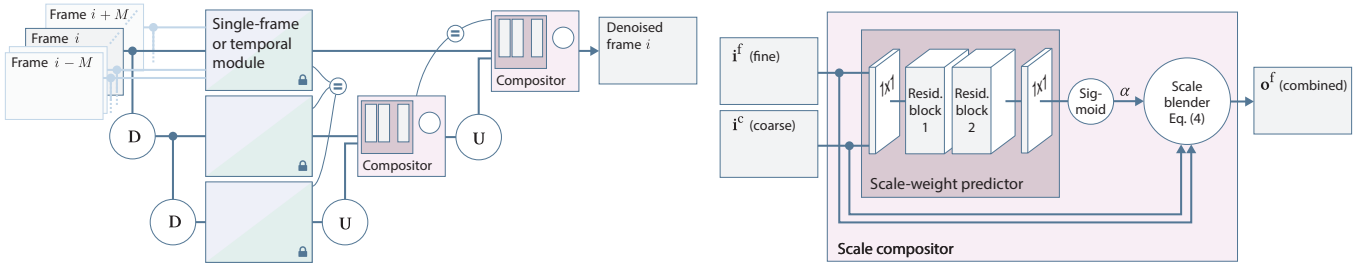


Fig. 4. The multi-scale single-frame/temporal denoiser first decomposes the inputs into several scales. **D** represents a downsampling operation and **U** represents upsampling. The scales are denoised independently and combined using a spatially varying blending parameter α that balances low frequencies from the finer and coarser scales. The parameter α is produced by a relatively shallow CNN (right).

A distinct feature of the kernel-predicting network is that the denoised color $\mathbf{d}_p$ is reconstructed as a linear combination of input colors in a $k \times k$ neighborhood $\mathcal{N}(p)$ around p. The stack of residual blocks can thus be interpreted as an extractor of spatial features that are passed down to another module, which produces a $k \times k$ kernel of scalar weights. The denoised color $\mathbf{d}_p$ is then obtained by applying the weights to the input colors and summing over them:

$$\mathbf{d}_p = g_p(\mathbf{x}; \boldsymbol{\theta}) = \sum_{q \in \mathcal{N}(p)} w_{pq}\, \mathbf{c}_q. \tag{3}$$

The kernel weights are normalized using the softmax function:

$$w_{pq} = \frac{\exp([\mathbf{z}_p]_q)}{\sum_{q' \in \mathcal{N}(p)} \exp([\mathbf{z}_p]_{q'})}, \tag{4}$$

where $\mathbf{z}_p \in \mathbb{R}^{k \times k}$ represents the kernel predicted for pixel p, and $[.]_i$ retrieves the i-th entry in a linearized version of the kernel. Softmax normalization ensures that $0 \leq w_{pq} \leq 1, \forall q \in \mathcal{N}(p)$ and $\sum_{q \in \mathcal{N}(p)} w_{pq} = 1$. In Section 8, we provide a theoretical argument for the superiority of the kernel-based reconstruction, in terms of optimization speed, over predicting the colors directly.

3.2 Source-aware Encoder

Input features obtained from different renderers are not always equivalent. Their varying samplers, reconstruction filters, and variance estimators create the need for source-aware treatment. The tasks of denoising, however, is the same irrespective of the renderer.

To enable the denoising of images across different renderers with potentially varying sets of auxiliary buffers, we therefore introduce

source-aware encoders into our architecture. Encoding modules, with input features unique to the renderer and independent parameters, constitute the first few layers of the network. All following layers of the network have identical parameters across all sources.

Our approach to training source-aware encoders is similar in spirit to the work of Andreas et al. [2016] in visual question answering, which used dynamically assembled task-specific modules. By sharing a common denoising back-end for all data sources, the source-aware encoder is effectively designated to extract a common, low-level representation to unify the various inputs. Such design enables adaptation to new data sources by training new encoders while keeping the rest of the network fixed. The retraining procedure is light on training data as the source-aware encoders are rather shallow, consisting of only two 3×3 convolutional layers.

A source-aware encoder, feature extractor, and kernel predictor, as illustrated in Figure 2, are the main building blocks of our system and jointly represent a *single-frame module* for obtaining a denoised image from the input and features of a single frame.

3.3 Temporal Denoiser

The single-frame module may produce an animated sequence with severe temporal artifacts—flickering—when executed on a sequence of frames independently. This is because each of the denoised frames is "wrong" in a slightly different way. In order to achieve temporal stability, we use the same approach as most prior works: we consider a sequence of frames—a temporal neighborhood $\mathcal{T} = [\{\mathbf{c}^{i-M}, \mathbf{f}^{i-M}\}, \cdots \{\mathbf{c}^{i}, \mathbf{f}^{i}\}, \cdots \{\mathbf{c}^{i+M}, \mathbf{f}^{i+M}\}]$ of $2M+1$ input tuples—when denoising a single frame. This has two benefits: first, the temporal neighbors provide additional information that helps to reduce the error in $\mathbf{d}$; second, since the neighborhoods of consecutive frames overlap, the residual error in each frame will be correlated, reducing the highly perceptive temporal flicker.

Our solution to incorporating temporal neighbors is driven by a number of observations. We target applications in which the denoiser has access to both past and future frames. The temporal neighborhood may occasionally be asymmetric, e.g. when denoising the first or last frame of a sequence. And lastly, the cost of denoising a sequence should scale well to large temporal neighborhoods.

We propose to pre-process individual frames of the sequence independently and combine them using a temporal kernel-predicting network. The complete architecture is detailed in Figure 3. We first extract spatial features from each of the input frames. In order to align the features from animated content, we project each of the center frame's pixels p to all input frames in $\mathcal{T}$ using motion vectors, and gather the spatial features of the input frames along the motion trajectory at the position p. We either obtain motion vectors from the renderer or use optical flow. The gathered features at each pixel are then concatenated and input into a temporal-feature extractor, which consists of three residual blocks. As in the single-frame network, the extractor produces features that are then fed into a one-layer kernel predictor, only this time we have an independent kernel predictor for each of the frames. At each pixel, the multiple kernels are jointly normalized using a softmax. The joint normalization ensures that we can obtain the final denoised frame by simply adding the kernel-weighted noisy inputs.

There are two ways to ensure that in each frame, the kernels for a pixel p in the center frame i are applied to pixels that correspond well content-wise to p. The first approach is to warp the kernels pixel-by-pixel into adjacent frames using motion vectors. This is equivalent to applying unwarped kernels to warped images. The second approach is to rigidly offset—or "reposition"—kernels according to inverse motion vectors the center frame. While both approaches performed on par in our experiments, we opt for the second approach, as we consider it the corresponding inverse to "gathering" spatial features.

Parameter Reuse and Training. We use the same set of network parameters across all the instantiations of the source-aware encoder and spatial-feature extractor. These two modules are pre-trained independently as part of a single-frame network, which is optimized using a set of noisy/ground-truth image pairs. The parameters are then locked (e.g. treated as constant during later backpropagation) and the modules instantiated across all input frames in the temporal neighborhood. Training the rest of the temporal network then requires optimizing only the parameters of the relatively small temporal-feature extractor and kernel predictor. The optimization of the temporal modules is performed using pairs of noisy input *sequences* and reference images for the center frame of the sequence.

Discussion. Our approach to temporal denoising is substantially different from the work by Chaitanya et al. [2017] who utilize recurrent connections to accumulate information over subsequent frames. The two works differ primarily due to the very different application they target. Since their target is real-time rendering, Chaitanya et al. are able to build a temporal context using only past frames. On the other hand, we are able to make use of a symmetrical neighborhood around the frame to be denoised. In Section 7 we investigate the effect of different types of temporal context and neighborhood size.

3.4 Multi-scale Architecture

While denoising algorithms are typically good at removing high-frequency noise, they tend to leave low-frequency residual noise; CNN-based denoisers are no exception. Inspired by the work of Delbracio et al. [2014], we propose to improve the performance by filtering at different spatial scales.

Scale Decomposition. For an input frame (or a sequence of frames), we construct a three-level pyramid with the input on top and lower levels being obtained by a uniform 2×2 down-sampling. Each lower/coarser level spans a four-times larger region than the previous one. The auxiliary buffers in $\mathbf{f}$ are also down-sampled with the only caveat that variance buffers are additionally divided by a factor of 4 to account for reduction in noise. The (down-sampled) inputs of each scale enter a single-frame or a temporal module, depending on whether we are denoising a single frame or a sequence.

Scale Composition. The denoised results from individual scales are progressively combined—from the coarsest to the finest scale—using a *scale-compositing module*. The module accepts two images produced by denoising two adjacent scales of a frame (or a sequence); a coarse-scale image $\mathbf{i}^{\mathrm{c}}$ and fine-scale image $\mathbf{i}^{\mathrm{f}}$. The images are input to a convolutional network that extracts a per-pixel scalar weight

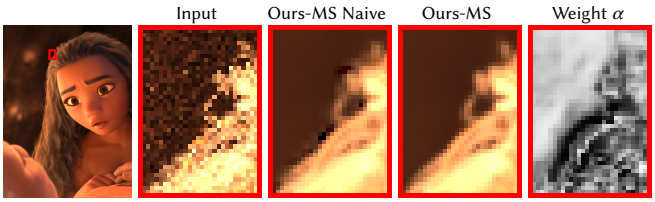


Fig. 5. Multi-scale reconstruction. Aliasing in the mip-map decomposition can lead to ringing artifacts in the reconstruction (Ours-MS Naive). Our weighted reconstruction (Ours-MS) modulates the contributions from coarser scales on a per-pixel basis using weights α (brighter corresponds to higher contribution of coarser scale), thereby alleviating ringing. © Disney.

α_p, which is used to blend the two images producing the output:

$$\mathbf{o}_p = \mathbf{i}^{\mathrm{f}}_p - \alpha_p[\mathbf{UDi}^{\mathrm{f}}]_p + \alpha_p[\mathbf{Ui}^{\mathrm{c}}]_p. \tag{5}$$

The blending takes the fine-scale image $\mathbf{i}^{\mathrm{f}}$ and replaces its low frequencies with low frequencies obtained from the coarse-scale image $\mathbf{i}^{\mathrm{c}}$. The $\mathbf{D}$ and $\mathbf{U}$ are 2×2-downsampling and nearest-neighbor-upsampling operators, respectively, $\mathbf{UDi}^{\mathrm{f}}$ extracts low frequencies from the fine-scale image, and $\mathbf{Ui}^{\mathrm{c}}$ refers to the upsampled result from the coarse-scale; see the right half of Figure 4 for an illustration.

Parameter Reuse and Training. Similarly to the temporal architecture, the single-frame and temporal modules applied to different scales share the same set of pre-trained, locked parameters. The scale-weight predictors across different scales also share the same set of parameters. Since the scale-compositing module appears twice in the three-level hierarchy, it is optimized using backpropagation twice for each entry in the training set, improving data-efficiency.

Discussion. The key difference between our multi-scale architecture and the one proposed by Delbracio et al. [2014] is that we employ a weighted scale composition, where the weights are predicted by the network. In hierarchical schemes, ringing artifacts due to aliasing in the decomposition are a recurrent problem. Our scale composition mitigates these artifacts by weighting the contribution of the coarser levels of the mip-map layers on a per-pixel basis. This mechanism effectively disables the multi-scale reconstruction if the denoised output is inconsistent across the mip-map layers, which alleviates ringing artifacts as we illustrate in Figure 5.

4 ASYMMETRIC LOSS FUNCTIONS

In some applications, feature-film production in particular, it may be desirable to retain some residual noise rather than sacrifice detail by over-blurring. Often this choice is an artistic decision and could be made on a movie, or scene, basis. Therefore it is important that the end users of the denoiser have control over the level to which residual noise is retained. Next, we describe the concept of the *asymmetric loss*, which allows the denoiser (and the user) to trade between variance and bias.

Assuming the network is to be optimized using loss ℓ, we allow controlling the aggressiveness of the denoiser by instead optimizing it using a modified, asymmetric loss

$$\ell'_\lambda(\mathbf{d}, \mathbf{r}, \mathbf{c}) = \ell(\mathbf{d}, \mathbf{r}) \cdot (1 + (\lambda - 1)H\left((\mathbf{d} - \mathbf{r})(\mathbf{r} - \mathbf{c})\right)), \tag{6}$$

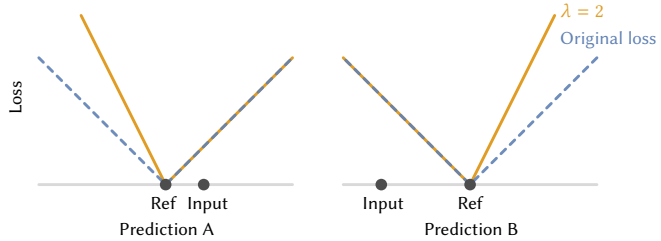


Fig. 6. Asymmetric loss function. The dashed blue line indicates an original loss function and the orange line is the asymmetric version with $\lambda = 2$. The side of the asymmetry varies per pixel depending on whether the input value at that pixel 'Input' is larger or smaller than the ground-truth 'Ref'.

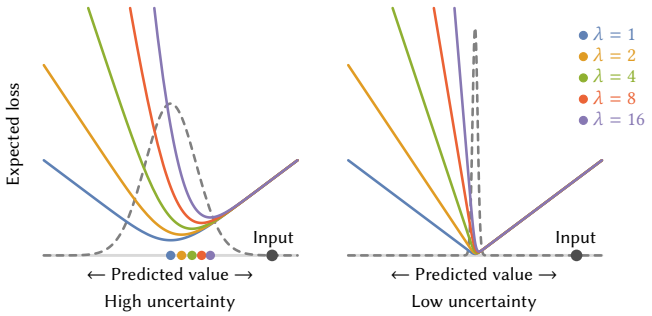


Fig. 7. Expected loss as a function of the predicted intensity for a given pixel for a wide and narrow distribution p indicated by dashed lines. The noisy input color is denoted by 'Input'. Colored solid lines are the expected loss under asymmetric loss with varying levels of asymmetry (blue amounts to a symmetric loss). The wider the likelihood, which can be interpreted as larger uncertainty of the network, the more the minimum of this function shifts towards the input as the asymmetry level increases.

where the Heaviside function $H(\cdot)$ returns 1 if the argument is positive and 0 otherwise. If the differences $(\mathbf{d} - \mathbf{r})$ and $(\mathbf{r} - \mathbf{c})$ have the same sign—i.e. the filtered result and the input are *not* on the same "side" relative to the reference—then we penalize such result by multiplying the original loss ℓ by the slope parameter λ; see Figure 6 for illustration.

The loss ℓ'_λ is asymmetric in the sense that, given two solutions that are equally close to the reference and $\lambda > 1$, the loss favors the solution that deviates less from the input. This has the effect of producing solutions that retain some of the input noise but only in situations when the minimum loss cannot be reached.

4.1 Decision-theoretic analysis

The mechanism that allows a neural network to remain conservative in certain situations when trained with an asymmetric loss function can be understood through a decision-theoretic analysis. For a (pixel-wise) loss $\ell(\mathbf{r}_p, \mathbf{d}_p)$ and a distribution $p(\mathbf{r}_p \mid \mathbf{c}_p)$ which governs the probability of observing a particular reference given a noisy input pixel, p, the *Bayes optimal* solution [Murphy 2012] is given by $\mathrm{argmin}_{\mathbf{d}_p} \tau(\mathbf{d}_p \mid \mathbf{c}_p)$ where

$$\tau(\mathbf{d}_p \mid \mathbf{c}_p) = \mathbb{E}_{p(\mathbf{r}_p \mid \mathbf{c}_p)}\left[\ell(\mathbf{r}_p, \mathbf{d}_p)\right] = \int \ell(\mathbf{r}_p, \mathbf{d}_p) p(\mathbf{r}_p \mid \mathbf{c}_p)\, \mathrm{d}\mathbf{r}_p. \tag{7}$$

For loss functions that operate only on the distance between the two arguments, i.e. $\ell(\mathbf{r}, \mathbf{d}) = \ell(\mathbf{d} - \mathbf{r}, 0)$, Equation (7) takes the form

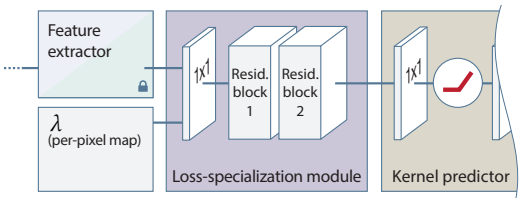


Fig. 8. We train a shallow loss-specialization module and kernel predictor on top of a pre-trained feature extractor (either spatial or spatio-temporal.) The loss-specialization module additionally receives a per-pixel map of the asymmetric slope parameter λ that controls denoising behavior.

of a convolution of the loss with the likelihood,

$$\tau(\mathbf{d}_p \mid \mathbf{c}_p) = \int \ell(\mathbf{d}_p - \mathbf{r}_p, 0) p(\mathbf{r}_p \mid \mathbf{c}_p)\, \mathrm{d}\mathbf{r}_p. \tag{8}$$

In case of symmetric loss functions and unimodal, symmetric distributions p, the convolution changes only the shape of the loss, not its minimum.

Our proposed ℓ'_λ (Equation (6)) is, however, asymmetric with a steeper slope on the side opposite the input to penalize strong deviations (e.g. excessive blurring). The convolution will thus shift the minimum towards the noisy input with the parameter λ controlling the amount of offsetting; see Figure 6 and Figure 7 for an illustration. Therefore, solutions closer to the noisy input will be preferred. The offsetting from the minimum is more pronounced when the distribution p is wide, and less-so when p is narrow.

If we assume that the network implicitly learns to estimate $q \approx p(\mathbf{r}_p \mid \mathbf{c}_p)$, it will minimize the expected loss $\mathbb{E}_q\, \ell'_\lambda(\mathbf{r}_p, \mathbf{d}_p, \mathbf{c}_p)$ by adjusting its output based on its uncertainty in the relationship between input and reference and the value of λ. For pixels where the uncertainty is high, it will tend to retain some of the input noise, rather than blurring, thus preserving detail.

4.2 Run-time artistic control

The simplest way to use an asymmetric loss function is to train a network from scratch with ℓ'_λ for a fixed slope parameter $\lambda > 1$. Here, the choice of λ makes the denoiser more or less conservative. We take an alternative approach where a single network is optimized for a range of slope parameters at once. We train with random values for λ, and provide λ as an input to the network. This optimizes the network for a whole class of loss functions that can be chosen from by the user at run-time, offering direct control over the behavior of the denoiser. Higher values lead to conservative denoising, permitting the denoiser to leave residual noise. $\lambda = 1$ reverts back to the symmetric loss.

4.3 Modular training

We take a modular approach to training for asymmetric loss functions. We use a feature extractor that was pre-trained with loss $\ell(\mathbf{d}, \mathbf{r})$, and optimize a lightweight *loss-specialization* module, which receives an additional single-channel input λ, using the asymmetric loss $\ell'_\lambda(\mathbf{d}, \mathbf{r}, \mathbf{c})$; see Figure 8. To support spatially varying λ at run-time, we use a different λ for each pixel during training; we draw $\log(\lambda)$ from a uniform distribution $\mathcal{U}(0, \log 10)$ thereby optimizing for each pixel in a mini-batch using a slightly different loss function.

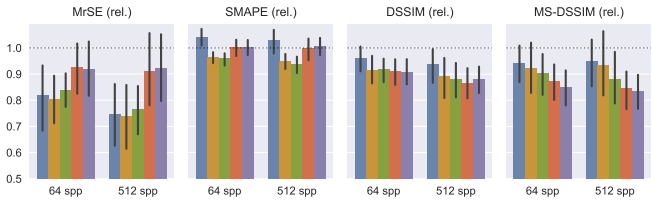


Fig. 9. Adaptive sampling proportional to various guiding metrics (■ relative variance, ■ relative squared error, ■ SMAPE, ■ DSSIM, ■ MS-DSSIM) with average budgets of 64 and 512 samples per pixel. The average error over the Moana test set is reported according to multiple metrics. The metrics are relative to the quality of uniform sampling (lower is better). All guiding metrics are predicted, i.e. computed with no knowledge of the ground-truth, using a relatively small network. The symmetric absolute percentage error has the interesting property of offering a good compromise according to perceptual evaluation metrics (DSSIM, MS-DSSIM) and others alike; see Figure 19 for a visualization of the sampling maps.

Because λ enters the network only near the end, all preceding computation can be cached, enabling a rapid user interaction with the parameter and a user-friendly adjustment of the performance.

5 ADAPTIVE SAMPLING

In this section, we discuss how to further reduce the final error via adaptive sampling, which makes the likely uneven noise distribution over the image plane (e.g. due to varying materials, depth of field, or lighting conditions) more uniform. A common solution is to adapt the number of per-pixel samples proportional to the relative variance of each pixel's sample mean. This approach, however, disregards the denoising step, which has a large impact on the reconstruction error. Consequently, many algorithms [Li et al. 2012; Moon et al. 2014; Rousselle et al. 2011] alternate sampling and denoising steps, and distribute samples not according to the input variance, but rather according to the estimated reconstruction error.

We follow a similar approach: the first iteration starts with 16 samples per pixel (non-adaptive). After each iteration we denoise the image and execute an *error-predicting network* that predicts error maps. The next iterations double the total number of samples across all pixels, allocating samples to pixels proportionally to the predicted error maps. The error-predicting network consists of six residual blocks, and learns to estimate error maps for the *guiding metric* $\ell(\mathbf{d}, \mathbf{r})$ from pairs of noisy and denoised images $(\mathbf{c}, \mathbf{d})$. The network is optimized using an L1 loss.

We considered four guiding metrics: the relative squared error (MrSE), SMAPE (Equation (2)) , structural dissimilarity (DSSIM), and its multi-scale version (MS-DSSIM). We trained four error-predicting networks, each optimized to predict one of these metrics, and tested the resulting adaptive sampling on 11 scenes from Moana. We also considered adaptive sampling driven by the relative variance of the input as a baseline solution that does not acknowledge the denoiser. Figure 9 shows the average performance of each of the error-predicting networks over the 11 scenes, according to various loss functions. The errors are expressed relative to the error achieved by uniform sampling with the same average sample count. Beyond observing that each guiding metric for sampling achieves good results according to the corresponding loss function, e.g. distributing

samples according to the predicted DSSIM error-map achieves a low DSSIM loss after denoising, we notice that the input's relative variance is a poor guide according to perceptual metrics. Interestingly, predicting SMAPE offers a good compromise on all metrics.

6 EXPERIMENTAL SETUP

In this section we describe the various datasets used for training and evaluation. We also describe how our model was trained in detail.

6.1 Training, Validation, and Testing Data

For training, we used three data sets—Moana, Cars, and Tungsten—where the first two contain production data and the last one consists of data from publicly available scenes rendered with the Tungsten renderer. Each dataset features different visual content and was generated with a different renderer (Disney's Hyperion renderer, RenderMan, and Tungsten respectively) with different approaches to obtaining motion vectors. The training sets consists of 7-frame sequences—each rendered at several sample-per-pixel (spp) rates that were produced by writing out intermediate results progressively during rendering. We rendered a high-quality reference for the center frame of each sequence and use it as the target for prediction. We held out several frames from each of these sets for validation (e.g. evaluating convergence and picking hyperparameters).

In order to evaluate performance, we prepared five test sets. Three of them comprise additional held-out frames from the same sources as our training data and include the same visual content. The remaining two—Olaf and Coco—represent content that none of the networks in our experiments was trained on, enabling us to test the ability to generalize and adapt to unseen data. Detailed information about each dataset is given in Table 1.

In all cases, the inputs to the network consist of log(1 + color) (3 channels), relative color-variance (1 channel), log(1 + albedo) (3 channels), relative albedo-variance (1 channel), normal (3 channels) and relative normal-variance (1 channel). Here, relative variance is an estimate of the pixel variance divided by the corresponding squared sample mean. For non-negative features, such a relative variance is bounded between 0 and 1. The definition of each buffer may vary depending on the renderer, e.g. color-variance is obtained differently for low-discrepancy and independent samplers.

Similarly to Bako et al. [2017], we decompose rendered outputs into diffuse and specular buffers and factorize the diffuse channel into irradiance and albedo. The three resulting components are denoised separately by a *single* network, wich an input flag indicating irradiance decomposition. This is unlike Bako et al., who employ a different network for denoising each component. We use the same source-aware encoder for all components.

6.2 Implementation and Training

We implement our networks in TensorFlow [Abadi et al. 2015] and optimized them for the SMAPE loss (Equation (2)) using Adam. Trainable weights are initialized using Xavier initialization [Glorot and Bengio 2010]. All KPCNs predict 21×21 scalar kernels.

To perform training, we randomly extract 128×128 patches and feed them into the network in mini-batches of size 12 (learning rate, $\eta = 10^{-4}$) for training the single-frame and multi-scale modules, and mini-batches of size 3 ($\eta = 0.25 \times 10^{-4}$) for training the temporal network. The patches are selected adaptively depending on the content using the selection process described by Bako et al. [2017].

During training, we evaluate the performance on a validation set—we use one that corresponds to the training set but contains different images—after every 2048 iterations. We terminate training when the training wall-clock time reaches seven days and retain the best-performing instance.

Table 1. Training, validation, and test datasets used in our experiments. The data column reports the number of unique, 7-frame-long sequences × the number of different spp rates that they were obtained with.

Name	Used for	Data	Characteristics
Moana	Training	174×4	Hyperion renderer; adaptive low-discrepancy sampling; rendered motion vectors
	Validation	5×4	
	Testing	4×4	
Olaf	Testing	10×4	
Cars	Training	290×3	RenderMan renderer; uniform low-discrepancy sampling; optical-flow motion vectors
	Validation	6×3	
	Testing	6×3	
Coco	Testing	10×4	
Tungsten	Training	1200	Tungsten renderer; uniform independent sampling; rendered motion vectors
	Validation	3×4	
	Testing	6×4	

6.3 Comparisons

We compare our proposed denoiser to two state-of-the-art denoisers: NFOR [Bitterli et al. 2016] and a variant of the recurrent approach proposed by Chaitanya et al. [2017]. In order to ensure a fair comparison to the latter, we pre-trained a single-frame, direct-predicting network with the same dimensions as our proposed network. We then added recurrent connections to obtain a temporal context and trained it on sequences to directly predict the denoised color of the center frame. In this way, the recurrent network has an equivalent number of parameters and access to the same amount of training data as our proposal. We refer to this approach as R-DP. We also considered a direct reimplementation of the method by Chaitanya et al. [2017], but it never yielded better results than R-DP. We thus focus on comparing our method and R-DP, which puts emphasis on the main concepts rather than particular implementation details.

7 RESULTS

We now demonstrate the impact of the various components of our modular architecture, as well as our asymmetric loss and adaptive sampling scheme. While some improvements may appear subtle, they are important for high quality results and use in production environments; the results highlighted in this section can be better appreciated on full resolution images, and we refer the reader to our supplemental material which features an interactive web-based viewer that allows for direct comparison of all methods. We also provide a video for comparing temporal results.

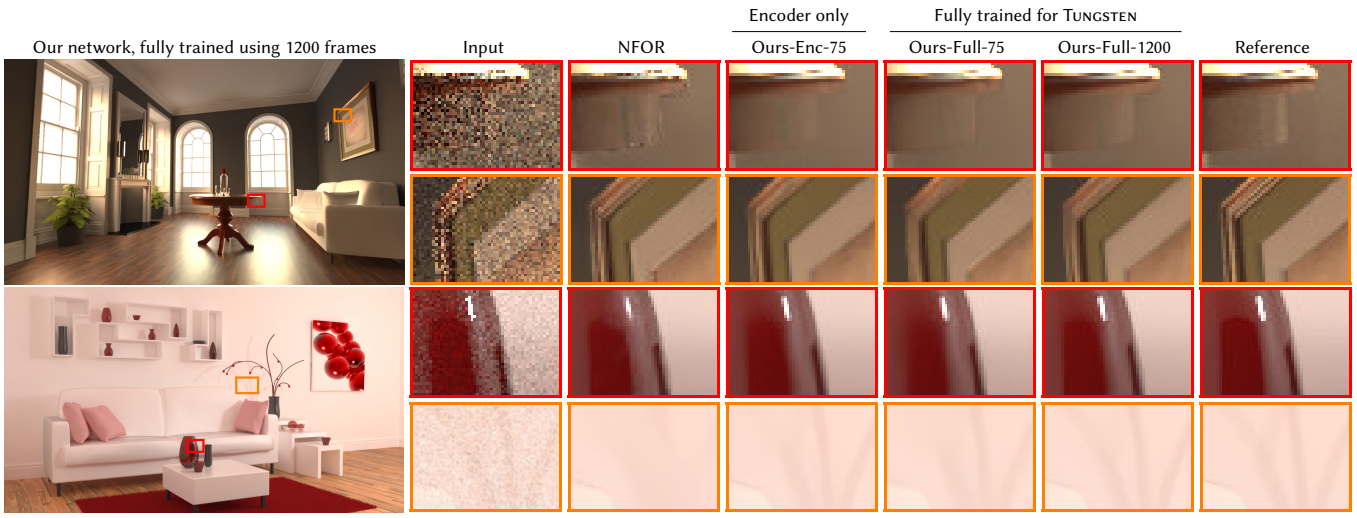


Fig. 10. Comparison to the NFOR denoiser [Bitterli et al. 2016] on the public Tungsten dataset. We experimented with two network variants: 1) training a Tungsten-aware encoder for a fixed network pre-trained on Moana and Cars using 75 training Tungsten frames (Ours-Enc-75), and 2) training a Tungsten-specialized network from scratch using 75 and 1200 training frames (Ours-Full-75, Ours-Full-1200). The best result overall is obtained with the specialized network using the full 1200-frames training set, but with the smaller 75-frame training set we better preserve details when training an encoder (Ours-Enc-75) for an existing network than when training a specialized network from scratch (Ours-Full-75). All network configurations lead to more visually pleasing and consistent results than NFOR. See Figure 11 for plots showing the network reconstruction accuracy at varying training set sizes, relative to NFOR.

Table 2. Source-aware encoders. We evaluate the performance of four networks on data from two renderers according to the mean DSSIM error relative to the input. Networks trained on data from one renderer perform poorly on test data from the other renderer (yellow cells). Networks trained on inputs from both renderers (bottom two lines) yield good performance in all situations. Using a dedicated source-aware encoder for each renderer allocates the first few layers to source-specific processing. This enables lightweight adaptation to new renderers and/or feature sets (see Figure 11).

DSSIM (rel.)	Hyperion		RenderMan	
	Moana	Olaf	Cars	Coco
Trained on Moana only	13.31%	5.25%	18.89%	18.68%
Trained on Cars only	21.08%	8.67%	7.89%	12.25%
Trained on both	13.19%	5.05%	7.97%	12.43%
Trained on both w/ encoders	12.91%	5.07%	7.87%	12.18%

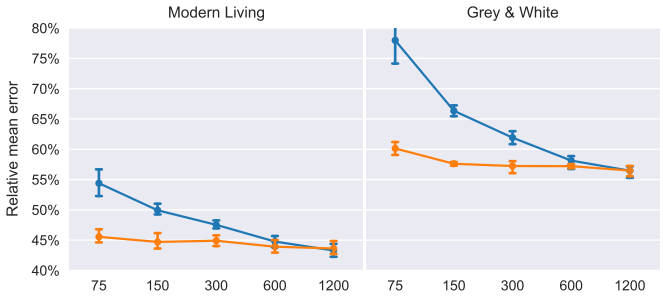


Fig. 11. Comparison to the NFOR denoiser on the public Tungsten dataset. We plot the DSSIM error relative to NFOR when: 1) training a Tungsten-aware encoder for a pre-trained network with frozen weights (orange line, the original network was trained using Moana and Cars data), and 2) training a Tungsten-specific network from scratch (blue line). We used varying training set sizes (horizontal axis), and averaged the errors over sampling rates of 32 to 256 spp. For smaller training sets, training a Tungsten-aware encoder for an existing network gives better results than training from scratch. Overall, we get more robust results than NFOR in both cases. See Figure 10 for a visual comparison of the reconstructions.

7.1 Source-aware encoder

Table 2 shows denoising quality on the Hyperion-rendered (Moana, Olaf) and RenderMan-rendered (Cars, Coco) datasets, when trained either using only the Moana or Cars training set, or using their union. The network performs poorly on the RenderMan sets when trained only with the Hyperion data, and vice versa, suggesting a poor generalization across rendering engines. However, when trained using datasets from both renderers—Moana and Cars—the network can robustly handle both sources, including content that it has not experienced (Olaf, Coco). The combined training yields marginally better results than networks specialized for a particular renderer, presumably thanks to its increased training set size. With a dedicated source-aware encoder for each renderer (fourth line), the first few network layers are allocated to source-specific processing. This enables lightweight adaptation to new renderers and/or feature sets by training a new frontend only while keeping all other weights fixed. In Figures 10 and 11, we investigate this adaptation, and show the performance as a function of the training-set size. We conducted two experiments: 1) training a Tungsten-aware encoder for a pre-trained network with frozen weights (which used the combined Moana-Cars training set), and 2) training the network from scratch using a separate Tungsten training set. We used

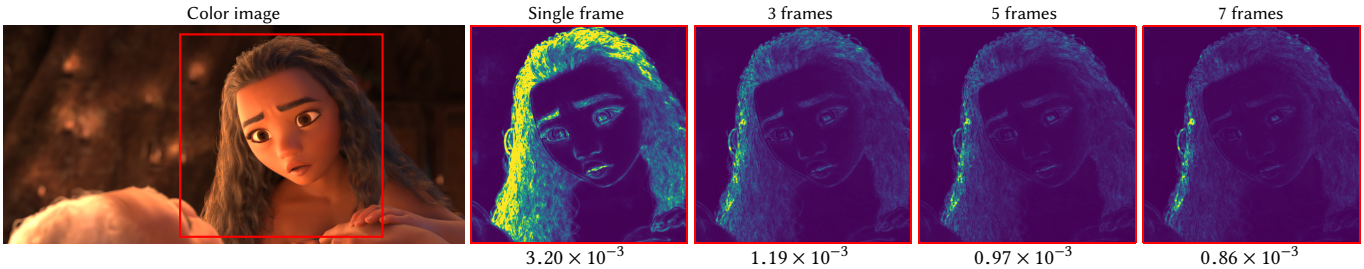


Fig. 12. Temporal stability. Crops show the mean DSSIM between pairs of adjacent frames over a sequence with 24 renders with different random-number seeds of the same scene. Bright colors correspond to high temporal instability. Reported numbers are averages over the image. © Disney.

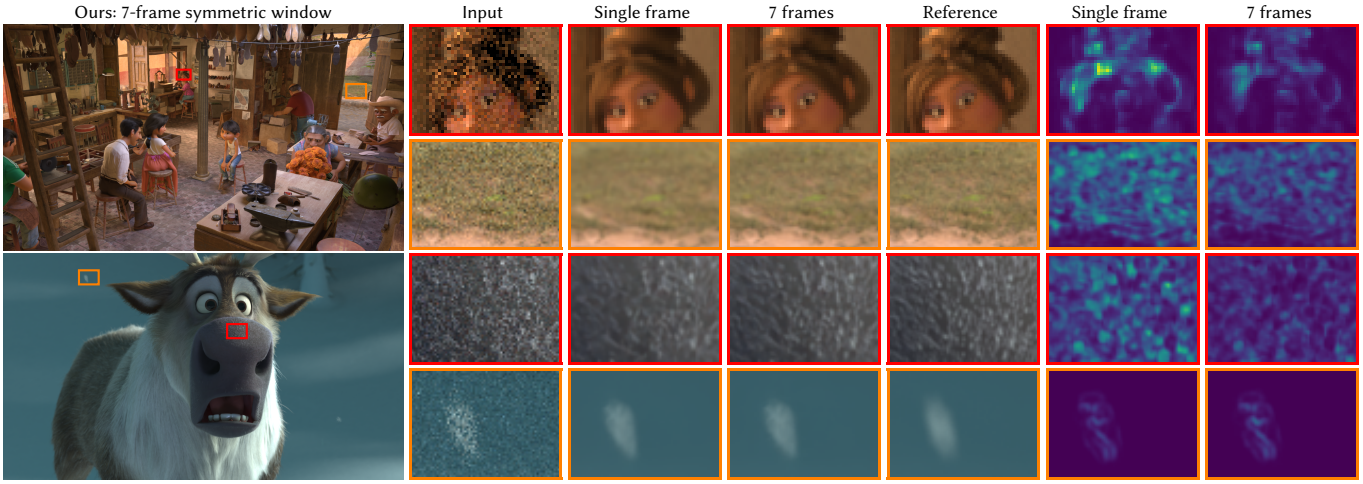


Fig. 13. Temporal denoising performance. Considering larger temporal windows helps not only with temporal flickering (see Figure 12), but can also increase the quality of the reconstruction. The reconstruction error (DSSIM, right-most two columns) is reduced for largely static content (top three rows of crops), and stable for moving content (bottom row of crops). A quantitative analysis of the performance is presented in Figure 15. © Disney / Pixar, © Disney.

various subsets of the Tungsten training data (from only 75, up to all 1200), to evaluate the network performance as the volume of training data increases. The performance values used in Figure 11 are relative to the NFOR denoiser [Bitterli et al. 2016]. For small sets, training a Tungsten-aware encoder yields better results than training from scratch, but as the training dataset size increases, training from scratch eventually yields similar results. While, ideally, one would combine the full Tungsten training set with the Moana and Cars ones to achieve best performance, our experiment shows that source-aware encoders provide an inexpensive means to adapt trained networks to new content (and renderer) with a relatively small amount of training data. For a visual comparison of the reconstructions, please see Figure 10.

7.2 Temporal Denoiser

One of the main goals of our denoiser is to improve temporal coherence and reduce flickering in denoised sequences. We demonstrate the increased stability of the temporal denoiser in Figure 12, by visualizing how the average difference between denoised adjacent frames in a static sequence decreases as we increase the temporal window; please see our accompanying video for more temporal stability results on production sequences.

While the main goal of our temporal denoiser is to alleviate flickering artifacts, it has the added benefit of improving detail reconstruction in mostly static regions, as we illustrate in Figure 13.

Our temporal architecture uses the proposed temporal combiner, which predicts reconstruction kernels for all frames in the temporal window at once. To compare our design to the recurrent architecture used by Chaitanya et al. [2017], we replaced the temporal combiner in our modular architecture with a recurrent combiner that predicts denoised colors directly. We denote this approach R-DP. To train the R-DP, we pretrained a spatial-feature extractor (two weeks), fixed its weights, and used its last feature layer as the input to the recurrent combiner. The spatial-feature extractor was trained with a direct-predicting backend to maximize the R-DP performance.

In Figure 14, we compare our proposed kernel-predicting temporal combiner (Ours) to the direct-predicting recurrent combiner (R-DP), and the NFOR denoiser. R-DP is designed to use only previous frames, whereas NFOR can handle symmetric temporal windows; we therefore show results for our kernel-predicting combiner in both configurations. On our test data, R-DP results suffer from over-blurring, as well as brightness- and color-shift artifacts. NFOR offers sharper results, but tends to leave residual noise. Overall our kernel-predicting combiner offers consistently more robust results, that are both sharper and with fewer residual noise artifacts.

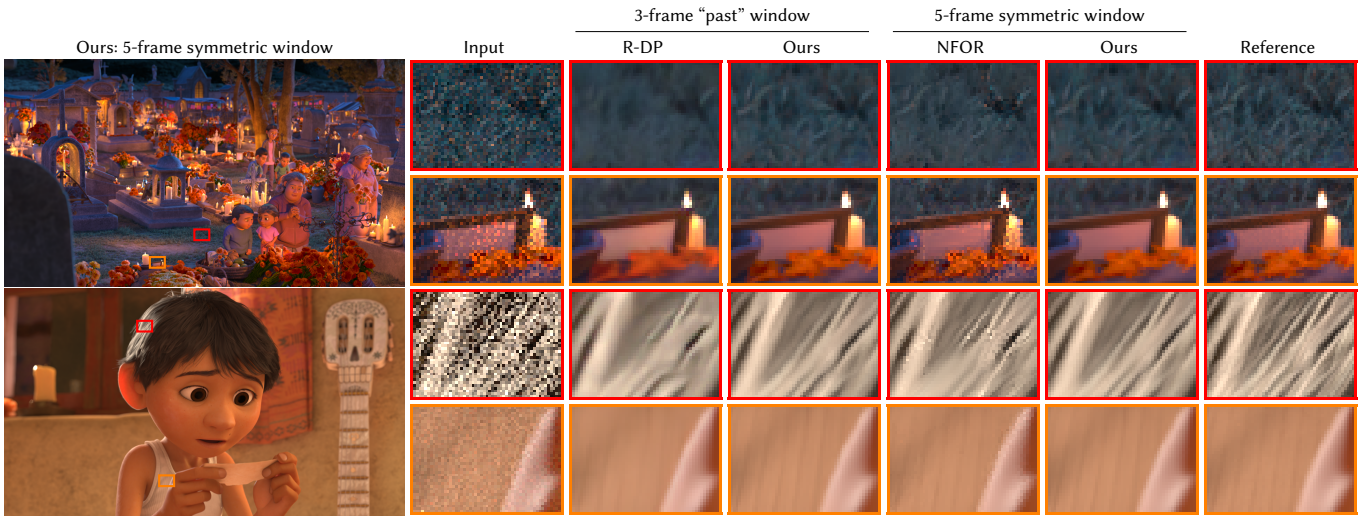


Fig. 14. Comparison of temporal denoisers. We compare reconstruction results obtained using a direct-predicting recurrent network (R-DP) to our kernel-predicting temporal combiner, both using only information from *past* 3 frames in the sequence and trained under equivalent conditions. R-DP suffers from significant brightness, color-shift, and over-blurring issues compared to our architecture. When comparing to the NFOR denoiser [Bitterli et al. 2016], we use a symmetric temporal window. The NFOR results are sharp overall, but suffer from residual noise artifacts. Our network consistently produces smooth results while preserving details. See Figure 15 for numerical comparison using various metrics on our full test set for all denoisers. © Disney / Pixar.

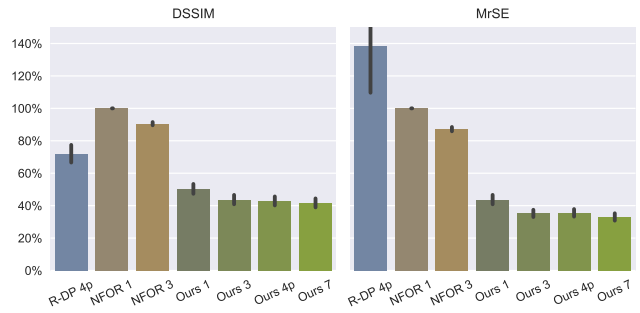


Fig. 15. Error averaged over the Olaf and Coco test sets relative to the NFOR single-frame denoiser, for two evaluation metrics (lower is better). The recurrent direct-predicting combiner 'R-DP 4p' and 'Ours 4p' operate on past four frames. All other denoisers use symmetric temporal windows indicated by the number in their name.

Figure 15 shows a quantitative comparison between all methods on two evaluation metrics, relative to the performance of a single-frame NFOR baseline, averaged over the Olaf and Coco test sets.

7.3 Multi-scale architecture

Residual low-frequency noise artifacts, blotches, are sometimes visually distracting. As illustrated in Figure 16, network depth is a dominant factor. As we increase the network depth, low-frequency blotches gradually vanish, even though the kernel size is fixed. Effectively, with 24 residual blocks, the network leverages information gathered over a 97×97 pixels footprint when predicting the 21×21 kernel, which explains the improved handling of low frequencies. Using our proposed multi-scale kernel-predicting architecture, we can drastically reduce low-frequency artifacts, achieving visually pleasing results with a shallow 6-block network.

7.4 Asymmetric Loss

We used the asymmetric loss to fine-tune a trained network to enable user control over the variance–bias trade off. We illustrate this in Figure 17, which features denoising results with an increasingly asymmetric loss. Increasing λ allows the network to perfer solutions that retain some noise instead of enforcing an overly smooth result. Leaving a small amount of residual noise helps preserving input details. See the accompanying video for a detail comparison.

In Figure 18, we compare the noise produced by the asymmetric loss to a naive approach of adding the input noise: we simply blend between the noisy input and a denoised image obtained using a network optimized for the standard symmetric loss. The blending weight β was tuned to minimize the mean squared error between the blended image and the result obtained with the asymmetric loss. The naive way of "blending in" the input noise preserves fireflies and the final image inherits the heavy-tail noise distribution of the input. In contrast, the asymmetric loss unifies the noise and produces a visually more uniform noise which increases in regions of low confidence and decreases in easy-to-denoise areas.

7.5 Adaptive Sampling

In Figure 19, we compare the reconstruction accuracy using uniform and adaptive sampling, according to both MrSE and DSSIM. We distributed samples using the SMAPE loss predicted by our network, which resulted in sample distributions that capture the general distribution obtained using the ground-truth loss, computed using the reference image. In the worst case (bottom row), the adaptive result was slightly worse according to DSSIM, but still offered an improvement according to MrSE. In the best case, both the DSSIM and MrSE losses were reduced by approximately 40% over uniform sampling.

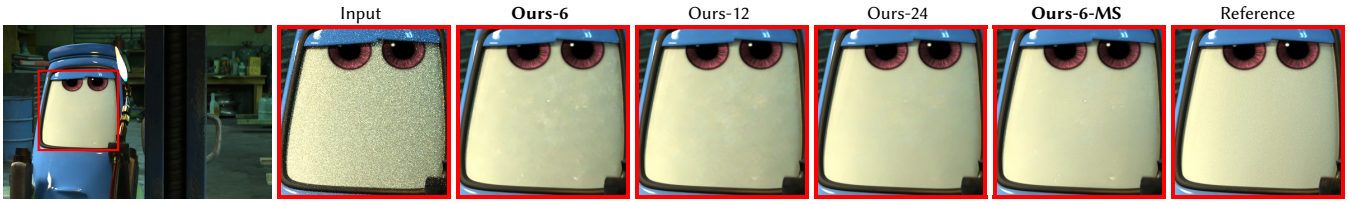


Fig. 16. Multi-scale reconstruction. The occurrence of residual low-frequency noise is dominated by the network depth, not the kernel size. Blotches visible in the reconstruction using our single-frame network with 6 residual blocks (Ours-6) gradually vanish as we increase the network depth (Ours-12, Ours-24), all of which use a 21×21 kernel prediction. Our multi-scale architecture (Ours-6-MS) can yield smooth result even with a shallow network, enabling good results in memory- or run-time-constrained environments. © Disney / Pixar.

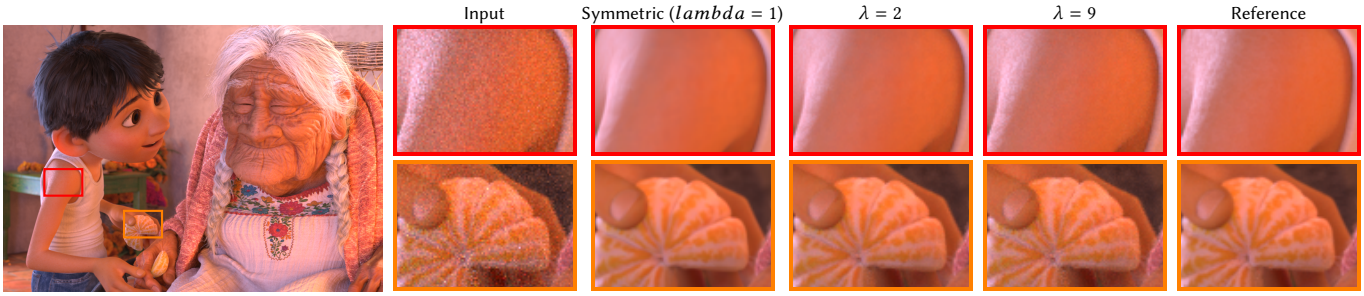


Fig. 17. Asymmetric loss. The asymmetry parameter λ gives the user fine-grained control over the denoiser variance–bias trade-off. An increased slope parameter λ results in a more conservative filter, which better preserves details at the expense of an increased level of residual noise. © Disney / Pixar.

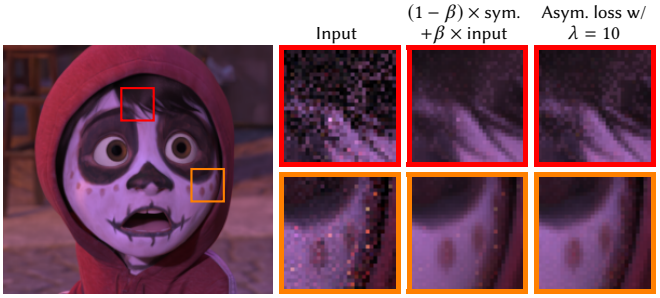


Fig. 18. Asymmetric loss. A comparison of two approaches for reducing blur (bias). The insets in the middle column show a blend between the input (left insets) and a denoised image produced w/ a symmetric loss. The right insets show results w/ an asymmetric loss. Although images are similar on average, the asymmetric leaves noise with an aesthetically more pleasing distribution. © Disney / Pixar.

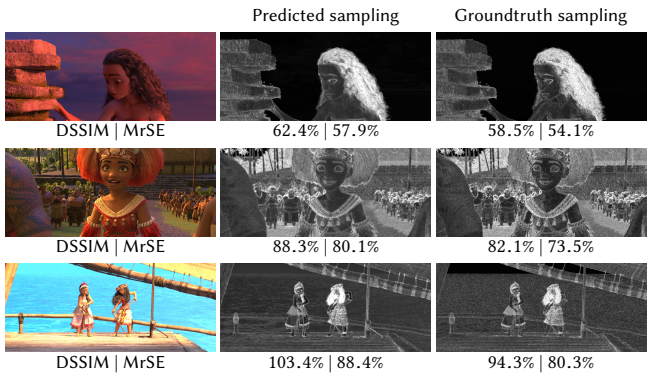


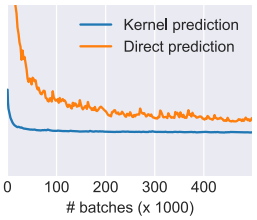
Fig. 19. Sampling maps. We denoise 11 scenes with adaptive sampling using the error-predicting network for SMAPE and visualize the results for the best (top row) median (middle row), and worst (bottom row) error relative to uniform sampling, according to DSSIM. The reconstruction shown (left) uses the predicted sampling map (center), which correctly captures the distribution of the ground-truth sampling map (right). The relative reconstruction error relative to uniform sampling, when using the predicted and ground-truth maps, are given under each (lower is better). © Disney.

7.6 Runtime Cost

The denoising time of a video sequence of resolution 1920×804 with the 7-frame temporal denoiser with 21×21 kernel prediction is 10.2s per frame. This time can be broken down in the cost of the spatial-feature extractor (3.0s per frame), running the temporal combiner (2s per frame), and kernel prediction (5.2s per frame). These numbers are averages over 100 experiments and are recorded with an Nvidia Titan X (Pascal) GPU. The frames are denoised in 8 overlapping spatial tiles of size 544×466 to fit the GPU memory.

8 ANALYSIS OF KPCN

Kernel prediction has been observed to converge faster than direct prediction [Bako et al. 2017; Niklaus et al. 2017; Vogels 2016]. The convergence plot on the right compares two independently trained networks that differ only in the reconstruction approach. The kernel-predicting network converges faster with greater stability. In what follows, we provide a theoretical justification for these empirical observations.



We analyze and compare the convergence behavior of a shallow, convex analogue of KPCN against its direct prediction variant. In Appendix A, we demonstrate that kernel prediction is implicitly performing *mirror descent* [Beck and Teboulle 2003] on a function constrained on the probability simplex with an entropic regularizer (see Bubeck [2015]; Shalev-Shwartz et al. [2012] for an in-depth treatment of mirror descent). We can then leverage established results in convex optimization to show that optimizing the kernel prediction problem enjoys an exponential improvement in convergence speed over otherwise equivalent direct prediction methods with respect to the dimensionality of the parameter space.

Since the structure of the KPCN and DPCN networks proposed by Bako et al. [2017] differ only in the final layer, we consider the following "shallow", convex analogue of KPCN. Given a set of data points $\mathbf{x}^n \in \mathbb{R}^d$ and corresponding targets $y^n \in \mathbb{R}$ for $n = 1, \ldots, N$, we aim to solve

$$\underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\, \ell(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\, \frac{1}{N}\sum_{n=1}^{N} f_n(y^n - \boldsymbol{\theta}^\top \mathbf{x}^n), \tag{9}$$

where f_n is a convex loss function (e.g. squared or absolute error). For direct prediction, the parameter space is $\boldsymbol{\Theta} \subset \mathbb{R}^d$, a compact convex subset of $\mathbb{R}^d$. For kernel prediction the parameter space $\boldsymbol{\Theta} = \Delta_d = \{\boldsymbol{\theta} \in \mathbb{R}^d_+ : \sum_{j=1}^d \theta_j = 1\}$ is restricted to the probability simplex.

The following proposition characterises the difference in rate of convergence between kernel and direct prediction. Specifically it compares the loss evaluated on the averaged sequence of solutions after $T \geq 1$ update steps, $\ell\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta}^{(t)}\right)$ with the loss at the optimum, $\ell(\boldsymbol{\theta}^*)$. This is exactly the setting in which ubiquitously used *adaptive* convex optimization algorithms (e.g. AdaGrad [Duchi et al. 2011] and Adam [Kingma and Ba 2014]) have been analyzed. Recently, Balduzzi et al. [2017] used this framework to obtain convergence guarantees for a more general class of non-convex, non-smooth deep neural networks (including convolutional networks with ReLU non-linearities).

Proposition 8.1. *Define* $E_{Kernel} \triangleq \ell\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta}^{(t)}\right) - \ell(\boldsymbol{\theta}^*)$ *as the suboptimality of the solution to* (9) *with* $\boldsymbol{\Theta} = \Delta_d$ *after* T *iterations Similarly, define* E_{Direct} *as the suboptimality of the solution to the same regression problem where* $\boldsymbol{\Theta} \subset \mathbf{R}^d$. *So we have*

$$E_{Kernel} = O\left(\sqrt{\frac{\log d}{T}}\right), \quad \textit{and} \quad E_{Direct} = O\left(\sqrt{\frac{d}{T}}\right).$$

The proof is provided in Appendix A. Proposition 8.1 states that kernel prediction achieves *exponentially faster convergence*—in terms of the dimension of the parameter space, d—than direct prediction. This analysis of the simplest, convex analogue of kernel prediction goes some way towards explaining the large empirical improvement in convergence speed and stability of KPCN over its direct prediction variants.

9 DISCUSSION AND FUTURE WORK

In this section, we elaborate on the motivations and implications of some of our design choices, as well as outline ongoing experiments with alternative architectures.

9.1 Multi-scale Reconstruction

Relative to the computational cost, our multiscale approach provides only marginal improvements. This is because—at higher SPP inputs particularly—the network on which it is based (see Figure 4) already performs well in preventing low-frequency artifacts and blotches. However, we do observe a noticeable benefit at lower sampling rates or for networks with fewer parameters.

Kernel Size. Recent experiments indicate that the kernel size for the final reconstruction can be reduced to 5×5, while still handling low-frequency noise well, using our multi-scale architecture.

Relation to U-Net. An alternative implementation of our multi-scale reconstruction could be to incorporate the U-net architecture [Chaitanya et al. 2017; Ronneberger et al. 2015], which naturally decomposes the *feature representation*—rather than the image—at multiple scales, by having a kernel prediction at the end of each U-Net scale. Ongoing experiments indicate this yields similar reconstruction quality, while providing computational benefits.

9.2 Training Modules in Parts

While training modules in parts offers some compelling advantages, such as only training a new source-aware encoder to support an additional data source, it is not a requirement: the whole network can in theory be trained end-to-end. The main motivation for training our framework in parts is the complexity of its modules (in terms of number of residual blocks and bandwidth, i.e. convolutional kernels per block), which is such that it cannot fit in memory. We have successfully experimented with end-to-end training of a lighter network configuration—including the source-aware encoder, spatial feature extractor, temporal combiner and scale-compositing modules. This lighter network configuration comes at a minor loss of quality, presumably because of the reduced modeling power. We have, however, not yet explored whether training in parts compromises performance compared to end-to-end training.

9.3 Asymmetric Loss

Our asymmetric loss functions use an inherent notion of uncertainty based on Bayesian decision theory to adapt the level of retained residual noise. A fully Bayesian approach could be considered, where the network would maintain a posterior distribution over the output space which could then also be used to guide adaptive sampling.

10 CONCLUSION

We have presented a modular denoising architecture that extends kernel-predicting networks [Bako et al. 2017; Vogels 2016], enabling temporal and multiscale filtering. We have shown the theoretical benefit in terms of convergence speed of kernel prediction over direct prediction which supports our empirical observations and further justifies its use in our modular denoising architecture.

We proposed source-aware encoders which are able to robustly handle diverse data from three different rendering systems over a wide range of sampling rates. Our temporal approach extracts and combines feature representations from neighboring frames rather than building a temporal context using recurrent connections. As such it requires fewer reference images (relative to the size of the

training set) for training. We evaluated our system by comparing its performance to recently published methods demonstrating consistent state-of-the-art results across all test data.

We also propose an asymmetric loss function which offers user control on the denoiser variance–bias tradeoff. We provide a decision-theoretic justification which posits that, when uncertain, the network will choose to retain residual noise rather than over-blurring. We empirically confirm that the asymmetric loss retains subtle details but crucially does not simply increase the noise level uniformly.

11 ACKNOWLEDGEMENTS

We thank the following Blendswap artists for creating scenes in the Tungsten dataset: Jay-Artist, Mareck, MrChimp2313, nacimus, NovaZeeke, SlykDrako, thecali, and Wig42.

The denoiser was trained and tested on production imagery but the results were not part of the released productions.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. (2015). http://tensorflow.org/

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *IEEE Conference on Computer Vision and Pattern Recognition* (June 27–30). 39–48.

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. Graphics (Proc. SIGGRAPH)* 36, 4, Article 97 (July 2017), 14 pages.

David Balduzzi, Brian McWilliams, and Tony Butler-Yeoman. 2017. Neural Taylor approximations: Convergence and exploration in rectifier networks. In *Proc. 34th International Conference on Machine Learning (Proc. Machine Learning Research)*, Vol. 70. PMLR, Sydney, Australia, 351–360.

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.

Amir Beck and Marc Teboulle. 2003. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters* 31, 3 (2003), 167–175.

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.

Mark R. Bolin and Gary W. Meyer. 1998. A perceptually based adaptive sampling algorithm. In *Proc. 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM Press, New York, NY, USA, 299–309.

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.

Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. 2008. Nonlocal image and movie denoising. *International Journal of Computer Vision* 76, 2 (01 Feb 2008), 123–139.

Sébastien Bubeck. 2015. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning* 8, 3–4 (2015), 231–358.

Harold Christopher Burger, Christian J. Schuler, and Stefan Harmeling. 2012. Image denoising: Can plain neural networks compete with BM3D?. In *IEEE Conference on Computer Vision and Pattern Recognition* (June 16–21). IEEE Computer Society, 2392–2399.

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graphics (Proc. SIGGRAPH)* 36, 4, Article 98 (July 2017), 12 pages.

Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. 2006. Image denoising with block-matching and 3D filtering. In *Proc. SPIE*, Vol. 6064. 606414–606414–12.

Mauricio Delbracio, Pablo Musé, Antoni Buades, Julien Chauvier, Nicholas Phelps, and Jean-Michel Morel. 2014. Boosting Monte Carlo rendering by ray histogram fusion. *ACM Trans. Graphics* 33, 1, Article 8 (Feb. 2014), 15 pages.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proc. 13th International Conference on Artificial Intelligence and Statistics* (May 13–15). JMLR.org, 249–256.

Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, Kevin Dale, Greg Humphreys, Matthias Zwicker, and Henrik Wann Jensen. 2008. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graphics* 27, 3, Article 33 (Aug. 2008), 10 pages.

Moritz Hardt, Ben Recht, and Yoram Singer. 2016. Train faster, generalize better: Stability of stochastic gradient descent. In *Proc. 33nd International Conference on Machine Learning* (June 19–24). JMLR.org, 1225–1234.

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* (June 27–30). IEEE Computer Society, 770–778.

Viren Jain and H. Sebastian Seung. 2008. Natural image denoising with convolutional networks. In *Advances in Neural Information Processing Systems 21* (December 8–11). Curran Associates, Inc., 769–776.

Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc Van Gool. 2016. Dynamic filter networks. In *Advances in Neural Information Processing Systems 29* (December 5–10). Curran Associates, Inc., 667–675.

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A machine learning approach for filtering Monte Carlo noise. *ACM Trans. Graphics (Proc. SIGGRAPH)* 34, 4, Article 122 (July 2015), 12 pages.

Alexander Keller, Luca Fascione, Marcos Fajardo, Iliyan Georgiev, Per H. Christensen, Johannes Hanika, Christian Eisenacher, and Gregory Nichols. 2015. The path tracing revolution in the movie industry. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH '15)*. ACM Press, 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). arXiv:1412.6980

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. In *Proc. National Academy of Sciences*, Vol. 114. National Academy of Sciences, 3521–3526.

Gabriel Krummenacher, Brian McWilliams, Yannic Kilcher, Joachim M. Buhmann, and Nicolai Meinshausen. 2016. Scalable adaptive stochastic optimization using random projections. In *Advances in Neural Information Processing Systems 29* (December 5–10). Curran Associates, Inc., 1750–1758.

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

Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012. SURE-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 31, 6, Article 194 (Nov. 2012), 9 pages.

Aurélien Lucchi, Brian McWilliams, and Thomas Hofmann. 2015. A variance reduced stochastic Newton method. *CoRR* abs/1503.08316 (2015). arXiv:1503.08316

Michael D. McCool. 1999. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Trans. Graphics* 18, 2 (April 1999), 171–194.

Ben Mildenhall, Jonathan T. Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. 2017. Burst denoising with kernel prediction networks. *CoRR* abs/1712.02327 (2017). arXiv:1712.02327

Don P. Mitchell. 1987. Generating antialiased images at low sampling densities. In *Proc. 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. ACM Press, New York, NY, USA, 65–72.

Bochang Moon, Nathan Carr, and Sung-Eui Yoon. 2014. Adaptive rendering based on weighted local regression. *ACM Trans. Graphics* 33, 5, Article 170 (Sept. 2014), 14 pages.

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. *ACM Trans. Graphics (Proc. SIGGRAPH)* 35, 4, Article 40 (July 2016), 10 pages.

Kevin P. Murphy. 2012. *Machine learning - a probabilistic perspective*. MIT Press.

Simon Niklaus, Long Mai, and Feng Liu. 2017. Video frame interpolation via adaptive convolution. In *IEEE Conference on Computer Vision and Pattern Recognition* (July 21–26). IEEE Computer Society, 2270–2279.

Ryan S. Overbeck, Craig Donner, and Ravi Ramamoorthi. 2009. Adaptive wavelet rendering. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 28, 5, Article 140 (Dec. 2009), 12 pages.

Mahesh Ramasubramanian, Sumanta N. Pattanaik, and Donald P. Greenberg. 1999. A perceptually based physical error metric for realistic image synthesis. In *Proc. 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 73–82.

Garvesh Raskutti and Sayan Mukherjee. 2015. The information geometry of mirror descent. *IEEE Trans. Information Theory* 61, 3 (2015), 1451–1457.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention - 18th International Conf.* (October 5–9). Springer, 234–241.

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2011. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 30, 6, Article 159 (Dec. 2011), 12 pages.

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2012. Adaptive rendering with non-local means filtering. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 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 Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. ACM Press, 131–138.

Pradeep Sen and Soheil Darabi. 2012. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Trans. Graphics* 31, 3, Article 18 (June 2012), 15 pages.

Shai Shalev-Shwartz et al. 2012. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning* 4, 2 (2012), 107–194.

Thijs Vogels. 2016. *Kernel-predicting convolutional networks for denoising Monte Carlo renderings*. Master's thesis. ETH Zürich, Zürich, Switzerland.

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.

Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. 2017. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems 30* (December 4–9). Curran Associates, Inc., 4151–4161.

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. *Computer Graphics Forum (Proc. Eurographics)* 34, 2 (May 2015), 667–681.

A MIRROR DESCENT

In this section we give a basic definition of the mirror descent algorithm [Beck and Teboulle 2003]. We then show the equivalence between gradient descent updates to the shallow kernel prediction problem in (9) and a special case of the mirror descent algorithm. Definition A.1 and Proposition A.3 rely on standard concepts from convex optimization. We refer the reader to Bubeck [2015] for a comprehensive treatment of the subject.

Definition A.1 (Mirror descent). For a convex function $\ell(\boldsymbol{\theta})$, which is L-Lipschitz with respect to an arbitrary norm, $\|\cdot\|$, mirror descent is an iterative algorithm for solving $\operatorname{argmin}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \ell(\boldsymbol{\theta})$. Let the *mirror map* $\Phi : \boldsymbol{\Theta} \to \mathbb{R}$ be a function ρ-strongly convex under $\|\cdot\|$. The mirror descent update at iteration t with a learning rate $\eta > 0$ is given by

$$\mathbf{z}^{(t)} = \mathbf{z}^{(t-1)} - \eta \nabla \ell(\boldsymbol{\theta}^{(t-1)}) \tag{10}$$

$$\boldsymbol{\theta}^{(t)} = \operatorname*{argmax}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \boldsymbol{\theta}^\top \mathbf{z}^{(t)} - \Phi(\boldsymbol{\theta}) \tag{11}$$

Proposition A.2. *For a convex function in $\boldsymbol{\theta}$, $\ell(\boldsymbol{\theta})$, performing gradient descent kernel prediction updates is equivalent to mirror descent updates on* (9) *with entropic regularizer $\Phi(\boldsymbol{\theta}) = \sum_{j=1}^{d} \theta_j \log \theta_j$ where the parameter space $\boldsymbol{\Theta} = \Delta_d$ is restricted to the probability simplex.*

Proof. The solution to (11) under the conditions in the proposition is $\boldsymbol{\theta}^{(t)} = \operatorname{softmax}(\mathbf{z}^{(t)})$. Recall from Section 3.1, for (9) kernel prediction performs the following update

$$\mathbf{z}^{(t)} = \mathbf{z}^{(t-1)} - \eta \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{(t-1)}), \quad \boldsymbol{\theta}^{(t)} = \operatorname{softmax}(\mathbf{z}^{(t)}).$$

This is precisely an update iteration of the *normalized exponential gradient descent* algorithm, a special case of mirror descent [Shalev-Shwartz et al. 2012]. □

Proposition A.3 (Theorem 4.2 from Bubeck [2015]). *Define the radius $R^2 = \sup_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \Phi(\boldsymbol{\theta}) - \Phi(\boldsymbol{\theta}^{(1)})$ where $\boldsymbol{\theta}^{(1)} \in \operatorname{argmin}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \Phi(\boldsymbol{\theta})$ and $\boldsymbol{\Theta}$ is a closed convex set. Mirror descent as described in Proposition A.1 with a learning rate, $\eta = \frac{R}{L}\sqrt{\frac{2\rho}{T}}$ achieves a convergence rate*

$$E \triangleq \ell\left(\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\theta}^{(t)}\right) - \ell(\boldsymbol{\theta}^*) \le RL\sqrt{\frac{2}{\rho T}}.$$

We are now ready to provide the proof of Proposition 8.1.

Proof of Proposition 8.1. From Proposition A.2 we have that performing kernel prediction updates is equivalent to mirror descent with $\Phi(\boldsymbol{\theta}) = \sum_{j=1}^{d} \theta_j \log \theta_j$. Φ is 1-strongly convex with respect to $\|\cdot\|_1$, so $\rho = 1$. We also have that $\boldsymbol{\Theta} = \Delta_d$ so $R^2 = \log d$.

Performing direct prediction updates using stochastic gradient descent is equivalent to performing mirror descent updates with the mirror map $\Phi(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2$ which is $\rho = 1$-strongly in $\|\cdot\|_2$. In this setting, $\boldsymbol{\Theta} \subset \mathbb{R}^d$ so the radius $R^2 = d$. Plugging these into Proposition A.3 yields the result. □

Discussion. In the convex setting, the optimal convergence rate for stochastic first-order optimization methods (such as SGD) is $O(1/\sqrt{T})$ and cannot be improved without further assumptions (e.g. using second order information, or variance reduction [Lucchi et al. 2015]).

However, popular methods for optimizing deep networks such as AdaGrad and Adam perform *adaptive* updates which can greatly improve convergence speed [Duchi et al. 2011]. However, this improvement is typically only realized under restrictive assumptions (e.g. data sparsity or low-rankness [Krummenacher et al. 2016]). Practically, adaptive optimization methods often show no improvement over well-tuned SGD [Wilson et al. 2017].

In contrast, mirror descent with entropic regularization shows a theoretical exponential improvement over SGD regardless of the data distribution. In fact, mirror descent can be shown to be a second-order algorithm [Raskutti and Mukherjee 2015]. This goes some way to explaining why—even though both KPCN and DPCN employ the Adam optimizer—KPCN shows a substantial improvement in training speed and convergence stability over DPCN.

This result has implications beyond just the training speed of KPCN. Recent work by Hardt et al. [2016] has shown that in both convex and non-convex settings optimizers which converge faster exhibit better generalization properties.

Kernel prediction has been proposed concurrently as *adaptive convolution* for video interpolation [Niklaus et al. 2017]. It should be noted that the analysis developed here also applies. However, Mildenhall et al. [2017] modified the kernel prediction output to remove the softmax. This breaks the equivalence with mirror descent and so the theoretical improvement no longer applies.