Neural Photo-Finishing

ETHAN TSENG* and YUXUAN ZHANG, Princeton University, USA
LARS JEBE, XUANER ZHANG, ZHIHAO XIA, and YIFEI FAN, Adobe, USA
FELIX HEIDE†, Princeton University, USA
JIAWEN CHEN†, Adobe, USA

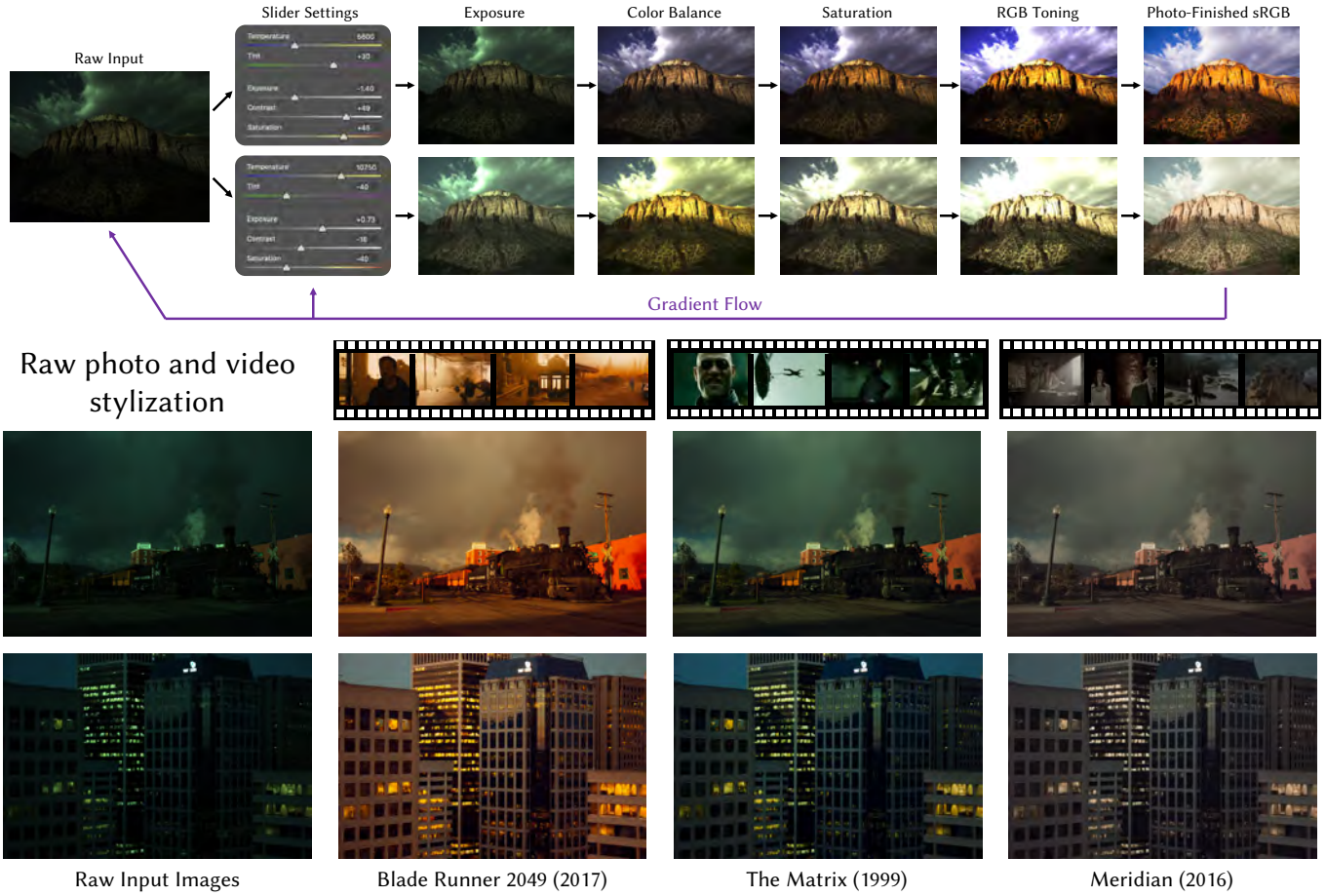

Fig. 1. We propose Neural Photo-Finishing, an end-to-end differentiable pipeline for rendering sRGB images from raw inputs controlled by meaningful parameters. We accurately model a commercial raw processing pipeline (Adobe Camera Raw) with a sequence of neural networks, enabling partial derivatives to be evaluated anywhere in the pipeline with respect to any input or upstream parameter. Unlike previous methods, we supervise our model using intermediary program tap-outs (top). We demonstrate applications ranging from raw photo and video style transfer (bottom), slider regression for commercial camera ISPs, improved demosaicking, and adversarial attacks on image classifiers.

*Part of this work was done during an internship at Adobe.
†Denotes equal contribution between Princeton and Adobe.

Authors' addresses: Ethan Tseng, eftseng@princeton.edu; Yuxuan Zhang, yz8614@princeton.edu, Princeton University, USA; Lars Jebe, jebe@adobe.com; Xuaner Zhang, cecilia77@berkeley.edu; Zhihao Xia, zxia@adobe.com; Yifei Fan, yifan@adobe.com, Adobe, USA; Felix Heide, fheide@princeton.edu, Princeton University, USA; Jiawen Chen, jiawen@adobe.com, Adobe, USA.

0730-0301/2022/12-ART238
https://doi.org/10.1145/3550454.3555526

Image processing pipelines are ubiquitous and we rely on them either directly, by filtering or adjusting an image post-capture, or indirectly, as image signal processing (ISP) pipelines on broadly deployed camera systems. Used by artists, photographers, system engineers, and for downstream vision tasks, traditional image processing pipelines feature complex algorithmic branches developed over decades. Recently, image-to-image networks have made

great strides in image processing, style transfer, and semantic understanding. The differentiable nature of these networks allows them to fit a large corpus of data; however, they do not allow for intuitive, fine-grained controls that photographers find in modern photo-finishing tools.

This work closes that gap and presents an approach to making complex photo-finishing pipelines differentiable, allowing legacy algorithms to be trained akin to neural networks using first-order optimization methods. By concatenating tailored network proxy models of individual processing steps (e.g. white-balance, tone-mapping, color tuning), we can model a non-differentiable reference image finishing pipeline more faithfully than existing proxy image-to-image network models. We validate the method for several diverse applications, including photo and video style transfer, slider regression for commercial camera ISPs, photography-driven neural demosaicking, and adversarial photo-editing.

CCS Concepts: • **Computing methodologies → Computational photography**.

Additional Key Words and Phrases: image processing, photo-finishing, raw processing

ACM Reference Format:
Ethan Tseng, Yuxuan Zhang, Lars Jebe, Xuaner Zhang, Zhihao Xia, Yifei Fan, Felix Heide, and Jiawen Chen. 2022. Neural Photo-Finishing. *ACM Trans. Graph.* 41, 6, Article 238 (December 2022), 15 pages. https://doi.org/10.1145/3550454.3555526

1 INTRODUCTION

Almost every modern photography workflow relies on dedicated image processing software that converts camera sensor measurements into finished photos for human viewing. Drawing inspiration from the traditional process of developing film in a darkroom, today's photo-finishing software, such as Adobe Camera Raw [Adobe 2022a], Adobe Lightroom [Adobe 2022c], Darktable [Darktable 2022], or PicsArt [PicsArt 2022], allows users to develop their digital raw photos by manipulating a suite of intuitive sliders, including exposure, saturation, contrast, and tone [Hu et al. 2018]. Millions of users, from professional creatives to casual photographers, rely on photo-finishing pipelines for artistic rendering.

The conventional practice in existing photo-finishing workflows is for the artist to interactively tweak sliders until they are satisfied with the look. However, this is a time-consuming task to perform for each individual photo due to the vast search space. Furthermore, user fatigue can result in inconsistent results across a suite of images if a consistent look is desired. Although preset "filter" settings and automatic slider setting tools have been proposed [Hu et al. 2018], these existing automatic tuning approaches act as "one-size fits all" functions and cannot be easily adapted in an image-dependent way to different styles according to the preferences of a professional photographer. Consequently, users often deviate from automatic slider settings. Smartphone camera applications such as iPhone's native camera, Snapchat, and Instagram all apply preset slider filters in the sRGB space. However, these functions inherit the same problem in that they cannot be easily tuned to different photographic styles. Moreover, the image transformations are not applied to the raw data which limits their expressiveness. For example, an overexposed sky in a high dynamic range scene cannot be recovered once clipped by the finishing pipeline.

Researchers have proposed several automatic photo-finishing techniques; however, they all have limitations that prevent widespread adoption. Early machine learning approaches to style transfer [Gatys et al. 2016] operate entirely on photo-finished sRGB space, foregoing raw data entirely. These methods need to hallucinate content missing in the sRGB image, such as clipped shadows and highlights, which limits their editing capabilities. Moreover, these approaches need to learn an inverse photo-finishing pipeline to convert sRGB images into synthesized raw data [Afifi and Abuolaim 2021]. Recent stylization techniques that rely on generative adversarial networks [Karras et al. 2019] only allow editing via modifying a latent code–an unintuitive control that entangles contrast, tone, illumination, and color rendering. Latent space editing is less intuitive to a photographer than sliders which have meaning to artists. For example, it is not obvious how one perturbs a latent vector to specifically increase saturation. Researchers have explored using neural networks for individual processing operations in a camera pipeline. However, existing works focus on a few select image processing operations that are pertinent to the application [Gharbi et al. 2016, 2017] or choose to define their own specialized image renderers [Chen et al. 2018, 2017a]. While successful for a given setting, these works do not offer intuitive user control over the photo-finishing process through the suite of parameters found in conventional pipelines. To this end, Hu et al. [2018] propose a reinforcement learning approach for automatic photo-finishing. However, it requires that the imaging pipeline be fully differentiable and is thus limited to a small set of operators. Because of these limitations, mainstream artists and professionals continue to rely on traditional photo-finishers.

Another line of research attempts to incorporate differentiability into existing image processing pipelines. One approach is to extract approximate gradients through finite differences [Chen et al. 2017b]; however, each gradient query requires multiple evaluations of an expensive function. The number of function evaluations quickly scales with the number of parameters, making it computationally impractical for many tasks. Another research direction aims to model image signal processors using neural networks [Chen et al. 2018; Mosleh et al. 2020; Tseng et al. 2019; Yu et al. 2021]; however, these methods fail for photo-finishing pipelines with rich image processing operations. For example, Darktable contains 67 image adjustment operations which are divided into five groups called Basic, Tone, Color, Correction, and Effect. Halide [Li et al. 2018], a domain-specific language for image processing, provides tools for writing differentiable pipelines. Unfortunately, many complex operators found in modern photo-finishing pipelines are not differentiable, restricting automatic optimization methods to slow zeroth-order approaches. Existing methods that propose to approximate these pipelines with differentiable proxies [Tseng et al. 2019; Yu et al. 2021], offer also no alternative. While the use of differentiable proxies itself has been investigated extensively in the field of neural architecture search, we show that existing proxy models cannot accurately represent complex photo finishing pipelines.

In this work we close this gap and introduce a differentiable photo-finishing pipeline that supports a rich variety of operations, including tone-mapping, exposure control, and white balance. In lieu of a single large network that attempts to model photo-finishing

end-to-end, we decompose the pipeline and model each stage with a small set of tailored proxy networks. Each proxy is trained on intermediates tapped out from the reference pipeline. This allows us to simultaneously bypass non-differentiable operations while ensuring high accuracy compared to single-network or neural architecture search methods, that fail to learn the complex and large space of combinatorial operations. We validate the utility of the proposed pipeline with extensive experiments on diverse applications: flexible automatic raw to sRGB stylization, joint demosaicking and denoising, and adversarial photo-editing.

We deliver the following contributions:

- We construct a differentiable photo-finishing pipeline accurately modeling a commercial software renderer.
- To our knowledge, our pipeline is the first differentiable photo-finisher to handle standard raw image metadata, allowing it to handle images from a variety of real cameras.
- We demonstrate how to build a variety of applications that incorporate a differentiable photo-finisher as its core.

Scope and Reproducibility. In this work, we present a method for making complex photo-finishing pipelines differentiable. We focus on Adobe Camera Raw as it is a broadly used production-quality pipeline. This comes with the benefit that the proposed models have the potential to directly be used by a large community of ACR users. At the same time, as it is a commercial software, source code is unavailable and we used a developer tool to serialize per-stage intermediates. To ensure that our method is reproducible by the community, we will follow standard SIGGRAPH practice for systems papers and release datasets for intermediates tapped out from photo-finishing a large raw image collection. We will also release the source code and trained models for our method so that the community can reproduce and build upon our results. We further note that the input-output pairs and models we provide allow others to directly compare to our method.

2 RELATED WORK

We give an overview of work related to the proposed method below.

Photo-Finishing. Professional creatives digitally "develop" photographs starting from raw image formats (e.g., Adobe Digital Negative (DNG) [Adobe 2022b], ProRAW [Apple 2022]) to allow for maximum flexibility and control over the rendering process. Typically, users follow heuristic guidelines [Eismann et al. 2018] but it is difficult for humans to maintain consistency across every image they want to retouch. The MIT-Adobe FiveK dataset [2011], one of first raw datasets for learning artistic style, was created by human retouchers and displays such consistency problems. To automate exposure adjustments, Hu et al. [2018] propose a white-box photo-finisher based on reinforcement learning. However, their approach requires explicit gradients which prevents it from being applied to existing photo-finishers written in traditional frameworks. Bayesian or zeroth-order search [Bergstra and Bengio 2012; Bergstra et al. 2013; Nishimura et al. 2018] for photo-finishing parameter presets quickly become computationally intractable due to the high-dimensional search space. Recently, Tseng et al. [2019] approximates black-box image processing pipelines with neural networks but this approach fails once the ISPs become too complex, which we demonstrate empirically in this work. Halide [Li et al. 2018] offers a differentiable programming framework, but many existing commercial packages are still written in traditional languages and will require effort comparable to a full port to a deep learning framework. DEXTER [Ahmad et al. 2019] attempts to automatically port legacy pipelines to Halide; however, it is currently unable to port all of the core functions necessary for a photo-finishing pipeline. Furthermore, many operators rely on lookup tables and a direct Halide translation will not result in differentiable code. PyNET [Ignatov et al. 2020] learns the raw to sRGB mapping for specific smartphone image signal processors (ISPs) but the learned function cannot be modified by parametric controls. Our method is the first that accurately models a complex commercial photo-finishing pipeline while being differentiable with respect to both parameter settings and input images.

Raw Camera Image Processing. Raw sensor measurements suffer from many sources of degradation including, but not limited to, lens aberrations, color filter subsampling, photon shot noise, channel crosstalk, and read noise. A typical ISP pipeline in a camera aims to reconstruct a high-quality image from degraded measurements with a sequence of modules [Karaimer and Brown 2016], each addressing a specific defect. Cameras that need to produce images in real time rely on specialized hardware to realize these ISP operations [Hegarty et al. 2014; ON Semi MT9P001 2017; Ramanath et al. 2005; Shao et al. 2014; Zhang et al. 2011].

Alternative approaches include optimization-based methods [Heide et al. 2014] that operate orders of magnitude slower than real-time ISPs. Machine learning based methods focus on specific tasks, such as demosaicking [Gharbi et al. 2016], tone mapping [Gharbi et al. 2017], low-light imaging [Abdelhamed et al. 2018; Chen et al. 2018; Diamond et al. 2021], high dynamic range (HDR) reconstruction [Mildenhall et al. 2021; Onzon et al. 2021], and dehazing [Shi et al. 2021]. In contrast to traditional ISP pipelines, these methods require high-end GPUs, do not provide intuitive knobs for image adjustment but, nevertheless, remain attractive in their ability to be tailored to specific tasks.

To bring data-driven parameter optimization to ISP processing, recent approaches have explored camera pipeline optimization directly using conventional ISPs or proxy functions. Existing methods [Mosleh et al. 2020; Nishimura et al. 2018; Tseng et al. 2019] either do not provide accurate gradients or they do not provide interpretable proxy blocks akin to a conventional image processing pipeline. Recently, Conde et al. [2022] propose a learnable dictionary-based ISP that is reversible for sRGB-to-raw mapping. While this pipeline could be used to learn the bidirectional mapping of black-box ISPs, it does not provide interactive controls to finish a raw image with interpretable parameters. ReconfigISP [Yu et al. 2021] utilize neural architecture search to design new ISPs, but with existing ISP blocks or their differentiable approximations. However, they do not model complex commercial photo-finishers with fine controls or integrate with existing raw image metadata. While the use of differentiable proxies itself has been investigated extensively in the field of neural architecture search [Wu et al. 2019; Zhou et al. 2020, 2019], we provide additional experiments to validate

that the specific proxy models and optimization scheme proposed in our work are responsible for successfully representing complex photo-finishing pipelines. Specifically, we confirm in this work that ReconfigISP's overall pipeline, the proxy used in ReconfigISP with our method, and a monolithic proxy [Tseng et al. 2019], all fail to approximate complex photo-finishing pipelines, such as the ones in Adobe Camera Raw. In this work, we tackle this problem by concatenating a sequence of proxy functions, where each is tailored to a corresponding block in a complex photo-editing pipeline.

Neural Style Transfer. A large body of work explores style transfer between images. Image Analogies [Hertzmann et al. 2001] and Image Quilting [Efros and Freeman 2001] are early methods that employ patch-based texture matching. Video color grading is a related area for transferring styles to videos [Bonneel et al. 2013; Design 2022; Du et al. 2021; Hurkman 2010]. Recent methods have leveraged neural networks to perform artistic style transfer [Dumoulin et al. 2017; Gatys et al. 2016; Xia et al. 2020; Yoo et al. 2019; Zhu et al. 2017] and aim to maintain the overall content of an image while incorporating features from another domain (e.g., applying paintbrush overlays). Coincidentally, this research direction has introduced new perceptual losses [Johnson et al. 2016; Zhang et al. 2018], broadly adopted to train generative neural networks by penalizing image features at a higher level than per-pixel differences. Most existing approaches to style transfer [Abdal et al. 2019] that allow image editing within a network's latent space can be both unwieldy and unintuitive to control. Furthermore, these works often still change image content when the user only wanted to perform high-level retouching (e.g., global contrast, white balance). Afifi et al. [Afifi and Brown 2020a,b; Afifi et al. 2019] propose several white balance editing techniques for post-processed images; however, much of the effort is spent on approximately inverting the camera photo-finishing pipeline. Similarly, several works [Afifi et al. 2021; Liu et al. 2020] attempt to recover clipped HDR features by inverting photo-finishing. As a result, despite the advances in style transfer powered by modern machine learning, mainstream artists and consumers still rely on traditional software to finish raw images using their wide range of intuitive controls.

3 PHOTO-FINISHING AND RAW PROCESSING

In this section we describe a mathematical model for a typical photo-finishing pipeline such as those underlying Adobe Camera Raw and Darktable. We then show how to model them with differentiable proxy functions.

3.1 Photo-Finishing Pipeline Model

A conventional photo-finishing pipeline consists of a series of individual processing blocks that are applied sequentially to an input raw image. We formalize this as

$$\mathbf{I}_\mathrm{F} = f_\mathrm{PIPE}\left(\mathbf{I}_\mathrm{R}, \mathbf{S}, \mathbf{M}, \mathbf{H}\right) = f_\mathrm{PIPE}\left(\mathbf{I}_\mathrm{R}, \bigcup_{i=1}^{n} \mathbf{S}_i, \bigcup_{i=1}^{n} \mathbf{M}_i, \bigcup_{i=1}^{n} \mathbf{H}_i\right), \quad (1)$$

where f_PIPE maps the raw image $\mathbf{I}_\mathrm{R}$ to the finished image $\mathbf{I}_\mathrm{F}$ in a color space such as sRGB. The overall pipeline function f_PIPE is parameterized by $\mathbf{S}_i$, which we will refer to as "sliders" in reference to their role in interactive photo-finishing, camera metadata $\mathbf{M}_i$, and cached image statistics $\mathbf{H}_i$. Here, $i \in \{1, \ldots, n\}$ indexes one of the n processing blocks in the pipeline. A typical photo-finisher is a true pipeline; that is, a composition of functions

$$f_\mathrm{PIPE}\left(\mathbf{I}_\mathrm{R}, \mathbf{S}, \mathbf{M}, \mathbf{H}\right) = f_n\left(\ldots f_1\left(\mathbf{I}_\mathrm{R}, \mathbf{S}_1, \mathbf{M}_1, \mathbf{H}_1\right), \mathbf{S}_n, \mathbf{M}_n, \mathbf{H}_n\right), \quad (2)$$

where each f_i represents an individual stage. Each stage maps an input image with resolution $w \times h \times c$ to its corresponding output

$$f_i : \mathbb{R}^{whc} \times \Omega_\mathbf{S}^i \times \Omega_\mathbf{M}^i \times \Omega_\mathbf{H}^i \rightarrow \mathbb{R}^{whc}, \forall i \in \{1 \ldots n\}, \quad (3)$$

where c is the number of channels, and the space $\Omega_\mathbf{S}^i, \Omega_\mathbf{M}^i, \Omega_\mathbf{H}^i$ describe the slider parameter space, metadata space and image statistics input to stage i. Although categorical slider values and binary switches may exist in general image processing pipelines, our model relaxes all parameters as well as image statistics to be real-valued. We allow image metadata to be categorical since computing their gradients is typically not required.

In this work, we model the Adobe Camera Raw (ACR) photo-finishing pipeline, which follows the mathematical formulation above. We illustrate this pipeline in Figure 2 and describe its nine steps here.

(1) *Exposure:* Camera sensors have dynamic sensitivity (ISO) and optical black levels. The exposure stage first normalizes signal levels by using the baseline exposure and black level metadata from $\mathbf{M}_\mathrm{EXP}$. The slider $\mathbf{S}_\mathrm{EXP}$ = {Exposure} then scales the image intensity after normalization.

(2) *Tone Mapping:* This operation compresses a high dynamic range input to a smaller dynamic range. For performance, ACR uses cached histogram statistics from $\mathbf{H}$ to partition the input into highlight and shadow regions, which are affected by the Highlights and Shadows sliders, respectively.

(3) *Color Balance:* Different camera sensors have different per-unit frequency responses to the incoming light. Factory calibration under controlled lighting allows manufacturers to incorporate metadata $\mathbf{M}_\mathrm{WB}$ mapping sensor responses to linear RGB color under known illuminants such as D50 or D65. The manufacturer's "As Shot" white balance, together with user-controlled sliders $\mathbf{S}_\mathrm{WB}$ = {Temperature, Tint}, control the white point and indirectly the 3×3 color correction matrix.

(4) *Saturation*: This operation controls the overall color saturation of the final image. It is controlled by $\mathbf{S}_\mathrm{SATURATION}$ = {Saturation}. The operator can be defined in the HSV, HSL, or YUV color spaces and scales the saturation component of HSV and HSL or the chroma components of YUV.

(5) *Texture:* This operation controls the overall sharpness of the final image. Increasing the $\mathbf{S}_\mathrm{TEXTURE}$ = {Texture} slider corresponds to boosting the local contrast in pixel neighborhoods, resulting in an unsharp mask effect. Decreasing this slider corresponds to blurring the image.

(6) *Color Tables:* ACR lets vendors customize the look of their images with lookup tables defined in HSV space and stored in metadata. These tables are not parameterized by any slider values ($\mathbf{S}_\mathrm{CT} = \emptyset$) but are still an essential part of the pipeline.

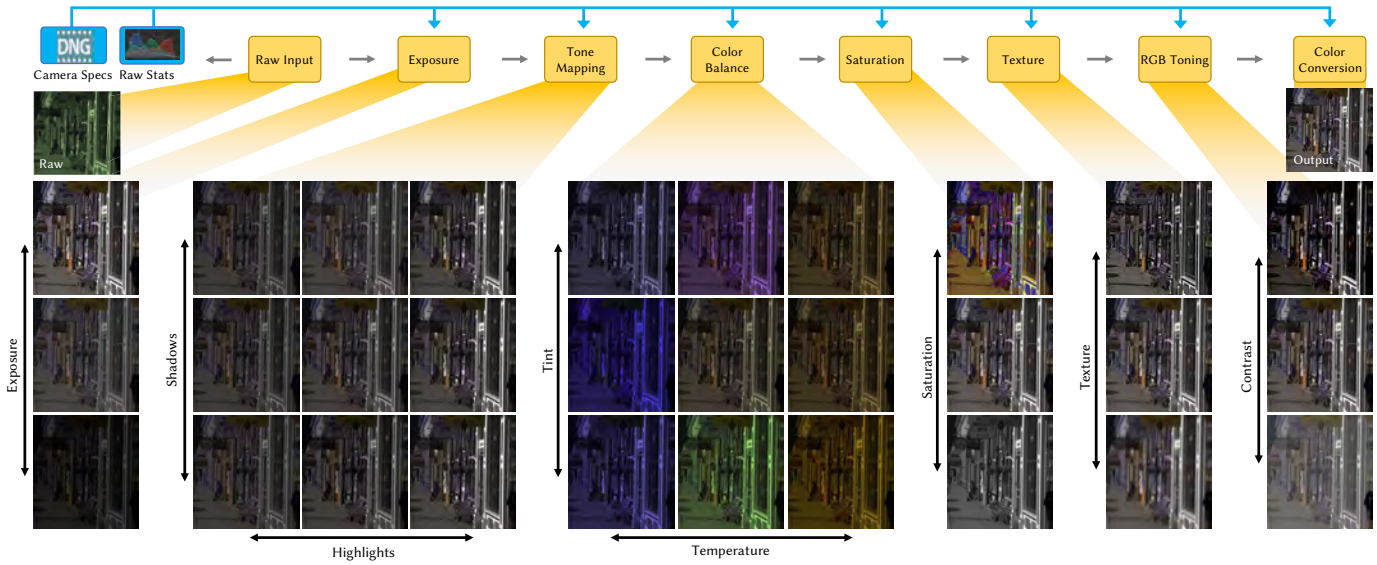

Fig. 2. Intermediary image tap-outs from Adobe Camera Raw (ACR). The ordering of stages is illustrative and does not necessarily reflect the actual pipeline, which depends on factors such as software version and camera model. ACR is parameterized by semantically meaningful "sliders" that can drastically affect the final output. Intermediate outputs illustrate that each block performs a complex transformation. Approximating the entire pipeline with a single proxy supervised only with the final output is challenging due to the combinatorial number of samples required and vanishing gradients. We instead compose a pipeline of proxy functions tailored for each block, supervising each with intermediary tap-outs.

Fig. 3. Existing methods using a single U-Net proxy to approximate the mapping for an entire pipeline [Tseng et al. 2019] fail to fit to a complex pipeline such as Adobe Camera Raw (ACR). The proposed method is able to handle the wide variety of operations in ACR by introducing per-block proxies that are tailored to areawise or pointwise operations.

(7) *RGB Toning:* This operation consists of channel-agnostic operations that affect the overall tone of the RGB channels equally. It is controlled by $\mathbf{S}_{\text{RGB}} = \{\text{Contrast}\}$.

(8) *Color Conversion:* This final operation converts linear data into a non-linear color space such as sRGB for display. The operator f_{CC} typically consists of applying a color correction matrix and a gamma curve as dictated by standards.

While the specific implementation details of each block are proprietary, the Supplementary Document provides examples of tap-outs from these blocks. Except for how to handle camera metadata (part of the public DNG specification [Adobe 2022b]), the proposed method does not require knowledge of how each block is implemented. It only requires a high level pipeline structure and input/output observations of each block.

3.2 Proxy Learning for Parameter Optimization

Every operation in the ACR pipeline is a complex function of the input image and its slider controls. Due to the large slider space and the pipeline's sequential topology, small changes early in the pipeline can lead to dramatic changes in the finished image (see Figure 2). Consequently, coarse approximation methods can miss important cases and fail to reproduce the desired behavior. Tseng et al. [2019] model the pipeline end to end with a learning procedure

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \mathcal{L}\left(f(\mathbf{I}_R, \mathbf{S}), \tilde{f}^{\mathbf{W}}(\mathbf{I}_R, \mathbf{S})\right), \tag{4}$$

where f is the finishing pipeline and $\tilde{f}^{\mathbf{W}}$ is a neural network parameterized by weights $\mathbf{W}$. Assuming the case of a single fixed camera (constant $\mathbf{M}$), the network $\tilde{f}^{\mathbf{W}}$ takes as input the raw image $\mathbf{I}_{\text{R}}$ and all sliders $\mathbf{S}$ to predict the final output. Figure 3 shows an example where this method fails to produce a modest retouching by ACR. This is due of two fundamental issues: vanishing samples and cached image statistics.

Vanishing Samples. Vanishing gradients are a well-known problem in machine learning that refers to small gradient signal arriving at early layers of a deep network [He et al. 2016]. We draw attention to the antipode problem of *vanishing samples*. Specifically, if one is to train a network on pairs $(\mathbf{I}_{\text{R}}, \mathbf{I}_{\text{F}})$, slider values need to be drawn from the whole parameter space $\mathbf{S}$. Not only is this space combinatorially large, as image samples propagate through the pipeline, the range of f_i may not cover the domain of f_{i+1} (often because

values become biased towards darker and less saturated colors). Methods such as Latin Hypercube Sampling [McKay et al. 1979] can reduce the sampling requirement but may miss corner cases, which is unacceptable for our applications.

Cached Image Statistics. Commercial photo-finishers precompute and cache global statistics such as histograms $\mathbf{H}$ on the raw input $\mathbf{I}_\text{R}$ and use it at various stages downstream. Statistics such as median, quantiles, and saturated pixel count are important for anchoring operations such as dynamic range compression, exposure adjustment, and tone curve changes. Conventional convolutional neural networks (e.g., Pix2Pix [Isola et al. 2016]) cannot learn these statistics because of their limited receptive field. Sliding window networks aggregate information only on local patches and thus are also unable to represent truly global statistics. Some works such as Onzon et al. [2021] do address this issue by precomputing the histogram and using it for inference. However, they do not consider backpropagating through the histogram computation step.

4 DIFFERENTIABLE PHOTO-FINISHING

We tackle the problems of vanishing samples and cached statistics by decomposing the training of a differentiable architecture into a sequence of approximators. Each approximator represents a single pipeline block and is conditioned on input statistics. Specifically, we propose using proxy functions $\tilde{f}_i^{W_i}$ parameterized by weights $\mathbf{W}_i$

$$\tilde{f}_i^{W_i} \approx f_i, \forall i \in \{1 \dots n\}, \tag{5}$$

and seek weights such that

$$f_n \circ \cdots \circ f_2 \circ f_1 \approx \tilde{f}_n^{\mathbf{W}_n} \circ \cdots \circ \tilde{f}_2^{\mathbf{W}_2} \circ \tilde{f}_1^{\mathbf{W}_1}. \tag{6}$$

We tailor each proxy function $\tilde{f}_i$ so that they better approximate the behavior of each block. Each proxy is trained using intermediate tap-outs $(\mathbf{I}_i, \mathbf{I}_{i+1})$. Mathematically, training is

$$\mathbf{W}^* = \bigcup_{i=1}^{n} \mathbf{W}_i^* = \underset{\cup_{i=1}^n \mathbf{W}_i}{\arg\min} \sum_{i=1}^{n} \mathcal{L}\left(f_i(\mathbf{I}_i, \mathbf{S}_i, \mathbf{M}_i, \mathbf{H}_i), \tilde{f}_i^{\mathbf{W}_i}(\mathbf{I}_i, \mathbf{S}_i, \mathbf{M}_i, \mathbf{H}_i)\right), \tag{7}$$

where $\mathcal{L}$ is an image-space loss function. By training per-block instead of end-to-end, we allow our pipeline to cover more cases while avoiding the need for a combinatorially large number of samples.

As a comparison, the end-to-end training scheme of Tseng et al. [2019] generates training data by uniform grid sampling k values per slider and produces $O\left(k^{|\mathbf{S}|}\right)$ samples. By dividing up the pipeline, we only need to densely sample each block, resulting in $O\left(\sum_{i=1}^{n} k^{|\mathbf{S}_i|}\right)$ samples where $|\mathbf{S}_i|$ is the number of sliders in the block i. Typically $|\mathbf{S}_i| \leq 3$, so our approach scales as a low-degree polynomial in the number of blocks. In ACR there are a total of 14 blocks in the Basic Panel, with only a few sliders per block. For instance, Color Balance has two sliders while Texture only has one. In our experiments, we only need approximately 10 one-megapixel raw images per block for a total of 10×14 images (compared to an end-to-end strategy that would require 10^{14}).

4.1 Pipeline Proxy Functions

Next, we describe how we implement each neural proxy and their corresponding losses. We divide operations into three categories: neural pointwise, neural areawise, and differentiable programs (see Figure 4).

Neural Pointwise. Pointwise operators consist of operations that affect the image at a per-pixel level without consideration of neighboring pixels. Examples include Saturation, Vibrance, and RGB Toning. We model these operations using a multi-layer perceptron (MLP) with 3 layers. We choose this architecture since it operates on one pixel at a time, unlike convolutional neural networks with non-unity kernel windows. To train these neural operators, we use images that densely sample the RGB cube $x \in [0, 1]^3$ and the associated sliders $s \in [0, 1]^{|\mathbf{S}_i|}$ with 100 samples per axis and obtain the corresponding $y = f_i(x, s)$. Since f_i is per-pixel, this lets us cover the entire transformation space. We use L1-loss $\mathcal{L} = \mathcal{L}_1$ for all pointwise operators. We refer to the Supplementary Document for details.

Neural Areawise. Areawise operators are nonlinear filters that depend on a pixel's neighbors (in addition to slider values and cached statistics). Examples include tone mapping and texture. We use a network architecture that applies a cascade of 3×3 convolutional filters. This allows the network to learn areawise operations such as smoothing and sharpening in addition to cross-channel modifications such as dynamic range compression. For training these operators we use a combination of per-pixel L1-Loss and spatial gradient loss to penalize errors in the way the network affects the pixel neighborhoods: $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_\nabla$. We refer to the Supplementary Document for details.

Differentiable Program Operations. Certain operations are precisely defined by the DNG specification [Adobe 2022b] or by standard definitions such as the Planckian locus [Wyszecki and Stiles 1982] and can therefore be explicitly written as a differentiable program. Examples include color conversion, which is a 3×3 matrix multiply, and gamma correction, which is a per-pixel power function. All of the information necessary to make these operations applicable to different camera types can be found in the DNG metadata, please refer to the online DNG SDK [Adobe 2022b].

Some stages are conditioned on global image statistics of the raw input such as histograms, quantiles, or per-channel saturated pixel counts. For neural proxies, we concatenate these statistics to the input to each stage along the channel dimension. For differentiable programs, the statistics parameterize the differentiable function. For example, a global contrast adjustment curve could be parameterized by the image mean, such that the contrast adjustment never changes the pixel values that are equal to the mean. The specific set of statistics used as input to the proxy depends on the pipeline stage it is modeling.

4.2 Slider Regression

We validate the accuracy of our proxies through a slider regression experiment. Consider an image $I_\text{TARGET} \in \mathbf{I}_\text{F}$ that was rendered with slider values $S_\text{TARGET} \in \mathbf{S}$; that is $I_\text{TARGET} = f_\text{PIPE}(I, S_\text{TARGET}, \mathbf{M}, \mathbf{H})$.

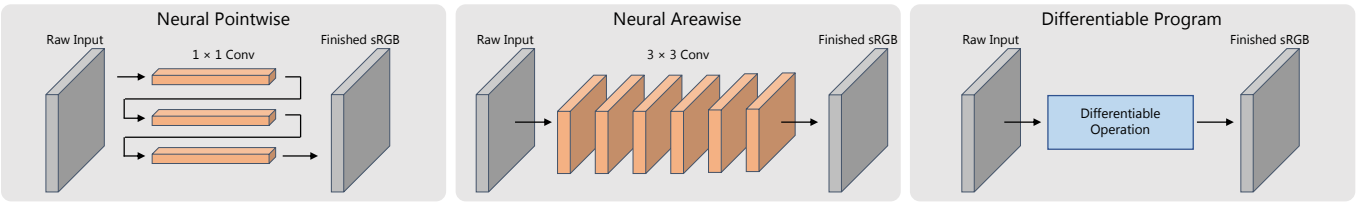

Fig. 4. Architectures for each of the supported operations. Neural pointwise operators are modeled using MLPs since they only need to affect a single pixel. Neural areawise operators consist of a cascade of 3×3 convolutions for modeling areawise operations and channel mixing. Differentiable programs follow the DNG specification to incorporate metadata and is written in a differentiable manner. Each operator also has a set of sliders and relevant precomputed image statistics as an implicit input.

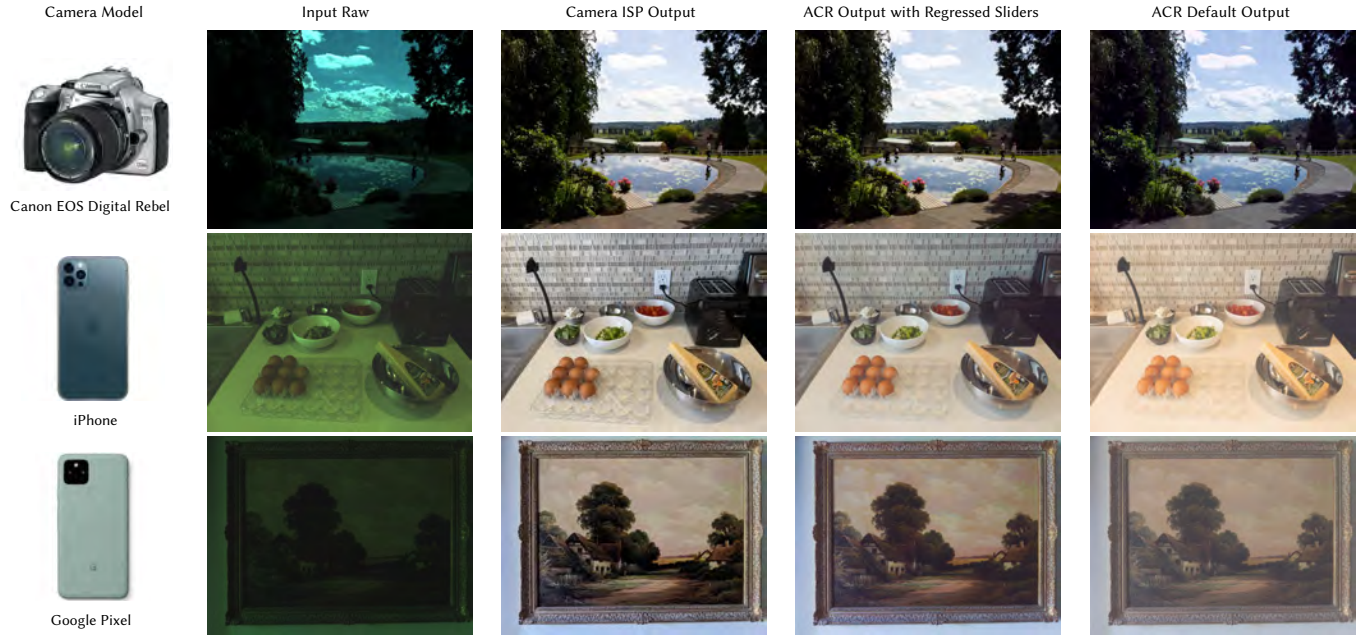

Fig. 5. Camera ISP approximation with ACR. We use our differentiable proxy to solve for slider values that makes ACR render images that resemble the output of black box camera ISPs. ACR's rendition using regressed slider values (third column) are much closer to the ISP output (second column) than its rendition under default settings (last column). We acknowledge that both ACR renditions are hazier and not as sharp as the ISP images as we do not implement proxies for dehazing and sharpening.

We then solve the following optimization problem

$$S^* = \arg\min_S \mathcal{L}\left(I_{\text{TARGET}}, f_{\text{PIPE}}(\mathbf{I}_\text{R}, S, \mathbf{M}, \mathbf{H})\right), \tag{8}$$

and verify whether S^* matches S_{TARGET}. This experiment confirms that the gradient flow through our composite f_{PIPE} is useful and allows us to deduce the slider settings used to finish a photograph. See Figure 5 for a few results and the supplement for the full set.

Commercial camera ISPs, including those in smartphones, automatically finish raw sensor captures and each is tuned to have a distinctive look. These pipelines are typically black boxes with no user-accessible parameters. Slider regression lets us predict settings such that a similar look can be reproduced in a photo-finishing application. In this experiment, we captured synchronized image pairs (the raw image $\mathbf{I}_\text{R}$ and the corresponding ISP output $\mathbf{I}_{\text{ISP}}$) using three cameras: a iPhone 12 Pro Max, a Google Pixel 3 and a Canon EOS Digital Rebel. Raw images are in the DNG format with proper metadata. With this data, we solve the same optimization as Eq. (8), setting $I_{\text{TARGET}} = I_{\text{ISP}}$ to reproduce the manufacturer's look using ACR. See Figure 5 for results. Note that the approximation accuracy is limited by the span of ACR under the slider settings of the target output. The Supplementary Document contains additional examples.

5 IMPLEMENTATION

We implement our neural photo-finishing pipeline entirely in PyTorch. We use a dataset of 1000 raw images in the DNG format as input and uniformly sample 100 points for each slider when generating the data for each intermediate. For camera metadata, we extract the relevant information from the input DNG files' eXtensible Metadata Platform (XMP) header using the open source DNG

Table 1. **Approximation Accuracy.** We compare the accuracy of the proposed stage-wise proxy approach against an existing end-to-end single network method. Our approach adequately models the ACR pipeline across a range of slider values while the single network method is unable to handle the complex parameter space with sufficient accuracy.

	Approximation Accuracy PSNR (dB)
Proposed	35.3
Tseng et al. [2019]	16.7

Table 2. **Slider Regression.** We perform slider regression using first-order optimization, enabled by our ACR proxy, against zeroth-order methods. Regression with $\tilde{f}_{\text{ACR}}$ achieves high accuracy in a short runtime whereas BayesOpt [2014] fails to find meaningful sliders even when allowed to run for 30 minutes. CMAES manages to improve over BayesOpt but still does not achieve the same regression accuracy as our proposed method.

	Loss MSE	Regression Accuracy PSNR (dB)
Proposed regress with $\tilde{f}_{\text{ACR}}$	0.00007	43.4
CMAES [2006]	0.00172	30.9
BayesOpt [2014]	0.01584	19.1

SDK. XMP metadata is machine readable and we normalize values to a useful numerical range by consulting each tag's corresponding documentation in the DNG specification. We drive training data generation using ACR's developer tool. It takes as input a DNG, an XMP file specifying all non-default slider values, an XMP "profile" that sets the remaining defaults (we use `Adobe Color`), and a list of stages to tap out. The tool produces a TIFF for each stage in its native color space and precision (`int16` or `float32`) and the final 8-bit sRGB output. Further details can be found in the Supplementary Document.

6 EVALUATION

Next, we assess the expressiveness and generalizability of our proxy on two tasks: *approximation accuracy*–how well can the proxy model the ground truth pipeline, and *slider regression*–how useful are its gradients in reverse engineering the sliders values used to produce a target image.

Approximation Accuracy with Different Proxy Networks. We compare how well our proposed proxy renders unseen image examples as compared to the work of Tseng et al. [2019]. To this end, we train both proxy networks on the ACR dataset discussed in Sec. 5. Table 1 shows that our proposed method approximates the ACR pipeline with a PSNR nearly 20 dB higher than the end-to-end proxy network, validating its stage-wise architecture. Figure 3 shows qualitative comparisons which illustrate that a single network is incapable of accurately reproducing ACR outputs, particularly when it comes to tint, texture, exposure, and temperature. We evaluate the approximation performance on a test set of ten 1MP images across three camera models: Canon EOS Digital Rebel, Canon Powershot S90, and Phase One P65+.

Slider Regression Comparison Against 0th-order Optimization Methods. Next, we evaluate the inverse parameter optimization from Sec. 4.2. Specifically, given a manually photo-finished sRGB target image, we aim to find the slider values that produced this image from the raw input by minimizing the loss between proxy-predicted images and the target. This optimization is done with first-order stochastic gradient descent using a pretrained differentiable proxy with fixed weights. We compare this proposed first-order optimization against two 0th-order methods: BayesOpt [Martinez-Cantin 2014], a derivative-free method based on Gaussian Processes, and vanilla CMAES [Hansen 2006], an evolutionary strategy. The results reported in Table 2 validate that 0th-order optimization approaches do not offer an alternative to our approach: they are both slow (due to derivative-free sampling) and unable to find adequate slider values (see Figure 6). For these experiments we regress the temperature, tint, contrast, and saturation sliders and we evaluate on a test set of ten 1MP images across two camera models: Canon EOS Digital Rebel and Phase One P65+. For experimental and implementation details and more results, please refer to the Supplementary Document.

6.1 Comparing Against Multi-stage ReconfigISP

We now compare against the recent work of ReconfigISP [Yu et al. 2021]. Although ReconfigISP tackles an orthogonal problem—that is, designing new ISPs by rearranging blocks through the use of differentiable proxies, we compare for completeness. Specifically, instead of fitting ACR by using our proposed proxy architecture, we use the SRCNN proxy architecture from ReconfigISP. ReconfigISP proposed to use the SRCNN network as a "one-size-fits-all" architecture. However, we show that this architecture is unable to fit the operations of a complex photo finishing pipeline such as ACR, and consequently performs poorly in the downstream regression tasks. In Figure 7, we measure its ability to fit individual pipeline blocks. We observe that the ReconfigISP proxy achieves more than 20 dB lower approximation accuracy, corresponding to more than two orders of magnitude in mean-squared error. This validates the proposed catered architecture design, both qualitatively and quantitatively. See the Supplementary Document for additional discussion.

Next, following ReconfigISP, we optimize a chain of SRCNN proxies. We use the proposed optimization method with the same loss function. In Figure 8, we report the results for the same slider regression experiments from the previous paragraph with the proposed method and with ReconfigISP. Again, the low approximation accuracy achieved by the ReconfigISP in turn results in poor performance in downstream tasks. We found that slider regression was not possible with the SRCNN proxies, which we also document qualitatively and quantitatively in the Supplementary Document.

These two experiments confirm that ReconfigISP, as either a proxy in isolation, or as multi-stage method, fails for slider regression on complex photo-finishing pipelines like the one targeted in this work.

6.2 Slider Regression with Monolithic Proxies

We next compare the slider regression performance when using a monolithic U-Net [Tseng et al. 2019] or our proposed architecture. Specifically, we trained networks to fit the ACR pipeline (either as a single network or stage-by-stage as proposed) and then we

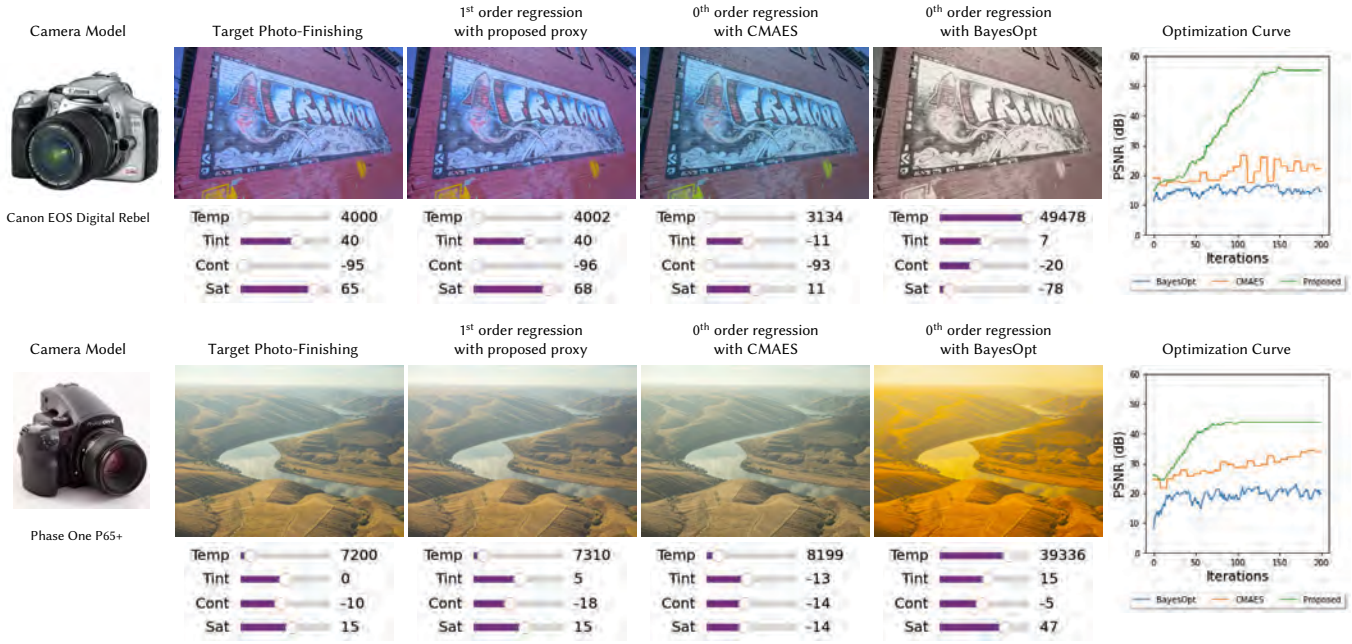

Fig. 6. Comparing slider regression against 0th-order methods. Above, we show the regression result when running our proposed first-order regression with the differentiable proxy chain versus regression with 0th-order methods. The 0th-order methods converge to sub-optimal solutions and are unable to find the specific settings that would match the finished look. In contrast, our first-order method utilizes gradient information and converges to the correct settings. This is seen in the plots, which describe the PSNR in the finished image using the best sliders found by each method at each iteration. Note that these slider settings are plugged into the true ACR renderer for evaluation. See Supplementary Document and Video for additional results.

Fig. 7. Proxy Comparison of Proposed and ReconfigISP [2021] SRCNN proxy architecture. The "one-size-fits-all" architecture used by ReconfigISP is unable to accurately model the operations found in complex commercial pipelines such as ACR. Here, we show that our proposed architectures act as better proxies for the individual modules.

use the frozen network to perform slider regression using first-order gradient descent. Examples are shown in Figure 8. Since the alternative proxy architecture from [Tseng et al. 2019] is unable to accurately fit ACR, it is consequently ineffective for slider regression. As shown in the figure, the converged slider values are far away from the target setting. See the Supplementary Document for additional quantitative and qualitative results.

7 RAW STYLE TRANSFER

Photo-finishing can be thought of as a special type of style transfer. When an artist digitally develops a raw image, they adjust slider values to achieve a desired "look". For example, one could adjust the sliders to match the warm tone of Blade Runner 2049 (2017) or to resemble the green tints of The Matrix (1999). This differs significantly from "artistic style transfer" across different domains; e.g., converting photos to resemble Van Gogh paintings [Gatys et al. 2016]. Photo-finishing does not change the content of an image. Similar to the process of developing film in a darkroom, digital photo finishing operates on a constrained manifold of renditions.

In this section, we show how to use our proposed neural photo finisher in an automatic style transfer system. Unlike existing one-size-fits-all "filters" found in commercial software such as iPhone's "Magic Wand", our system can be adapted to multiple styles. Our approach is to build a neural network around the photo finisher. The same architecture can be trained once for each style (Figure 9). For the network to generalize to any input raw image (but for a single style), it crucially needs to be conditioned on the input, and to observe a sufficient number of examples of the target style during training. Conditioning on the input is straightforward: we use an encoder network that takes as input the raw image and produces learned global and local features. These feature maps are then fed to fully connected layers to predict slider values for the photo finisher. To further assist the network in determining global slider adjustments such as global contrast, we augment feature maps

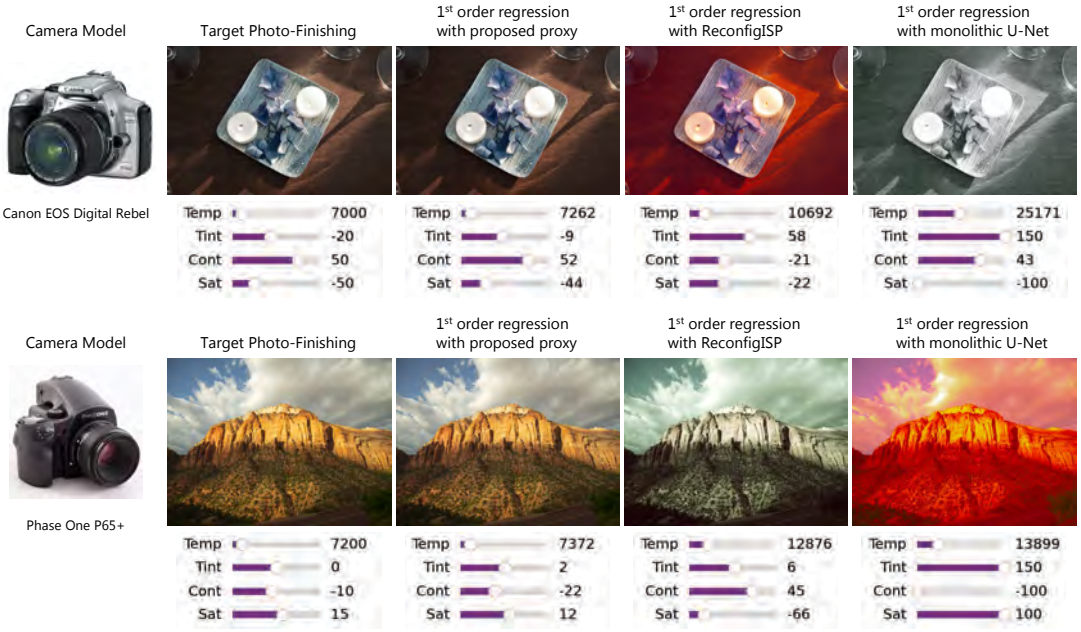

Fig. 8. Comparing slider regression against existing Monolithic (Tseng et al. [2019] and Multi-stage Proxies (ReconfigISP [2021]). Above, we show the regression result when running our proposed first-order regression with the differentiable proxy chain versus regression with a monolithic U-Net [Tseng et al. 2019] and with ReconfigISP [Yu et al. 2021] that employs multi-stage SRCNN proxies. Performing slider regression with these alternative proxy architectures results does not give a solution close to the true slider settings. This is due to the poor approximation accuracy that is obtained when using these alternatives. In contrast, our carefully designed architecture allows us to better fit the ACR pipeline and allows us to perform accurate slider regression. Note that all methods use first-order gradient descent for slider regression.

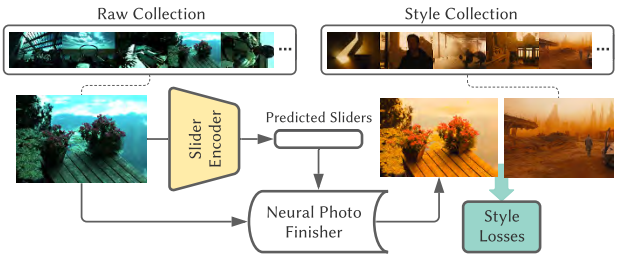

Fig. 9. We perform photo-finishing style transfer, where we train an encoder to dynamically predict ACR slider settings conditioned on an input image. Similar to the "Auto Adjust" or "Magic Wand" feature on many commercial photo-editing applications, our network predicts sliders based on the input image content. However, unlike existing "one-size-fits-all" automatic adjustment algorithms, we can cater our network to different styles by training it on different target style collections. Raw images are scaled for display.

with image statistics in the form of histograms extracted from the raw input.

Style Loss. To teach the network what a style is, we show the network frames from short film clips, which have been professionally edited to preserve a consistent look with different content. For any content/style image pair, we compute their style loss

$$\mathcal{L}_{\text{STYLE}} = \mathcal{L}_{\text{GRAM}} + \lambda_1 \mathcal{L}_{\text{LUMA}} + \lambda_2 \mathcal{L}_{\text{CHROMA}}$$

with

$$\mathcal{L}_{\text{GRAM}} = \mathcal{L}(G(\mathbf{I}_{\text{F}}), G(\mathbf{I}_{\text{S}}))$$
$$\mathcal{L}_{\text{LUMA}} = \mathcal{L}\left(H_{1\text{d}}(\mathbf{I}_{\text{F}}^{\text{Y}}), H_{1\text{d}}(\mathbf{I}_{\text{S}}^{\text{Y}})\right)$$
$$\mathcal{L}_{\text{CHROMA}} = \mathcal{L}\left(H_{2\text{d}}(\mathbf{I}_{\text{F}}^{\text{UV}}), H_{2\text{d}}(\mathbf{I}_{\text{S}}^{\text{UV}})\right),$$

where $\mathbf{I}_{\text{F}} = f_{\text{ACR}}(\mathbf{I}_{\text{R}}, E(\mathbf{I}_{\text{R}}))$ is the photo-finished output, G are the Gram matrices of feature layers of a pre-trained VGG-19 encoder (we use conv_{1…5}_1 as in Johnson et al. [2016]), $H_{1\text{d}}$ and $H_{2\text{d}}$ are one- and two-dimensional soft histograms, applied to the Y and UV channels respectively, and λ_1 and λ_2 are scalar weights (see Supplementary Document for details). The Gram matrix removes the locality of VGG features in order to focus on style over content, while the Y and UV histogram losses capture the "color palette" of the target style. Since the entire style transfer network is differentiable, we can pretrain and fix the photo finisher and backpropagate the style loss onto only the slider encoder. Note that since style loss is computed on the finished output, style images $\mathbf{I}_{\text{S}}$ need not be raw. We found a simple L2 loss directly on histograms to converge better than a Wasserstein distance. Figure 10 demonstrates our system transferring different styles to different content images.

We note that our system is not tied to a specific differentiable endpoint loss. We choose the VGG loss because it is used in style transfer applications [Johnson et al. 2016], and has been shown through user studies to be a reasonable measure of human perception [Zhang et al.

Table 3. **Finishing-driven Demosaicking and Denoising.** Our proposed pipeline allows backpropagation of gradients through both the photo-finishing proxy and a joint demosaicking and denoising network onto the Bayer raw input. Hence, we are able to train the latter using a post-finishing loss. To evaluate performance, we first demosaic a set of 160 Bayer images using both the end-to-end learned algorithm f_{NN} as well as a pretrained baseline network f_{DE} [Gharbi et al. 2016]. We then finish the demosaicked images with ACR and compare the results to ground truth. Since ground truth is computed by finishing a low-noise long exposure image using ACR [Abdelhamed et al. 2018], the bottom right entry is infinite.

	Short Exposure PSNR (dB)	Long Exposure PSNR (dB)
Learned Demosaic f_{NN}	27.0	27.2
Default ACR Demosaic f_{DE}	20.0	∞

2018]. To further reduce subjectivity, our stylization experiments use distinctly different target styles (e.g., Matrix vs. Blade Runner).

Video Style Transfer. Our approach is also temporally stable and can be applied to raw video (CinemaDNG and similar standards are now widespread in film production). Since the network predicts slider values rather than images, we can efficiently stylize a video by running inference only on keyframes and interpolating slider values. We include video results in the Supplementary Material.

Ethical Considerations. This work allows users to determine semantically meaningful sliders that will photo-finish raw videos with the look of existing films such as The Matrix (1999). Although this does make it easier for anyone to copy a style and then add additional changes, we hope that this work will encourage artists to build upon existing looks while giving proper attribution.

8 LOW PHOTON COUNTS TO SRGB

Most camera sensors are sensitive to a broad range of wavelengths. To record color, most use a filter array to select wavelength bands, typically in an RGGB Bayer configuration. The interpolation of these wavelength-filtered photon counts to obtain multi-channel raw images is known as demosaicking (see also Sec. 2). Demosaicking is one of the earliest steps in developing a digital photograph and takes place even before the slider-parameterized photo-finishing blocks heretofore discussed. It is an ill-posed problem made more challenging by the presence of imaging noise.

Previously, we showed that we can backpropagate loss function gradients onto sliders and use those gradients to train a style transfer network. Next, we demonstrate that our neural photo-finisher also allows backpropagating gradients onto the Bayer input, which permits training a joint demosaicking and denoising network. Specifically, we aim to train a neural network f_{NN} that takes as input a Bayer raw image and, *independent* of exposure time (and therefore image noise), outputs a linear RGB image such that after finishing, the sRGB output resembles the same finishing process applied to a corresponding long-exposure Bayer image with low noise. Formally, we aim to balance two objectives. First, when given a long-exposure Bayer image, reproduce the ACR demosaicker f_{DE} and finishing pipeline f_{PIPE}:

$$\mathcal{L}_{\text{LONG}} = \mathcal{L}(f_{\text{PIPE}}(f_{\text{NN}}(B_{\text{LONG}})), f_{\text{PIPE}}(f_{\text{DE}}(B_{\text{LONG}}))).$$

Next, when given a corresponding short-exposure Bayer image B_{SHORT} with reduced signal-to-noise ratio, the network should denoise it such that it still produces the same finished result:

$$\mathcal{L}_{\text{SHORT}} = \mathcal{L}(f_{\text{ACR}}(f_{\text{NN}}(B_{\text{SHORT}})), f_{\text{ACR}}(f_{\text{DE}}(B_{\text{LONG}}))).$$

We train f_{NN} on the Smartphone Image Denoising Dataset [Abdelhamed et al. 2018], which features such short/long Bayer pairs.

Figure 13 shows qualitative results on two Bayer images. We observe that f_{NN} maintains good reconstruction performance on both noisy and clean inputs. In contrast, the default algorithm f_{DE} leaves behind noticeable noise in the finished sRGB image. The quantitative results reported in Table 3 validate our network's improvement over the conventional pipeline by 7 dB in PSNR for short exposure inputs.

In this section, we showed that we can construct a high-quality Bayer to sRGB neural network by adding a demosaicking and denoising module to our photo-finishing proxy. This end-to-end differentiable pipeline allows slider-based photo editing without sacrificing the well-understood steps of a camera ISP [Chen et al. 2018; Diamond et al. 2021].

9 ADVERSARIAL PHOTO-FINISHING

In this section, we leverage our learned proxy to investigate an adversarial attack that fools classifiers run on images finished by one photographer but leaves classification performance unchanged for another photographer. Specifically, let S_1, S_2 denote two sets of slider settings corresponding to two different photographers. We seek a small perturbation δ such that the output $f_{\text{PIPE}}(\mathbf{I}_{\text{R}} + \delta, S_1, \mathbf{M}, \mathbf{H})$ is incorrectly classified by a pretrained sRGB classifier G, while $f_{\text{PIPE}}(\mathbf{I}_{\text{R}} + \delta, S_2, \mathbf{M}, \mathbf{H})$ remains correctly classified. In other words, we investigate if the *same* raw input is fed to two *different* photo-finishers that only differ in their slider settings, can we find an adversarial pattern that targets one but not the other? To test whether such an "attack" exists, we solve the following optimization problem

$$\begin{aligned} \delta^* = \arg\max_{\delta} (\quad & \mathcal{L}_{\text{CE}}(I_{\text{GT}}, G(f_{\text{PIPE}}(\mathbf{I}_{\text{R}} + \delta, S_1, \mathbf{M}, \mathbf{H}))) \\ & - \mathcal{L}_{\text{CE}}(I_{\text{GT}}, G(f_{\text{PIPE}}(\mathbf{I}_{\text{R}} + \delta, S_2, \mathbf{M}, \mathbf{H}))) \quad) \\ s.t. \quad & ||\delta||_2 < \epsilon, \end{aligned} \tag{9}$$

where $\mathcal{L}_{\text{CE}}$ denotes the cross-entropy loss, I_{GT} denotes the ground truth label, and ϵ denotes the maximum perturbation allowed. We set $\epsilon = 0.3$ in our experiment. We test our method for different classifier networks G, with ResNet variants [He et al. 2016], Inception [Szegedy et al. 2016], and MobileNet [Howard et al. 2017] as backbones. As the composite pipeline $G \circ f_{\text{PIPE}}$ is differentiable, we solve Eq. (9) using stochastic gradients with the Adam optimizer [Kingma and Ba 2015] (see the Supplementary Document for details).

We evaluate our attack on the ImageNet dataset [Deng et al. 2009]. As ImageNet lacks raw images, we first pretrain an sRGB to raw synthesis network on our dataset under the L2 loss, which we then apply to ImageNet. Next, with the raw and sRGB image pairs, we evaluate the proposed attack for two manually chosen

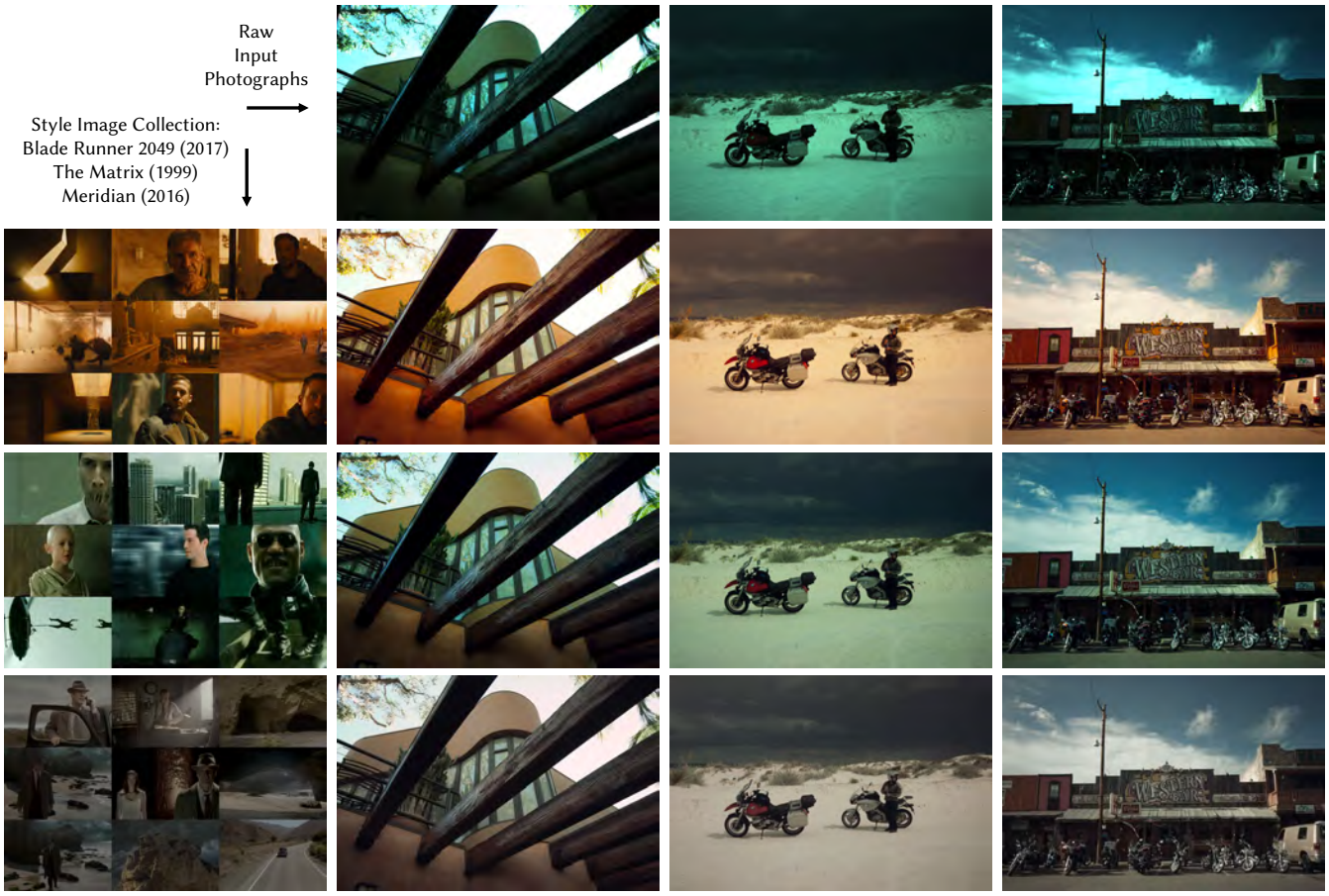

Fig. 10. **Raw Style Transfer.** Test-time style transfer results on still photographs. The top row shows the raw input images (scaled for display). The other rows depict various output stylizations. The left column shows a subset of the collection of style images used to train the slider encoder. We train the encoder separately for each style while our proxy is pretrained and fixed during both training and inference.

Table 4. **Adversarial Photo-Finishing.** We report the top 1 accuracy on the ImageNet dataset after the proposed attack on ACR whose output we pass to classifier G. Our attack successfully deceives all four classifiers for the pipeline with slider settings S_1 while leaving predictions for S_2 intact.

Classifier G	ResNet101	ResNet50	InceptionV3	MobileNet
No attack	77.37	76.13	77.29	74.04
S_1 Pipeline	0.18	0.03	0.21	0.13
S_2 Pipeline	81.34	78.89	80.27	75.28

slider sets (S_1 and S_2) and report quantitative results in Table 4. The proposed attack decreases the classification accuracy of the pipeline parameterized by S_1 to less than 1%, while leaving the performance of the pipeline parameterized by S_2 unchanged. This trend is also confirmed in the qualitative results in Figure 14. The same raw perturbation δ is transformed to a much stronger adversarial pattern for S_1 than for S_2, especially near edges around the object of interest. This explains how the attack deceives the pipeline with sliders S_1 while preserving the performance for S_2. Note that although our sRGB to raw synthesis network may not perfectly reproduce the characteristics of raw images, our findings still hold as the *same* synthetic raw image is input to both pipelines S_1 and S_2. We again refer the reader to the Supplementary Document for additional details.

10 DISCUSSION AND CONCLUSION

Limitations. A fundamental limitation of our method is that training separate proxies requires "opening the black box" and accessing the inputs and outputs of intermediate stages of the reference pipeline. Because of this, it may not be possible to apply our method to closed systems such as hardware ISPs. Furthermore, while our implementation models a wide variety of operations, it does not model every component of Adobe Camera Raw. Notably, we do not yet have proxies for the visually important dehazing and sharpening steps, nor can we handle geometric operations such as cropping or lens distortion correction. We hope to implement these as future work.

Our technique relies on image statistics which are computed from histograms. Even when using soft histograms, gradient flow through these image statistics is poor because they are only non-zero at a

Fig. 11. **Video Style Transfer.** We apply the three movie styles from Figure 10 to a few raw videos of our own. Our approach of predicting slider values makes the style transfer temporally consistent throughout the sequence even when it contains large changes in dynamic range (e.g., from sunlight to shade). See the Supplementary Material for all video results.

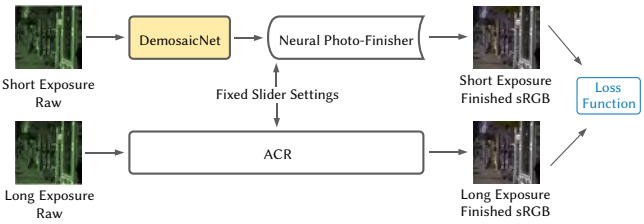

Fig. 12. End-to-end trained low-light demosaicking and denoising driven by a photo-finishing loss. We train a joint demosaicking and denoising network atop our neural photo-finisher. Our combined network outputs denoised images that resemble those rendered by ACR under normal illumination.

sparse set of bin positions. The popular max-pooling operation used in machine learning avoids this problem because a small fraction of pixels still admit gradient flow. But with histograms, the set of bins with non-zero bins is small regardless of image size. A related operation that hinders gradient flow are lookup tables (e.g., the *Color Tables* ACR stage). These lookup tables can approximate an arbitrary per-pixel operation and do not necessarily encode a smooth function. The larger the table, the more prone it is to local minima and saddle points, leading to less consistent performance for applications like style transfer and demosaicking. This problem can be somewhat ameliorated by carefully choosing training hyper-parameters such as the learning rate. Estimating robust global image statistics and accurately modeling lookup tables while ensuring gradient flow is interesting theoretical work we wish to pursue.

Conclusion and Future Work. In this paper, we proposed a fine-grained differentiable camera renderer. Our pipeline allows for gradient flow from final output sRGB all the way back to raw photon counts, all parameterized by a comprehensive set of meaningful sliders. We view our work as a bridge between traditional photo finishing workflows and machine learning research. Whereas previous methods circumvented the non-differentiability of image processing pipelines by defining their own differentiable pipelines or via approximation and inversion, we instead resolve this gap through a stage-wise approximation that preserves the pipeline's semantics.

We demonstrate the potential of the our differentiable finishing pipeline on a variety of applications. From style transfer to low-light photo-finishing to an adversarial attack, we demonstrate its immediate impact on a wide gamut of disciplines. We hope that the release of this renderer will provide a tool for artists, researchers, and practitioners to connect machine learning with raw images.

While our stage-wise architecture immediately allows for manual rearrangement of the modules to explore different rendering schemes, investigating automated architecture search for such pipelines when coupled to downstream imaging or vision modules may prove as an exciting future direction.

REFERENCES

Rameen Abdal, Yipeng Qin, and Peter Wonka. 2019. Image2StyleGAN: How to Embed Images Into the StyleGAN Latent Space?. In *IEEE International Conference on Computer Vision (ICCV)*. 4431–4440.

Abdelrahman Abdelhamed, Stephen Lin, and Michael S. Brown. 2018. A High-Quality Denoising Dataset for Smartphone Cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1692–1700.

Adobe. 2022a. Adobe Camera Raw. https://www.adobe.com/products/photoshop/cameraraw.html.

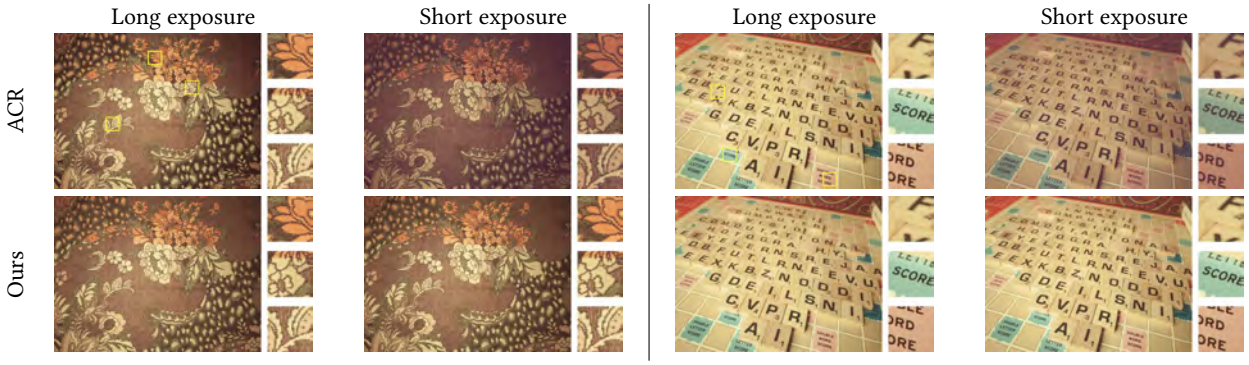

Fig. 13. **Qualitative demosaicking results.** We visualize the demosaicking and denoising results from the default ACR demosaicker and our network on selected Bayer raw pairs from SIDD. Our demosaicking network is capable of reconstructing clean outputs from both noisy (short-exposure) and clean (long-exposure) Bayer raw inputs.

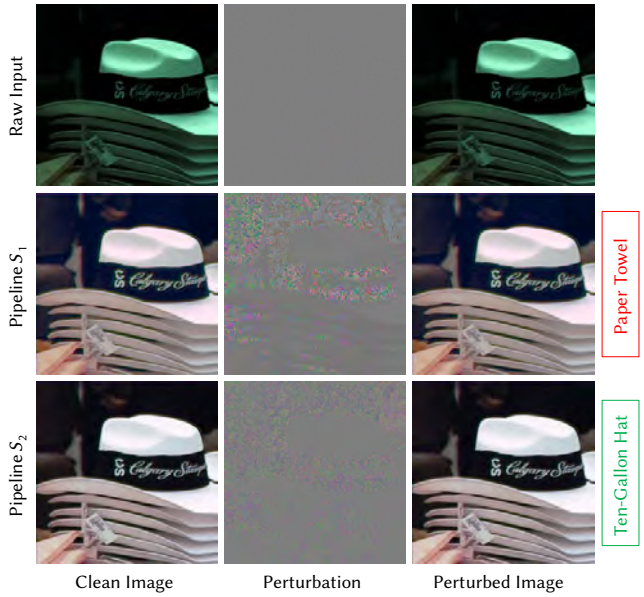

Fig. 14. **Adversarial Photo-Finishing.** Using the proposed proxy function, we learn an adversarial raw perturbation δ that map to a stronger adversarial pattern for pipeline S_1, corresponding to photographs from photographer 1, than S_2, corresponding to photographs from photographer 2, especially at object edges.

Adobe. 2022b. Adobe Digital Negative. https://helpx.adobe.com/photoshop/digital-negative.html.

Adobe. 2022c. Adobe Lightroom. https://adobe.com/products/photoshop-lightroom.html.

Mahmoud Afifi and Abdullah Abuolaim. 2021. Semi-Supervised Raw-to-Raw Mapping. *British Machine Vision Conference (BMVC)* (2021).

Mahmoud Afifi and Michael S. Brown. 2020a. Deep White-Balance Editing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1394–1403.

Mahmoud Afifi and Michael S. Brown. 2020b. Interactive White Balancing for Camera-Rendered Images. In *IS&T Color and Imaging Conference (CIC)*. 136–141.

Mahmoud Afifi, Konstantinos G. Derpanis, Björn Ommer, and Michael S. Brown. 2021. Learning Multi-Scale Photo Exposure Correction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 9153–9163.

Mahmoud Afifi, Abhijith Punnappurath, Abdelrahman Abdelhamed, Hakki Can Karaimer, Abdullah Abuolaim, and Michael S. Brown. 2019. Color Temperature Tuning: Allowing Accurate Post-Capture White-Balance Editing. In *IS&T Color Imaging Conference (CIC)*. 1–6.

Maaz Bin Safeer Ahmad, Jonathan Ragan-Kelley, Alvin Cheung, and Shoaib Kamil. 2019. Automatically Translating Image Processing Libraries to Halide. *ACM Transactions on Graphics* 38, 6, Article 204 (2019), 13 pages.

Apple. 2022. Apple ProRAW. https://support.apple.com/en-us/HT211965.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, 10 (2012), 281–305.

James Bergstra, Dan Yamins, and David D Cox. 2013. Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. In *Python in Science Conference*. 13–20.

Nicolas Bonneel, Kalyan Sunkavalli, Sylvain Paris, and Hanspeter Pfister. 2013. Example-Based Video Color Grading. *ACM Transactions on Graphics* 32, 4, Article 39 (2013), 12 pages.

Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. 2011. Learning Photographic Global Tonal Adjustment with a Database of Input / Output Image Pairs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 97–104.

Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. 2018. Learning to See in the Dark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3291–3300.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017b. ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models. *ACM Workshop on Artificial Intelligence and Security*, 15–26.

Qifeng Chen, Jia Xu, and Vladlen Koltun. 2017a. Fast Image Processing With Fully-Convolutional Networks. In *IEEE International Conference on Computer Vision (ICCV)*. 2516–2525.

Marcos V. Conde, Steven McDonagh, Matteo Maggioni, Alevs Leonardis, and Eduardo Pérez-Pellitero. 2022. Model-Based Image Signal Processors via Learnable Dictionaries. In *AAAI Conference on Artificial Intelligence (AAAI)*. 481–489.

Darktable. 2022. Darktable. https://www.darktable.org.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 248–255.

Blackmagic Design. 2022. DaVinci Resolve. blackmagicdesign.com/products/davinciresolve.

Steven Diamond, Vincent Sitzmann, Frank Julca-Aguilar, Stephen Boyd, Gordon Wetzstein, and Felix Heide. 2021. Dirty Pixels: Towards End-to-End Image Processing and Perception. *ACM Transactions on Graphics* 40, 3, Article 23 (2021), 15 pages.

Zheng-Jun Du, Kai-Xiang Lei, Kun Xu, Jianchao Tan, and Yotam Gingold. 2021. Video Recoloring via Spatial-Temporal Geometric Palettes. *ACM Transactions on Graphics* 40, 4, Article 150 (2021), 16 pages.

Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. 2017. A Learned Representation for Artistic Style. In *International Conference on Learning Representations (ICLR)*.

Alexei A. Efros and William T. Freeman. 2001. Image Quilting for Texture Synthesis and Transfer. *ACM Transactions on Graphics*, 341–346.

Katrin Eismann, Wayne Palmer, and Dennis Dunbar. 2018. *Adobe Photoshop Restoration & Retouching* (4th ed.). New Riders.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2016. Image Style Transfer Using Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2414–2423.

Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. 2016. Deep Joint Demosaicking and Denoising. *ACM Transactions on Graphics* 35, 6, Article 191

(2016), 12 pages.
Michaël Gharbi, Jiawen Chen, Jonathan T. Barron, Samuel W Hasinoff, and Frédo Durand. 2017. Deep Bilateral Learning for Real-Time Image Enhancement. *ACM Transactions on Graphics* 36, 4, Article 118 (2017), 12 pages.
Nikolaus Hansen. 2006. The CMA Evolution Strategy: A Comparing Review. *Towards a New Evolutionary Computation* (2006), 75–102.
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)*. 770–778.
James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, and Pat Hanrahan. 2014. Darkroom: Compiling High-level Image Processing Code into Hardware Pipelines. *ACM Transactions on Graphics* 33, 4, Article 144 (2014), 11 pages.
Felix Heide, Markus Steinberger, Yun-Ta Tsai, Mushfiqur Rouf, Dawid Pająk, Dikpal Reddy, Orazio Gallo, Jing Liu, Wolfgang Heidrich, Karen Egiazarian, Jan Kautz, and Kari Pulli. 2014. FlexISP: A Flexible Camera Image Processing Framework. *ACM Transactions on Graphics* 33, 6, Article 231 (2014), 13 pages.
Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. 2001. Image Analogies. *ACM Transactions on Graphics*, 327–340.
Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv:1704.04861* (2017).
Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. 2018. Exposure: A White-Box Photo Post-Processing Framework. *ACM Transactions on Graphics* 37, 2, Article 26 (2018), 17 pages.
Alexis Van Hurkman. 2010. *Color Correction Handbook: Professional Techniques for Video and Cinema*. Peachpit Press.
Andrey Ignatov, Luc Van Gool, and Radu Timofte. 2020. Replacing Mobile Camera ISP with a Single Deep Learning Model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2275–2285.
Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2016. Image-to-Image Translation with Conditional Adversarial Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5967–5976.
Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *European Conference on Computer Vision (ECCV)*. 694–711.
Hakki Can Karaimer and Michael S. Brown. 2016. A Software Platform for Manipulating the Camera Imaging Pipeline. In *European Conference on Computer Vision (ECCV)*. 429–444.
Tero Karras, Samuli Laine, and Timo Aila. 2019. A Style-Based Generator Architecture for Generative Adversarial Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4217–4228.
Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.
Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. 2018. Differentiable Programming for Image Processing and Deep Learning in Halide. *ACM Transactions on Graphics* 37, 4, Article 139 (2018), 13 pages.
Yu-Lun Liu, Wei-Sheng Lai, Yu-Sheng Chen, Yi-Lung Kao, Ming-Hsuan Yang, Yung-Yu Chuang, and Jia-Bin Huang. 2020. Single-Image HDR Reconstruction by Learning to Reverse the Camera Pipeline. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1648–1657.
Ruben Martinez-Cantin. 2014. BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits. *Journal of Machine Learning Research* 15, 115 (2014), 3735–3739.
M. D. McKay, R. J. Beckman, and W. J. Conover. 1979. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics* 21, 2 (1979), 239–245.
Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P. Srinivasan, and Jonathan T. Barron. 2021. NeRF in the Dark: High Dynamic Range View Synthesis from Noisy Raw Images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 16190–16199.
Ali Mosleh, Avinash Sharma, Emmanuel Onzon, Fahim Mannan, Nicolas Robidoux, and Felix Heide. 2020. Hardware-in-the-loop End-to-end Optimization of Camera Image Processing Pipelines. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7526–7535.
Jun Nishimura, Timo Gerasimow, Rao Sushma, Alexsandar Sutic, Chyuan-Tyng Wu, and Gilad Michael. 2018. Automatic ISP Image Quality Tuning Using Nonlinear Optimization. In *IEEE International Conference on Image Processing (ICIP)*. 2471–2475.
ON Semi MT9P001. 2017. MT9P001: 1/2.5-Inch 5 Mp CMOS Digital Image Sensor. https://www.onsemi.com/pdf/datasheet/mt9p001-d.pdf.
Emmanuel Onzon, Fahim Mannan, and Felix Heide. 2021. Neural Auto-Exposure for High-Dynamic Range Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7706–7716.
PicsArt. 2022. PicsArt. https://picsart.com/.
Rajeev Ramanath, Wesley E. Snyder, Youngjun Yoo, and Mark S. Drew. 2005. Color Image Processing Pipeline. *IEEE Signal Processing Magazine* 22, 1 (2005), 34–43.
Ling Shao, Ruomei Yan, Xuelong Li, and Yan Liu. 2014. From Heuristic Optimization to Dictionary Learning: A Review and Comprehensive Comparison of Image Denoising Algorithms. *IEEE Transactions on Cybernetics* 44, 7 (2014), 1001–1013.
Zheng Shi, Ethan Tseng, Mario Bijelic, Werner Ritter, and Felix Heide. 2021. ZeroScatter: Domain Transfer for Long Distance Imaging and Vision through Scattering Media. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3475–3485.
Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2818–2826.
Ethan Tseng, Felix Yu, Yuting Yang, Fahim Mannan, Karl St. Arnaud, Derek Nowrouzezahrai, Jean-François Lalonde, and Felix Heide. 2019. Hyperparameter Optimization in Black-box Image Processing using Differentiable Proxies. *ACM Transactions on Graphics* 38, 4, Article 27 (2019), 14 pages.
Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 10734–10742.
Günter Wyszecki and W. S. Stiles. 1982. *Color Science: Concepts and Methods, Quantitative Data and Formulae* (2nd ed.). Wiley.
Xide Xia, Meng Zhang, Tianfan Xue, Zheng Sun, Hui Fang, Brian Kulis, and Jiawen Chen. 2020. Joint Bilateral Learning for Real-time Universal Photorealistic Style Transfer. In *European Conference on Computer Vision (ECCV)*. 327–342.
Jaejun Yoo, Youngjung Uh, Sanghyuk Chun, Byeongkyu Kang, and Jung-Woo Ha. 2019. Photorealistic Style Transfer via Wavelet Transforms. In *IEEE International Conference on Computer Vision (ICCV)*. 9035–9044.
Ke Yu, Zexian Li, Yue Peng, Chen Change Loy, and Jinwei Gu. 2021. ReconfigISP: Reconfigurable Camera Image Processing Pipeline. In *IEEE International Conference on Computer Vision (ICCV)*. 4248–4257.
Lei Zhang, Xiaolin Wu, Antoni Buades, and Xin Li. 2011. Color Demosaicking by Local Directional Interpolation and Nonlocal Adaptive Thresholding. *Journal of Electronic Imaging* 20, 2 (2011), 023016.
Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 586–595.
Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. 2020. EcoNAS: Finding Proxies for Economical Neural Architecture Search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 11396–11404.
Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. 2019. BayesNAS: A Bayesian Approach for Neural Architecture Search. In *International Conference on Machine Learning (ICML)*. 7603–7613.
Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *IEEE International Conference on Computer Vision (ICCV)*. 2242–2251.