Neural Photo-Finishing Supplementary Information

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

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 Supplementary Information. *ACM Trans. Graph.* 41, 6, Article 238 (December 2022), 28 pages. https://doi.org/10.1145/3550454.3555526

In this Supplementary Document, we provide additional details on the methods and additional results in support of the findings from the Main Document.

1 PHOTO-FINISHING OPERATORS

To demonstrate the effect of the described processing blocks in the Main Document, we show examples of intermediary tap-outs for each block in Figures 1, 2, 3, 4, 5, 6, 7, 8.

*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

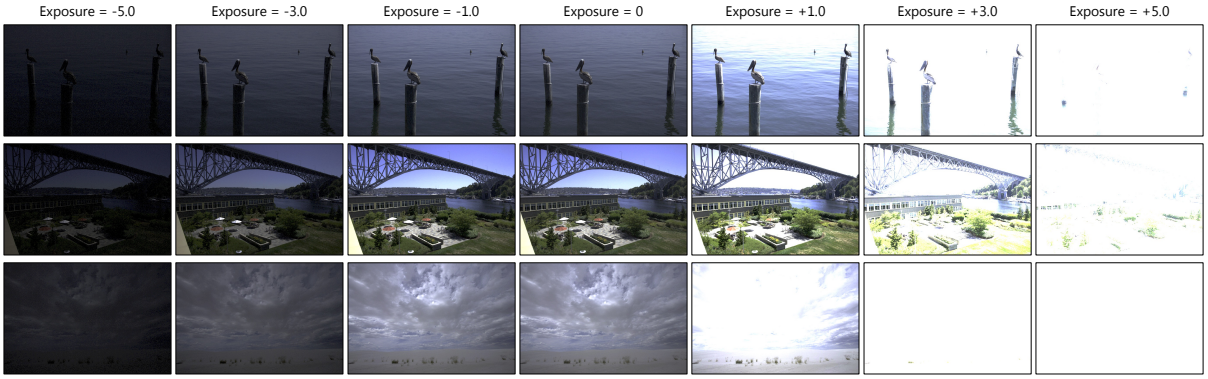

Fig. 1. Exposure Tap-outs: The exposure block scales the image intensity by a number of stops. Note that the tap-outs have been equally brightened for visualization.

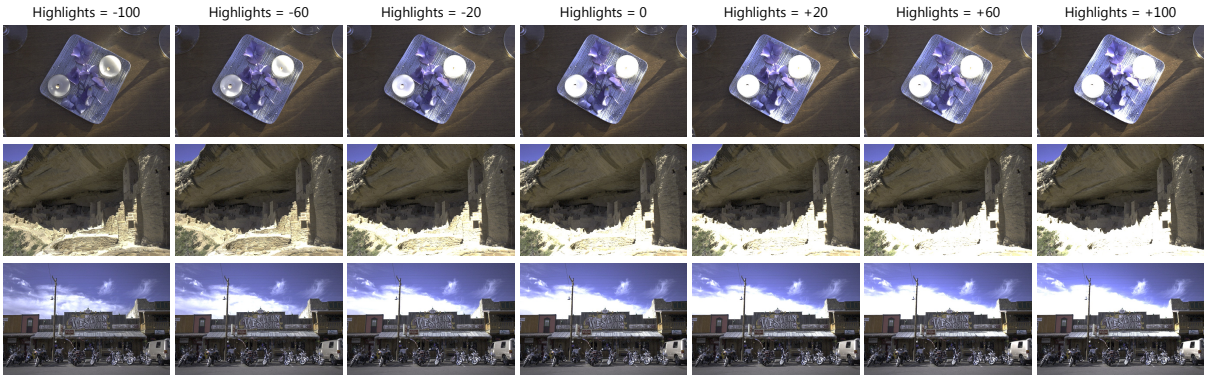

Fig. 2. Highlights Tap-outs: The highlights slider affects the dynamic range for the bright regions of an image. Observe the candles in the top row, the sunny areas in the middle row, and the clouds in the bottom row. Note that the tap-outs have been equally brightened for visualization.

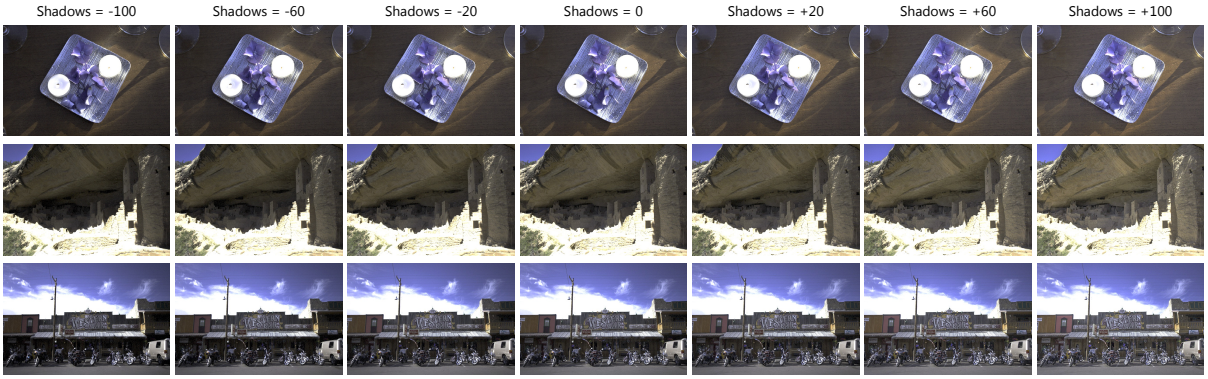

Fig. 3. Shadows Tap-outs: The shadows slider affects the dynamic range for the dark regions of an image. Observe the shadows in the top row, the structures under the cliff in the middle row, and the walls beneath the ceiling in the bottom row. Note that the tap-outs have been equally brightened for visualization.

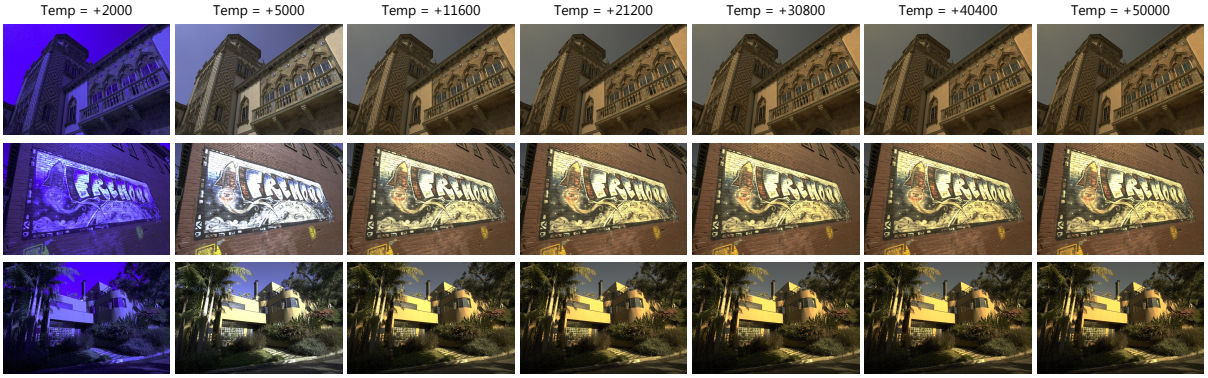

Fig. 4. Temperature Tap-outs: The temperature slider changes the white balance of an image following a non-linear Kelvin scale. The temperature is usually set to be around +5000. Note that the tap-outs have been equally brightened for visualization.

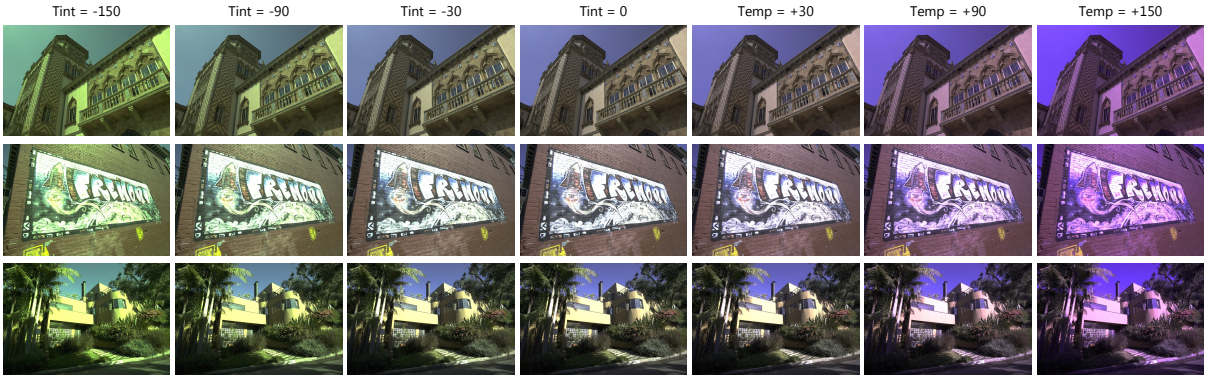

Fig. 5. Tint Tap-outs: The tint slider changes the white balance along a vector that is orthogonal to the temperature curve. Note that the tap-outs have been equally brightened for visualization.

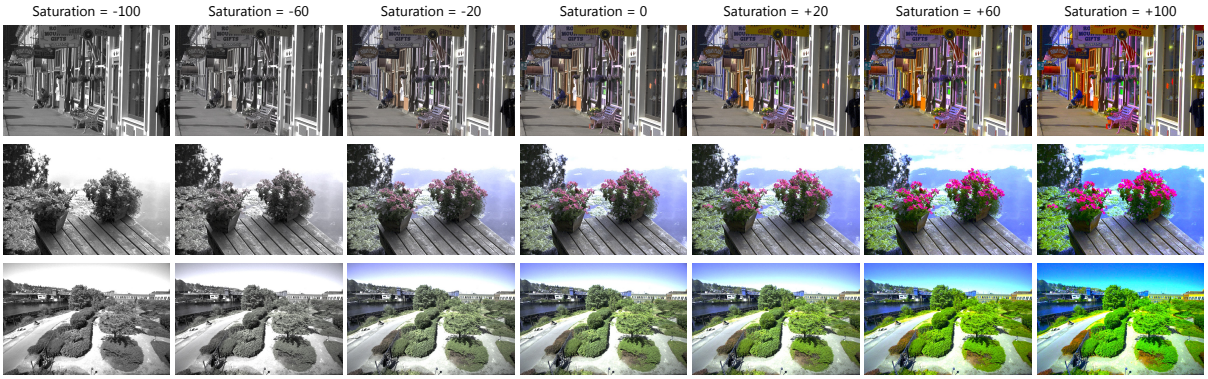

Fig. 6. Saturation Tap-outs: The saturation slider affects the saturation of colors in the image. Low values of saturation removes color while high values of saturation accentuate color. Note that the tap-outs have been equally brightened for visualization.

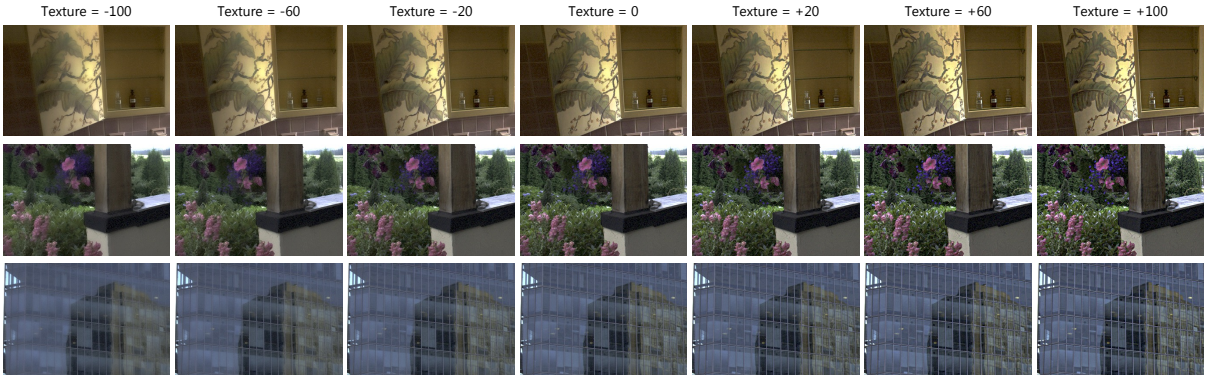

Fig. 7. Texture Tap-outs: The texture slider sharpens or blurs the entire image. Note that the tap-outs have been equally brightened for visualization.

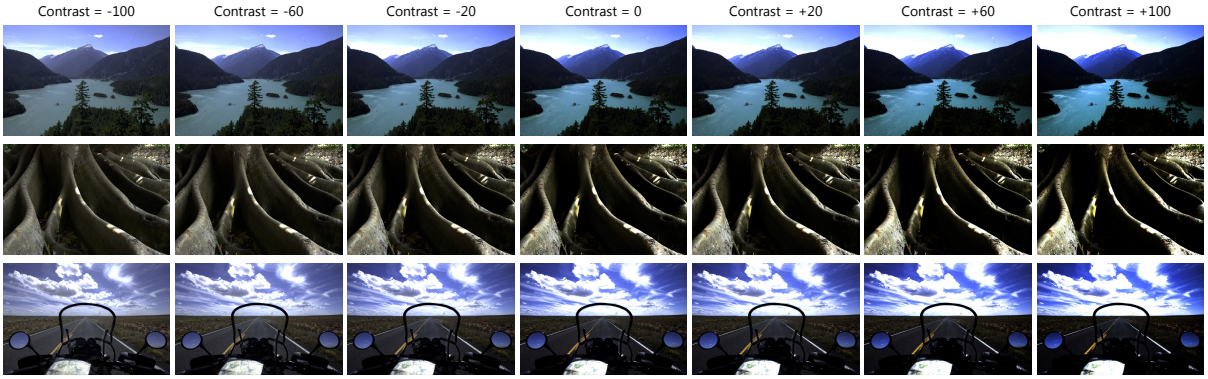

Fig. 8. Contrast Tap-outs: The contrast slider affects the contrast of the image. Note that the tap-outs have been equally brightened for visualization.

2 DIFFERENTIABLE PHOTO-FINISHER NETWORK ARCHITECTURE

We describe our network architecture in more detail in Tables 1 and 2.

For camera types we used the Canon EOS DIGITAL REBEL, Canon Powershot S90, Phase One P65+, and the iPhone 13. The relevant specifications for color balancing are shown in Tables 3, 4, 5, 6. In addition to these specifications, there are additional specifications such as the Baseline Exposure and the camera profiles. All of this metadata can be extracted from the DNG.

Table 1. Neural pointwise architecture. In the table, "conv-n(a)-k(b)" represents a convolution layer with a output channels using a $b \times b$ kernel. Each "Leaky Relu" has slope 0.2.

Layer	Convolution layer	Activation
0	conv-n512-k1	Leaky Relu
1	conv-n256-k1	Leaky Relu
2	conv-n3-k1	-

Table 2. Neural areawise architecture. In the table, "conv-n(a)-k(b)" represents a convolution layer with a output channels using a $b \times b$ kernel. Each "Leaky Relu" has slope 0.2.

Layer	Convolution layer	Activation
0	conv-n64-k3	Leaky Relu
1	conv-n64-k3	Leaky Relu
2	conv-n64-k3	Leaky Relu
3	conv-n32-k3	Leaky Relu
4	conv-n3-k3	-

Table 3. Canon EOS DIGITAL REBEL Specifications

Calibration Illuminant 1	Calibration Illuminant 2	Color Matrix 1	Color Matrix 2	Forward Matrix 1	Forward Matrix 2
2850.0	6500.0	$\begin{bmatrix} 0.9120 & -0.3127 & -0.0453 \\ -0.6868 & 1.4138 & 0.3011 \\ -0.1708 & 0.2368 & 0.9501 \end{bmatrix}$	$\begin{bmatrix} 0.8250 & -0.2044 & -0.1127 \\ -0.8092 & 1.5606 & 0.2664 \\ -0.2893 & 0.3453 & 0.8348 \end{bmatrix}$	$\begin{bmatrix} 0.6792 & 0.2075 & 0.0776 \\ 0.3336 & 0.8297 & -0.1633 \\ 0.0678 & -0.2033 & 0.9606 \end{bmatrix}$	$\begin{bmatrix} 0.6792 & 0.2075 & 0.0776 \\ 0.3336 & 0.8297 & -0.1633 \\ 0.0678 & -0.2033 & 0.9606 \end{bmatrix}$

Table 4. Canon Powershot S90 Specifications

Calibration Illuminant 1	Calibration Illuminant 2	Color Matrix 1	Color Matrix 2	Forward Matrix 1	Forward Matrix 2
2850.0	6500.0	$\begin{bmatrix} 2.0380 & -1.4279 & 0.0975 \\ 0.0145 & 0.7942 & 0.2325 \\ 0.0374 & 0.0172 & 0.5441 \end{bmatrix}$	$\begin{bmatrix} 1.2374 & -0.5016 & -0.1049 \\ -0.1677 & 0.9902 & 0.2078 \\ -0.0083 & 0.0852 & 0.4683 \end{bmatrix}$	$\begin{bmatrix} 0.3585 & 0.5755 & 0.0303 \\ -0.0481 & 1.1440 & -0.0959 \\ -0.0480 & -0.2733 & 1.1464 \end{bmatrix}$	$\begin{bmatrix} 0.5118 & 0.5246 & -0.0720 \\ 0.0974 & 1.1959 & -0.2933 \\ -0.0082 & -0.1431 & 0.9764 \end{bmatrix}$

Table 5. Phase One P65+ Specifications

Calibration Illuminant 1	Calibration Illuminant 2	Color Matrix 1			Color Matrix 2			Forward Matrix 1			Forward Matrix 2		
2850.0	6500.0	0.7756	0.0244	−0.0761	0.8035	0.0435	−0.0962	0.8927	−0.1853	0.2569	0.9014	−0.0632	0.1261
		−0.5300	1.2774	0.2832	−0.6001	1.3872	0.2320	0.3743	0.7114	−0.0857	0.3949	0.8045	−0.1994
		−0.0881	0.2622	0.5858	−0.1159	0.3065	0.5434	−0.0852	−0.7970	1.7073	−0.0239	−0.3454	1.1945

Table 6. iPhone 13 Specifications

Calibration Illuminant 1	Calibration Illuminant 2	Color Matrix 1			Color Matrix 2			Forward Matrix 1	Forward Matrix 2
2850.0	6500.0	1.3033	−0.6401	−0.2454	1.0197	−0.3937	−0.1386	N/A	N/A
		−0.4121	1.4405	−0.0522	−0.4354	1.3496	0.0852		
		−0.0387	0.1579	0.5390	−0.1087	0.2457	0.4132		

3 SLIDER REGRESSION VALIDATION ON COMMERCIAL ISPS

We show additional examples of finding sliders that will cause ACR to match the look of commercial ISPs. Results for the Canon EOS Digital Rebel camera are shown in Figure 9. Results for the iPhone 12 Pro Max are shown in Figure 10. Results for the Pixel 3 are shown in Figure 11.

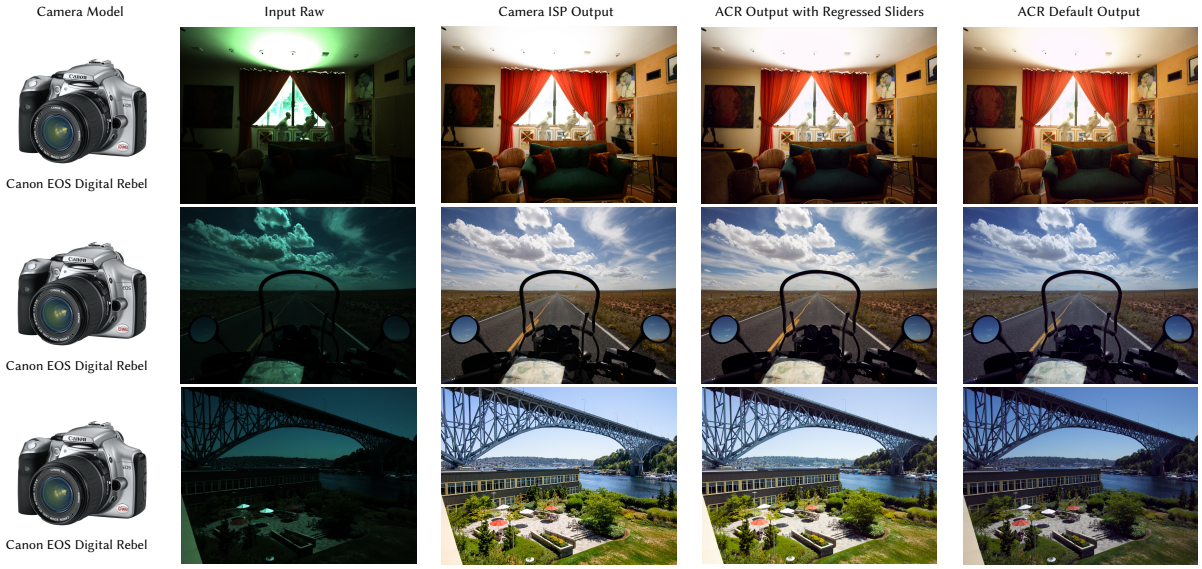

Fig. 9. Additional results on camera ISP approximation with ACR. These results are for a Canon EOS Digital Rebel camera.

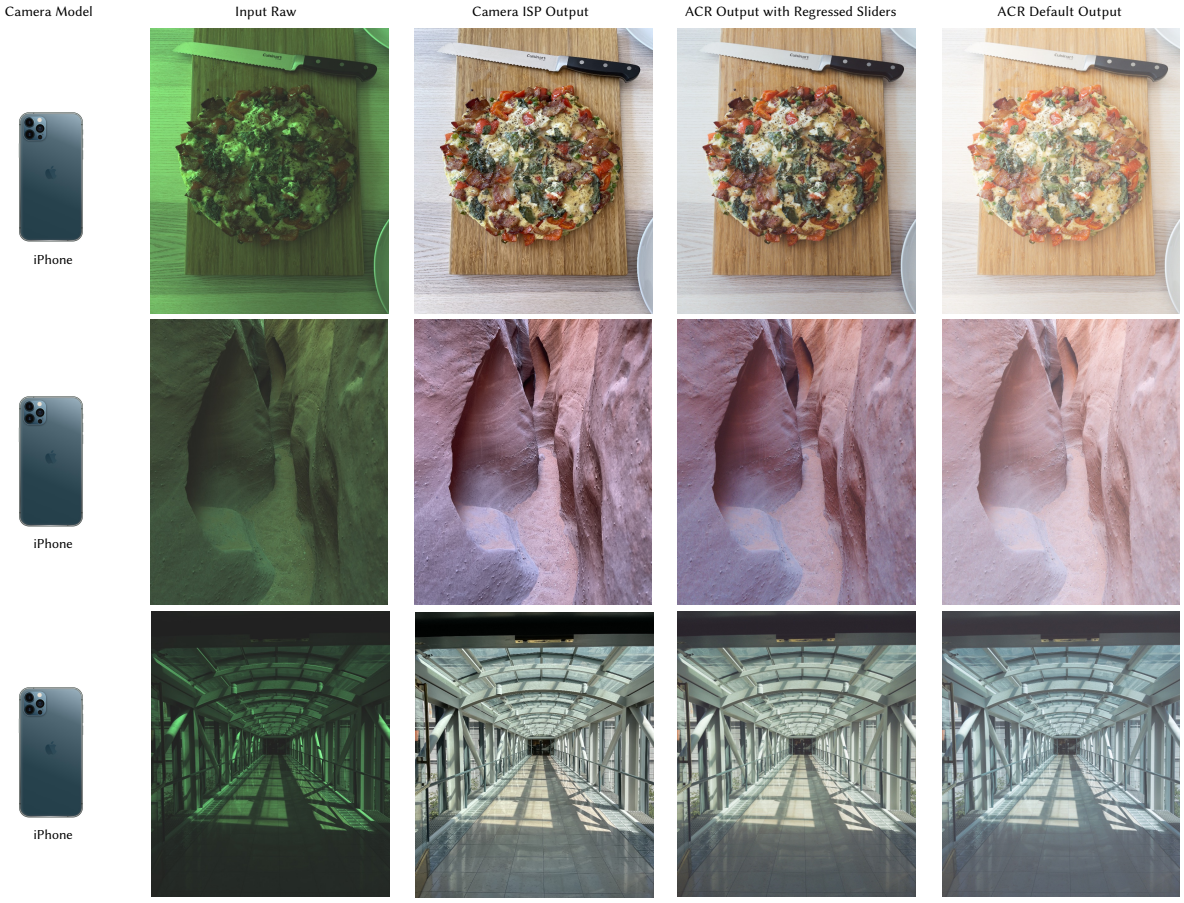

Fig. 10. Additional results on camera ISP approximation with ACR. These results are for a iPhone 12 Pro Max.

Fig. 11. Additional results on camera ISP approximation with ACR. These results are for a Pixel 3.

4 SLIDER REGRESSION COMPARISON AGAINST 0TH-ORDER OPTIMIZATION METHODS

For the slider regression experiments described in the Main Document, we set the number of queries to ACR to be a constant T for all methods for a fair comparison. That means that when first-order optimization is run we query our ACR proxy at most T times. Similarly, CMAES and BayesOpt are allowed to query the true ACR at most T times. For the benchmark shown in the Main Document we set $T = 200$. We ran our first-order regression approach using PyTorch on Nvidia GPUs. For CMAES and BayesOpt we ran them by querying the optimized build of ACR on CPU, specifically on a MacBook Pro with an Intel i9 core. We implemented CMAES using the pycma Github repository [Hansen et al. 2019]. The slider ranges are normalized to [0,1], the initial guess is set to all 0.5, and the standard deviation is set to 0.25. We implemented BayesOpt using the BayesianOptimization Github repository [Nogueira 2014]. We used 10 initial random guesses. Additional qualitative results are shown in Figures 12 and 13.

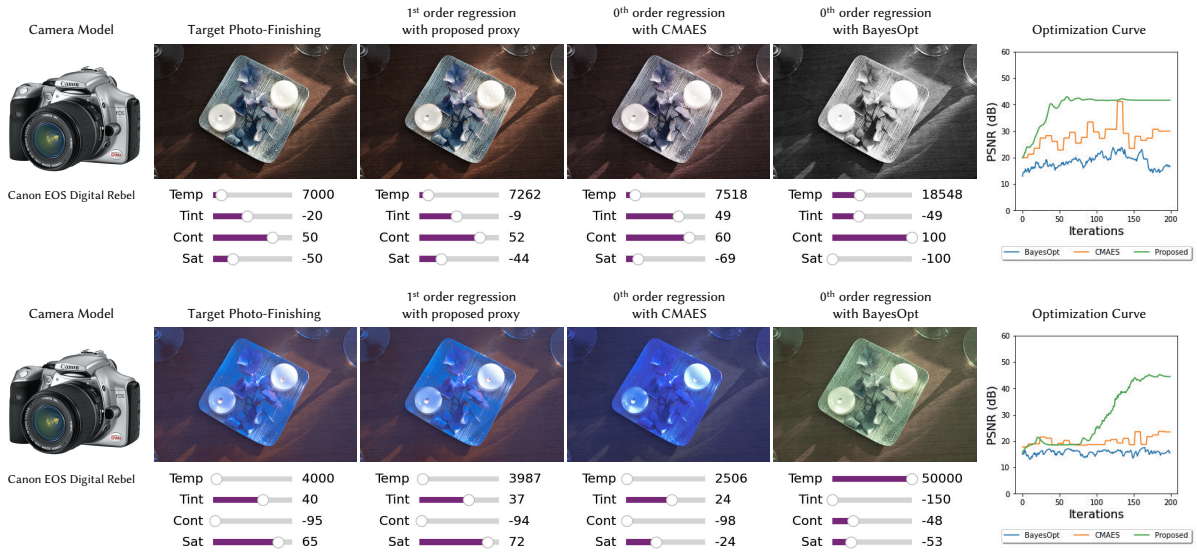

Fig. 12. Additional comparisons on slider regression against 0th-order methods. These results are for a Canon EOS Digital Rebel camera.

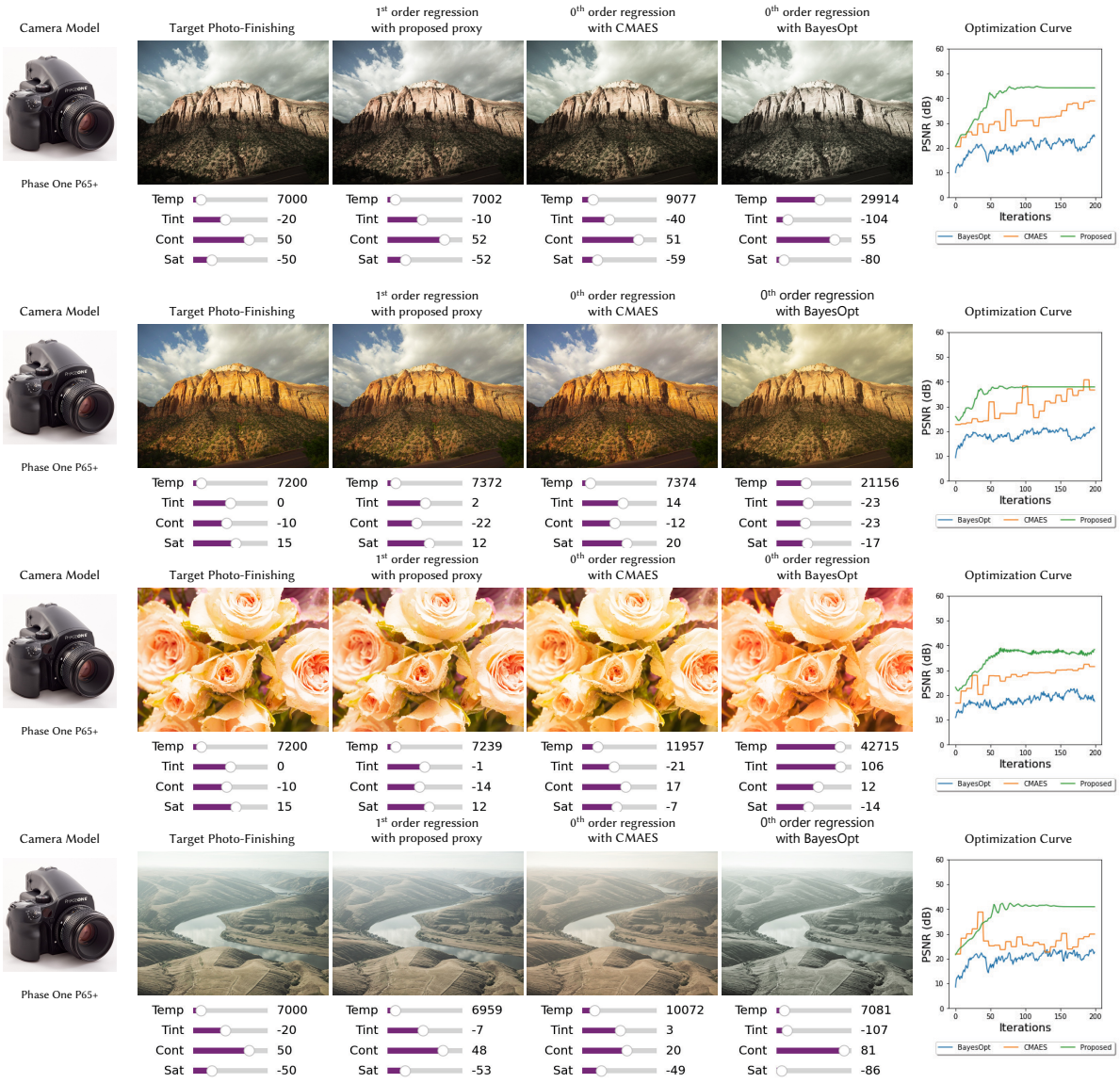

Fig. 13. Additional comparisons on slider regression against 0th-order methods. These results are for a Phase One P65+ camera.

5 COMPARISONS AGAINST RECONFIGISP AND MONOLITHIC/MULTI-STAGE PROXIES

In this section, we compare the proposed method against the recent method of ReconfigISP [Yu et al. 2021]. We highlight that ReconfigISP aims to design new ISPs by rearranging blocks through the use of differentiable proxies. Although ReconfigISP tackles a different problem and does not address pipelines of similar complexity to ACR, we ran an additional experiment to compare our full method and the proxy architecture in isolation against ReconfigISP.

Specifically, we perform two comparison experiments. The first experiment compares the fitting quality of the proposed proxy model compared to ReconfigISP's SRCNN proxy on ACR's white balance and saturation modules. The results are shown in Figure 15 and Figure 14 and the quantitative PSNR scores are reported in Table 7. Although the authors of ReconfigISP proposed to use the SRCNN architecture as a "one-size-fits-all" architecture, it is unable to accurately approximate the complex modules that are used in industrial ISPs such as ACR. We observed that the approximation accuracy achieved is substantially lower than that which is achieved by our catered architecture design, both quantitatively and qualitatively.

Our second experiment is a comparison on slider regression where we assess the entire proposed method with and without the proxy architecture proposed by ReconfigISP [Yu et al. 2021]. To this end, we replace the proxies in our ACR pipeline proxy with ReconfigISP's SRCNN proxies. Then we perform the same slider regression experiment as described in Section 6 of the Main Document. See Figure 16 and Figure 17 for qualitative results and see Table 8 for quantitative PSNR scores. The low approximation accuracy achieved by the ReconfigISP approach in turn result in poor performance in downstream tasks such as this slider regression task. In general, we found that useful slider regression was not possible with the SRCNN proxies, please see the figures for examples. In contrast, our catered design provides the higher approximation accuracy needed for this slider regression experiment.

Overall, while the use of differentiable proxies itself has been investigated extensively in the field of neural architecture search, these additional experiments validate that the specific proxy models and optimization scheme proposed in our work are essential choices in the proposed method. We confirm that the proxy from ReconfigISP in isolation or used for slider regression end-to-end fail to approximate complex image processing pipelines.

Table 7. **Module fitting comparison against ReconfigISP.** Comparison between proposed proxy architecture and SRCNN architecture of ReconfigISP [Yu et al. 2021]. We report over 10 dB PSNR improvement on the white balance and saturation modules. This in turn results in better gradients for downstream applications such as slider regression, see Figure 16 and Figure 17 and Table 8.

	White Balance PSNR (dB)	Saturation PSNR (dB)
Proposed	**44.5**	**56.4**
ReconfigISP [Yu et al. 2021]	32.5	31.6

Table 8. **Slider regression comparison against ReconfigISP and Monolithic U-Net.** We compare slider regression performance when using our proxies to model ACR versus using ReconfigISP's SRCNN proxies/a single monolithic U-Net to model ACR. Due to the lower approximation accuracy of the aforementioned methods, neither achieves the same slider regression accuracy as we do with our method.

	Regression Accuracy PSNR (dB)
Proposed	**43.4**
ReconfigISP [Yu et al. 2021]	17.0
Tseng et al. [2019]	11.5

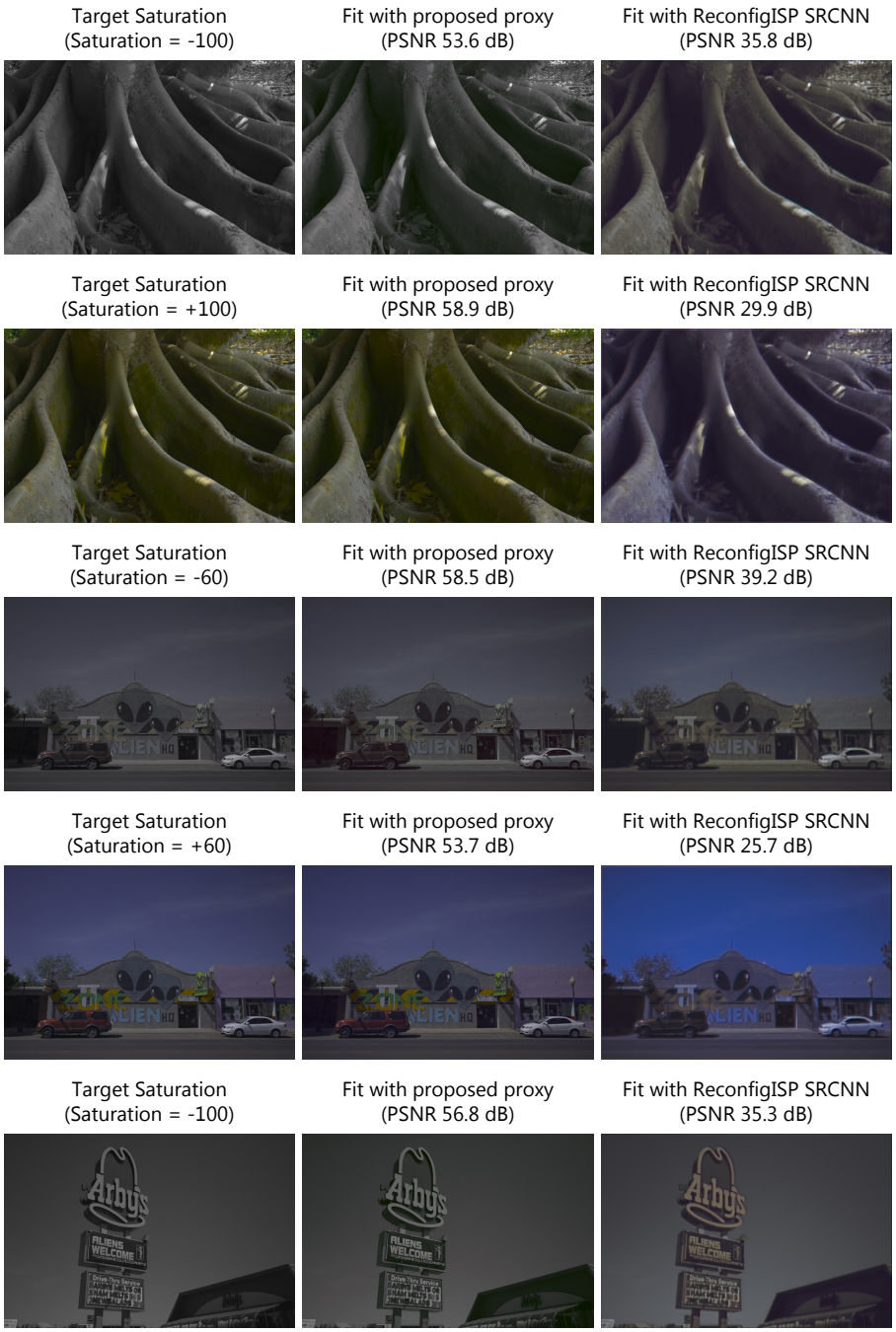

Fig. 14. Saturation comparisons on module fitting against ReconfigISP's SRCNN proxy. The SRCNN architecture is unable to accurately model the saturation operation within ACR.

Fig. 15. White Balance comparisons on module fitting against ReconfigISP's SRCNN proxy. The SRCNN architecture is unable to accurately model the complex white balancing operation within ACR.

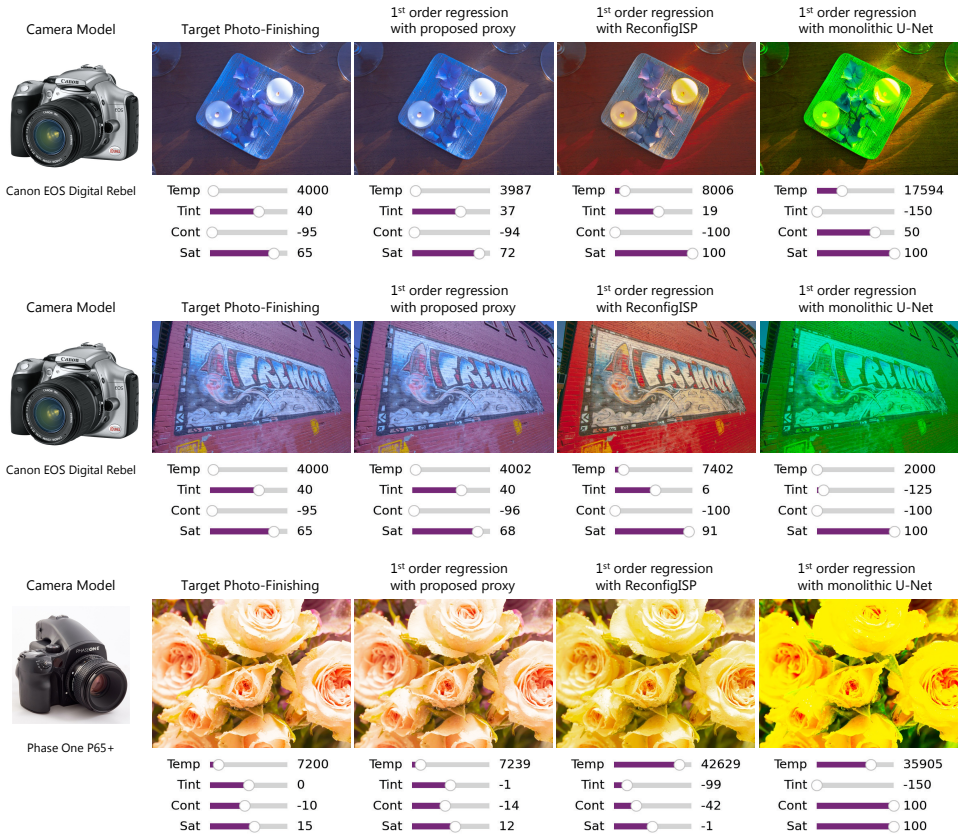

Fig. 16. Additional comparisons on slider regression against regression with ReconfigISP's SRCNN proxy/monolithic U-Net. The low approximation accuracy obtained when using the alternative architectures results in lower slider regression performance.

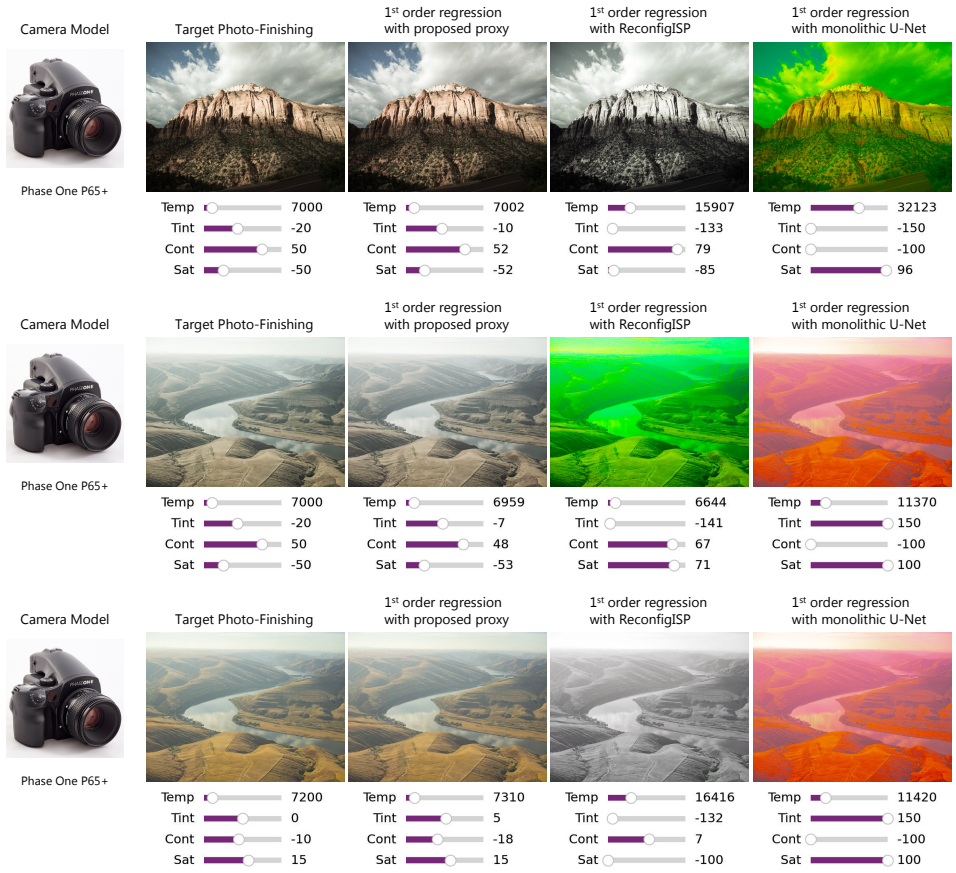

Fig. 17. Additional comparisons on slider regression against regression with ReconfigISP's SRCNN proxy/monolithic U-Net. The low approximation accuracy obtained when using the alternative architectures results in lower slider regression performance.

6 COMPARISONS AGAINST MONOLITHIC PROXIES

Next, we perform slider regression comparisons against a monolithic U-Net network [Tseng et al. 2019]. The low approximation accuracy achieved with the monolithic network also results in poor slider regression performance for the same reasons as with the SRCNN proxies from the previous section. See Figure 16 and Figure 17 for qualitative results and see Table 8 for quantitative PSNR scores. These results confirm that the regressed sliders deviate far from the target and, indeed, the vanishing samples problem discussed in the manuscript necessitates the proposed multi-stage approach.

7 RAW STYLE TRANSFER

Figure 18 and Figure 19 show additional movie style transfer results on still photographs. The slider encoder, which is the neural network that predicts the parameters of our differentiable finishing pipeline, is trained separately for each movie style. See Figure 20 for architecture details. Each collection of style images consists of 14-29 handpicked movie frames, but we only show 9 representative frames from each collection in the figure.

Parameter Linearization. When predicting sliders to match a target style, it is necessary to linearize the effect of the sliders on the final output image. For example, the temperature slider spans a range of temperatures from 2000 K to 50000 K. The effect of the slider value within this range is not linear: A change from 3000 K to 4000 K leads to a much larger change in the output than a change from, e.g., 10000 K to 11000 K, both in terms of perceived color change and absolute pixel value change. Regressing sliders for a certain style without linearizing the slider leads to larger gradients when the color temperature large (warm), and to smaller gradients when the color temperature is small (cold). Intuitively, the partial derivative at a single pixel

$$\left.\frac{\partial \mathbf{S}}{\partial \mathbf{I}_{\mathbf{S}}(x,y)}\right|_{\mathbf{S}=\mathbf{S}_0} \tag{1}$$

for a given slider setting $\mathbf{S}_0$ becomes very large when a large change in the slider position $\mathbf{S}$ corresponds to only a small change in the output image pixel $\mathbf{I}_{\mathbf{S}}(x,y)$. As a result, training is less stable and the learned temperature slider is often biased towards colder colors. To combat this, we empirically sample temperature slider values that are roughly equidistant in perceived color of the finished output. We then fit a 4^{th}-degree polynomial to these values and use this polynomial as input encoding for the proxy's temperature slider (Figure 21).

Style Loss. For our style loss $\mathcal{L}_{\text{STYLE}} = \mathcal{L}_{\text{GRAM}} + \lambda_1 \mathcal{L}_{\text{LUMA}} + \lambda_2 \mathcal{L}_{\text{CHROMA}}$ we found $\lambda_1 = 1000$ and $\lambda_2 = 500$ to yield the best results.

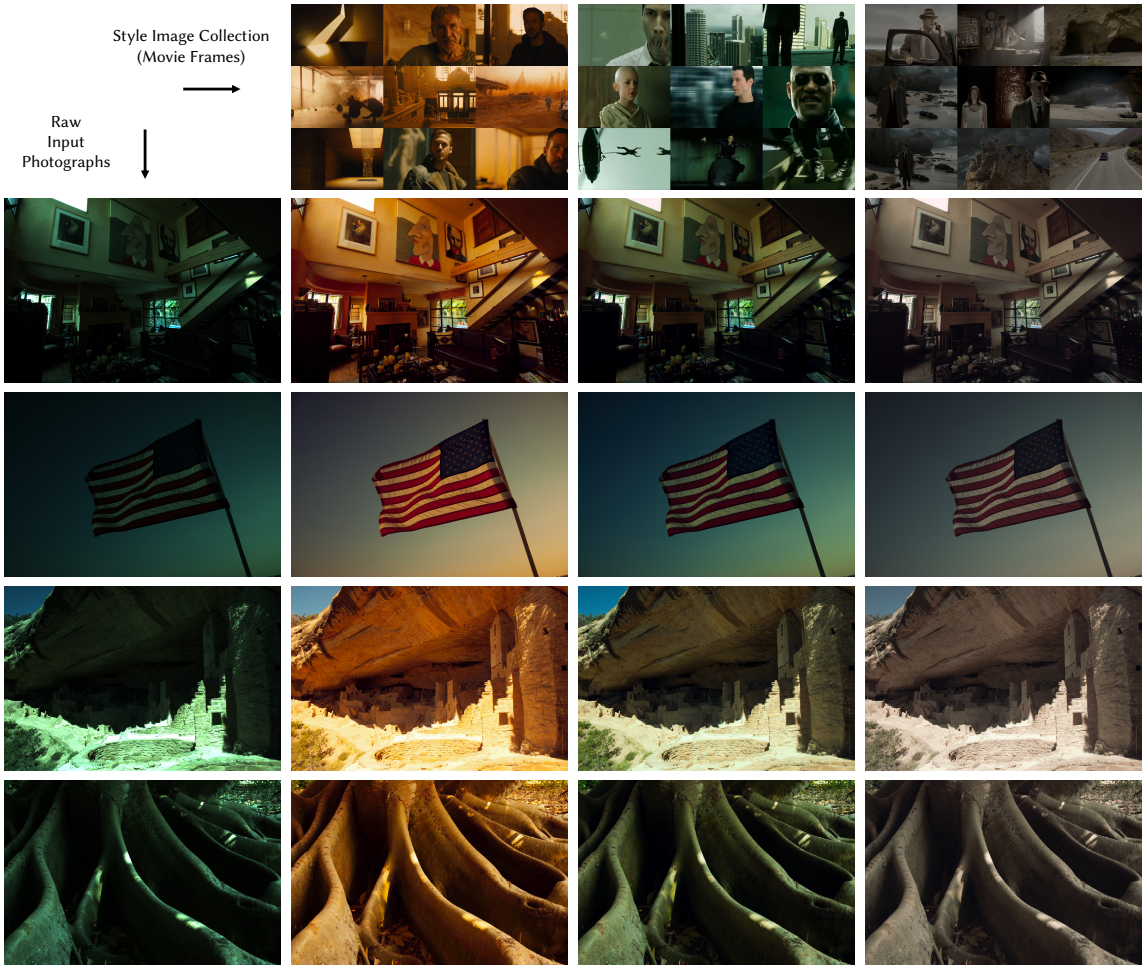

Fig. 18. Additional results (a) on Raw Style Transfer. The left column shows the raw input images, while the other columns show the style transfer output for different movie styles. The top row shows a subset of images from the collection of style images that are used to train the slider encoder. The encoder is trained separately for each style, while the differentiable proxy is pre-trained and fixed during both training and inference.

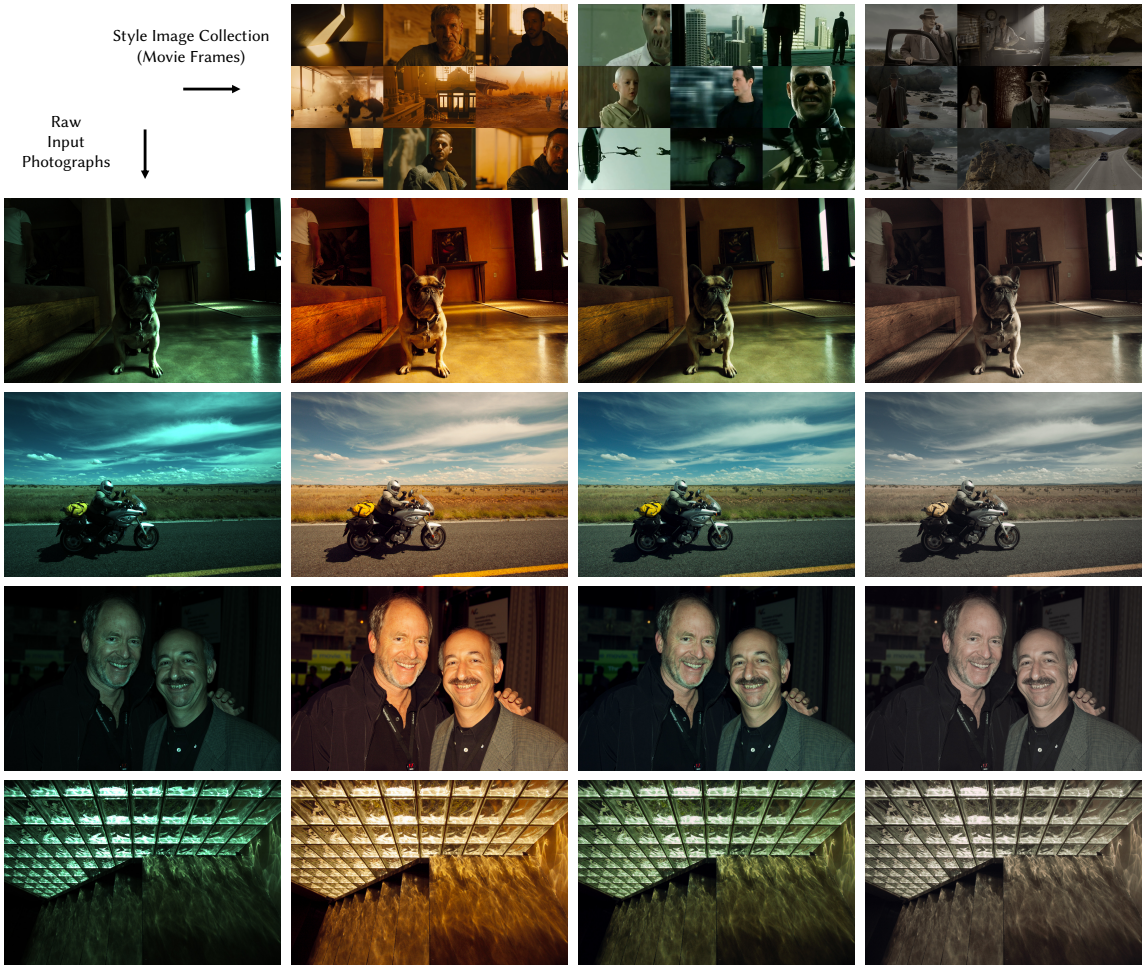

Fig. 19. Additional results (b) on Raw Style Transfer. The left column shows the raw input images, while the other columns show the style transfer output for different movie styles. The top row shows a subset of images from the collection of style images that are used to train the slider encoder. The encoder is trained separately for each style, while the differentiable proxy is pre-trained and fixed during both training and inference.

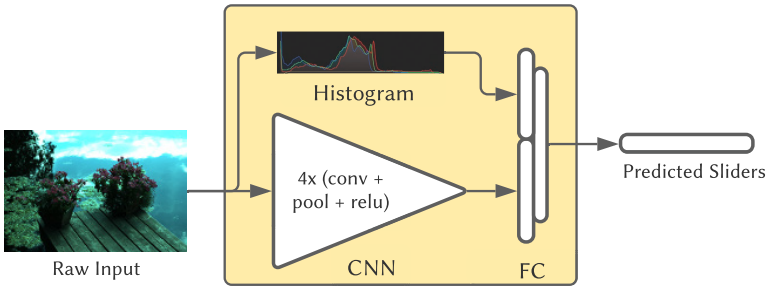

Fig. 20. Slider Encoder architecture. The encoder's task is to predict a set of sliders (i.e., proxy parameters) to match the target style. We use a lightweight architecture consisting of a shallow CNN with 4 strided convolutional and pooling layers, followed by two fully connected layers. The input to the first fully connected layer is the concatenation of the CNN's output features map and a color histogram calculated directly on the raw image.

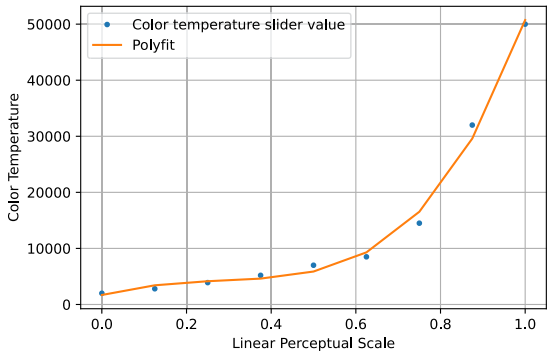

Fig. 21. Linearizing the non-linear temperature slider effect.

8 ADVERSARIAL PHOTO-FINISHING

In this section, we provide additional qualitative results for adversarial photo-finishing in Figure 22. As discussed in the Main Document, we leverage the learned proxy function to design an adversarial attack that fools a classifier run on images photo-finished by a given photographer S_1, while leaving classification on the same raw image, but finished by other photographers S_2, unaltered. As shown in the figure, the same raw perturbation δ is transformed to a much stronger adversarial pattern for pipeline S_1 than pipeline S_2, especially near edges around the object of interest, which explains how the attack deceives S_1 while at the same time maintaining the performance for S_2.

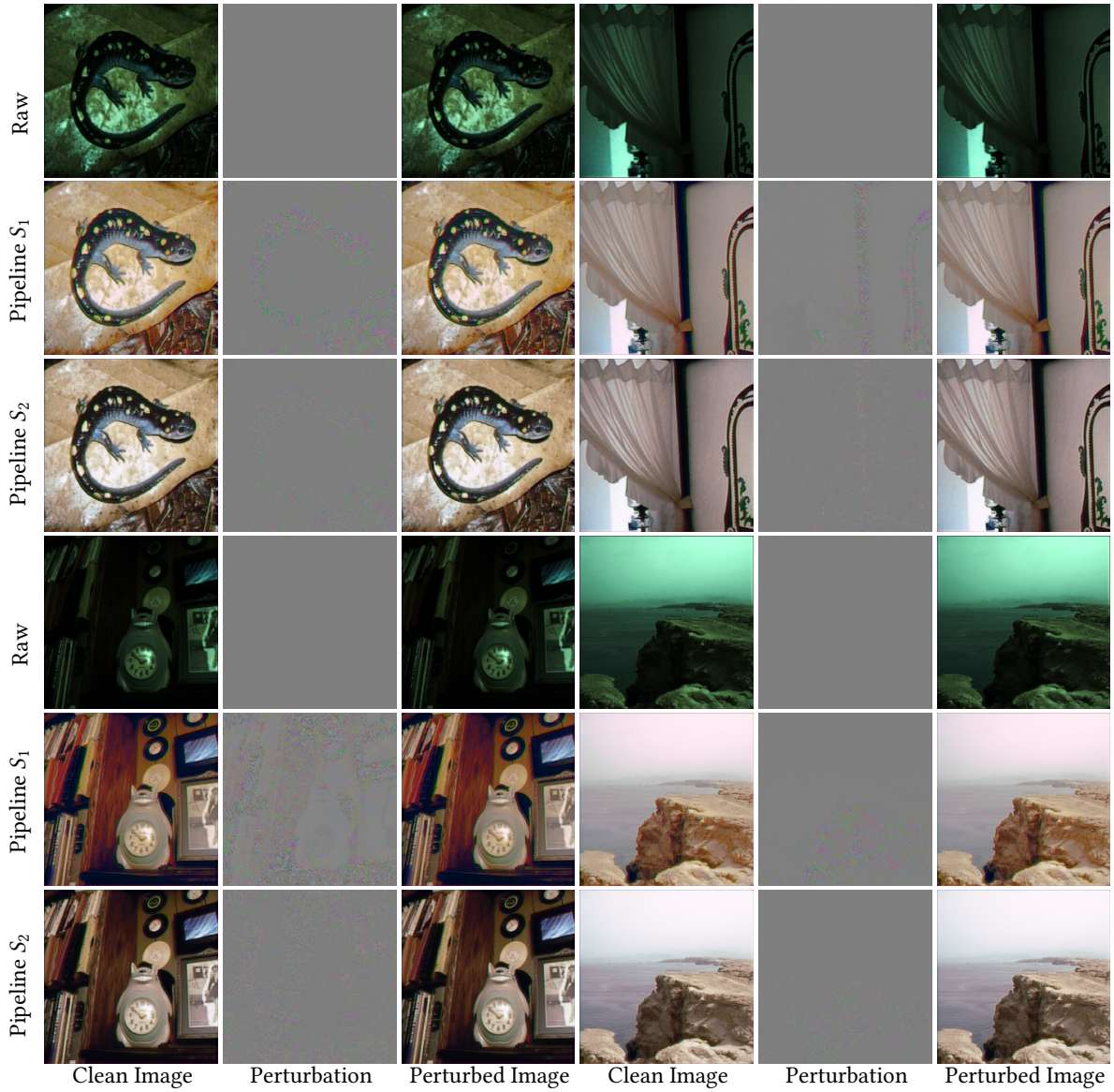

Fig. 22. **Adversarial Photo-Finishing.** Using the proposed proxy function, we learn an adversarial raw perturbation δ that map to a stronger adversarial pattern for pipeline S_1 than pipeline S_2, especially at object edges.

9 GENERALIZATION TO OTHER PIPELINES

Our proposed method generalizes beyond ACR to other photo finishing pipelines. We demonstrate this by modeling the Darktable pipeline, training neural network proxies of a representative set of processing steps. Just like with ACR, a framework must be devised that interfaces with the software to generate training data, automatically processing raw files through a predetermined sequence of stages while tapping out intermediaries from between those stages. Otherwise, the overall network architectures remain unchanged.

The Darktable pipeline is divided into modules, roughly analogous to processing steps in ACR. Photos are finished by selecting and arranging modules into a precise, user-defined order, called the pixelpipe, and processing input files in that order. We generated training data using a pixelpipe of 12 modules, chosen either because they are permanent fixtures of the Darktable pipeline that should not be removed, or because they are ubiquitous photo finishing operations that generalize well to other pipelines. The 12 modules in the pixelpipe are defined as follows:

(1) *Rawprepare:* This module is automatically active for any raw input; it simply defines camera-specific black and white point parameters before demosaicing.

(2) *Demosaic:* Sensor cells in digital cameras cannot detect color, only lightness. Colors are recorded by covering cells with color filters, arranged in a mosaic pattern (often the Bayer filter array). Full color images can then be reconstructed by applying algorithms to "demosaic" these filter recordings by interpolating from neighboring cells. This module allows the user to switch between different demosaicking algorithms.

(3) *Colorin:* This module remaps from the color space of the image source to a common working color space that can be shared between other modules as the image is processed.

(4) *Colorout:* This module remaps the image color space to sRGB before export.

(5) *Gamma:* This module applies a nonlinear operation that takes advantage of the nonlinear perception of light by humans to adjust luminance or tristimulus values.

(6) *Temperature:* Temperature, along with tint, are both used to adjust the white balance in an image by adjusting for a hue along the blue-amber axis or the green-magenta axis, respectively. This module allows for the direct setting of RGB color channel coefficients so as to change an image's white balance.

(7) *Highlights*: Highlight clipping occurs when digital cameras capture input that is more intense than the camera's maximum clipping threshold - all input above this threshold is clipped down to the maximum value. This module attempts to reconstruct the lost information in pixels where one or more RGB channels has been clipped. The user can choose between 3 different methods of estimating the original photographed scene.

(8) *Sharpen:* This module sharpens input images by using an UnSharp mask to increase the contrast around edges. Users can control the radius of the gaussian blur used with the mask, the amount of sharpening done, and the threshold contrast difference above which sharpening occurs.

(9) *Filmicrgb:* This module is designed to remap the tonal range of a captured scene to the range of the given display in order to emulate classic film. Users have control over a myriad of sliders, grouped into the parameters of the input scene, highlight reconstruction, artistic enhancements, and parameters of the output display.

(10) *Exposure:* This module can be used to adjust the overall brightness of an image. Users can toggle between manual mode, where they set exposure, black level, and clipping threshold parameters themselves, or

automatic mode, where exposure parameters are automatically set by analyzing the image's histogram and adjusting such that a specified percentile is shifted to a given target zone.

(11) *Flip:* This module flips the input image along either the horizontal, vertical, or both axes.

(12) *Colorbalancergb:* This is an advanced color-grading module that can perform precise operations on the RGB color space. It has tunable parameters for hue shift, global vibrance, contrast, linear chroma grading, perceptual saturation grading, perceptual brilliance grading, masking, and more.

In the absence of a developer tool to access intermediary image tap-outs, we developed a Python wrapper function for interfacing with the Darktable pipeline and modified its source code to dump intermediaries. Similar to other pipelines, Darktable is a nondestructive photo finisher, storing changes entirely within XMP sidecar files; these represent a sequence of operations that can be performed on any raw file to generate a finished output image. More precisely, XMP files encode a pixelpipe, specifying the order of processing steps as well as their slider values and blend parameters. By programmatically generating XMP files and passing them into Darktable with the same input images, it becomes possible to automatically generate large training sets for proxy networks while only interfacing with the pipeline via sidecar files. Therefore, our Python wrapper is very straightforward. We create a format string that corresponds to the XMP of our 12-module pixelpipe, with all salient parameters set as variables. The script then sweeps through all of the relevant slider values for each step, inserting the parameter values into the string. Darktable encodes parameters for each module by concatenating their float values together and converting to hexadecimal. The XMP format string also contains a binary "enabled" flag for each of the process modules, which too is set to a variable for non-essential stages. This allows our script to progressively enable our pixelpipe stages one-by-one, rendering all intermediary images. Darktable has a command-line interface (CLI) that can process a raw file given an XMP sidecar without using the GUI. Once the format string is complete, it is written to a temporary file that is passed as input to the CLI. See Figure 23 for examples of function outputs. The wrapper function was then tested on both a Windows and Linux build of Darktable. The former was done locally, while the latter was made possible by creating a Docker image of a Linux system and building Darktable in the image.

However, Darktable's source code does not support the tap-out of intermediary images between processing modules. We introduce this capability by creating a new function that dumps out images between steps, and modify the C files for each module in our chosen pixelpipe to call this function after they process their input. We chose to dump the intermediaries as ImageStack TMP files, as they could encode Darktable's float32 buffer format; these are later converted to TIFF files by a separate Python script.

REFERENCES

Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. 2019. CMA-ES/pycma on Github. https://doi.org/10.5281/zenodo.2559634

Fernando Nogueira. 2014. Bayesian Optimization: Open Source Constrained Global Optimization Tool for Python. https://github.com/fmfn/BayesianOptimization

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.

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.

Fig. 23. Intermediary tap-outs from the Darktable CLI Python wrapper function, sweeping through the contrast and sharpness slider range.