# Supplementary Information
# Hyperparameter Optimization in Black-box Image Processing using Differentiable Proxies

Ethan Tseng, Felix Yu, and Yuting Yang – Princeton University
Fahim Mannan and Karl St. Arnaud – Algolux
Derek Nowrouzezahrai – McGill University
Jean-François Lalonde – Université Laval
Felix Heide – Princeton University and Algolux

In this document we provide additional discussion and results in support of the primary text.

## 1 Setup Details

Here we describe the general framework for optimizing hyperparameters for a given ISP.

**Dataset.** The dataset requires three individual components. An example is also shown in Figure S.1. Images are taken from the SIDD Dataset [Abdelhamed et al. 2018].

1) Unprocessed input images $\mathbf{I}$: Unprocessed images that are the input into the ISP system.

2) ISP Output + hyperparameters $\mathbf{O}_{\text{ISP}} + \mathcal{P}$: Images that have been outputted by the ISP. These image are generated by taking input images, randomly sampling hyperparameters that we wish to optimize, and passing them through the ISP with these hyperparameters configured.

3) Ground truth images $\mathbf{I}_{\text{GT}}$: for each raw input image, there also needs to be a corresponding ground truth image. This image contains characteristics that we are interested in capturing using our hyperparameters. For example, our experiments have low noise images captured through long exposure, and therefore our goal is to find a set of hyperparameters such that when applied to the ISP, gives us high-SNR results.

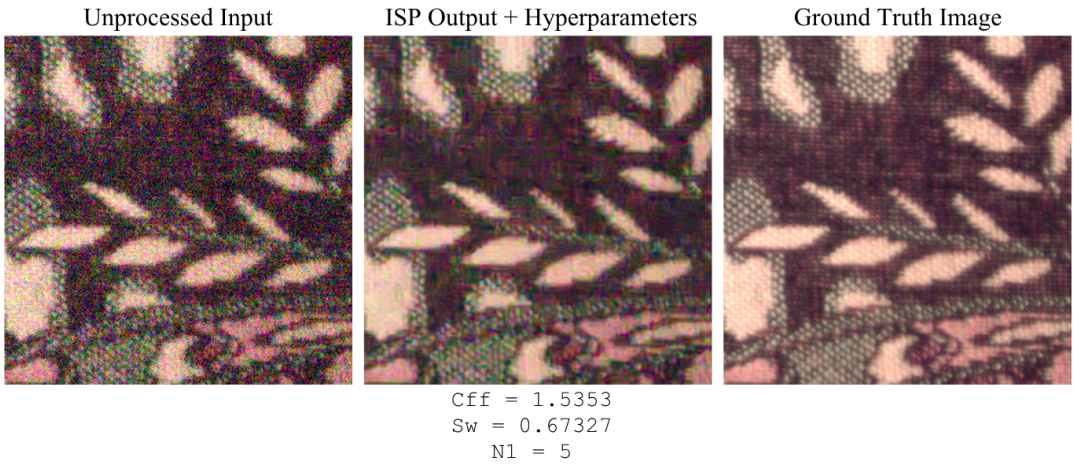



**Figure S.1:** *Example of the three components needed for a single sample of data. Left: Unprocessed noisy image. Middle: Denoised image using the BM3D algorithm, with the specified hyperparameters. Right: Ground truth low noise image.*

**Stage 1: Proxy Model Training.** The first stage of the pipeline is to train the differentiable proxy function. As stated in the paper, we use a convolutional neural network to approximate the ISP in question, and aim to minimize $\mathcal{W}$ with respect to $\mathcal{L}_{\text{STAGE 1}}$. In order to do this, we use the first two components of the dataset: the unprocessed images $\mathbf{I}$, as well as the ISP output ($\mathbf{O}_{\text{ISP}}$) + hyperparameters ($\mathcal{P}$). The unprocessed images are passed through the proxy model, with hyperparameter channels concatenated on as well, in order to produce an output image ($\mathbf{O}_{\text{PROXY}}$), which can then be used to find $\mathcal{W}^*$. Additional details on the exact setup used to train the proxy models for each of our experiments are found in the later sections of the this document. Figure S.2 also illustrates this process.

**Stage 2: Hyperparameter Optimization.** The second stage of the pipeline is to optimize the hyperparameters so that our proxy model output matches the ground truth as closely as possible. We fix the weights in our CNN proxy model, and initialize a set of hyperparameters $\mathcal{P}$ to be varied. In our experients, the initialization is randomly sampled from a given search space. From there, we feed in input images, along with $\mathcal{P}$ appended as additional channels, and obtain an output image. This output image is then fed in to our Stage 2 objective function $\mathcal{L}_{\text{TASK}}$ along with the ground truth image. Through this, we can calculate the gradients and backpropogate in to the hyperparameters.

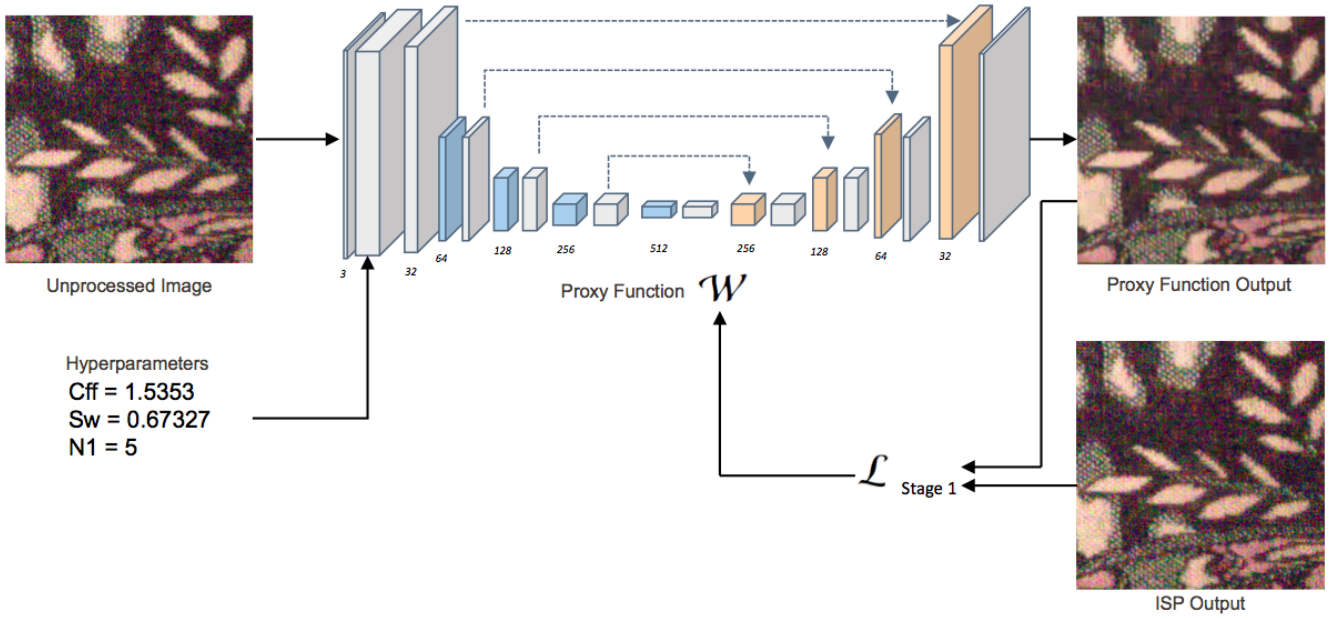



**Figure S.2:** *Setup of Stage 1, training of the proxy model.*

Because we give a channel per hyperparameter, each pixel of the channel receives a gradient. To apply gradients across the channel to a single hyperparameter, we take the gradients for all individual pixels in the channel and average the values before appropriately stepping with the optimizer. Furthermore, we make sure to keep all hyperparameters within their defined bounds after each modification. For hyperparameters that take on discrete values, we allow them to wander in the continuous range during the epoch, but then reassign it the discrete value that lies closest to the current value before each validation stage. This allows us to find the valid hyperparameter set that minimizes the objective function, $\mathcal{P}^*$. See Figure S.3 for the diagram.

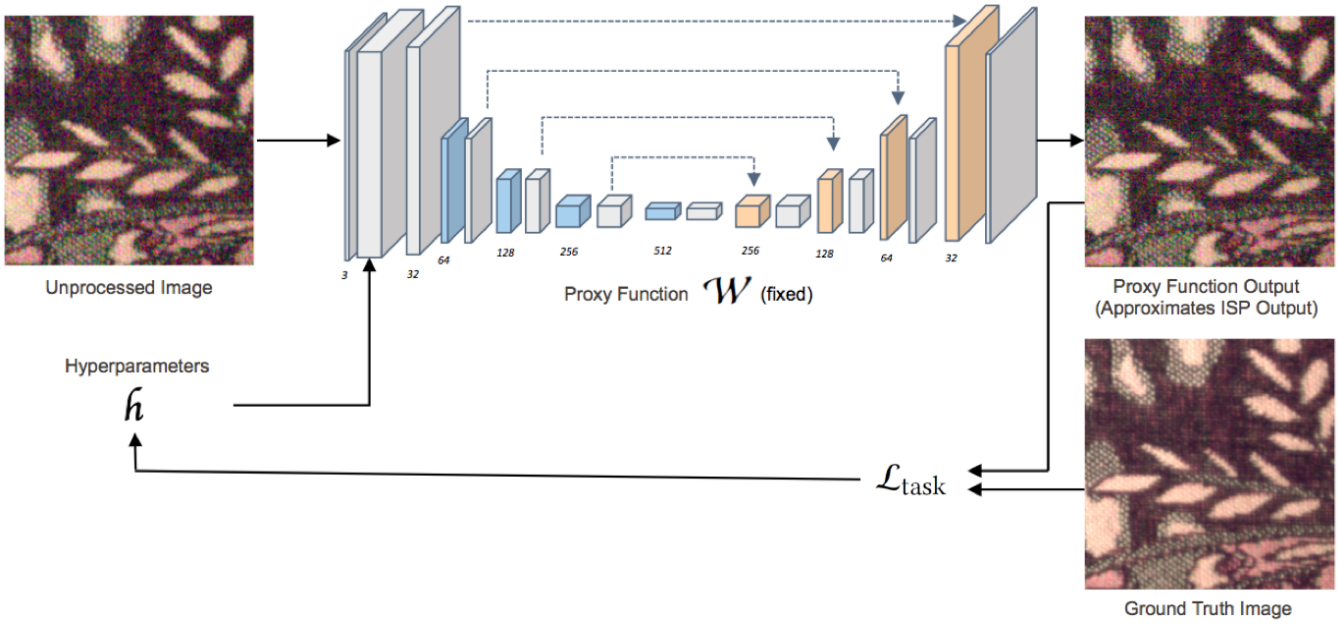



**Figure S.3:** *Setup of Stage 2, optimizing the hyperparameter.*

**Stage 3: Hyperparameter Evaluation.** Finally, after we have optimized hyperparameters, we test these parameters by tuning the real ISP system with them. We can then compare the outputs of the ISP using these hyperparameters versus default parameters. Such comparisons are in the main paper, and additional comparisons can be found in Sec. S2.

## 2 Additional Details on Software BM3D Hyperparameter Optimization

**Dataset.** We work with the Smartphone Image Denoising Dataset (SIDD) [Abdelhamed et al. 2018]. We selected the 9 available scenes from the SIDD dataset that were taken with the iPhone 7. From these 9 scenes we select scene instances where the lighting conditions are "normal" and ISO $> 200$, which resulted in 23 total scene instances. For each of these scene instances we take 100 random crops of size $512 \times 512$. Our final image dataset consists of 6900 image patches of size $512 \times 512$ taken with the iPhone 7. We intensity normalize the noisy images and ground truth images using the 99th percentile linear scale. Next, we estimate the standard deviation of the noise using the naive standard shifted difference estimation method from [Heide et al. 2016]. This noise estimate is then scaled with a linear parameter `cff`, also following [Heide et al. 2016]. In addition to this scalar, we tune the patch size, `n1`, colorspace `cspace`, Wiener transform type `wtransform`, Wiener transform type, and neighborhood size `neighborhood` of the BM3D algorithm itself [Dabov et al. 2007].

In addition to this vanilla BM3D experiment, we have also run experiments with a cascade of two BM3D runs, which we dub "Cascaded BM3D", with orthogonal color spaces (yCbCr and OPP color space), where we linearly mix the output of both denoising runs after denoising using a scale factor `sw`. This cascade has less parameters, as we are running in both colorspaces effectively. Morover, we fix the neighborhood size to the default in this test.

We run BM3D and Cascaded BM3D [Dabov et al. 2007] with random uniformly sampled hyperparameters on this image dataset to generate denoised image patches that will be used to train the proxy models. The range that hyperparameters can take on is shown in Table 1 and Table 2.

| Hyperparameter Name | Search Range | Default value |
|---|---|---|
| cff | Continuous in [1,5] | 1.5 |
| n1 | Discrete in $\{2, 4, 8\}$ | 8 |
| cspace | Discrete in $\{0, 1\}$ | 0 |
| wtransform | Discrete in $\{0, 1\}$ | 0 |
| neighborhood | Discrete in $\{1, 2, 3, 4, 5, 6, 7, 8\}$ | 8 |

**Table 1:** *Hyperparameter ranges and defaults for BM3D.*

| Cascaded Hyperparameter Name | Search Range | Default value |
|---|---|---|
| cff | Continuous in [1,5] | 2 |
| sw | Continuous in [0,1] | 0.35 |
| n1 | Discrete in $\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ | 3 |

**Table 2:** *Hyperparameter ranges and defaults for Cascaded BM3D.*

**Stage 1: Proxy Model Training.** The proxy CNN model used to approximate BM3D and Cascaded BM3D is a variant on the UNet, with the following modifications. First, instead of starting off with 64 channels and increasing until reaching 1024 channels, we start at 32 and increase until 512. Second, we add a skip connection by adding the input image to the final output layer [Brooks et al. 2018].

This proxy model is then trained to fit the respective datasets for BM3D and Cascaded BM3D. We define $\mathcal{L}_{\text{STAGE 1}} = \mathcal{L}_1$ and optimize using ADAM. The learning rate is set to $0.0005$ and decreases every 7 epochs such that $LR_{i+1} = 0.1LR_i$. We use a batch size of 20 and train until convergence. For BM3D this occurs at around epoch 30. See Figure S.16 for the convergence curve of BM3D. After convergence, the proxy models output images similar to that of the BM3D algorithms. See Figure S.11 for examples of proxy model outputs compared with the output of the original BM3D denoising.

**Stage 2: Hyperparameter Optimization.** We first extend the search range for the hyperparameters, as shown in Table 3 and Table 4.

| Hyperparameter Name | Extended Search Range |
|---|---|
| cff | Continuous in [1,20] |
| n1 | Discrete in $\{2, 4, 8\}$ |
| cspace | Discrete in $\{0, 1\}$ |
| wtransform | Discrete in $\{0, 1\}$ |
| neighborhood | Discrete in $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ |

**Table 3:** *Extended hyperparameter search ranges for BM3D.*

| Cascaded Hyperparameter Name | Extended Search Range |
|---|---|
| cff | Continuous in [1,20] |
| sw | Continuous in [0,1] |
| n1 | Discrete in $\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ |

**Table 4:** *Extended hyperparameter search ranges for Cascaded BM3D.*

We then randomly uniformly initialize the hyperparameters within their search ranges. We define $\mathcal{L}_{\text{TASK}} = \mathcal{L}_{\text{MSE}}$. We optimize the hyperparameters using the ADAM optimizer. The learning rate is set to $0.001$ and decreases every 2 epochs such that $LR_{i+1} = 0.8LR_i$. The hyperparameters are optimized until convergence, which generally occurs quickly, after around 10 epochs. Again, refer to Figure S.16 for convergence curves of the hyperparameters during Stage 2.

**Stage 3: Hyperparameter Evaluation.** In order to evaluate the efficacy of the hyperparameters obtained from Stage 2, we run BM3D and Cascaded BM3D with the hyperparameters we found. We then compare the denoised image results to other state of the art denoising methods. See Figures S.5, S.6, S.7, S.8, S.9, S.10 for comparisons. We also run BM3D on the SIDD Benchmark using the hyperparameters found in Stage 2, the results of which are shown in the main paper.

## 3 Additional Details on Hardware ISP Hyperparameter Optimization

As described in the Section 3 of the main document, the hardware ISP used to run the experiments is the ARM MALI-C71 ISP, and we tune the hyperparameters that are also listed within there.

**Datasets.** We work with two sets of images, the Rainbow dataset and the SIDD dataset [Abdelhamed et al. 2018]. For the Rainbow dataset, we obtain a single raw input image as well as the ground truth through the imaging methods described in section 6, and for the SIDD dataset, we extract 5000 $1024 \times 1024$ image patches from low light RAW images taken through the Google Pixel. From there, we randomly sample 5000 sets of hyperparameters for each set, and apply them to the images to create the ISP Outputs. For the Rainbow dataset, the same raw image is used as the ISP input for all hyperparameters, while for SIDD, one image is used per set of hyperparameters. As stated in the main paper, all hyperparameters are normalized to be between 0 and 1.

**Stage 1: Proxy Model Training.** As stated in the main text, the proxy CNN model used to approximate the hardware setup is a variant on the UNet. See Section 6 of the paper for the full description. This proxy model is then trained to fit their respective datasets. For the hardware dataset, we define $\mathcal{L}_{\text{STAGE 1}} = \mathcal{L}_{\text{MSE}}$, optimized using ADAM. The learning rate is set to $0.0005$, with a learning rate reduction schedule of $90\%$ every 10 epochs. We use a batch size of 12, and train until convergence. For the Rainbow dataset, this occurs at around epoch 400, while for the SIDD dataset, this occurs at around epoch 200. See Figure S.16 for the convergence curves of these models. After convergence, the proxy models output images similar to that of the ISP given raw input images and a set of hyperparameters. See Figure S.12 and Figure S.15 for examples of proxy model outputs compared with the output of the original ISP. For these figures, the input image is shown in grayscale since it is a single channel RAW image.

**Stage 2: Hyperparameter Optimization.** During stage 2, we randomly initialize the hyperparameters $\mathcal{P}$. Afterwards, as mentioned in the primary text, we define $\mathcal{L}_{\text{STAGE 2}} = \mathcal{L}_{\text{PERC}} + \lambda\mathcal{L}_1$, where $\mathcal{L}_{\text{PERC}}$ is the AlexNet variant of perceptual loss from [Zhang et al. 2018]. In our experiments, we set $\lambda = 2$. We then optimize the hyperparameters using the ADAM optimizer and a learning rate of $0.000005$, and use the same learning rate exponential decay scheme that is described in Stage 1. The hyperparameters are optimized until convergence, which generally occurs relatively quickly, after around 20 epochs. Again, refer to Figure S.16 for convergence curves of the hyperparameters during Stage 2.

**Stage 3: Hyperparameter Evaluation.** In order to evaluate the efficacy of the hyperparameters found on the Rainbow dataset, we extract the optimized hyperparameters and feed them back into the hardware. The hardware then generates ISP outputs using both these optimized hyperparameters, as well as default hyperparameters provided by a "golden-eye" expert. We then compare these images both qualitatively and quantitatively to a defined ground truth. These comparisons and values can be found in Section 6 of the main text.

## 4 Additional Details on Black-box Optimization Benchmark

**Test Functions.** As mentioned in Section 5 of the primary text, we use four test functions in order to compare the optimization performance of our differentiable proxy with four commonly used optimization methods. The inputs into each of these functions are 20-dimensional ($d = 20$), and the function returns a scalar value. Some of the optimization methods rely on a search range as input as well. For these functions, the search range is defined by a hypercube where $x_i \in [\text{lower bound}, \text{upper bound}]\ \forall i = 1, ..., d$. Table 5 gives the exact details to the test functions, as well as their corresponding search ranges.

| Function Name | Equation | Search Range |
|---|---|---|
| Ackley [2012] | $f(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + e$ | $[-32.768, 32.768]$ |
| Rastrigin [1974] | $f(\mathbf{x}) = 5d + \sum_{i=1}^{d}\left(x_i^2 - 5\cos\left(2\pi x_i\right)\right)$ | $[-5.12, 5.12]$ |
| Step-2 [2013] | $f(\mathbf{x}) = \sum_{i=1}^{d}\lfloor x + 0.5\rfloor^2$ | $[-100, 100]$ |
| Alpine1 [2013] | $f(\mathbf{x}) = \sum_{i=1}^{n}\lvert x_i\sin(x_i) + 0.1x_i\rvert$ | $[-10, 10]$ |

**Table 5:** *Equations and search ranges. Each equation is 20-dimensional, so in all cases, $d = 20$. The search range defines the upper and lower bound of the hypercube of possible input values into the functions.*

Furthermore, we modify each of the functions in the following manner. First, for each function, we generate a translational vector. Each

dimension of the vector is uniformly sampled from a range of at most $25\%$ of the search range for the function. This vector is then fixed across all experiments for the particular function, and is added to any input passed into the function. The purpose of this vector is to shift the minimum for each of the functions away from both the origin, and from the center of the search range. This avoids that an optimizer uses the "lucky guess" of 0, which is often a meaningful initial guess.

We add Gaussian noise to the output of the function. For each function, we randomly sample points within the hypercube search space, and evaluate them in the noiseless function (after applying the translation). This allows us to get an estimate of what the maximum value the function takes on within the search range is. We then set the standard deviation of the noise added to the output to $3\%$ of this approximate maximum value. Table 6 details these values.

| Function Name | Standard Deviation of noise |
|---|---|
| Ackley | $22 \times 0.03 = 0.66$ |
| Rastrigin | $700 \times 0.03 = 21.0$ |
| Step-2 | $200,000 \times 0.03 = 6,000$ |
| Alpine1 | $140 \times 0.03 = 4.2$ |

**Table 6:** *Standard deviations used to add random Gaussian noise to the evaluation of each of the functions. Each of the standard deviations is equal to $3\%$ of the approximate maximum value of the function within the search range.*

**Training the Differentiable Proxy.** For each function, we need train a differentiable proxy model to approximate it. The training data is generated by sampling the hypercube search space, and evaluated through the noisy translated function to produce corresponding labels. These inputs and outputs are then normalized such that the search space lies between the $[-1, 1]$ hypercube, and the outputs all lie between $[0, 1]$.

The model itself (depicted in Figure S.4) is a 9-layer fully connected neural network, where the number of nodes at each layer is 32, 64, 128, 256, 256, 128, 64, 32, 1, respectively. The weights of the model are optimized using the ADAM method, with a learning rate of $0.005$ and this learning rate decays after every epoch, so that $LR_{i+1} = LR_i \times 0.95$. The model is trained for 3 epochs, since after the second epoch, no noticeable improvement in the fit is made.

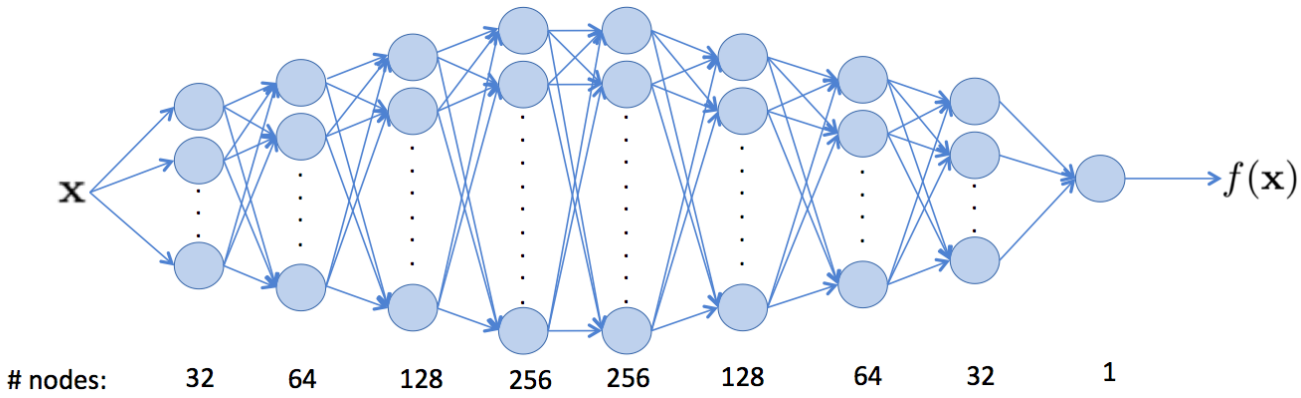



**Figure S.4:** *Network architecture of the differentiable proxy model used to learn the test functions.*

**Using Differentiable Proxy to minimize the function.** After the proxy model is trained on the noisy translated data, we use it to find an input that minimizes the output of the model. We do this by first freezing the weights in the model, and then randomly initializing an input vector that lies within the hypercube search space (which has been scaled to be between $[-1, 1]$). This input is then passed through the model, which then produces an output that we can calculate the gradient with, and we backpropogate this gradient. We run this step for 500 iterations using the ADAM optimizer with a learning rate of 0.05. This produces an input vector that we can then compare against other methods. We repeat this process twice more, giving us a total of three optimized vectors, from three different random initial starting points.

**Comparing Optimization Methods.** As stated in the primary text, we compare our differentiable proxy model to four other widely used optimization methods: the Powell method [1965], Nelder-Mead method [1965], Hyper Optimization [2013], and Bayesian Optimization [2014].

Powell method takes in the noisy, translated function, as well as three initial starting points. We run the Powell Method for 500 function evaluations, which then gives us three optimized vectors.

Nelder-mead method is run in a similar fashion, with three initial starting points, and run for 500 function evaluations. Again, this outputs three optimized vectors.

For Hyper Optimization, we use the Tree-of-Parzen-Estimators (TPE) algorithm defined in [Bergstra et al. 2011] to search through the search space. We pass in the noisy, translated function with the search space, and let the algorithm run for 500 iterations in order to produce an optimized vector. For fairness, we also run this method three times to produce three optimized vectors.

For our last method, we run Bayesian Optimization with three initializations by passing in both the function as well as the search space. Due to the slow runtime of the algorithm, we terminate the method after 200 iterations, which still takes orders of magnitudes longer than all other methods compared. This produces one optimized vector.

Finally, the optimized vectors for each of the methods are fed into a noiseless translated version of the function. For methods that have more than one optimized input vector, we report the best result, which are shown in section 5 of the main text.

## 5 Additional Details on Image Metrics

Here, we describe the image metrics that are used within the main text. For a detailed description of traditional image metrics, we refer the reader to [Phillips and Eliasson 2018] and [Baxter et al. 2012].

**Detail Accuracy Metric.** Measured within the Random Ellipse ROI, the Detail Accuracy metric is a weighted geometric average of two full reference image differences, one based on SSIM (Structural SIMilarity) [Wang et al. 2004] and the other on FSIM (Feature Similarity) [Zhang et al. 2011]. In particular we compute the Detail metric as

$$1 - \sqrt{\text{SSIM} + \sqrt{\text{FSIM}}}.$$

**Color Accuracy KPI.** The color accuracy metric (CAM) is based on the CIEDE2000 [Luo et al. 2001] differences between linear light patch averages over the two rainbows (color and step chart) and chromatically adapted target values based on photospectrometer measurements made directly on the monitor display (denoted as $P_M$). 40 patches $P_1, ..., P_{40}$ are used to compute these differences $\Delta(P_i, P_M)$, $i \in [40]$, which are then aggregated with a Minkowski norm with a large $p$ (we use $p = 4$) so that outliers are strongly penalized. An additional ROI, located outside of the monitor within the capture, provides another patch average with target value (0,0,0). Overall,

$$\text{CAM} = \left( \frac{1}{40} \sum_{i=1}^{40} \Delta(P_i, P_M) \right)^4.$$

**Zipper KPI.** The Zipper KPI is based on the following building block: Within a 3x3 pixel square, consider the four-pixel "T" (the "zipper's tooth") and its five-pixel "U" complement. Define "T" to be darker than "U" if the maximum of "T" is smaller than the minimum of "U". Taking the worst difference over the various cases (lighter/darker + up/right/down/left) gives an effective "zipper tooth" detector at the pixel level. It is measured on hyperbolic wedge blocks.

**Color Moire.** Color Moire is an artifact that arises from sampling under the Nyquist frequency. Measured within the hyperbolic wedge ROIs, the Color Moir KPI is the lowest L1-norm of channel differences, thresholded to account for global white balance error. See [Phillips and Eliasson 2018] and [Imatest ], for further details.

## 6 Additional Details on Image Tonemapping

**Dataset.** Use the SUN dataset [Xiao et al. 2010] to generate training and testing samples using the laplacian tonemapper from [Paris et al. 2011]. The individual algorithm parameters and their role are indicated in the main document. Our implementation is borrowed from [Yang et al. 2016] and [Ragan-Kelley et al. 2012].

**Stage 1: Proxy Model Training.** The proxy CNN model used to approximate global tone mapping is a variant on the UNet, with the following modifications. To validate that the proxy model is indeed able to capture global image operations at a moderate model size, instead of starting off with 64 channels and increasing until reaching 1024 channels, we start at 32 and increase until 512.

The proxy model is then trained to fit the data set. We define $\mathcal{L}_{\text{STAGE 1}} = \mathcal{L}_1$ and optimize using ADAM. The learning rate is set to $0.0001$ and we train for 200 epochs.

**Stage 2: Hyperparameter Optimization.** We utilize ADAM as in the previous experiments for the second stage and we define $\mathcal{L}_{\text{STAGE 2}} = \mathcal{L}_{\text{MSE}}$.

**Performance on a Single Scene Type.** The SUN dataset consists of a wide variety of scene semantics, so we performed an additional experiment with a dataset that consists of only one scene type, namely images of faces. The dataset that we chose for this experiment is the MTFL dataset [Zhang et al. 2014].

The proxy CNN model is the same UNet used for the SUN dataset. For the first stage we use 5,490 face images and apply the laplacian tonemapper with randomized parameters to these images to create our training set. We define $\mathcal{L}_{\text{STAGE 1}} = \mathcal{L}_{\text{MSE}}$ and train using ADAM. The learning rate is set to $0.0001$ and we train for 200 epochs.

For the second stage we apply the laplacian tonemapper with manually chosen hyperparameters to face images that were not used for Stage 1 to create our test set. We define $\mathcal{L}_{\text{STAGE 2}} = \mathcal{L}_1$ and again optimize using ADAM.

We found that qualitative visual performance of the proposed method was just as good as on the SUN dataset, which demonstrates that changing the scene semantics does not negatively affect performance. Figure S.24 gives examples of our method successfully recovering hyperparameters for the tonemapper that create output images of visually similar quality to the desired image. Note that these examples consist of images from the test set and were not seen during proxy training in Stage 1, and hence the network did not aggressively overfit.

## 7 Loss Function Visualizations

As stated in the main text, the landscape of the performance metric with respect to the hyperparameters of the imaging system is highly complex and non-linear. In order to visualize this phenomenon in the context of our application, we plot the $\mathcal{L}_{\text{TASK}}$ loss, which in this case is $\mathcal{L}_{\text{TASK}} = \mathcal{L}_{\text{PERC}} + 2\mathcal{L}_1$, with respect to three of the hyperparameters on the ARM MALI-C71 ISP, with 500 points sampled across the range of the hyperparameters for each plot. As can be seen in Figures S.17, S.18, and S.19, the loss function takes on a complex rugged shape with respect to each of the hyperparameters.

## 8 Additional Experiments using Wide Field of View Optics

In this section, we provide additional experiments for wide field of view (FoV) camera lenses. Typical wide FoV lenses introduce geometric distortions, i.e., spatial offsets of focused rays. These offsets often increase with the distance from the chief ray, and hence are apparent in off-axis regions on the sensor.

Fig. S.20 shows experimental results for a $150^\circ$ lens, and, for comparison, a capture of the $50^\circ$ lens from the prototype described in the main manuscript. The hyperparameter optimization for the wide FoV lens substantially improves denoising performance, color reproduction and low-light performance of the imaging system.

Moreover, the proposed method can be extended to fish-eye lens systems with ultra wide FoV. Fig. S.21 shows an extension of the proposed calibration setup using two screens, one for the central region and one for the peripheral region. The per-pixel reference calibration is successful for this multi-screen setup.

## 9 Additional Tracking Experiments

In this section, we demonstrate temporal tracking experiments for tuning with temporal sequences. We evaluated the proposed method over temporal sequences of captured automotive scenes. Specifically, we evaluated tracking performance over videos processed by an ISP with default parameters and another with our fined tuned ISP. For the tracking experiments, we use the detector described in the main manuscript in Section 6. The tracking algorithm evaluated on top of the detector output is the SORT method [Bewley et al. 2016] which provides efficient real-time tracking. Fig. S.22 shows tracking results for a sequence during night driving. Optimizing the ISP hyperparameters significantly improves the robustness of the tracking output due to more robust detections which are temporally more stable as without optimizing the respective parameters. Fig. S.23 shows additional tracking results for capture sequences acquired during day time. ISPs struggle in edge scenarios, such as low-light, and hence the marginal improvement is smaller as in the night driving scenarios.

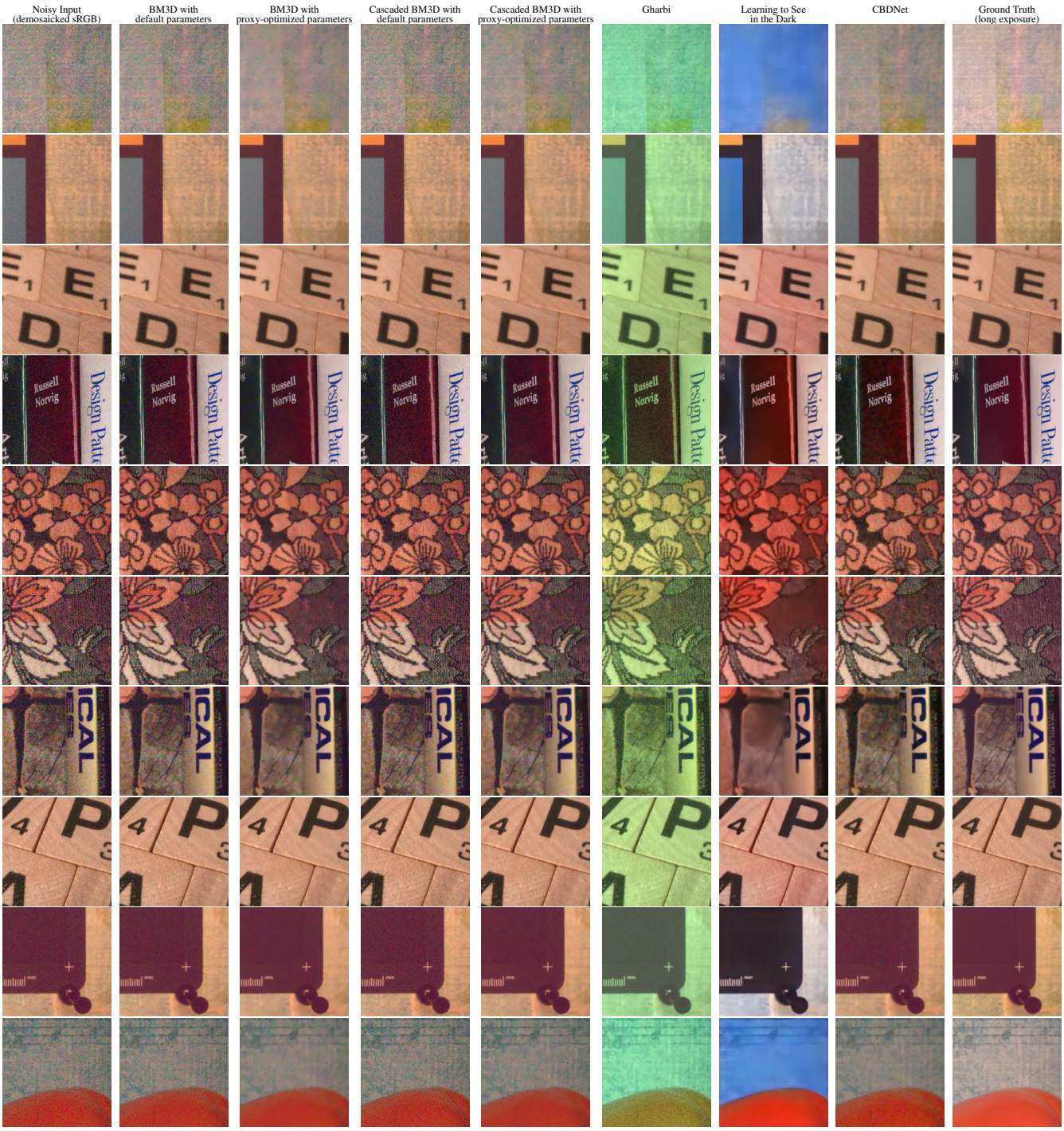



**Figure S.5:** *BM3D results*

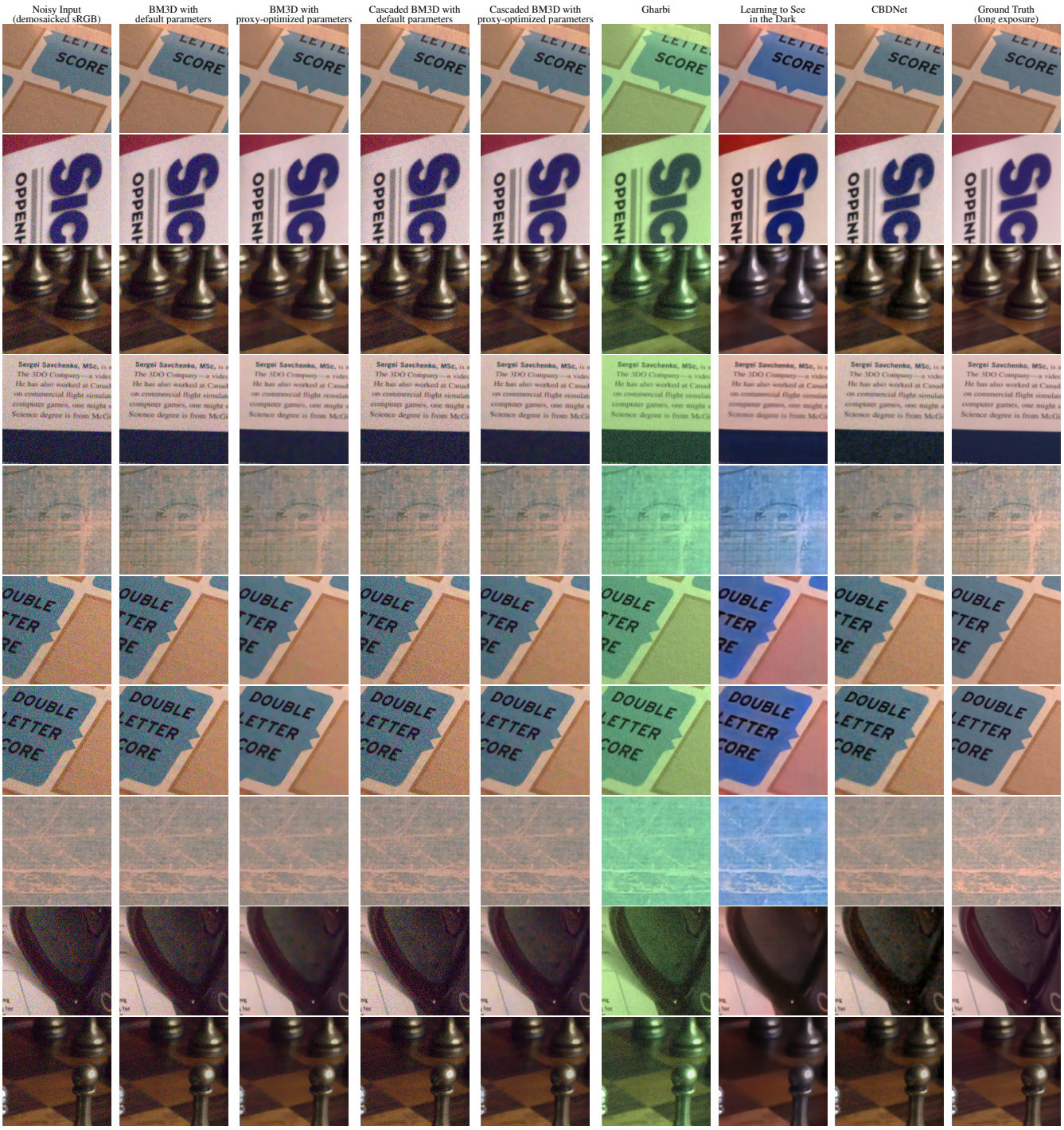



**Figure S.6:** *BM3D results (continued)*

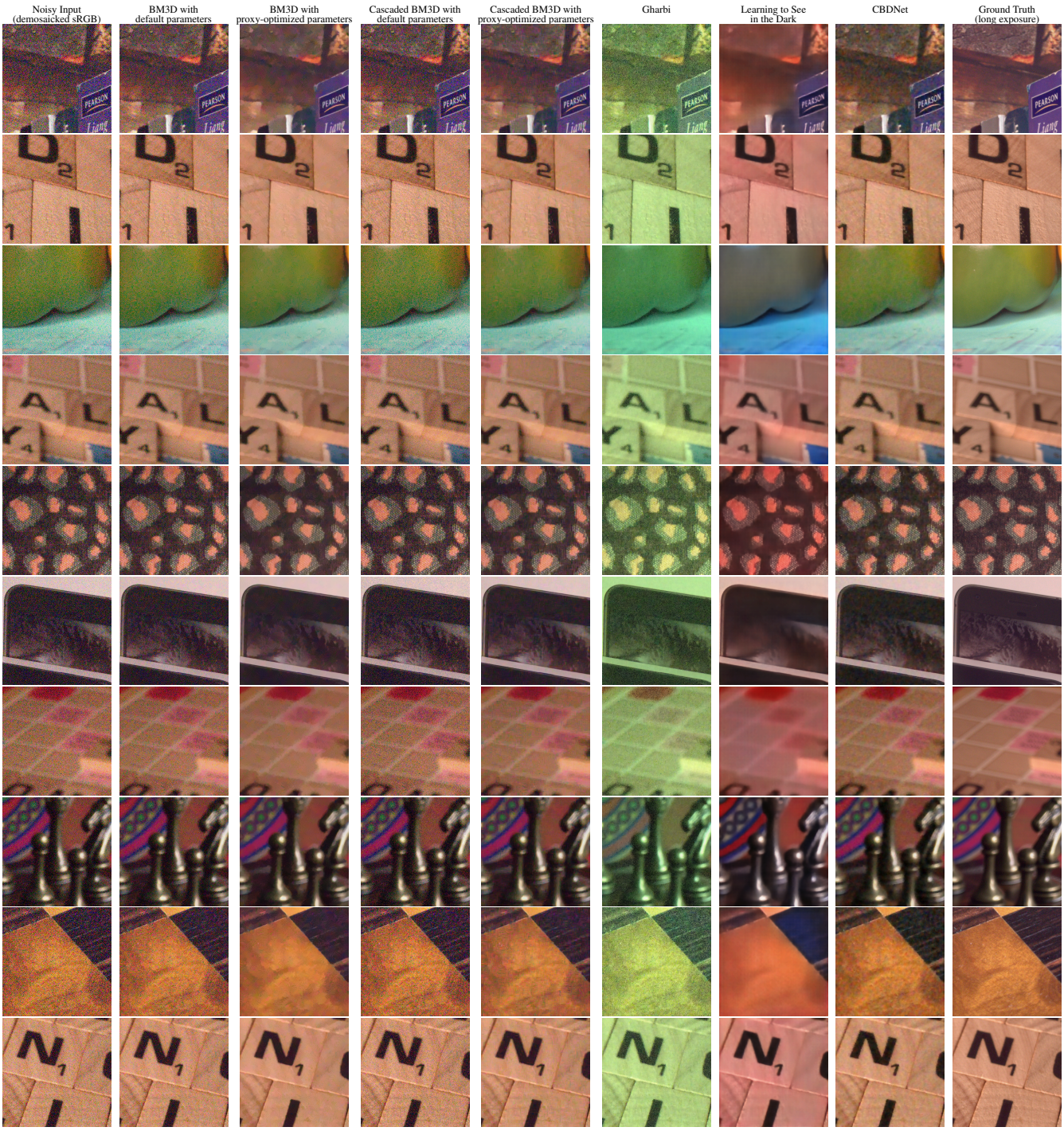



**Figure S.7:** *BM3D results (continued)*

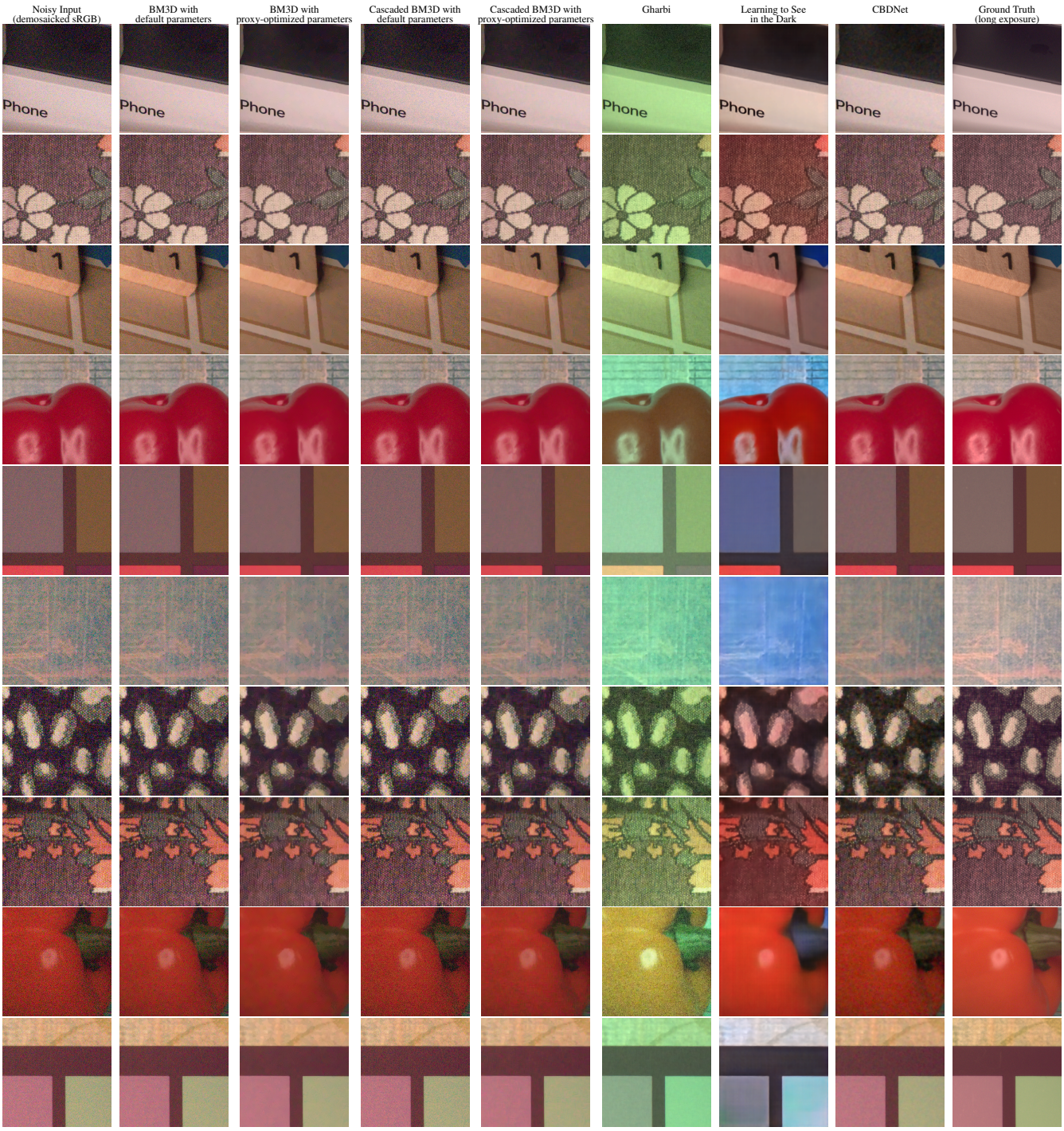



**Figure S.8:** *BM3D results (continued)*




**Figure S.9:** *BM3D results (continued)*

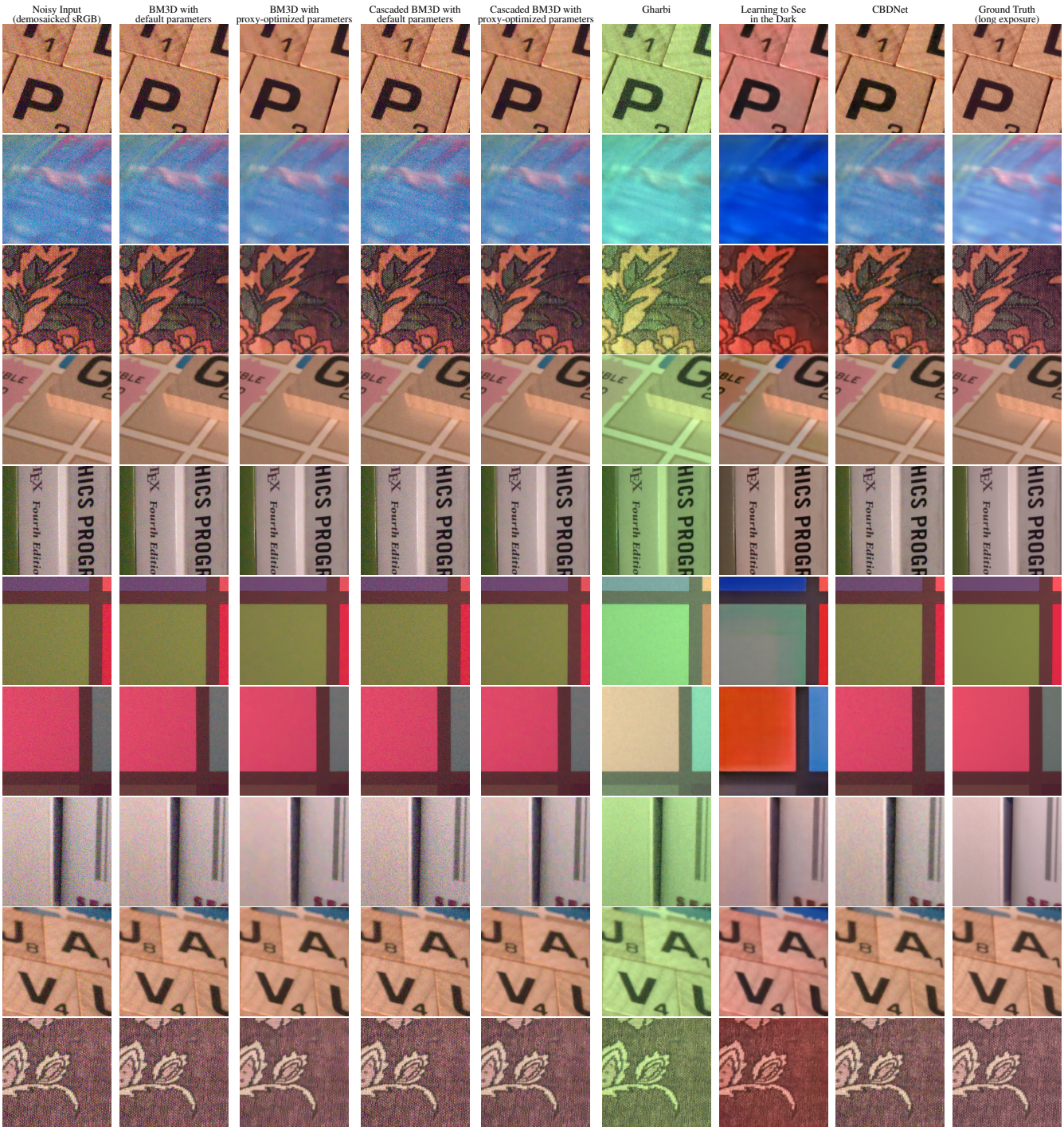



**Figure S.10:** *BM3D results (continued)*

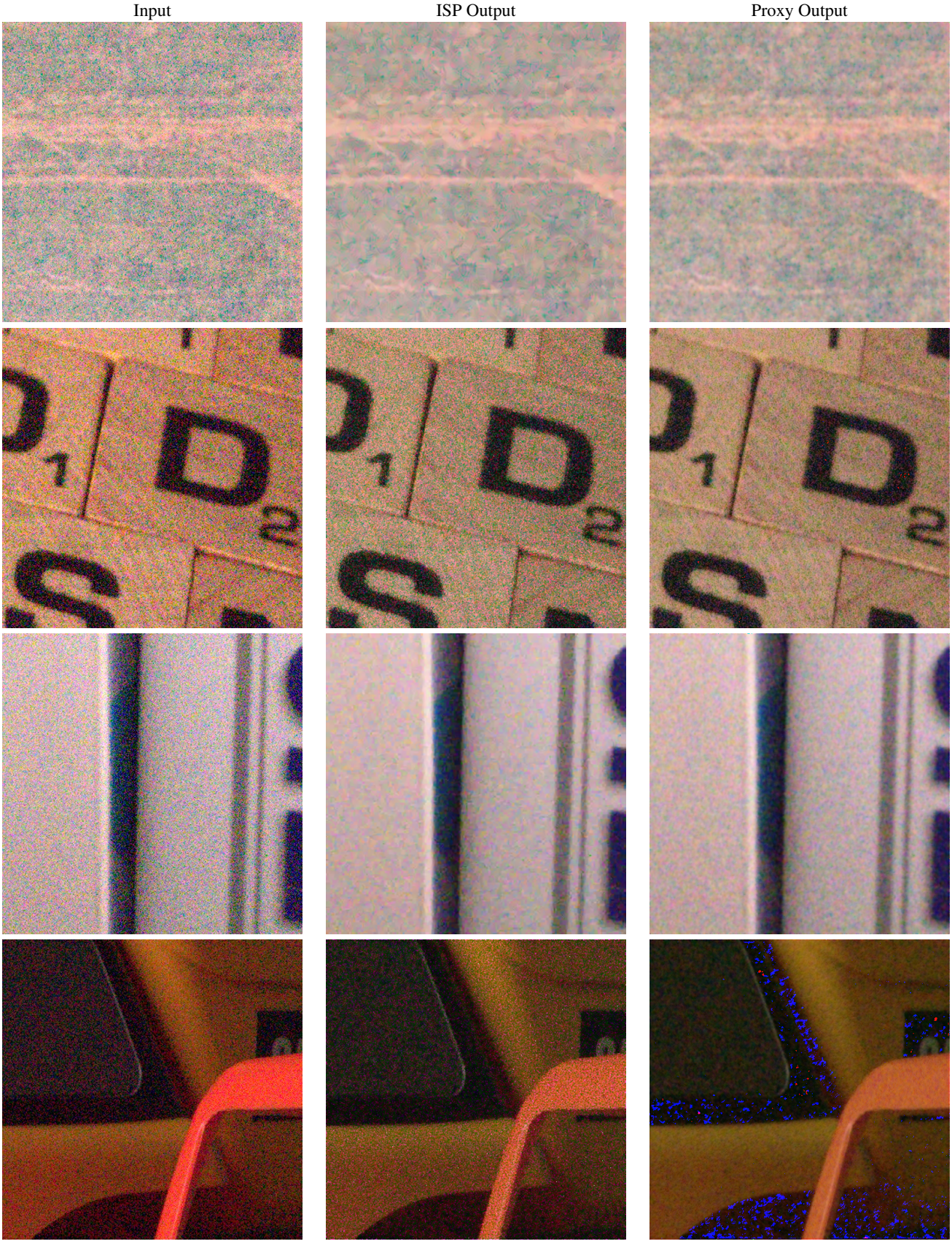



**Figure S.11:** *Outputs of the Proxy Model for the BM3D algorithm on the SIDD Dataset.*

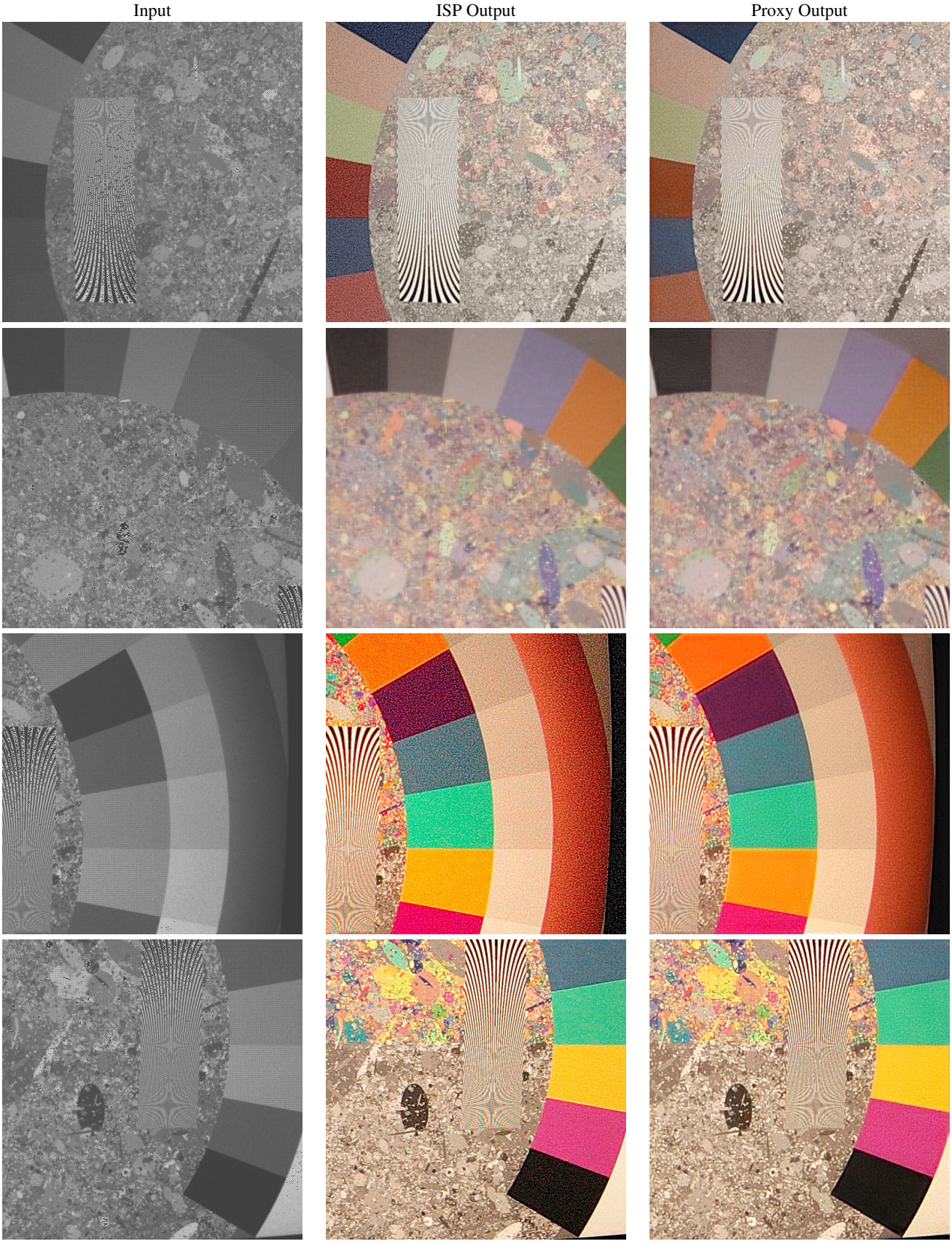



**Figure S.12:** *Outputs of the Proxy Model for the* ARM MALI-C71 ISP *on the Rainbow Dataset.*

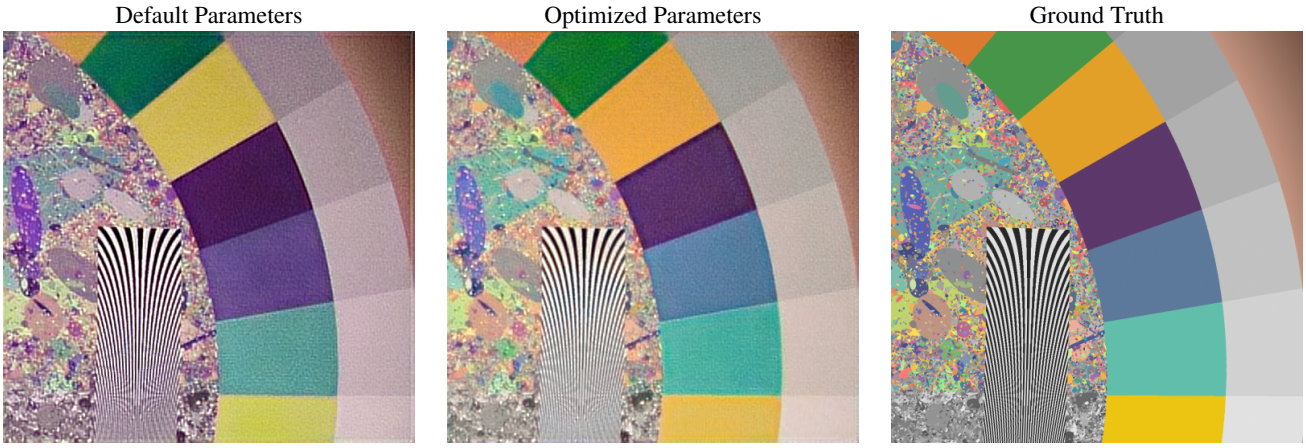



**Figure S.13:** *Fine-tuning results on the rainbow 16x gain with free white balance data set. Compared to the output of the proxy model when given default ISP parameters (left), the output of the proxy model when given optimized parameters (middle) improves visual quality compared to the ground truth (right). The optimized parameters were obtained with random initialization.*

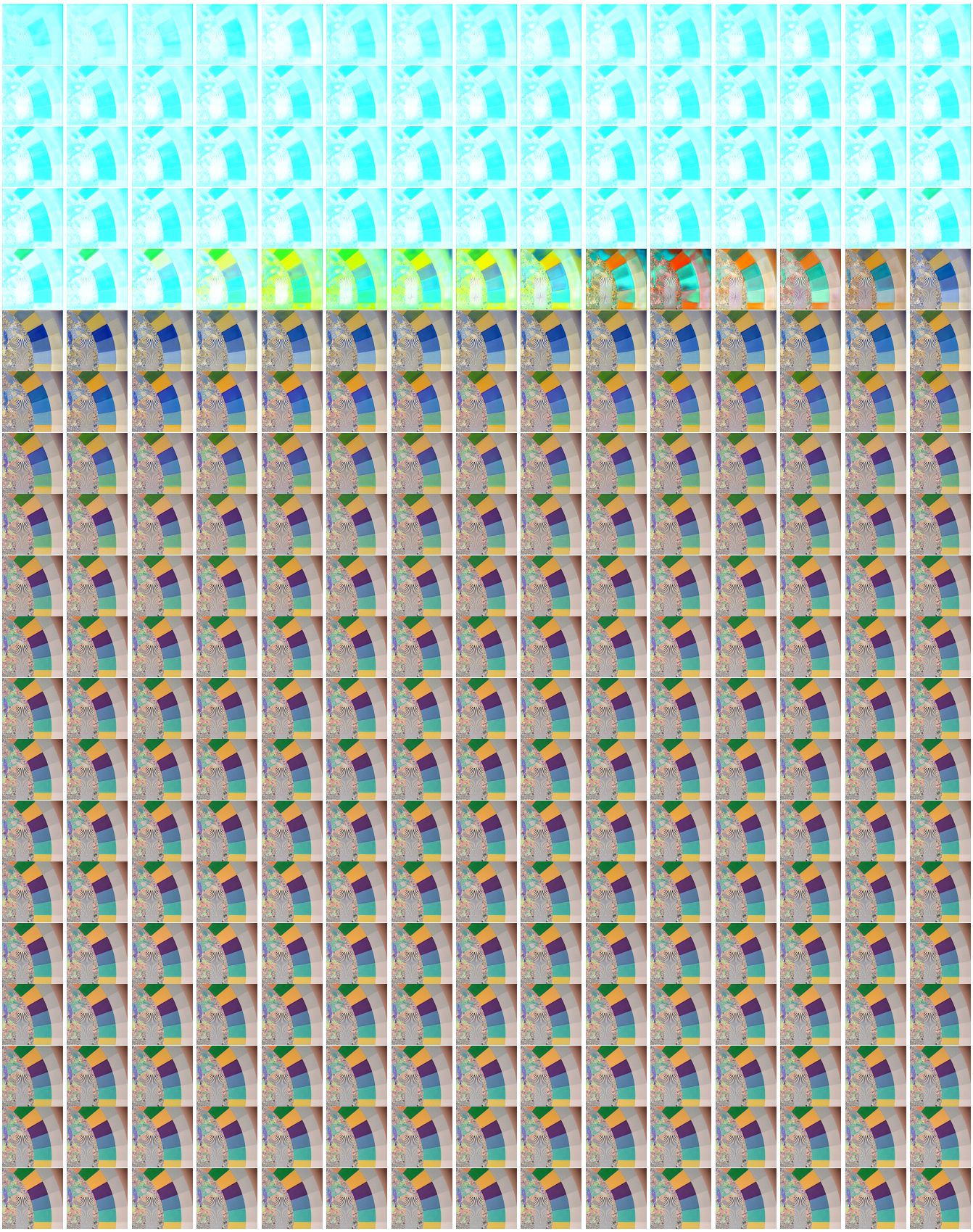


**Figure S.14:** *Timelapse of fine-tuning procedure starting from random initialization.*

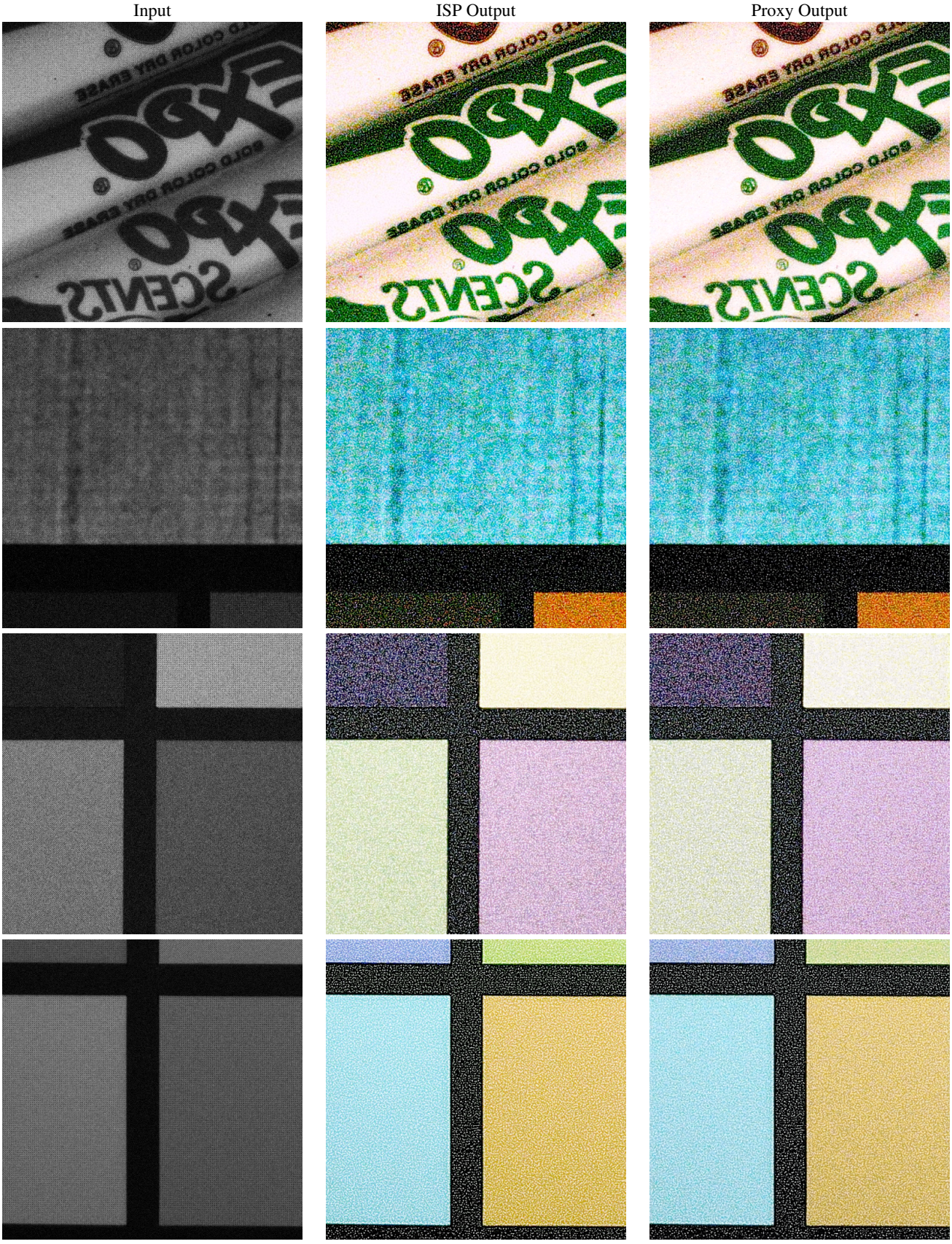



**Figure S.15:** *Outputs of the Proxy Model for the* ARM MALI-C71 ISP *on the SIDD Dataset.*

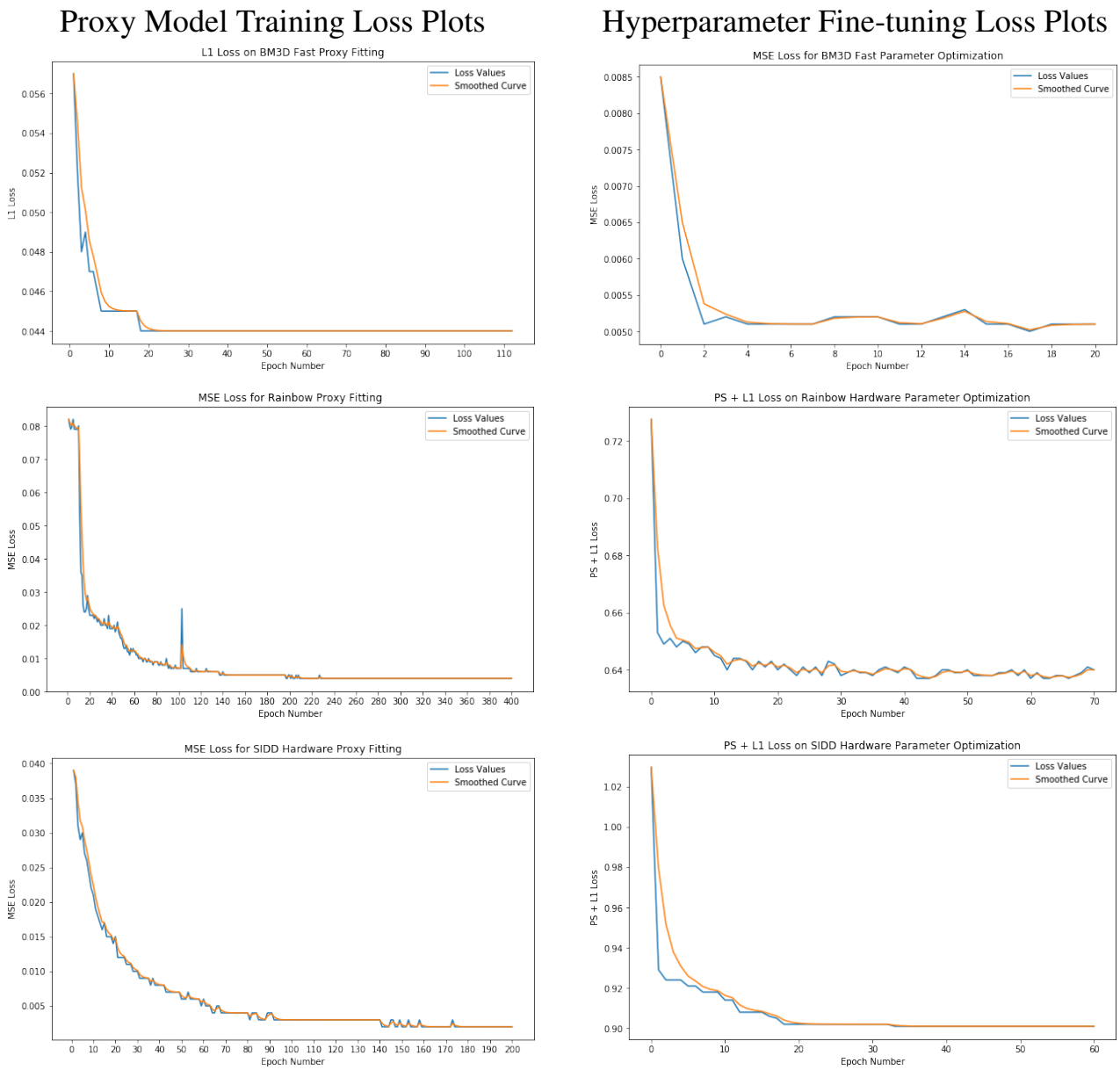



**Figure S.16:** *Loss plots for proxy model training and hyperparameter fine-tuning.*

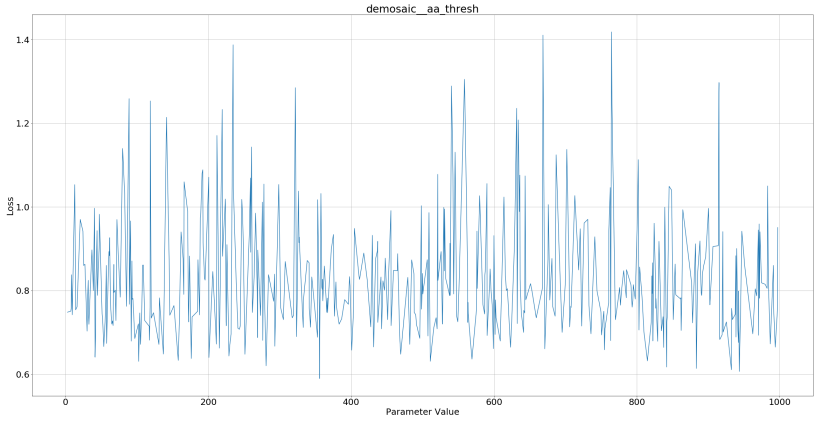



**Figure S.17:** *Loss plot for Demosaicking aa thresh.*

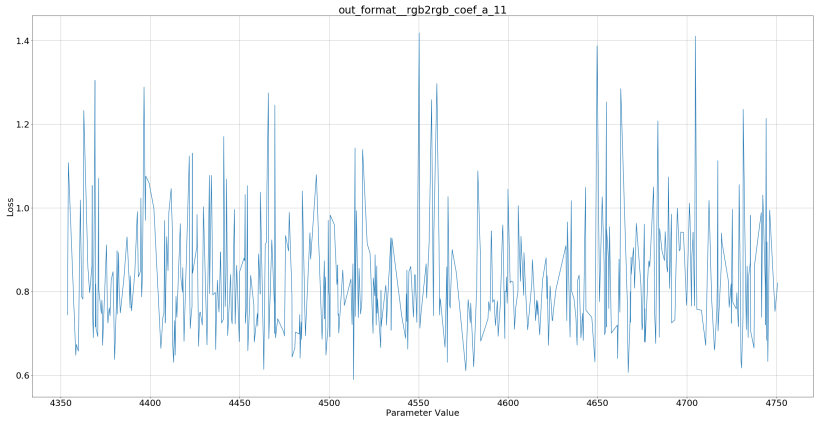



**Figure S.18:** *Loss plot for Colorspace Conversion coef a 11.*

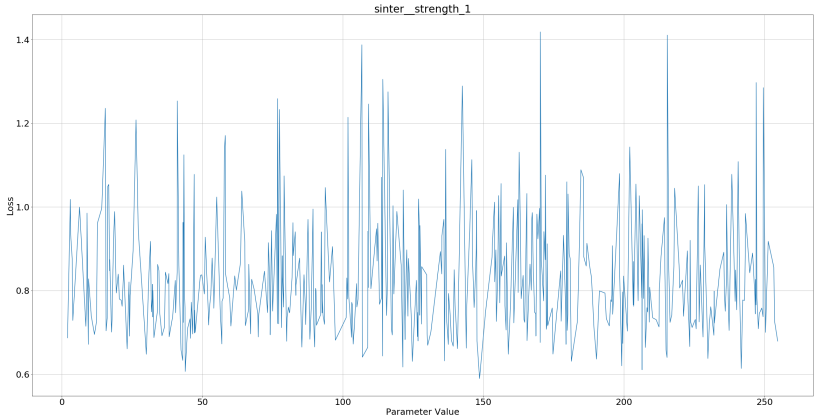



**Figure S.19:** *Loss plot for Denoising Strength 1.*

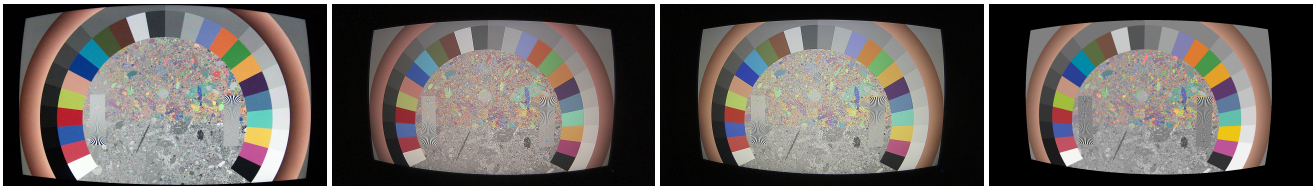


**Figure S.20:** *Hyperparameter Optimization for Wide Field of View Lenses. From left to right: Example capture of the* $50^\circ$ *lens used in the main manuscript with the screen at 50 cm distance, capture using lens system with* $150^\circ$ *field of view using default ISP parameters, results of hyperparameter optimization using the same* $150^\circ$ *lens system, synthetic reference target for the* $150^\circ$ *lens. The results demonstrate that the proposed method generalizes to optical systems with large field of view.*

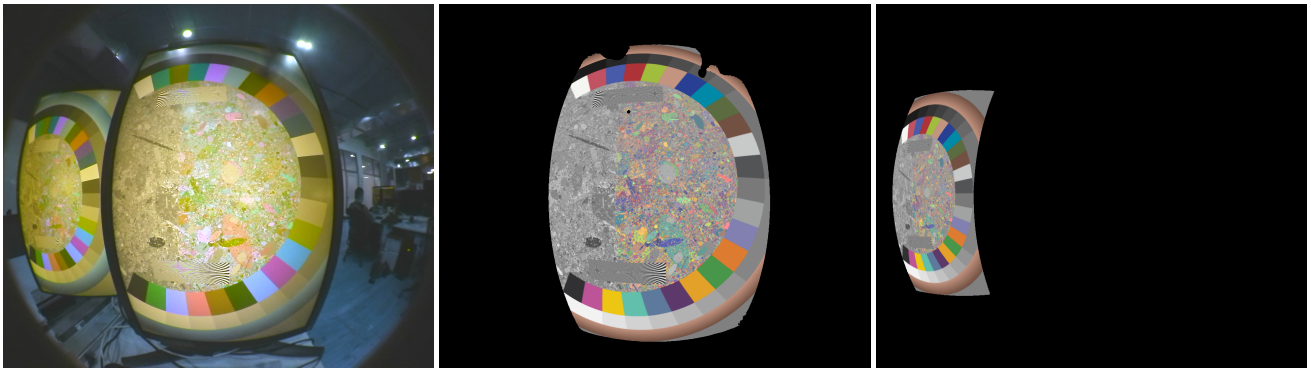


**Figure S.21:** *Calibration for extreme fish-eye optics. The proposed system can be extended with multiple screens to support ultra wide field of view optical systems. From left to right: Capture with a* $180^\circ$ *lens system with screens at about 1 m distance, synthetic reference chart of center screen after calibration, synthetic reference chart of periphery screen after calibration. The calibration functions robustly for on-axis and peripheral regions.*

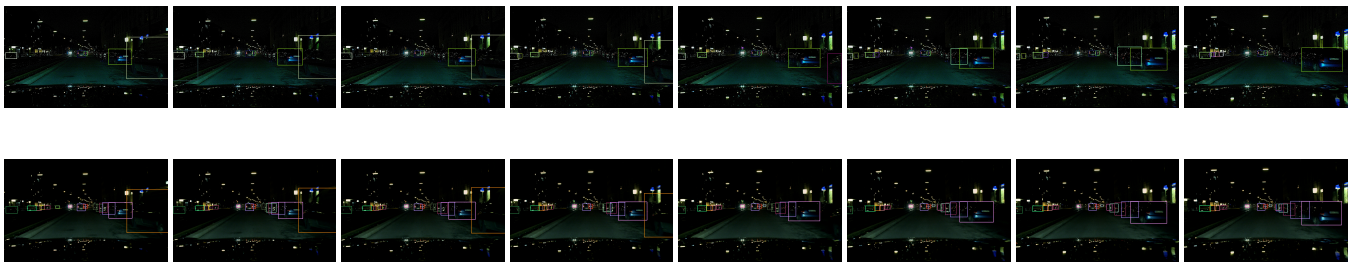


**Figure S.22:** *Effect of ISP hyperparameter optimization on tracking. The top row shows a sequence of frames acquired during night time driving. This sequence has been processed by a computer vision stack consisting of object detection and tracking (see text). A detected track is identified with a bounding box color. The top sequence is processed with a manually tuned ISP. The bottom sequence shows the same RAW frames processed by an ISP with optimized parameters. The tracks are significantly more complete and lack tracking ID switches. Please zoom in document for better viewing.*



**Figure S.23:** *Effect of ISP hyperparameter optimization on tracking. The top row shows a sequence of frames acquired during day time driving. This sequence has been processed by a computer vision stack consisting of object detection and tracking (see text). A detected track is identified with a bounding box color. The top sequence is processed with a manually tuned ISP. The bottom sequence shows the same RAW frames processed by an ISP with optimized parameters. The tracks are significantly more complete and lack tracking ID switches. Please zoom in document for better viewing.*

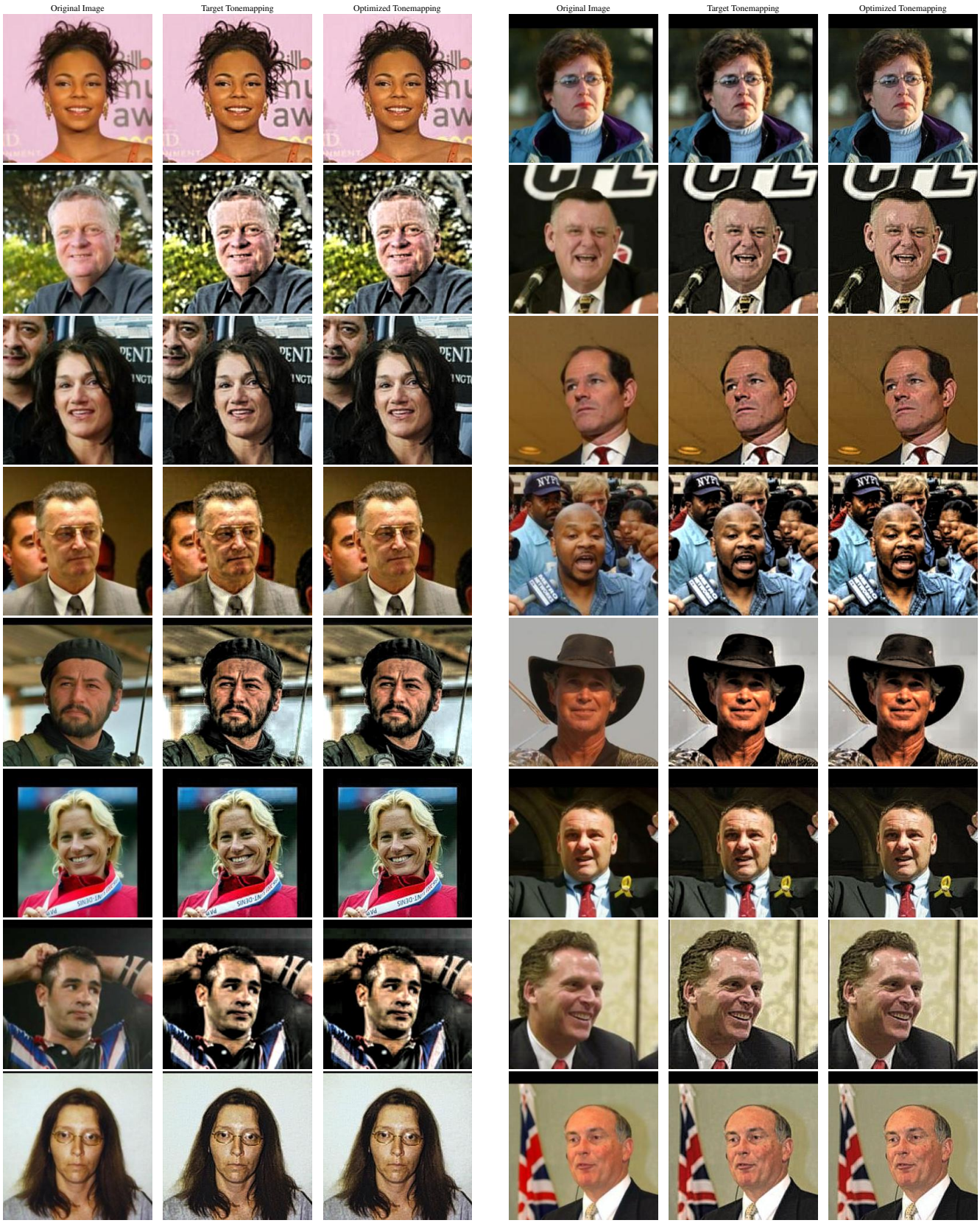



**Figure S.24:** *Example results for additional tonemapping experiments on the MTFL dataset.*

## References


ABDELHAMED, A., LIN, S., AND BROWN, M. S. 2018. A high-quality denoising dataset for smartphone cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1692–1700.

ACKLEY, D. 2012. *A Connectionist Machine for Genetic Hillclimbing*. The Springer International Series in Engineering and Computer Science. Springer US.

BAXTER, D., CAO, F., ELIASSON, H., AND PHILLIPS, J. 2012. Development of the i3a cpiq spatial metrics. vol. 8293, 8293 – 8293 – 12.

BERGSTRA, J. S., BARDENET, R., BENGIO, Y., AND KÉGL, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2546–2554.

BERGSTRA, J., YAMINS, D., AND COX, D. D. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, Citeseer, 13–20.

BEWLEY, A., GE, Z., OTT, L., RAMOS, F., AND UPCROFT, B. 2016. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, 3464–3468.

BROOKS, T., MILDENHALL, B., XUE, T., CHEN, J., SHARLET, D., AND BARRON, J. T. 2018. Unprocessing images for learned raw denoising. *arXiv preprint arXiv:1811.11127*.

DABOV, K., FOI, A., KATKOVNIK, V., AND EGIAZARIAN, K. 2007. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Trans. Image Processing 16*, 8, 2080–2095.

HEIDE, F., DIAMOND, S., NIESSNER, M., RAGAN-KELLEY, J., HEIDRICH, W., AND WETZSTEIN, G. 2016. ProxImaL: Efficient image optimization using proximal algorithms. In *Proceedings of ACM SIGGRAPH*.

IMATEST. Nyquist aliasing. http://www.imatest.com/docs/nyquist-aliasing/. [Online; accessed 5-January-2019].

JAMIL, M., AND YANG, X. 2013. A literature survey of benchmark functions for global optimization problems. *CoRR abs/1308.4008*.

LUO, M. R., CUI, G., AND RIGG, B. 2001. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur 26*, 5, 340–350.

MARTINEZ-CANTIN, R. 2014. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *The Journal of Machine Learning Research 15*, 1, 3735–3739.

NELDER, J. A., AND MEAD, R. 1965. A simplex method for function minimization. *The computer journal 7*, 4, 308–313.

PARIS, S., HASINOFF, S. W., AND KAUTZ, J. 2011. Local laplacian filters: Edge-aware image processing with a laplacian pyramid. *ACM Trans. Graph. 30*, 4, 68–1.

PHILLIPS, J. B., AND ELIASSON, H. 2018. *Camera Image Quality Benchmarking*. John Wiley & Sons.

POWELL, M. 1965. A method for minimizing a sum of squares of non-linear functions without calculating derivatives. *The Computer Journal 7*, 4, 303–307.

RAGAN-KELLEY, J., ADAMS, A., PARIS, S., LEVOY, M., AMARASINGHE, S., AND DURAND, F. 2012. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *ACM Trans. Graph. 31*, 4 (July), 32:1–32:12.

RASTRIGIN, L. A. 1974. Systems of extremal control. *Nauka*.

WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing 13*, 4, 600–612.

XIAO, J., HAYS, J., EHINGER, K. A., OLIVA, A., AND TORRALBA, A. 2010. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, IEEE, 3485–3492.

YANG, Y., PRESTWOOD, S., AND BARNES, C. 2016. Vizgen: Accelerating visual computing prototypes in dynamic languages. *ACM Transactions on Graphics (Proc. SIGGRAPH ASIA) 35*, 6 (Dec.).

ZHANG, L., ZHANG, L., MOU, X., ZHANG, D., ET AL. 2011. Fsim: a feature similarity index for image quality assessment. *IEEE transactions on Image Processing 20*, 8, 2378–2386.

ZHANG, Z., LUO, P., LOY, C. C., AND TANG, X. 2014. Facial landmark detection by deep multi-task learning. In *Proc. ECCV*, 94–108.

ZHANG, R., ISOLA, P., EFROS, A. A., SHECHTMAN, E., AND WANG, O. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE International Conference on Computer Vision and Pattern Recognition*.