Learning Efficient Image Processing Pipelines

by

Michaël Gharbi

S.M., École Polytechnique (2013)
S.M., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2018

Author .....
Department of Electrical Engineering and Computer Science
August 31, 2018

Certified by .....
Frédo Durand
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Learning Efficient Image Processing Pipelines

by

Michaël Gharbi

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

The high resolution of modern cameras puts significant performance pressure on image processing pipelines. Tuning the parameters of these pipelines for speed is subject to stringent image quality constraints and requires significant efforts from skilled programmers. Because quality is driven by perceptual factors with which most quantitative image metrics correlate poorly, developing new pipelines involves long iteration cycles, alternating between software implementation and visual evaluation by human experts. These concerns are compounded on modern computing platforms, which are increasingly mobile and heterogeneous. In this dissertation, we apply machine learning towards the design of high-performance, high-fidelity algorithms whose parameters can be optimized automatically on large-scale datasets via gradient-based methods. We present applications to low-level image restoration and high-performance image filtering on mobile devices.

Thesis Supervisor: Frédo Durand
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I am grateful to my adviser, Frédo Durand for his optimism and unwavering trust. I learned a lot from Frédo's eclecticism, perseverance, and holistic thinking. I thank my thesis committee — Bill Freeman, Antonio Torralba, Wojciech Matusik, and Sylvain Paris — who never ceased to inspire me. In particular, thanks Sylvain for bringing me on board before I even had any research credentials; your mentorship has been invaluable. Thanks to my earlier teachers and mentors — Florent Gusdorf, Oliver Bournez, Edith Méthou, Dominique Obert, Marie-Paule Brisville and Thierry Devaux — who helped me find my way along this academic journey.

During my time at MIT, I have been incredibly lucky to work with brilliant researchers. Andrew Adams, Miika Aittala, Jonathan T. Barron, Gaurav Chaurasia, Jiawen Chen, Sam W. Hasinoff, Jaako Lehtinen, Tzu-Mao Li, Tomasz Malisiewicz, Lukas Murmann, Jonathan Ragan-Kelley, YiChang Shih: thank you, the work in this thesis would not have been possible without you. Andrew and Jonathan, thanks for your precious advice and also for making my first Siggraph memorable. Thanks to Marc Levoy and the rest of the GCam team for an unforgettable summer.

Along the way, the Graphics and Vision groups at MIT have been a persistent source of support, fun and intellectual stimulation; thanks to all of you. In particular, Jenny Chan, Adriana Schulz, Valentina Shin and Vlad Bychkovsky, thanks for welcoming me in the group. Luke Anderson and Prafull Sharma, it has been fun working with you, keep the lunch tradition alive! Adrian Dalca, your talent as a photographer, researcher and mentor has been a guiding star. Guha Balakrishnan, do not change the post-conference trip recipe. Zoya Bylinskii, I apologize for always antagonizing you during our frequent debates; I will miss them. You are the academic every young student should aspire to be. Andrea Tacchetti, I will never forget the countless hours we spent hacking together. Looking back, I could not have been assigned a better office at CSAIL. Thanks Tiam Jaroensri for keeping me on the run, and for sharing some Scottish delicacies. Abe Davis, your camaraderie, natural eloquence and talent

with sound-making devices has provided me with enjoyable distractions. Aleksandar Zlateski and Nishchal Bhandari, your cheers and talent at card games kept me on my toes at the occasional Thursday tournament. Thanks to our admins Bryt, Geraldine and Kshama for making sure all the lab's students were fed regularly, slept a few hours a day, and left the lab once in a while to catch some sunlight. Thanks to The Infrastructure Group: you have always delivered the friendliest help and advice, despite my uninterrupted pestering.

Beyond MIT, the many talented friends I met made life in Cambridge a bliss. I shall forever cherish these memories. Ali, our den on the Charles has truly been a home away from home. Marie Christine, thanks for philosophizing with me during our regular hot chocolate appointments. Thanks Thibaut for coming up with useful ways to procrastinate on weekends [Perol et al., 2018]. Thomas, your energy and generosity kept me moving (though I still do not understand how on Earth you can sleep so little). Thanks to the indefatigable Cambridge travel crew — Carole, Chloé, Eric, Lisa, Lucie, Rémi and Alex. Thanks to Julia, Brittany, Giulia and Paolo, Dan and Sophia, Shu and Gloria for messing around with me. Albert, Camille, Guillaume, Ombline, Ségolène, Simon, Stéphane, I have been lucky to have you close despite the physical (and sometime temporal) distance.

All my gratitude goes to my mother, Estelle and sister, Sarah, as well as the rest of my family for their patience and unconditional love. My last words are for Roberta: your warmth, smarts and optimism made the PhD a lovely jaunt.

Contents

1 Introduction **27**

1.1 Overview 28

1.2 Online content 31

2 Background **33**

2.1 Optimizing image processing algorithms 33

2.2 Automatic photographic editing 35

2.3 Neural networks for image processing 35

3 Transform Recipes for Cloud Image Processing **37**

3.1 Related work 39

3.2 Reducing data transfers 42

3.2.1 Transform Recipes 42

3.2.2 Reconstructing the filtered image 47

3.2.3 Data compression 49

3.3 Evaluation 51

3.3.1 Benchmark dataset 53

3.3.2 Expressiveness and robustness 55

3.3.3 Practical prototype system 58

4 Bilateral Learning for Real-Time Image Enhancement **65**

4.1 A fast architecture for photographic enhancement 68

4.1.1 Low-resolution prediction of bilateral coefficients 70
4.1.2 Image features as a bilateral grid 74
4.1.3 Upsampling with a trainable slicing layer 75
4.1.4 Assembling the full-resolution output 76
4.1.5 Training procedure 79
4.2 Results 80
4.2.1 Reproducing image operators 80
4.2.2 Learning from human annotations 85
4.2.3 Performance 87
4.3 Discussion and limitations 89

5 Joint Demosaicking and Denoising 93
5.1 Related work 95
5.2 Learning to jointly demosaick and denoise 97
5.2.1 Network architecture 99
5.2.2 Joint denoising with multiple noise levels 101
5.2.3 Training details 101
5.3 Curating a dataset of challenging images 102
5.3.1 Obtaining a ground-truth and the corresponding mosaic . . . 103
5.3.2 Challenging patches are rare 103
5.3.3 Mining difficult image patches 106
5.4 Results 109
5.4.1 Demosaicking noise-free images 110
5.4.2 Training set and training time 110
5.4.3 Joint denoising and demosaicking results 112
5.4.4 Processing linear data 112
5.4.5 Alternative mosaick patterns 115
5.4.6 Variations on the network configuration 115
5.4.7 Running time 115

5.5 Limitations 116

6 Conclusion **121**

6.1 Future directions 122

List of Figures

3-1 Cloud computing is often thought as an ideal solution to enable complex algorithms on mobile devices; by exploiting remote resources, one expects to reduce running time and energy consumption. However, this is ignoring the cost of transferring data over the network. When accounted for, this overhead often takes away the benefits of the cloud, especially for data-heavy photo applications. We introduce a new photo processing pipeline that reduces the amount of transmitted data. The core of our approach is the *transform recipe*, a representation of the image transformation applied to a photo that is compact and can be accurately estimated from an aggressively compressed input photo. These properties make cloud computing more efficient in situations where transferring standard jpeg-compressed images would be too costly in terms of energy and/or time. In the example above where we used our most aggressive setting, our approach reduces the amount of transferred data by about 80× compared to using images compressed with standard jpeg settings while producing an output visually similar to the ground-truth result computed directly from the original uncompressed photo. 38

3-2 A major advantage of our transform recipes is that we can evaluate them using severely degraded input photos (a,d). Processing such a strongly compressed image generates a result with numerous jpeg artifacts (b,e). We can nevertheless use such pairs of degraded input–output photos to build a recipe that, when applied on the original artifact-free photo (c) produces a high quality output (f) that closely approximates the reference output (g) directly computed from the original photo. The close-ups (c–g) show the region with highest error in our method (f). 43

3-3 To reconstruct the output image we decompose the input as a shallow Laplacian pyramid. The lowpass residual is linearly re-scaled using the ratio coefficients $R_c(p)$. For the chrominance, we don't use the complete multiscale decomposition: each block in the high-pass of the chrominance undergoes and affine transformation parameterized by $\mathbf{A}_c$ and $\mathbf{b}_c$. The luminance channel is reconstructed using a combination of an affine transformation ($\mathbf{A}_Y$, $\mathbf{b}_Y$), a piecewise linear transformation ($\{q_i\}$), and a linear scaling of the Laplacian pyramid level ($\{m_\ell\}$). . . 44

3-4 Recipe coefficients computed for the photo in Figure 3-2. We remapped the values to the $[0; 1]$ range for display purposes. (a) lowpass residual R_c. (b,c) affine coefficients of the chrominance $\mathbf{A}_c$ and $\mathbf{b}_c$. (d) affine coefficients of the luminance $\mathbf{A}_Y$ and $\mathbf{b}_Y$. (e) multiscale coefficients $\{m_\ell\}$. (f) non linear coefficients $\{q_i\}$. 50

3-5 Adding a small amount of noise to the upsampled degraded image makes the fitting process well-posed and enables our reconstruction (b) to closely approximate the ground-truth output (a). Because of the downsampling and JPEG compression, the degraded input (not shown) exhibits large flat areas (in particular in the higher Laplacian pyramid levels). Without the added noise, the fitting procedure might use the corresponding recipe coefficients as affine offsets. This generates artifacts (c) when reconstructing from the high quality input (which does have high frequency content). 52

3-6 In this close-up from a *Detail Manipulation* example, processing directly the downsampled input before fitting the recipe fails to capture the high frequencies of the transformation. Errors are particularly visible in the eyes and hair. 52

3-7 We overlap the blocks of our recipes (b) and linearly interpolate the reconstructed pixel values between overlapping blocks so as to avoid visible blocking artifacts (c). 53

3-8 The additional features to model the transformation of the luminance are critical to capture subtle local effects of the ground truth output (a). (b) Using only the affine terms, most of the effect on high frequencies is lost on the wall, and the wooden shutter. (c) Adding the Laplacian pyramid coefficients, the high frequency detail is more faithfully captured on the wall. (d) With both the non-linear and multiscale terms, our model better captures local contrast variations. This is particularly visible on the texture of the wooden shutter. 59

3-9 We analyze the quality of our reconstruction for different settings of our recipe and various degrees of input compression. Not surprisingly the reconstruction quality decreases as input and output compression increase. We report the average PSNR on the whole dataset. 60

3-10 Plot of the power consumption over time of three computation schemes: purely local, JPEG transfer, and our approach. For this measurement, we used a Samsung Galaxy S3 connected to a 3G network; the test application is Local Laplacian Filters and the input is a 4-megapixels image. Our approach uses less energy and is faster than the other two. 61

3-11 Our method handles a large variety of photographic enhancements. We filter a highly degraded copy (not shown) of the reference input (a). From the resulting degraded output, we compute the recipe parameters. We then reconstruct an output image (c) that closely approximates the reference output (b) computed from the original high-quality input. (d) and (e) are a close-up on the region of highest error in our reconstruction. (f) is a rescaled difference map that emphasizes the location of our errors. As shown on the L_0 smoothing example, our method cannot handle well filters that significantly alter the structure of the input. 62

3-12 Additional examples. We filter a highly degraded copy (not shown) of the reference input (a). From the resulting degraded output, we compute the recipe parameters. We then reconstruct an output image (c) that closely approximates the reference output (b) computed from the original high-quality input. (d) and (e) are a close-up on the region of highest error in our reconstruction. (f) is a rescaled difference map that emphasizes the location of our errors. 63

4-1 Our novel neural network architecture can reproduce sophisticated image enhancements with inference running in real time at full HD resolution on mobile devices. It can not only be used to dramatically accelerate reference implementations, but can also learn subjective effects from human retouching. 66

4-2 Our new network architecture seeks to perform as much computation as possible at a low resolution, while still capturing high-frequency effects at full image resolution. It consists of two distinct streams operating at different resolutions. The *low-resolution* stream (top) processes a downsampled version $\tilde{\mathrm{I}}$ of the input I through several convolutional layers so as to estimate a bilateral grid of affine coefficients A. This low-resolution stream is further split in two paths to learn both local features L^i and global features G^i, which are fused (F) before making the final prediction. The global and local paths share a common set of low-level features S^i. In turn, the *high-resolution* stream (bottom) performs a minimal yet critical amount of work: it learns a grayscale guidance map g used by our new *slicing* node to upsample the grid of affine coefficients back to full-resolution $\bar{A}$. These per-pixel local affine transformations are then applied to the full-resolution input, which yields the final output O. 68

4-3 Our *low-level* convolutional layers are fully learned and can extract semantic information. Replacing these layers with the standard bilateral grid splatting operation causes the network to lose much of its expressive power. In this example of our *Face brightening* operator (a-b), the network with hardcoded splatting (d) cannot detect the face properly because the grid's resolution is too low. Instead, it slightly brightens all skintones, as is visible on the hands. Our progressive downsampling with strided convolutions learns the semantic features required to solve this task properly (c), brightening only the face while darkening the background like in the reference. 71

4-4 The global features path in our architecture allows our model to reason about the full image, e.g., for subjective tasks such as reproducing subjective human adjustments that may be informed by intensity distribution or scene type (a). Without the global path, the model can make local decisions that are spatially inconsistent (b). Here, the network fails to recognize that the blue area in the top-left corner also belongs to the sky and should therefore receive the same correction as the area just below it. 73

4-5 Our new slicing node is central to the expressiveness of our architecture and its handling of high-resolution effects. Replacing this node with a standard bank of learnable deconvolution filters reduces expressiveness (b) because no full-resolution data is used to predict the output pixels. Thanks to its learned full-resolution guidance map, our slicing layer approximates the desired enhancement with much higher fidelity (c), thereby preserving the edges of the input (a) and capturing the high-frequency transformations visible in the ground-truth output (d). . . 77

4-6 The color transform matrix (left) and per-channel tone curves (right) used to produce the guidance map g, as learned by one instance of our model. 78

4-7 Our slicing node uses a learned guidance map. Using luminance as guide causes artifacts with the HDR+ pipeline reproduction, in particular with posterization artifacts in the highlights on the forehead and cheeks (b). In contrast, our learned guide (c) correctly reproduces the ground truth (d). 81

4-8 Coefficient maps for the affine color transform. The vertical axis corresponds to the learned guidance channel, while the horizontal axis unrolls the 3x4 sets of coefficients. Each thumbnail, one example of which is highlighted, shows a 16x16 low-resolution map. 81

4-9 Our method (d) can learn to replicate the correct effect (b) for operations that are not scale invariant, such as the Local Laplacian filter shown here (a–b). Methods like Bilateral Guided Upsampling that only apply the operation at low-resolution (insets (a–b)) produce a different-looking output (c). The difference is most noticeable in the areas pointed by the arrows. 84

4-10 We compare the speed and quality of our algorithm against two modern network architectures: *U-Net* (adapted from Isola et al. [2016]) and *dilated convolutions* Yu and Koltun [2015]. The runtimes were averaged over 20 iterations, processing a 4 megapixel image on a desktop CPU. The PSNR numbers refer to the *Local Laplacian* task. Given an insufficient *depth*, U-Net and dilated convolutions fail to capture the large scale effects of the Local Laplacian filter, leading to low PSNRs. Competitive architectures run over 100 times slower than ours, and use orders of magnitude more memory. Our model's performance is displayed for a range of parameters. The version we used to produce all the results is highlighted in red. See Figure 4-11 for details on the speed/quality trade-off of our model. 86

4-11 We show PSNRs for the *Local Laplacian* task and the computation time required to predict the bilateral coefficients with several settings of our model's parameters. Each curve represent a grid depth d. For each curve the grid's spatial resolution varies in $\{8, 16, 32\}$. The reference model we used to produced all the results is highlighted with a square marker. Unsurprisingly, models with larger grid depth perform better (green). Doubling the number of intermediate features also provides a 0.5 dB improvement (red curve). Runtimes were measured on an Intel Core i7-5930K. 87

4-12 At the expense of extra computation at full-resolution, our model can be extended with richer affine regression features. Here, by using a 3-level Gaussian pyramid as features ϕ, we can better capture the high-frequency details in the the *Local Laplacian (strong)* task. 90

4-13 Our algorithm fails when the image operator strongly violates our modeling assumptions. (a) Haze reduces local contrast, which limits the usefulness of our guidance map. It also destroys image details that cannot be recovered with our affine model (e.g., on the whiteboard). (b) Matting has successfully been modeled by locally affine models on 3×3 neighborhoods Levin et al. [2008]. However, this affine relationship breaks down at larger scales (like a grid cell in our model) where the matte no longer follows tonal or color variations and is mostly binary. This limits the usefulness of our bilateral grid. (c) For colorization, the learned guidance map is at best a nonlinear remapping of the grayscale input. Our model can thus only learn a local color per discrete intensity level, at a spatial resolution dictated by the grid's resolution. Our output is plagued with coarse variations of colors that are muted due to our L_2 loss (see the road line, and the tree/sky boundary). 91

4-14 Our method can learn accurate and fast approximations of a wide variety of image operators, by training on input/output pairs processed by that operator. These operators can be complicated "black box" image processing pipelines where only a binary is available, such as HDR+ or Photoshop filters/actions. Some operators, such as face-brightening, requires semantic understanding. Our model is even capable of learning from highly subjective human-annotated input/output pairs, using the MIT-Adobe FiveK dataset. 92

5-1 We propose a data-driven approach for jointly solving denoising and demosaicking. By carefully designing a dataset made of rare but challenging image features, we train a neural network that outperforms both the state-of-the-art and commercial solutions on demosaicking alone (group of images on the left, insets show error maps), and on joint denoising–demosaicking (on the right, insets show close-ups). The benefit of our method is most noticeable on difficult image structures that lead to *moiré* or *zippering* of the edges. 94

5-2 Our proposed architecture. The first layer of the network packs 2×2 blocks in the Bayer image into a 4D vector to restore translation invariance and speed up the processing. We augment each vector with the noise parameter σ to form 5D vectors. Then, a series of convolutional layers filter the image to interpolate the missing color values. We finally unpack the 12 color samples back to the original pixel grid and concatenate a masked copy of the input mosaick. We perform a last group of convolutions at full resolution this time to produce the final features. We linearly combine them to produce the demosaicked output. 98

5-3 Most of the patches in a generic training dataset (here Imagenet and Mirflickr) are easy cases for modern demosaicking algorithms (in this figure, we measure the PSNR of AHD Hirakawa and Parks [2005]). For a network to perform well in challenging situations, it needs to be trained on challenging patches, i.e., on data that lies on the tail of the patch distribution. This plot shows that our dataset contains more such patches. Further, not shown in this figure is the fact that demosaicking failures on our patches lead to more visually unpleasant artifacts: we explicitly selected the patches for this reason. 104

5-4 A network trained on a standard image dataset (second row) creates noticeable artifacts in its output such as the zippering on the thin yellow line, confusion around curves in the first and last example, and moiré in the third example. When training the same network on our new dataset of difficult cases, these artifacts are mostly gone (third row). The last row shows the difference map between the two network outputs, and the first row is the ground-truth image. (best viewed in digital form) 105

5-5 HDR-VDP run on the demosaicked output of LDI-NAT Zhang et al. [2011]. First, row full image. Second row, HDR-VDP correctly detects the zipper pattern due to luminance variations. It signals some anomalies in the output image but with a low probability of detection. It works only on luminance, therefore misses the chrominance moiré artifacts. 107

5-6 Frequency Gain due to moire 109

5-7 PSNR comparison joint denoising and demosaicking at different levels of Gaussian noise with standard deviation σ. The metric is averaged across all four datasets from Table 5.1: mcm, kodak, vdp, moiré. . . . 113

5-8 Our algorithm trained on sRGB data generalizes to real (linear) RAW images. It successfully removes color moiré on the fabric at various noise levels whereas dcraw does not (first and second rows). In comparison, Klatzer et al. [2016] does not generalize well to widely different noise levels: it denoises too aggressively the ISO100 image (first row), and produces artifacts on the ISO6400 image (second row). This is particularly visible on the smooth wall. Our output is free of checkerboard patterns and staircase artifacts (third row). DCRaw exhibits these artifacts on the red lettering and Klatzer et al. has checkboard on the blue line on the right. 113

5-9 Comparison of our approach with Adobe Camera Raw, FlexISP Heide et al. [2014] on noise-free images. Exhaustive results can be found in supplementary material. 118

5-10 Joint denoising and demosaicking results. Our approach outperforms previous best techniques on noisy data in challenging images on different levels of Gaussian noise with standard deviation σ. Exhaustive results can be found in supplementary material. 119

List of Tables

3.1 Upsampling the degraded input and adding a small amount of noise is critical to the quality of our reconstruction. We report the average PSNR on the whole dataset for a 4×4 downsampling of the input and JPEG compression with quality $Q = 50$. The recipes use a block size $w = 64$ and both the multiscale and non-linear features are enabled. 51

3.2 Reconstruction results per enhancement. $\%_{up}$ refers to the compression of the input as a fraction of the uncompressed bitmap data, $\%_{down}$ to that of the recipe (or output in case of jpeg and jdiff). For an uncompressed input, our method matches the quality of a jpeg compressed image. The benefits of become more apparent as the input image is further compressed. In the "standard" setting, the input is downsampled by a factor 2 in each dimension and JPEG compressed with a quality $Q = 30$, the block size for the recipe is $w = 32$. The jpeg and jdiff methods are given the same input, and use $Q = 80$ for the output compression. The L_0 enhancement is a failure case for our method. 56

3.3 Reconstruction results per enhancement. $\%_{up}$ refers to the compression of the input as a fraction of the uncompressed bitmap data, $\%_{down}$ to that of the recipe (or output in case of jpeg and jdiff). The "medium" setting is a 4× downsampling with $Q = 50$ and $w = 64$. The "low" quality setting is a 8× downsampling of the input with $Q = 80$ and $w = 128$. The jpeg and jdiff methods are given the same input, and use $Q = 80$ for the output compression. 57

3.4 De-activating either the luminance curve or the multiscale features, decreases the reconstruction quality by around 2 dB. We report the average PSNR on the whole dataset for a non-degraded input. The recipes use a block size $w = 64$. 58

3.5 We compare the end-to-end latency and energy cost of using our recipes for cloud processing compared to local processing of the enhancement and the jpeg baseline. Our method always saves energy and time compared to the baseline jpeg method. The measurements are from a Samsung Galaxy S5, processing a 8-megapixels image, and are averaged over 20 runs (we also report one standard deviation). 64

4.1 Details of the network architecture. *c*, *fc*, *f* and *l* refer to convolutional, fully-connected, fusion and pointwise linear layers respectively. 74

4.2 We compare accuracy to Bilateral Guided Upsampling (BGU) and Transform Recipes (TR). Note that BGU and TR are "oracle" techniques, as they run the code used to evaluate each image operator at a reduced or full resolution, and so can be thought of as providing an upper-bound on performance. Despite its disadvantage, our model sometimes performs better than these oracle baselines due its expressive power and ability to model non-scale-invariant operators. 83

4.3 Mean L_2 error in La*b* space for retouches from the 5 photographers in the MIT5k dataset (A,B,C,D,E); lower is better. Our algorithm is capable of learning a photographer's retouching style better than previous work, yet runs orders of magnitudes faster. The comparisons in the first two groups are evaluated on the dataset from photographer C favored by previous techniques; see main text for details. In the third group we report our results on the remaining 4 photographers for completeness. Metrics taken from previous work Yan et al. [2016]; Hwang et al. [2012] are denoted by †. 85

5.1 PSNR comparison of our approach to state of the art techniques on the demosaicking-only scenario. First and second column show evaluation on standard datasets. Third and fourth column show comparisons on our datasets containing images prone to luminance artifacts and color moiré respectively. No denoising is applied for any of the competing methods. (*) For Klatzer et al., we used the published model which was trained on linear data and ran it on *linearized* images. The training code was not available at the time of publication. 111

5.2 Evaluation on linear data for both noise-free and noisy data. We report PSNR in both linear and sRGB space. We feed a single estimate of the average noise level to our network a test time. We also fine tuned our network to linearized sRGB. Among all competing techniques, only KhashabiKhashabi et al. [2014] and Klatzer Klatzer et al. [2016] techniques were specifically designed for linear data. Results on noisy images are excluded from the table for methods that do not attempt to denoise. 114

5.3 Runtime of different demosaicking algorithms in their publicly available implementations. Our approach is faster than previous high quality techniques like FlexISP Heide et al. [2014]. Timings with an asterisk (*) are reported from the respective original paper. 116

Chapter 1

Introduction

The high resolution of modern cameras puts significant performance pressure on image processing pipelines. Tuning the numerous parameters of these pipelines for speed is subject to stringent image quality constraints and requires significant efforts from skilled programmers. Because quality is determined by perceptual factors with which most quantitative image metrics correlate poorly, developing new pipelines involves long iteration cycles, alternating between software implementation and visual evaluation by human experts. These concerns are compounded on modern computing platforms, which are increasingly mobile and heterogeneous.

This dissertation demonstrates that differentiable algorithms, together with carefully curated training datasets and improved image quality metrics, promise renewed and scalable progress for image processing.

We employ supervised learning towards the design of high-performance, high-fidelity algorithms whose parameters can be trained and optimized automatically on large-scale datasets via gradient-based methods. This approach has many advantages. Removing the burden of parameter-tuning shortens development cycles: the design efforts can instead be spent on maximizing an algorithm's runtime performances. Meanwhile, image quality is monitored and optimized on a carefully assembled dataset of input and output image pairs. Human supervision is therefore factored out and

happens *beforehand* rather than *alongside* development. It is implicitly captured in the training dataset which constitutes a fixed quality target for the algorithm. Further, machine learning opens up new opportunities for personalized algorithms that can adapt to a user's taste and, for instance, learn from their personal retouching style. Finally, differentiable imaging pipelines can be optimized automatically. Thus, they can afford many more parameters than traditional algorithms, well beyond what a human programmer could reasonably adjust manually. We will see that this can lead to significant image quality improvements, but also creates new challenges in designing efficient programs.

1.1 Overview

This dissertation presents efficient image processing pipelines. Its contributions range from low-level image restoration problems, to higher-level tasks such as photographic enhancement.

In chapter 3, we describe a cloud infrastructure and a local approximation technique that lay the foundations of our subsequent differentiable pipeline. This technique speeds up photographic enhancement on mobile devices, and reduces power usage by an order of magnitude. Later, we improved over this method with a novel neural network architecture, inspired by the bilateral grid [Chen et al., 2007]. Our model processes high-resolution images on a smartphone in milliseconds, provides a real-time viewfinder at 1080p resolution, and matches the quality of the best off-line approximation techniques (chapter 4). In chapter 5, we discuss an algorithm for joint image demosaicking and denoising that significantly outperforms the optimization-based state-of-the-art and runs over an order of magnitude faster. We also describe a methodology to acquire a dataset of challenging images, which was key to achieve these improvements.

Chapter 2 reviews recent work in image processing and provides background and

context to the rest of this dissertation. The remainder of this section gives a more detailed overview of the content of each chapter.

Transform Recipes (chapter 3) Cloud image processing is often proposed as a solution to the limited computing power and battery life of mobile devices: it allows complex algorithms to run on powerful servers with virtually unlimited energy supply. Unfortunately, this overlooks the time and energy cost of uploading the input and downloading the output images. When transfer overhead is accounted for, processing images on a remote server becomes less attractive and many applications do not benefit from cloud offloading. The pipeline presented in chapter chapter 3 aims to change this in the case of image enhancements that preserve the overall content of an image. Our key insight is that, in this case, the server can compute and transmit a description of the *transformation* from input to output, which we call a *transform recipe*. At equivalent quality, our recipes are much more compact than JPEG images: this reduces the client's download. Furthermore, recipes can be computed from highly compressed inputs which significantly reduces the data uploaded to the server. The client reconstructs a high-fidelity approximation of the output by applying the recipe to its local high-quality input. We demonstrate our results on 168 images and 10 image processing applications, showing that our recipes form a compact representation for a diverse set of image filters. With an equivalent transmission budget, they provide higher-quality results than JPEG-compressed input/output images, with a gain of the order of 10 dB in many cases. We demonstrate on a mobile phone prototype that a transform recipe-based pipeline runs 2-4× faster and uses 2-7× less energy than both a local or a naive cloud computation.

Real-time photographic enhancement on mobile (chapter 4) Performance is a critical challenge in mobile image processing. Given a reference imaging pipeline, or even human-adjusted pairs of images, we seek to reproduce the enhancements and enable real-time evaluation. For this, we introduce a new neural network architecture

inspired by bilateral grid processing and local affine color transforms. Using pairs of input/output images, we train a convolutional neural network to predict the coefficients of a locally-affine model in bilateral space. Our architecture learns to make local, global, and content-dependent decisions to approximate the desired image transformation. At runtime, the neural network consumes a low-resolution version of the input image, produces a set of affine transformations in bilateral space, upsamples those transformations in an edge-preserving fashion using a new *slicing* node, and then applies those upsampled transformations to the full-resolution image. Our algorithm processes high-resolution images on a smartphone in milliseconds, provides a real-time viewfinder at 1080p resolution, and matches the quality of state-of-the-art approximation techniques on a large class of image operators. Unlike previous work, our model is trained off-line from data and therefore does not require access to the original operator at runtime. This allows our model to learn complex, scene-dependent transformations for which no reference implementation is available, such as the photographic edits of a human retoucher.

Low-level image processing: joint demosaicking and denoising (chapter 5) Demosaicking and denoising are the key first stages of the digital imaging pipeline, but they are also a severely ill-posed problem that infers three color values per pixel from a single noisy measurement. Earlier methods rely on hand-crafted filters or priors and still exhibit disturbing visual artifacts in hard cases such as moiré or thin edges. We introduce a new data-driven approach for these challenges: we train a neural network on a large corpus of images instead of using hand-tuned filters. While deep learning has shown great success, its naive application using existing training datasets does not give satisfactory results for our problem because these datasets lack hard cases. To create a better training set, we present metrics to identify difficult patches and techniques for mining community photographs for such patches. Our experiments show that this network and training procedure outperform state-of-the-art both on

noisy and noise-free data. Furthermore, our algorithm is an order of magnitude faster than the previous best performing techniques.

1.2 Online content

This dissertation is largely based on work that has appeared in ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2015, Proceedings of SIGGRAPH Asia 2016, and Proceedings of SIGGRAPH 2017) [Gharbi et al., 2015, 2016, 2017]. Video overviews of these publications, together with the software implementations and datasets related to them can be found online at `http://www.mgharbi.com`.

Chapter 2

Background

In this chapter, we provide a general overview of related works in image processing. Literature relevant to specific contributions of this dissertation is further discussed in the corresponding chapters.

2.1 Optimizing image processing algorithms

Image enhancement algorithms have been the focus of a great deal of research, but most sophisticated algorithms remain too expensive to be evaluated quickly on mobile devices, where the vast majority of digital images are captured and processed.

Previous works have identified specific critical operations and developed novel algorithms to accelerate them. For instance, Farbman et al. [2011a] introduced *convolution pyramids* to accelerate linear translation-invariant filters. Similarly, many approaches have been proposed to accelerate bilateral filtering, due to the ubiquity of edge-aware image processing [Tomasi and Manduchi, 1998; Paris and Durand, 2006; Chen et al., 2007; Adams et al., 2010].

Systems contributions have also sought to facilitate the implementation of high-performance executables e.g. [Ragan-Kelley et al., 2012a; Hegarty et al., 2014; Mullapudi et al., 2016]. They typically require programmer expertise, and their runtime

cost still grows with the complexity of the pipeline.

One way to accelerate an operator is to simply apply it at low resolution and upsample the result. A naïve upsampling will generally lead to an unacceptably blurry output, but this issue can often be ameliorated by using a more sophisticated upsampling technique that respects the edges of the original image. Joint bilateral upsampling [Kopf et al., 2007] does this by using a bilateral filter on a high-resolution guidance map to produce a piecewise-smooth edge-aware upsampling. Bilateral space optimization [Barron et al., 2015; Barron and Poole, 2016] builds upon this idea by solving a compact optimization problem inside a bilateral grid, producing upsampled results which are maximally smooth.

In chapter 3 we focus on learning the *transformation* from input to output instead of the output itself. Our model can approximate a large class of complex, spatially-varying operators with a collection of simple local models—which we call *transform recipe*—that is tailored to a given input/output pair. The task of computing the operator and fitting the recipe is offloaded to the cloud while the mobile device need only apply the recipe, thereby saving time and energy. In a similar fashion, Chen et al. [2016] approximate an image operator with a grid of local affine models in bilateral space, the parameters of which are fit to an input/output pair in a manner resembling the guided filter [He et al., 2013]. By performing this model-fitting on a low-resolution image pair, this technique enables real-time on-device computation. In chapter 4, we build upon this bilateral space representation, but rather than fitting a model to approximate a single instance of an operator from a pair of images, we construct a rich CNN-like model that is trained to apply the operator to any unseen input. This bypasses the need for the original operator at runtime and opens up the opportunity to learn non-algorithmic transformations (i.e., hand-adjusted input/output image pairs). This also allows us to optimize the affine coefficients to model the operator running at full resolution, which is important for filters that vary with scale.

2.2 Automatic photographic editing

Several prior work seeked to train models to automatically correct photographs from input/output image pairs provided by a human retoucher. This task was introduced by Bychkovsky et al. [2011b], who estimate global brightness/contrast adjustments that characterize the personal style of 5 trained photographers. They train a regression model with handcrafted features that capture both low-level information and semantic content (e.g., faces) on a dataset of 5000 raw images. Hwang et al. [2012] approach the problem with a coarse-to-fine search for the best-matching scenes that takes more than a minute for a 500×333 image. Kaufman et al. [2012a] learn local color and contrast manipulations from hard-coded features (faces, blue skies, clouds, underexposed areas), running over 2 minutes for a VGA image. More recently, Yan et al. [2016] use a compact pixel-wise neural network and handcrafted features. Their network takes 1.5 s to process a 1 megapixel image (on top of the time needed for object detection, dense image segmentation, and scene recognition used in their features). The model we present in chapter 4 can learn similar global tonal adjustments and generalizes to more complex effects, including color corrections and local edits, in addition to being much faster.

2.3 Neural networks for image processing

Convolutional neural networks have revolutionized classification problems in computer vision Krizhevsky et al. [2012]; Szegedy et al. [2015]; Simonyan and Zisserman [2014]. Recently, they have achieved significant progress on low-level vision and image processing tasks such as pixel-wise object segmentation Long et al. [2015a]; Badrinarayanan et al. [2015]; Noh et al. [2015], monocular depth and normals estimation Eigen et al. [2014]; Wang et al. [2015], view interpolation Flynn et al. [2016], deconvolution Xu et al. [2014], filter approximation Xu et al. [2015], image colorization Cheng et al. [2015]; Zhang et al. [2016]; Larsson et al. [2016]; Iizuka et al. [2016a], style-transfer

Gatys et al. [2016], optical flow Dosovitskiy et al. [2015a]; Ilg et al. [2016], image inpainting Eigen et al. [2013]; Pathak et al. [2016], super-resolution [Dong et al., 2014], image matting [Shen et al., 2016], image synthesis Dosovitskiy et al. [2015b] and general image-to-image "translation" tasks [Isola et al., 2016]. The model we propose in chapter 5 follows a similar approach. It is tailored to the complex, yet crucial problem of joint demosaicking and denoising raw image samples to produce clean color images. As we shall see, this efficient architecture does not suffice to improve over the state-of-the-art, optimization-based techniques. Our new data acquisition scheme turns out to be essential.

Recent work has even explored training deep networks within a bilateral grid [Jampani et al., 2016] though this work does not directly address image transformations in that space, and instead focuses on classification and semantic segmentation. Some architectures have been trained to approximate a general class of imaging operators. Xu et al. [2015] develop a three-layer network in the gradient domain to accelerate edge-aware smoothing filters. Liu et al. [2016] propose an architecture to learn recursive filters for denoising, image-smoothing, inpainting and color interpolation. They jointly train a collection of recursive networks and a convolutional network to predict image-dependent propagation weights. While some of this work can process low-resolution images on a desktop GPU at interactive rates, they remain too slow for practical use. The algorithm of chapter 4 processes high-resolution images on mobile, in real time.

Chapter 3

Transform Recipes for Cloud Image Processing

Mobile devices such as cell phones and cameras are exciting platforms for computational photography, but their limited computing power, memory and battery life can make it challenging to implement advanced algorithms. Cloud computing is often hailed as a solution allowing connected devices to benefit from the speed and infinite power supply of remote servers. Cloud processing is also imperative for algorithms that require large databases, e.g. [Laffont et al., 2014; Shih et al., 2013a] and can streamline applications development, especially in the face of the mobile platform fragmentation. Unfortunately, the cost of transmitting the input and output images can dwarf that of local computation, making cloud solutions surprisingly expensive both in time and energy [Barr and Asanović, 2006; Huang et al., 2012]. For instance, a 6 seconds computation on a 16-megapixels image captured by a Samsung Galaxy S5 would consume 20J. Processing the same image on the cloud would take 14 seconds and draw 54J[1]. In short, efficient cloud photo enhancement requires a reduction of the data transferred that is beyond what lossy image compression can achieve while keeping

[1]Transferring the jpeg images with their default compression settings (4MB each way), over a 4G network with bandwidth 4Mbits/s, according to our estimate of the average power draw and modeling like in [Kumar et al., 2013]

image quality high.

In this work, we focus on photographic enhancements that preserve the overall content of an image (in the sense of style vs. content) and do not spatially warp it. This includes content-aware photo enhancement [Kaufman et al., 2012b], dehazing [Kim et al., 2013], edge-preserving enhancements [Paris et al., 2011a; Farbman et al., 2008], colorization [Levin et al., 2004], style transfer [Shih et al., 2014; Aubry et al., 2014a], time-of-day hallucination [Shih et al., 2013a] but excludes dramatic changes such as inpainting. For this class of enhancements, the input and output images are usually highly correlated. We exploit this observation to construct a representation of the

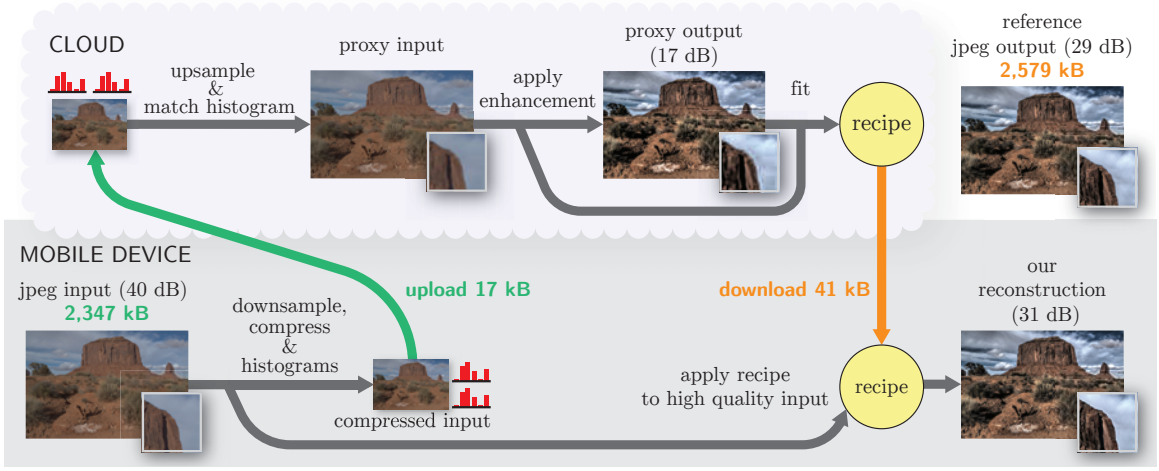


Figure 3-1: Cloud computing is often thought as an ideal solution to enable complex algorithms on mobile devices; by exploiting remote resources, one expects to reduce running time and energy consumption. However, this is ignoring the cost of transferring data over the network. When accounted for, this overhead often takes away the benefits of the cloud, especially for data-heavy photo applications. We introduce a new photo processing pipeline that reduces the amount of transmitted data. The core of our approach is the *transform recipe*, a representation of the image transformation applied to a photo that is compact and can be accurately estimated from an aggressively compressed input photo. These properties make cloud computing more efficient in situations where transferring standard jpeg-compressed images would be too costly in terms of energy and/or time. In the example above where we used our most aggressive setting, our approach reduces the amount of transferred data by about 80× compared to using images compressed with standard jpeg settings while producing an output visually similar to the ground-truth result computed directly from the original uncompressed photo.

transformation from input to output that we call a *transform recipe*. By design, recipes are much more compact than images. And because they capture the *transformation* applied to an image, they are forgiving to a lower input quality. These are the key properties that allow us to cut down the data transfers. With our method, the mobile client uploads a downsampled and highly degraded image instead of the original input, thus reducing the upload cost. The server upsamples this image back to its original resolution and computes the enhancement. From this low quality input-output pair, the server fits the recipe and sends it back to the client instead of the output, incurring a reduced download cost. In turns, the client combines the recipe with the original high quality input to reconstruct the output (Fig. 3-1). While our recipes are lossy, they provide a high-fidelity approximations of the full enhancement yet are one to two orders of magnitude more compact.

Transform recipes use a combination of local affine transformations of the YUV channels of a shallow Laplacian pyramid as well as local tone curves for the highest pyramid level. This makes them flexible and expressive, so as to adapt to a variety of enhancements, with a moderate computational footprint on the mobile device. We evaluate the quality of our reconstruction on several standard photo editing tasks including detail enhancement, general tonal and color adjustment, and recolorization. We demonstrate that in practice, our approach translates into shorter processing times and lower energy consumption with a proof-of-concept implementation running on a smartphone instrumented to measure its power usage.

3.1 Related work

Image compression Image compression has been a long standing problem in image processing [Rabbani and Jones, 1991]. The data used to represent digital images is reduced by exploiting their structure and redundancy. Lossless image compression methods and formats rely on general purpose compression techniques such as run-

length encoding (BMP format), adaptive dictionary algorithm [Ziv and Lempel, 1977; Welch, 1984], entropy coding [Huffman, 1952; Witten et al., 1987] or a combination thereof [Deutsch, 1996]. These methods lower the data size of natural images to around 50% to 70% that of an uncompressed bitmap, which in practice is not sufficient for cloud image processing tasks. Alternatively, lossy image compression methods offer more aggressive compression ratios at the expense of some visible degradation. They minimize the loss of information relevant to the human visual system based on heuristics (e.g. using chroma sub-sampling). The most widespread lossy compression methods build on transform coding techniques: the image is transformed to a different basis, and the transform coefficients are quantized. Popular transforms include the Discrete Cosine Transform (JPEG [Wallace, 1992]) and the Wavelet Transform (JPEG2000 [Skodras et al., 2001]). Images compressed using these methods can reach 10% to 20% of their original size with little visible degradation but often, this is still not sufficient for cloud photo applications. More recently, methods that exploit the intra-frame predictive components of video codecs have emerged (WebP, BPG). While they do improve the compression/quality trade-off, the improvements at very low bit-rates are marginal, and these methods remain uncommon. Unlike traditional image compression techniques, we seek to compress the description of a transformation from the input of an image enhancement to the output. We build upon, and use traditional image compression techniques as building blocks in our strategy: any lossy compression technique can be used to compress the input image before sending it to the cloud. JPEG is a natural candidate since it is ubiquitous and often implemented in hardware on mobile devices. We also use lossless image compression to further compress our recipes.

In the context of video games, Lee et al. [2014] describe a data compression scheme to minimize network transfers. While the overall objective is related to ours, the targeted application and the associated constraints are different, and the proposed method does not apply in our context. Levoy [1995] proposes a collaborative strategy

for high-quality image rendering on the cloud. The server generates a high and a low-quality renderings. It sends the compressed difference image to a client who combines it with its own local low-quality rendering to produce the final output. We similarly re-distribute the workload between client and server but our goal is to reduce data transfer.

Recipes as a regression on image data Using the input image to predict the output is inspired from shape recipes [Freeman and Torralba, 2002; Torralba and Freeman, 2003]. In the context of stereo shape estimation, shape recipes are regression coefficients that predict band-passed shape information from an observed image of this shape. They form a low-dimensional representation of the high-dimensional shape data. Shape recipes describe the shape in relation to the image, and constitute a set of rules to reconstruct the shape from the image. These recipes are simple as they let the image bear much of the complexity of the representation. Similarly, our transform recipes capture the transformation between the input and output images, factoring out their structural complexity. Since our goal is to reduce data transfers, our model needs to remain simple whereas shape recipes do not have this constraint and use many more parameters. Our recipes also differ from shape recipes in that we have a notion of spatially varying transformation: we fit different models for each block in a grid subdivision of the image. In contrast, shape recipes are applied globally to an image sub-band. Finally, our recipes are computed from low quality images and applied back to the high quality input whereas shape recipes have to be computed from and applied to high quality data.

Other methods also share similar ideas to our recipes although in a different context. For instance, Bychkovsky et al. [2011c] and Berthouzoz et al. [2011] use a generic representation of photographic edits that they use in conjunction with machine learning to assist users with adjusting their pictures whereas we focus on speed and compactness in the context of cloud applications. Farbman et al. [2011b] introduce

convolution pyramids as a means to accelerate linear translation-invariant image filters. Mantiuk and Mantiuk and Seidel [2008] use a generic representation of tone mapping operators to speed them up and analyze them. In comparison, we are interested in a wider range of possibly spatially varying edits such as stylization and detail enhancement, and we specifically study their latency and energy consumption when computed in the cloud.

Energy saving LiKamWa et al. [2013] describe an approach to reduce the energy consumption when taking a picture with a mobile device. Our work is complementary and seeks to reduce the energy consumption and latency when editing photographs.

3.2 Reducing data transfers

We reduce the upload cost by downsampling and aggressively compressing the input image I. This yield a degraded image $\bar{I}$. Upon reception by the server, $\bar{I}$ is upsampled back to the original resolution. Let us denote this new low quality image by $\tilde{I}$. The server processes $\tilde{I}$ to generate a low fidelity output $\tilde{O}$. The server then assembles a recipe $f_{\tilde{I},\tilde{O}}$ such that $f_{\tilde{I},\tilde{O}}(I) \approx O$, where O is the ground-truth output that would result from the direct processing of I. Recipes are designed to be compact and therefore incur a low download cost (Fig. 3-1).

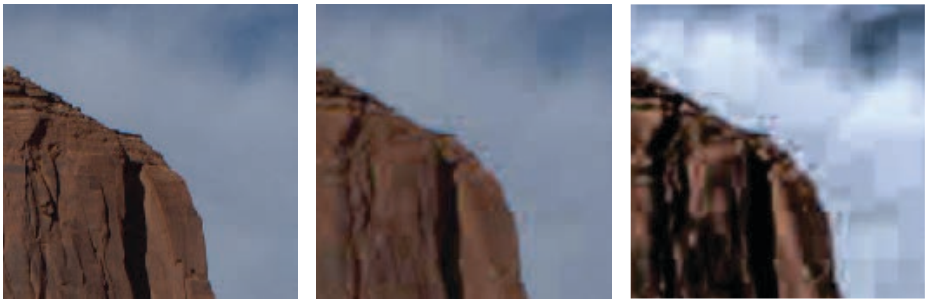
To reconstruct the final output, the client downloads the recipe and combines it with the original high-quality input I (Fig. 3-3). Our objective with recipes is to represent a variety of standard photo enhancements. In particular, we seek to capture effects that may be spatially varying, multi-scale, and nonlinear.

3.2.1 Transform Recipes

We define recipes over $w \times w$ blocks where w controls the trade-off between compactness and accuracy. Small blocks yield a fine-grained description of the transformation but

(a) degraded input sent to the server
4×4 bicubic downsampling
JPEG compression Q = 20

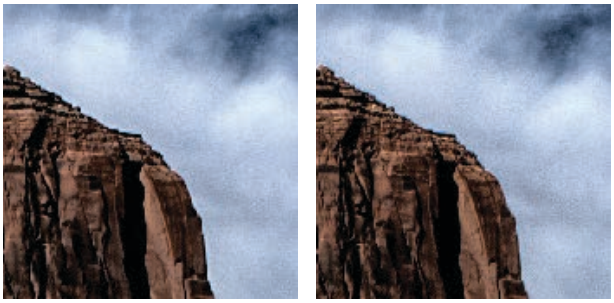
(b) output of the filter
applied to the degraded input
PSNR = 18.8 dB



(c) original full
quality input

(d) degraded input
close-up on (a)

(e) output of the filter
close-up on (b)



(f) our reconstruction
$\%_{\text{down}} = 0.4\%$
PSNR = 32.6 dB

(g) reference output
computed on the
full-quality input

Figure 3-2: A major advantage of our transform recipes is that we can evaluate them using severely degraded input photos (a,d). Processing such a strongly compressed image generates a result with numerous jpeg artifacts (b,e). We can nevertheless use such pairs of degraded input–output photos to build a recipe that, when applied on the original artifact-free photo (c) produces a high quality output (f) that closely approximates the reference output (g) directly computed from the original photo. The close-ups (c–g) show the region with highest error in our method (f).

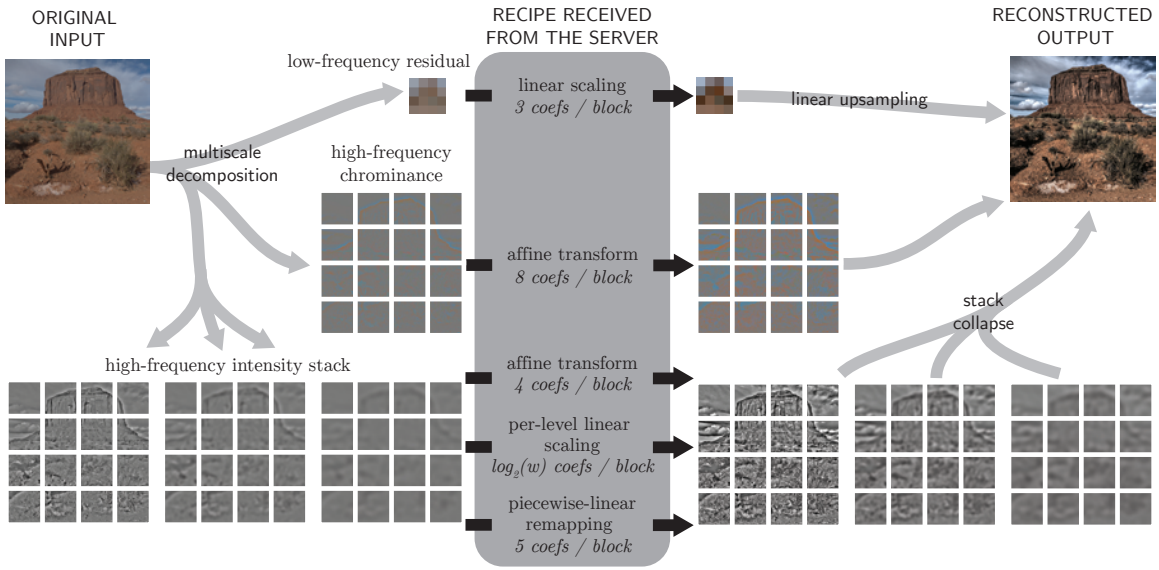


Figure 3-3: To reconstruct the output image we decompose the input as a shallow Laplacian pyramid. The lowpass residual is linearly re-scaled using the ratio coefficients $R_c(p)$. For the chrominance, we don't use the complete multiscale decomposition: each block in the high-pass of the chrominance undergoes and affine transformation parameterized by $\mathbf{A}_c$ and $\mathbf{b}_c$. The luminance channel is reconstructed using a combination of an affine transformation ($\mathbf{A}_Y$, $\mathbf{b}_Y$), a piecewise linear transformation ($\{q_i\}$), and a linear scaling of the Laplacian pyramid level ($\{m_\ell\}$).

require more storage space whereas large blocks form a coarser and more concise description. We use overlapping blocks to allow for smooth transitions between blocks and prevent blocking artifacts (Fig 3-7), that is, we lay out the $w \times w$ blocks on a $\frac{w}{2} \times \frac{w}{2}$ grid so that each pair of adjacent blocks share half their area. We work in the same YC_bC_r color space as JPEG to separate the luminance Y from the chrominance (C_b, C_r) while still manipulating values that span $[0; 255]$ [Hamilton, 1992].

To capture multi-scale effects, we decompose the images using Laplacian pyramids [Burt and Adelson, 1983]. That is, we split I and O into $n+1$ levels $\{L_\ell[I]\}$ and $\{L_\ell[O]\}$ where the resolution of each level is half that of the preceding one. The first n levels represent the details at increasingly coarser scales and the last level is a low-frequency residual. We set $n = \log_2(w)$ so that each block is represented by a single pixel in the residual. We compute a recipe in three steps: we first represent the

transformation of the residual, then that of the other pyramid levels, and finally we quantize and encode the result of the first two steps in an off-the-shelf compressed format.

Low-pass residual

We found that even small errors in the residual propagate to large areas in the final reconstruction and can produce conspicuous artifacts in smooth regions like skies. Since the residual is only a small fraction of the data, we can afford to represent it with greater precision.

We represent the low-frequency part of the transformation by the ratio of the residuals, i.e., for each pixel p and each channel $c \in \{Y, C_\mathrm{b}, C_\mathrm{r}\}$:

$$R_c(p) = \frac{L_n[O_c](p) + 1}{L_n[I_c](p) + 1} \tag{3.1}$$

where 1 is added to prevent divisions by zero and to ensure that R_c is constant when $L_n[O_c] = L_n[I_c]$, that is, for edits like sharpening that only modify the high frequencies. This compresses better in the final stage.

Modeling the high Frequencies

Representing the high-frequency part of the transformation requires much more data. Using per-pixel ratios would not be competitive with standard image compression. To model the high-frequency effect of the transformation, we combine all the levels of the pyramids except the residual into a single layer:

$$H[I] = \sum_{\ell=0}^{n-1} \mathrm{up}(L_\ell[I]) \tag{3.2}$$

where $\mathrm{up}(\cdot)$ is an operator that upsamples the levels back to the size of the original image I. In practice, we use the property of Laplacian pyramids that all the levels

including the residual add up to the original image to compute $H[I] = I - \text{up}(L_n[I])$, which is more efficient. We use the same scheme to compute $H[O]$.

Next, we process $H[I]$ and $H[O]$ block by block independently. Our strategy is to fit a parametric function to the pixels within each block to represent their transformation. In our early experiments, we found that while the chrominance transformation were simple enough to be well approximated by an affine function, no single simple parametric function was able to capture alone the diversity of luminance transformations generated by common photographic edits. Instead, our strategy is to rely on a combination of several functions that depend on many parameters and use a sparse regression to keep only a few of them in the final representation.

Chrominance channels

Let O_{CC} denote the two chrominance channels of O. $O_{CC}(p)$ is the the 2D vector containing the chrominance values of O at pixel p. Within each $w \times w$ block $\mathcal{B}$, we model the high frequencies of the chrominance $H[O_{CC}](p)$ as an affine function of $H[I](p)$. We use a standard least-squares regression and minimize:

$$\sum_{p \in \mathcal{B}} \| H[O_{CC}](p) - \mathbf{A}_\text{c}(\mathcal{B}) H[I](p) - \mathbf{b}_\text{c}(\mathcal{B}) \|^2 \tag{3.3}$$

where $\mathbf{A}_\text{c}$ and $\mathbf{b}_\text{c}$ are the 2×3 matrix and the 2D vector defining the affine model.

Luminance channel

We represent the high frequencies of the luminance $H[O_Y]$ with a more sophisticated function comprising several components. The first component is an affine function akin to that used for the chrominance channel: $\mathbf{A}_Y(\mathcal{B}) H[I](p) + b_Y(\mathcal{B})$ with $\mathbf{A}_Y$ and b_Y the 1×3 matrix and the scalar constant defining the affine function. Intuitively, this affine component captures the local changes of brightness and contrast. Next, we capture scale-dependent effects with a linear function of the pyramid levels:

$\sum_{\ell=0}^{n-1} m_\ell(\mathcal{B})\,\mathrm{up}(L_\ell[I_Y])$ where $\{m_\ell(\mathcal{B})\}$ are the coefficients of the linear combination within the block $\mathcal{B}$ and $\mathrm{up}(\cdot)$ is used to ensure that all levels are upsampled back to the original resolution. Last, we add a term to account for the nonlinear effects. We use a piecewise linear function made of p segments over the luminance range of the block. To define this function, we introduce $p-1$ regularly spaced luminance values $y_i = \min_{\mathcal{B}} H[I_Y] + \frac{i}{p}(\max_{\mathcal{B}} H[I_Y] - \min_{\mathcal{B}} H[I_Y]),\ i \in \{1,\ldots,p-1\}$ and the segment functions $s_i(\cdot) = \max(\cdot - y_i, 0)$ where we omit the dependency on $\mathcal{B}$ of y_i and s_i for clarity's sake. Intuitively, s_i generates a unit change of slope at the i^{th} node y_i. Equipped with these functions, we define our nonlinear term as $\sum_{i=1}^{p-1} q_i(\mathcal{B}) s_i(H[I_Y])$ where the $\{q_i(\mathcal{B})\}$ coefficients control the change of slope between two consecutive linear segments.

To fit the complete model, we use LASSO regression [Tibshirani, 1994], i.e., we minimize a standard least-squares term:

$$\sum_{p\in\mathcal{B}} \Big\| H[O_Y](p) - \mathbf{A}_Y(\mathcal{B}) H[I](p) - b_Y(\mathcal{B}) - \sum_{\ell=0}^{n-1} m_\ell(\mathcal{B})\,\mathrm{up}(L_\ell[I_Y])(p) - \sum_{i=1}^{p-1} q_i(\mathcal{B}) s_i(H[I_Y](p)) \Big\|^2 \quad (3.4)$$

to which we add the L_1 norm of the free parameters $\mathbf{A}_Y(\mathcal{B})$, $b_Y(\mathcal{B})$, $\{m_\ell(\mathcal{B})\}$, and $\{q_i(\mathcal{B})\}$.

Solving the optimization problems (Eq. 3.3 and Eq. 3.4) for each overlapping block, and computing the lowpass ratio (Eq. 3.1), we now have all the coefficients of the recipe (Fig. 3-4). In the next section, we describe how we reconstruct the output image from the full quality input image and these recipe coefficients.

3.2.2 Reconstructing the filtered image

We reconstruct the output O using the recipe described in the previous section and the input I. We proceed in two steps; we first deal with the multiscale part of the recipe

and then with its high-frequency component. This process is illustrated in Figure 3-3.

From Equation 3.1, for each pixel p and each channel $c \in \{Y, C_\mathrm{b}, C_\mathrm{r}\}$, we get the expression for the lowpass residual:

$$L_n[O_c](p) = R_c(p)\big(L_n[I_c](p) + 1\big) - 1 \tag{3.5}$$

Then, we reconstruct the intensity levels $\ell \in [0; n-1]$ of the multiscale component of Equation 3.4:

$$L_\ell[\hat{O}_Y] = m_\ell L_\ell[I_Y] \tag{3.6}$$

Together with the luminance channel of Equation 3.5, this gives a complete Laplacian pyramid $\{L_\ell[\hat{O}_Y]\}$. We collapse this pyramid to get an intermediate output luminance channel $\hat{O}_Y$. At this point, the reconstruction is missing some high-frequency luminance components from Equation 3.4 and the chrominance channels C_r and C_b.

To reconstruct the remaining high-frequency information $H[O]$, we first compute $H_\mathcal{B}[O]$ within each block then linearly blend the results of overlapping blocks. For the chrominance, we apply the affine remapping that we estimated in Equation 3.3:

$$H_\mathcal{B}[O_{CC}](p) = \mathbf{A}_\mathrm{c}(\mathcal{B})H[I](p) + \mathbf{b}_\mathrm{c}(\mathcal{B}) \tag{3.7}$$

And for the intensity channel, we apply the affine remapping and the nonlinear functions that we computed in Equation 3.4:

$$H_\mathcal{B}[O_Y](p) = \mathbf{A}_Y(\mathcal{B})H[I](p) + b_Y(\mathcal{B}) + \sum_{i=1}^{p-1} q_i(\mathcal{B})s_i(H[I_Y](p)) \tag{3.8}$$

We linearly interpolate $H_\mathcal{B}[O_{CC}]$ and $H_\mathcal{B}[O_Y]$ between overlapping blocks to get $H[O_{CC}]$ and $H[O_Y]$

Finally, we put all the pieces together. We linearly upsample the chrominance residual (Eq. 3.5) and add the multiscale component (Eq. 3.6), and the linearly

interpolated high-frequency chrominance (Eq. 3.7) and luminance (Eq. 3.8):

$$O = \mathrm{up}(L_n[O_{CC}]) + \hat{O}_Y + H[O_{CC}] + H[O_Y] \tag{3.9}$$

where we implicitly lift the quantities in the right hand side into $YC_{\mathrm{b}}C_{\mathrm{r}}$ vectors depending on the channels they contain.

3.2.3 Data compression

Once computed by the server, we assemble the recipe coefficients described in Section 3.3.1 as a multi channel image (Fig.3-4). Each channel of the recipe is independently uniformly quantized to 8 bits. In practice the quantization error is small and does not affect the quality of the reconstruction. (less than 0.1 dB)

We compress the coefficient maps using an off the shelf standard lossless image compression method. In our implementation, we save the lowpass residual as a 16-bits float TIFF file, and the highpass coefficients as tiles in a 8-bits grayscale PNG file. This compression takes benefit both from the spatial redundancy of the recipe and the sparsity induced by the LASSO regression and further reduces the data by about a factor of 2.

Upstream Compression

So far, we have only described how to cut down the data downloaded by the client using transform recipes. Because recipes capture the *transformation* applied to an image they are resilient to degradation of the input (Fig. 3-2). We leverage this property and reduce the data uploaded to the server by sub-sampling and compressing the input image using a lossy compression algorithm with aggressive settings (we use JPEG). Upon receiving the data, the server *upsamples* the degraded image back to its original resolution and adds a small amount of Gaussian noise to it ($\sigma = 0.5\%$ of the range, see Section 3.2.3). The server then processes the upsampled image to

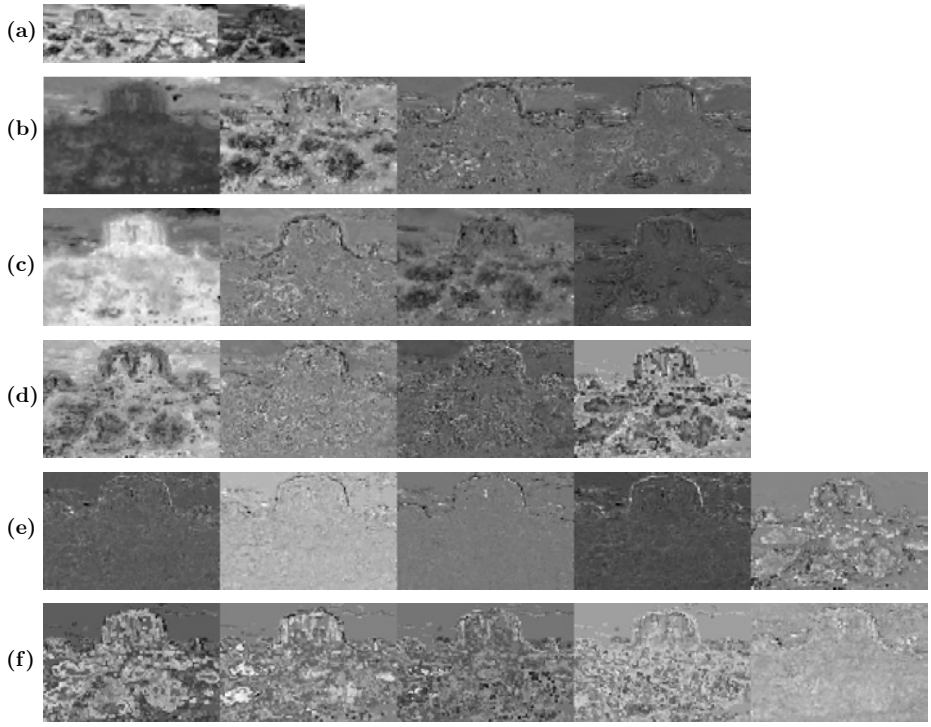


Figure 3-4: Recipe coefficients computed for the photo in Figure 3-2. We remapped the values to the $[0; 1]$ range for display purposes. (a) lowpass residual R_c. (b,c) affine coefficients of the chrominance $\mathbf{A}_c$ and $\mathbf{b}_c$. (d) affine coefficients of the luminance $\mathbf{A}_Y$ and $\mathbf{b}_Y$. (e) multiscale coefficients $\{m_\ell\}$. (f) non linear coefficients $\{q_i\}$.

produce a low quality output. The fitting procedure of Section 3.2.1 is applied to this low quality input/output pair. The client in turn reconstructs the output image by combining his high quality input with the recipe.

Input Pre-processing

Adding noise serves two purposes. Since the downsampling and JPEG compression both drastically reduce the high frequency content of the input image, adding noise allows us to re-introduce some high frequencies from which we can estimate the transformation. Furthermore, the low quality of the input can make the fitting procedures ill-posed which creates severe artifacts at reconstruction time (Fig. 3-5). The added noise acts as a regularizer.

Alternatively we could apply the desired image enhancement directly to the low

resolution input, which would also lower the computational cost. This would require adapting the parameters of the enhancement accordingly but in certain cases, it is unclear how the low-resolution processing relates to the high-resolution processing. Besides, with this approach, we cannot estimate the high-frequency part of the transformation. As shown in Figure 3-6, this approach fails for enhancements that have a non trivial high-frequency component. We summarize the importance of upsampling the input and adding noise in Table 3.1.

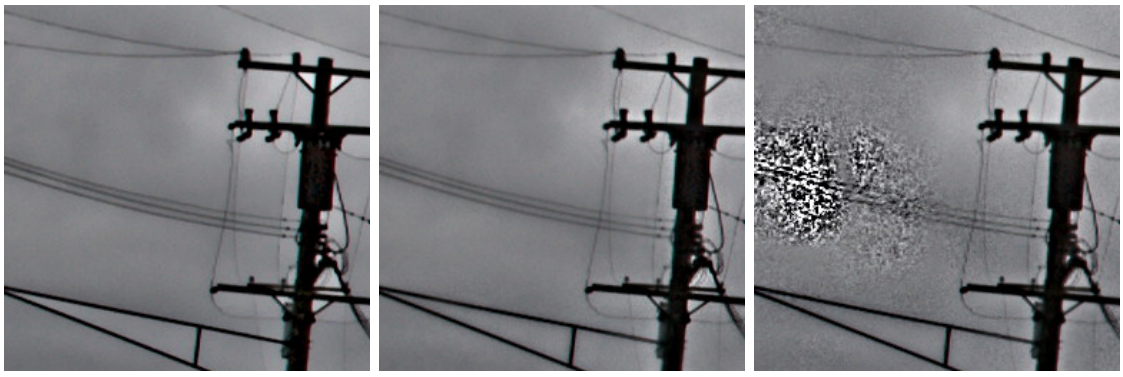
	with noise	without noise
with upsampling	**34.9 dB**	33.5 dB
without upsampling	28.7 dB	27.6 dB

Table 3.1: Upsampling the degraded input and adding a small amount of noise is critical to the quality of our reconstruction. We report the average PSNR on the whole dataset for a 4×4 downsampling of the input and JPEG compression with quality $Q = 50$. The recipes use a block size $w = 64$ and both the multiscale and non-linear features are enabled.

3.3 Evaluation

We evaluate the quality of our reconstruction on several image processing applications (§ 3.3.1). We conduct stress tests by degrading the quality of the input to several degrees and analyze the effect of our design choices (§ 3.3.2). Latency and physical energy consumption are measured using a proof-of-concept implementation on an Android smartphone (§ 3.3.3).

We express the compression performance as a fraction of the original, uncompressed 24-bits bitmap. Though an uncompressed bitmap would never be used in the context of cloud image processing, this provides us with a fixed reference for all our comparisons. A typical out-of-the-phone JPEG has a data ratio in the 10% to 15% realm. A PNG image has a typical data ratio of 50% to 75% for natural color images. Besides visual comparison, we report both PSNR and SSIM to quantify the quality of our



(a) reference

(b) our method PSNR = 34.7 *dB*

(c) no noise added to the upsampled input before processing, PSNR = 28.1 *dB*

Figure 3-5: Adding a small amount of noise to the upsampled degraded image makes the fitting process well-posed and enables our reconstruction (b) to closely approximate the ground-truth output (a). Because of the downsampling and JPEG compression, the degraded input (not shown) exhibits large flat areas (in particular in the higher Laplacian pyramid levels). Without the added noise, the fitting procedure might use the corresponding recipe coefficients as affine offsets. This generates artifacts (c) when reconstructing from the high quality input (which does have high frequency content).

(a) reference

(b) our method PSNR = 34.7 *dB*

(c) no input upsampling before processing, PSNR = 24.2 *dB*

Figure 3-6: In this close-up from a *Detail Manipulation* example, processing directly the downsampled input before fitting the recipe fails to capture the high frequencies of the transformation. Errors are particularly visible in the eyes and hair.

(a) reference (b) our method PSNR = 42.1 dB (c) no block overlap, PSNR = 34.7 dB

Figure 3-7: We overlap the blocks of our recipes (b) and linearly interpolate the reconstructed pixel values between overlapping blocks so as to avoid visible blocking artifacts (c).

reconstruction.

3.3.1 Benchmark dataset

We selected a range of applications representative of typical photo editing scenarios. Some of these applications might not necessarily require cloud processing, but they demonstrate the expressiveness of our model. We gathered 168 high quality test images from Flickr and the MIT-Adobe fiveK dataset [Bychkovsky et al., 2011c] to use as input to the image processing algorithms. The image resolution range between 2-megapixels and 8-megapixels. MIT-Adobe fiveK dataset provides raw images that give us baseline for reconstruction quality.

From these high quality inputs, we generated degraded inputs for several quality levels using a combination of bicubic downsampling (2 to 8 times smaller images in each dimension) and aggressive JPEG compression (with quality parameters ranging from 30 to 80, using Adobe Photoshop's quantization tables). Our JPEG compression uses the quantization tables from Adobe Photoshop. This new set of images illustrate what a typical client would send for processing in the cloud using our technique.

We ran each algorithm on this large set of inputs for different quality settings of

the degraded input and measured end-to-end approximation fidelity.

Photo editing and Photoshop actions We manually applied various image editing tools to the input images and recorded them as Photoshop Actions. The edits include tonal adjustments, color correction, non-trivial masking, spatially varying blurs, unsharp masking, high pass boost, vignetting, white balance adjustments, and detail enhancement. The set of operations include both global and local edits, and were created as a stress test for our model. These actions are an example of filters that would simply not be available on a mobile device.

Dehazing This algorithm [Kim et al., 2013] estimates an atmospheric light vector from the input image, computes a transmission map and removes the corresponding haze. The transmission map, and therefore the transformation applied to the input image exhibits sharp transitions but they follow the input's content.

Edge preserving detail enhancement We test both the Local Laplacian Filter [Paris et al., 2011a] and another edge-preserving decomposition [Farbman et al., 2008]. These methods are challenging for our approach, which does not have any notion of edge preservation.

Style Transfer [Aubry et al., 2014a] alters the distribution of gradients in an image to match a target distribution using the Local Laplacian filter. The algorithm outputs an image whose style matches that of the examples image.

Portrait Style Transfer [Shih et al., 2014] spatially matches the input image to a target example whose style is to be imitated using dense correspondence. Because of the correspondence step and the database query, this algorithm is a good candidate for cloud processing. The multi-scale, spatially varying transfer is of interest to test our recipe. The algorithm also adds post-processing not well tied to our model (adding highlights in the eyes). We disabled this post-processing during our evaluation.

Time of Day Hallucination [Shih et al., 2013a] is an example of algorithm that requires a large database and costly computer vision matching.

Colorization [Levin et al., 2004] requires solving a large sparse linear system which becomes costly on mobile platforms for large images. We input scribbles on images from the MIT5K dataset.

Matting Is beyond the scope of our work since the output image does not resembles a photography. It is nonetheless useful in the photo editing context and is a good stress test for expressiveness. We use the implementation of KNN-matting available on the authors' webpage [Chen et al., 2012], and use as inputs the publicly available data from [Rhemann et al., 2009].

L_0 **smoothing** [Xu et al., 2011] is a paradigm of methods where the stylized output is significantly different from the input. We include it as a failure case.

3.3.2 Expressiveness and robustness

We processed our entire dataset at various levels of compression and found that in all cases except the most extreme settings our approach produces outputs visually similar to the ground truth, i.e., the direct processing of the uncompressed input photo. Figure 3-11 Figure 3-12 and shows a few example reconstruction for the "standard" quality setting. We compare our technique to two baseline alternatives set to match our data rate:

- *jpeg*: the client sends a JPEG, the server processes it and transmit the output back as a JPEG image.
- *jdiff*: the client sends a JPEG input image. The server sends back the difference between this input and the output it computes from it. The difference is also

quality	enhancement	#	$\%_{up}$	$\%_{down}$ ours	$\%_{down}$ jpeg	$\%_{down}$ jdiff	PSNR (dB) ours	PSNR (dB) jpeg	PSNR (dB) jdiff	SSIM ours	SSIM jpeg	SSIM jdiff
non-degraded input	LocalLaplacian	20	71.7	1.7	9.1	7.3	**38.1**	35.6	36.6	**0.97**	0.94	0.94
$w = 32$	Dehazing	20	51.4	1.4	9.8	7.6	**42.6**	39.0	41.8	**0.98**	0.95	0.97
	Detail Manipulation	20	71.7	1.6	5.6	4.2	39.0	37.1	**41.7**	0.98	0.94	**0.98**
	L_0	12	62.5	2.0	3.7	7.8	36.7	**42.6**	35.4	0.95	**0.98**	0.84
	Matting	12	41.7	1.0	3.1	7.2	37.1	**48.3**	42.7	0.97	**1.00**	0.96
	Photoshop	10	68.3	1.4	11.9	9.4	44.3	42.4	**44.8**	**0.99**	0.98	0.98
	Recoloring	10	33.4	0.8	3.2	1.1	**49.3**	42.5	47.7	**1.00**	0.96	0.98
	Portrait Transfer	20	31.5	1.1	3.9	2.5	**42.9**	41.6	41.3	**0.99**	0.98	0.97
	Time of Day	24	47.5	1.8	14.0	12.4	**42.7**	39.9	40.8	**0.98**	0.98	0.97
	Style Transfer	20	76.8	1.1	15.3	12.4	37.3	**43.4**	42.0	0.97	**0.99**	0.98
	all	168	55.6	1.4	7.9	7.2	41.0	41.2	**41.5**	**0.98**	0.97	0.96
standard	LocalLaplacian	20	0.8	1.8	2.4	2.1	**37.0**	26.0	26.7	**0.97**	0.67	0.70
$2\times$ downsampling	Dehazing	20	0.9	1.4	2.1	1.7	**36.1**	28.0	28.9	**0.97**	0.75	0.79
$Q = 30$	Detail Manipulation	20	0.8	1.7	1.9	1.7	**36.8**	27.9	28.8	**0.98**	0.81	0.85
$w = 32$	L_0	12	1.8	2.1	1.8	2.9	**35.8**	34.7	33.0	0.94	**0.95**	0.89
	Matting	12	0.8	0.9	1.0	2.5	**35.2**	32.0	32.1	0.96	**0.97**	0.92
	Photoshop	10	0.8	1.5	2.1	1.9	**42.6**	29.6	30.4	**0.99**	0.80	0.82
	Recoloring	10	0.7	1.4	1.0	0.4	**47.3**	38.2	39.4	**0.99**	0.92	0.94
	Portrait Transfer	20	0.8	1.1	1.3	1.1	**41.1**	32.9	33.8	**0.99**	0.89	0.90
	Time of Day	24	1.5	1.8	2.9	2.9	**38.6**	26.7	27.3	**0.97**	0.75	0.78
	Style Transfer	20	1.0	1.1	2.8	2.3	**34.9**	25.6	26.0	**0.96**	0.67	0.70
	all	168	1.0	1.5	1.9	2.0	**38.5**	30.2	30.6	**0.97**	0.82	0.83

Table 3.2: Reconstruction results per enhancement. $\%_{up}$ refers to the compression of the input as a fraction of the uncompressed bitmap data, $\%_{down}$ to that of the recipe (or output in case of jpeg and jdiff). For an uncompressed input, our method matches the quality of a jpeg compressed image. The benefits of become more apparent as the input image is further compressed. In the "standard" setting, the input is downsampled by a factor 2 in each dimension and JPEG compressed with a quality $Q = 30$, the block size for the recipe is $w = 32$. The jpeg and jdiff methods are given the same input, and use $Q = 80$ for the output compression. The L_0 enhancement is a failure case for our method.

quality	enhancement	#	$\%_{up}$	$\%_{down}$ ours	$\%_{down}$ jpeg	$\%_{down}$ jdiff	PSNR (dB) ours	PSNR (dB) jpeg	PSNR (dB) jdiff	SSIM ours	SSIM jpeg	SSIM jdiff
medium	Local Laplacian	20	0.3	0.5	0.7	0.6	**34.6**	23.3	23.6	**0.96**	0.55	0.56
4× downsampling	Dehazing	20	0.3	0.4	0.6	0.5	**34.3**	26.1	26.7	**0.95**	0.68	0.70
$Q = 50$	Detail Manipulation	20	0.3	0.5	0.6	0.5	**33.9**	24.3	24.7	**0.97**	0.70	0.72
$w = 64$	L_0	12	0.7	0.6	0.6	0.9	**32.5**	29.9	29.6	**0.90**	0.90	0.88
	Matting	12	0.3	0.4	0.3	0.9	**32.9**	28.9	29.0	0.94	**0.95**	0.91
	Photoshop	10	0.3	0.4	0.6	0.6	**40.0**	26.9	27.3	**0.98**	0.74	0.75
	Recoloring	10	0.3	0.4	0.4	0.1	**44.6**	35.6	36.3	**0.99**	0.89	0.91
	Portrait Transfer	20	0.4	0.4	0.4	0.4	**36.7**	30.1	30.6	**0.97**	0.83	0.83
	Time of Day	24	0.6	0.6	0.9	0.9	**35.8**	24.8	25.1	**0.95**	0.66	0.68
	Style Transfer	20	0.4	0.3	0.8	0.7	**32.8**	23.0	23.2	**0.94**	0.56	0.57
	all	168	0.4	0.5	0.6	0.6	**35.8**	27.3	27.6	**0.96**	0.75	0.75
low	Local Laplacian	20	0.1	0.2	0.2	0.2	**32.0**	21.3	21.4	**0.95**	0.45	0.46
8× downsampling	Dehazing	20	0.1	0.1	0.2	0.2	**31.5**	24.3	24.7	**0.93**	0.63	0.64
$Q = 80$	Detail Manipulation	20	0.1	0.1	0.2	0.2	**31.0**	21.8	22.1	**0.95**	0.63	0.64
$w = 128$	L_0	12	0.3	0.2	0.2	0.3	**30.1**	26.5	26.7	**0.87**	0.85	0.85
	Matting	12	0.1	0.2	0.1	0.3	**30.2**	25.9	26.0	0.92	**0.94**	0.90
	Photoshop	10	0.1	0.1	0.2	0.2	**36.2**	25.0	25.3	**0.97**	0.71	0.72
	Recoloring	10	0.1	0.1	0.1	0.0	**40.9**	32.6	32.9	**0.99**	0.87	0.87
	Portrait Transfer	20	0.2	0.1	0.2	0.1	**31.8**	27.6	27.8	**0.94**	0.78	0.78
	Time of Day	24	0.3	0.2	0.3	0.3	**33.5**	23.3	23.4	**0.93**	0.61	0.61
	Style Transfer	20	0.2	0.1	0.2	0.2	**30.3**	20.8	20.9	**0.92**	0.46	0.47
	all	168	0.2	0.1	0.2	0.2	**32.7**	24.9	25.1	**0.94**	0.69	0.69

Table 3.3: Reconstruction results per enhancement. $\%_{up}$ refers to the compression of the input as a fraction of the uncompressed bitmap data, $\%_{down}$ to that of the recipe (or output in case of jpeg and jdiff). The "medium" setting is a 4× downsampling with $Q = 50$ and $w = 64$. The "low" quality setting is a 8× downsampling of the input with $Q = 80$ and $w = 128$. The jpeg and jdiff methods are given the same input, and use $Q = 80$ for the output compression.

JPEG compressed. The client adds the difference back to its local input.

As shown in Table 3.2 and Table 3.3, for all applications, our approach performs equally or better than these alternatives, often by a large margin. Figure 3-9 illustrates the robustness of our recipes for various compression ratios of the input and output. We recommend two level of compression: "standard" that corresponds to about 1.5% of the data and generates images often indistinguishable from the ground truth, and "medium quality" that corresponds to about 0.5% of the data and only introduce minor deviations. In practice, one would choose the level of accuracy/compression depending on the application.

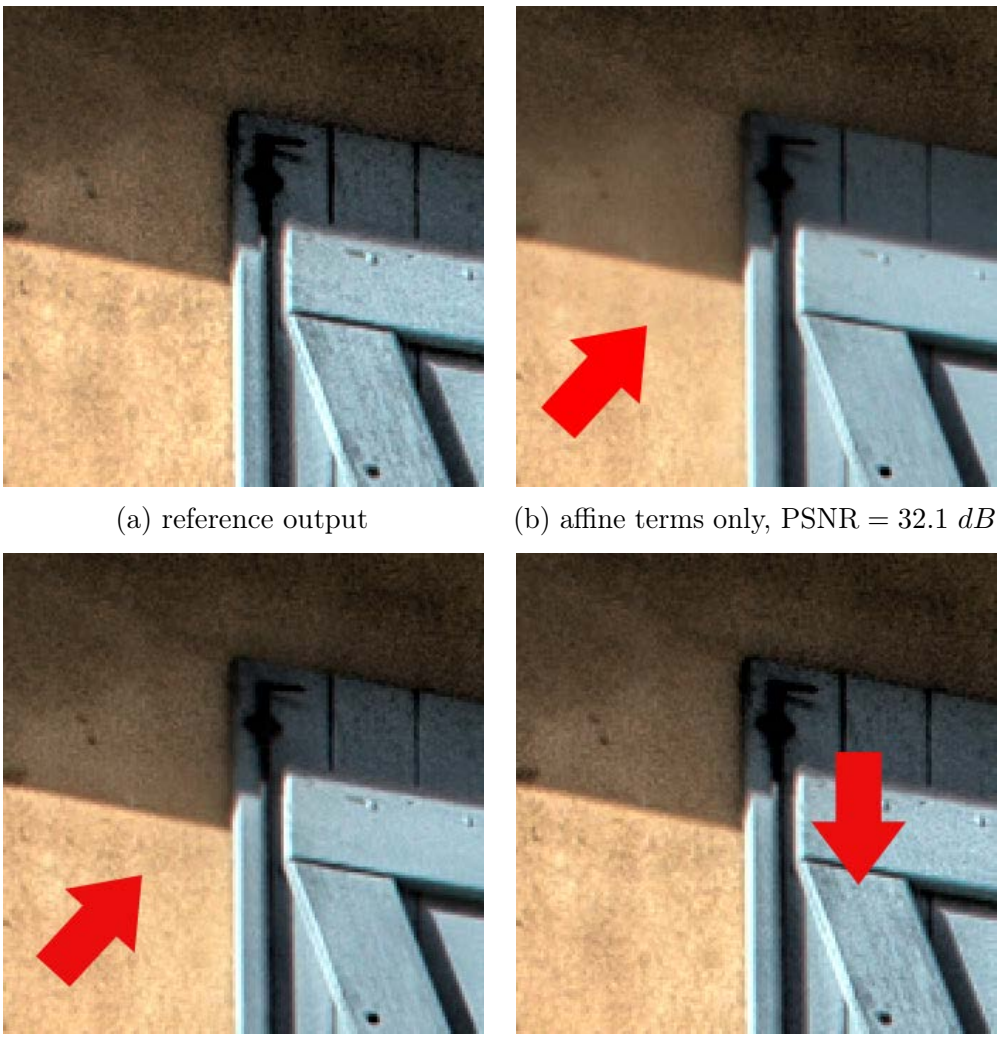
The extra features for the luminance channel are critical to the expressiveness of our model. Both the non-linear luminance curve and the multiscale features help capture subtle and local effects (Fig. 3-8). We quantify the improvement these features provide in Table 3.4.

	with luminance curve	without
with multiscale	**40.2 dB**	38.3 dB
without	38.5 dB	36.6 dB

Table 3.4: De-activating either the luminance curve or the multiscale features, decreases the reconstruction quality by around 2 dB. We report the average PSNR on the whole dataset for a non-degraded input. The recipes use a block size $w = 64$.

3.3.3 Practical prototype system

We implemented a client-server poof-of-concept system to validate runtime and energy consumption. We use a Samsung Galaxy S5 as the client device. Three image processing scenarios are tested: on-device processing, naive cloud processing — where we transfer the input and output compressed as JPEG with the default settings from the cellphone, and our recipe-based processing. We consider network connections through WIFI and cellular LTE network. Our server runs an Intel Xeon E5645 CPU, using 12 cores.For on-device processing we use a Halide-optimized



(a) reference output

(b) affine terms only, PSNR = 32.1 dB

(c) affine and multiscale terms for the luminance, PSNR = 33.6 dB

(d) affine, multiscale and non-linear terms for the luminance, PSNR = 36.4 dB

Figure 3-8: The additional features to model the transformation of the luminance are critical to capture subtle local effects of the ground truth output (a). (b) Using only the affine terms, most of the effect on high frequencies is lost on the wall, and the wooden shutter. (c) Adding the Laplacian pyramid coefficients, the high frequency detail is more faithfully captured on the wall. (d) With both the non-linear and multiscale terms, our model better captures local contrast variations. This is particularly visible on the texture of the wooden shutter.

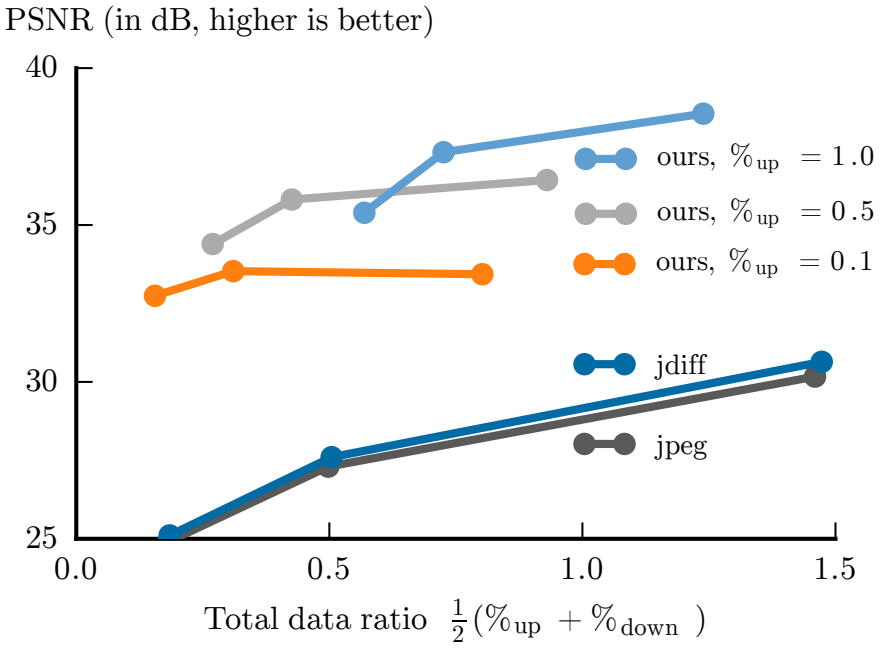


Figure 3-9: We analyze the quality of our reconstruction for different settings of our recipe and various degrees of input compression. Not surprisingly the reconstruction quality decreases as input and output compression increase. We report the average PSNR on the whole dataset.

implementation [Ragan-Kelley et al., 2013b, 2012b] of the test filters, through the Java Native Interface on Android. We evaluate local Laplacian filter [Paris et al., 2011a] with fifty pyramids for detail enhancement, on an 8-megapixels input.

To ensure accurate power measurement, we replace the cellphone battery with a power generator, and record the current and voltage drained from the generator. The power usage plot in Fig. 3-10 shows the energy consumption at each step for the three scenarios on the Local Laplacian filter. By reducing the amount of data transferred both upstream and downstream, our method greatly reduces transmission costs and cuts down both end-to-end computation time and power usage (Fig. 3.5).

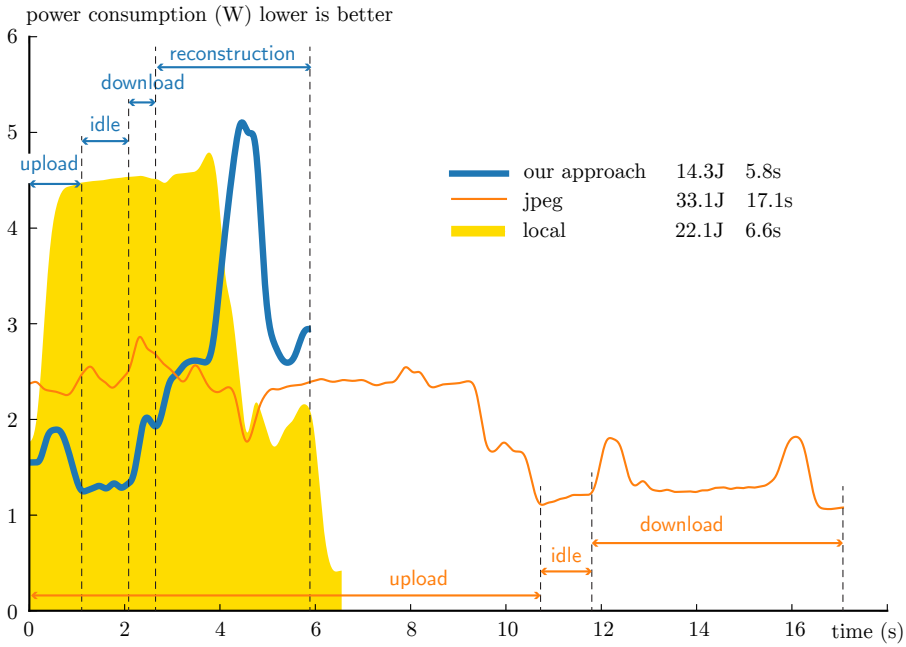
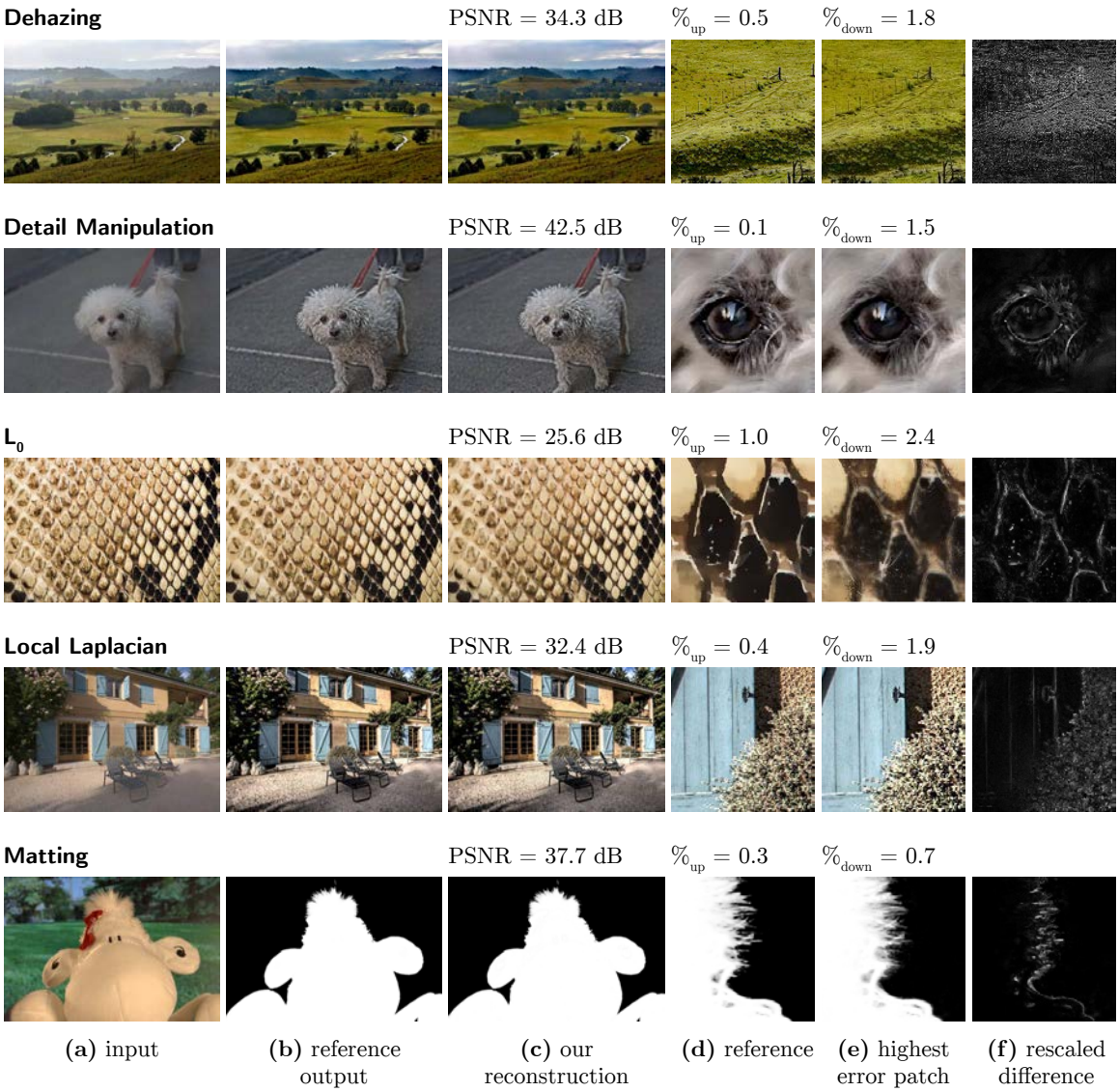


Figure 3-10: Plot of the power consumption over time of three computation schemes: purely local, JPEG transfer, and our approach. For this measurement, we used a Samsung Galaxy S3 connected to a 3G network; the test application is Local Laplacian Filters and the input is a 4-megapixels image. Our approach uses less energy and is faster than the other two.



(a) input | **(b)** reference output | **(c)** our reconstruction | **(d)** reference | **(e)** highest error patch | **(f)** rescaled difference

Figure 3-11: Our method handles a large variety of photographic enhancements. We filter a highly degraded copy (not shown) of the reference input (a). From the resulting degraded output, we compute the recipe parameters. We then reconstruct an output image (c) that closely approximates the reference output (b) computed from the original high-quality input. (d) and (e) are a close-up on the region of highest error in our reconstruction. (f) is a rescaled difference map that emphasizes the location of our errors. As shown on the L_0 smoothing example, our method cannot handle well filters that significantly alter the structure of the input.

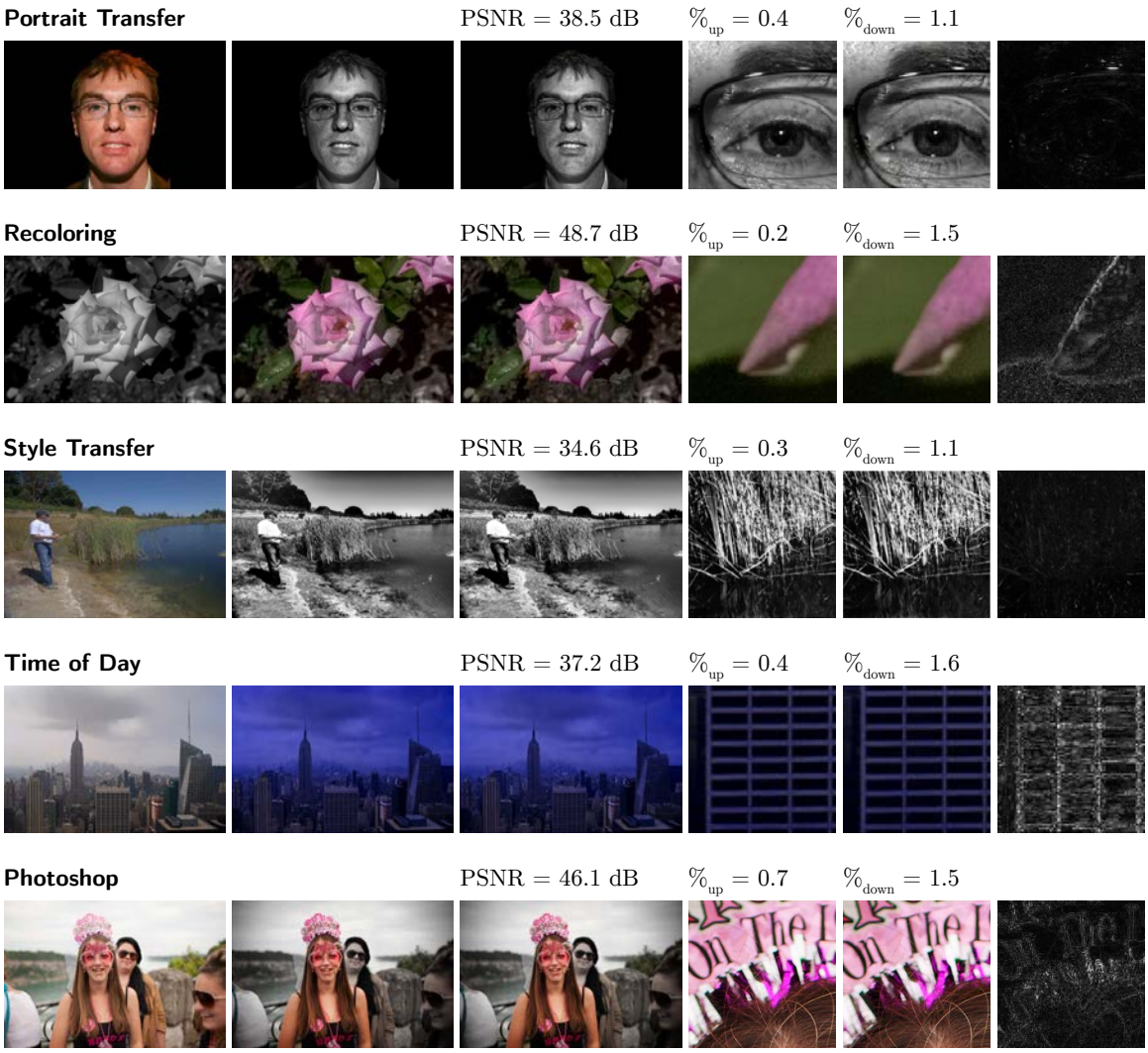


Figure 3-12: Additional examples. We filter a highly degraded copy (not shown) of the reference input (a). From the resulting degraded output, we compute the recipe parameters. We then reconstruct an output image (c) that closely approximates the reference output (b) computed from the original high-quality input. (d) and (e) are a close-up on the region of highest error in our reconstruction. (f) is a rescaled difference map that emphasizes the location of our errors.

		LTE		WIFI	
enhancement	local	jpeg	recipe	jpeg	recipe
Local Laplacian	23.2 ± 1.5 J	30.6 ± 2.4 J	10.6 ± 1.2 J	14.6 ± 0.6 J	7.9 ± 1.2 J
	7.0 ± 0.5 s	14.7 ± 1.9 s	4.9 ± 0.2 s	9.2 ± 0.3 s	3.9 ± 0.1 s
Style Transfer	135.6 ± 2.4 J	44.1 ± 0.7 J	20.7 ± 1.5 J	30.5 ± 0.7 J	23.2 ± 1.3 J
	30.6 ± 0.9 s	23.6 ± 2.2 s	12.4 ± 0.7 s	17.3 ± 1.5 s	11.4 ± 0.3 s

Table 3.5: We compare the end-to-end latency and energy cost of using our recipes for cloud processing compared to local processing of the enhancement and the jpeg baseline. Our method always saves energy and time compared to the baseline jpeg method. The measurements are from a Samsung Galaxy S5, processing a 8-megapixels image, and are averaged over 20 runs (we also report one standard deviation).

Chapter 4

Bilateral Learning for Real-Time Image Enhancement

In chapter 3, we presented a client-server infrastructure to off-load costly computations to the cloud. The image operator was approximated with simple, locally-affine models that are compact and easy to transmit over the network. These models were then combined with the input image on the mobile client to produce the final output, in an efficient manner. In this chapter, we take a step further and lift the requirements for a powerful remote server altogether.

We present a machine learning approach where the effect of a reference filter, pipeline, or even subjective manual photo adjustment is learned by a deep network that can be evaluated quickly and with cost independent of the reference's complexity. As in chapter 3, we focus on photographic enhancements that do not spatially warp the image or add new edges, e.g. [Aubry et al., 2014b; Hasinoff et al., 2016].

We share the motivation of prior work that seeks to accelerate "black box" image processing operations, for example by processing a low-resolution image and then using the low-resolution output to approximate a high-resolution equivalent [Gharbi et al., 2015; Chen et al., 2016]. For some operations, these approaches can achieve large speedups but they suffer from significant limitations: the underlying image processing

Figure 4-1: Our novel neural network architecture can reproduce sophisticated image enhancements with inference running in real time at full HD resolution on mobile devices. It can not only be used to dramatically accelerate reference implementations, but can also learn subjective effects from human retouching.

operation must be somewhat scale-invariant (Figure 4-9), and must be fast to evaluate at low resolution. In addition, these techniques rely on the availability of an explicit reference implementation, and therefore cannot be used to learn an implicitly-defined operation from a database of human annotated input/output pairs.

Many deep learning architectures have been used for image-to-image transformations, e.g. [Long et al., 2015b; Xu et al., 2015; Liu et al., 2016; Yan et al., 2016; Isola et al., 2016]. However, most prior work incur a heavy computational cost that scales linearly with the size of the input image, usually because of the large number of stacked convolutions and non-linearities that must be evaluated at full resolution. This general form allows for flexible models to be learned, but this expressivity comes at a price: such architectures are orders of magnitude too slow for real-time viewfinder applications, requiring seconds to process a 1 megapixel image on the best desktop GPUs—more than 1000× slower than our proposed model (2 ms on GPU). Our speedup is enabled by specifically targeting photographic transformations, which are often well-approximated with linear operations in bilateral space [Chen et al., 2016], and accordingly learning our model in this space.

We present a new network architecture that is capable of learning a rich variety of photographic image enhancements and can be rapidly evaluated on high-resolution

inputs. We achieve this through three key strategies:

1) We perform most predictions in a low-resolution bilateral grid [Chen et al., 2007], where each pixel's x, y coordinates are augmented with a third dimension which is a function of the pixel's color. To do this, we introduce a new node for deep learning that performs a data-dependent lookup. This enables the so-called slicing operation, which reconstructs an output image at full image resolution from the 3D bilateral grid by considering each pixel's input color in addition to its x, y location. 2) We follow previous work which has observed that it is often simpler to predict the *transformation* (chapter 3) from input to output rather than predicting the output directly e.g., [Shih et al., 2013b; Gharbi et al., 2015; Chen et al., 2016]. This is why our architecture is designed to learn, as an intermediate representation, a local affine color transformation that will be applied to the input through a new multiplicative node. 3) While most of our learning and inference is performed at low resolution, the loss function used during training is evaluated at full resolution, which causes the low-resolution transformations we learn to be directly optimized for their impact on high-resolution images.

Taken together, these three strategies (slicing, affine color transform, and full-resolution loss) allow us to perform the bulk of our processing at a low resolution (thereby saving substantial compute cost) yet reproduce the high-frequency behavior of the reference operator.

We demonstrate the expressiveness of our model on a benchmark of 7 applications including: approximating published image filters [Aubry et al., 2014b; Hasinoff et al., 2016], reverse-engineering black-box Photoshop actions, and learning the retouching style of photographers [Bychkovsky et al., 2011b] from a set of manually corrected photographs. Our technique produces output whose quality is comparable to or better than previous work, while being more widely applicable by not requiring some reference implementation of the image operation being approximated, being end-to-end learnable from input/output image pairs, and running in real-time on mobile hardware. The forward pass of our network takes 14 ms to process a full screen resolution 1920×1080

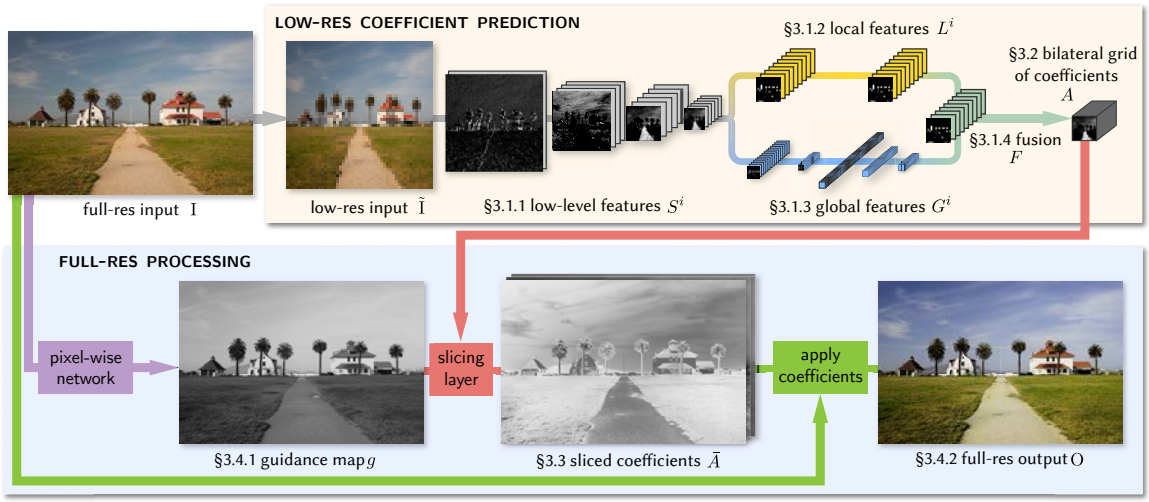


Figure 4-2: Our new network architecture seeks to perform as much computation as possible at a low resolution, while still capturing high-frequency effects at full image resolution. It consists of two distinct streams operating at different resolutions. The *low-resolution* stream (top) processes a downsampled version $\tilde{I}$ of the input I through several convolutional layers so as to estimate a bilateral grid of affine coefficients A. This low-resolution stream is further split in two paths to learn both local features L^i and global features G^i, which are fused (F) before making the final prediction. The global and local paths share a common set of low-level features S^i. In turn, the *high-resolution* stream (bottom) performs a minimal yet critical amount of work: it learns a grayscale guidance map g used by our new *slicing* node to upsample the grid of affine coefficients back to full-resolution $\bar{A}$. These per-pixel local affine transformations are then applied to the full-resolution input, which yields the final output O.

image on a Google Pixel phone, thereby enabling real-time viewfinder effects at 50 Hz.

4.1 A fast architecture for photographic enhancement

We propose a new convolutional network architecture that can be trained to perform fast image enhancement (Figure 4-2). Our model is designed to be expressive, preserve edges, and require limited computation at full resolution. It is fully end-to-end trainable and runs in real-time at 1080p on a modern smartphone.

We perform most of the inference on a low-resolution copy $\tilde{I}$ of the input I in the

low-res stream (Fig. 4-2, top), which ultimately predicts local affine transforms in a representation similar to the bilateral grid [Chen et al., 2016]. In our experience, image enhancements often depend not only on local image features but also on global image characteristics such as histograms, average intensity, or even scene category.

Therefore, our low-res stream is further split into a *local* path and a *global* path. Our architecture then fuses these two paths to yield the final coefficients representing the affine transforms.

The *high-res* stream (Fig. 4-2, bottom) works at full resolution and performs minimal computation but has the critical role of capturing high-frequency effects and preserving edges when needed. For this purpose, we introduce a slicing node inspired by bilateral grid processing [Paris and Durand, 2006; Chen et al., 2007]. This node performs data-dependent lookups in the low-resolution grid of affine coefficients based on a learned *guidance map*. Given high-resolution affine coefficients obtained by slicing into the grid with the full-resolution guidance map, we apply local color transforms to each pixel to produce the final output O. At training time, we minimize our loss function at *full resolution*. This means that the low-res stream, which only processes heavily downsampled data, still learns intermediate features and affine coefficients that can reproduce high-frequency effects.

As a first approximation, one can think of our work as alleviating the need for the reference filter at runtime in Chen et al. [2016]'s Bilateral Guided Upsampling. In a sense, we seek to predict the affine color transform coefficients in the bilateral grid given a low-resolution version of the image. However, there are several key elements that go beyond this. First, the downsampling into the bilateral grid is learned. Second, the guidance image is also learned and not restricted to luminance. Finally, we apply the loss function not on the affine coefficients, but on the final image at full resolution, which allows us to capture high-frequency effects and handle operators that are not scale-invariant (Figure 4-9). We illustrate the role of each component of our architecture with an ablation study in Figures 4-3, 4-4, 4-5 and 4-7.

4.1.1 Low-resolution prediction of bilateral coefficients

The input $\tilde{\mathrm{I}}$ to the low-res stream has a fixed resolution 256×256. It is first processed by a stack of strided convolutional layers $(S^i)_{i=1,\ldots,n_S}$ to extract *low-level* features and reduce the spatial resolution. Then, in a design inspired by Iizuka et al. [2016b], the last low-level features are processed by two asymmetric paths: the first path $(L^i)_{i=1,\ldots,n_L}$ is fully convolutional [Long et al., 2015b] and specializes in learning local features that propagate image data while retaining spatial information. The second path $(G^i)_{i=1,\ldots,n_G}$ follows the design of standard classification networks [Krizhevsky et al., 2012] and learns uses both convolutional and fully-connected layers to learn a fixed-size vector of global features (e.g. high-level scene category, indoor/outdoor, etc.) with a receptive field covering the entire low-resolution image $\tilde{\mathrm{I}}$. The outputs of the two paths, G^{n_G} and L^{n_L}, are then fused into a common set of features F. A pointwise linear layer outputs a final array A from the fused streams. We interpret this array as a bilateral grid of affine coefficients (Section 4.1.2). Since we produce a 3D bilateral grid from a 2D image in a content-dependent fashion, we can view the *low-res* stream as implementing a form of *learned splatting*.

Low-level features

We first process the low-resolution image $S^0 := \tilde{\mathrm{I}}$ with a stack of standard strided convolutional layers with stride $s = 2$ (Figure 4-2):

$$S^i_c[x, y] = \sigma\left(b^i_c + \sum_{x',y',c'} w^i_{cc'}[x', y'] S^{i-1}_{c'}[sx + x', sy + y']\right) \tag{4.1}$$

Where $i = 1, \ldots, n_S$ indexes the layers, c and c' index the layers' channels, w^i is an array of weights for the convolutions, b^i is a vector of biases, and the summation is over $-1 \leq x', y' \leq 1$ (i.e., the convolution kernels have 3×3 spatial extent). We use the ReLU activation function $\sigma(\cdot) = \max(\cdot, 0)$ and use zero-padding as the boundary condition in all convolutions.

These low-level layers progressively reduce the spatial dimensions by a total factor of 2^{n_S}. Thus n_S has two effects: 1) it drives the spatial downsampling between the low-resolution input $\tilde{\mathrm{I}}$ and the final grid of affine coefficients—the higher n_S, the coarser the final grid, and 2) n_S controls the complexity of the prediction: deeper layers have an exponentially larger spatial support and more complex non-linearities (by composition); thus, they can extract more complex patterns in the input. Figure 4-3 shows a comparison with a network in which the low-level layers have been removed, and replaced by a hard-coded splatting operation [Chen et al., 2007]. Without these layers, the network loses much of its expressive power. Our architecture, uses $n_S = 4$ low-level layers. Table 4.1 summarizes the dimensions of each layer.

Local features path

The last low-level features layer S^{n_S} is then processed by a stack of $n_L = 2$ convolutional layers L^i in the local path (Figure 4-2, yellow). These layers take the same form as

Figure 4-3: Our *low-level* convolutional layers are fully learned and can extract semantic information. Replacing these layers with the standard bilateral grid splatting operation causes the network to lose much of its expressive power. In this example of our *Face brightening* operator (a-b), the network with hardcoded splatting (d) cannot detect the face properly because the grid's resolution is too low. Instead, it slightly brightens all skintones, as is visible on the hands. Our progressive downsampling with strided convolutions learns the semantic features required to solve this task properly (c), brightening only the face while darkening the background like in the reference.

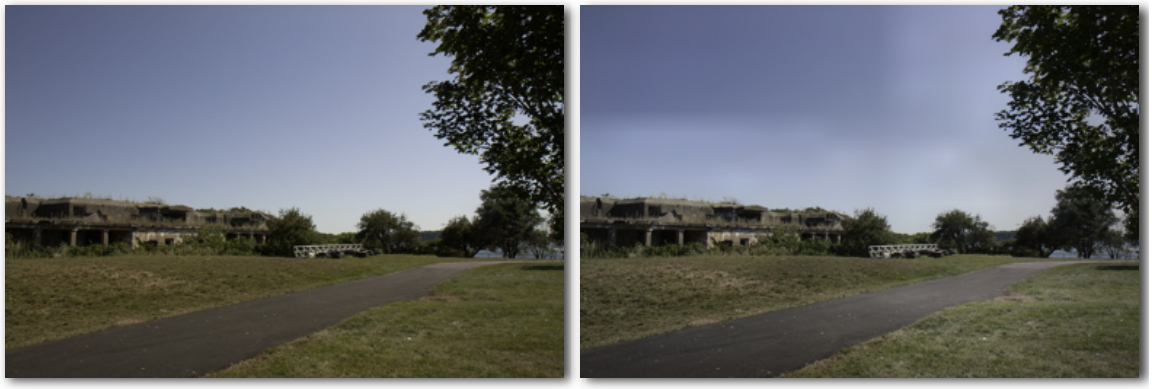
Equation (4.1), identifying $L^0 := S^{n_S}$, but this time with stride $s = 1$. We keep both the spatial resolution and number of features constant in the local path. Because the resolution is held constant, the spatial support of the filters only grows linearly with n_L. A deep enough stack of convolution layers, roughly measured by $n_S + n_L$, is critical to capturing useful semantic features [Krizhevsky et al., 2012]. If a higher spatial resolution is desired for the final grid of coefficients, one can reduce n_S and increase n_L to compensate accordingly, so as not to reduce the expressiveness of the network. Without the local path, the predicted coefficients would lose any notion of spatial location.

Global features path

Like the local path, the global features path branches out from S^{n_S}, that is $G^0 := S^{n_S}$. It comprises two strided convolutional layers (Equation (4.1), with $s = 2$) followed by three fully-connected layers, for a total of $n_G = 5$ global layers (Figure 4-2, blue).

One consequence of using fully-connected layers is that the resolution of the input $\tilde{\mathrm{I}}$ needs to be fixed, since it dictates the dimensions of G^2 and the number of network parameters that act on it. As we will see in Section 4.1.3, thanks to our slicing operator, we can still process images of any resolution, despite the size of the *low-res* stream being fixed.

The global path produces a 64-dimensional vector that summarizes global information about the input and acts as a prior to regularize the local decisions made by the local path. Without global features to encode this high-level description of the input, the network can make erroneous local decisions that lead to artifacts as exemplified by the large-scale variations in the sky in Figure 4-4.



(a) **ours**, with global features (b) without global features

Figure 4-4: The global features path in our architecture allows our model to reason about the full image, e.g., for subjective tasks such as reproducing subjective human adjustments that may be informed by intensity distribution or scene type (a). Without the global path, the model can make local decisions that are spatially inconsistent (b). Here, the network fails to recognize that the blue area in the top-left corner also belongs to the sky and should therefore receive the same correction as the area just below it.

Fusion and linear prediction

We fuse the contributions of the local and global paths with a pointwise affine mixing followed by a ReLU activation:

$$F_c[x,y] = \sigma\left(b_c + \sum_{c'} {w'}_{cc'} G_{c'}^{n_G} + \sum_{c'} w_{cc'} L_{c'}^{n_L}[x,y]\right) \tag{4.2}$$

This yields a $16 \times 16 \times 64$ array of features from which, we make our final 1×1 linear prediction to produce a 16×16 map with 96 channels:

$$A_c[x,y] = b_c + \sum_{c'} F_{c'}[x,y] w_{cc'} \tag{4.3}$$

Table 4.1: Details of the network architecture. c, fc, f and l refer to convolutional, fully-connected, fusion and pointwise linear layers respectively.

	S^1	S^2	S^3	S^4	L^1	L^2	G^1	G^2	G^3	G^4	G^5	F	A
type	c	c	c	c	c	c	c	c	fc	fc	fc	f	l
size	128	64	32	16	16	16	8	4	–	–	–	16	16
channels	8	16	32	64	64	64	64	64	256	128	64	64	96

4.1.2 Image features as a bilateral grid

So far we have described our model as a neural network. We now shift our perspective to that of a bilateral grid. To facilitate this, in a slight abuse of notation, we will occasionally treat the final feature map A as a multi-channel bilateral grid whose third dimension has been unrolled:

$$A_{dc+z}[x, y] \leftrightarrow A_c[x, y, z] \tag{4.4}$$

where $d = 8$ is the depth of the grid. Under this interpretation, A can be viewed as a $16 \times 16 \times 8$ bilateral grid, where each grid cell contains 12 numbers, one for each coefficient of a 3×4 affine color transformation matrix. This reshaping operation lets us interpret the strided convolutions in Equation (4.1) as acting in the bilateral domain, where they correspond to a convolution in the (x, y) dimensions and express full connectivity in the z and c dimensions. This operation is therefore more expressive than simply applying 3D convolutions in the grid, which would only induce local connectivity on z [Jampani et al., 2016]. It is also more expressive than standard bilateral grid splatting which discretizes I into several intensity bins then box filters the result [Chen et al., 2007]; an operation that is easily expressed with a 2-layer network. In a sense, by maintaining a 2D convolution formulation throughout and only interpreting the last layer as a bilateral grid, we let the network decide when the 2D to 3D transition is optimal.

4.1.3 Upsampling with a trainable slicing layer

So far we have described how we learn to predict a bilateral grid of coefficients A from a low-resolution image $\tilde{\mathrm{I}}$ using the *low-res* stream of our network. We now need to transfer this information back to the high-resolution space of the original input I to produce our final output image. To this end, we introduce a layer based on the bilateral grid *slicing* operation [Chen et al., 2007]. This layer takes as input a single-channel guidance map g and a feature map A (viewed as a bilateral grid) with a much lower spatial resolution than g. It performs a data-dependent lookup in the final feature map A. The layer is sub-differentiable with respect to both A and g. This allows us to backpropagate through it at train time.

The result of the slicing operator is a new feature map $\bar{A}$ with the same spatial resolution as g, obtained by tri-linearly interpolating the coefficients of A at locations defined by g:

$$\bar{A}_c[x, y] = \sum_{i,j,k} \tau(s_x x - i)\tau(s_y y - j)\tau(d \cdot g[x, y] - k)A_c[i, j, k] \tag{4.5}$$

Using a linear interpolation kernel $\tau(\cdot) = \max(1 - |\cdot|, 0)$, and where s_x and s_y are the width and height ratios of the grid's dimensions w.r.t. the full-resolution image's dimensions. Essentially, each pixel is assigned the vector of coefficients whose depth in the grid is given by the gray scale value $g[x, y]$, i.e., loosely speaking $A_c[i, j, g[x, y]]$. Flownet2 [Ilg et al., 2016] and Spatial Transformer Networks [Jaderberg et al., 2015] have used similar interpolation operators for in-network spatial warping. We fix the spatial resolution of the grid to 16×16, and its depth to $d = 8$.

The slicing operation is parameter-free and can be implemented efficiently in an OpenGL shader [Chen et al., 2007]. It acts as a bottleneck layer that constrains the representation of the neural network to a low-dimensional space. This both simplifies the learning problem and speeds up the processing time [Barron et al., 2015; Barron and Poole, 2016]. Crucially, performing inference within a bilateral grid forces our model's

predictions to follow the edges in g, thereby regularizing our predictions towards *edge-aware* solutions (unlike standard networks based on transpose-convolutions or "deconvolution layers", Figure 4-5). This design decision tends to benefit photographic manipulation tasks such as ours and enables our significant speedup over more general models due to the low dimensionality of A (Figure 4-10).

This data-dependent lookup is critical to the expressive power of our model. As we will see in Section 4.1.4, it allows us to predict a complex operation on the full-resolution image using a collection of much simpler local models.

4.1.4 Assembling the full-resolution output

So far, we have described how to obtain and upsample the bilateral grid of affine coefficients. The rest of the processing is done at full resolution. It should therefore be simple and easily-parallelizable to minimize computational cost. From the full-resolution input I, we extract a set of n_ϕ full-resolution features ϕ that fulfill two roles: 1) they are combined to predict the guidance map g used in the slicing node, and 2) they are used as regression variables for the local affine models.

The most cost-efficient approach is to use the channels of the input image as features, that is $\phi = \mathrm{I}$ (with $n_\phi = 3$) and the local affine models are color transformations. All our results use this fast formulation.

Guidance map auxiliary network

We define g as a simple pointwise nonlinear transformation of the full-resolution features:

$$g[x, y] = b + \sum_{c=0}^{2} \rho_c\big(\boldsymbol{M}_c^\top \cdot \phi_c[x, y] + b'_c\big) \tag{4.6}$$

Where $\boldsymbol{M}_c^\top$ are the rows of a 3×3 color transformation matrix, b and b'_c are scalar biases, and ρ_c are piecewise linear transfer functions parametrized as a sum of 16

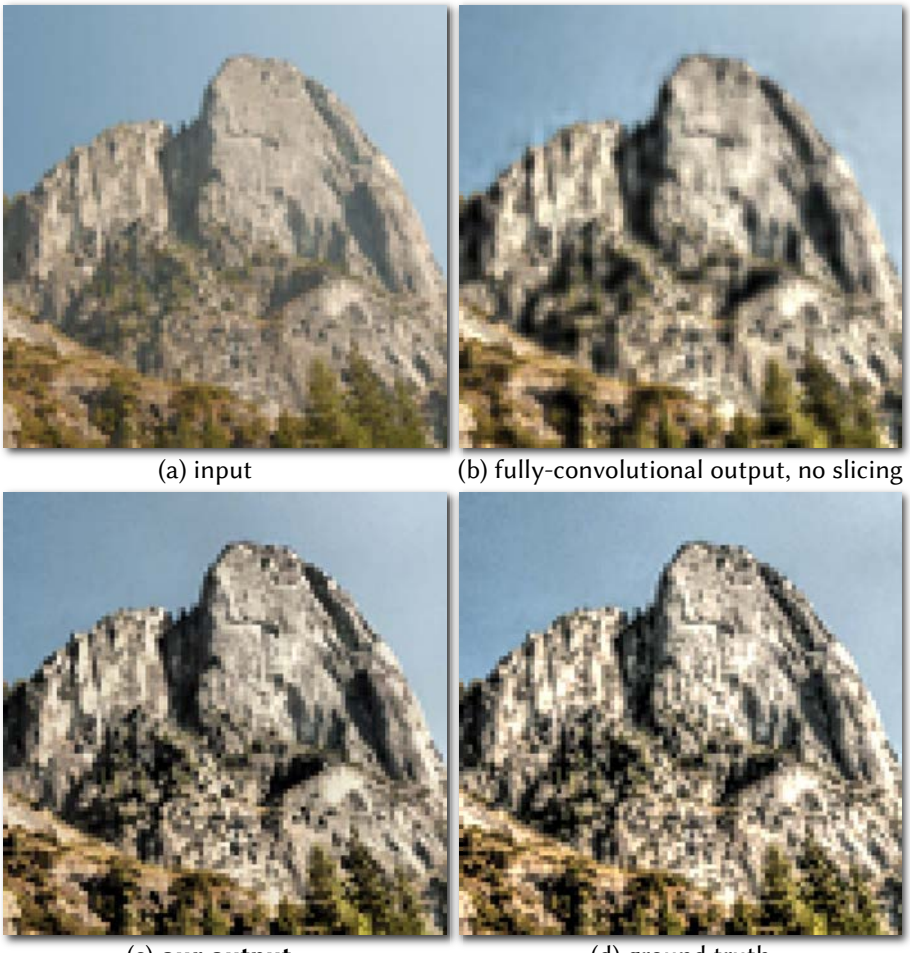


Figure 4-5: Our new slicing node is central to the expressiveness of our architecture and its handling of high-resolution effects. Replacing this node with a standard bank of learnable deconvolution filters reduces expressiveness (b) because no full-resolution data is used to predict the output pixels. Thanks to its learned full-resolution guidance map, our slicing layer approximates the desired enhancement with much higher fidelity (c), thereby preserving the edges of the input (a) and capturing the high-frequency transformations visible in the ground-truth output (d).

scaled ReLU functions with thresholds $t_{c,i}$ and slopes $a_{c,i}$:

$$\rho_c(x) = \sum_{i=0}^{15} a_{c,i} \max(x - t_{c,i}, 0) \tag{4.7}$$

The parameters $\boldsymbol{M}$, a, t, b, b' are learned jointly with the other network parameters. $\boldsymbol{M}$ is initialized to the identity and a, t, b, and b' are initialized such each ρ_c is an identity mapping over $[0, 1]$, which is necessary to avoid learning a degenerate g. Figure 4-7 shows the impact of using this learned guide and Figure 4-6 shows an example of the color transformation matrix and tone curve that are learned for the corresponding task.

Assembling the final output

Although image operators may be complex when viewed at the scale of an entire image, recent work has observed that even complicated image processing pipelines can often be accurately modeled as a collection of simple local transformations [He and Sun, 2015; Gharbi et al., 2015; Chen et al., 2016]. We therefore model each channel of our final output O_c as an affine combination of the full-resolution features, with

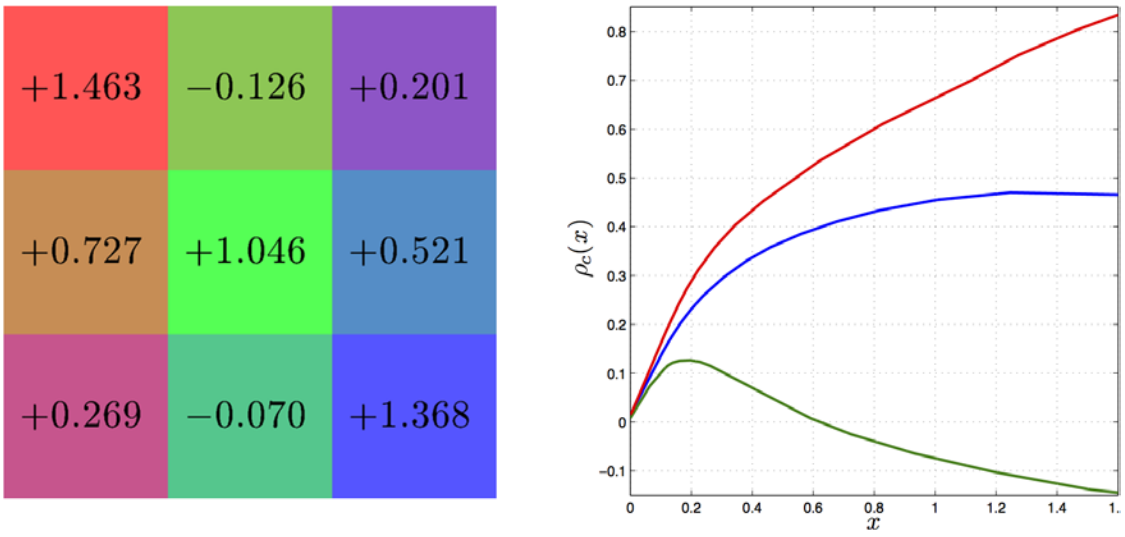


Figure 4-6: The color transform matrix (left) and per-channel tone curves (right) used to produce the guidance map g, as learned by one instance of our model.

coefficients defined by the channels of the sliced feature map $\bar{A}$:

$$\mathrm{O}_c[x,y] = \bar{A}_{n_\phi+(n_\phi+1)c} + \sum_{c'=0}^{n_\phi-1} \bar{A}_{c'+(n_\phi+1)c}[x,y]\,\phi_{c'}[x,y] \tag{4.8}$$

Interpolated affine transformations similar to this have been used successfully for matting [Levin et al., 2008], intrinsic image decomposition [Bousseau et al., 2009] and time of day transfer [Shih et al., 2013b]. For such models, the size of the patch in which the affine model is fit drives the trade-off between efficiency and quality. At the extreme, it is always possible to achieve a perfect reconstruction of any operator by fitting an independent model at every pixel (i.e., the patch size is 1×1). For small patches (e.g., 3×3), an affine model can faithfully reproduce many image operators. As the patch grows larger, the affine relationship no longer holds for all but trivial operators, though others have shown that this limitation can be mitigated using piecewise linear functions [Yuan and Sun, 2011] or non-linear and edge-aware components [Gharbi et al., 2015]. See Figure 4-8 for a visualization of the 3D bilateral grid of affine coefficients A corresponding to the input/output pair in Figure 4-2. One of the 12 channels of the 2D coefficients after slicing can also be seen in Figure 4-2.

4.1.5 Training procedure

We train our network on a dataset $\mathcal{D} = \{(\mathrm{I}_i, \mathrm{O}_i)\}_i$ of full-resolution input/output pairs for a given operator. We optimize the weights and biases by minimizing the L_2 loss on this training set:

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_i \|\mathrm{I}_i - \mathrm{O}_i\|^2 \tag{4.9}$$

We additionally regularize the weights with an L_2 weight decay of 10^{-8}. The weights for the convolutional and fully-connected layers are initialized according to [He et al., 2015] and the biases are initialized to 0. We use batch normalization [Ioffe and Szegedy, 2015] between each pair of intermediate feature maps, and we optimize

the network parameters with the ADAM solver [Kingma and Ba, 2015]. We train with a batch size of 4 to 16 (depending on the resolution) and a learning rate of 10^{-4}. The remaining parameters in ADAM are kept to the values recommended by the authors. Our model is implemented in Tensorflow [Abadi et al., 2015] and Halide [Ragan-Kelley et al., 2012a]. For all experiments, models are trained on an NVIDIA Titan X (Maxwell) for 30 epochs, which typically takes 2–3 days.

4.2 Results

We evaluate our model's ability to reproduce both algorithmic image operators (Section 4.2.1) and human-annotated retouches (Section 4.2.2). Our model is faster than both standard neural networks and state-of-the-art filter approximation techniques and runs in real-time on mobile device (Section 4.2.3).

A selection of our results on different tasks can be seen in Figure 4-14. Our output is generally accurate and, even when it differs from the ground-truth, it remains plausible. Despite the heavy spatial and bilateral downsampling inherent to our approach, image artifacts are rare and unobjectionable. This is because of the edge-aware nature of the bilateral grid and our model's capacity to learn smooth output transformations. Our outputs are usually slightly softer (e.g. on the HDR+ example of Figure 4-14) because the highest-frequency transformations like sharpening and the correction of chroma aberrations can introduce new edges not present in the input, which our model does not handle

4.2.1 Reproducing image operators

We evaluate the accuracy of our model on several tasks composed of programmatically-defined image operators:

. *HDR+* [Hasinoff et al., 2016] – a complex hand-engineered photographic pipeline that includes color correction, auto-exposure, dehazing, and tone-mapping.

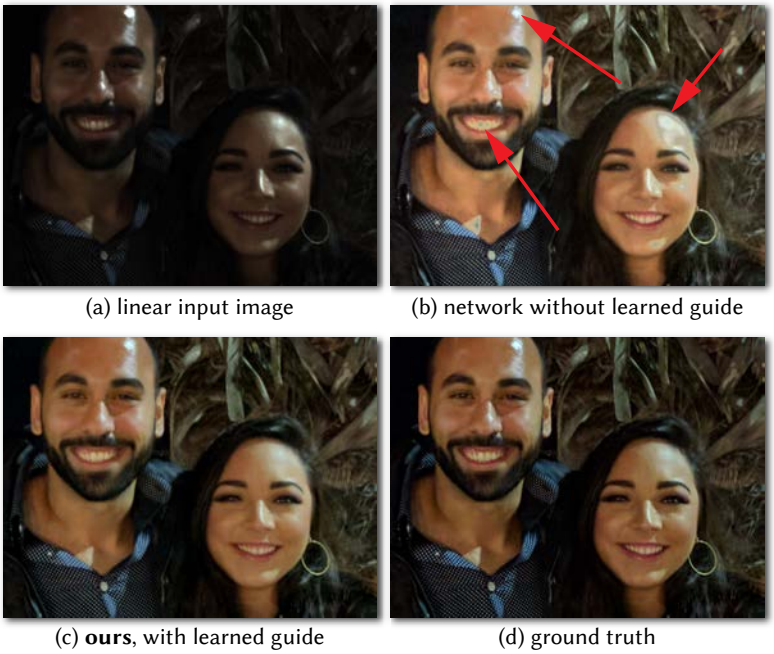


Figure 4-7: Our slicing node uses a learned guidance map. Using luminance as guide causes artifacts with the HDR+ pipeline reproduction, in particular with posterization artifacts in the highlights on the forehead and cheeks (b). In contrast, our learned guide (c) correctly reproduces the ground truth (d).

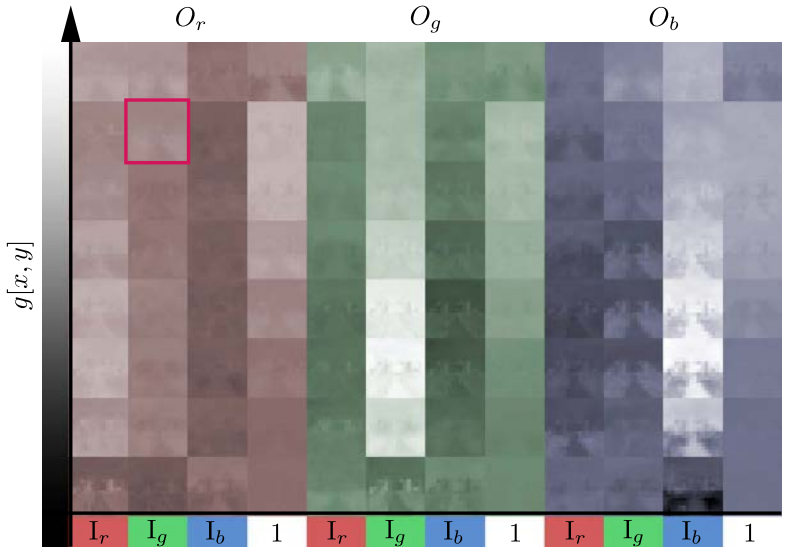


Figure 4-8: Coefficient maps for the affine color transform. The vertical axis corresponds to the learned guidance channel, while the horizontal axis unrolls the 3x4 sets of coefficients. Each thumbnail, one example of which is highlighted, shows a 16x16 low-resolution map.

. the *Local Laplacian* filter [Paris et al., 2011b] – an edge-preserving, multi-scale (yet non-scale-invariant) operator used for detail enhancement (we use two different strengths for the effect),

. the *Style Transfer* task of [Aubry et al., 2014b] (which happens to be based on the Local Laplacian),

. a *Face brightening* task using a dataset of labeled faces [Jain and Learned-Miller, 2010],

. several different black-box *Adobe Photoshop (PS)* filters and user-created "actions"[1].

PSNRs for these tasks using our model and baseline approaches can be found in Table 4.2.

We use two variants of the style transfer task. In the first variant (*Style Transfer*), we learn to transform any new input towards a unique fixed style. In the second, more challenging variant (*n-Styles Transfer*) we adapt our network to take two input images (concatenated along their channel axis) and predict the results of transferring the style of one image to the other (again using the algorithm of Aubry et al. [2014b]). In this variant the network does not learn to predict a single consistent output; but rather, it learns to extract the desired transformation from the target image and apply that transformation to the input image.

Datasets

Besides HDR+ and the face brightening dataset, all the effects were applied to the *unprocessed* set of the MIT "FiveK" dataset [Bychkovsky et al., 2011b]. We reserve 500 images for validation and testing, and train on the remaining 4500. We augment the data with random crops, flips and rotations. We generated the dataset for *n-Styles Transfer* by mapping each image in the MIT "FiveK" dataset to 100 distinct images (the style targets).

[1] `http://designbump.com/photoshop-actions-for-instagram-effects/`

Baseline

The previous work closest in spirit to our goals are Bilateral Guided Upsampling (BGU) [Chen et al., 2016] and Transform Recipes (TR, chapter 3) [Gharbi et al., 2015] to which we compare our outputs. However, whereas our technique learns a photographic operator offline from a dataset of images, BGU and TR use no prior training and instead fit specially-tailored models to an input/output pair in an online fashion. BGU and TR therefore require direct access to the image operator, as they require the ability to run that image operator on images (either downsampled on-device or full-resolution on a server, respectively). This makes our comparisons against these baselines somewhat biased against our technique, as these baselines make more limiting assumptions about what is available, and also cannot learn to approximate a general instance of an image operator from data. Regardless, we report metrics for these techniques as a kind of "oracle" baseline.

As we have seen in chapter 3, Transform Recipes assumes that a mobile device

Table 4.2: We compare accuracy to Bilateral Guided Upsampling (BGU) and Transform Recipes (TR). Note that BGU and TR are "oracle" techniques, as they run the code used to evaluate each image operator at a reduced or full resolution, and so can be thought of as providing an upper-bound on performance. Despite its disadvantage, our model sometimes performs better than these oracle baselines due its expressive power and ability to model non-scale-invariant operators.

Task (PSNR, dB)	Ours	BGU	TR
HDR+	28.8	26.9	29.0
Local Laplacian	33.5	32.2	38.6
Local Laplacian (strong)	30.3	20.6	31.8
Face brightening	33.7	30.9	33.9
Style Transfer	23.9	21.9	31.7
n-Styles Transfer	27.6	21.9	33.7
PS eboye	45.0	33.5	41.5
PS early bird	25.9	22.2	32.8
PS instagram	40.3	37.1	40.7
PS infrared	38.4	34.5	38.7
PS false colors	38.1	34.3	38.6
PS lomo-fi	26.2	24.1	34.4

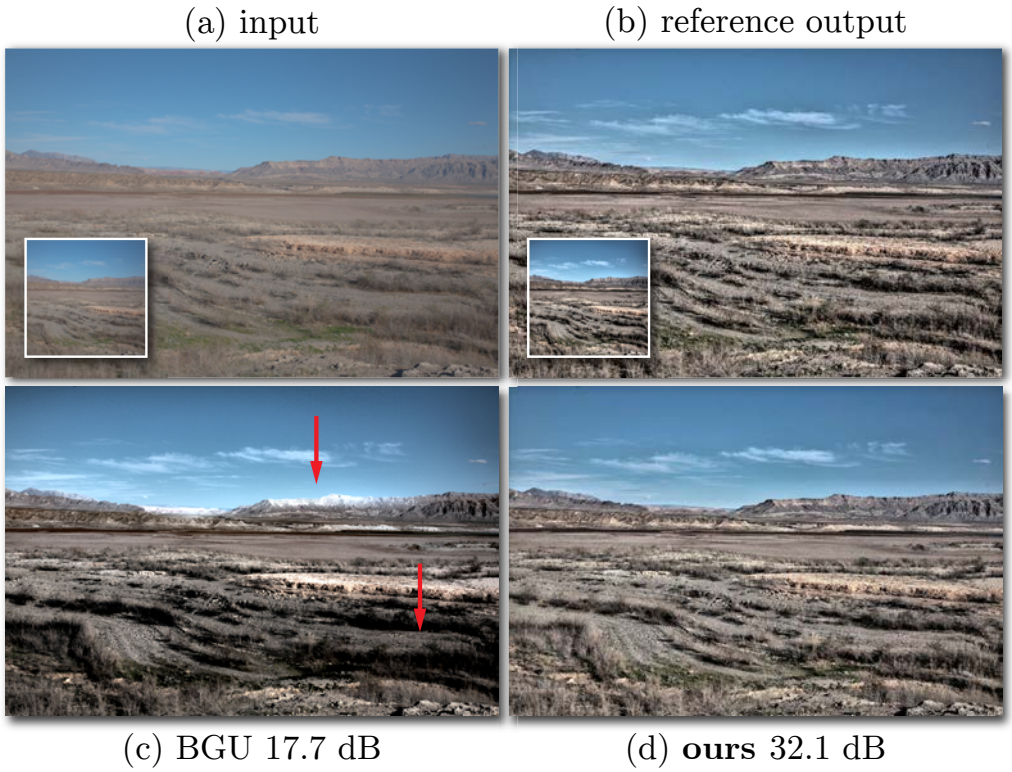


Figure 4-9: Our method (d) can learn to replicate the correct effect (b) for operations that are not scale invariant, such as the Local Laplacian filter shown here (a–b). Methods like Bilateral Guided Upsampling that only apply the operation at low-resolution (insets (a–b)) produce a different-looking output (c). The difference is most noticeable in the areas pointed by the arrows.

would send a highly compressed (and therefore degraded) image to a server for processing, and would receive an inexpensive "recipe" for approximating an image transformation from that server. For this evaluation, we ignore TR's client-server setup and run the model at the recommended settings described in subsection 3.3.2 on uncompressed, full-resolution images. This ensures the quality of TR is optimal, making the baseline as competitive as possible. In the intended use case of the method, the image quality typically decreases by 3–5 dB.

BGU assumes that the image operator be run on a low-resolution version of the input before fitting the model to the low-res input/output pair. We could not run the HDR+ filter at low resolution, so we used full-resolution input/output pairs and

Table 4.3: Mean L_2 error in La*b* space for retouches from the 5 photographers in the MIT5k dataset (A,B,C,D,E); lower is better. Our algorithm is capable of learning a photographer's retouching style better than previous work, yet runs orders of magnitudes faster. The comparisons in the first two groups are evaluated on the dataset from photographer C favored by previous techniques; see main text for details. In the third group we report our results on the remaining 4 photographers for completeness. Metrics taken from previous work Yan et al. [2016]; Hwang et al. [2012] are denoted by †.

photographer	method	La*b*	L-only
C *random250*	ours	**7.8**	**5.5**
	Yan [2016]	9.9†	5.7†
	Bychkovsky [2011b]	–	5.8†
	Hwang [2012]	15.01†	–
C *highvar50*	ours	**7.1**	**5.2**
	Yan [2016]	9.9†	8.4†
	Bychkovsky [2011b]	–	–
	Hwang [2012]	12.03†	–
A	ours	11.7	9.8
B	ours	7.4	5.0
D	ours	10.0	7.7
E	ours	8.8	6.2

created the low-resolution inputs to BGU by downsampling. We do however follow the correct procedure for the *Local Laplacian* and *Style Transfer* tasks for which we have an implementation and directly apply the filter at low resolution. For these non scale-invariant tasks, the advantage of our technique becomes clearer (Figure 4-9).

4.2.2 Learning from human annotations

We also evaluate accuracy with regards to human annotations using the MIT-Adobe "FiveK" dataset [Bychkovsky et al., 2011b], and our performance compared to previous work is presented in Table 4.3. This task measures our model's ability to learn a highly subjective image operator which requires a significant amount of learning and semantic reasoning. We report mean L_2 error in La*b* space (lower is better) for retouches by the 5 photographers (A,B,C,D,E) in the MIT "FiveK" dataset, though

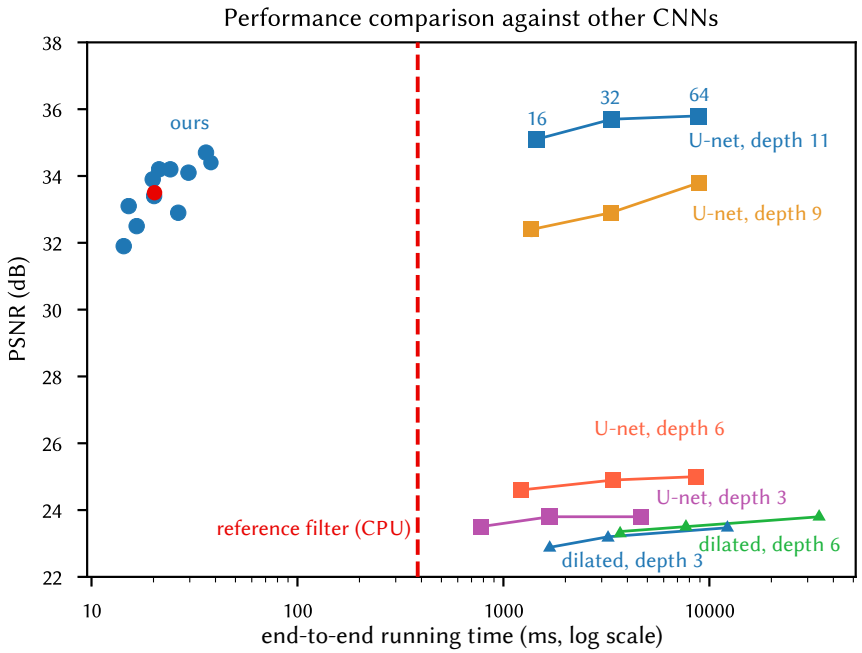


Figure 4-10: We compare the speed and quality of our algorithm against two modern network architectures: *U-Net* (adapted from Isola et al. [2016]) and *dilated convolutions* Yu and Koltun [2015]. The runtimes were averaged over 20 iterations, processing a 4 megapixel image on a desktop CPU. The PSNR numbers refer to the *Local Laplacian* task. Given an insufficient *depth*, U-Net and dilated convolutions fail to capture the large scale effects of the Local Laplacian filter, leading to low PSNRs. Competitive architectures run over 100 times slower than ours, and use orders of magnitude more memory. Our model's performance is displayed for a range of parameters. The version we used to produce all the results is highlighted in red. See Figure 4-11 for details on the speed/quality trade-off of our model.

previous work only presents results on photographer C [Yan et al., 2016; Hwang et al., 2012]. We use the "Random 250" and "High Variance 50" dataset splits presented in [Hwang et al., 2012], which have 250 randomly-chosen and 50 user-weighted images in the test set, respectively.

This is a much more difficult task, and inconsistencies in the retouches of photographers has been pointed out previously [Yan et al., 2016]. For example we found that retoucher B in this dataset was more self-consistent, and was easier for our network to learn. Nonetheless, our model, trained separately on each artist's corrections, consistently predicts reasonable adjustments and outperforms previous work.

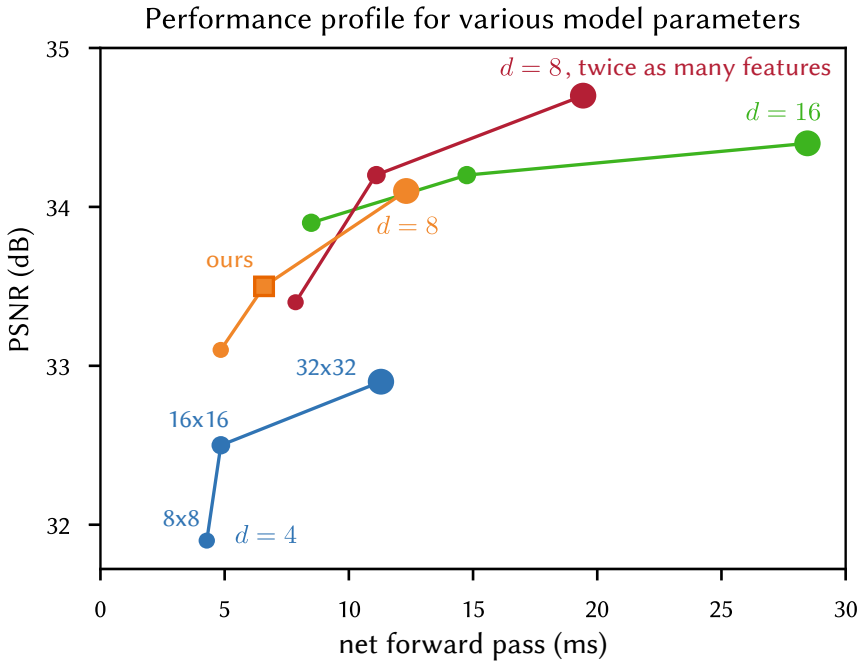


Figure 4-11: We show PSNRs for the *Local Laplacian* task and the computation time required to predict the bilateral coefficients with several settings of our model's parameters. Each curve represent a grid depth d. For each curve the grid's spatial resolution varies in $\{8, 16, 32\}$. The reference model we used to produced all the results is highlighted with a square marker. Unsurprisingly, models with larger grid depth perform better (green). Doubling the number of intermediate features also provides a 0.5 dB improvement (red curve). Runtimes were measured on an Intel Core i7-5930K.

4.2.3 Performance

We implemented our technique on a Google Pixel phone running Android 7.1.1. Our implementation processes viewfinder-resolution 1920×1080 images in realtime, at 40–50 Hz. We extract 8-bit preview frames in YUV420 format using the *Camera2* API. These images are downsampled to 256×256, converted to floating point RGB, then fed into our network. After the network produces its output (a bilateral grid of affine coefficients), we transfer them to the GPU as a set of three 3D RGBA textures, where they are sliced and applied to the full-resolution input to render the final processed preview. Overall throughput is under 20 ms, with 14 ms spent on inference (CPU), overlapped with 1 ms to upload coefficients and 18 ms to render on the GPU. As a

point of comparison, running an optimized implementation [Ragan-Kelley et al., 2012a] of the Local Laplacian filter [Paris et al., 2011b] on the same device takes over 200 ms. Running the same filter at the reduced 256×256 resolution and applying Bilateral Guided Upsampling [Chen et al., 2016] with the same grid dimensions takes 17 ms (compared to our 14 ms) but loses some of the filter's intended effect (Figure 4-9). Our processing time scales linearly with input size, taking 61 ms to process a 12-megapixel image. While it usually has higher fidelity, Transform Recipes [Gharbi et al., 2015] requires 2.95 seconds per image, nearly two orders of magnitude below real-time viewfinder performance. Most notably, neither Transform Recipes nor Bilateral Guided Upsampling can apply effects learned from human retouches, or "black box" operators such as Photoshop filters or HDR+.

Other recent neural-network based architectures that could be used for such learning are also far from real-time. In Figure 4-10, we compare our technique against a U-Net architecture [Ronneberger et al., 2015] adapted from Isola et al. [2016], and a linear network based on dilated convolutions [Yu and Koltun, 2015]. We explore several settings for the *depth* (number of layers, 3 to 11) and the *width* (number of filters, 16 to 64) in these architectures, covering a variety of speed and quality levels. For U-Net, "depth" refers to the number of downsampling steps and "width" refers to the channels in the first convolutional layers (these are doubled at each downsampling step, see Isola et al. [Isola et al., 2016] for details). In the dilated convolution network, "depth" is the number of dilated convolution layers, and "width", the number of channels in each layer. Our hybrid CPU/OpenGL technique is over 2 orders of magnitude faster than both architectures on a desktop CPU. On GPU (not shown), the performance gap is identical for the forward pass of the network, but data transfer becomes the bottleneck for our method. End-to-end, our runtime is still over an order of magnitude faster. Moreover, both U-Net and dilated convolution require significantly more memory, which makes them ill-suited for mobile processing. For this benchmark we used an Intel Core i7-5930K at 3.5GHz with 4 cores and a Titan

X (Maxwell) GPU.

We explored the speed/quality trade-offs of our architecture for the *Local Laplacian* task varying several parameters: changing the depth of the grid d from 4 to 16, the grid's spatial dimensions from 8×8 to 32×32 and doubling the number of channels (compared to the numbers reported in Table 4.1). The summary can be found in Figure 4-11.

4.3 Discussion and limitations

All our results use the simplest full-resolution features $\phi = \mathrm{I}$; i.e., both the guide g and the affine regression targets are the color channels of the input image (Section 4.1.4). If one relaxes the real-time rendering constraint, one can extend our model by extracting features from the high-resolution image. In Figure 4-12, we show an example where ϕ is a 3-level Gaussian pyramid. The bilateral grid then contains $3 \times 12 = 36$ affine parameters (12 for each scale). Accordingly we triple the number of intermediate features in the network compared to the numbers in Table 4.1. This roughly slows down the network by a factor 3-4, but provides a 2 dB boost in quality on the *Local Laplacian (strong)* task.

We also explored using our architecture to learn tasks beyond image enhancement, like matting, colorization, dehazing, and monocular depth prediction. These experiments had limited success, as the strong modeling assumptions required for fast photographic correction make our model poorly suited to different tasks whose output cannot be easily expressed as local pointwise transformations of the input image (Figure 4-13).

Figure 4-12: At the expense of extra computation at full-resolution, our model can be extended with richer affine regression features. Here, by using a 3-level Gaussian pyramid as features ϕ, we can better capture the high-frequency details in the the *Local Laplacian (strong)* task.

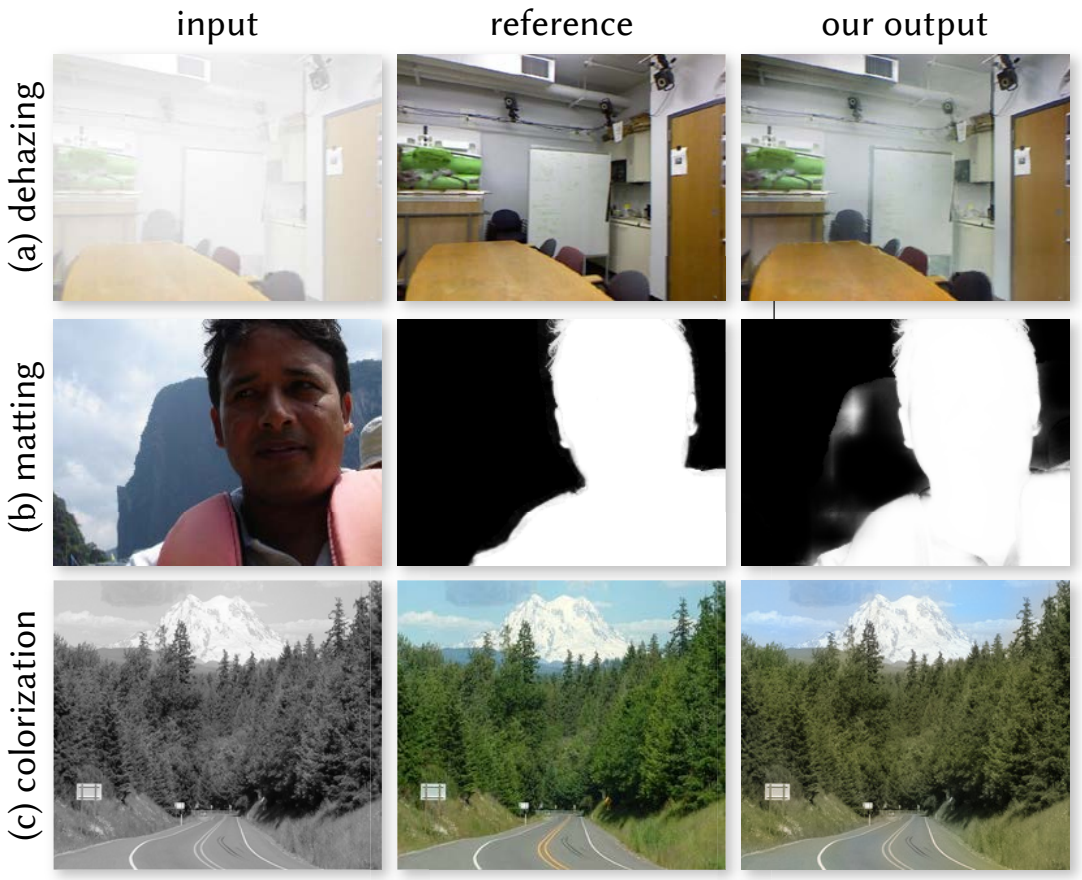


Figure 4-13: Our algorithm fails when the image operator strongly violates our modeling assumptions. (a) Haze reduces local contrast, which limits the usefulness of our guidance map. It also destroys image details that cannot be recovered with our affine model (e.g., on the whiteboard). (b) Matting has successfully been modeled by locally affine models on 3×3 neighborhoods Levin et al. [2008]. However, this affine relationship breaks down at larger scales (like a grid cell in our model) where the matte no longer follows tonal or color variations and is mostly binary. This limits the usefulness of our bilateral grid. (c) For colorization, the learned guidance map is at best a nonlinear remapping of the grayscale input. Our model can thus only learn a local color per discrete intensity level, at a spatial resolution dictated by the grid's resolution. Our output is plagued with coarse variations of colors that are muted due to our L_2 loss (see the road line, and the tree/sky boundary).

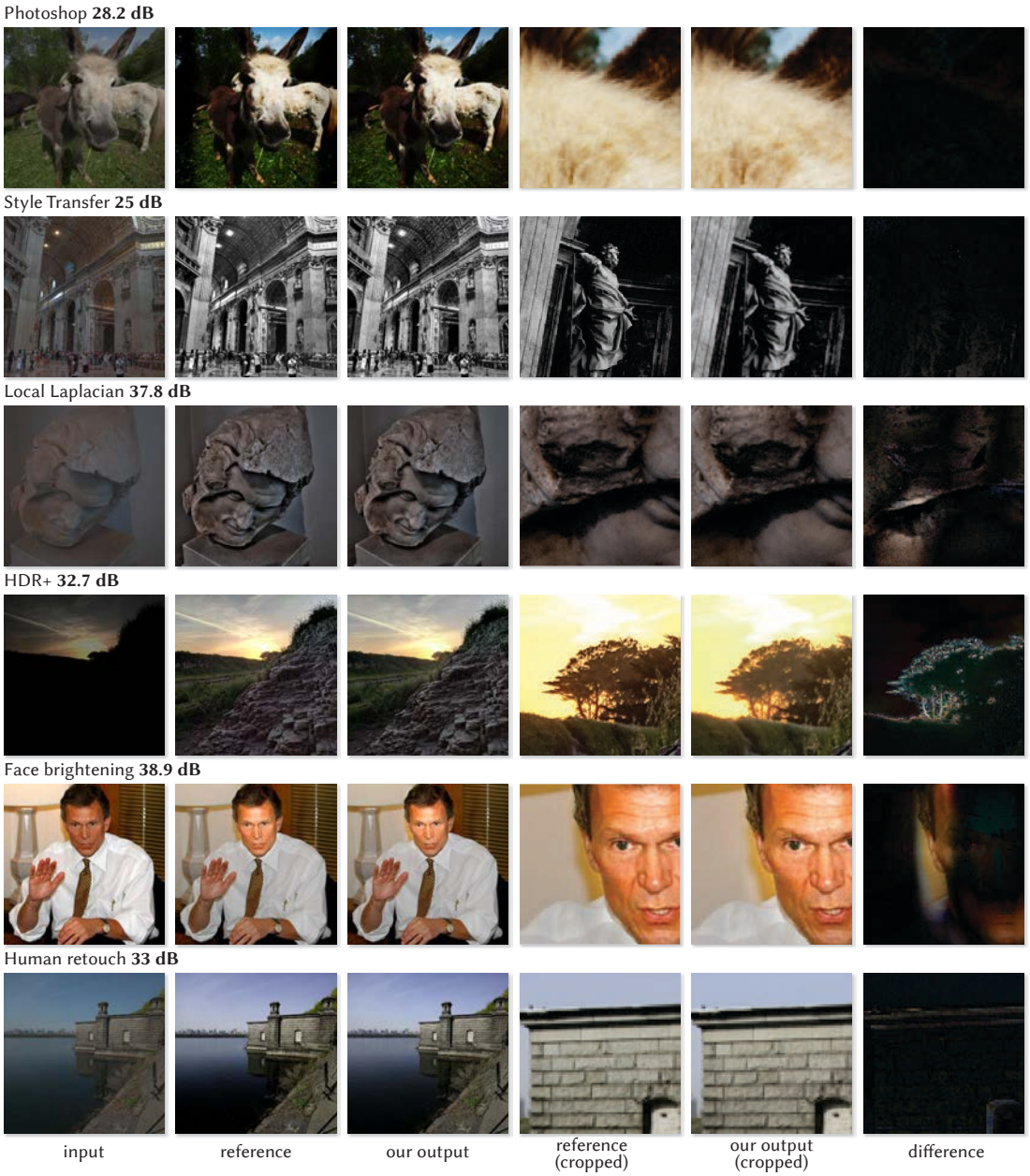


Figure 4-14: Our method can learn accurate and fast approximations of a wide variety of image operators, by training on input/output pairs processed by that operator. These operators can be complicated "black box" image processing pipelines where only a binary is available, such as HDR+ or Photoshop filters/actions. Some operators, such as face-brightening, requires semantic understanding. Our model is even capable of learning from highly subjective human-annotated input/output pairs, using the MIT-Adobe FiveK dataset.

Chapter 5

Joint Demosaicking and Denoising

In chapters 3 and 4, we presented general models to approximate a large class of image filters or even photographic edits from human retouchers. The assumptions made by these models are central to their efficient runtime. Unfortunately, some lower-level image processing operations require more precise control over the pixel transformations, which precludes the use of such approximations. For these operations, general neural networks can help. But as we will see throughout this chapter, an adequate trainable model provides only half of the solution. High quality training data is paramount if one wishes to improve the quality of core image processing tasks that have been optimized for decades. This data is difficult to obtain. In this chapter, we show how a carefully designed training corpus and a domain-specific model can drastically improve the quality of image demosaicking and denoising.

Demosaicking and denoising are simultaneously the crucial first steps of most digital camera pipelines. They are quintessentially ill-posed reconstruction problems: at least two-thirds of the data is missing and the existing data is corrupted with noise. Furthermore, complex aliasing issues arise because the red, green and blue channels are sampled at different locations and at different rates. While most image areas are easy to address, the rare challenging regions can still lead to catastrophic failure and visually disturbing artifacts such as checkerboard patterns, zippering around edges,

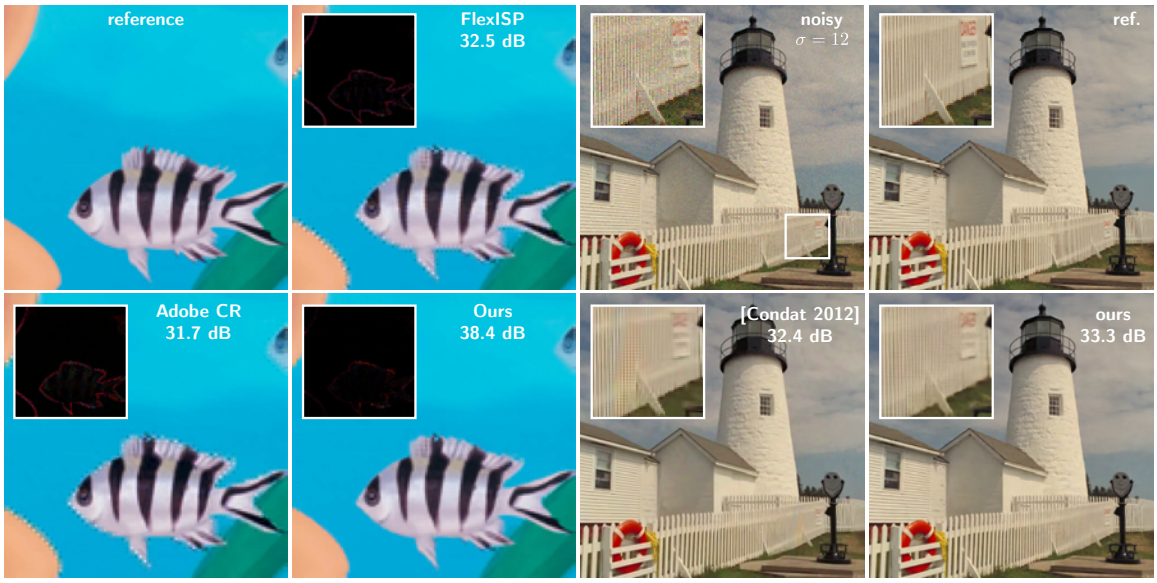


Figure 5-1: We propose a data-driven approach for jointly solving denoising and demosaicking. By carefully designing a dataset made of rare but challenging image features, we train a neural network that outperforms both the state-of-the-art and commercial solutions on demosaicking alone (group of images on the left, insets show error maps), and on joint denoising–demosaicking (on the right, insets show close-ups). The benefit of our method is most noticeable on difficult image structures that lead to *moiré* or *zippering* of the edges.

and moiré.

For modularity, demosaicking and denoising are often solved independently and sequentially. This unfortunately leads to error accumulation because demosaicking needs to cope with unreliable samples and denoising suffers from the non-linear and variable per-pixel noise introduced by demosaicking. It has long been recognized that exploiting the regularity of natural images is key to lifting underdetermination. Traditional techniques have hard-coded hand-crafted heuristics into local filters [Cok, 1987; Laroche and Prescott, 1994; Buades et al., 2009]. Heide et al. [2014] proposed a joint solution to denoising and demosaicking by embedding a non-local natural image prior into an optimization approach. However, their prior is still hand-crafted and the combination of optimization and a non-local prior leads to a steep increase in computation cost.

In contrast, we address demosaicking and denoising jointly using a data-driven local filtering approach for efficiency. We train our model on a large set of ground truth data to optimally leverage regularities found in natural images. We build on the success of deep learning and convolutional neural networks, e.g. [LeCun et al., 2015]. While data-driven local-filtering has been explored previously [Klatzer et al., 2016; Tian et al., 2014; Lansel and Wandell, 2011], assembling a quality training set is always key and we found that the characteristics of demosaicking and denoising make this a challenge, in particular because catastrophically hard inputs are rare and because salient artifacts are not well captured by standard image metrics. Another challenge is that deep learning often requires to train a new network or to fine-tune an existing one for even slightly different instances of a problem. This is particularly problematic for issues such as sensor noise, whose strength varies with the ISO setting, and other imaging characteristics.

Our contributions to joint denoising-demosaicking are a Convolutional Neural Network capable of handling a *wide range* of noise levels and a procedure to build a training set rich in challenging images prone to moiré and artifacts. We demonstrate that our approach enables higher-quality results than previous work and runs faster on both CPU and GPU.

5.1 Related work

Demosaicking is a well-studied problem and most algorithms perform well in flat regions of the image. But all tend to struggle around strong edges and textured areas (Figure 5-1). This leads to conspicuous artifacts such as *zippering*, color moiré and loss of detail. Many approaches derive edge-adaptive interpolation schemes to control such artifacts [Laroche and Prescott, 1994]. A popular solution is to design nonlinear filters that avoid interpolating across the strong local edges [Li et al., 2008]. The key ingredient for demosaicking is to leverage cross-channel dependencies to recover

details beyond the Nyquist frequency of each channel. Correlations between color channels can be captured by the *smooth hue* prior [Cok, 1987] where color ratios or differences are modeled as smoothly varying signals. Algorithms based on this heuristic interpolate channels sequentially starting with the luminance component i.e. green channel [Zhang et al., 2009; Chang and Tan, 2004]. The demosaicked green channel is then used to guide the chrominance interpolation. In these techniques, image quality is adversely affected when the smooth hue heuristic does not hold, leading to false color (Figure 5-1). Hirakawa and Parks [2005] use median filtering on color differences to mitigate the effect. But such post-processing techniques have drawbacks like excessive blurring, and do not fundamentally change the issue of color moiré. We propose to replace hand-crafted filters by a machinery that can jointly interpolate the three color channels, is fully trainable and can learn to disambiguate error-prone patterns directly from natural images without relying on hard-coded heuristics.

Self-similarity and data-driven demosaicking Recent methods overcome the ill-posedness of demosaicking by exploiting local self-similarity in natural images and fill in the missing color information from similar neighboring patches [Buades et al., 2009; Zhang et al., 2011]. He et al. [2012] use SVM regression to learn on-line a demosaicking process tailored to the input image. Another approach to the demosaicking problem is to employ machine-learning. Kwan and Xiaolin [2004] adopt a classification approach to select one of two discrete directions of interpolation with hand-designed features. Some techniques employ fully connected shallow neural network architectures with small spatial footprints [Go et al., 2000; Kapah and Hel-Or, 2000]. Early data-driven techniques used simple architectures and hard-coded heuristics. They were trained on small datasets of up to a few hundred images, and do not compare favorably with the state-of-the-art. This has been attributed to the lack of appropriate training datasets [Zhang et al., 2009]. Learning-based methods enable experimentation with new sensor designs and alternative mosaick patterns [Lansel and Wandell, 2011; Tian

et al., 2014]. In this work, we gather a dataset of millions of difficult patches from online photo collections according to the severity of the artifacts produced by a baseline demosaicking method. We train a model directly from the input mosaick to the final color image and achieve state-of-the art quality.

Joint denoising and demosaicking Demosaicking is further complicated by the presence of noise. Estimates of edge orientation in noisy data are less reliable which leads to noticeable artifacts in the demosaicked image. The techniques that perform these steps sequentially usually start with denoising [Park et al., 2009]. A notable exception, Akiyama et al. [2015] first denoise the Bayer array viewed as a four-channels quarter-resolution image. Recent attempts have shown the advantages of joint approaches [Hirakawa and Parks, 2006; Condat and Mosaddegh, 2012]. Jeon and Dubois [2013] optimize a set of filters for discrete noise levels. Heide et al. [2014] use a global primal-dual optimization with a self-similarity prior. The nearest neighbor search and the iterative nature of the algorithm makes it slow and somewhat impractical. Khashabi et al. [2014] demonstrate a learning approach that generalizes to non-Bayer mosaick patterns. Klatzer et al. [2016] use a sequential energy minimization approach which can be interpreted as a convolutional network with trainable activation functions and where intermediate layers are constrained to output a color image. Klatzer et al. can learn a noise model from data but this model is tailored to a single noise level and fixed after training. Instead, we expose a runtime parameter and train our network so it adapts to a wide range of noise levels.

5.2 Learning to jointly demosaick and denoise

Demosaicking and denoising have traditionally been addressed using nonlinear filter design, incorporating prior heuristics about inter- and intra-channel correlation, behavior around edges, and exploiting intra-image patch similarity. A convolutional network seems a natural choice for the problem in this context. First, it enables discovery

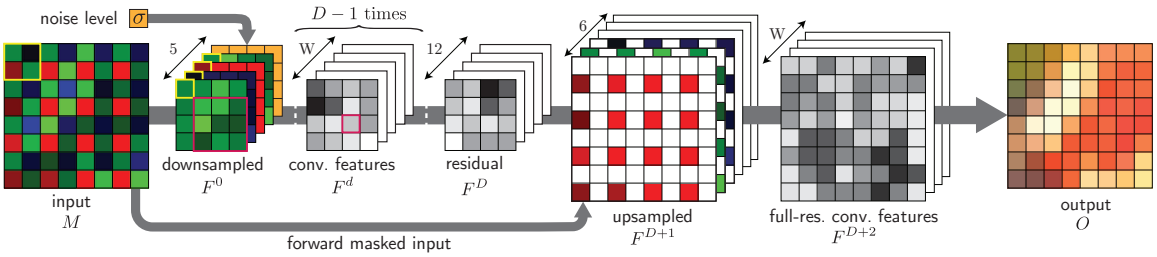


Figure 5-2: Our proposed architecture. The first layer of the network packs 2×2 blocks in the Bayer image into a 4D vector to restore translation invariance and speed up the processing. We augment each vector with the noise parameter σ to form 5D vectors. Then, a series of convolutional layers filter the image to interpolate the missing color values. We finally unpack the 12 color samples back to the original pixel grid and concatenate a masked copy of the input mosaick. We perform a last group of convolutions at full resolution this time to produce the final features. We linearly combine them to produce the demosaicked output.

of natural correlations in the data. Second, the network can represent a superset of the pipelines implemented by many previous techniques while all its parameters are optimized jointly to minimize a single objective.

A network alone is not sufficient to tackle denoising/demosaicking. We will see in Section 5.3 that the choice of training data has critical impact, especially because difficult inputs are rare yet cause visually disturbing artifacts.

We cast joint denoising and demosaicking as a supervised learning problem: we train our algorithm on a set of input measurements for which the desired output is known. We create the training set from millions of sRGB images, generating the corresponding mosaicked arrays by leaving out two color channels per pixel and adding noise. We then build a convolutional neural network and train it in an end-to-end fashion. The inputs are the mosaicked array M with a single channel per pixel and an estimate σ of the noise level; the output is an image O of the same size with a RGB triplet per pixel. We start our exposition focusing on demosaicking and then discuss noise.

5.2.1 Network architecture

We use a standard feed-forward network architecture to implement our demosaicking operator (Figure 5-2). Our network is composed of $D+1$ convolutional layers. Each convolution layer has W outputs and uses kernels of size $K \times K$. We denote by F^d the feature map of the d-th layer. In addition to the input mosaick M, the network takes as input an estimate of the noise level σ. We first describe the general architecture of the network. Details on how σ comes into play can be found in § 5.2.2. Since the Bayer mosaick is ubiquitous, we specialize our network to exploit its structure. We show however in § 5.4.5 that our approach generalizes to non-Bayer patterns.

We first rearrange the samples of the Bayer input mosaick to obtain a quarter-resolution multi-channel image which makes the spatial pattern translation invariant with a period of 1 pixel and reduces the computational cost of the subsequent steps. The first layer F^0 extracts 2×2 patches from M and packs them as a 4 channel feature map indexed by c.

$$F_c^0(x,y) = \mathrm{M}\Big(2x + (c \bmod 2),\ 2y + \Big\lfloor \frac{c}{2} \Big\rfloor\Big) \tag{5.1}$$

The bulk of the processing is performed at this lower resolution by the next D layers. They share the same structure and consist in convolutions with a bank of filters of spatial footprint $K \times K$ followed by a point-wise ReLU non-linearity $f(\cdot) = \max(0, \cdot)$.

$$F_c^d = f\left(b_c^d + \sum_{c'=1}^{W} w_{cc'}^d * F_{c'}^{d-1}\right) \text{ for } c \in \{1 \dots W\} \tag{5.2}$$

b_c^d is a scalar bias for the c-th channel of layer d, and $w_{cc'}^d$ is a two-dimensional convolution kernel of size $K \times K$. Each layer uses a total of W^2 such filters. The final low-resolution feature map F^D has 12 channels instead of W (and accordingly uses $12W$ filters). These final features correspond to the color samples of a 2×2 neighborhood. We upsample them back to full-resolution, reversing the process of

Equation 5.1. We also concatenate masked copies of the input mosaick M as channels in F^{D+1}. The masks m_c effectively isolate the RGB color samples on three distinct channels.

$$F_c^{D+1}(x,y) = m_c(x,y)\mathrm{M}(x,y) \text{ for } c \in \{1\dots3\} \tag{5.3}$$

$$F_c^{D+1}(x,y) = F_{c'}^{D}\left(\left\lfloor\frac{x}{2}\right\rfloor, \left\lfloor\frac{y}{2}\right\rfloor\right) \text{ for } c \in \{4\dots6\} \tag{5.4}$$

Here $c' = 4(c-4) + 1 + (x \bmod 2) + 2(y \bmod 2)$. This implements a form of residual network: we fast-forward the *identity mapping* to deeper layers, thereby allowing the network to learn a residual instead of the absolute mapping. Propagating the identity through many non-linear layers is harder an uses more parameters than with this shorcut [He et al., 2016]. However, we do not force the network to use both the fast-forwarded identity and the non-linear stack in fixed proportions but rather let it learn the appropriate mix: we perform a last convolution (Equation (5.2)), at full resolution this time, to produce F^{D+2}. The final output O of the network is an affine combination of the last feature maps F^{D+2}.

$$\mathrm{O}_c(x,y) = b_c^{\mathrm{O}} + \sum_{c'} w_{cc'}^{\mathrm{O}} F_{c'}^{D+2}(x,y) \tag{5.5}$$

Overall we opted for a thin (small W), deep (large D) architecture that is most similar to that of [Simonyan and Zisserman, 2014]. We experimented with networks of depth from $D = 5$ up to 20. For each convolutional layer, we used kernels with spatial footprint $K = 3$. The network thus implements a non-linear filter with a receptive field of $2D(K-1) + K + 1$ pixels with respect to the input's resolution. We pad the input of each convolution layer by $\frac{K-1}{2}$ pixels on each side so that the spatial dimension does not decrease with depth. The network thus also learns the boundary condition and does not reduce the dimensions of the input image, which would happen if we were to keep only the valid part of the convolutions. While processing the image at full-resolution with the color masks of Equation (5.4) directly applied to the input

mosaick is possible (§ 5.4.5), it incurs a higher computational cost since the network then processes four times as many pixels. This also reduces the receptive field of the final layer. We did not find this alternative approach to significantly affect the denoising/demosaicking performances.

5.2.2 Joint denoising with multiple noise levels

A combination of Poisson and Gaussian noises in *linear* space accurately models camera noise [Foi et al., 2008]. Because we work with white-balanced gamma-corrected sRGB images, we use an additive Gaussian noise model as [Jeon and Dubois, 2013] recommends.

We want to alleviate the need for a specialized network for each noise level. Instead, we train a single network on a continuous range of noise levels and explicitly add the noise level as an input parameter to the network. At training time and for each new input M, we randomly sample a noise level $\sigma \in [\sigma_1, \sigma_2]$. We corrupt M with a centered additive Gaussian noise of variance σ^2 before feeding it to the network. We also provide the network with the scalar estimate of the noise level σ as extra input (Figure 5-2). In practice, since camera model and settings are stored alongside the raw data and one can rely on offline noise calibration, the noise level is typically known and used to inform demosaicking. In order to incorporate this new information into the convolutional architecture, we spatially replicate the noise level to match the input dimensions of the first layer F^0 and concatenate it as an extra channel: F^0 now has 5 channels (Figure 5-2). [Burger et al., 2012] used a similar approach for denoising-only in a non-convolutional setup.

5.2.3 Training details

At training time, we use a dataset $\mathcal{D} = \{(\sigma_i, \mathrm{M}_i, \mathrm{I}_i)\}_i$ of mosaicked/ground-truth image patches where M_i is generated from I_i and corrupted with additive white Gaussian noise of variance σ_i^2 on-line. We optimize the weights and biases by minimizing the

normalized L_2 loss on this training set:

$$\mathcal{L}\left(\left\{w^{(d)}, b^{(d)}\right\}_d\right) = \frac{1}{p^2|\mathcal{D}|}\sum_i ||\mathrm{O}_i - \mathrm{I}_i||^2 \tag{5.6}$$

In all our experiments, we use a patch size $p = 128$ pixels for the training samples. The filter weights $w^{(d)}$ are initialized according to [He et al., 2015] and the biases $b^{(d)}$ are first set to 0. The optimization is carried out by ADAM [Kingma and Ba, 2015], a flavor of stochastic gradient descent that maintains an adaptive estimate of the first and second order moments of the gradient and uses them to be independent of any diagonal rescaling of the gradient. We use a batch size of 64, and an initial learning rate of 10^{-4}. We used an L_2 weight decay of 10^{-8} on $w^{(d)}$. All other parameters are left at the value recommended by the authors. As learning progresses, we decrease the learning rate by a factor 10 whenever the validation error on an independent dataset stalls, typically twice, after 10 epochs. In our experience, higher initial learning rates fail to converge to a good solution. The training is performed with a customized version of CAFFE [Jia et al., 2014] on a NVIDIA Titan X, and usually takes 2-3 weeks.

5.3 Curating a dataset of challenging images

When trained on standard datasets, our neural network works well on average but produces disturbing artifacts on a number of hard cases, the common plague of demosaicking and denoising. These challenges are due to two important issues. First, hard cases are rare and get diluted by the vastly more common easy areas. Second, metrics such as L^2 or PSNR fail to notice demosaicking artifacts that are salient to humans.

We now present an algorithm for detecting challenging patches and focus the training on them using a combination of adaptive training based on human visual difference predictors and a new metric optimized to detect moiré artifacts.

We first train a network on standard datasets and use it to demosaick and denoise millions of ground-truth photographs in order to mine for hard cases. We look for two classes of artifacts frequently missed by the network: luminance artifacts and color moiré. Inspired by curriculum learning [Bengio et al., 2009], we adaptively build a new dataset composed of these artifact-prone patches. We use this dataset to fine-tune or train the network from sratch. This improves the model's performance on difficult cases and can be seen as reweighting the loss function to give more weight to artifact-prone patches. Next, we discuss our selection strategy and the metrics we use.

5.3.1 Obtaining a ground-truth and the corresponding mosaic

We start with a large number of sRGB images downloaded from the web to generate ground truth data. We create a mosaick from each image, add noise, and use this pair for training. We restrict our selection to images with at least 16 Mpix to favor higher quality images. To avoid biasing the network towards distortions caused by the camera pipeline that first created the downloaded images, we downsample them by a factor 4 using bicubic interpolation and use this as ground-truth. While more complex downsampling techniques are possible [Khashabi et al., 2014], they do not help in our context: our training images are JPEG-compressed and come from unknown and diverse sources.

We create the mosaicked and noisy image M by retaining only one color channel per pixel according to the Bayer pattern from the sRGB image. We also augment patches from the training set with random rotations in 90° steps, random left-right mirror images, and 1-pixel shifted copies in either dimension. This augments the training data by a factor 32× and provides some rotational and translational invariance.

5.3.2 Challenging patches are rare

Publicly available demosaicking datasets contain a few hundred images which is insufficient for training the thousands of parameters of a deep network. Instead, we

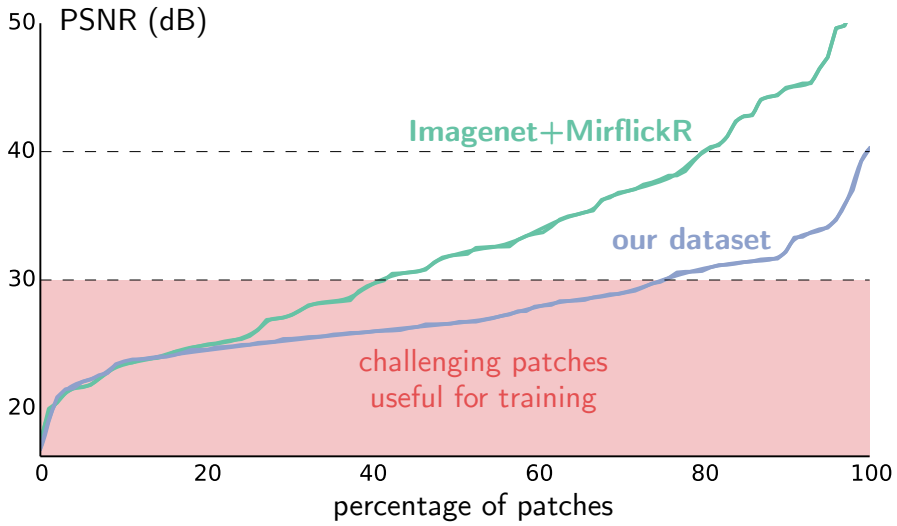


Figure 5-3: Most of the patches in a generic training dataset (here Imagenet and Mirflickr) are easy cases for modern demosaicking algorithms (in this figure, we measure the PSNR of AHD Hirakawa and Parks [2005]). For a network to perform well in challenging situations, it needs to be trained on challenging patches, i.e., on data that lies on the tail of the patch distribution. This plot shows that our dataset contains more such patches. Further, not shown in this figure is the fact that demosaicking failures on our patches lead to more visually unpleasant artifacts: we explicitly selected the patches for this reason.

trained our first network using 1.3 million images from Imagenet [Deng et al., 2009] and 1 million images from MirFlickr [Huiskes and Lew, 2008]. While this network matches the PSNR statistics of previous work, a closer inspection reveals artifacts near thin edges and complex textures (see Figure 5-4). Large quantity of training samples do not guarantee convincing demosaicking.

A random selection of images is mainly composed of smooth patches as these dominate natural images [Levin et al., 2012]. Challenging structures only make up a small fraction, shown as the tail of the patch distribution in Figure 5-3. Smooth patches account for a majority of the training time, even though results on such cases are already perceptually indistinguishable from ground truth. We compensate for this by assembling a training set with more difficult patches.

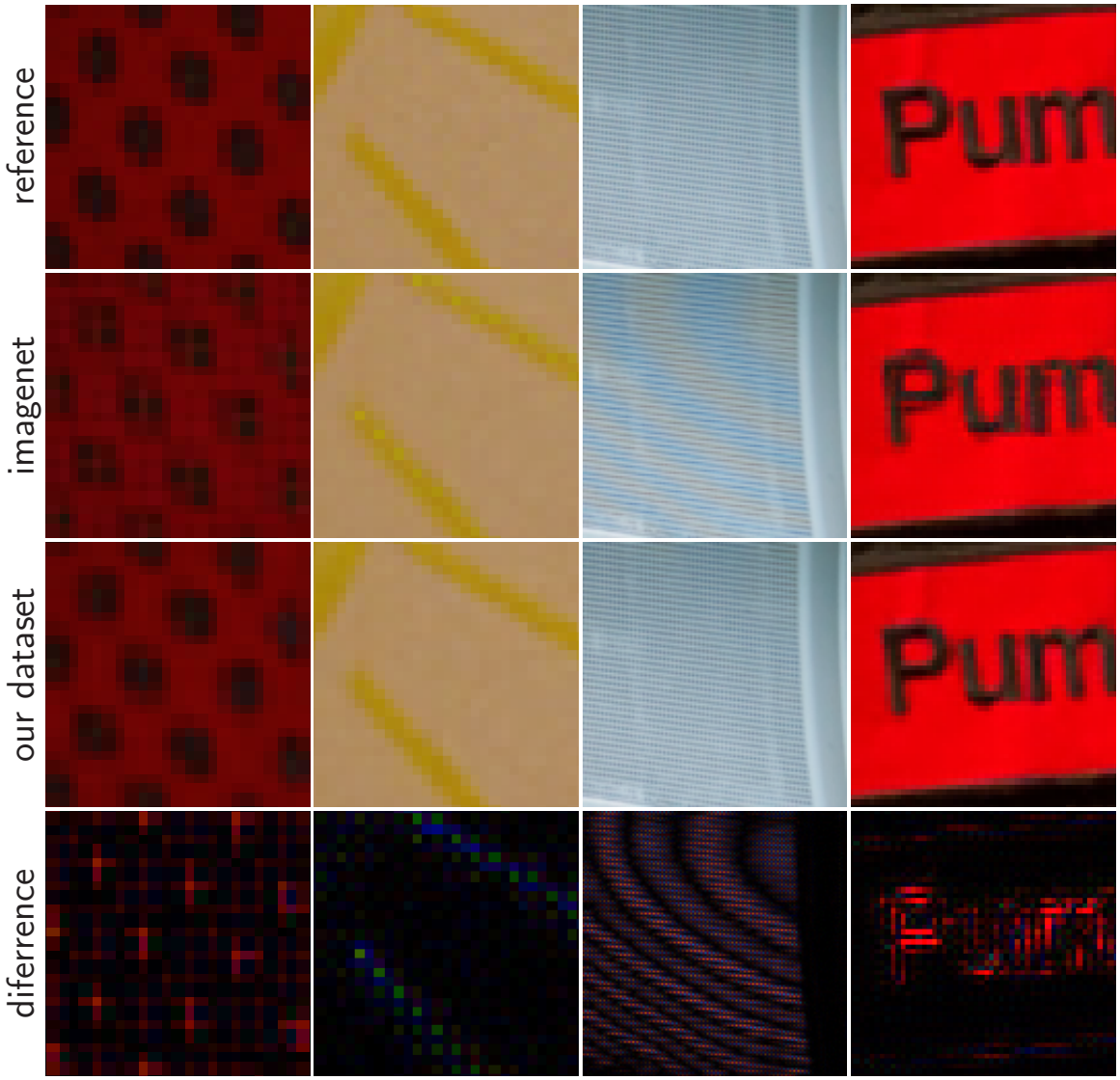


Figure 5-4: A network trained on a standard image dataset (second row) creates noticeable artifacts in its output such as the zippering on the thin yellow line, confusion around curves in the first and last example, and moiré in the third example. When training the same network on our new dataset of difficult cases, these artifacts are mostly gone (third row). The last row shows the difference map between the two network outputs, and the first row is the ground-truth image. (best viewed in digital form)

5.3.3 Mining difficult image patches

We create a database of difficult patches by applying the first network (trained on Imagenet) to millions of new patches we download from the web and retaining the failure cases. We detect patches that pose two specific challenges: luminance artifacts around thin structures (e.g. *zipper*) and color moiré. We use separate metrics to detect these cases. Rejecting trivial cases effectively reweighs the loss function (Eq. (5.6)) towards challenging ones.

Salient luminance artifacts We first use the perceptually based HDR-VDP2 [Mantiuk et al., 2011] to detect luminance artifacts around thin edges. We have found that standard metrics like PSNR, SSIM, S-CIELAB do not capture perceptual artifacts as convincingly as HDR-VDP2. It has also been empirically shown to correlate well with human judgement for simpler demosaicking [Sergej and Mantiuk, 2014]. It compares the visibility of local artifacts as well as overall image quality to a reference. It models the response of the human visual system including phenomena such as spectral sensitivity, luminance adaptation and frequency masking and is calibrated against contrast sensitivity measurements. For each new image, we apply demosaicking using the pre-trained network. We then compare the network's output to the ground truth using HDR-VDP and compute a probability of artifacts at each pixel. We smooth the probability map using Gaussian blur ($\sigma = 3$) and extract up to 30 local maxima if the artifact probability exceeds 0.1. This resulted in 2,489,180 problematic patches from 1,393,107 15.2 Mpix images ($\sim$ 3% of total pixels). We adjusted the metric to approximate the response of a human viewing a 2560×1600 30-inch sRGB display from a distance of 1m. HDR-VDP detects high-frequency luminance artifacts (Fig. 5-5a); training our network on these patches yielded drastically improved results. The metric however misses color moiré artifacts because it only analyzes the luminance channel (Fig. 5-5, bottom row).

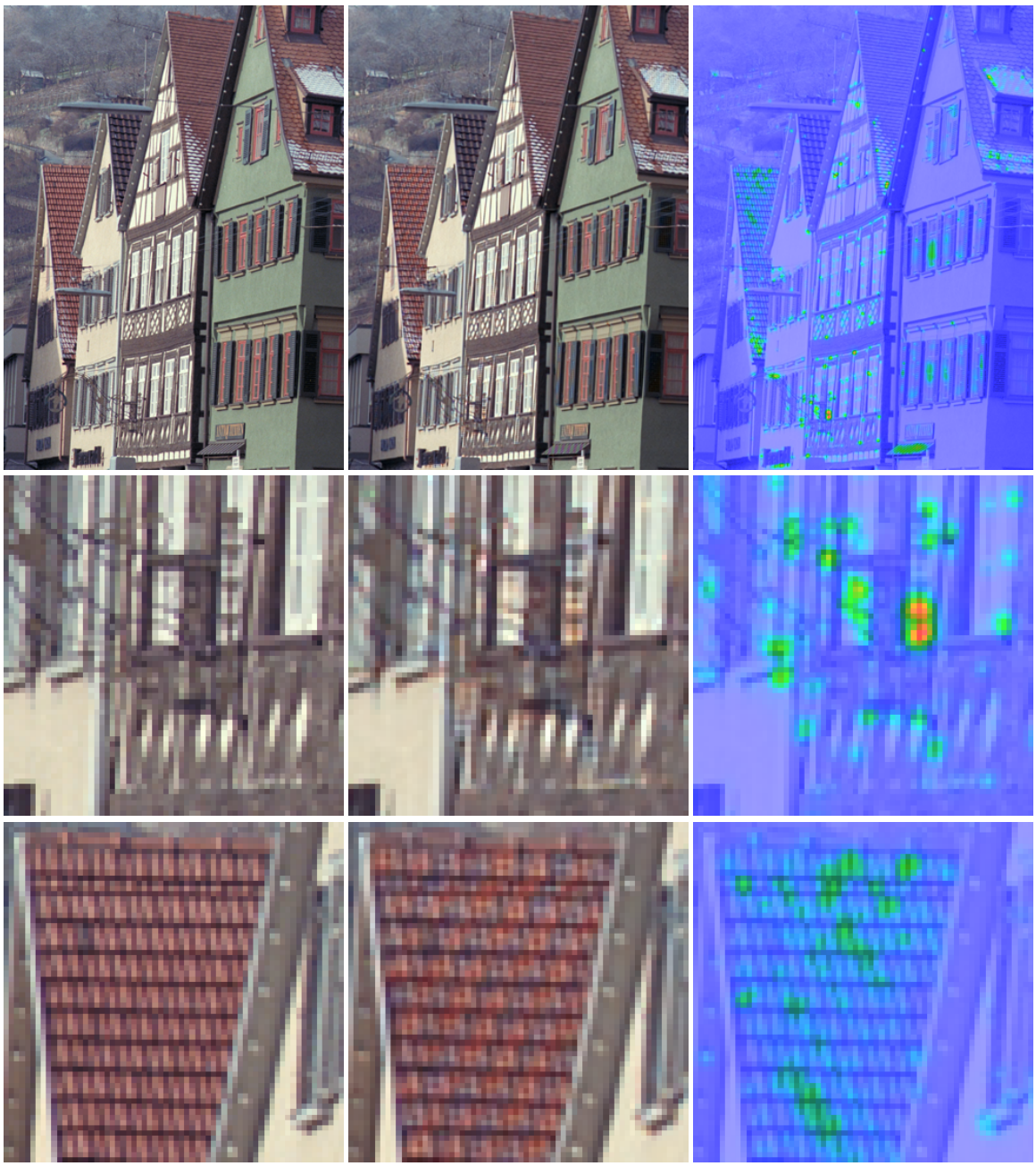


Figure 5-5: HDR-VDP run on the demosaicked output of LDI-NAT Zhang et al. [2011]. First, row full image. Second row, HDR-VDP correctly detects the zipper pattern due to luminance variations. It signals some anomalies in the output image but with a low probability of detection. It works only on luminance, therefore misses the chrominance moiré artifacts.

Moiré and aliasing Moiré is an interference pattern caused by aliasing. Repetitive details close to or smaller than the resolution of the sampling grid can give rise to artificial low frequency patterns. Mosaic images have their color channels at spatial offsets; moiré appears as distracting false color bands (Fig.5-6b) after demosaicking because of erroneous interpolation of color samples. The effect of aliasing is best understood in the Fourier domain because it introduces undesirable frequencies. We quantify moiré artifacts by measuring the change in frequency content from the ground-truth I to the demosaicked image O image. We first convert both I and O to the *Lab* space and compute the 2D Fourier transform of each channel $\mathcal{F}_{\mathrm{I}}(\omega)$ and $\mathcal{F}_{\mathrm{O}}(\omega)$ respectively. We then compute the gain of the demosaicked image with respect to the input at each frequency.

$$\rho(\omega) = \begin{cases} \log\left(\frac{|\mathcal{F}_{\mathrm{O}}(\omega)|^2+\eta}{|\mathcal{F}_{\mathrm{I}}(\omega)|^2+\eta}\right) & \text{if } |\omega| \leq r \\ 1 & \text{otherwise} \end{cases} \tag{5.7}$$

We only compare the gain in frequencies lower than r to mitigate boundary effects and high-frequency noise. We smooth the gain map with Gaussian blur and mark the patch as aliased if the maximum gain value across all channels and frequencies exceeds a threshold t. For 128×128 patches, we set the the low-pass radius $r = 0.95\pi$, the standard deviation of the Gaussian kernel to 3, and gain map threshold to $t = 2$. This criterion consistently selects moiré-prone patches. Figure 5-6 shows the gain map for an aliased patch.

These moiré patches are rare; they lie at the end of the tail of natural patch distribution (Figure 5-3). We found 0.05% of patches from 2 million images patches are aliased. Nonetheless, these artifacts are important because they can still affect large areas of an image (e.g. a 128×128 patch), making it unusable.

(a) input (b) demosaicked output (c) error (d) amplitude gain in the Fourier domain

Figure 5-6: Frequency Gain due to moire

5.4 Results

We evaluate our network in various conditions. Unless stated otherwise, all the experiments in this section use a network with $D = 15$ layers (each with $W = 64$ 3×3 filters) trained from scratch on 2,590,186 128×128 hard patches. The network has 559,776 trainable parameters. We stop the training when the error on a separate validation set of 4000 images stops decreasing. We test all techniques on another dataset of 2000 images. All three datasets are independent and have been mined in the same fashion, as described in Section 5.3. Half of the test set was mined using the HDR-VDP metric (we refer to this half as the *vdp* test set). The other half was assembled using the moiré metric (we refer to it as *moiré*). The parameters of competing techniques are set to the values recommended by their authors, often tuned on the Kodak/McMaster datasets included in our comparison. Our main metric is PSNR where the error is averaged over pixels *and* color channels before taking the logarithm.

First, we compare our algorithm against previous work on the demosaicking-only task with noise-free sRGB images (Table 5.1, in particular no denoising is applied). This evaluation illustrates that high PSNR statistics can obscure subtle perceptual artifacts: we demonstrate this on hard cases from our testing dataset (Figure 5-9). We then present our results on demosasicking noisy inputs, which we refer to as

joint denoising and demosasicking (Figure 5-7). Although our network is trained on 8-bits sRGB data, we also evaluate our network on linear RGB data (Table 5.2) and non-Bayer mosaicks. This shows that our approach generalizes to other demosaicking conditions. We finally describe implementation details and show that our algorithm is faster than the previous best-performing methods on both CPU and GPU.

5.4.1 Demosaicking noise-free images

We first evaluate our algorithm on noise-free inputs from two common demosaicking datasets: McMaster [Zhang et al., 2011] and Kodak [Li et al., 2008]. Table 5.1 (first two columns) show that our network outperforms the previous techniques on these datasets. These results alone however are not sufficient because these datasets are known to have flaws and to misrepresent the statistics of digital images [Levin et al., 2012]. To provide a more accurate depiction of the demosaicking challenges, we also compare our technique with the state of the art on a testing set of 2000 hard cases not seen during training (Table 5.1 third and fourth columns). Our method produces consistently better results quantitatively and the improvement is also visually significant (Figure 5-9). Our network (trained on difficult cases) successfully handles complex patterns and generates artifact-free results. We also compare to the widely used Adobe Camera Raw software. Results for all the datasets and techniques can be found in the supplemental material. Since the test images are noise-free, no denoising has been applied in this experiment.

5.4.2 Training set and training time

We initially trained our network on the 1.3 million images from Imagenet [Deng et al., 2009] and 1 million from MirFlickr [Huiskes and Lew, 2008] instead of our dataset of difficult cases. Despite reaching competitive PSNR levels (on-par with FlexISP [Heide et al., 2014]), the network produced noticeable artifacts, mainly along thin structures and moiré-prone textures. We believe that this is due to the inherent bias

	kodak	mcm	vdp	moiré
bilinear	32.9	32.5	25.2	27.6
Adobe Camera Raw 9	33.9	32.2	27.8	29.8
Klatzer* [2016]	35.3	30.8	28.0	30.3
Gunturk [2002]	35.8	33.2	29.3	31.3
Lu [2010]	36.0	33.4	29.4	31.4
Li [2005]	36.1	33.1	29.2	31.5
Hirakawa [2005]	36.1	33.8	28.6	30.8
Condat [2011]	35.5	33.3	28.4	30.9
Condat [2012]	36.1	33.6	29.6	31.9
Jeon [2013]	36.4	34.0	27.8	30.4
Hirakawa [2006]	36.5	33.9	30.0	32.1
Hamilton [1997]	36.9	35.2	28.9	30.9
Zhang [2005]	37.3	34.7	30.3	32.4
Buades [2009]	37.3	35.5	29.7	31.7
Zhang (NLM) [2011]	37.9	36.3	30.1	31.9
Getreuer [2011]	38.1	36.1	30.8	32.5
Heide [2014]	40.0	38.6	27.1	34.9
ours	**41.2**	**39.5**	**34.3**	**37.0**

Table 5.1: PSNR comparison of our approach to state of the art techniques on the demosaicking-only scenario. First and second column show evaluation on standard datasets. Third and fourth column show comparisons on our datasets containing images prone to luminance artifacts and color moiré respectively. No denoising is applied for any of the competing methods. (*) For Klatzer et al., we used the published model which was trained on linear data and ran it on *linearized* images. The training code was not available at the time of publication.

of these standard datasets towards trivial cases like smooth patches or unambiguous edges. Training a network on the difficult cases significantly improves visual quality (Figure 5-4). We found that fine-tuning the *Imagenet+MirFlickr* network or retraining from scratch worked equally well. All the results we report are trained from scratch on the hard examples only. Accuracy is numerically competitive after a day of training but image quality improves with longer training. Week-long training is common with deep networks and has no impact on the practicality of our approach since it is done only once before the algorithm is deployed.

5.4.3 Joint denoising and demosaicking results

We now present results for joint denoising and demosaicking (Figure 5-10). We train on images corrupted with continuous levels of noise $\sigma \in [0; 20]$. Similar to previous work, we model noise in the white-balanced gamma-corrected images as signal-independent white Gaussian noise [Jeon and Dubois, 2013]. During evaluation, we tested images at 6 levels of noise within the range used for training (Fig. 5-7). Our results consistently outperform previous techniques on all noise levels.

We also experimented with networks trained on a single noise level instead of continuous levels and did not observe noteworthy change in result quality (Fig. 5-7). This suggests that the network is already optimally trained, and does not require fine-tuning for each noise level.

5.4.4 Processing linear data

Khashabi et al. [2014] suggest that demosaicking should be evaluated on raw RGB data with an affine noise model [Foi et al., 2008; Hasinoff et al., 2010]. In previous experiments, we instead trained and evaluated on sRGB to facilitate comparisons with state-of-the-art techniques that choose to do the same. Without any further training on linear data or affine noise models, our sRGB trained network outperforms the best techniques on the MSR 16-bits linear Panasonic testing set [Khashabi et al., 2014]

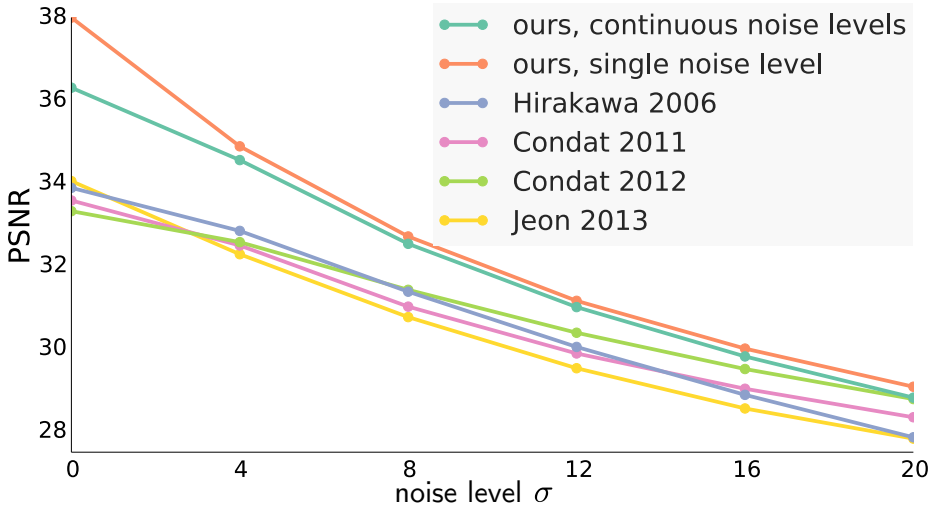


Figure 5-7: PSNR comparison joint denoising and demosaicking at different levels of Gaussian noise with standard deviation σ. The metric is averaged across all four datasets from Table 5.1: mcm, kodak, vdp, moiré.

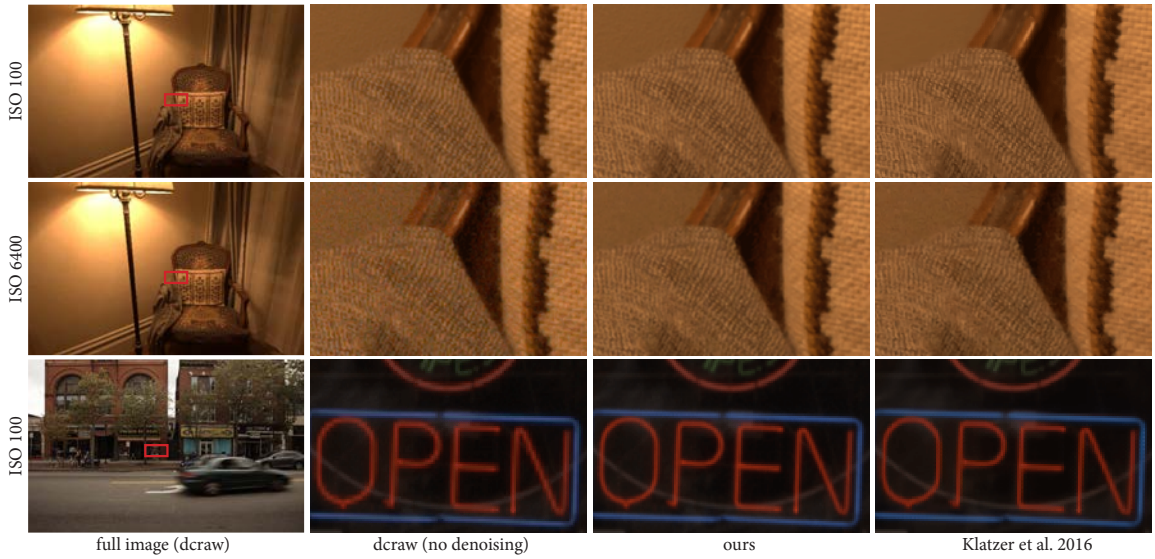


Figure 5-8: Our algorithm trained on sRGB data generalizes to real (linear) RAW images. It successfully removes color moiré on the fabric at various noise levels whereas dcraw does not (first and second rows). In comparison, Klatzer et al. [2016] does not generalize well to widely different noise levels: it denoises too aggressively the ISO100 image (first row), and produces artifacts on the ISO6400 image (second row). This is particularly visible on the smooth wall. Our output is free of checkerboard patterns and staircase artifacts (third row). DCRaw exhibits these artifacts on the red lettering and Klatzer et al. has checkboard on the blue line on the right.

(Table 5.2). Since noise parameters for individual images are not provided in this dataset, we estimate the average noise variance and use it as our noise parameter.

We also fine-tuned our network on our dataset of hard cases *linearized* from sRGB and observed virtually the same performance. This shows that our network is not restricted to sRGB data and generalizes well to linear data. Real RAW training data would be ideal, but available datasets do not contain enough challenging cases: we observed no quality improvement when training on MIT5k [Bychkovsky et al., 2011a] or the MSR training set. Figure 5-8 shows the output of our algorithm on real (linear) RAW images captured by a Canon 5D mark II at various ISO levels.

	noise-free		with noise	
	linear	sRGB	linear	sRGB
bilinear	30.9	24.9	–	–
Hamilton [1997]	36.7	30.0	–	–
Hirakawa [2005]	37.2	31.3	–	–
Zhang (NAT) [2011]	37.6	31.6	–	–
Gunturk [2002]	38.2	31.0	–	–
Lu [2010]	38.3	31.0	–	–
Zhang (NLM) [2011]	38.4	32.1	–	–
Zhang [2005]	38.8	31.7	–	–
Getreuer [2011]	39.4	32.9	–	–
Khashabi [2014]	39.4	32.6	37.8	31.5
Heide* [2014]	40.0	33.8	–	–
Klatzer [2016]	40.9	34.6	**38.8**	**32.6**
ours	41.6	35.3	38.4	32.5
ours (f.t.)	**42.7**	**35.9**	38.6	**32.6**

Table 5.2: Evaluation on linear data for both noise-free and noisy data. We report PSNR in both linear and sRGB space. We feed a single estimate of the average noise level to our network a test time. We also fine tuned our network to linearized sRGB. Among all competing techniques, only KhashabiKhashabi et al. [2014] and Klatzer Klatzer et al. [2016] techniques were specifically designed for linear data. Results on noisy images are excluded from the table for methods that do not attempt to denoise.

5.4.5 Alternative mosaick patterns

With a few simple modifications, our method generalizes to non-Bayer patterns. We experimented with the Fuji X-Trans pattern. Compared to the Bayer network illustrated in Figure 5-2, we no longer process the image at quarter resolution. Instead, the mosaicked input RGB values are kept at full-resolution on separate planes: we remove layers F^0 and F^{D+1}. The X-Trans pattern is 6x6 pixels; this would imply a much more aggressive downsampling. At train time, we apply a color mask to the ground truth that converts it to the non-Bayer mosaick. We trained this modified network from scratch for three days on our dataset of hard-cases. We evaluate this new network on the MSR Panasonic X-Trans dataset [Khashabi et al., 2014]. The table below shows that our algorithm consistently performs better than previous techniques.

	linear	sRGB
Khashabi [2014]	36.9	30.6
Klatzer [2016]	39.6	33.1
ours	**39.7**	**33.2**

5.4.6 Variations on the network configuration

Using as few as $D = 7$ layers has a minimal impact on general accuracy, but patches prone to moiré are significantly degraded: they benefit from the large spatial footprint of the deeper network. $W = 64$ filters per layer worked well, using $W = 128$ was superfluous and $W = 32$ decreased the PSNR.

5.4.7 Running time

Unless stated otherwise, we benchmark all methods with 1MPix images on an Intel Core i7-3770K and GeForce Titan 700 and report the average time over 10 runs. Our technique is linear in the pixel count. Superlinear competitors can be made to run in linear-time by processing the image in tiles. We use the Halide image

	CPU (ms/Mpix)	GPU (ms/Mpix)
bilinear	127	–
Hamilton [1997]	385	–
Condat [2011]	566	–
Lu [2010]	737	–
Li [2005]	1117	–
Hirakawa [2006]	1618	–
Gunturk [2002]	1991	–
Hirakawa [2005]	2998	–
Condat [2012]	11,211	–
Jeon [2013]	14,728	–
Zhang [2005]	30,642	–
Khashabi [2014]	36,157*	–
Zhang (NLM) [2011]	264,243	–
Zhang (NAT) [2011]	1,700,510	–
Heide [2014]	1,815,111	3000*
Klatzer [2016]	3,560,510	1600*
ours	**2,932**	**325**

Table 5.3: Runtime of different demosaicking algorithms in their publicly available implementations. Our approach is faster than previous high quality techniques like FlexISP Heide et al. [2014]. Timings with an asterisk (*) are reported from the respective original paper.

processing language [Ragan-Kelley et al., 2013a] to implement our network. Our CPU implementation of a $D = 15$ layers network is 8× faster than ATLAS Caffe [Jia et al., 2014] and 3.5× faster than Caffe with Intel's Math Kernel Library. It processes image at 3s/Mpix on a modern desktop CPU. Our approach is up to two orders of magnitude faster than previous high-quality techniques that use global optimization like FlexISP [Heide et al., 2014] and other non-local techniques [Zhang et al., 2011] (Table 5.3).

5.5 Limitations

Our approach relies on image metrics to detect challenging patches and build a ground-truth dataset. We used HDR-VDP for luminance artifacts, but it is not perfect and we can benefit from a better metric. Also, if the sRGB ground truth is corrupted with

color moiré, our network will learn the corruption; a no-reference moiré detector is required to alleviate this.

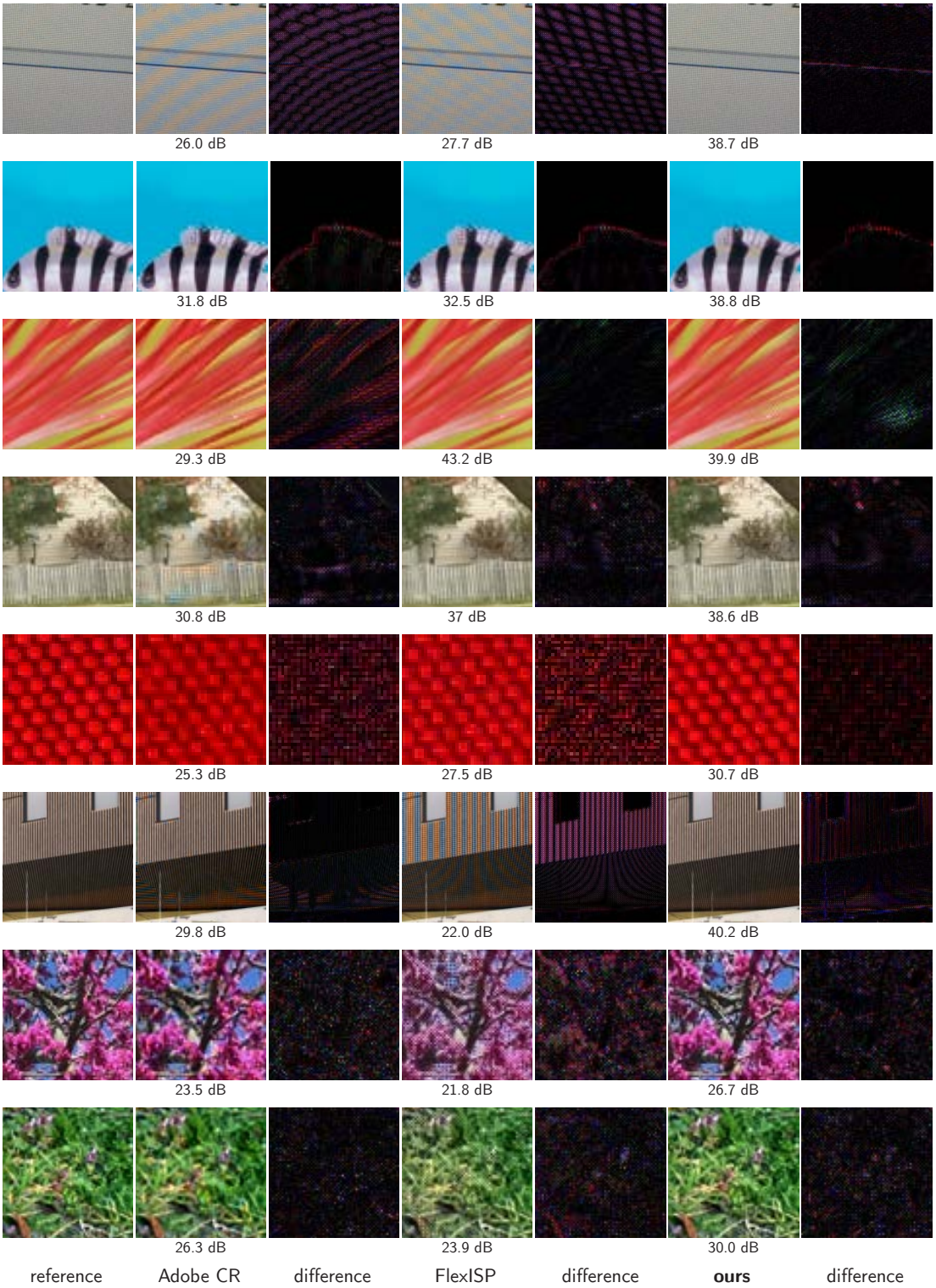


Figure 5-9: Comparison of our approach with Adobe Camera Raw, FlexISP Heide et al. [2014] on noise-free images. Exhaustive results can be found in supplementary material.

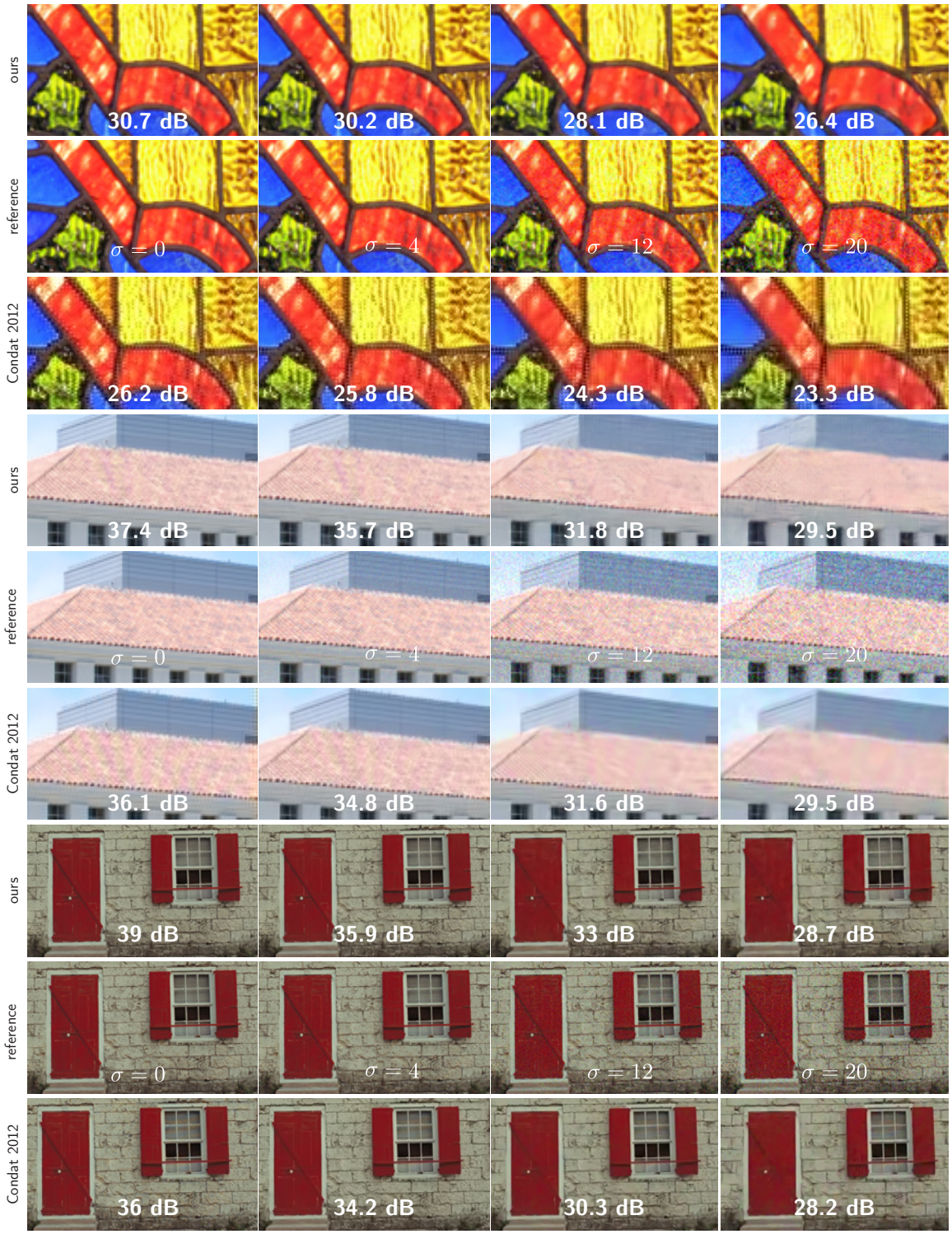


Figure 5-10: Joint denoising and demosaicking results. Our approach outperforms previous best techniques on noisy data in challenging images on different levels of Gaussian noise with standard deviation σ. Exhaustive results can be found in supplementary material.

Chapter 6

Conclusion

In this thesis, we have developed fast algorithms for high-fidelity image processing. These algorithms are parameterized such that they can be reconfigured to approximate new photographic transformations on-demand (chapters 3 and 4). Their computation is fully differentiable, which makes them amenable to automatic parameter optimization. This property, paired with appropriate training datasets, led to significant quality improvements on difficult, long-standing image processing problems (chapter 5).

We have seen that cloud image enhancement can only be made more efficient if the power and time cost of data transfers is minimized. In the case of algorithms that preserve the content of an image and do not spatially warp it, we have shown that we can exploit the similarity between the input and the output images. We have introduced *transform recipes*, a flexible representation which captures how images are locally modified by an image operator. Recipes dramatically reduce the bandwidth required for cloud computation. They are compact and allow us to work on the server with a highly-degraded input-output pair, and apply the recipe on the client to the high-quality input. Finally, we have demonstrated that transform recipes can support a wide range of algorithms and, in practical scenarios, lead to reduced power and time costs. Transform recipes laid out the basic building blocks for general filter approximation that inspired our subsequent work.

We have presented a new network architecture that performs image enhancement in real-time on full-resolution images while still accurately capturing high-frequency effects. Unlike transform recipes, which are tailored to a single image, our model is trained using many pairs of input/output images, allowing it to learn from a reference implementation of some algorithm or even from human adjustments. By performing most of its computation within a bilateral grid and by predicting local affine color transforms, our model is able to strike the right balance between expressiveness and speed. We train it end-to-end and optimize the loss function at full resolution (despite most of the model working at a heavily reduced resolution). Thus, our model is capable of learning full-resolution and non-scale-invariant effects. Its accuracy has been demonstrated on a variety of different image operators, pipelines, and subjective human-annotated datasets.

Some image processing tasks require more direct control over the pixels. We demonstrated that a joint approach based on a convolutional network can significantly improve the quality of demosaicking and denoising. It can resolve even challenging situations that usually result in zippering or moiré artifacts. However, we have seen that traditional supervised learning must be adapted because the vast majority of image regions are easy to address and the real hard cases do not occur enough and are not well characterized by even advanced perceptual image metrics. We have proposed an adaptive approach as well as a new moiré detection metric to tackle these challenges. Our method outperforms state-of-the-art solutions in terms of both perceptual and statistical visual quality, while being an order of magnitude faster.

6.1 Future directions

The applications we have presented in this dissertation show that gradient-based optimization of highly-parameterized pipelines can improve the speed and quality of many image processing tasks. The tools we used to develop these models provide

gradients for compositions of a limited set of operators. This often turned out to be insufficient for our purpose, for instance when we required particular access patterns or custom image manipulation routines. We implemented these custom operations as low-level extensions to deep learning packages, which also required manually deriving and implementing the gradients. This is an error-prone process that slows down prototyping, because the forward and gradient code needs to be sufficiently optimized to keep training time reasonable. Having derivatives as first-class citizen in a programming language and automatic code optimization would greatly streamline this workflow. We have already taken steps in this direction, adding automatic differentiation [Griewank and Reese, 1991] to the Halide programming language, together with an automatic scheduler that synthesizes fast derivative code [Li et al., 2018]. The general optimization of low-level code for pipelines that grow increasingly complex remains a challenging endeavor. Can machine learning help us explore some of the performance trade-offs?

The human eye is highly sensitive to rare, but catastrophic image artifacts. Assembling datasets that exhibit these issues to reweigh the loss function and give them more prominence, like we have done for demosaicking, is a delicate task. New metrics that correlate well with human perception could automate this selection process and generalize more easily to other problems. Recent work on perceptual losses built from neural networks are a step in this direction [Johnson et al., 2016; Berardino et al., 2017; Zhang et al., 2018]. The demosaicking problem also illustrates that oftentimes, only a small fraction of an image's pixels do in fact call for the more sophisticated reconstruction algorithm (e.g. smooth patches are easy to demosaick). A *no-reference* metric that could efficiently detect such patterns, and adaptively apply the costly reconstruction only where it is truly needed, would improve performance. Large and complex pipelines, that are optimized from data also raise new challenges. In particular, their failures modes are not well understood. The systematic analysis of such failures (e.g. [Goodfellow et al., 2014; Elsayed et al., 2018]) and the uncertainty

around model predictions [Kendall and Gal, 2017] is still a largely open problem.

Finally, machine learning offers unprecedented opportunities to automate tedious and repetitive tasks. By and large, automation is beneficial, but in a creative context, it could also become normalizing. The model of chapter 4 suggests we can make our algorithms more personalized. But it only scratches the surface and still requires thousands of training example pairs. Can our programs learn from limited datasets containing only a dozen of examples? Can they progressively adapt to a particular user's taste or style, tapping into sparse feedback signals?

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`.

Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum*, 2010.

H. Akiyama, M. Tanaka, and M. Okutomi. Pseudo four-channel image denoising for noisy cfa raw data. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 4778–4782, Sept 2015. doi: 10.1109/ICIP.2015.7351714.

Mathieu Aubry, Sylvain Paris, Samuel W. Hasinoff, Jan Kautz, and Frédo Durand. Fast local laplacian filters: Theory and applications. *ACM Trans. Graph.*, 33(5): 167:1–167:14, September 2014a. ISSN 0730-0301. doi: 10.1145/2629645. URL `http://doi.acm.org/10.1145/2629645`.

Mathieu Aubry, Sylvain Paris, Samuel W Hasinoff, Jan Kautz, and Frédo Durand. Fast local laplacian filters: Theory and applications. *ACM TOG*, 2014b.

Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

Kenneth C. Barr and Krste Asanović. Energy-aware lossless data compression. *ACM Trans. Comput. Syst.*, 24(3):250–291, August 2006. ISSN 0734-2071. doi: 10.1145/1151690.1151692. URL `http://doi.acm.org/10.1145/1151690.1151692`.

Jonathan T Barron and Ben Poole. The fast bilateral solver. *ECCV*, 2016.

Jonathan T Barron, Andrew Adams, YiChang Shih, and Carlos Hernández. Fast bilateral-space stereo for synthetic defocus. *CVPR*, 2015.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380. URL `http://doi.acm.org/10.1145/1553374.1553380`.

Alexander Berardino, Valero Laparra, Johannes Ballé, and Eero Simoncelli. Eigen-distortions of hierarchical representations. In *Advances in Neural Information Processing Systems*, pages 3533–3542, 2017.

Floraine Berthouzoz, Wilmot Li, Mira Dontcheva, and Maneesh Agrawala. A framework for content-adaptive photo manipulation macros: Application to face, landscape, and global manipulations. *ACM Transactions on Graphics*, 30(5), 2011.

Adrien Bousseau, Sylvain Paris, and Frédo Durand. User-assisted intrinsic images. *ACM TOG*, 2009.

Antoni Buades, Bartomeu Coll, Jean-Michel Morel, and Catalina Sbert. Self-similarity driven color demosaicking. *Image Processing, IEEE Transactions on*, 18(6):1192–1202, 2009.

H.C. Burger, C.J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2392–2399, June 2012. doi: 10.1109/CVPR.2012.6247952.

Peter J Burt and Edward H Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983.

Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *The Twenty-Fourth IEEE Conference on Computer Vision and Pattern Recognition*, 2011a.

Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. *CVPR*, 2011b.

Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011c.

Lanlan Chang and Yap-Peng Tan. Effective use of spatial and spectral correlations for color filter array demosaicking. *Consumer Electronics, IEEE Transactions on*, 50(1):355–365, 2004.

Jiawen Chen, Sylvain Paris, and Frédo Durand. Real-time edge-aware image processing with the bilateral grid. *ACM TOG*, 2007.

Jiawen Chen, Andrew Adams, Neal Wadhwa, and Samuel W Hasinoff. Bilateral guided upsampling. *ACM TOG*, 2016.

Qifeng Chen, Dingzeyu Li, and Chi-Keung Tang. Knn matting. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–876, June 2012. doi: 10.1109/CVPR.2012.6247760.

Zezhou Cheng, Qingxiong Yang, and Bin Sheng. Deep colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 415–423, 2015.

David R Cok. Signal processing method and apparatus for producing interpolated chrominance values in a sampled color image signal, February 10 1987. US Patent 4,642,678.

Laurent Condat. A new color filter array with optimal properties for noiseless and noisy color image acquisition. *Image Processing, IEEE Transactions on*, 20(8): 2200–2210, 2011.

Laurent Condat and Saleh Mosaddegh. Joint demosaicking and denoising by total variation minimization. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 2781–2784. IEEE, 2012.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

P. Deutsch. Deflate compressed data format specification version 1.3, 1996.

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. *ECCV*, 2014.

A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, Dec 2015a. URL `http://lmb.informatik.uni-freiburg.de//Publications/2015/DFIB15`.

Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1538–1546, 2015b.

David Eigen, Dilip Krishnan, and Rob Fergus. Restoring an image taken through a window covered with dirt or rain. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 633–640, 2013. doi: 10.1109/ICCV.2013.84. URL `http://dx.doi.org/10.1109/ICCV.2013.84`.

David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *NIPS*, 2014.

Gamaleldin F. Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alex Kurakin, Ian J. Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both human and computer vision. *CoRR*, abs/1802.08195, 2018. URL `http://arxiv.org/abs/1802.08195`.

Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. In *ACM Transaction on Graphics (SIGGRAPH)*, SIGGRAPH '08, pages 67:1–67:10, New York, NY, USA, 2008. ACM. ISBN 978-1-4503-0112-1. doi: 10.1145/1399504.1360666. URL `http://doi.acm.org/10.1145/1399504.1360666`.

Zeev Farbman, Raanan Fattal, and Dani Lischinski. Convolution pyramids. *ACM TOG*, 2011a.

Zeev Farbman, Raanan Fattal, and Dani Lischinski. Convolution pyramids. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 30(6), 2011b.

John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. URL `http://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Flynn_DeepStereo_Learning_to_CVPR_2016_paper.html`.

Alessandro Foi, Mejdi Trimeche, Vladimir Katkovnik, and Karen Egiazarian. Practical poissonian-gaussian noise modeling and fitting for single-image raw-data. *Image Processing, IEEE Transactions on*, 17(10):1737–1754, 2008.

William T. Freeman and Antonio Torralba. Shape recipes: Scene representations that refer to the image. In *Vision Sciences Society Annual Meeting*, pages 25–47. MIT Press, 2002.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

Pascal Getreuer. Color demosaicing with contour stencils. In *Digital Signal Processing (DSP), 2011 17th International Conference on*, pages 1–6. IEEE, 2011.

Michaël Gharbi, YiChang Shih, Gaurav Chaurasia, Jonathan Ragan-Kelley, Sylvain Paris, and Frédo Durand. Transform recipes for efficient cloud photo enhancement. *ACM TOG*, 2015.

Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. Deep joint demosaicking and denoising. *ACM TOG*, 2016.

Michaël Gharbi, Jiawen Chen, Jonathan T Barron, Samuel W Hasinoff, and Frédo Durand. Deep bilateral learning for real-time image enhancement. *ACM TOG*, 2017.

Jinwook Go, Kwanghoon Sohn, and Chulhee Lee. Interpolation using neural networks for digital still cameras. *Consumer Electronics, IEEE Transactions on*, 46(3): 610–616, 2000.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Andreas Griewank and Shawn Reese. On the calculation of Jacobian matrices by the Markowitz rule. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 126–135. 1991.

Bahadir K Gunturk, Yucel Altunbasak, and Russell M Mersereau. Color plane interpolation using alternating projections. *Image Processing, IEEE Transactions on*, 11(9):997–1013, 2002.

Eric Hamilton. Jpeg file interchange format. *C-Cube Microsystems*, 1992.

John F Hamilton Jr and James E Adams Jr. Apparatus for utilizing a digitized image signal, May 13 1997. US Patent 5,629,734.

Samuel W Hasinoff, Frédo Durand, and William T Freeman. Noise-optimal capture for high dynamic range photography. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 553–560. IEEE, 2010.

Samuel W Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T Barron, Florian Kainz, Jiawen Chen, and Marc Levoy. Burst photography for high dynamic range and low-light imaging on mobile cameras. *ACM TOG*, 2016.

Fang-Lin He, Yu-Chiang Frank Wang, and Kai-Lung Hua. Self-learning approach to color demosaicking via support vector regression. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 2765–2768. IEEE, 2012.

Kaiming He and Jian Sun. Fast guided filter. *CoRR*, 2015.

Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *TPAMI*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imageNet classification. *CoRR*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, and Pat Hanrahan. Darkroom: compiling high-level image processing code into hardware pipelines. *ACM TOG*, 2014.

Felix Heide, Markus Steinberger, Yun-Ta Tsai, Mushfiqur Rouf, Dawid Pajkak, Dikpal Reddy, Orazio Gallo, Jing Liu, Wolfgang Heidrich, Karen Egiazarian, et al. Flexisp: a flexible camera image processing framework. *ACM Transactions on Graphics (TOG)*, 33(6):231, 2014.

Keigo Hirakawa and Thomas W Parks. Adaptive homogeneity-directed demosaicing algorithm. *Image Processing, IEEE Transactions on*, 14(3):360–369, 2005.

Keigo Hirakawa and Thomas W Parks. Joint demosaicing and denoising. *Image Processing, IEEE Transactions on*, 15(8):2146–2157, 2006.

Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks, 2012. URL `http://doi.acm.org/10.1145/2307636.2307658`.

D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, Sept 1952. ISSN 0096-8390. doi: 10.1109/JRPROC.1952.273898.

Mark J Huiskes and Michael S Lew. The mir flickr retrieval evaluation. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pages 39–43. ACM, 2008.

Sung Ju Hwang, Ashish Kapoor, and Sing Bing Kang. Context-based automatic local image enhancement. *ECCV*, 2012.

Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)*, 35(4), 2016a.

Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM TOG*, 2016b.

Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CoRR*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, 2016.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

Vidit Jain and Erik Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010.

Varun Jampani, Martin Kiefel, and Peter V. Gehler. Learning sparse high dimensional filters: Image filtering, dense CRFs and bilateral neural networks. *CVPR*, 2016.

Gwanggil Jeon and Eric Dubois. Demosaicking of noisy bayer-sampled color images with least-squares luma-chroma demultiplexing and noise level estimation. *Image Processing, IEEE Transactions on*, 22(1):146–156, 2013.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3063-3. doi: 10.1145/2647868.2654889. URL `http://doi.acm.org/10.1145/2647868.2654889`.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

Oren Kapah and Hagit Z Hel-Or. Demosaicking using artificial neural networks. In *Electronic Imaging*, pages 112–120. International Society for Optics and Photonics, 2000.

Liad Kaufman, Dani Lischinski, and Michael Werman. Content-aware automatic photo enhancement. *Computer Graphics Forum*, 2012a.

Liad Kaufman, Dani Lischinski, and Michael Werman. Content-aware automatic photo enhancement. *Computer Graphics Forum*, 31(8):2528–2540, 2012b. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2012.03225.x. URL `http://dx.doi.org/10.1111/j.1467-8659.2012.03225.x`.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems*, pages 5580–5590, 2017.

Daniel Khashabi, Sebastian Nowozin, Jeremy Jancsary, and Andrew W Fitzgibbon. Joint demosaicing and denoising via learned nonparametric random fields. *Image Processing, IEEE Transactions on*, 23(12):4968–4981, 2014.

Jin-Hwan Kim, Won-Dong Jang, Jae-Young Sim, and Chang-Su Kim. Optimized contrast enhancement for real-time image and video dehazing. *J. Vis. Comun. Image Represent.*, 24(3):410–425, April 2013. ISSN 1047-3203. doi: 10.1016/j.jvcir.2013.02.004. URL `http://dx.doi.org/10.1016/j.jvcir.2013.02.004`.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Teresa Klatzer, Kerstin Hammernik, Patrick Knobelreiter, and Thomas Pock. Learning joint demosaicing and denoising based on sequential energy minimization. In *2016 IEEE International Conference on Computational Photography (ICCP)*, pages 1–11. IEEE, 2016.

Johannes Kopf, Michael F Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM TOG*, 2007.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *NIPS*, 2012.

Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mob. Netw. Appl.*, 18(1):129–140, February 2013. ISSN 1383-469X. doi: 10.1007/s11036-012-0368-0. URL `http://dx.doi.org/10.1007/s11036-012-0368-0`.

Cindy Kwan and Wu Xiaolin. A classification approach to color demosaicking. In *International Conference on Image Processing*, 2004.

Pierre-Yves Laffont, Zhile Ren, Xiaofeng Tao, Chao Qian, and James Hays. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM Transaction on Graphics (SIGGRAPH)*, 33(4):149:1–149:11, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601101. URL `http://doi.acm.org/10.1145/2601097.2601101`.

Steven Lansel and Brian Wandell. Local linear learned image processing pipeline. In *Imaging Systems and Applications*, page IMC3. Optical Society of America, 2011.

Claude A Laroche and Mark A Prescott. Apparatus and method for adaptively interpolating a full color image utilizing chrominance gradients, December 13 1994. US Patent 5,373,322.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. In *European Conference on Computer Vision (ECCV)*, 2016.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for cloud gaming. Technical Report MSR-TR-2014-115, 2014.

Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM Transaction on Graphics (SIGGRAPH)*, 23(3):689–694, August 2004. ISSN 0730-0301. doi: 10.1145/1015706.1015780. URL `http://doi.acm.org/10.1145/1015706.1015780`.

Anat Levin, Dani Lischinski, and Yair Weiss. A closed-form solution to natural image matting. *TPAMI*, 2008.

Anat Levin, Boaz Nadler, Fredo Durand, and William T Freeman. Patch complexity, finite pixel correlations and optimal denoising. In *Computer Vision–ECCV 2012*, pages 73–86. Springer, 2012.

Marc Levoy. Polygon-assisted jpeg and mpeg compression of synthetic images. In *Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 21–28, New York, NY, USA, 1995. ACM. ISBN 0-89791-701-4. doi: 10.1145/218380.218392. URL `http://doi.acm.org/10.1145/218380.218392`.

Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. Differentiable programming for image processing and deep learning in halide. *ACM TOG*, 2018.

Xin Li. Demosaicing by successive approximation. *Image Processing, IEEE Transactions on*, 14(3):370–379, 2005.

Xin Li, Bahadir Gunturk, and Lei Zhang. Image demosaicing: A systematic survey. In *Electronic Imaging 2008*, pages 68221J–68221J. International Society for Optics and Photonics, 2008.

Robert LiKamWa, Bodhi Priyantha, Matthai Philipose, Lin Zhong, and Paramvir Bahl. Energy characterization and optimization of image sensing toward continuous mobile vision. In *Proc. of International Conference on Mobile Systems, Applications, and Services*, pages 69–82. ACM, 2013.

Sifei Liu, Jinshan Pan, and Ming-Hsuan Yang. Learning recursive filters for low-level vision via a hybrid neural network. *ECCV*, 2016.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015a.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015b.

Yue M Lu, Mina Karzand, and Martin Vetterli. Demosaicking by alternating projections: theory and fast one-step implementation. *Image Processing, IEEE Transactions on*, 19(8):2085–2098, 2010.

Rafał Mantiuk and Hans-Peter Seidel. Modeling a generic tone-mapping operator. *Computer Graphics Forum (Proc. of Eurographics)*, 27(2), 2008.

Rafat Mantiuk, Kil Joong Kim, Allan G Rempel, and Wolfgang Heidrich. HDR-VDP-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions. In *ACM Transactions on Graphics (TOG)*, volume 30, page 40. ACM, 2011.

Ravi Teja Mullapudi, Andrew Adams, Dillon Sharlet, Jonathan Ragan-Kelley, and Kayvon Fatahalian. Automatically scheduling halide image processing pipelines. *ACM TOG*, 2016.

Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015. URL `http://arxiv.org/abs/1505.04366`.

Sylvain Paris and Frédo Durand. A fast approximation of the bilateral filter using a signal processing approach. *ECCV*, 2006.

Sylvain Paris, Samuel W. Hasinoff, and Jan Kautz. Local laplacian filters: Edge-aware image processing with a laplacian pyramid. In *ACM Transaction on Graphics (SIGGRAPH)*, SIGGRAPH '11, pages 68:1–68:12, New York, NY, USA, 2011a. ACM. ISBN 978-1-4503-0943-1. doi: 10.1145/1964921.1964963. URL `http://doi.acm.org/10.1145/1964921.1964963`.

Sylvain Paris, Samuel W Hasinoff, and Jan Kautz. Local laplacian filters: edge-aware image processing with a laplacian pyramid. *ACM TOG*, 2011b.

Sung Hee Park, Hyung Suk Kim, Steven Lansel, Manu Parmar, and Brian A Wandell. A case for denoising before demosaicking color filter array data. In *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, pages 860–864. IEEE, 2009.

Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context encoders: Feature learning by inpainting. 2016.

Thibaut Perol, Michaël Gharbi, and Marine Denolle. Convolutional neural network for earthquake detection and location. *Science Advances*, 4(2):e1700578, 2018.

Majid Rabbani and Paul W. Jones. *Digital Image Compression Techniques*. Society of Photo-Optical Instrumentation Engineers (SPIE), Bellingham, WA, USA, 1st edition, 1991. ISBN 0819406481.

Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *ACM TOG*, 2012a.

Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *ACM Transactions on Graphics*, 31(4): 32:1–32:12, July 2012b. ISSN 0730-0301. doi: 10.1145/2185520.2185528. URL `http://doi.acm.org/10.1145/2185520.2185528`.

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *SIGPLAN Not.*, 48(6):519–530, June 2013a. ISSN 0362-1340. doi: 10.1145/2499370.2462176. URL `http://doi.acm.org/10.1145/2499370.2462176`.

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI, pages 519–530, New York, NY, USA, 2013b. ACM. ISBN 978-1-4503-2014-6. doi: 10.1145/2491956.2462176. URL `http://doi.acm.org/10.1145/2491956.2462176`.

C. Rhemann, C. Rother, Jue Wang, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1826–1833, June 2009. doi: 10.1109/CVPR.2009.5206503.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015.

Tomasz Sergej and Radosław Mantiuk. Perceptual evaluation of demosaicing artefacts. In *Image Analysis and Recognition*, pages 38–45. Springer, 2014.

Xiaoyong Shen, Xin Tao, Hongyun Gao, Chao Zhou, and Jiaya Jia. Deep automatic portrait matting. *ECCV*, 2016.

Yichang Shih, Sylvain Paris, Frédo Durand, and William T. Freeman. Data-driven hallucination of different times of day from a single outdoor photo. *ACM Transaction on Graphics (SIGGRAPH)*, 32(6):200:1–200:11, November 2013a. ISSN 0730-0301. doi: 10.1145/2508363.2508419. URL `http://doi.acm.org/10.1145/2508363.2508419`.

Yichang Shih, Sylvain Paris, Frédo Durand, and William T Freeman. Data-driven hallucination of different times of day from a single outdoor photo. *ACM TOG*, 2013b.

YiChang Shih, Sylvain Paris, Connelly Barnes, William T. Freeman, and Frédo Durand. Style transfer for headshot portraits. *ACM Transaction on Graphics (SIGGRAPH)*, 33(4):148:1–148:14, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601137. URL `http://doi.acm.org/10.1145/2601097.2601137`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal Processing Magazine*, 18(5):36–58, 2001.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

Qiyuan Tian, Steven Lansel, Joyce E Farrell, and Brian A Wandell. Automating the design of image processing pipelines for novel color filter arrays: Local, linear, learned (l3) method. In *IS&T/SPIE Electronic Imaging*, pages 90230K–90230K. International Society for Optics and Photonics, 2014.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. *ICCV*, 1998.

A. Torralba and W.T. Freeman. Properties and applications of shape recipes. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–383–90 vol.2, June 2003. doi: 10.1109/CVPR.2003.1211494.

G.K. Wallace. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, Feb 1992. ISSN 0098-3063. doi: 10.1109/30.125072.

Xiaolong Wang, David Fouhey, and Abhinav Gupta. Designing deep networks for surface normal estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 539–547, 2015.

T.A. Welch. A technique for high-performance data compression. *Computer*, 17(6): 8–19, June 1984. ISSN 0018-9162. doi: 10.1109/MC.1984.1659158.

Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. Image smoothing via l0 gradient minimization. *ACM Transaction on Graphics (SIGGRAPH)*, 30(6):174:1–174:12, December 2011. ISSN 0730-0301. doi: 10.1145/2070781.2024208. URL `http://doi.acm.org/10.1145/2070781.2024208`.

Li Xu, Jimmy SJ Ren, Ce Liu, and Jiaya Jia. Deep convolutional neural network for image deconvolution. In *Advances in Neural Information Processing Systems*, pages 1790–1798, 2014.

Li Xu, Jimmy Ren, Qiong Yan, Renjie Liao, and Jiaya Jia. Deep edge-aware filters. *ICML*, 2015.

Zhicheng Yan, Hao Zhang, Baoyuan Wang, Sylvain Paris, and Yizhou Yu. Automatic photo adjustment using deep neural networks. *ACM TOG*, 2016.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, 2015.

Lu Yuan and Jian Sun. High quality image reconstruction from raw and jpeg image pair. *ICCV*, 2011.

Fan Zhang, Xiaolin Wu, Xiaokang Yang, Wenjun Zhang, and Lei Zhang. Robust color demosaicking with adaptation to varying spectral correlations. *Image Processing, IEEE Transactions on*, 18(12):2706–2717, 2009.

Lei Zhang and Xiaolin Wu. Color demosaicking via directional linear minimum mean square-error estimation. *Image Processing, IEEE Transactions on*, 14(12):2167–2178, 2005.

Lei Zhang, Xiaolin Wu, Antoni Buades, and Xin Li. Color demosaicking by local directional interpolation and nonlocal adaptive thresholding. *Journal of Electronic Imaging*, 20(2):023016–023016, 2011.

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. *ECCV*, 2016.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep networks as a perceptual metric. In *CVPR*, 2018.

J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977. ISSN 0018-9448. doi: 10.1109/TIT.1977.1055714.