Transform recipes for efficient cloud photo enhancement

Citation: Michael Gharbi, YiChang Shih, Gaurav Chaurasia, Jonathan Ragan-Kelley, Sylvain Paris, and Fredo Durand. 2015. Transform recipes for efficient cloud photo enhancement. ACM Trans. Graph. 34, 6, Article 228 (October 2015), 12 pages.

As Published: http://dx.doi.org/10.1145/2816795.2818127

Publisher: Association for Computing Machinery (ACM)

Persistent URL: http://hdl.handle.net/1721.1/99939

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Transform Recipes for Efficient Cloud Photo Enhancement

Michaël Gharbi MIT CSAIL
YiChang Shih MIT CSAIL
Gaurav Chaurasia MIT CSAIL
Jonathan Ragan-Kelley Stanford
Sylvain Paris Adobe
Frédo Durand MIT CSAIL

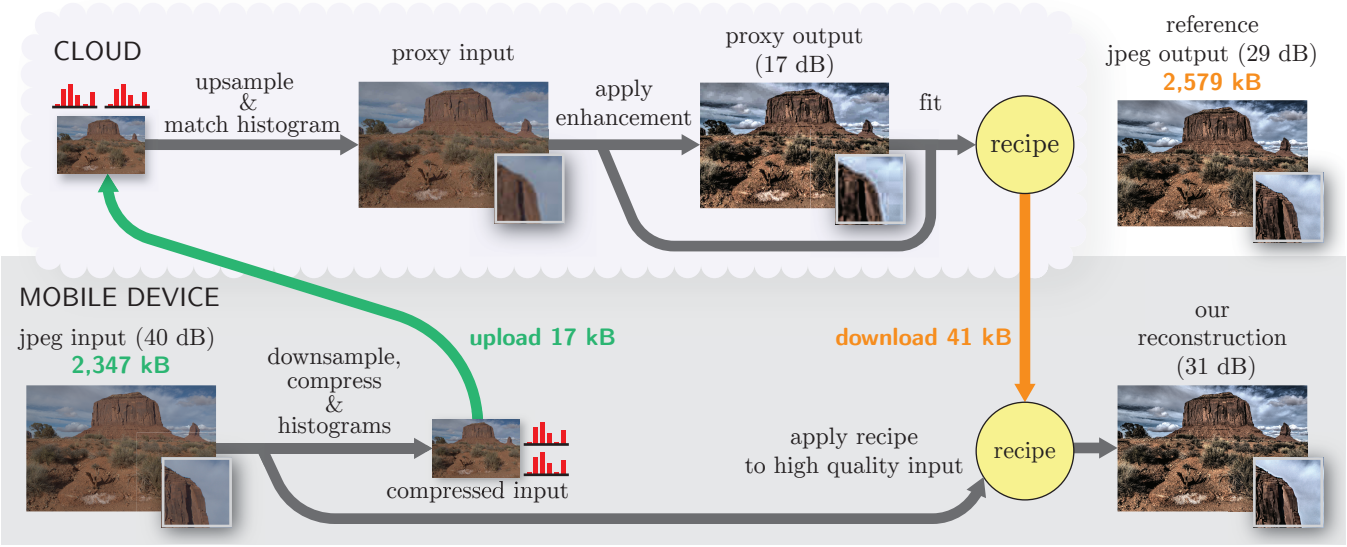


Figure 1: *Cloud computing is often thought as an ideal solution to enable complex algorithms on mobile devices; by exploiting remote resources, one expects to reduce running time and energy consumption. However, this ignores the cost of transferring data over the network, the overhead of which often negates the benefits of the cloud, especially for data-heavy image processing applications. We introduce a new image 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 photograph that is compact and can be accurately estimated from an aggressively compressed input photograph. 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 an aggressive setting, our approach reduces the amount of transmitted data up to* 80× *compared to standard JPEG compressed images. It produces an output visually similar to the ground-truth result computed directly from the original uncompressed photograph. Photograph from the MIT 5k dataset [2011].*

Abstract

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. We aim 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 the utility of recipes on a mobile phone by profiling the energy consumption and latency for both local and cloud computation: a transform recipe-based pipeline runs 2-4× faster and uses 2-7× less energy than local or naive cloud computation.

CR Categories: I.4.2 [Image Processing]: Compression; I.4.10 [Image Processing]: Applications; I.4.10 [Image Processing]: Image Representation;

Keywords: mobile image processing, energy-efficient cloud computing, image filter approximation

1 Introduction

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 power supply of remote servers. Cloud processing is also impera-

tive for algorithms that require large databases, e.g. [Laffont et al. 2014; Shih et al. 2013], and can simplify application development, especially in the face of the mobile platform fragmentation. Unfortunately, the cost of transmitting the input and output images can exceed 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-second computation on a 16-megapixel image captured by a Samsung Galaxy S5 consumes 20J. Processing the same image in the cloud would take 14 seconds and consume 54J due to the network communication overhead[1]. Efficient cloud photo enhancement requires a reduction of the data transferred that is beyond what lossy image compression can achieve while keeping 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. 2012], dehazing [Kim et al. 2013], edge-preserving enhancements [Paris et al. 2011; Farbman et al. 2008], colorization [Levin et al. 2004], style transfer [Shih et al. 2014; Aubry et al. 2014], time-of-day hallucination [Shih et al. 2013] 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 *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 data transfers. With our method, the mobile client uploads a downsampled and highly compressed 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 turn, the client combines the recipe with the original high quality input to reconstruct the output (Fig. 1). While our recipes are lossy, they provide high-fidelity approximations for full image enhancement, while being 2 to 3 orders of magnitude more compact than the raw bitmap data and 1 to 2 orders of magnitude smaller than JPEG-compressed images.

Transform recipes use a combination of local affine transformations of the YUV channels of a shallow Laplacian stack as well as local tone curves. This makes them flexible and expressive, so as to adapt to a variety of enhancements, and impose only 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 our results on 168 images spanning 10 image processing applications 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.

2 Related Work

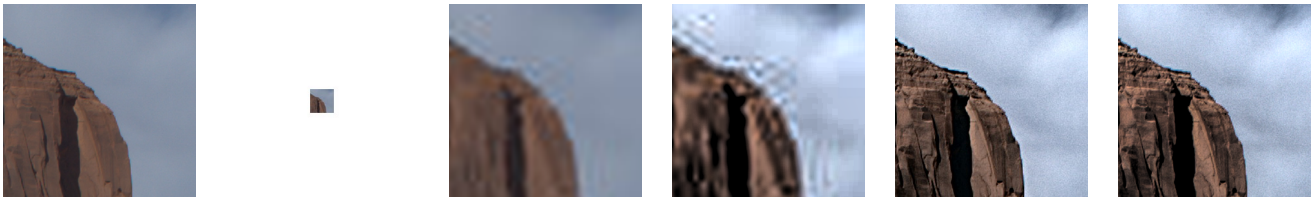
Image compression Image compression has been a long standing problem in image processing [Rabbani and Jones 1991]. The data used to represent digital images is reduced by exploiting their structure and redundancy. Lossless image compression methods and formats rely on general purpose compression techniques 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. [2011] and Berthouzoz et al. [2011] use a generic representation of photo-

[1] Transferring the JPEG images with their default compression settings (4MB each way), over a 4G network at $4\text{Mbits}/s$, according to our measurement of the average power draw and modeling like in [Kumar et al. 2013]



(a) *original full-quality input on the client*

(b) *downsampled and jpeg-compressed input received by the server along with histograms of (a). The data uploaded is around* 0.1% *of the raw bitmap image.*

(c) *proxy input built on the server by upsampling (b) and matching histograms from (a).*

(d) *proxy output obtained by applying the filter to the proxy input (c)*

(e) *reconstruction on the client using the recipe and (a). Downloading the recipe costs* 0.1% *of the raw bitmap data, or around* 1% *of the original JPEG.* PSNR $= 31.4$dB

(f) *reference output computed from the original full-quality input (a)*

Figure 2: *A major advantage of our transform recipes is that we can compute them from a severely degraded input-output pair. (a) shows a crop from the high quality input of Fig. 1 on the client. The server receives a downsampled and compressed version of this image (b) along with histograms from (a). It upsamples it and matches the histograms (c). Processing such a strongly compressed image generates a result with numerous artifacts (d). We can nevertheless use this pair of proxy images to build a recipe that, when applied on the original artifact-free photo (a) produces a high quality output (e) that closely approximates the reference output (f) directly computed from the original photo. These close-ups show the region with the highest error for our method.*

graphic 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. [2011] introduce *convolution pyramids* as a means to accelerate linear translation-invariant image filters. 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 Reducing Data Transfers

We process an input image I with a filter f to produce an output image $O = f(I)$. We are interested in the regime where computing f is costly enough to justify using a remote server. As we previously discussed, network communications introduce a significant overhead in this case, more so when the bandwidth is low. We alleviate this issue by reducing data transfer.

Overview To lower the upstream cost, we generate a low-resolution jpeg-compressed version $\bar{I}$ of the input image I on the client. Then we upload this image $\bar{I}$ to the server along with the histogram of the original input I (described in Section 4.1). Upon reception, the server upsamples $\bar{I}$ back to the original resolution and applies histogram transfer to compute a proxy input image $\tilde{I}$. It processes $\tilde{I}$ to generate a proxy output $\tilde{O} = f(\tilde{I})$. The server then computes a compact recipe r using $\tilde{I}$ and $\tilde{O}$ that describes the *transformation* from one to the other, i.e. $r(\tilde{I}) \approx f(\tilde{I})$. We want the recipe r to remain valid in a neighborhood of the proxy pair and compute a high quality approximation of the reference output when applied to the original input, i.e., $r(I) \approx f(I)$. The final step is for the client to download the recipe and apply it on the original input I. Figure 1 illustrates our pipeline.

Our objective with transform recipes is to represent a variety of standard photo enhancements that may be spatially varying, multi-scale, and nonlinear. Each recipe is specific to a given input/filter pair. Instead of computing a generic approximation of a particular filter f that is valid for all inputs, we compute an input driven transform recipe rec that can approximate a number of different image processing filters. In the rest of this section we describe both how we decompose images, and how we compute a low-order model that captures the effect of f on I in this representation: the recipe.

3.1 Transform Recipes

We first describe how to build a recipe r from any input I and the corresponding output O. We experimentally found that (a) multi-scale effects play an important role, and (b) recipes only need to coarsely model the chrominance transformation while a more sophisticated representation of the luminance transformation is desirable. Following these observations, our construction uses a multi-scale decomposition of the images and luminance–chrominance separation. This allows our recipes to adapt the granularity at which the image transformation is represented for each component.

Spatially adaptive The effects of image filters may be complex at the scale of an entire image but are often simpler at a local scale, because image data itself is simpler. The power of our approach comes from the fact that we fit recipes in a spatially-varying manner on local image blocks. This allows us to capture complex effects even though our image representation and transforms are both relatively simple. We define recipes over $w \times w$ blocks where w controls the trade-off between compactness and accuracy. Small blocks give a fine-grained, accurate description of the transformation but require more storage, whereas large blocks give a coarser but more concise description. 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, and we linearly interpolate pixel values weighted by the distance to the center of the block to prevent blocking artifacts.

Image decomposition We work in $YC_\mathrm{b}C_\mathrm{r}$ color space like JPEG, to separate luminance Y from chrominance $(C_\mathrm{b}, C_\mathrm{r})$ while still operating on values that span $[0, 255]$ [Hamilton 1992]. To capture multi-scale effects, we decompose an image into a Laplacian stack – a multi-scale image decomposition similar to a Laplacian pyramid, but with all levels stored at the original resolution instead of subsampling which makes it easier to fit recipes. To compute a stack, we first build a Laplacian pyramid [Burt and Adelson 1983]: we split I

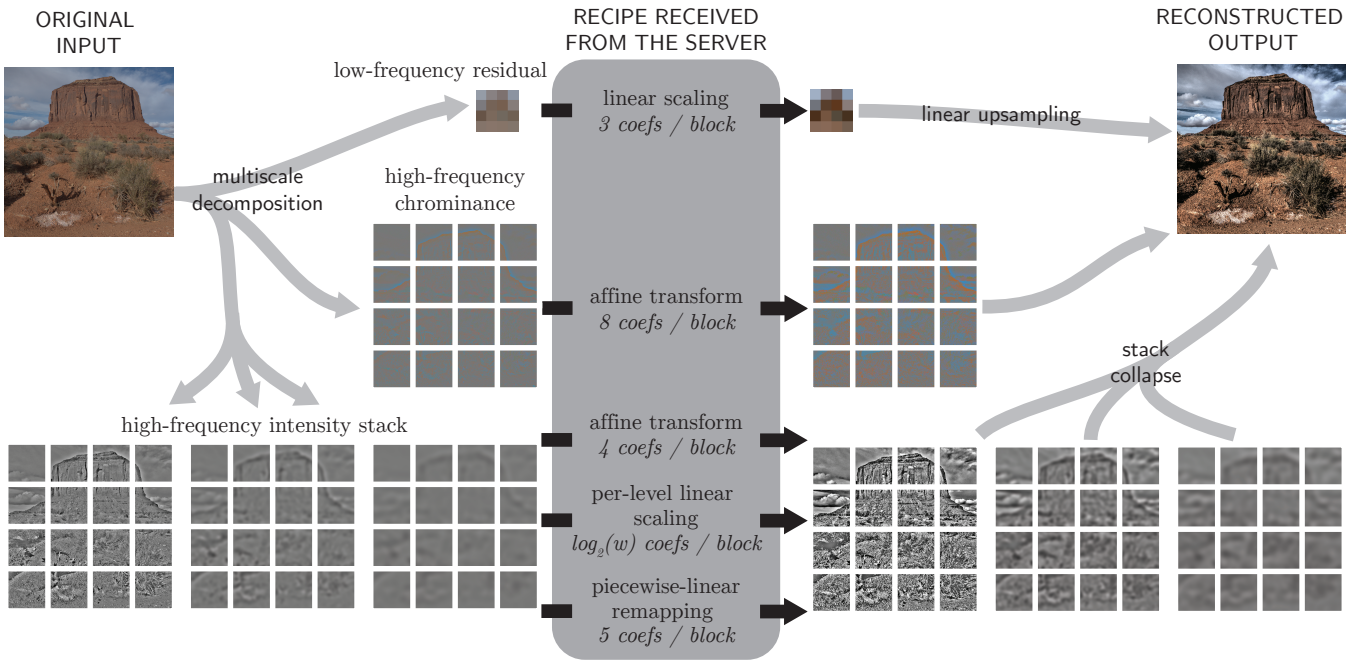


Figure 3: *To reconstruct the output image we decompose the input as a shallow Laplacian stack. We linearly rescale the lowpass residual using the ratio coefficients R_c. The high-frequency chrominance undergoes an affine transformation parameterized by $\mathbf{A}_c$ and $\mathbf{b}_c$. We reconstruct the luminance channel using a combination of an affine transformation ($\mathbf{A}_Y$, $\mathbf{b}_Y$), a linear scaling of the Laplacian stack levels ($\{m_\ell\}$), and a piecewise linear transformation ($\{q_i\}$). Photograph from the MIT 5k dataset [2011].*

and O into $n+1$ levels where the resolution of each level is half that of the preceding level in each dimension. The first n levels represent the details at increasingly coarser scales, and the last level is the low frequency residual. We set $n = \log_2(w)$ so that each block is represented by a single pixel in the residual. Finally, we upsample all the levels back to the original resolution (with the exception of the residual that we keep unchanged) to produce a Laplacian stack. We name the input and output stack levels $\{L_\ell[I]\}$ and $\{L_\ell[O]\}$ respectively.

With this decomposition, we compute a recipe in three steps: we first represent the transformation of the low-pass residual, then the transformations of the other stack levels, and finally quantize and encode these transformations in an off-the-shelf compressed image format.

3.1.1 Representing the transformation of the residual

Each pixel of the residual affects a large area in the final reconstruction and even a small error can produce conspicuous artifacts in smooth regions like sky. Since the residual is only a small fraction of the data, we represent its transformation 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 $c \in \{Y, C_\mathrm{b}, C_\mathrm{r}\}$:

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

We add 1 to prevent division by zero and ensure that R_c is constant when $L_n[O_c](p) = L_n[I_c](p)$ for all p, e.g., for edits like sharpening that only modify the high frequencies. This allows better compression in the final stage. This part of the recipe is illustrated at the top of Figure 3.

3.1.2 Representing the transformation of the stack levels

Each stack level has as many pixels as the original image so we cannot afford a per-pixel representation as with the residual. Instead, we represent the transformation within each block as a function parameterized by a small number of parameters. Further, when possible, we work on the combined high-frequency data:

$$H[I] = \sum_{\ell=0}^{n-1} L_\ell[I] \tag{2}$$

instead of using the individual levels $\{L_\ell[I]\}$.

We experimentally found that chrominance transformations are simple enough to be well approximated by affine functions, but that no single simple parametric function captures the diversity of luminance transformations generated by common photographic enhancements. Instead, we rely on a combination of multiple functions and use a sparse regression to retain only a few parameters in the final representation.

Chrominance channels Let $O_{CC}(p)$ be the 2D vector containing the chrominance values of the output 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 standard least-squares regression and minimize:

$$\sum_{p \in \mathcal{B}} \| H[O_{CC}](p) - \mathbf{A}_\mathrm{c}(\mathcal{B}) H[I](p) - \mathbf{b}_\mathrm{c}(\mathcal{B}) \|^2 \tag{3}$$

where $\mathbf{A}_\mathrm{c}$ and $\mathbf{b}_\mathrm{c}$ are the 2×3 matrix and the 2D vector defining the affine model. This part of the recipe is shown in the middle row of Figure 3.

Luminance channel We represent the high frequencies of the luminance $H[O_Y]$ with a more sophisticated function consisting of

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. This affine component captures the local changes of brightness and contrast. Next, we apply a per-block multiplicative factor $m_\ell(\mathcal{B})$ to each stack level $L_\ell[I_Y]$. These multiplicative factors collectively model multi-scale effects. Last, we add a term to account for non-linearities. We use a piecewise linear function made of k segments over the luminance range of the block. To define this function, we introduce $k-1$ regularly spaced luminance values $y_i = \min_{\mathcal{B}} H[I_Y] + \frac{i}{k}(\max_{\mathcal{B}} H[I_Y] - \min_{\mathcal{B}} H[I_Y])$, $i \in \{1, \ldots, k-1\}$ 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 creates a unit change of slope at the i^{th} node y_i. We use these functions to define our nonlinear term as $\sum_{i=1}^{k-1} q_i(\mathcal{B}) s_i(H[I_Y])$ 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 that combines the three components that we just discussed and sum over all the pixels of the block $\mathcal{B}$:

$$\sum_{p \in \mathcal{B}} \Big\| H[O_Y](p) - \mathbf{A}_Y(\mathcal{B})H[I](p) - b_Y(\mathcal{B}) - \sum_{\ell=0}^{n-1} m_\ell(\mathcal{B}) L_\ell[I_Y](p) - \sum_{i=1}^{k-1} q_i(\mathcal{B}) s_i(H[I_Y](p)) \Big\|^2 \quad (4)$$

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 $\{q_i(\mathcal{B})\}$. Compared to a L_2-regularized regression, the LASSO has the benefit of creating sparse coefficient maps that allow more compact encoding, reducing the size of the recipe by up to 30%. Both methods otherwise entail almost equal computation time and are visually indistinguishable.

Solving the optimization problems (Eq. 3 and 4) for each overlapping block, and computing the low-pass ratio (Eq. 1) gives all the recipe coefficients (Fig. 4). Our local per-patch optimization yields results visually identical to a global optimization strategy, while being several orders of magnitude faster.

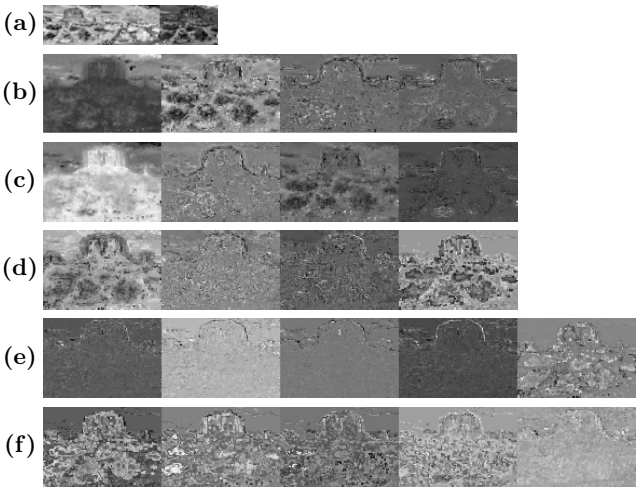


Figure 4: *Recipe coefficients computed for the photo in Figure 2. We remapped the values to the* $[0; 1]$ *range for display purposes.* (a) *lowpass residual* R_c. (b,c) *affine coefficients of the chrominance* $\mathbf{A}_c$ *and* $\mathbf{b}_c$. (d) *affine coefficients of the luminance* $\mathbf{A}_Y$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

3.1.3 Encoding recipe data

Once we have the recipe coefficients described in Section 3.1, we pack them as a multi-channel image (Fig.4). We quantize each channel of the recipe to 8 bits. In practice, this error is small and does not affect the quality of the reconstruction (< 0.1 dB).

We compress these coefficient maps using off-the-shelf lossless image compression methods. We save the lowpass residual as a 16-bit float TIFF file, and the highpass coefficients as tiles in a 8-bit grayscale PNG file. This compression exploits both the spatial redundancy of the recipe and the sparsity induced by the LASSO regression and further reduces the data by more than a factor 2.

3.1.4 Reconstructing on the client

We reconstruct the output O on the client by applying the recipe received from the server and described in the previous section on the client's high quality input I. We proceed in two steps: we first perform the same image decomposition of the original input, and then apply the corresponding recipe coefficients to each term of this decomposition as illustrated in Figure 3.

For each pixel p and each channel $c \in \{Y, C_\text{b}, C_\text{r}\}$, we obtain the output low-pass residual by applying the ratios R from Equation 1 onto the input residual:

$$L_n[O_c](p) = R_c(p)\big(L_n[I_c](p) + 1\big) - 1 \quad (5)$$

We reconstruct the intensity levels $\ell \in [0; n-1]$ of the multiscale component by multiplying them by the factors m_ℓ from Equation 4:

$$L_\ell[\hat{O}_Y] = m_\ell L_\ell[I_Y] \quad (6)$$

The above gives the complete Laplacian stack $\{L_\ell[\hat{O}_Y]\}$ together with the luminance channel that we upsample back to the original resolution (Equation 5). We collapse this stack to get an intermediate output luminance channel $\hat{O}_Y$. At this point, the reconstruction is missing the high-frequency luminance components from Equation 4 and the chrominance channels C_r and C_b.

We reconstruct the remaining high-frequency information $H[O]$ by first computing $H_\mathcal{B}[O]$ within each block then linearly blending the results of overlapping blocks. For chrominance, we apply the affine remapping that we estimated in Equation 3:

$$H_\mathcal{B}[O_{CC}](p) = \mathbf{A}_\text{c}(\mathcal{B})H[I](p) + \mathbf{b}_\text{c}(\mathcal{B}) \quad (7)$$

For luminance, we apply the affine remapping and the nonlinear functions that we computed in Equation 4:

$$H_\mathcal{B}[O_Y](p) = \mathbf{A}_Y(\mathcal{B})H[I](p) + \mathbf{b}_Y(\mathcal{B}) + \sum_{i=1}^{k-1} q_i(\mathcal{B}) s_i(H[I_Y](p)) \quad (8)$$

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 combine all the terms to assemble the reconstructed output. We linearly upsample the chrominance residual (Eq. 5), add the multiscale component (Eq. 6), and the high-frequency chrominance (Eq. 7) and luminance (Eq. 8) linearly interpolated between overlapping blocks:

$$O = \text{up}(L_n[O_{CC}]) + \hat{O}_Y + H[O_{CC}] + H[O_Y] \quad (9)$$

where $\text{up}(\cdot)$ is the operator that upsamples the residual back to the original resolution and we implicitly lift the quantities in the right-hand side to $YC_\text{b}C_\text{r}$ vectors depending on the channels that they contain.

3.2 Compressing the input image

We now describe the algorithm to compress the input data sent to the server. In our approach, the server's purpose is to compute a recipe and not the final output. We therefore send a modified input to the server as long as the computed recipe remains faithful. We exploit this by aggressively compressing the input image sent to the server. We use the image received on the server to create a proxy input $\tilde{I}$ that we process with the filter f to get a proxy output $\tilde{O} = f(\tilde{I})$. These two images are then used to construct the recipe as previously described.

To compress the input image, we first downsample it (typically by a factor of 4 in each dimension) followed by aggressive JPEG compression. The compressed image represents around 1% of the original uncompressed image. This minimizes the latency and energy consumption due to the upload.

The visual quality of this image is poor but from a signal processing perspective, the medium and low frequencies of the input image are only lightly impacted by the downsampling and compression. The quality degradation is concentrated in the high frequencies. We empirically observed two main issues. First, some fine details completely disappear and leave a nearly constant region. In some blocks, this results in constant features for the regression described in Equation 4. The optimization might use this constant term as an affine offset. This issue becomes apparent when the recipe is later applied to the original image whose corresponding terms are non-constant (Fig. 5). Second, high-frequency artifacts appear and edges get softened, which degrade the accuracy of the estimated recipe and the final reconstructed image. We alleviate these issues by upsampling the image back to the original resolution when received by the server and reshaping the histograms of its 3 highest-frequency bands so that they match those of the original input, as described by Heeger and Bergen [1995]. In practice, we compute a 3-level Laplacian pyramid, add low-amplitude Gaussian noise ($\sigma = 0.5\%$ of the intensity range) to avoid constant regions, and apply histogram transfer to each level to match the distribution of coefficients of the original input image. This process adds a minimal computational overhead: 3 histogram computations on the client and 3 histogram transfers and a 3-level pyramid on the server. It only requires the transmission of 3 ZIP-compressed coarsely sampled histograms in addition to the small compressed image.

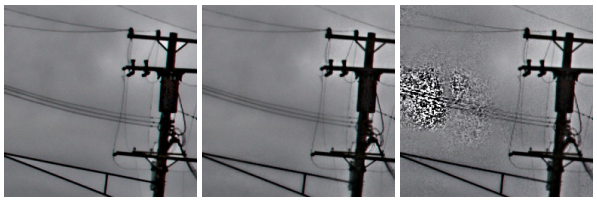
The table below reports the PSNR with and without this preprocessing. It shows that the two components of our proxy construction, upsampling and high-frequency shaping, contribute to the quality of the final result. Further details are provided in supplemental material.

	freq. shaping	no freq. shaping
upsampling	**35.8 dB**	33.5 dB
no upsampling	28.7 dB	29.6 dB

Figure 6 compares our approach to directly using the low resolution compressed image. The latter option faces two challenges, as we previously discussed, the high-frequency signal is highly degraded, and some filters are not scale-invariant, i.e., their output on a downsampled image is not a faithful approximation of the full-resolution result [Jeong et al. 2011].

4 Evaluation

We evaluated the quality of our reconstruction on 10 different image processing applications (§ 4.1, summarized in Fig. 11). We studied the effect of degrading the input to different degrees, and analyzed the impact of different components of our recipe model (§ 4.2). Finally, we measured the real-world latency and physical energy consumption of our approach using a prototype implementation with a PC server and an Android smartphone (§ 4.3). Systematic results for different quality settings and the impact of each feature in the transform recipe pipeline are provided in the supplemental material.



(a) *reference* **(b)** *our method,* PSNR = 34.7 dB **(c)** *no noise added to the upsampled input before processing,* PSNR = 28.1 dB

Figure 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 6: *In this close-up from a* Detail Manipulation *example, processing directly the downsampled input before fitting the recipe fails to capture the higher frequencies of the transformation. Errors are particularly visible in the eyes and hair. Image, MIT 5k [2011].*

Throughout our analysis, we express the compression factor as a fraction of the original uncompressed 24-bit bitmap. Though an uncompressed bitmap is never used in the context of cloud image processing, it provides a fixed quality reference for all of our comparisons. For natural color images, the typical out-of-the-phone data ratios for JPEG and PNG are $10-15\%$ $50-75\%$ respectively. Besides visual comparison, we report both PSNR and SSIM to quantify the quality of our reconstruction.

4.1 Test applications

We selected a range of applications representative of typical photo editing scenarios to evaluate the expressiveness of our approach in capturing different visual effects. Some of these applications require a large database, and hence cloud processing, while others could be performed locally but are still computationally intensive We gathered 168 high-quality images from Flickr and the MIT-Adobe fiveK dataset [Bychkovsky et al. 2011]. Their resolutions range from 2 to 8 megapixels. The MIT-Adobe fiveK photos are in raw format and provide a baseline to evaluate reconstruction quality.

	input subsampling	input Q	w	$\%_{up}$	$\%_{down}$ ours	$\%_{down}$ jpeg	$\%_{down}$ jdiff	PSNR (dB) ours	PSNR (dB) jpeg	PSNR (dB) jdiff	SSIM ours	SSIM jpeg	SSIM jdiff
	—	—	32	55.6	1.4	7.9	7.2	41.0	41.2	**41.5**	**0.98**	0.97	0.96
standard	2×	30	32	1.0	1.5	1.9	2.0	**38.5**	30.2	30.6	**0.97**	0.82	0.83
medium	4×	50	64	0.4	0.5	0.6	0.6	**35.8**	27.3	27.6	**0.96**	0.75	0.75
low	8×	80	128	0.2	0.1	0.2	0.2	**32.7**	24.9	25.1	**0.94**	0.69	0.69

Table 1: *Mean compression factor and reconstruction quality for our ten applications test suite. (The full table of per-application results at all four quality levels is included as supplemental material, with the input and output images for every tested configuration.)* $\%_{up}$ *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 a losslessly compressed input, our method matches the quality of a JPEG-compressed output. We provide this setting as a reference only and do not recommend using it in practice. The benefits of transform recipes become more dramatic as the input image is compressed. Q is the quality setting for the JPEG compression of the input. We only recommend the "low" setting when data size is paramount, because the degradation of the results can be significant in some cases. The* jpeg *and* jdiff *methods are given the same input, and use $Q = 80$ for the output compression. Since* PSNR *and* SSIM *don't necessarily tell the full story, we refer to the supplemental material for a more in-depth visual comparison.*

From these high-quality inputs, we generated proxy inputs for several quality levels using a combination of bicubic downsampling (2 to 8 times smaller in each dimension) and JPEG compression (with quality parameters between 30 and 80). We use JPEG quantization tables from Adobe Photoshop. This new set of images are what the server processes in our technique.

We ran each algorithm on this set of inputs for different quality settings and measured end-to-end approximation fidelity.

Our 10 tested applications were:

Photo editing and Photoshop actions We constructed a complex Photoshop Action including tonal adjustments, color correction, non-trivial masking, spatially varying blurs, unsharp masking, high-pass boost, vignetting, white balance adjustments, and detail enhancement. These operations include both global and local edits, and were created as a stress test for our model. This is also an example of using existing code which is not available for a mobile device, but can be run remotely on a server.

Dehazing This algorithm [Kim et al. 2013] estimates an atmospheric light vector from the input image and removes the corresponding haze. The transformation applied to the input image exhibits sharp transitions often colocated with edges in the input image.

Edge preserving detail enhancement We test both local Laplacian filtering [Paris et al. 2011] and WLS filtering [Farbman et al. 2008]. These methods are challenging for our approach because the recipes are not explicitly edge-aware.

Style Transfer This algorithm [Aubry et al. 2014] alters the distribution of gradients in an image to match a target distribution using a variant of local Laplacian filtering. The algorithm outputs an image whose style matches that of the example image.

Portrait Style Transfer Shih et al. [2014] spatially matches a casual portrait to a target example automatically found in a database; at its core, this technique relies on computationally expensive dense correspondence. Because of correspondence estimation and database dependence, this algorithm is a good candidate for cloud processing. The algorithm also copies the eye highlights from the example as a post-process; we disabled this step because it is not supported by our approach.

Time of Day Hallucination This technique by Shih et al. [2013] requires a large database and a costly computer vision analysis.

Colorization Levin et al. [2004] requires solving a large sparse linear system that is costly on mobile platforms for large images. We manually created the scribbles on images from the MIT-5K dataset.

Matting This application lies at the fringe of the scope of our work since the output image does not resemble a photograph. It is nonetheless useful in the photo editing context and is a good stress case for expressiveness. We use the implementation of KNN-matting available on the authors' webpage [Chen et al. 2012], and use the publicly available data from Rhemann et al [2009] as inputs.

L_0 smoothing Xu et al. [2011] aggressively remove details and texture from photographs for stylization. We include it as a failure case.

4.2 Reconstruction quality

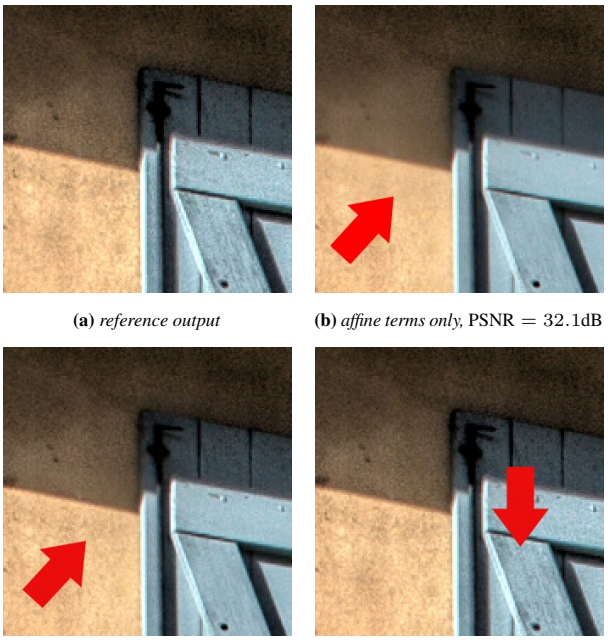
We processed our entire dataset at various levels of compression and found that our approach produces outputs that are visually similar to the ground truth, i.e., the direct processing of the uncompressed input, in all cases except with the most extreme settings and the L_0 smoothing filter. Figure 11 shows a representative example reconstruction for each application for the quality setting that we consider "standard."

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 processes it and sends back the difference encoded in JPEG. The client adds the difference back to its local input.

As shown in Table 1, our approach consistently outperforms these traditional alternatives by a significant margin. *jpeg* and *jdiff* match the quality of our technique only when using full-resolution non-degraded input and much less compression of the output. A complete quality study, per image filter can be found in the supplemental material. In general, with our "standard" setting, our method requires just 1-2% of the total input/output data (5-10% of the baseline JPEG solution) to reproduce the reference output to 35 dB and often over 40 dB across all test images for 9 of our 10 test applications. With the more aggressive "medium" setting, it requires just 0.5% of the data while still achieving a high fidelity of 33-40 dB.

Robustness to input degradation Figure 8 illustrates the robustness of our recipes for various compression ratios of the input



(a) *reference output* **(b)** *affine terms only,* PSNR = 32.1dB

(c) *affine and multiscale terms for the luminance,* PSNR = 33.6dB **(d)** *affine, multiscale and non-linear terms for the luminance,* PSNR = 36.4dB

Figure 7: *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 details are more faithfully captured on the wall. (d) Our recipe: with both the non-linear and multiscale terms, our preserves local contrast variations. This is particularly visible on the texture of the wooden shutter (best appreciated digitally). Image, MIT 5k dataset [2011].*

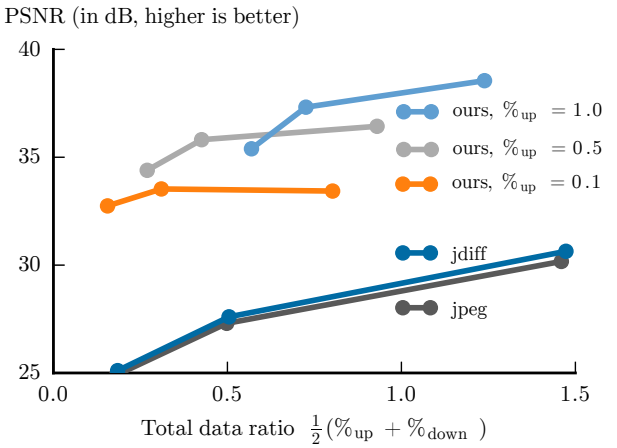


Figure 8: *The quality of our reconstruction remains high for different settings of our recipe and various degrees of input compression. Not surprisingly, the reconstruction quality decreases as input and output compression increase. We report the average PSNR on the whole dataset. In comparison, both the* jpeg *and* jdiff *versions perform poorly under low data allowance; they require an order of magnitude more data to match our quality.*

and output. We recommend two levels of compression: "standard," which corresponds to about 1.5% of the total data (upstream and downstream) and generates images often indistinguishable from the ground truth, and "medium," which corresponds to about 0.5% of the data and only introduces minor deviations. In practice, one would choose the level of accuracy/compression depending on the application.

Impact of additional luminance terms The extra features for the luminance channel are critical to the expressiveness of our model. Both the nonlinear luminance curve and the multiscale features help capture subtle and local effects (Fig. 7). We summarize these improvements in the table below for an uncompressed input; a detailed report per category can be found in Table 2.

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

4.3 Runtime performance & energy efficiency

We implemented a client-server system to evaluate the real-world runtime and energy consumption. We used a Samsung Galaxy S5 as the client device. We tested three image processing scenarios: on-device processing, naive cloud processing – where we transferred the input and output compressed as JPEG with the default settings of the cellphone, and our recipe-based processing. We measured network connections through WiFi and LTE. Our server is a MacBook Pro running an Intel Core i7 2.7GHz with 4 cores. For on-device processing, we used a Halide-optimized implementation [Ragan-Kelley et al. 2012; Ragan-Kelley et al. 2013] of the test filters, through the Java Native Interface on Android.

We implemented and experimented with 5 filters on this setup: Local Laplacian, Colorization, Style Transfer, Portrait Transfer and Time of Day hallucination. We ran Local Laplacian filtering [Paris et al. 2011] with 40 discretization pyramids levels, which enables fine-scale detail enhancement, on an 8-megapixel input. For Style Transfer, we used 3 iterations and 20 discretization levels, as the authors recommend [Aubry et al. 2014]. We used the default parameters from the authors' implementation for the remaining filters. Portrait Transfer and Time of Day require a large database; we did not implement them on a mobile device as they need to run on the server.

To ensure accurate power measurement, we replaced the cellphone battery with a power generator, and recorded the current and voltage drawn from the generator. The power usage plot in Figure 9 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 as well as power usage (Fig. 9).

For typical mobile network bandwidth (2Mbps upstream, 8 Mbps downstream), our approach ran $2\times$ faster that the *jpeg* option in most cases and up to $4\times$ faster than local processing. Our energy consumption is between $2-7\times$ lower than local processing, and between $2-3\times$ lower than transferring JPEG images (Fig. 12). Our advantage becomes more significant as the network connection gets slower and less reliable. The standard *jpeg* cloud becomes unpractical because of drastically increasing processing time and energy consumption (Fig. 10). Thanks to low data transfer, our method still provides gains over local processing in such situations and makes cloud image processing possible in the face of network scarcity for applications that cannot be run locally.

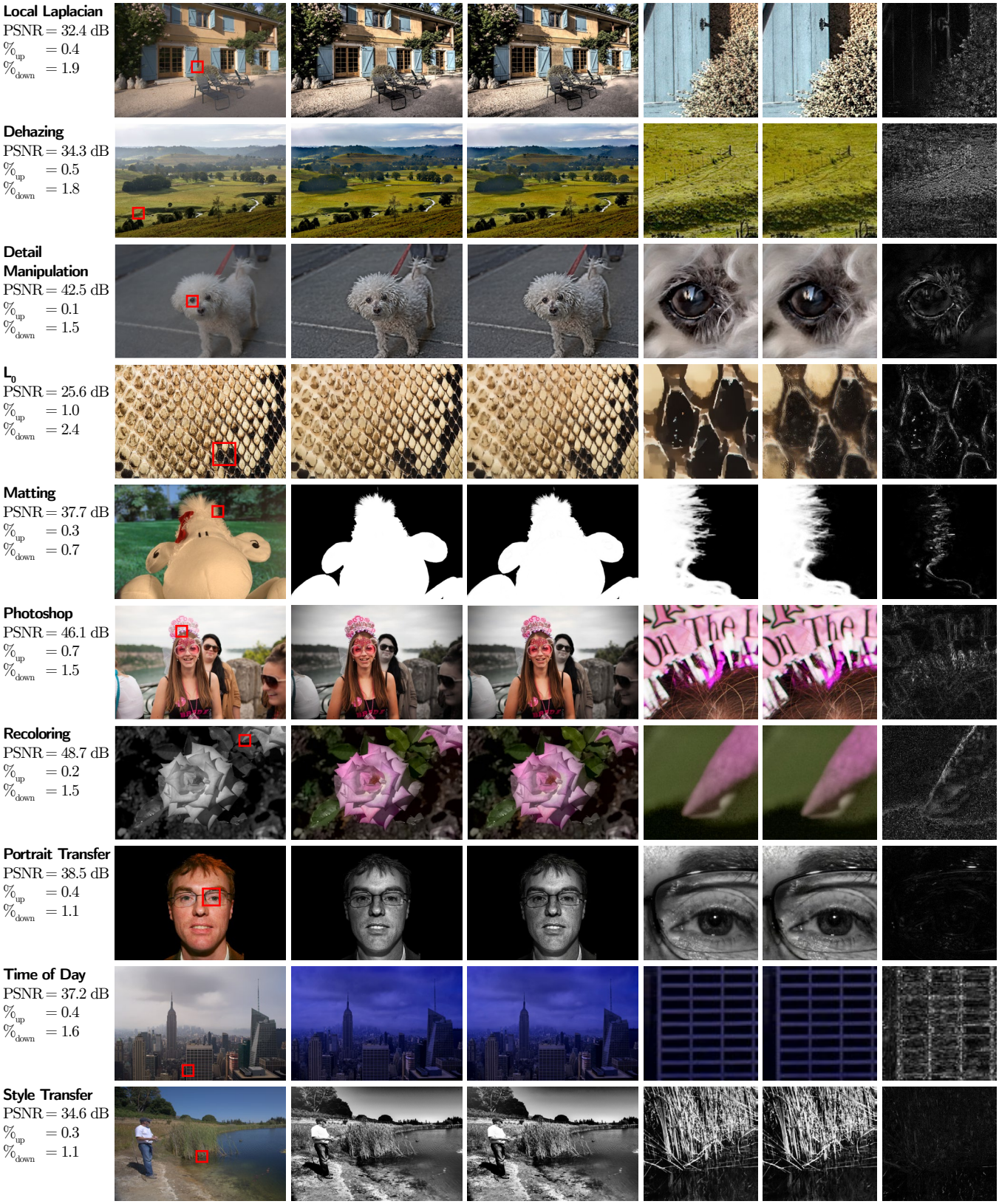


Figure 11: *Our method handles a large variety of photographic enhancements. We filter a highly degraded copy (not shown) of the reference input (a). From the resulting degraded output, we compute the recipe parameters. We then reconstruct an output image (c) that closely approximates the reference output (b) computed from the original high-quality input. (d) and (e) are a close-up on the region of highest error in our reconstruction (shown in red on (a)). (f) is a rescaled difference map that emphasizes the location of our errors. As shown on the L_0 smoothing example, our method is not well-suited for filters that significantly alter the structure of the input. Matting example from Rhemann et al. [2009], other photographs from the MIT 5k dataset [2011].*

		PSNR (dB)	relative PSNR (dB)						
features	Block overlap	✓	·	✓	·	·	✓	✓	·
	Multiscale	✓	·	·	✓	·	✓	·	✓
	Non-linear	✓	·	·	·	✓	·	✓	✓
enhancement	Local Laplacian	34.6	**-3.9**	**-3.2**	**-5.2**	-2.2	**-4.7**	-1.4	**-4.2**
	Dehazing	34.3	-0.8	-0.2	-2.0	-0.3	-1.5	-0.2	-1.8
	Detail Manipulation	33.9	**-4.4**	**-3.6**	-1.5	**-3.2**	-1.0	-2.5	-0.6
	L_0	32.5	**-3.5**	-3.0	-1.3	-2.7	-0.8	-2.1	-0.6
	Matting	32.9	-2.3	-1.5	-2.0	-1.0	-1.3	-0.1	-0.8
	Photoshop	40.0	**-3.7**	**-3.0**	**-3.4**	-1.7	-2.3	-1.0	-1.2
	Recoloring	44.6	-1.3	-0.1	-1.4	-1.2	-0.1	-0.2	-1.3
	Portrait Transfer	36.7	**-4.0**	-2.9	-2.6	-2.2	-1.3	-1.1	-1.4
	Time of Day	35.8	-1.5	-2.1	-2.2	-1.0	-1.6	-1.7	-0.7
	Style Transfer	32.8	**-4.2**	**-3.5**	**-4.0**	-0.8	**-3.3**	-0.0	-0.7
	all	35.8	-3.0	-2.3	-2.6	-1.6	-1.8	-1.0	-1.3

Table 2: *Per-application* PSNR *for recipes with an average input compression rate of* $\%_{up} = 1.0$*, and output compression rate* $\%_{down} = 1.5$*, including the relative effect of individual features of our representation.* Block overlap *allows us to linearly interpolate between overlapping block and avoids blocking artifacts.* Multiscale *refers to the use of several pyramid levels to represent the high-pass of the luminance channel. This allows us to capture subtle multiscale effects.* Non-linear *refers to the use of a piecewise-linear tone curve in the representation of the high-pass luminance. This allows us to faithfully capture tonal variations.*

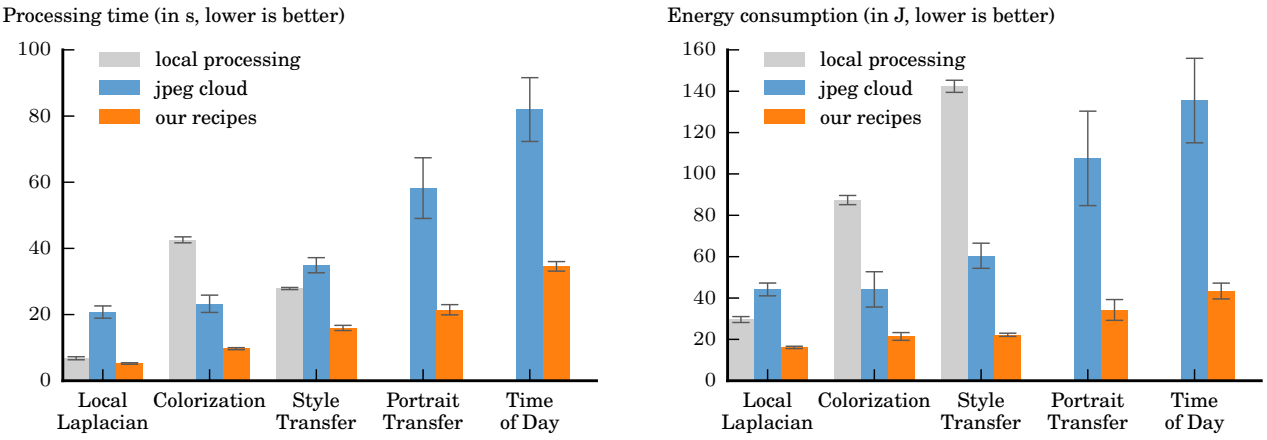


Figure 12: *Processing time and energy consumption for the filters for three computation schemes: purely local processing, transferring JPEG images to and from a cloud server, and our approach using transform recipes with a cloud server. The measurements were performed on a Samsung Galaxy S5 on LTE with a bandwidth of 2 Mbps upstream and 8 Mbps downstream. This is a typical bandwidth one would get on a coffee shop's WiFi or under average LTE coverage. Our method always saves time and energy compared to the other two approaches. We averaged over 20 runs and report an error range of one standard deviation. The higher variance for JPEG is mainly due to network speed fluctuations.*

4.4 Discussion & limitations

Compared to filter-specific approaches, our generic recipes separate filter design from compressed representation; they can be used *as-is* in existing pipelines and do not require tweaking if the filter is later changed. One could specialize the recipes for a given application, but the gain would often be limited since our generic recipes are already high-fidelity and compact thanks to the LASSO regression performing a model selection tailored to each specific filter and image.

The core idea of transform recipes—to exploit similarity between input and output—assumes a strong correlation between the input and the output of an image processing algorithm. As a result, our method does not capture the dramatic transformations of L_0 smoothing as effectively as the nine other applications in our tests. It also cannot represent significant addition of content, such as inpainting or clone brushing or significant alteration of the image structure such as Non-Photorealistic Renderings (cartoon, paintbrush effects, etc). Our existing model also does not account for spatial transformations. This excludes operations such as barrel distortion correction and perspective rectification. A parametric warp could be added, but we so far chose to focus on style transformations.

Although recipe coefficients are coarsely sampled, they are applied on image data sampled at every pixel. This allows them to generate results that adapt to the fine-grain structure of the image, including to features below the size of a block. In particular, within a small patch, it is possible to capture edge-aware effects even though the representation has no notion of edge-awareness. The example of the Matting Laplacian makes this clear. The output alpha matte is extremely edge aware (black on one side of the edge, white on

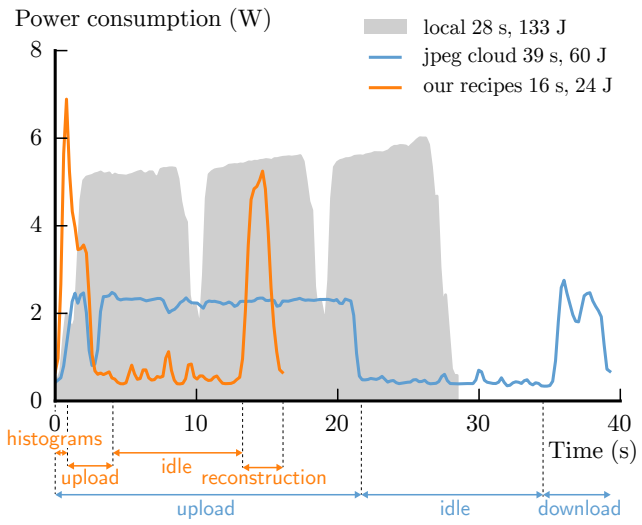


Figure 9: *Power consumption over time for three computation schemes: purely local processing, transferring JPEG images to and from a server, and our approach using transform recipes with a server. Measurements from a Samsung Galaxy S5 connected to an LTE network (with a bandwidth of 2 Mbps upstream and 8 Mbps downstream) running our Style Transfer application. Our approach uses less energy and takes less time than either local computation or traditional server offload.*

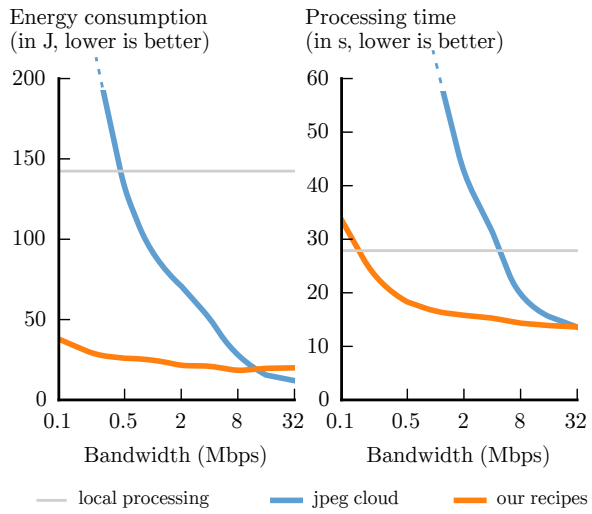


Figure 10: *Time and Energy vs. bandwidth for the Style transfer filter. JPEG cloud is not usable when the bandwidth is limited and reaches 630s and 533J for a 100kbps bandwidth. Our method still offers energy and time savings compared to local processing in this regime. For filters that cannot be run on the mobile device (database or private code) our solution makes processing feasible even in the face of scarce connectivity.*

the other, finely following the edge contour in between) yet Levin et al. [2008] showed that it can usually be represented as a local linear combination of the RGB channels, the simplest regression model one can think of. Similarly, our recipes rely on local regressions with simple, yet richer channels than RGB alone (luminance, chrominance, stack levels, etc.)

Overall, the development of good recipes requires a trade-off between expressiveness, compactness and reconstruction cost. In some scenarios, enhancement-specific recipes might provide significant gains. We believe our framework provides a strong and general foundation on which to build future systems.

5 Conclusion

The effectiveness of cloud-based image enhancement is limited by the time and energy cost of data transfers. In the case of algorithms that preserve the photographic content of an image, we have shown that we can dramatically reduce the data transferred between a mobile client and the cloud by exploiting the similarity between the input and the output. To do this, we introduced transform recipes, a flexible representation which capture how images are locally modified by an enhancement. Recipes dramatically reduce the bandwidth required for cloud computation because they encode edits compactly, and are robust to input image degradation. They allow the client to send a highly-compressed input to the server for processing, receive a compact recipe in return, and apply the recipe locally on the high-quality input. Finally, we demonstrated that transform recipes can efficiently represent a wide range of photographic processing algorithms, requiring only 1% of the combined input/output data—an order of magnitude less than transmitting reasonable-quality JPEG images—to achieve consistently over 30 dB fidelity to the original filter. In practical scenarios, this compression leads to dramatically reduced time and energy costs compared to either local processing or traditional cloud offload with full images.

Acknowledgements

We thank the SIGGRAPH reviewers for their constructive comments. This work was partially funded by the Qatar Computing Research Institute, DARPA agreement FA8750-14-2-0009, the Stanford Pervasive Parallelism Lab and a gift from Adobe.

References

AUBRY, M., PARIS, S., HASINOFF, S. W., KAUTZ, J., AND DURAND, F. 2014. Fast local laplacian filters: Theory and applications. *ACM Trans. Graph. 33*, 5 (Sept.), 167:1–167:14.

BARR, K. C., AND ASANOVIĆ, K. 2006. Energy-aware lossless data compression. *ACM Trans. Comput. Syst. 24*, 3 (Aug.), 250–291.

BERTHOUZOZ, F., LI, W., DONTCHEVA, M., AND AGRAWALA, M. 2011. A framework for content-adaptive photo manipulation macros: Application to face, landscape, and global manipulations. *ACM Transactions on Graphics 30*, 5.

BURT, P. J., AND ADELSON, E. H. 1983. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications 31*, 4, 532–540.

BYCHKOVSKY, V., PARIS, S., CHAN, E., AND DURAND, F. 2011. Learning photographic global tonal adjustment with a database of input / output image pairs. In *IEEE Conference on Computer Vision and Pattern Recognition*.

CHEN, Q., LI, D., AND TANG, C.-K. 2012. Knn matting. In *IEEE Conference on Computer Vision and Pattern Recognition*, 869–876.

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

FARBMAN, Z., FATTAL, R., LISCHINSKI, D., AND SZELISKI, R. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. In *ACM Transaction on Graphics (SIGGRAPH)*, ACM, New York, NY, USA, SIGGRAPH '08, 67:1–67:10.

FARBMAN, Z., FATTAL, R., AND LISCHINSKI, D. 2011. Convolution pyramids. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia) 30*, 6.

FREEMAN, W. T., AND TORRALBA, A. 2002. Shape recipes: Scene representations that refer to the image. In *Vision Sciences Society Annual Meeting*, MIT Press, 25–47.

HAMILTON, E. 1992. Jpeg file interchange format. *C-Cube Microsystems*.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '95, 229–238.

HUANG, J., QIAN, F., GERBER, A., MAO, Z. M., SEN, S., AND SPATSCHECK, O., 2012. A close examination of performance and power characteristics of 4g lte networks.

HUFFMAN, D. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE 40*, 9 (Sept), 1098–1101.

JEONG, W.-K., JOHNSON, M. K., YU, I., KAUTZ, J., PFISTER, H., AND PARIS, S. 2011. Display-aware image editing. In *International Conference on Computational Photography*.

KAUFMAN, L., LISCHINSKI, D., AND WERMAN, M. 2012. Content-aware automatic photo enhancement. *Computer Graphics Forum 31*, 8, 2528–2540.

KIM, J.-H., JANG, W.-D., SIM, J.-Y., AND KIM, C.-S. 2013. Optimized contrast enhancement for real-time image and video dehazing. *J. Vis. Comun. Image Represent. 24*, 3 (Apr.), 410–425.

KUMAR, K., LIU, J., LU, Y.-H., AND BHARGAVA, B. 2013. A survey of computation offloading for mobile systems. *Mob. Netw. Appl. 18*, 1 (Feb.), 129–140.

LAFFONT, P.-Y., REN, Z., TAO, X., QIAN, C., AND HAYS, J. 2014. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM Transaction on Graphics (SIGGRAPH) 33*, 4 (July), 149:1–149:11.

LEE, K., CHU, D., CUERVO, E., KOPF, J., GRIZAN, S., WOLMAN, A., AND FLINN, J. 2014. Outatime: Using speculation to enable low-latency continuous interaction for cloud gaming. Tech. Rep. MSR-TR-2014-115.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM Transaction on Graphics (SIGGRAPH) 23*, 3 (Aug.), 689–694.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2008. A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 2 (Feb), 228–242.

LEVOY, M. 1995. Polygon-assisted jpeg and mpeg compression of synthetic images. In *Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '95, 21–28.

LIKAMWA, R., PRIYANTHA, B., PHILIPOSE, M., ZHONG, L., AND BAHL, P. 2013. Energy characterization and optimization of image sensing toward continuous mobile vision. In *Proc. of International Conference on Mobile Systems, Applications, and Services*, ACM, 69–82.

MANTIUK, R., AND SEIDEL, H.-P. 2008. Modeling a generic tone-mapping operator. *Computer Graphics Forum (Proc. of Eurographics) 27*, 2.

PARIS, S., HASINOFF, S. W., AND KAUTZ, J. 2011. Local laplacian filters: Edge-aware image processing with a laplacian pyramid. In *ACM Transaction on Graphics (SIGGRAPH)*, ACM, New York, NY, USA, SIGGRAPH '11, 68:1–68:12.

RABBANI, M., AND JONES, P. W. 1991. *Digital Image Compression Techniques*, 1st ed. Society of Photo-Optical Instrumentation Engineers (SPIE), Bellingham, WA, USA.

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

RAGAN-KELLEY, J., BARNES, C., ADAMS, A., PARIS, S., DURAND, F., AND AMARASINGHE, S. 2013. 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*, ACM, New York, NY, USA, PLDI, 519–530.

RHEMANN, C., ROTHER, C., WANG, J., GELAUTZ, M., KOHLI, P., AND ROTT, P. 2009. A perceptually motivated online benchmark for image matting. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1826–1833.

SHIH, Y., PARIS, S., DURAND, F., AND FREEMAN, W. T. 2013. Data-driven hallucination of different times of day from a single outdoor photo. *ACM Transaction on Graphics (SIGGRAPH) 32*, 6 (Nov.), 200:1–200:11.

SHIH, Y., PARIS, S., BARNES, C., FREEMAN, W. T., AND DURAND, F. 2014. Style transfer for headshot portraits. *ACM Transaction on Graphics (SIGGRAPH) 33*, 4 (July), 148:1–148:14.

SKODRAS, A., CHRISTOPOULOS, C., AND EBRAHIMI, T. 2001. The jpeg 2000 still image compression standard. *IEEE Signal Processing Magazine 18*, 5, 36–58.

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

TORRALBA, A., AND FREEMAN, W. 2003. Properties and applications of shape recipes. In *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, II–383–90 vol.2.

WALLACE, G. 1992. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics 38*, 1 (Feb), xviii–xxxiv.

WELCH, T. 1984. A technique for high-performance data compression. *Computer 17*, 6 (June), 8–19.

WITTEN, I. H., NEAL, R. M., AND CLEARY, J. G. 1987. Arithmetic coding for data compression. *Communications of the ACM 30*, 6, 520–540.

XU, L., LU, C., XU, Y., AND JIA, J. 2011. Image smoothing via l0 gradient minimization. *ACM Transaction on Graphics (SIGGRAPH) 30*, 6 (Dec.), 174:1–174:12.

ZIV, J., AND LEMPEL, A. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory 23*, 3 (May), 337–343.