Deep Bilateral Learning for Real-Time Image Enhancement

MICHAËL GHARBI, MIT CSAIL
JIAWEN CHEN, Google Research
JONATHAN T. BARRON, Google Research
SAMUEL W. HASINOFF, Google Research
FRÉDO DURAND, MIT CSAIL / Inria, Université Côte d'Azur

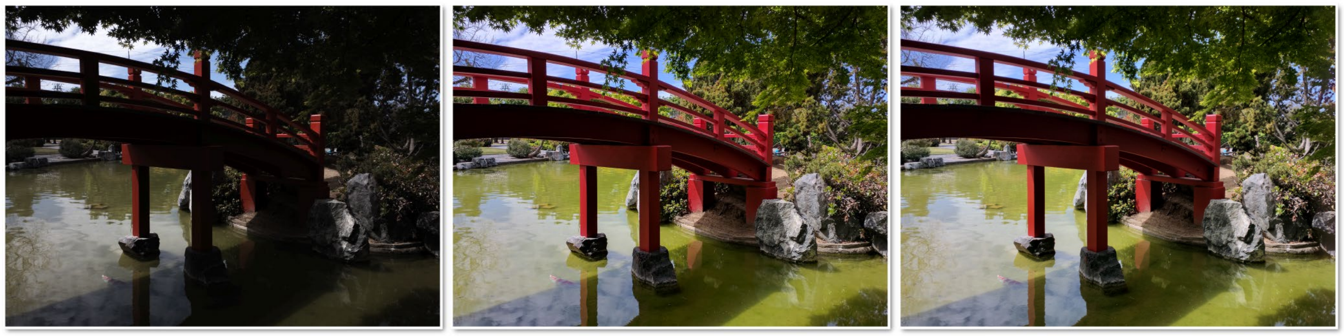
12 megapixel 16-bit linear input
(tone-mapped for visualization)
tone-mapped with HDR+
400 – 600 ms
processed with our algorithm
61 ms, PSNR = **28.4 dB**

Fig. 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.

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.

CCS Concepts: • **Computing methodologies** → **Computational photography**; **Image processing**;

Additional Key Words and Phrases: real-time image processing, deep learning, data-driven methods, convolutional neural networks

ACM Reference format:
Michaël Gharbi, Jiawen Chen, Jonathan T. Barron, Samuel W. Hasinoff, and Frédo Durand. 2017. Deep Bilateral Learning for Real-Time Image Enhancement. *ACM Trans. Graph.* 36, 4, Article 118 (July 2017), 12 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073592

1 INTRODUCTION

The high resolution of images and videos produced by contemporary cameras and mobile devices puts significant performance pressure on image processing algorithms, requiring sophisticated code optimization by skilled programmers. While systems contributions have sought to facilitate the implementation of high-performance executables, e.g. [Hegarty et al. 2014; Mullapudi et al. 2016; Ragan-Kelley et al. 2012], they require programmer expertise, their runtime cost still grows with the complexity of the pipeline, and they are only applicable when source code is available for the filters. Additionally, because image enhancement is subjective, it is often desirable to learn an enhancement model directly from human adjustments, e.g. [Bychkovsky et al. 2011]. To this end, 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. We focus on photographic enhancements that do not spatially warp the image or add new edges, e.g. [Aubry et al. 2014; Hasinoff et al. 2016].

We share the motivation of prior work that seeks to accelerate "black box" image processing operations, either by using a remote server, e.g. [Gharbi et al. 2015] or by processing a low-resolution image and then using the low-resolution output to approximate a high-resolution equivalent [Chen et al. 2016]. For some operations, these approaches can achieve large speedups but they suffer from significant limitations: the underlying image processing operation must be somewhat scale-invariant (Figure 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. [Isola et al. 2016; Liu et al. 2016; Long et al. 2015; Xu et al. 2015; Yan 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* from input to output rather than predicting the output directly e.g., [Chen et al. 2016; Gharbi et al. 2015; Shih et al. 2013]. 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. 2014; Hasinoff et al. 2016], reverse-engineering black-box Photoshop actions, and learning the retouching style of photographers [Bychkovsky et al. 2011] 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 image on a Google Pixel phone, thereby enabling real-time viewfinder effects at 50 Hz.

2 RELATED WORK

Though image enhancement algorithms have been the focus of a great deal of research, most sophisticated algorithms are too expensive to be evaluated quickly on mobile devices, which is where the vast majority of digital images are captured and processed. Because of this, previous work has identified specific critical operations and developed novel algorithms to accelerate them. For instance, Farbman et al. [2011] 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 [Adams et al. 2010; Chen et al. 2007; Paris and Durand 2006; Tomasi and Manduchi 1998].

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.

Gharbi et al. [2015] focus on learning the *transformation* from input to output instead of the output itself. They approximate a large class of complex, spatially-varying operators with a collection of simple local models—a *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. Similarly, 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. 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 (Figure 9).

Neural networks for image processing. Recently, deep convolutional networks have achieved significant progress on low-level vision and image processing tasks such as depth estimation [Eigen et al. 2014], optical flow [Ilg et al. 2016], super-resolution [Dong et al. 2014], demosaicking and denoising [Gharbi et al. 2016; Zhang et al. 2016], image matting [Shen et al. 2016], colorization [Iizuka et al. 2016], and general image-to-image "translation" tasks [Isola et al. 2016]. Recent work has even explored learning deep networks within a bilateral grid [Jampani et al. 2016] though this work does not address our task of learning image transformations in that space, and instead focuses on classification and semantic segmentation. Some architectures have been trained to approximate a general class of 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 our application: real-time processing of high-resolution images on a mobile device.

Automatic photo editing. Our model can be trained to automatically correct photographs from input/output image pairs provided by a human retoucher. This is the task introduced by Bychkovsky et al. [2011], 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. [2012] 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). Our model can learn similar global tonal adjustments and generalizes to more complex effects, including color corrections and local edits, in addition to being much faster.

3 OUR ARCHITECTURE

We propose a new convolutional network architecture that can be trained to perform fast image enhancement (Figure 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 $\tilde{\mathrm{I}}$ of the input I in the *low-res* stream (Fig. 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. 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 [Chen et al. 2007; Paris and Durand 2006]. 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.'s Bilateral Guided Upsampling [2016]. 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 9). We illustrate the role of each component of our architecture with an ablation study in Figures 3, 4, 5 and 7.

3.1 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. [2016], 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. 2015] and specializes in learning local features that propagate image data while retaining spatial information. The second path $(G^i)_{i=1,\ldots,n_G}$ 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 3.2). 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*.

3.1.1 *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 2):

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

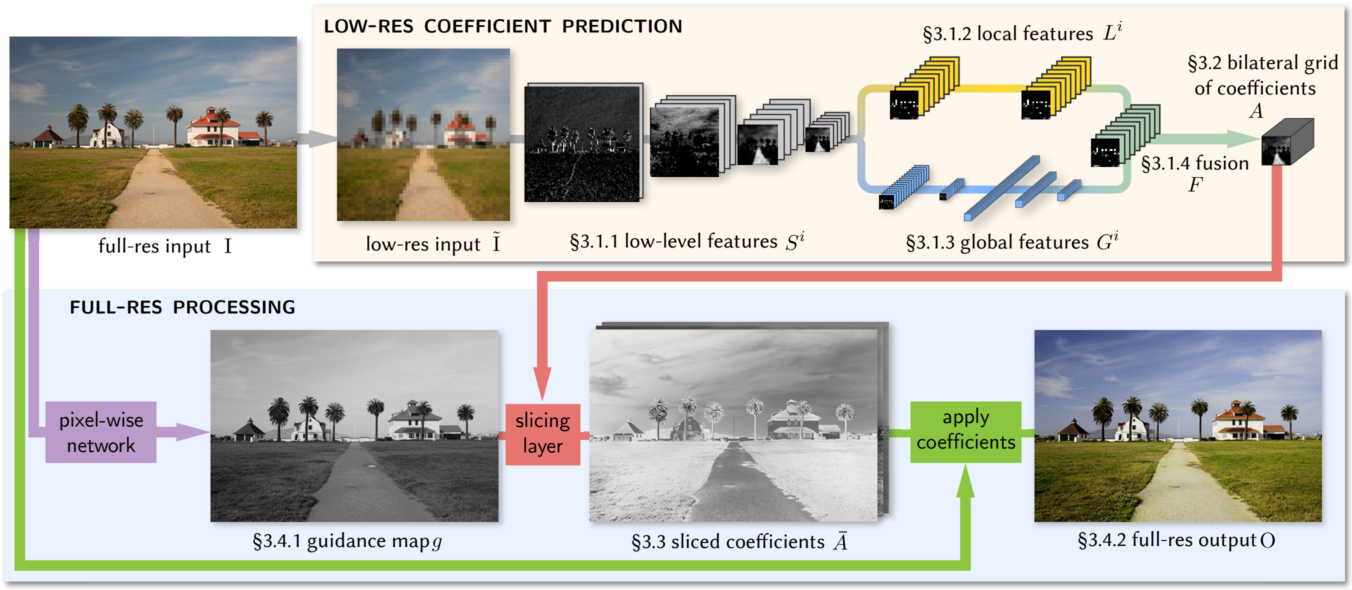
LOW-RES COEFFICIENT PREDICTION
full-res input I
low-res input Ĩ
§3.1.1 low-level features S^i
§3.1.2 local features L^i
§3.1.3 global features G^i
§3.1.4 fusion F
§3.2 bilateral grid of coefficients A
FULL-RES PROCESSING
pixel-wise network
§3.4.1 guidance map g
slicing layer
§3.3 sliced coefficients $\bar{A}$
apply coefficients
§3.4.2 full-res output O

Fig. 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 Ĩ 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.

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 Ĩ 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 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 1 summarizes the dimensions of each layer.

3.1.2 *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 2, yellow). These layers take the same form as Equation (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.

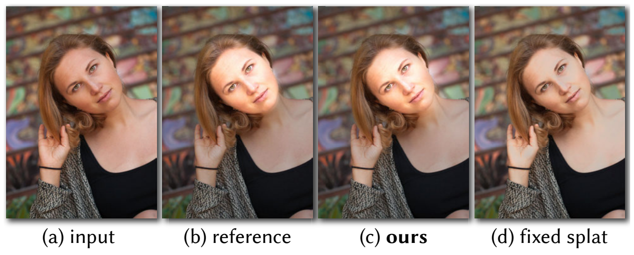

Fig. 3. Our *low-level* convolutional layers are fully learned and can extract semantic information. Replacing these layers with the standard bilateral grid splatting operation causes the network to lose much of its expressive power. In this example of our *Face brightening* operator (a-b), the network 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.

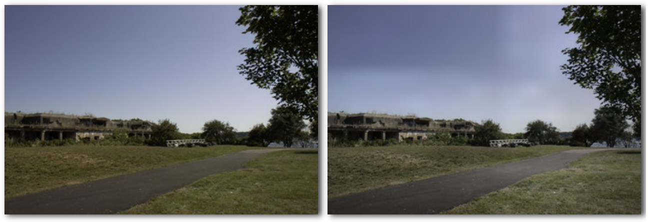
(a) **ours**, with global features (b) without global features

Fig. 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.

3.1.3 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 (1), with $s = 2$) followed by three fully-connected layers, for a total of $n_G = 5$ global layers (Figure 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 3.3, 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.

3.1.4 Fusion and linear prediction. We fuse the contributions of the local and global paths with a pointwise affine mixing followed by a ReLU activation:

$$F_c[x,y] = \sigma\left(b_c + \sum_{c'} w'_{cc'} G^{n_G}_{c'} + \sum_{c'} w_{cc'} L^{n_L}_{c'}[x,y]\right) \tag{2}$$

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:

$$A_c[x,y] = b_c + \sum_{c'} F_{c'}[x,y] w_{cc'} \tag{3}$$

Table 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

3.2 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:

$$A_{dc+z}[x,y] \leftrightarrow A_c[x,y,z] \tag{4}$$

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 lets us interpret the strided convolutions in Equation (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.

3.3 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:

$$\bar{A}_c[x,y] = \sum_{i,j,k} \tau(s_x x - i)\tau\left(s_y y - j\right)\tau(d \cdot g[x,y] - k) A_c[i,j,k] \tag{5}$$

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

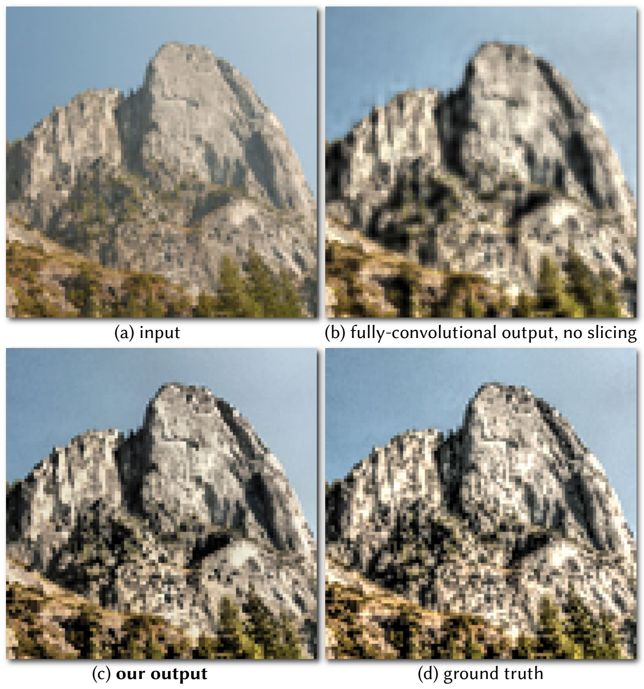

Fig. 5. Our new slicing node is central to the expressiveness of our architecture and its handling of high-resolution effects. Replacing this node with a standard bank of learnable deconvolution filters reduces expressiveness (b) because no full-resolution data is used to predict the output pixels. Thanks 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).

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 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 10).

This data-dependent lookup is critical to the expressive power of our model. As we will see in Section 3.4.2, it allows us to predict a complex operation on the full-resolution image using a collection of much simpler local models.

3.4 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.

3.4.1 Guidance map auxiliary network. We define g as a simple pointwise nonlinear transformation of the full-resolution features:

$$g[x,y] = b + \sum_{c=0}^{2} \rho_c\left(\boldsymbol{M}_c^\top \cdot \phi_c[x,y] + b'_c\right) \tag{6}$$

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

$$\rho_c(x) = \sum_{i=0}^{15} a_{c,i} \max(x - t_{c,i}, 0) \tag{7}$$

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 7 shows the impact of using this learned guide and Figure 6 shows an example of the color transformation matrix and tone curve that are learned for the corresponding task.

3.4.2 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 [Chen et al. 2016; Gharbi et al. 2015; He and Sun 2015]. We therefore model each channel of our final output O_c as an affine combination of the full-resolution features, with coefficients defined by the channels of the sliced feature map $\bar{A}$:

$$\mathrm{O}_c[x,y] = \bar{A}_{n_\phi+(n_\phi+1)c} + \sum_{c'=0}^{n_\phi-1} \bar{A}_{c'+(n_\phi+1)c}[x,y]\,\phi_{c'}[x,y] \tag{8}$$

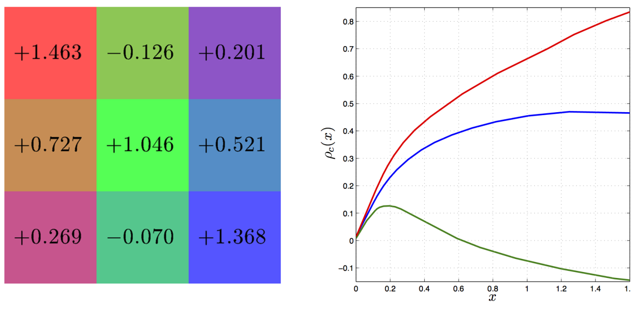

Fig. 6. The color transform matrix (left) and per-channel tone curves (right) used to produce the guidance map g, as learned by one instance of our model.

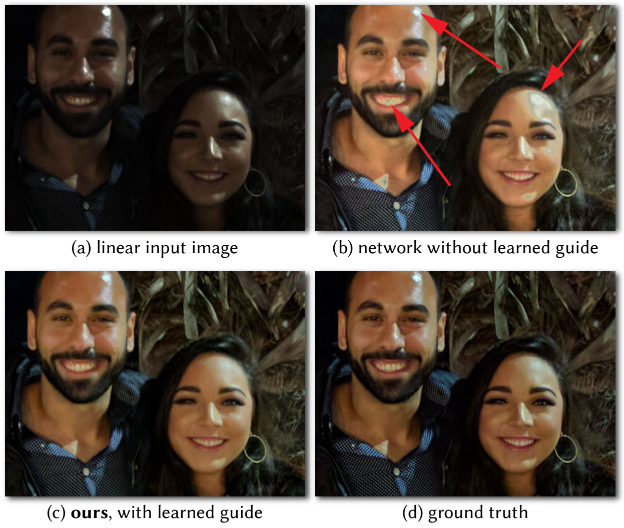
(a) linear input image
(b) network without learned guide
(c) **ours**, with learned guide
(d) ground truth

Fig. 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).

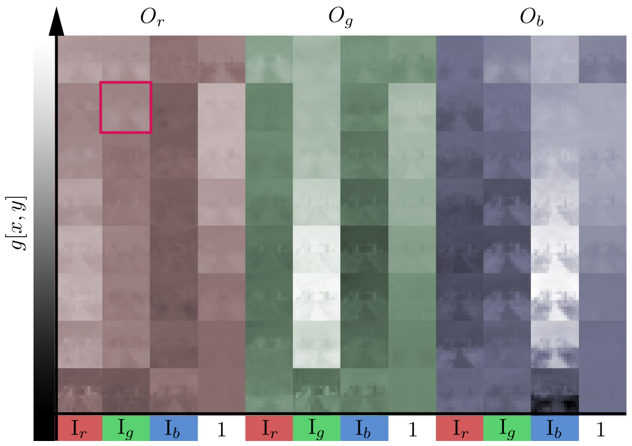

Fig. 8. Coefficient maps for the affine color transform. The vertical axis corresponds to the learned guidance channel, while the horizontal axis unrolls the 3x4 sets of coefficients. Each thumbnail, one example of which is highlighted, shows a 16x16 low-resolution map.

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. 2013]. 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 8 for a visualization of the 3D bilateral grid of affine coefficients A corresponding to the input/output pair in Figure 2. One of the 12 channels of the 2D coefficients after slicing can also be seen in Figure 2.

3.5 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:

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_i ||\mathrm{I}_i - \mathrm{O}_i||^2 \tag{9}$$

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. 2012]. For all experiments, models are trained on an NVIDIA Titan X (Maxwell) for 30 epochs, which typically takes 2–3 days.

4 RESULTS

We evaluate our model's ability to reproduce both algorithmic image operators (Section 4.1) and human-annotated retouches (Section 4.2). 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 4.3).

A selection of our results on different tasks can be seen in Figure 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 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.

4.1 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.
. the *Local Laplacian* filter [Paris et al. 2011] – 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. 2014] (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 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. [2014]). 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.

4.1.1 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. 2011]. 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) .

4.1.2 Baseline. The previous work closest in spirit to our goals are Bilateral Guided Upsampling (BGU) [Chen et al. 2016] and Transform Recipes (TR) [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.

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

Table 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

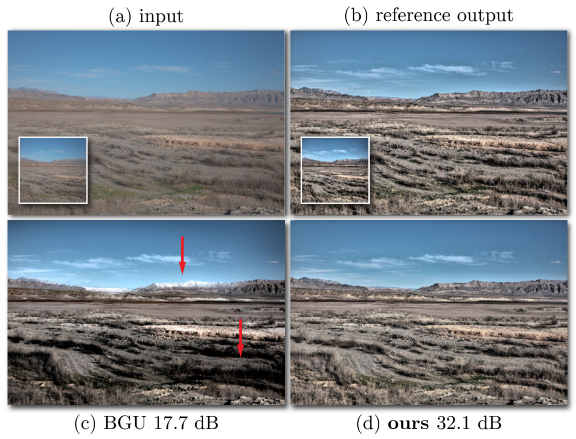

Fig. 9. Our method (d) can learn to replicate the correct effect (b) for operations that are not scale invariant, such as the Local Laplacian filter shown here (a–b). Methods like Bilateral Guided Upsampling that only apply the operation at low-resolution (insets (a–b)) produce a different-looking output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

Transform Recipes assumes that a mobile device would send a highly compressed (and therefore degraded) image to a server for processing, and would recieve an inexpensive "recipe" for approximating an image transformation from that server. Because TR's client-server setup is not relevant to the scope of this paper, we run the model (using the authors' recommended settings) on uncompressed, full-resolution images, thereby improving output quality and making our TR 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 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 9).

4.2 Learning from human annotations

We also evaluate accuracy with regards to human annotations using the MIT-Adobe "FiveK" dataset [Bychkovsky et al. 2011], and our performance compared to previous work is presented in Table 3. This task measures our model's ability to learn a highly subjective

Table 3. Mean L_2 error in L*a*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 [Hwang et al. 2012; Yan et al. 2016] are denoted by †.

photographer	method	L*a*b*	L-only
C *random250*	ours	**7.8**	**5.5**
	Yan [2016]	9.85†	–
	Bychkovsky [2011]	–	5.8
	Hwang [2012]	15.01†	–
C *highvar50*	ours	**7.1**	**5.2**
	Yan [2016]	8.36†	–
	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

image operator which requires a significant amount of learning and semantic reasoning. We report mean L_2 error in L*a*b* space (lower is better) for retouches by the 5 photographers (A,B,C,D,E) in the MIT "FiveK" dataset, though previous work only presents results on photographer C [Hwang et al. 2012; Yan et al. 2016]. 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.

4.3 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. 2012] of the Local Laplacian filter [Paris et al. 2011] on the same device takes over 200 ms. Running the same filter at the reduced 256×256 resolution and applying

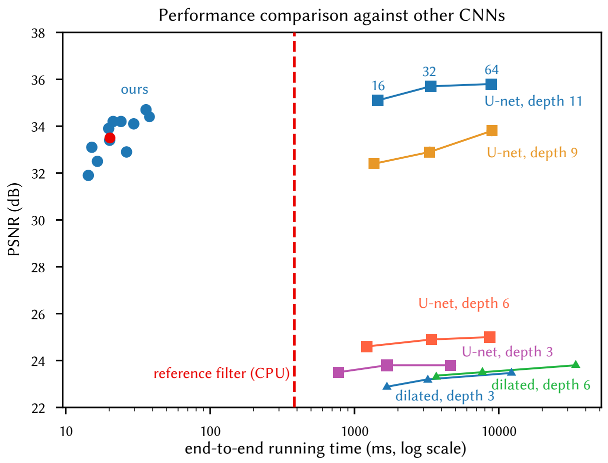

Fig. 10. We compare the speed and quality of our algorithm against two modern network architectures: *U-Net* (adapted from [Isola et al. 2016]) and *dilated convolutions* [Yu and Koltun 2015]. The runtimes were averaged over 20 iterations, processing a 4 megapixel image on a desktop CPU. The PSNR numbers 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 11 for details on the speed/quality trade-off of our model.

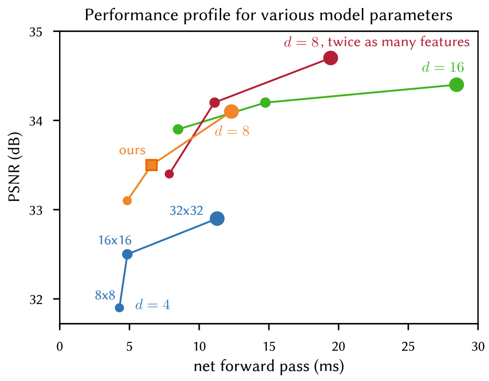

Fig. 11. We show PSNRs for the *Local Laplacian* task and the computation time required to predict the bilateral coefficients with several settings of our model's parameters. Each curve represent a grid depth d. For each curve the grid's spatial resolution varies in {8, 16, 32}. The reference model we 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.

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 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. Mosty 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 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 1). The summary can be found in Figure 11.

4.4 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 3.4). If one relaxes the real-time rendering constraint, one can extend our model by extracting features from the high-resolution image. In Figure 13, 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 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 12).

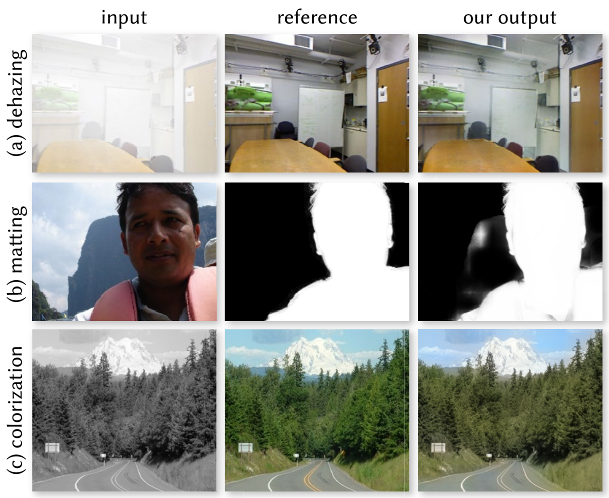

Fig. 12. Our algorithm fails when the image operator strongly violates our modeling assumptions. (a) Haze reduces local contrast, which limits the usefulness of our guidance map. It also destroys image details that cannot be recovered with our affine model (e.g., on the whiteboard). (b) Matting has successfully 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).

5 CONCLUSION

We have introduced a new neural network architecture that can perform image enhancement in real-time on full-resolution images while still capturing high-frequency effects. Our model is trained using pairs of input/output images, allowing it to learn from a reference implementation of some algorithm or 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 expressivity and speed. To build this model we have introduced two new layers: a data-dependent lookup that enables slicing into the bilateral grid, and a multiplicative operation for affine transformation. By training in an end-to-end fashion and optimizing our loss function at full resolution (despite most of our network being at a heavily reduced resolution), our model is capable of learning full-resolution and non-scale-invariant effects. The accuracy of our model has been demonstrated on a variety of different image operators, pipelines, and subjective human-annotated datasets.

ACKNOWLEDGMENTS

We thank the SIGGRAPH reviewers for their constructive comments. Special thanks to Marc Levoy for his valuable feedback. This work was partially funded by Toyota.

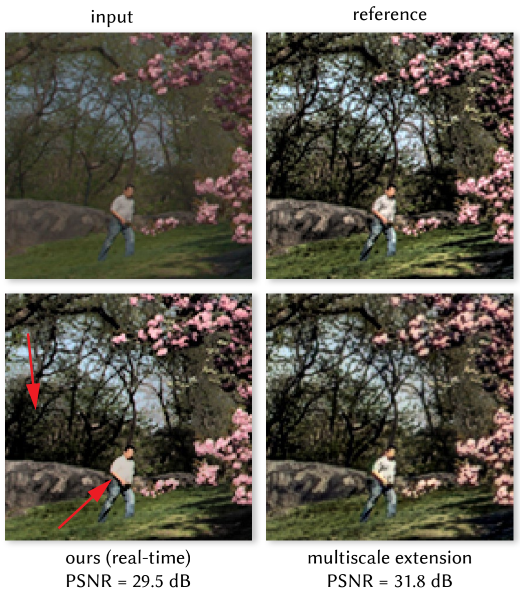

Fig. 13. At the expense of extra computation at full-resolution, our model can be extended with richer affine regression features. Here, by using a 3-level Gaussian pyramid as features ϕ, we can better capture the high-frequency details in the the *Local Laplacian (strong)* task.

REFERENCES

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. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/

Andrew Adams, Jongmin Baek, and Myers Abraham Davis. 2010. Fast High-Dimensional Filtering Using the Permutohedral Lattice. *Computer Graphics Forum* (2010).

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

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

Jonathan T Barron and Ben Poole. 2016. The Fast Bilateral Solver. *ECCV* (2016).

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

Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. 2011. Learning Photographic Global Tonal Adjustment with a Database of Input / Output Image Pairs. *CVPR* (2011).

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

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

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

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

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

Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. 2016. Deep Joint Demosaicking and Denoising. *ACM TOG* (2016).

Michaël Gharbi, YiChang Shih, Gaurav Chaurasia, Jonathan Ragan-Kelley, Sylvain Paris, and Frédo Durand. 2015. Transform Recipes for Efficient Cloud Photo Enhancement. *ACM TOG* (2015).

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

Kaiming He and Jian Sun. 2015. Fast Guided Filter. *CoRR* (2015).

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR* (2015).

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

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

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

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

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

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2016. Image-to-Image Translation with Conditional Adversarial Networks. *CoRR* (2016).

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

Vidit Jain and Erik Learned-Miller. 2010. *FDDB: A Benchmark for Face Detection in Unconstrained Settings.* Technical Report UM-CS-2010-009. University of Massachusetts, Amherst.

Varun Jampani, Martin Kiefel, and Peter V. Gehler. 2016. Learning Sparse High Dimensional Filters: Image Filtering, Dense CRFs and Bilateral Neural Networks. *CVPR* (2016).

Liad Kaufman, Dani Lischinski, and Michael Werman. 2012. Content-Aware Automatic Photo Enhancement. *Computer Graphics Forum* (2012).

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

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

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

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

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

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

Ravi Teja Mullapudi, Andrew Adams, Dillon Sharlet, Jonathan Ragan-Kelley, and Kayvon Fatahalian. 2016. Automatically Scheduling Halide Image Processing Pipelines. *ACM TOG* (2016).

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

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

Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. 2012. Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines. *ACM TOG* (2012).

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

Xiaoyong Shen, Xin Tao, Hongyun Gao, Chao Zhou, and Jiaya Jia. 2016. Deep Automatic Portrait Matting. *ECCV* (2016).

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

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

Li Xu, Jimmy Ren, Qiong Yan, Renjie Liao, and Jiaya Jia. 2015. Deep Edge-Aware Filters. *ICML* (2015).

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

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

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

Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. 2016. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *CoRR* (2016).

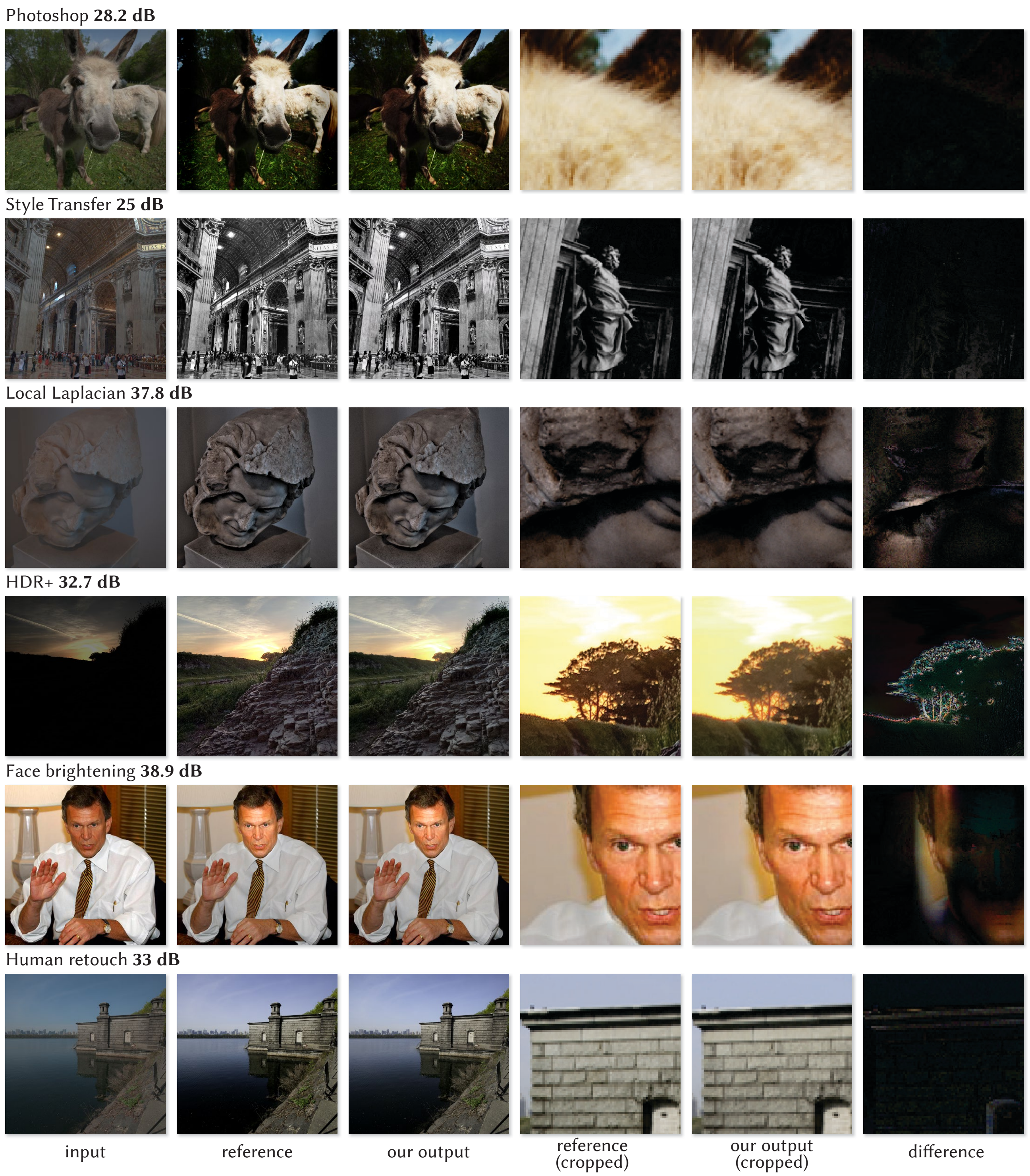

Fig. 14. Our method can learn accurate and fast approximations of a wide variety of image operators, by training on input/output pairs processed by that operator. These operators can be complicated "black box" image processing pipelines where only a binary is available, such as HDR+ or Photoshop filters/actions. 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. The difference is rescaled to use the full [0, 1] range.