Local Image Histograms for Learning Exposure Fusion

Justin Solomon and Ravi Sankar

1 Introduction

Local image histograms provide valuable descriptors of the behavior of an image around a given pixel. Rather than bucketing intensity values for all the pixels in an image, local histograms are defined separately for each pixel and represent the distribution of nearby intensities. Several image filters, including the median, mean-shift, and bilateral, can be described and computed using these histograms.

In the SIGGRAPH 2010 conference, a new technique was introduced for efficiently computing such local histograms at all points in an image [2]. While previous algorithms had time complexity proportional to the support of the histogram, the new approach always runs in $O(1)$ time per pixel, taking advantage of efficient Gaussian blur operations to simplify the local histogram computation process.

While [2] applies local image histograms to the re-expression of previously-known image filters, these histograms also provide interesting features that could be used to apply machine learning to computational photography. In this project, we propose the application of this technique to learning reasonable approaches to exposure fusion, in which images of a scene taken at different exposures are linearly combined at each pixel to generate a meaningful output using the entire dynamic range of the display device.

Ideally, although there obviously is a strong correlation between the histograms and values of adjacent pixels in a photograph, the histograms provide sufficiently strong descriptions of local behavior that this dependence can be ignored. Thus, we use examples of successfully-fused images and their per-pixel histograms as training data to learn a function mapping a single pixel and its corresponding histogram in each of the different exposures to a single output. This dependence likely is non-linear, since local histograms often exhibit complex or even bimodal distributions. For this reason, we apply the "Least-Squares SVM" kernelized regression technique, which implicitly makes use of high-order features [3]. In the particular setting of image processing with locally-weighted histograms, evaluation of the learned exposure fusion function can be made considerably more efficient by taking advantage of the fact that histograms change with relatively low frequency across most images; a key contribution of this project is a new algorithm for efficient evaluation that makes image-processing-by-example a more feasible task.

This project represents one of the first data-driven techniques in computational photography introduced to the graphics community. Its timings compare favorably with comparable approaches and more standard image processing techniques, and the output exhibits relatively few undesirable artifacts.

2 Background

Local image histograms are well-explored objects in computer graphics and provide a generalized framework for expressing several common image filters. The histograms are stored at all pixels of an image and represent the distribution of nearby intensities. A general theory of image processing can be built around the processing and manipulation of these histograms, providing a common language used to describe various operations. For instance, the median filter seeks the 50% point of the CDF of each pixel's local histogram, the mean-shift snaps to nearby histogram peaks, and the bilateral can be written as the ratio of two histogram values. These simple operations provide strong evidence that local histograms are valuable features in themselves that contain considerable information about a pixel and its neighborhood.

Here, we apply the technique introduced in [2] for computing per-pixel local histograms. While naïve methods have existed for some time in which the intensities of a given neighborhood of a pixel are simply binned, these methods have two severe drawbacks. First, the binning procedure takes longer depending on neighborhood size; "local" histograms representing hundreds of nearby pixels will take much more time to compute than histograms of the one-ring of a pixel. More importantly, these sorts of local histograms are anisotropic, often accidentally favoring certain directions over others by using square

neighborhoods and 1-0 weights. Instead, [2] describes an efficient technique for using isotropic, weighted neighborhoods around each point. The algorithm can be described using Gaussian blurs of the input image passed through various look-up tables, and thus the local histograms for all pixels can be computed in $O(nmh)$ time for an $n \times m$ image with h histogram bins.

Of course, there is no reason why an exposure fusion function should be a linear function of the counts in local histogram bins. Thus, it is necessary to apply a nonlinear regression technique that is able to find more complex relationships between input and output variables. Luckily, although we have a relatively high-dimensional problem, a glut of training data is available: every image contains a diverse set of features and thousands of pixels with unique–if similar–histograms.

Still, the regression problem here is difficult, since it involves fitting a fairly complex function of several variables. Several parametric and nonparametric techniques were implemented and tested, including regression trees, Gaussian processes, and support vector regression; for the most part, these techniques were fairly extreme examples of either under-fitting or unacceptable timing and were discarded. One algorithm, however, that performed fairly well in both arenas is the least-squares support vector machine (LS-SVM), introduced in [4]. The LS-SVM attempts to re-express a least-squares problem similar to Gaussian process regression in the language of support vector machines. LS-SVM's tend to produce considerably more support vectors, raising the amount of time it takes to evaluate the regression function, but produce reliable fits for exposure fusion.

3 Performing Fusion

In the end, performing exposure fusion from the LS-SVM output involves evaluating the following function at each pixel:

$$f(x) = b + \sum_{i=1}^{m} \alpha_i K(x, x^{(i)}) \tag{1}$$

where $x^{(1)}, \ldots, x^{(m)}$ is the set of support vectors, b and $\{\alpha_i\}$ are values from regression, and K is the kernel function. For this project, we use the Gaussian radial basis functions $K(x, y) = \exp(-\gamma \|x - y\|^2)$ as kernels, although the algorithm easily could be adapted to other kernel functions.

Since the regression function f is evaluated at each pixel of the image, it is highly important to make the evaluation as efficient as possible. Thus, before describing the entire exposure fusion algorithm, we proceed by describing an algorithm for efficient evaluation of f.

Substituting in the Gaussian radial basis into Equation 1, we find our final expression for the exposure fusion function:

$$f(x) = b + \sum_{i=1}^{m} \alpha_i e^{-\gamma \|x - x^{(i)}\|^2} \tag{2}$$

For exposure fusion, our feature vector x can be written as the sum of orthogonal vectors $x = x_{hist} + x_{rgb}$, where x_{hist} contains histogram samples from the different exposure images and x_{rgb} contains the pixel color in each exposure. The key observation here is that x_{hist} changes with relatively low frequency across the image determined by the neighborhood size of the histograms; intuitively this makes sense since the sets of nearby intensities to two adjacent pixels should have considerable overlap. Thus, x_{hist} can be computed and manipulated in a much smaller image than the full-sized output.

In particular, we expand the exponent as follows:

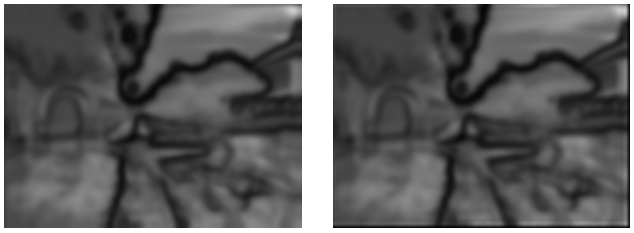
$$\begin{aligned} \|x - x^{(i)}\|^2 = &\|x\|^2 - 2x_{hist} \cdot x_{hist}^{(i)} \\ &- 2x_{rgb} \cdot x_{rgb}^{(i)} + \|x^{(i)}\|^2 \end{aligned}$$

The $\|x^{(i)}\|^2$ term can be precomputed for each data point $x^{(i)}$ ahead of time. Similarly, the $\|x\|^2$ term can be computed once for an input x before evaluating the sum over i. These simplifications leave only the two inner product terms. The $x_{hist} \cdot x_{hist}^{(i)}$ term comprises the most computational work if computed directly, since there are 10-20 samples in histogram space over multiple exposures that must be multiplied. Fortunately, as argued above, the histogram dot product terms can be computed on a lower-resolution grid and upsampled since they change with low frequency; this leaves only the rgb inner products, which take considerably less time.

The expansion above moves the learned exposure function from an intractable curiosity to a practical approach that can be applied to photographs in reasonable time. To demonstrate that approximating $x_{hist} \cdot x_{hist}^{(i)}$ on a low-resolution grid does not significantly affect output, Figure 1 compares upsampled and exact versions of these values to show how similar they are.

4 Technical Approach

As described earlier, our algorithm learns a function from $\mathbb{R}^N$ to $[0, 1]^3$, where N is the number of histogram features and colors are the output in RGB. As input, the



(a) (b)

Figure 1: (a) Exact and (b) upsampled values of $\|x - x^{(i)}\|$ for a sample image; note that they are virtually indistinguishable other than boundary artifacts, despite the fact that (b) was computed on a grid $10\times$ smaller.

learning algorithm takes p input photographs, taken at different exposures and aligned ahead of time using a standard approach. For each image, the algorithm produces local histograms at all the pixels using h bins. The vector for a given pixel is computed by concatenating the histograms from each of the exposure images and appending the color of the pixel at each exposure. Thus, we have $N = p(h + 3)$, since colors are represented using RGB values. In practice, we use 10 buckets ($h = 10$) and three exposure images ($p = 3$), giving $N = 39$ dimensions for each feature vector.

As training data, a number of photos were taken indoors and outdoors on the Stanford campus under varying lighting conditions. These images were fused using two different algorithms, both implemented in the Photomatix Pro software system. Both of these approaches, however, generated artifacts in various parts of the input images. To ensure that training data did not include these incorrect pixels, masks were drawn by hand highlighting parts of input images that were particularly well-fused. Since each image had a relatively large number of pixels and inputs were chosen because of their interest or variety of lighting, it was possible to be relatively conservative in choosing only particularly well-fused pixels for training. Given this conservative selection process as well as the use of two different algorithms for generating training data, an ideal learned model might even outperform either of the two training techniques.

The training data process described above generates more data than is usable or necessary for training a 39-dimensional LS-SVM; each input image contains millions of pixels, much of which are redundant for training purposes since they are similar to pixels nearby. Thus, images were downsized to 800×600 before histogram computation. Additionally, input pixels were chosen randomly from the masked regions.

The learning stage outputs LS-SVM parameters b and α_i described in Equation 1. The number m of support vectors often is close to the number of input images, which images it slow to evaluate the regression function. To speed up evaluation, the simple pruning technique described in [3] is implemented, in which support vectors corresponding to small α_i are discarded after fitting an LS-SVM; the reduced set of support vectors is used to train a new LS-SVM. This process is repeated several times, each time eliminating 5% of the training data until the set of support vectors is approximately 50% smaller.

Given the trained LS-SVM, running exposure fusion on a series of input images is a relatively straightforward process. Each pixel of the input is independently mapped to $\mathbb{R}^{\mathbb{N}}$ using its local histograms and intensities. Then, the LS-SVM is used to predict the fused intensity in RGB space by evaluating f with the acceleration described in Section 3.

5 Implementation

The method described above was implemented using C++ and Matlab. We implemented [2] as an addition to the ImageStack image processing library in C++,[1] using a fast IIR Gaussian kernel for generating histograms. ImageStack's built-in RANSAC image alignment method was used to align images before fusion or training to ensure that the same pixel in the different exposures corresponds to the same location in the scene. Alignment and histogram generation take place on the order of seconds for even relatively large-sized images and thus represent a reasonable approach for generating training data and feature vectors on input images; additionally, more suitable camera hardware could reduce alignment computation time or make it unnecessary, if photos could be taken one after the other.

The LS-SVMlab library was used to learn the regression function f.[2] The library not only finds α_i and b but also estimates the standard deviation parameter γ using heuristics from the set of training data. LS-SVMlab's methods for evaluating the resulting function, however, were deemed slow and were re-implemented with the modifications from Section 3 in C++.

We collected 63 test case photographs, creating 42 images worth of training data (one for each fusion method implemented in Photomatix Pro); this amount is clearly overkill given the dimensionality of our features and can

[1] http://code.google.com/p/imagestack/

[2] http://www.esat.kuleuven.be/sista/lssvmlab/

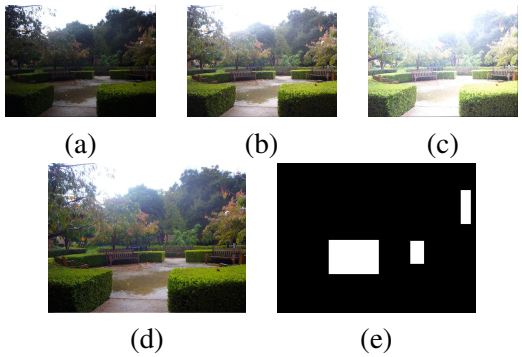


Figure 2: A sample training image at (a) dark, (b) medium, and (c) light exposures; Photomatix Pro fused these together to produce (d), and the pixels marked in the mask (e) were selected as potential training data.

Figure 3: An example of successful learned exposure fusion with source images above.

be used for testing as well as model fitting. Figure 2 shows an example training image.

6 Results

Figures 3 and 5 (final page) show examples of successful learned exposure fusion output. In general, the algorithm outlined above was successful for fusing a variety of input images, many of which contrasted significantly with those found in the training data.

The examples in Figures 3 and 5 were produced after pruning over 50% of the support vectors using the method described in Section 4. Additional pruning led to visible artifacts in the output images, whereas less pruning created no noticeable increase in quality. The final method runs in approximately one to two minutes per megapixel on a laptop equipped with an Intel Core 2 Extreme CPU.

Figure 4 (final page) shows one example where the learned fusion function was less successful. Visible artifacts appear in regions that are not well-represented in the training data and manifest themselves as low-frequency noise similar to the histogram distance functions. The main source of artifacts in the other examples provided here are due to problems with the alignment program rather than the exposure fusion function.

7 Conclusions and Future Work

The proposed technique yields a successful approach to exposure fusion that has proven effective on a variety of input images. With the modifications explained in Section 3, the approach can be carried out very efficiently, especially when compared to the hours taken to evaluate the learned filters proposed in [1], the closest comparable paper in the graphics literature.

Although exposure fusion is a valuable application in itself, perhaps the most worthwhile result of this project is a generalized approach to using LS-SVM regression for learning image filters using local histograms. The concept of "learning" an image filter is a valuable one, since artists manipulating photographs may prefer describing a complex filter by example rather than in the mathematical formalism of image processing; additionally, a system that could recognize patterns in how photos are edited could attempt to predict desirable filters before an artist begins his or her work.

A number of filtering applications could benefit from a similar approach. Potential applications of histogram-based image filter learning could include:

- Edge-preserving blurring
- Deconvolution
- Automatic upsampling
- Application of painterly or other artistic effects
- Scratch removal
- Depth of field fusion

Each of these applications could be approached with little to no modification of the LS-SVM method detailed here. Thus, an additional worthwhile goal for future research would be to improve the evaluation time of Equation 1 as much as possible. While the downsampling approach presented here helps improve runtimes considerably, Equation 1 still requires repeated evaluation of the

$\exp(\cdot)$ function for each pixel, which could be improved by use of various approximations and look-up tables.

Potential improvements aside, the success of the approach as-is indicates that automatic learning and evaluation of image filters is a worthwhile and promising avenue for future research.

References

[1] Hertzmann, Aaron et al. "Image Analogies." SIGGRAPH 2001, Los Angeles.

[2] Kass, Michael and Justin Solomon. "Smoothed Local Histogram Filters." SIGGRAPH 2010, Los Angeles.

[3] Suykens, J.A.K. et al. *Least Squares Support Vector Machines*. Singapore: World Scientific, 2002.

[4] Suykens J.A.K. and J. Vandewalle J.. "Least squares support vector machine classifiers." *Neural Processing Letters* 9.3 (1999): 293–300.

Figure 4: A less successful exposure fusion test.

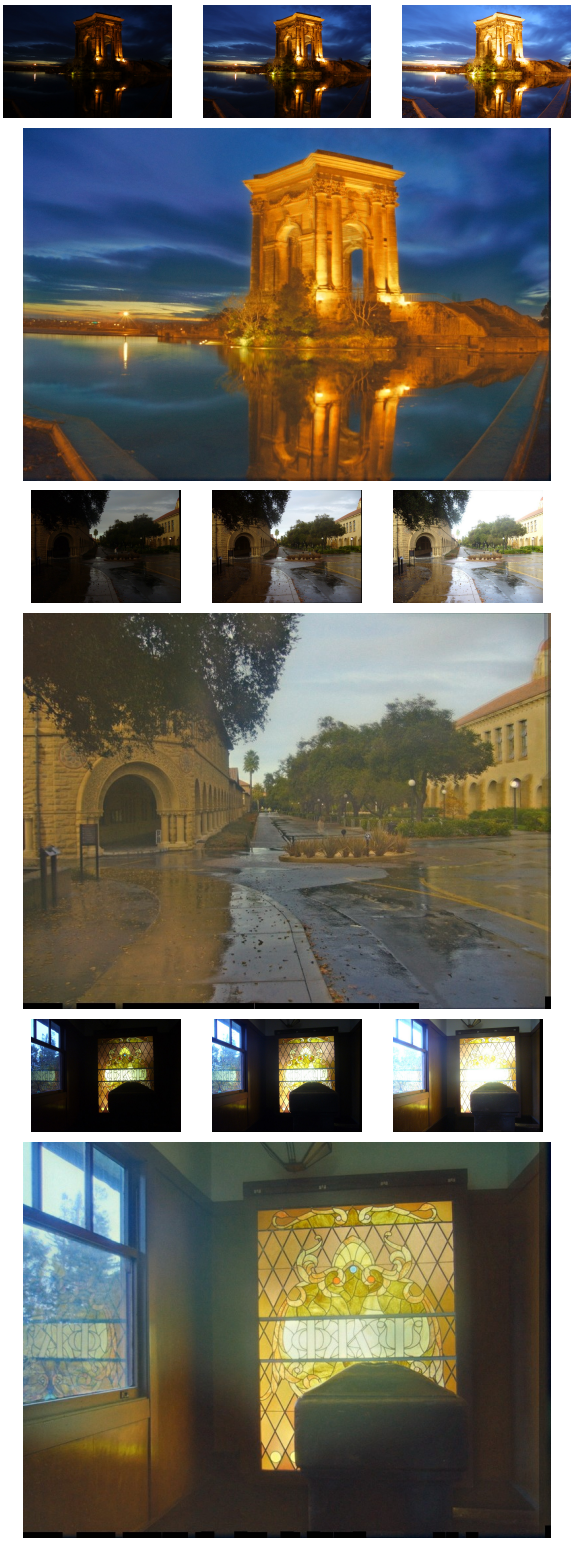


Figure 5: Additional examples of fused images.