User-Driven Image Colorization

Chandani Doshi
MIT
chandani@mit.edu

Lordique Fok
MIT
lfok@mit.edu

Yini Qi
MIT
qyn@mit.edu

Abstract

Image colorization is the process of automating the addition of color to a grayscale image. The process can either be done using user-driven methods that utilize varying amounts of user input to propagate color to the entire image, or data driven methods using neural networks.

In this paper, we describe an implementation of a user-driven method described by Levin et al. [2004]. Using only a few color scribbles generated by the user, the method automatically propagates the colors in space to produce a fully colorized image. The major premise of this algorithm is that neighboring pixels that have similar intensities should have similar colors. Therefore, to implement this algorithm, we use a quadratic cost function to obtain an optimization problem that could be solved with a standard MATLAB solver.

1. Introduction

Colorization is the computerized process of adding color to a grayscale image. The process was invented by Wilson Markle in 1970 to add color to the monochrome footage of the moon from the Apollo mission [Burns]. Markle's process begins with determining the gray level of every object in every shot in a monochrome film print [Markle and Hunt, 1984]. The user would note any movement of objects with shots, and, aided by a computer, add color to each object while keeping gray levels the same as in the monochrome original. Colorists would manually pick the colors for each object in a frame, and the colors were then assigned to other frames based on motion tracking.

A major difficulty with the traditional process of colorization is that it is time-consuming and expensive. To colorize a still image, the user first divides the image into regions, and then assigns a color to each region. Since automatic segmentation algorithms cannot accurately identify complex boundaries such as those between a subject's hair and face, the process of segmentation is labor-intensive. The user must manually specify such complicated boundaries between regions. Colorization of moving images also requires region-tracking to accommodate the apparent movement of objects between frames. Additional difficulty occurs when moving objects are captured by frames; in this case, tracking algorithms fail completely and colorization must be done manually.

Currently, there are two approaches for colorization: user-guided and data-driven. In this paper, we describe the user-guided approach in *Colorization Using Optimization* [Levin et al. 2004], an approach that requires neither precise manual segmentation nor accurate tracking. The goal of this technique is to render a plausible colorization rather than recovering the actual ground truth color. By scribbling the desired colors, the user indicates how each region should be colored, and then the technique automatically propagates colors to the remaining pixels.

1.1. Related Work

Since Markle's invention of computerized colorization, many other algorithms have been created to speed up the process, though many commercial software methods for colorization still rely on defining regions and tracking them between frames.

Recent colorization algorithms primarily fall into two categories: parametric and non-parametric. Parametric methods, such as those described in *Colorful Image Colorization* [Zhang et al. 2016], train on large datasets of color images to learn prediction functions. Zhang's method implements a feed-forward pass in a convolutional neural network that was trained on over a million color images in order to color an image without any user input.

On the other hand, non-parametric methods, also known as user-driven methods, require the user to first create a reference image with some amount of color input based on the grayscale image to be colored.

We decided to implement the non-parametric process developed by Levin et al. in *Colorization Using Optimization* [2004] due to the simplicity and accuracy of the algorithm. Levin et al. describe a process to obtain high quality colorization of images from a modest amount of user inputted color data. It allows the artist to choose the color of each object directly, so that the results can be refined by adding additional color input scribbles if

necessary. The technique proposed in the paper automatically propagates colors to the remaining, non-colored pixels by assuming that the neighboring pixels that have similar gray levels should also have similar colors.

2. Approach

We implemented the algorithm described in section 2 of Levin et al. using MATLAB. First, we converted the input image, which is in *RGB* color space, to an image in *YUV* color space, because that is the format specified for the algorithm. *Y* refers to the monochromatic luminance channel, or the intensity channel, of the image, while *U* and *V* refer to the chrominance channels that contain color information [Jack 2001].

The central basis of the algorithm involves minimizing the overall color differences between neighboring pixels if their intensities are similar. In order to do this, we minimize the local difference between the color at pixel **r** and the weighted average of the colors at its neighboring pixels. For our implementation we interpreted a pixel to be a neighbor if it was directly adjacent to the pixel, or directly diagonal to it. We want to minimize the distance between the color at the pixel **r**, given by U(**r**), and the weighted average of the colors at the neighboring pixels, where U(**s**) is the color at neighbor **s** and w_{rs} is the weight of **s** with respect to **r**.

$$J(U) = \sum_{r} (U(r) - \sum_{s \in N(r)} w_{rs} U(s))^2 \qquad (1)$$

In order to store relative neighbor intensity data for each pixel, we created a method of mapping the row and column indices to the pixel neighbor indices. Because MATLAB does not allow us to store an array in a cell of a matrix, we could not directly store pixel coordinates. Thus, we created a flat vector to store all the relative pixel intensities of each pixel's neighbors. Since there are eight possible neighbors for each pixel, and we included a storage slot for the pixel itself, the first nine elements of the vector correspond to the first pixel and all of its neighbors' relative intensities, the second nine elements correspond to the second pixel and all of its neighbors' relative intensities, and so on.

Next, we needed to construct a matrix A to contain all the relative intensities of the pixels. This matrix had to be stored sparsely, otherwise running the algorithm would be space inefficient. In our sparse matrix, each row **i** represents the pixel **i** in relation to all the other pixels. Thus, A(i,j) is the pairwise affinity between pixel i and j**.** In order to build the sparse matrix, we needed to map the row and column indices to the index of the pixel in question. Therefore, we created the rows and columns vectors to keep track of the weights of the pixels at each index.

To calculate the relative weights of the neighboring pixels, we tried both weighting functions described in the Levin et al. paper.

$$w_{rs} \propto e^{-(Y(r)-Y(s))^2/2\sigma_r^2} \qquad (2)$$

The first weighting function is based on the squared difference between the two intensities.

$$w_{rs} \propto 1 + \frac{1}{\sigma_r^2}(Y(r) - \mu_r)(Y(s) - \mu_r) \qquad (3)$$

The second is based on the normalized correlation between the two intensities where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma_r}$ are the mean and variance of the intensities in a window around **r**.

In the image segmentation algorithm described by Shi and Malik [1997], we need to find the second smallest eigenvector of the matrix *D-W=A,* where D is the diagonal matrix whose diagonal elements are the sum of the affinities (which is always 1 in our case) and W is the matrix of weights. The second smallest eigenvector of a symmetric matrix A is a unit norm vector x that minimizes $x^{\mathrm{T}}Ax$, and is orthogonal to the first eigenvector. Thus, we can solve the equation using the MATLAB "\" command on matrix A with respect to original colors in each channel.

3. Experimental Results

The results shown below were obtained using the correlation based weighting function. When compared to results using the Gaussian based weighting function, no significant differences were found, confirming assertions made in Levin et al. We used MATLAB's built-in least squares solver for sparse linear systems in order to perform the optimization of the key parameter *J(U)* (Equation 1).

Additionally, when we ran the algorithm, we found that the variance values were zero for pixels that were not scribbled with color, resulting in final images in which color was only propagated through these non-zero parts of the image. Thus, we added a thresholding condition to ensure variance was equal to a marginal value if it was too small.

Figure 1: Image rendered when there is no threshold on the variance

In Figure 2 and Figure 3, the first column is the original grayscale image, the second column is the image marked with color scribbles by the user, the third column is the colorized image generated by using the Gaussian based window in the Levin et al. implementation, and the fourth column is the colorized image generated by our implementation.

Figure 4 is an example of progressively refining the colorization of a still image by adding more scribbles. The artist starts with a small number of scribbles, and then adds more scribbles to regions where the color bleeds from one region to another.

From the results, one can see that our implementation causes a larger gradient in the image such that the color leaks outside the boundaries. This is due to the fact that we did not optimize our weighting function and threshold values.

4. Individual Contribution

I researched different ways of implementing the user-driven colorization. After the team decided to use the Levin et al. implementation, I worked on interpreting the Levin et al. algorithm that uses scribbles to colorize images based on the gray levels of the nearby pixels. I developed an understanding of how we can implement the algorithm using the built-in MATLAB least squares solver for sparse linear systems, and worked with Yini on the implementation.

5. Conclusion

Although considerable progress has been made in automatic colorization methods, user-driven colorization still offers considerable benefits over automatic colorization in terms of artistic control of the colors used in the final image. Additionally, user-driven colorization has the added benefit of giving consistent results across many image types, while state-of-the-art automatic colorization methods only work consistently on images that are in their finite training set. In particular, the algorithm described above allows users to give relatively small amounts of input when compared to other user-driven colorization algorithms.

Our implementation of the colorization algorithm successfully propagates sparse user color input throughout an image. However, our implementation requires more color input close to the edge of an object to achieve results similar to those presented in Levin et al [2004]. This was due to differences in optimization.

In the future, further progress can be made by optimizing the weighting function and the threshold values in order to ensure tighter color boundaries. We can also speed up the algorithm by using a multi-grid solver to give a faster (approximated) solution, as described in section 3 of Levin et al. [2004]. Additionally, further work could be done with regards to verifying and optimizing Levin et al.'s method of applying user-driven colorizations to frames of videos, and having the color input propagate through the video frames.

References

[1] BURNS, G. Colorization. Museum of Broadcast Communications: Encyclopedia of Television, http://www.museum.tv/archives/etv/index.html.

[2] JACK, K. 2001. *Video Demystified,* 3rd edition ed. Elsevier Science & Technology.WEISS, Y. 1999. Segmentation using eigenvectors: A unifying view. In *Proceedings ICCV, 975-982.*

[3] LEVIN, A., LISCHINSKI, D., WEISS, Y. 2004. Colorization using Optimization. In *Proceeding SIGGRAPH, 689-694.*

[4] MARKLE, W., AND HUNT, B., 1987. Coloring a black and white signal using motion detection. Canadian patent no. 1291260, Dec.

[5] SHI, J., AND MALIK, J. 1997. Normalized cuts and image segmentation. In Proc. IEEE Conf. Computer Vision and Pattern Recognition, 731–737.

[6] ZHANG, R., ISOLA, P., EFROS, A. 2016. Colorful Image Colorization. *arXiv 1603.08511.*

Figure 2: Images from Levin et al. [2004]. From left to right: original grayscale image, image with scribbles, colorized image generated by Levin et al. implementation, colorized image generated by our implementation

Figure 3: Our images. From left to right: same as above

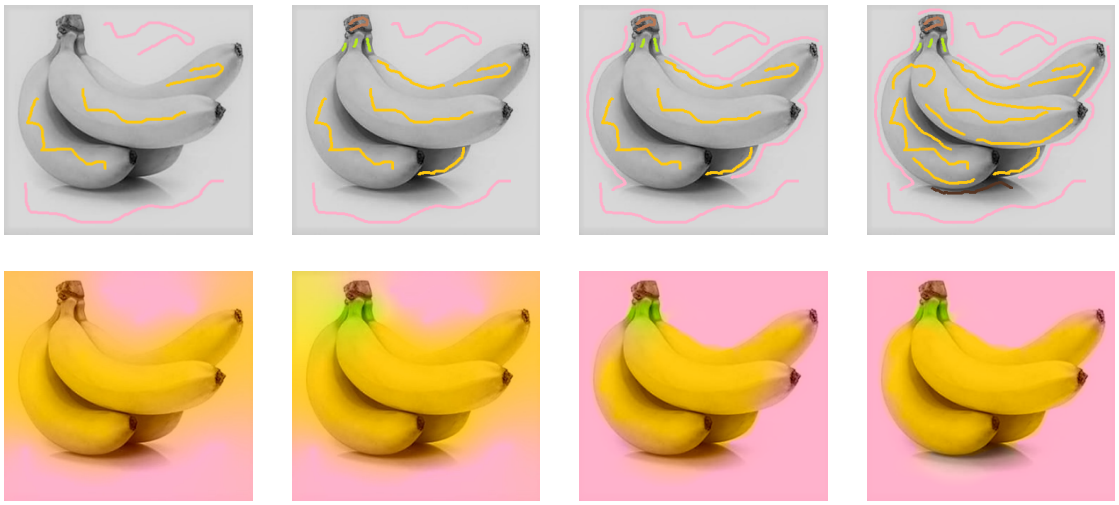

Figure 4: Progressive refinement of coloring by adding more scribbles