From Grayscale to Color: Digital Image Colorization using Machine Learning

Cris Zanoci and Jim Andress

November 15, 2015

1 Introduction

Image colorization is the process of adding colors to a grayscale picture using as a source a colored image with similar content. Colorization techniques are widely used is astronomy, MRI scans, and black-and-white image restoration. Given a grayscale image, there is no unique correct colorization in the absence of any information from the user. Even though many implementations require the user to specify initial colors in certain regions of the picture, our project focuses on automatic image colorization, without any additional input from the user. In this report we describe our first attempts at colorizing a gray image using a similar colored picture. We begin by implementing a simplistic method that transfers colors between pixels based on the similarity in their luminosity. We improve our approach by considering a larger feature space, that takes into account the provenance of each pixel and strives for spatial consistency. Then we apply PCA and KNN in the feature space, followed by a confidence voting that labels our gray pixels and determines from which colored pixels we have to transfer the chromatic channels.

2 Methods

2.1 Baseline

In our first attempt to transfer color to the gray-scale image we follow a 2002 paper by Welsh [1]. Both the training image and the test image are converted to the $l\alpha\beta$ color space, where l represents the luminance of a pixel, and $\alpha\beta$ are the two components of its color. We chose this space primarily because the correlation between its three components is minimal, thus allowing us to decouple the luminance from the chromatic channels. In order to assign a color to one of the pixels q in the gray image, we search for a pixel p in the colored picture whose luminance value and neighborhood luminance are most similar to those of q. The pixel whose color we transfer to q is chosen based on the weighted average of luminance and standard deviation in luminance values of a 5×5 square centered at p. Once a match is found, the α and β components are transferred to the gray pixel, while its original luminance is kept intact.

To avoid the expensive search over all the pixels in the training image, we sample it using a grid of equally spaced pixels. With a small enough distance between pixels in the grid, we can make sure that we sample all the regions of different intensity in the picture. Approximately 1% of the image was sampled. We also performed a linear scaling of luminances in the two pictures to account for global differences in luminance.

2.2 DCT-Based Features, kNN with Image Space Voting

The baseline classifier described in [1] and in Section 2.1 represents each pixel with only two values, the luminance at that point and the standard deviation of luminance in that pixel's neighborhood. However, these simple features are not nearly descriptive enough to accurately train a model that distinguishes between various textures. For this reason, we next decided to adopt the approach of Irony et. al and use a higher dimensional feature space while describing the image luminance textures [2]. Rather than storing two values, we now store the k^2 Discrete Cosine Transformation (DCT) coefficients generated over a $k \times k$ neighborhood of a given pixel. We chose DCT over other Fourier-related transforms because its representation consists only of real coefficients. These coefficients make informative and robust texture features since they compactly store most of the significant spectral information about the picture.

The algorithm begins by segmenting the training image into a series of disjoint labels. Again, we sample a grid of equally spaced pixels from the training image to decrease the computational cost of building and querying our model. Once the pixels have been sampled, we can then apply the function f to each where $f(x)$ represents the k^2 DCT coefficients computed in a $k \times k$ neighborhood around the pixel x.

At this point, each pixel is represented by a vector in $\mathbb{R}^{k^2}$ space, which has two key issues. First, we can fall prey to the curse of dimensionality since the representations of the pixels will become more spread out as k increases. In addition, although the feature space contains many points from each of the labels, there is no guarantee that these points are tightly clustered within the k^2 dimensional space. This is an issue since it would mean some of our classifier's predictive strength would be wasted trying to predict patterns within a specific label.

In order to address both of these issues, we apply PCA feature selection in two stages. To begin, we first attempt to decrease the spread within each label. To do so, we randomly sample from the training pixels' intra-differences (an intra-difference is defined as the difference between two vectors with the same label), perform PCA, and then throw out all but the d' eigenvectors associated with the smallest eigenvalues. This results in a transformation $T_{intra} : \mathbb{R}^{k^2} \to \mathbb{R}^{d'}$ which minimizes the spread of the images of each label. We run the training data through this projection, randomly sample from the transformed training pixels' inter-differences (an inter-difference is defined as the difference between two vectors with different labels), perform PCA, and then throw out all but the d eigenvectors associated with the largest eigenvalues. This then results in a transformation $T_{inter} : \mathbb{R}^{d'} \to \mathbb{R}^{d}$ which maximizes the spread between the images of each label. Finally, we can define a final feature mapping on a pixel x by $T(x) = T_{inter}(T_{intra}(f(x))) \in \mathbb{R}^d$.

Once we have selected the feature space for our learning algorithm, we then use it in a K-Nearest Neighbors classifier to predict a label for each of the pixels in the test image. However, this classifier is not able to accurately divide the test image into labeled regions since it operates solely on a per-pixel level. This means that it tries to optimize each pixel without considering how that pixel fits into the image as a whole. For this reason, the raw kNN predictions are only the first stage in the labeling process. Once we have computed a locally optimal label for each of the pixels, we then choose what Irony calls the dominant label $\ell^*(p)$ for a pixel p by computing

$$\ell^*(p) = \arg\max_{\ell} \text{conf}_p(\ell) = \frac{\sum_{q \in N(p,\ell)} W_q}{\sum_{r \in N(p)} W_r} \tag{1}$$

where $N(p)$ is a $k \times k$ neighborhood around p, $N(p, \ell)$ is the set of pixels in $N(p)$ which were predicted to have the label ℓ,

$$W_q = \frac{\exp(-D(T(q), T(M_q))}{\sum_{r \in N(q)} \exp(-D(T(r), T(M_r)))}, \tag{2}$$

M_q is the nearest training example to q which has the same label as q, and $D(\cdot, \cdot)$ is the Euclidean distance function in $\mathbb{R}^d$ space.

Once the dominant label has been computed, we can compute the two chromatic channel values $C(p)$ by determining the neighbors of p which have p's predicted label, determining the closest trainig pixel to each of these neighbors, and then taking a weighted average of the chromatic channels for these training pixels

$$C(p) = \frac{\sum_{q \in N(p,\ell*(p))} W_q C(M_q)}{\sum_{q \in N(p,\ell*(p))} W_q} \tag{3}$$

3 Results

The results of our baseline implementation are presented in Fig. 1. Even though we managed to transfer the color to most of the vegetation and the sky, the upper right corner of the sky and some parts of the mountain are miscolored. This defect and the long overall execution time motivated us to search for other methods.

For the second approach, we use 7×7 square to extract information about the neighborhood of each pixel. This results in a 49-dimensional feature space, which later gets reduced using PCA to 20, and then 10 dimensions. We use a grid that samples roughly 4% of the colored picture.

Figure 2 shows the application of the algorithm described in section 2.2. Note that even though many of the points inside the elephant's body were attributed the wrong label (c), our voting procedure relabels those points to achieve a consistent texture (d). Figure (e) shows that we are most confident in the labels attributed to the sky. This makes perfect sense, since the majority of the training image was the blue sky and hence it will make up most of the colorized image. Figure 3 demonstrates the improvement of our second approach over the naive baseline.

4 Future Work

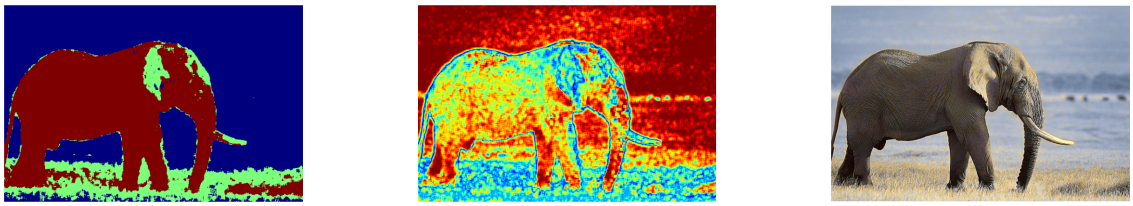
Since the above method clearly has its shortcomings, we will investigate more novel techniques, like Parzen windows, structured-SVMs, Markov Random Fields and maybe neural networks.

5 Appendix

(**a**) (**b**) (**c**)

Figure 1: The baseline colorization procedure. (**a**) A colored training image. (**b**) A gray-scale test image. (**c**) A colorized version of the test image.

(**a**) (**b**) (**c**)



(**d**) (**e**) (**f**)

Figure 2: The more complicated colorization procedure. (**a**) A colored training image. (**b**) The test image with the orginal color. (**c**) The results of running kNN on the test image. (**d**) The results of image-space voting on the kNN results. (**e**) The confidence of each pixel's prediction. (**f**) The colorized version of the test image.

(**a**) (**b**)

Figure 3: A comparison of our baseline and more complicated colorization results. (**a**) The baseline colorization. (**b**) Our more complex colorization.

References

[1] T. Welsh, M. Ashikhmin, and K. Mueller, "Transferring Color to Greyscale Images," 2002.

[2] R. Irony, D. Cohen-Or, and D. Lischinski, "Colorization by Example," *Eurographics Symposium on Rendering*, 2009.