Colorization of Grayscale Images with Machine Learning

by Vashisht Madhavan
CS194-26 Fall 2015 Final Project

Introduction

The aim of this project is to automatically color grayscale images using machine learning methods. There is extensive literature on using optimization methods or user-guided input to colorize an image, but the approach in this project recreates the results found by Guillaume Charpiat & co. The paper illustrates the use of kernelized SVMs and Graph Cut Optimization to provide accurate colorization. By extracting features from pixels in similar images, the model is able to assign aesthetically-believable colors to a given grayscale image. The details of the process are outlined in later sections.

Algorithm Overview

As with any ML experiment, the first step is getting data and extracting features and labels from it. In this case, since the task is to predict colors, I chose a subset of colors as my labels and created a high dimensional feature vector for each pixel, which captures information about what color the pixel is and what part of an image the pixel represents. Once I had features for each training image, I trained the SVMs and optimized hyperparameters via a validation set. Finally, I used markov random fields and graph cut optimization techniques to color a given grayscale image. More details about this process are outlined below.

1. Lab Space and Color Discretization

Before I extracted labels for each pixel, I first converted each training image to Lab space. This basically breaks each pixel down into components of luminance(L), which is essentially how light or dark the pixel is, and (a,b), a vector representing all colors of the spectrum. Lab space better approximates the way humans perceive color. It also separates the color lightness from its hue, which essentially means that different shades of red would be treated the same if we just looked at (a,b) values. This is optimal because it allows our predictive model to be robust to shades of colors.

With the (a,b) pairs as the color labels, there are 256 possible values for both a and b, making the class label space much too large for efficient prediction. Naturally, I decided to condense the color space with K-means clustering on (a,b) values. In the experiments run, I varied k between 16 and 32 and selected a value based on hyper-parameter optimization. Once these k colors were found, the color space of the feature matrix was discretized, by mapping original (a,b) values to the closest color in the k-color space.


left - original color image; right - discretized color image with k = 10

2. Features and Labels

The feature matrix was constructed by extracting the following from each pixel:

- Local SURF feature descriptors at 3 different scales
- Local mean and variance
- Local FFT magnitude

The features described above are all taken from Charpiat's experiments in the paper. SURF is very similar to the SIFT features discussed in lecture, except that SURF is much faster to compute and more robust to image transformations. The FFT magnitude and local mean and variance are supplementary, that ,according to Charpiat, are "biologically inspired" and apparently work well empirically.

Although Charpiat suggests 5x5 windows for each of the localized feature descriptors, I was able to evaluate window sizes of 5,10, and 20 and decided to go with a 20x20 window. This resulted in a very high-dimensional feature vector per pixel(786 dimensions). To avoid the curse of dimensionality, I used PCA to reduce dimensionality to between 32 and 64 features. The specific number of components to keep would be found during validation/hyper-parameter optimization.

Since the experiments I ran involved images that are 313 x 500, each image would produce 400,000 training points. To reduce the computation time, I sampled about 13% (20,000) of pixels from each image, which produced fairly accurate results.

3. SVM Classification

To predict colors for each pixel, I used sklearn's SVC module.

For each color class, I trained a 1 vs. all SVM classifier, which would predict 1 if the pixel is that color and 0 otherwise. Once these SVMs were trained, I passed in the feature set from a grayscale image and obtained classification scores from each model. Since each pixel may contain one, multiple, or no colors, it is more important to consider the conditional probability of a color given a pixel. Since SVMs are inherently non probabilistic, I needed to find another way to get scores for each classifier.

Although other classification methods, such as logistic regression, provide probability scores, it is tough to integrate the kernel trick with the models that already exist in the sklearn library. Alternatively, the SVM class in sklearn does provide a probability score option, but Charpiat argues that the margins computed by the SVMs are a suitable proxy for these probabilities. Thus, we use margins to compute the best color at a given pixel.

4. Graph Cuts and Colorization

After training SVMs, I was able to extract color labels for each pixel in a grayscale image. However, this simple prediction isn't enough. According to Charpiat, "in order to get spatially coherent colorization the solution should be computed globally, since any local search can yield suboptimal result". To incorporate this global information, I created Markov Random Field with nodes as the predicted color labels and edges proportional to euclidean color distance in Lab space. MRFs are suitable to this and other vision problems mainly because,for a given pixel, they incorporate information from far away pixels while also giving much more weight to local pixels. Then, as the paper suggests, I ran Graph Cut Optimization with the pygco library to get minimum energy pixel labelings. The equation is described below:

$$\sum_{\mathbf{p}} V_{\mathbf{p}}(c(\mathbf{p})) \; + \; \rho \sum_{\mathbf{p} \sim \mathbf{q}} \frac{|c(\mathbf{p}) - c(\mathbf{q})|_{Lab}}{g_{\mathbf{p},\mathbf{q}}},$$

The V function here is the "cost of choosing pixel color C(p) for pixel p", which in the case of SVMs is the negative probability score of a class. Since we are using margins for saving computation time, we set V to be the negative margin. ρ is the hyper-parameter that tradeoff the weight of local color scores to spatial coherence scores. It is estimated with cross validation. q corresponds to pixels neighboring p and the variable,g, is the harmonic mean of estimated color variations for pixels p and q.

Finally, I applied median filter to the image to reduce the effect of noise and outliers in the colors.

5. Validation and Parameter Search

In steps 1-3 there are a number of hyper-parameters that can be tuned by evaluating error on the validation set. Here is a list:

- K for K-Means to discretize colors
- PCA components
- SVM parameters
- Window size for localized features
- ρ for graph cuts

Since performing a full grid search over these hyper-parameters would be very computationally expensive, I chose a small set of values for each hyper-parameter and computed the validation error for different combinations of parameter values.

For each image in the validation set, I converted the color image to grayscale and colorized it with the trained model for a given set of parameter values. Once I had the actual colored image and the image colorized by the algorithm, I calculated the MSE over the (a,b) pairs in both images. I ran this for all combinations of hyper-parameter values and simply took the set with lowest MSE.

I used the MSE between the (a,b) pairs as my error metric, as it captures the euclidean color distance between the two images, disregarding the lightness of either image.

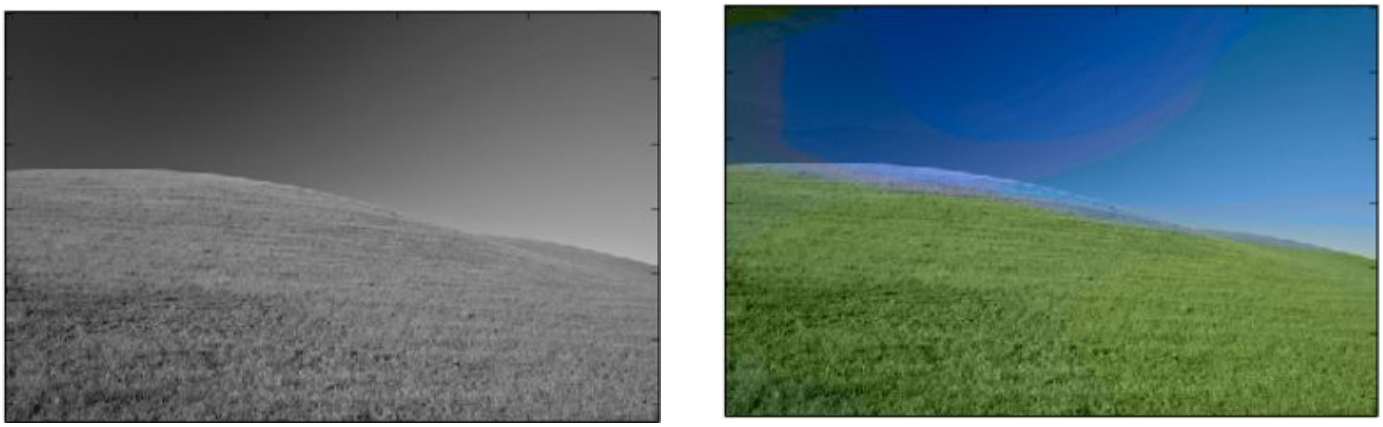
Results

The first experiment I ran was to color an image based on a set of related but different images. For this example, I used pictures of grassy plains with two training images. Here are the results

Two training images for the first experiment

The parameters found through minimizing the validation error are as follows:

- k = 35
- pca size = 40
- SVM gamma = .03
- SVM C = 1.0
- $\rho = 1.0$



The colorization on the right is not perfect, but the major regions of the image are colored correctly

These all have image sizes of 500x800 and we sample 20,000 pixels from each training image.

The next experiment I ran involved transferring colors between similar objects. Essentially, I take an image of a specific object, in this case a puppy, and use its color to colorize a similar image with slightly different color than the training image.

On the left we have a brown golden retriever and on the right is a blonde retriever, to which we will transfer the brown retrievers color

The parameters found through minimizing the validation error are as follows:

- k = 8
- pca size = 40
- SVM gamma = .25
- SVM C = .5
- $\rho = 1.0$


For the most part the results look pretty good. The puppy definitely has a more golden tone to it and the grass in the foreground is colored correctly. The back part of the image has some discoloration, due to the blurriness of the background of both images. There is also some color bleeding around the edges of the puppy, similar to the plains picture above.

The final example I chose was similar to the zebra colorization example in the image. I chose a picture of a tiger as an example and basically used the image's pixels to color a grayscale image of a tiger in a slightly different setting. Here are the results:


On the left we have the training image and the right is the grayscale image we are coloring

The parameters after tuning are:

- k = 10
- pca size = 64
- SVM gamma = .25
- SVM C = .5
- $\rho = 1.0$

Conclusion

Although the colorizations for the images are not perfect,for the most part they do pretty well. Especially considering that we only sample about 5-10% of the pixels in each training image. One way I could improve the results is by choosing higher definition images, which in turn would contain a larger number of pixels. Since the colorization process is computationally expensive and takes about 5 minutes to colorize even small images, I only used small, low-definition images in the interest of time and computation. Additionally, selecting good images for colorization was also a big challenge. I needed to find images that had the spatial positioning of objects and captured the same window of the object in question. Overall this problem was very interesting and challenging, since it involved aspects of computational photography and machine learning. To expand upon this project, I would try adding additional features to the feature matrix and try colorizing complex scenes with multiple objects. Another interesting challenge would be to parallelize this computation with a framework like Spark to speed things up and effectively compute colors for high-definition images.

References

1. G Charpiat. *Machine Learning Methods for Automatic Image Colorization* … , October 2009.
2. Y Boykov, O Veksler, and R Zabih. *Fast Approximate Energy Minimization via Graph Cuts*. Pattern Analysis and Machine …, 2001
3. A Sousa, R Kabirzadeh, P Blaes. *Automatic Colorization of Grayscale Images.* 2013
4. R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. F. Tappen, and C. Rother, *"A comparative study of energy minimization methods for markov random fields*," in ECCV (2), 9th European Conference on Computer Vision, Proceedings, Part II (A. Leonardis, H. Bischof, and A. Pinz, eds.), vol. 3952 of Lecture Notes in Computer Science, (Graz, Austria), pp. 16–29, Springer, May 2006.
5. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0809/ORCHARD/