Automatic Colorization of Grayscale Images

Austin Sousa Rasoul Kabirzadeh Patrick Blaes
Department of Electrical Engineering, Stanford University

1 Introduction

There exists a wealth of photographic images, from antique photography to low-resolution video, which lack color information. Assigning color to a black-and-white image is a highly ill-posed problem; given a certain image, there is often no "correct" attainable color. For example, a red balloon is indistinguishable from a blue balloon when photographed in black-and-white. Due to the indeterminate nature of the problem, image colorization techniques currently rely heavily on human interaction. Several vintage movies have been colorized entirely by hand, requiring every single frame to be inspected at great financial and time cost[6]. Here we present a method, using machine learning techniques, to assign aesthetically-believable colors to black-and-white photographs, using a colormap selected from a similar training image. The applications of such a method allow for a new appreciation of old, black and white photographs and cinema, along with allowing better interpretation of modern grayscale images such as those from CCTV cameras, astronomical photography, or electron microscopy.

2 Description of Method

Current literature suggests two classes of colorization methods – one in which a color is assigned to an individual pixel based on its intensity as learned from a color image with similar content, and another in which the image is segmented into regions, each of which are then assigned a single hue. Our method uses the former approach: we analyze the color content of a provided training image, and then attempt to predict colorization for a single target grayscale image on a per-pixel basis. While segmentation techniques have intrinsic appeal, and could result in more-consistent colorization, these methods rely on accurate image segmentation, which can be thrown off by shadows, texture, or color gradients. Furthermore, methods require subsequent manual identification of the objects in a scene – couch, chair, etc. – for which training color sets are then selected.

Figure 1 shows block diagrams of the training and colorizing processes. Training consists of (a) transforming and discretizing the color space, (b) selecting a subset of pixels to train on, c) extracting a set of features from each pixel using the luminescence channel, and (d) training a set of binary classifiers – one per each color. Subsequently, colorizing a grayscale image consists of (a) extracting the same set of features from each pixel, (b) estimating the probability for each color at each pixel, and c) selecting the color with the highest likelihood.

Once probabilities have been assigned to each pixel, we then apply a graphcut algorithm, which attempts to align color variation to intensity boundaries. We then apply a simple smoothing function to further reduce color discontinuities. Finally, the colormap is transformed back into RGB space, resulting in a final image.

Our method is implemented in Python 2.7, using the numpy, scikit-learn, and OpenCV libraries. Each of the processing steps is described below.

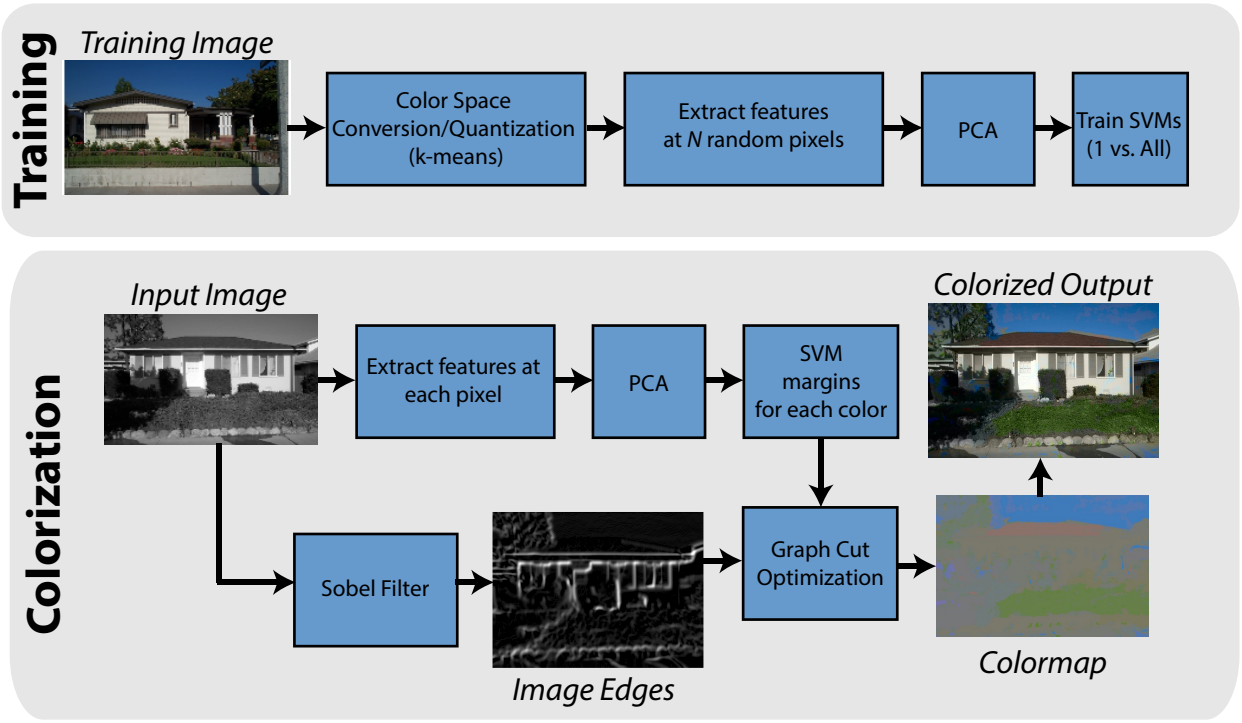

Figure 1: Here we show flowcharts illustrating the steps taken in our training and colorization pipelines.

2.1 Color-Space Discretization

Each image in the set of training files is converted to the Lab (Luminance, a, b) color space. This is used over RGB because euclidean distance in Lab is more similar to how humans perceive color differences. The Luminance channel becomes the grayscale image from which features are extracted. Color information is represented via the two 8-bit values (a,b), allowing for 256^2 possible hues. We then reduce the color space via k-means to a manageable subset of colors – typically ranging between 8 and 32 – and store this color mapping for use by the final output step. Once the reduced colormap is selected, we quantize the training image or images, and randomly select a subset of training pixels – typically 5000 pixels from each 640x480 pixel image, or about 1.6%.

2.2 Feature Extraction

At each pixel, we extract three parameter classes: SURF (Speeded Up Robust Features), FFT, and localized mean and variance. SURF descriptors for each pixel are extracted over three octaves, using a 20x20 window. To this we add the magnitude of the 2D FFT over the same 20x20 grid. Finally we compute the mean and variance, resulting in an unwieldy 786-point feature vector. When training, we then reduce the set of features from 786 points to 32 via PCA.

2.3 Classification

Our classification system consists of a set of support vector machines, one per each bin of the discretized color space, using a Gaussian kernel. This array of SVMs performs a binary classification for each color bin, predicting whether a pixel is or is not the corresponding color. This classification style is known as

One vs. All multilabel classification. It should be noted, however, that in colorizing, we do not use the simple true / false output of the SVMs, due to the inherent possibility of selecting more than one color, or no color at all. Instead, we use the margin of each SVM as a proxy for the probability of each color being the true color. These probabilities are then used in post-processing by the graphcut algorithm.

2.4 Colorization Process

Colorizing begins by computing the same 786-point feature vector for each pixel in the grayscale image. These features are then projected onto the lower-dimensional basis computed using PCA in the training step. Each reduced feature vector is then passed to the set of SVMs from which we obtain the geometric margin of the feature vector for each possible color. These margins are used as a proxy for the conditional probability of a given color given the local features. In order to include more global information, we model the image as a Markov Random Field in which nodes are given color labels and edge weights are proportional to the image gradients. The Graph Cut algorithm is used to find the minimum energy pixel labeling according to the following energy function at each pixel i

$$E_i = -\sum_j s_{c_j}(v_i) + \sum_{j \in \mathcal{N}(i)} g_j ||c_i - c_j||_2$$

In the first summation, $s_{c_j}(v_i)$ is the geometric margin of the feature vector at pixel i, v_i, for color classifier c_j. The second summation is over all neighbor pixels of i and includes the euclidean distance in color space and gradient magnitude g_j. This has the effect of "snapping" colored regions to edges in the image. We used the Graph Cut implementation provided by the authors of [5]. A median filter is then used on the output to remove small patches of outlier colors in the image.

2.5 Parameter Optimization

For most images, there is a continuum of equally-appropriate colors. As such, it is difficult to develop an error metric that accurately indicates a pleasing image from an ugly one. For purposes of parameter tuning, however, we incorporate a simple metric based on an image's true color. We begin by converting a color image to Lab, and quantize its color channels using the k-means color map selected in training. We then predict a new colormap via our algorithm, and define the colorization error as the average Euclidean distance between the predicted and quantized true-color colormaps. Several parameters were considered to be optimized – the number of color bins; the number of vectors kept by PCA; the standard deviation of the Gaussian kernel and SVM regularization parameter; the number of training examples; and the SURF window size. We performed several grid searches on various test images, using a compute cluster and MPI. In most cases the minimum colorization error seemed to agree with the best-looking case; however the selected parameters varied from image to image, and failed to show obvious trends.

3 Results

Our algorithm performs surprisingly well given the ill-posed nature of the problem. Figure 2 shows example runs of the algorithm for different images. The general colors of the objects are all present, and shows promise that realistic results are achievable. However, numerous improvements can be made.

Currently our algorithm does reasonably well based on the features obtained from localized information surrounding individual pixels. However, aside from graphcut minimization, we have not

explicitly used any features that contain global information about the image. The algorithm therefore cannot distinguish between pixels that locally look similar but globally are from different objects. For example, in the landscape scene, (figure 2c) our algorithm had difficulty differentiating flat mountain faces from sky with slight clouds.

A simple way to incorporate global features would be to segment the image into broad regions such as foreground and background, and then include this assignment as an additional feature. Alternatively, each segmented region of the image could be colorized independently, recombined, and processed via graphcut.

For video colorization, we can provide a training image to the algorithm for the first shot of the video, and then propagate colors to the subsequent frames. Our initial attempts show promise; however the algorithm lacks consistency from frame to frame. An improvement would be to incorporate some regularization between adjacent frames. Similarly, to limit the propagation of the errors to subsequent frames, we could retrain the algorithm on some time interval, or when the scene changes.

4 Conclusion

Our method shows promise in accurately applying color to grayscale images, and could likely be expanded to assist in the colorization of black-and-white films. The entire process of training and colorizing an 800x600 image takes between 30 and 45 minutes on a modern single-core processor. Since much of the processing is independent for each pixel, the algorithm could easily be parallelized to run on a compute cluster.

Our project code is available here: `https://github.com/prblaes/ImageColorization`

References

[1] Y Boykov, O Veksler, and R Zabih. Fast Approximate Energy Minimization via Graph Cuts. *Pattern Analysis and Machine …*, 2001.

[2] A Bugeau and V T Ta. Patch-based Image Colorization. *Pattern Recognition (ICPR)*, 2012.

[3] A Bugeau, V T Ta, and N Papadakis. Variational Exemplar-based Image Colorization. pages 1–9, March 2013.

[4] G Charpiat. Machine Learning Methods for Automatic Image Colorization. pages 1–27, October 2009.

[5] A Delong, A Osokin, H N Isack, and Y Boykov. Fast approximate energy minimization with label costs. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2173–2180, 2010.

[6] S Liu and X Zhang. Automatic grayscale image colorization using histogram regression. *Pattern Recognition Letters*, 2012.

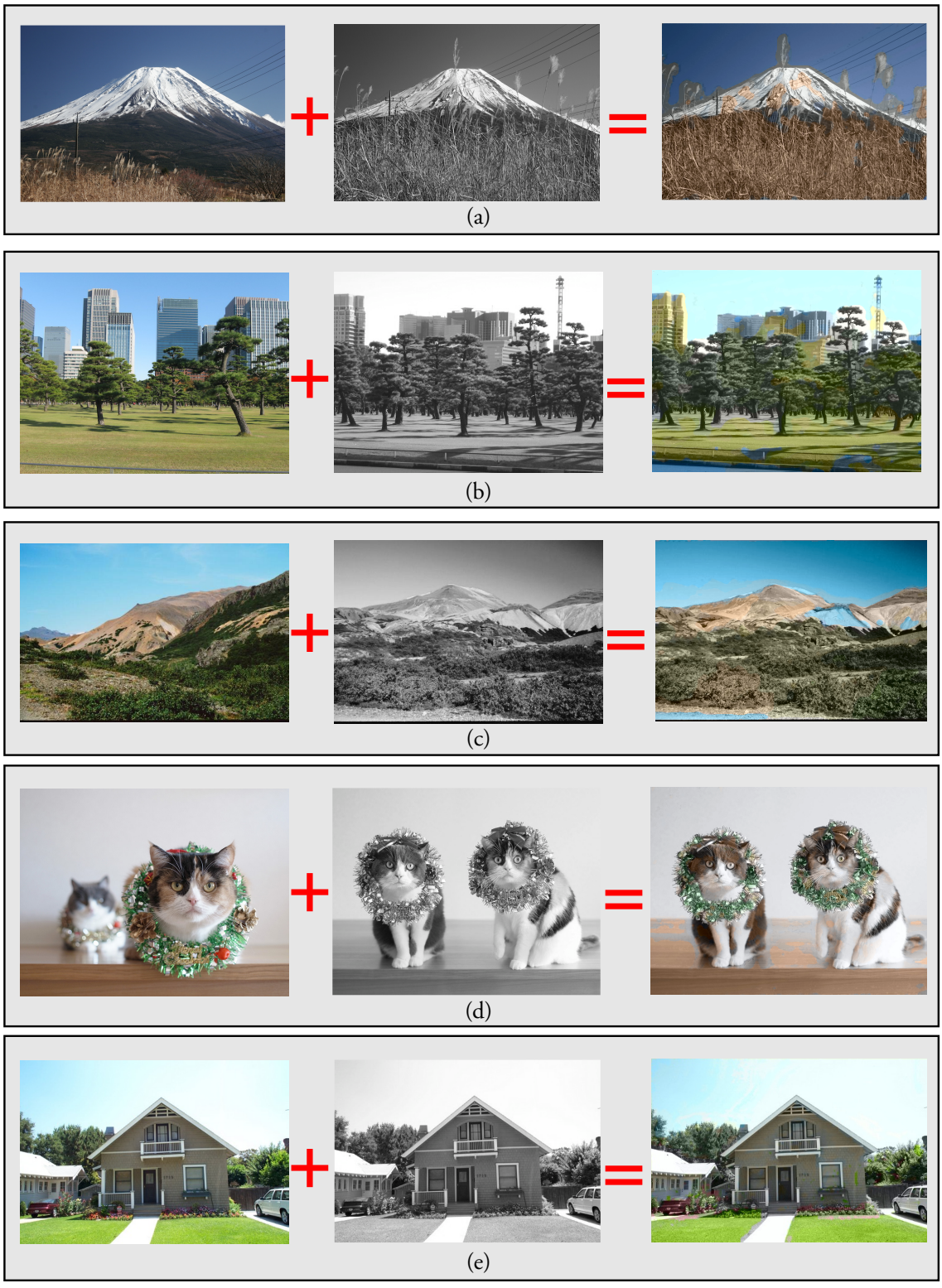

Figure 2: Some select examples using our colorization method. Panels (a)-(c) show reasonable performance on landscape scenes. We see, however, that in regions with few texture features such as the mountains in (c), the algorithm becomes confused and colors them as if they were part of the sky. Panel (d) shows good performance on animal subjects. Finally, in panel (e) we colorize a grayscale version of the same image that was trained upon. We see almost no error in this case even though we only trained on 2% of the pixels.