Image Colorization

Garrett Gordon
Massachusetts Institute of Technology
Cambridge, MA
gagordon@mit.edu

Abstract

In this project, I will be developing a user-assisted algorithm that colorizes black and white photos. This is a much researched topic in computer science, and has been a subject of interest since the advent of color photos. Many modern techniques have arisen, some relying less on human input than my approach, and some not at all. My solution involves taking a black and white image with human-inputted color scribbles over regions with seemingly similar color characteristics, and automatically colorizing the entire photo. I do this by splitting the problem into region-detection, then colorization, altering the human-inputted color accordingly based on intensity. My solution is ideal in scenarios where the original colors of the black and white photos are known by the user, and accuracy is more important than automation of the task.

1. Introduction

Colorization is something that we've grown accustomed to in our daily lives. Whether it be restorations of our favorite classic films, or seeing old photos of relatives in color for the first time, the digital revolution has allowed us to bring the past to life. Early attempts at colorization were quite rigorous, and required many hours of labor to colorize photos by hand [5]. With modern digital image processing, this has all but become a thing of the past.

In this paper, I will detail a technique of colorization, that takes in user input to colorize a black and white image. This method works under the assumption that within a photo, there are a number of connected regions. Within each connected region, pixels vary in brightness rather than color. In other words, each pixel's color within such a connected region lies on some monochromatic scale that is fixed. This allows us to pick a color for each region with user input, then define a function that scales that color according to the [0-255] brightness value in the grayscale image.

The process from end to end is:

- A user takes a grayscale image and scribbles colors on what they see as each contiguous sub-region of the image.
- We run the Canny Edge-Detection algorithm to divide the grayscale image into each sub-region.
- We determine the inputted color for each sub-region.
- We scale pixels of each sub-region according to their intensity in the original black and white image.

I will compare the results of this project with various methods, including both user-assisted colorization techniques as well as automatic, trained neural network techniques.

2. Related Work

Image colorization has been solved in a number of ways in the past. I based my approach off of several papers describing methods of using user scribbles to colorize an image.

2.1. Laplacian Coordinates

In recent work by Wallace Casaca *et al.* [3], images are segmented with laplacian coordinates, and user color inputs are propagated throughout the image according to their intensity in the input black and white image. My approach closely follows this structure but swaps the laplacian coordinates technique for Canny's Edge Detection Algorithm.

2.2. Canny's Algorithm

In past research by Dongdong Nie *et al.* [2], the input image is segmented with Canny's Algorithm. The colorization aspect takes user color input and outputs pixel intensity of sub-regions based on the minimization of a loss function. I combine both approaches and attempt to improve upon the colorization side of both Nie and Casaca's approaches.

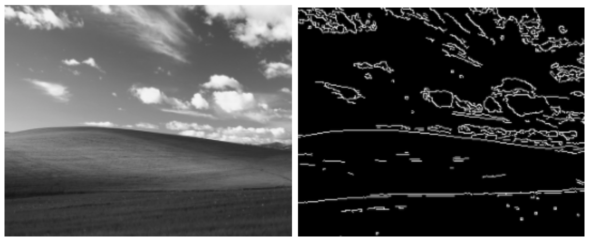
Figure 1. Bliss (Windows XP default background) before and after Canny's edge detection algorithm is applied.

2.3. Convolutional Neural Networks

An alternative to approaches with user-input involve CNNs trained on massive image databases that can colorize images without user input. What they sometimes lack in precise adherence to the correct color, they make up for in speed and minimal reliance on user input. In Yili Zhao *et al.*'s recent work, this process is laid out in detail [4]. A training set of 147 GB and over 1.2 million images was used to train their network. Experimental results show slight differences in color, reflecting the model's inability to exactly match the ground truth color, which is very difficult without prior knowledge of the color image. However the results almost always appear natural and realistic. I decided to not proceed with this approach to maximize adherence to the actual color, something that is easier to achieve with user assistance.

3. Approach

3.1. Region Detection

The first step in the proposed approach is splitting an input grayscale image into distinct contiguous regions with similar characteristics. The overall approach relies on the assumption that these contiguous areas in an grayscale image will have similar color characteristics. Canny's Edge Detection Algorithm is used to find edges in an image. This proves to be incredibly useful in detecting the regions of interest within an image. Canny's Algorithm works by first smoothing the image with a gaussian filter to remove noise. This aids in only detecting edges of large patterns in the image not minute details. The algorithm uses four filters to detect horizontal, vertical and diagonal edges and come to the edge gradient and direction [1].

The algorithm outputs a black and white image where white pixels represent edges. Figure 1 shows a visual example of this algorithm. I apply a gaussian filter after the edge detection to blur the edges and reduce some noise. Once we have this blurred edge image, we can go about finding each distinct region. For a pre-determined number of iterations, (can be changed depending on image) we select a pixel and run a function $get\_region(region, x, y)$. This function adds the current pixel to the region if it is not white, then recursively proceeds on each neighboring pixel that is not already in the region. If the pixel is white, it is added to the region, but no further recursive calls to the function are made. This effectively adds all pixels contained within some set of edges into one region set. We continue picking from the remaining pixels and running this algorithm until we have reached the number of iterations that were defined or there are no pixels left to be picked. Now an image R has been segmented into n contiguous sets of pixel coordinates $r_1 \cup r_2 \cup \cdots r_n = R$. Figure 2 shows what some of these regions look like for one of my test images.

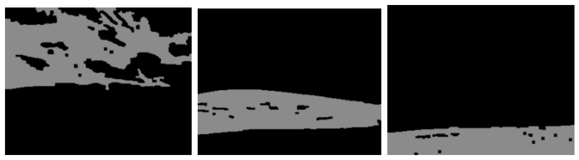
Figure 2. Some highlighted regions for Bliss (Windows XP default background) from Figure 1

3.2. Color Detection

Now that we have our sets of pixel coordinates $r_1 \cup r_2 \cup \cdots r_n = R$, we must detect a color for each. To do this, we examine each coordinate pair p_i in a region r_j. We check the RGB value (r, g, b) of p_i in the input image that the user has scribbled on in color. If $r \neq g \neq b$, the pixel is not grayscale, and is an inputted scribble color. We now map r_i to this (r, g, b) color c and store the pairing.

3.3. Colorization

At this stage, we have n sets of pixels $r_1 \cup r_2 \cup \cdots r_n = R$, each one mapping to a color $c_1, c_2, ..c_n$. With these two inputs along with the original grayscale image input G, we can output a colorized image X. We will proceed with colorizing X one region at a time. For a region r_i, we obtain it's color c_i from our color mapping. Given the color c_i, we generate a monochromatic color scheme of user-defined length l $s_1, s_2, ..s_l \in S$ where s_i is an RGB value ranging from black (0,0,0) to white (255,255,255). Intermediate values are scaled colors of c_i, in other words the same hue, but scaled in brightness. S is not linear, but a gaussian distribution centered around the color c_i, with parametric standard deviation σ. We can increase or decrease σ depending on how great we want the image's contrast to be. Now that we have the monochromatic color scheme S, we can iterate over each pixel coordinate $p_j \in r_i$. We get the brightness of the pixel by getting it's grayscale value g. We get a luminance value $l = g/255$ for p_j. Using this ratio, we get the corresponding color value in S, $S[len(S) * l]$. This will give us the color scaled by the brightness at pixel p_j, c_{ij}. We then set the corresponding pixel in X to c_{ij}. After repeating this process for each region r_i, the image will be fully colorized.

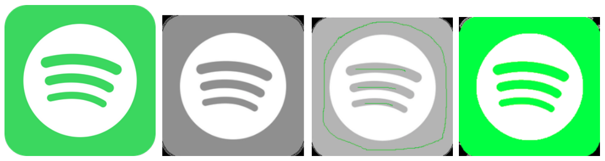
Figure 3. From left to right we have the original spotify logo, its grayscale rendering, user-inputted scribbles, and the model output

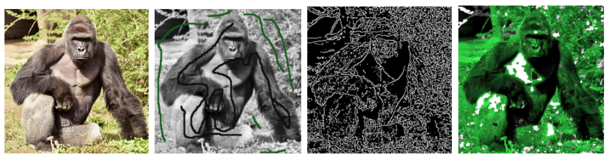
Figure 4. From left to right we have the original image of Harambe the gorilla, user-inputted scribbles, the edge detection output, and the model output. We notice that the edge detection model is missing the larger outline of the gorilla and background, and instead picking out hairs and leaves amongst other minute details. This is the crux of the limitations with my algorithm.

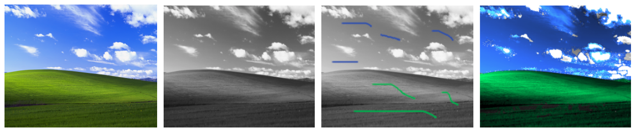
Figure 5. From left to right we have the original Bliss Windows XP image, its grayscale rendering, user-inputted scribbles, and the model output

4. Experimental Results

4.1. Basic Case

In images with no texture and sections separated by clear edges, the algorithm performs extremely well. When we colorize the spotify logo in Figure 3, it works perfectly (the scribbled color choice could be changed to fit better).

4.2. Texture and Noisy Regions

When texture and noise are prevalent in images, meaning there are many edges, Canny's algorithm can detect many small insignificant regions when what we are aiming to identify is larger color regions. While these small insignificant regions may actually be similar in hue in the original input image, the region detection algorithm still perceives many small subsections that propagate issues through the rest of my proposed method. Images with small patterns and many edges such as leaves as in Figure 4 or cities, are difficult for my method to colorize. The method has significantly better success when dealing with images with large obviously contiguous regions as in Figure 5,6 and 8.

4.3. Image Re-Colorization

An advantage of this proposed algorithm is the ability to re-color photos. The typical Convolutional Neural Network approach, or other method that does not rely on user-input

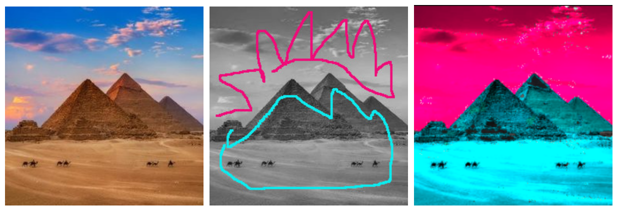
Figure 6. From left to right we have the original picture of the pyramids at Giza, the user-inputted scribbled image, and the re-colorized model output

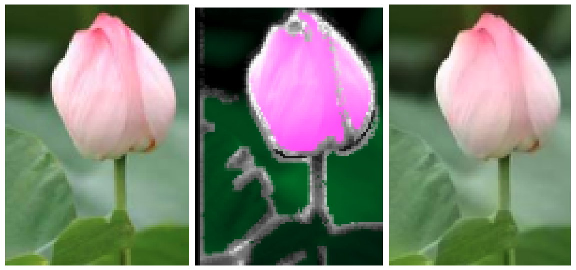
Figure 7. From left to right we have the original picture of a flower, my model's output from a grayscale version of the original, and D. Nie *et al*.'s model output

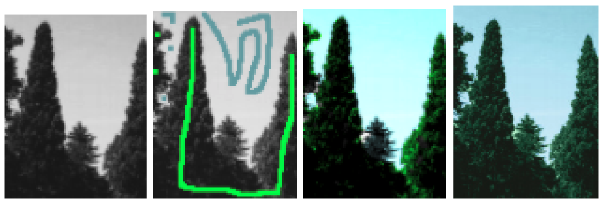
Figure 8. From left to right we have the grayscale image of some trees, the my inputted scribbles, my model's output, and Casaca *et al*.'s model output

lacks the ability to give images unnatural or unordinary colorings as they are trained on datasets of real-life images. As an example, Figure 6 displays an artistic re-colorization of the pyramids at Giza.

4.4. Comparison To Prior Works

Overall, my method has varied success when compared to prior works and is not as robust as other techniques. Observe the comparison in Figure 7 between my method and the method from Dongdong Nie *et al*. Aside from the different choice in coloring scheme, we notice blemishes and rough patches in my output that should be refined. My algorithm performs better in the comparison with Casaca *et al*. As previously stated, my algorithm has success when there are large clear patterns in the image, here being the trees and the sky.

5. Conclusion

In this paper, I present a new algorithm for colorizing grayscale images. The algorithm relies on user-input to choose colors for each sub-region of an image. Canny's edge detection algorithm is instrumental in finding the sub-regions to be colored in. Overall, the algorithm has varied success. It is not robust, as images with many fine edges are not handled properly. This issue could potentially be resolved by revising the region detection process to detect higher-level features, and perhaps using an approach different from Canny's edge detection algorithm altogether. Images that are handled properly by the region detection process have considerable success in re-creating color images from grayscale, and allow the user to artistically re-color, a feature not available to most implementations that lack user-input.

References

[1] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[2] S. X. Dongdong Nie, Lizhuang Ma and X. Xiao. Grey-scale image colorization by local correlation based optimization algorithm. *Lecture Notes in Computer Science*, 3736:13–23, 2005.

[3] M. C. Wallace Casaca and L. G. Nonato. Interactive image colorization using laplacian coordinates. *Lecture Notes In Computer Science*, 9257:675–686, 2015.

[4] D. X. Yili Zhao and Y. Zhang. Image colorization using convolutional neural network. *Communications in Computer and Information Science*, 634:238–244, 2016.

[5] J. Yumibe. Moving color: Early film, mass culture, modernism. *Rutgers University Press*, pages 71–74, 2012.