Colorizing black and white images

Yehya Albakri and Kelly Yen

I. Summary

Much of our historical data (namely black and white images, films, and art) fail to fully capture the importance and authenticity of historical moments. To address this issue, some artists spend countless hours individually colorizing these images, but this isn't a very sustainable method to bring color to all of the black and white historical data. Instead, we can create an algorithm that uses linear regression to associate certain colors in a training set to their corresponding grayscale values. This is a very rudimentary method to applying the concept of computer-colorized images, but it is not difficult to implement supplementary algorithms, such as edge detection and object recognition, to improve the results. The resulting images after applying the algorithm were not great in terms of color quality, but the algorithm did manage to generally apply the original color to the test images. This suggests that in order to attain higher quality colorizations, we would need to extract more information from the black and white image. Given that the current algorithm can only deal with a scale of one color in the image, it has the potential for racial and cultural bias. Depending on the race and cultural background of the people in the training set, the outputted images are severely limited to that range. Besides creating a more robust algorithm, a simple solution to this problem would be making sure that the training images are very similar to the testing set (in terms of culture, race, place, brightness, etc.). Although the current algorithm does work, it's rather limited in application and leaves a lot of opportunity for improvement and iteration.

II. Introduction

When it comes to black and white photographs and films, much of what we perceive lacks the emotion, drama, and humanizing connection that colors are able to evoke in viewers. In the 20th century, many filmmakers recognized these limitations, and spent thousands of dollars and countless hours laboriously coloring vintage films manually. Companies that create photo editing software such as Adobe photoshop also recognized those limitations, and incorporated colorizing capabilities in their products-- a tool that many users employ to colorize old posters and portraits. However the potential applications for this kind of technology goes beyond old movies and your family photo album.

Since the invention of photography centuries ago, we've generated millions of images documenting history's most important people, places, and events. However, it wasn't until 1861, when color photography was invented, that we were able to document those moments with the full color spectrum. As a result, we've viewed much of history through a black and white lens. If we were able to create an algorithm that can colorize historical images efficiently and accurately, without using intensive labor and resources, we would be able to introduce a new perspective of

history-- one that isn't limited by the emotionless and intangible nature of black and white images.

Although there are obvious benefits to being able to colorize black and white images, repainting history doesn't come without social and ethical concerns. The most obvious source of bias that would arise from creating an algorithm to colorize images would be the training data. Not only were cameras invented in Europe, but due to the complex and prevailing effects of colonialism and eurocentric mindsets, most camera owners throughout history were also European. Thus, most historical photographs featured wealthy white people, and creating or finding a training set of historical images that represents all ethnic and social demographics accurately would be incredibly difficult. Of course, we could use modern images in our training set, but doing so would not only undermine the integrity of our algorithm (as it would instill modern cultural and social standards on historic contexts), but also potentially fail to address the original issue as many modern datasets are still racially unbalanced. Additionally, machine learning algorithms inherently perpetuate a norm since they are designed to look for patterns, and use those patterns to make predictions. In other words, using ML and similar technologies to colorize images could lead to the erasure of anomalies in historical images-- a mistake that can have detrimental social and cultural consequences in certain scenarios.

III. Methodology

Our chosen method of colorizing images is with linear regression. Before being able to execute our algorithm, we needed to prepare a suitable training set. Using our class database of face images, we converted each png file into a matrix of pixel values, and compressed each image into 25 x 25 pixel grids. We stored the colored pixel values in matrix `M`, and the grayscale pixel values in matrix `G`. Next, we extracted each RGB layer of matrix `M` into `M_red`, `M_green`, and `M_blue`, respectively. Using the reshape function in Matlab, we turned each matrix of pixel values into vectors in all four matrices, and then transposed each matrix so that every row corresponded with an image rather than a pixel value. We now have four matrices (`G`, `M_red`, `M_green`, `M_blue`), all with dimensions 723 (the total number of images) x 625. At this point, we selected the first 300 rows of each matrix to act as our training dataset (`G_train`, `red_train`, `green_train`, and `blue_train`), and the next 100 rows of each matrix to act as our testing dataset (`G_test`, `red_test`, `green_test`, and `blue_test`).

Before moving on, we first need to address the issue of overfitting. Overfitting means that our final weights depend too closely on the training data, so that the weights aren't suitable for any test data that don't share a lot of similarity with the training data. In order to reduce overfitting, we employed ridge regression, which would essentially force the weight for a particular pixel to stay near 0 in the absence of sufficient data indicating otherwise. In terms of

our code, this meant that we appended an identity matrix scaled to 0.01 to the end of our `G_train` as shown here:

```
Gray_train_modified = [G_train; 0.01*eye(625)]
```

In order to keep every matrix dimensions homogenous so as to avoid matrix multiplication errors in the future, we also modified the three other matrices of our training set as shown here:

```
  Red_train_modified = [red_train; zeros(625, 625)];
Green_train_modified = [green_train; zeros(625, 625)];
 Blue_train_modified = [blue_train; zeros(625, 625)];
```

Now that both our training and test data are ready to be used, we move on to create our set of weights using linear regression. The system of linear equations we start with is as shown here:

```
Gray_train_modified * weightx = x_train_modified
```

Where x stands for red, green, or blue. In order to solve for `weightx`, we multiply the transpose of `Gray_train_modified` to both sides, and isolate `weightx` as shown here:

```
Weightx = (Gray_train_modified' * Gray_train_modified) /
      (Gray_train_modified' * x_trained_modified)
```

Now, we can multiply grayscale test images with our set of weights (weightr, weightg, and weightb), to predict the rgb pixel values of each grayscale pixel. To test that our algorithm works, we recreate the red, green, and blue layers of a grayscale test image with the following code:

```
r_layer = reshape(G_test(1,:) * weightr, 25, 25);
g_layer = reshape(G_test(1,:) * weightg, 25, 25);
b_layer = reshape(G_test(1,:) * weightb, 25, 25);
```

The colorized test image can be displayed with the following code:

```
rgbImage2 = cat(3, r_layer, g_layer, b_layer);
            imagesc(rgbImage2/255)
```

Figure 1. Using linear regression to colorize black and white images

IV. Detailed description:

In order to determine the accuracy of our results, we found the difference between the predicted rgb pixel values of each test image and the actual rgb pixel values, and then averaged this difference across all 100 test images. The code for this part of the algorithm is shown here:

```
sum = 0;
for i= 1:101
     red = mean(abs((G_test(i,:) * weightr) - red_test(i,:)));
     Green = mean(abs((G_test(i,:) * weightg) -
     green_test(i,:)));
     blue= mean(abs((G_test(i,:) * weightb) - blue_test(i,:)));

     average = (red + blue + green) / 3;
     sum = sum + average;
end
accuracy = sum/100;
```

Accuracy was determined by the average number of pixel values that the observed and predicted pixel values differed. The accuracy of our program with 300 training images, 100 test images, and 25x25 pixel sizes was 8.811. The difference between the original and colorized images can be seen here:

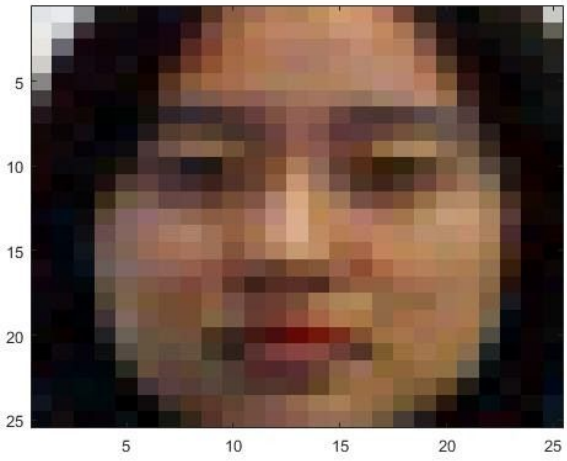

Figure 2. Colorized grayscale image

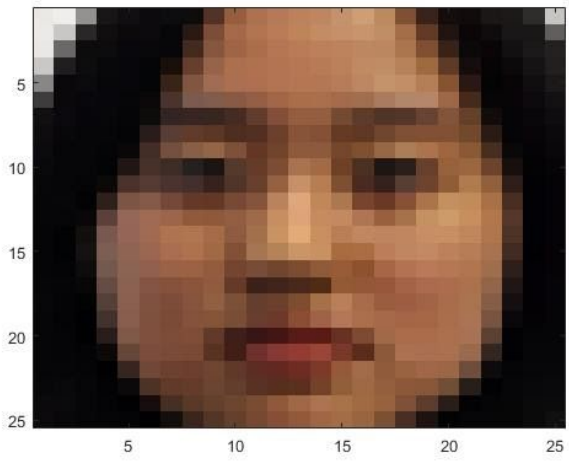

Figure 3. Original test image

Downsides of our algorithm

When experimenting with the number of images to include in the training set, we discovered that despite using rigid regression, overfitting was still an issue with our algorithm, as can be seen in the following images:

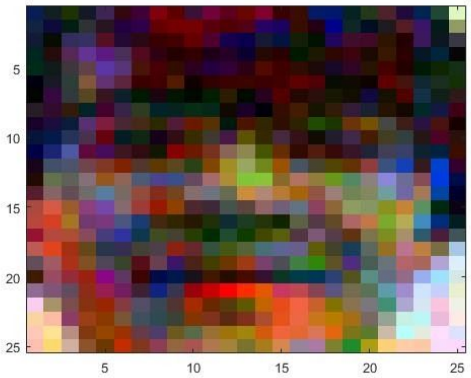

Figure 4. Colorized image with 610 training ‘ images, 25x25 pixels

Figure 5. Original image

Additionally, once we started working with images with higher resolution, the algorithm’s accuracy decreases noticeably (from 8.8 pixels off with 25x25 images, to 12.5 pixels off with 100x100 images). We can see that our algorithm has difficulty smoothing out lines, and determining which contours does and doesn’t exist. This is especially obvious around the eye region, where training images with glasses impacted how test images without glasses were colored.

Figure 6. Colorized image 100x100

Figure 7. Original image 100x100

It appears that the algorithm can deal with multiple colors but under limited conditions. The colors have to be taking up different brightnesses; that way, there wouldn't be much overlap between the colors. An example of this would be an apple sitting on a dark brown table. The brighter colors (like the red from the apple) would be assigned to higher grayscale values whereas the darker colors (the brown from the table and stem of the apple) would be assigned to the lower grayscale values. But, if this separation does not exist, then the colors would be mixed up and it would ruin the final image. An example of this would be if a leaf is introduced to the apple stem. The grayscale values of the leaf would be in the same range as that of the actual apple. There would be two colors trying to assign to the same range of grayscale values and the final image would include both red and green pixels in the leaf and the apple.

There are a few options for addressing the color limitation issue in a future generation of this algorithm. All of the methods we found include color clustering (where colors in a similar vicinity or within the same range are clustered and defined to a certain part of the image). This works in conjunction with either edge detection or object identification. Object identification (by color) would be more accurate, but edge detection would be easier to implement. In both cases, the training set would have to be similar (in colors and content) to the testing set.

Object identification works by identifying all the objects and layers in the image. That way, there would be distinct borders between different colors. It assigns certain colors to objects, then recognizes the objects (by similar shape) in the testing image and applies the corresponding color to each object. It essentially separates the image into multiple smaller images (with a way of identifying them) and applies the same grayscale to color conversion that we used, but individually to the objects in the divided image

Edge detection works in a very similar manner to object identification, but there are a few subtle differences. First, rather than identifying color clusters by shape, it does so by the fact that there is a border between the colors. This is done by a Sobel filter (an edge identifier). Because this method has no way of identifying the differences between clusters, the training set has to be more similar to the testing set than the object identification method. If we were trying to color a

house for example, we would need to have the training set be of similar-looking houses in the same position, taken from the same angle. If grayscale versions of the training set were used as the test images, then the algorithm would deceivingly work better than the object identification method only because we eliminated the algorithm's weakness which is position identification. All the algorithm does is identify clusters with other clusters in the same general position (or at the same mean position).

Finally, to both these methods, we would have to add a smoothing function to make the transition between colors more subtle. A smoothing function would not be needed for our current algorithm because we are using only one color and taking advantage of the natural smoothing in the image between its different shades.

Potential for social harm

Although our current algorithm doesn't have any obvious implications for social and ethical concerns, it could be applied to contexts that do have those implications. Since our algorithm depends solely on the training data to generate colors, it can only colorize images based on what it's seen before. If one were to use a training set with only white subjects, it would have no way of accurately colorzing an image of a non-white individual. Furthermore, skin tones come in a large variety of hues, and the same grayscale pixel value can correlate to dozens of different rgb values. Thus a person who has reddish undertones would appear the same as a person with more green or yellow undertones in black and white images. Using algorithms such as this to colorize images of people has the large potential of homogenizing skin tones, and perpetuating an average. Doing so can perpetuate the issues of current historical studies, where different minority groups are often inaccurately associated as the same despite large cultural disparities. Thus, the ethnic diversity of the training set is vital for an unbiased and accurate algorithm.

Social and cultural norms can also be homogenized by the use of this algorithm. For instance, clothing items and building facades can be easily mis-colored depending on the diversity of colors within the training set. It would be extremely easy for the creator of the training dataset to perpetuate an accepted norm by selecting for images that come from a similar cultural context. Again, the diversity of the training set is vital for an accurate algorithm.

V. Recommendations

To address the technical limitations of our algorithm, we can incorporate object identification and edge detection capabilities into our program. Doing so would allow the algorithm to determine where the contours are for each image, rather than depend entirely on the training images' similarity to the test image. Additionally, introducing a more robust method to address overfitting would allow for an algorithm that can work with a more diverse range of test images. Furthermore, testing this algorithm out on a different set of training images (i.e.

landscapes instead of portraits) could also prove useful in determining which aspects of the algorithm need alterations.

VI. References

Sousa, Austin, et al. “Automatic Colorization of Grayscale Images.” Department of Electrical Engineering, Stanford University, 2013.

Smith, Thomas. “Colorizing Images with a Convolutional Neural Network.” Towards Data Science, Medium, 23 Oct. 2019, towardsdatascience.com/colorizing-images-with-a-convolutional-neural-network-3692d71956e2.

Bugeau, Aurélie, et al. “Variational Exemplar-Based Image Colorization.” HAL Archives-Ouvertes, 2014, doi:hal-00803219.