CS 520 project 4:
Colorization

Group members	
Name	NetId
Mingxiao Lu	ml1329
Chenyang Cang	cc1607
Heng Liu	hl776
Shuya Zhao	sz466

Date: 12/15/2017

General Information

In this assignment, we firstly generate representative colors and patches based on K-means clustering. Then we construct a model based on logistic regression to determine a representative-to- representative map and train model with training data. Finally, we colorize gray image based on our model. To be exact, we determine new data's grey texture and then use our model to predict its corresponding color representative. We will discuss it in the following part.

Codes

There are 4 parts for this program.

K-means clustering (kmeans.py): to generate representative colors and patches. We set the number of K first then generate K clusters of representative colors and K clusters of representative patches separately. Then we find best K for colors and patches separately based on K's ERROR.

Clustering to clustering map (greytocolormap.py): to generate a matching map between each gray shades' clustering representative and its corresponding color clustering representative.

Construct a model to determine a representative-to-representative map (logistic.py): it constructs a model based on logistic regression to map representative gray patches to representative colors. We use input data to train this model and use trained model to colorize new gray image. Input data of new gray patches and get colored image.

Visualize (visualize.py): it contains all functions about changes of data and image. Functions in it can get colored or gray image from data, get data from image and get gray image from colored image.

Task Distributions

Code: Shuya Zhao and Heng Liu completed the code of kmeans.py and GreyToColorMap.py. And Mingxiao Lu and Chenyang Cang completed the code of logistic.py.

Test: Mingxiao Lu and Heng liu tested the properties of k means with different value of k, and error of our model.

Report: Chengyang Cang , Mingxiao Lu and Shuya Zhao completed the questions from 1 to 7. Heng Liu was in charge of the bonus part. He also modified the content of previous questions.

Assigned Questions

1. Taking the shade values to be integers between 0 and 255, how many possible color values are there? How many possible grays? How many possible 3x3 gray pixel patches? Should a 3x3 pixel patch be enough information to guess the center color, or not enough? Under what contexts might it be enough to try to associate every shade of gray with a specific color, and color an image on a pixel by pixel basis?

Each color value has 3 components, so when shade values are integers between 0 and 255, there are 256*256*256=16777216 color values. Each gray has only one component, so there are 256 possible grays. Each 3×3 gray pixel patch has 9 grays, so there are 256*9=2304 possible 3×3 gray pixel patches. A 3×3 gray pixel patch is not enough to guess the center color. A 3×3 pixel patch is a very small area in the picture. It can't be very specific. It's highly possible that there's little different between pixels in the patch, and they are all in the same texture. So 3×3 gray pixel patch provide such little information to guess the center color.

When the picture is small and monochrome, it may be enough to try to associate every shade of gray with a specific color. Classical color to gray conversion formula is Gray(r,g,b)=0.21r+0.72g+0.07b. Then same gray can be converted from different(r,g,b). Actually, the corresponding relationship between gray and the number of color is like normal distribution. So normally it's hard to associate shade of gray, whose values is near the middle of range, with a specific color. But if in a picture, two color values of every pixel are fixed or slightly changed, and the left color changes. Then in this picture, we can absolutely associate every shade of gray with a specific color.

2. Why might it be a good idea to reduce the color-space or the patch-space, for instance by lumping certain shades of green together into one color? What should a good coloring algorithm do if it was fairly confident a pixel should be green or red, but certainly not blue?

By reducing the color-space and the patch-space, capacity of training data can effectively reduce that can improve training efficient without reducing the effect of colorizing. For example, if there's color value is [120, 50, 80] and another color value is [130, 45, 75]. These two color values have slight different, but people can't clearly differentiate them. So it's fair to lump them together. And all values in the range between them can also be lumped together, this will obviously reduce number of possible color values and lower data capacity. Also, if two 3×3 gray pixel patches are slightly different, they maybe belong to same object in the picture and represent similar color in the color image. So it is fair to lump them together and reduce the patch-space.

When it was fairly confident a pixel should be green or red, but certainly not blue. It may remove patch-color pairs whose color is blue in the training data and retrain the model. If we remove patch-color pairs whose color is blue in the training data, the

training model will focus on distinguishing red and green instead of distinguishing red, green and blue. Then this model can distinguish red and green more correctly and maybe can confident this pixel should be red or confident this pixel should be green.

3. **Describe your algorithm, and the design or parameter choices that went into your algorithm. What was your reasoning? What were the difficulties you had to overcome?**

1). Algorithm and parameters:

Our algorithm could be divided into three parts: find representative shades in grey-scale and in color (RGB), build grey-scale-to-color map, and train model on training data.

a) The first part is based on the method of k-means. We chose k1=40 for clustering grey-scale values, and k2=35 for color values. The data structure of a piece of data of grey-scale value is 1 by 9 matrix, and that of a piece of data of color value is 1 by 3 matrix (R, G, B). With fixed value of k, we pick up k data points as initial centers randomly. After assigning centers to each data point, we recalculate the center of each cluster with the formula below. Then we try to associate each data point with the nearest center to it.

$$\underline{c}_i(t+1) = \frac{1}{|C_i(t)|} \sum_{\underline{x} \in C_i(t)} \underline{x}.$$

The process would be repeated until the center of each data point doesn't change anymore. It means that our training on given k is complete and optimal.

b) After we got the representatives of both grey-scale value and color value, we worked on building a certain map between them. For same center point, we can easily find the corresponding data of grey-scale value and color value. However, several different points are clustered to same grey-scale center, but their color values are classified into different color centers. To solve this problem, we mapped the same center of grey-scale value to the center of color value corresponding to most data points with same grey-scale cluster center. Based on the map, we could process our raw data into representatives of different shades.

c) Then we trained our model using logistic regression.

$$f_{\underline{a}}(\underline{x}) = \frac{1}{1 + e^{-(\underline{a}.\underline{x})}}.$$

In the above formula, matrix x in our input data and the value of f(x) would be the predicted value of output. Predicted value would be compared with target output to estimate the error our model. The weight matrix is denoted as *a*.

There is an important part in this step: process data, including extending input space and rescale input and target value. We would modulate the data structure of both input and output data. As for input data, we extended the input space with $\Phi(\mathrm{x})$.We extended (x_1, x_2, x_3, …, x_9) to (x_1, x_2, …, x_9, x_1^2, x_2^2, ..., x_9^2, x_1*x_2, x_1*x_3, …, x_8*x_9), which is quadratic input. After that, we also need to rescale value of input. Since our model is based on logistic regression, this function is valid to data between 0 to 1. Then we divide 255 for value at each dimension. As for output data, we labeled each representative of color value to make our target a single number, changing the size of matrix from 1 by to 1 by 1. Assume we have k classes after labeling each different representatives of color value. We use k to rescale target value. For each target, it is divided by k to make them between 0 to 1. And the weight matrix a used in logistic regression would have the size of 1 by 9 (linear input), or 1 by 54 (quadratic input). If necessary, we transpose input data (grey-scale value).

$$\underline{a}_{k+1} \models \underline{a}_k - \alpha \left[f_{\underline{a}_k}(\underline{x}) - y \right] \underline{x}.$$

The key part in training logistic regression model is updating weight matrix a. We utilized Stochastic Gradient Descent(SGD) to do it. For each loop of updating, we randomly pick up 1/5 of the entire training data, use each data point at each time. The amount of data for SGD guarantee that we could save time for each loop and pick up enough data to update our weight matrix as well.

The condition to decide whether stop the training logistic regression model or not is to see if color difference between predicted classes and target classes in total is under the maximum color difference we could accept. The higher bound for color

difference is 0.02 with the scale from 0 to 1 (6 with the scale from 0 to 255). To make our model more precise, we set the high bound for color difference as 0.01 on average.

d) Besides, for colorization, given a new gray graph, we firstly determine the representative of each 3*3 patch according to model's K-means gray shades' result. To be exact, we eject each 3*3 grey patch into corresponding model's grey clustering centers. And then we use linear regression model with Stochastic gradient descent to predict its color representative. Then, we get color data of image and convert data to an image.

2). Reasoning for parameters:

a) Input size: 1 by 54 matrix. Since logistic regression is a linear model, if we want to obtain the performance achieved by quadratic curve, we have to change our input data manually. The original input is x= [x1, x2, x3, x4, x5, x6, x7, x8, x9], we added x1*x2, x1*x3….and x1*x1, x2*x2… to x, it extended to 1 by 54. Obviously, the result of quadratic model is smoother and more natural. The transition on the edge is quite nicer than that of linear model.

Figure 1 Logistic regression with linear input

Figure 2 Logistic regression with quadratic input

b) K: We tried k in range (10,100) to determine an appropriate value. To make distances between data points and their cluster center as small as possible, we took the value at the elbow of the curve while avoiding too many clusters in the meantime.

From the figures 3 and figure 4, we could see that error reduces as the value of k increases. Both curves are monotonically decreasing, the curves are quite steep and they turn flat after the 'elbow'. For color clustering, the below is at around k=40. And for grey-scale-scale clustering, below is at around k=35.

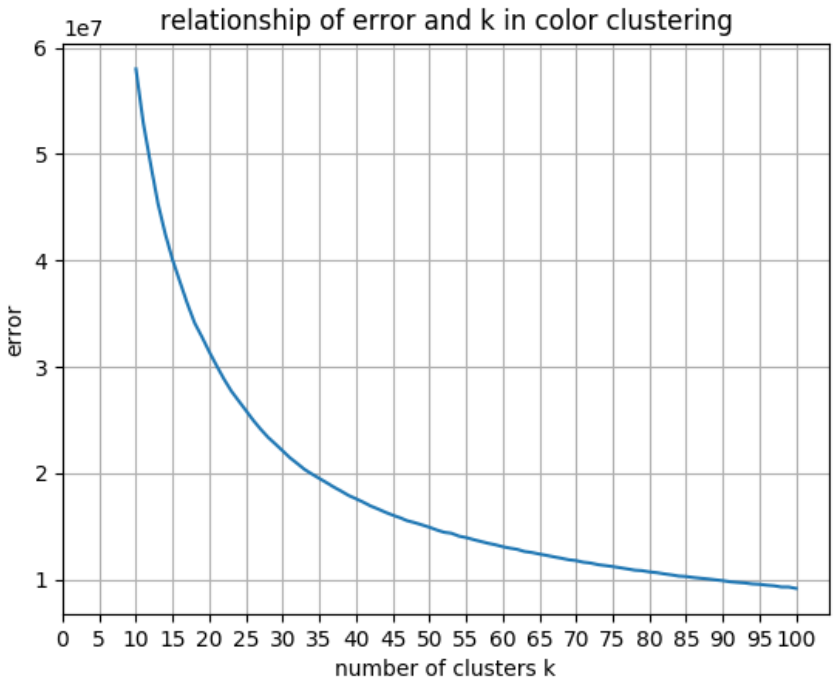

Figure 3 K means on color value

Figure 4 K means on grey-scale value

c) Step size:

$$\underline{a}_{k+1} \models \underline{a}_k - \alpha \left[f_{\underline{a}_k}(\underline{x}) - y \right] \underline{x}.$$

With step size, alpha=0.01, it is quite reasonable to update weight matrix a, since our model decreased the loss quite slow. Step size larger than 0.01 could lead to skip the minimum of the loss because of decreasing too fast.

d) Threshold of color difference: The threshold is set for accepted maximum color difference between predicted color and target color on average. We used the formula below to calculate color difference:

$$\text{dist}(\text{color1}, \text{color2}) = \sqrt{2 * (r1 - r2)\text{^}2 + 4 * (g1 - g2)\text{^}2 + 3 * (b1 - b2)\text{^}2}$$

On average, it requires each pair of prediction and target wouldn't have the gap beyond 2 in each dimension of color value with scale from 0 to scale. That's why we set threshold as 0.01 on average. The three following figures show that with the

difference of 2, human cannot identify the difference visibly.

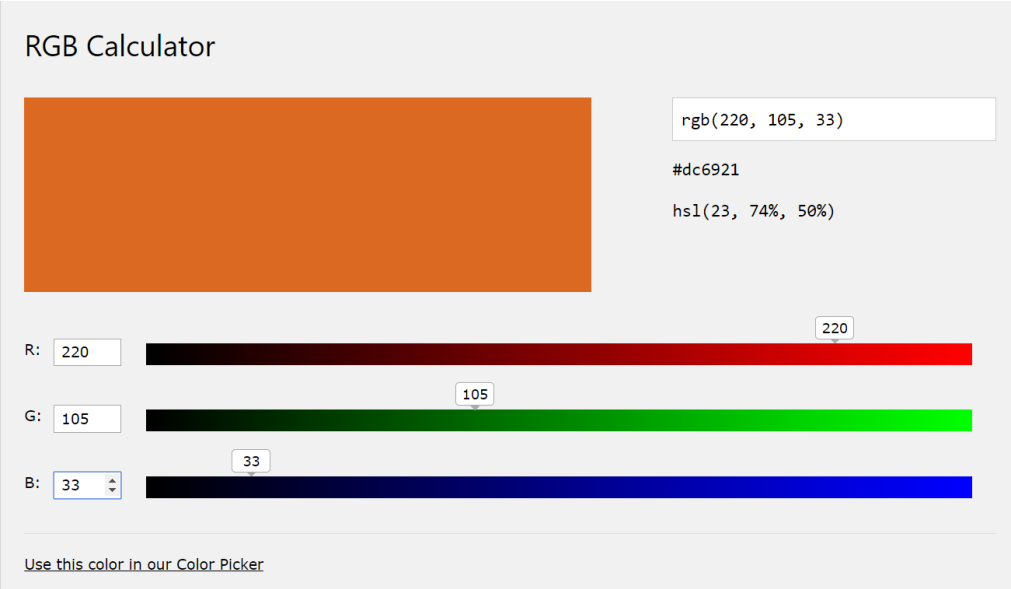

Figure 5 Color with RGB (220,105,33)

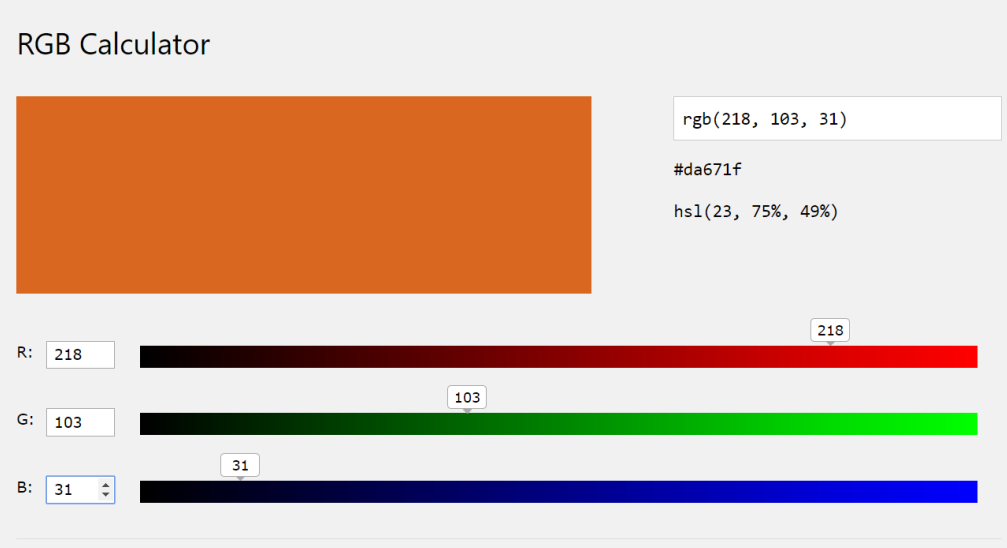

Figure 6 Color with RGB (218,103,31)

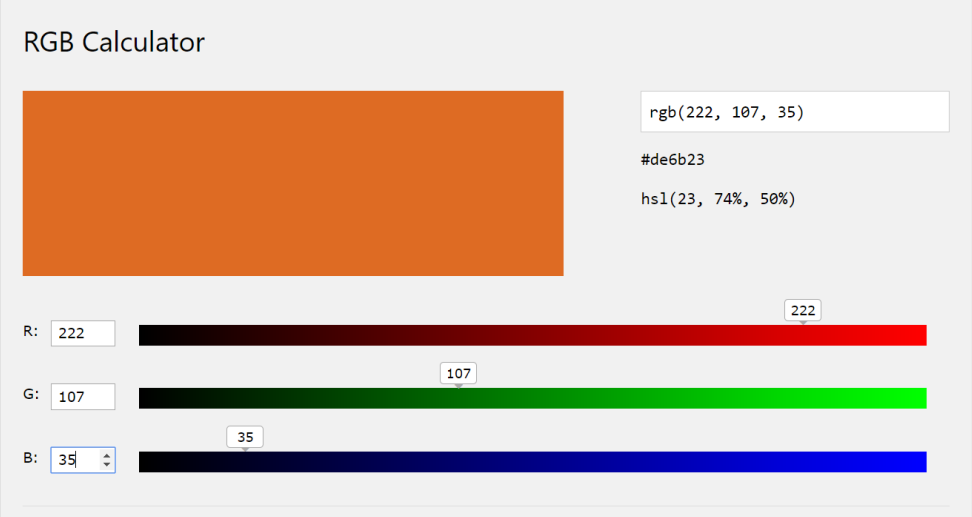

Figure 7 Color with RGB (222,107,35)

3). Difficulties:

Since logistic regression would rescale value between 0 to 1, we had to rescale our grey-scale value and color value by being divided by 255. And multiply 255 with output to restore the true value.

It would be more feasible to train with clustered color value rather than labels as target. Color value would make our model closer to the optic feature, not only numerical requirement. But to fit our method, we had to label each cluster.

4. How can you use the provided data to evaluate how good your coloring algorithm is? What kind of validation can you do? Quantify your algorithm's performance as best you can.

We divide our input data into three parts, 80% as training dataset, 10% for test dataset and 10% for validation dataset. According to the ratio, we randomly select three types of data from original dataset rather than pick them in order. Because random selection enables us to acquire more general information about the graph, and therefore algorithm will learn more features and perform better. For instance, if we just divide data according to the order of input, the last part of input data will not be included in the training data and therefore, our model will not learn any feature from that. So, if we test our model, it will perform poorly.

Based on validation dataset, we quantify our algorithms in three ways.

1) To evaluate the performance of K-means clustering result, we define K's *ERROR* as the Sum of squared distances of samples to their closest cluster center. Then we

compare diverse K's errors and pick a reasonable one.

In figure 8, we can see that around 40, it is the inflection point of this function's second derivative. So, before this point, error decreases rapidly, we cannot choose point from this period. And after this point, error decreases slowly. We cannot choose point from this period either, because if there are too much clusters, it will cause overfitting. Therefore, we choose k for 40.

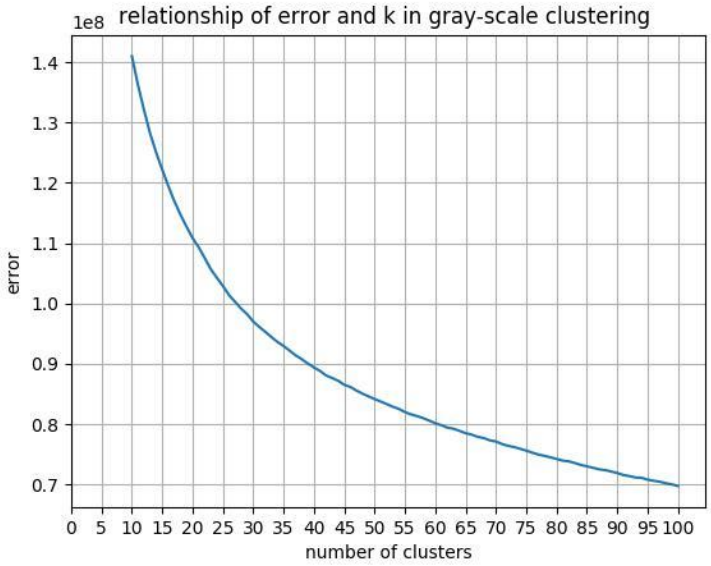

Figure8 Relations of error and k in grey-scale clustering

In Figure 9, for the similar reason like clusters in grey context, we choose color clustering k as 35.

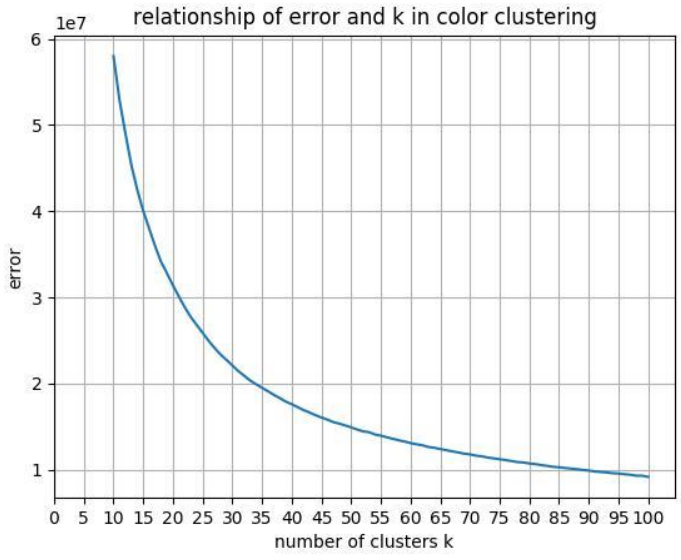

Figure9 Relations of error and k in color clustering

2) To evaluate the performance of our final algorithm, we use a mathematical approach.

We define score as the *MEAN ACCURACY* on the given test data and labels. Since we cluster input grey data and color data in the first step. Test data is the representative of grey data and labels are the representatives of color data. When applied each test data into our model, if its color label correctly corresponds to the original dataset label, we set this test data's accuracy to 1, otherwise, we set it to 0. Then we calculate the *MEAN ACCURACY.* We will compare some *MEAN ACCURACY* to show the improvement of our model.

a) Input data with just 9*1 grey matrix, *MEAN ACCURACY* = 0.55

b) Input data with 54*1 grey matrix, *MEAN ACCURACY* = 0.74. We use 54*1 grey matrix by expanding 9 vectors into 54 vectors. 9 original one, 9 X_i^2, and C_9^2 X_i*X_j. This expands input data feature and therefore, our model learns more feature and performs better.

3) Consider about human perception about differ colors, we use *COLOR DISTANCE* to evaluate our algorithm's performance.

$$\text{dist}(color_1, color_2) = \sqrt{2(r_1 - r_2)^2 + 4(g_1 - g_2)^2 + 3(b_1 - b_2)^2}$$

For each r, g, b, it ranges from (0,255). We calculate the sum *COLOR DISTANCE* for each model estimate color and actual color and then get average of it.

For input size with 9*1 matrix, *COLOR DISTANCE* = 11.7

For input size with 54*1 matrix, *COLOR DISTANCE* = 4.5

4) Consider about human's general conception of a whole graph, we also evaluate our algorithm by recover a new grey graph and compare them.

For input size with 9*1 matrix, we got Figure 10.

Figure 10 Recover graph with input size 9*1

For input size with 54*1 matrix, we got Figure 11.

Figure 11 Recover graph with input size 54*1

5. Once you have trained your algorithm on input.csv and color.csv, use it to provide colors for each of the example patch data in data.csv. Submit the computed colors in a csv file, each row corresponding to the color of the corresponding row in data.csv.

How to use our trained model to output colors, and then to render it? We divide the process into three parts. As seen below.

First part is to pre-process data. During this process, we change the 1*9 input format into 1*36 to fit out model. The way to convert is enumerate every possible pairs in 1*9 vector and do multiplication. Then we can get 36(C_9^2) values.

Second part is use model trained to output center colors.

Third part is to use color obtained to output a png format picture.

Figure 12 Processes of using our algorithm

The final color.csv is in the compressed files.

6. Where does your algorithm perform well, where does it perform poorly? How could you (with greater time and computational resources) improve its performance? What parameters or design decisions could you tweak? Would more data on more diverse images help or hinder your algorithm?

Figure 13 Outputting graph

1) Perform well: I think the part of color choice is good. After color clustering, the clusters of color are selected uniformly to balance different shades of color in original image. Blue shade dominates in original image, so a poor could transfer all grey-scale-scale values to different shades of blue rather than other colors, like yellow. Our model avoids this mistake, it shows as much shades of color as possible to make result closer to original color image.

2) Perform poorly: Because of clustering, some shades were discarded. The missing of subtle transition of color lead to transparent edge between neighbor areas. Increases the number of cluster could improve the result to some extent.

3) Tweak design: the parameters, k_1 and k_2, could be adjusted to make our k-means model more reasonable. Now we have k_1=40 and k_2=35, if we double the value of k, I think the problem of edges would be resolved.

Also, we could change the type of our training model. SVM and neural network both work well on this problem. Linear input on SVM could obtain a result as well as the result of quadratic input on Logistic regression.

Figure 14 Outputting graph of SVM

4) More data: On the one hand, more data on other images which have same color distribution as our target image would help us train the model by promoting precision, since these train data has strong similarity to target. On the other hand, if these data are collected from diverse images, which means they could have totally different distribution of colors, training data would confuse our model so that model perform poorly on target image.

7. Do you think this is a reasonable approach to the problem? What would you change?

1) Reasonable Approach:

Firstly, grey input size is a 9*1 matrix, which is vector space and we need to train a model to classify them into corresponding color output. Logistic Regression can deal with classification problems. Besides, since Logistic Regression introduces sigmoid function into Loss function and optimize it, LR can deal with some nonlinear features. And Stochastic Gradient Descent enables optimize process to acquire an optimal answer with reasonable time.

Secondly, we use *COLOR DISTANCE* to determine whether to end training model or not. If COLOR DISTANCE is less than a threshold, we end up the training and save the model. *COLOR DISTANCE* considers people's general conception among colors. Therefore, it is a reasonable ending condition for model training.

Thirdly, we decide to use 54*1 input grey matrix rather than 9*1 matrix. This approach expands the input data size and therefore, our model can learn more features and perform better.

2) Change:

Since Kernel Function can map nonlinear data into a higher dimensional space so that it is linear separable. Therefore, if we introduce it to our optimization function, our model may deal better with nonlinear data.

For Linear Regression model, we use COLOR DISTANCE as our stop condition. Maybe we can use Loss Function to determine when to stop. Or according to formula 1 below, we can set $a_{k+1} - a_k$ to a threshold to determine whether this function is converged or not. And if it reaches the threshold, we stop train our model.

$$a_{k+1} = a_k - \alpha[f_{a_k}(x) - y]x$$

For parameter, we will try to change learning rate α to see whether our model can perform better or not.

8. What would happen if you tried to color a grayscale image of a tiger, after training your algorithm on pictures of palm trees?

As we discussed in the question 1, the corresponding relationship between gray and the number of color is like normal distribution, one gray can be convert from different$(\mathrm{r}, \mathrm{g}, \mathrm{b})$. After training algorithm on pictures of palm trees, it makes model more likely to convert gray to colors of palm trees, like brown and green. Then if we use this model to color a grayscale of a tiger, it's more likely to covert gray to colors of palm trees instead of colors of tigers. It will get an image of a tiger with weird colors.

Bonus

Write your program to take in an image, convert it to grayscale, train your algorithm, and then use the algorithm to color example grayscale images. How does your algorithm do, visually?

For the color image, we first choose an image that's different from palm image, seen as below.

Training Data 1

Figure 15 Colored picture of tiger

As we can see, the main color is tiger picture is much different from that in palm picture. So we can predict the effect will not be satisfiable. Then we convert tiger picture to grayscale tiger picture using the formula in assignment 4 specification.

Figure 16 Grayscale tiger picture

So with two pictures above, we have data to train.

For the training part, we choose the model that trains palm trees. And we choose

the same k_1 and k_2 to cluster, and all other parameters are the same. Eventually, input a grayscale palm picture, we can get the color palm picture.

Figure 17 Colored palm picture

Analysis: As we can see, the color in above picture is more similar to color tiger picture than original palm picture. That's because we train our model to better restore tiger-like picture instead of palm-like picture. So the effect is not good.

Training Data 2

Figure 18 Colored palm-like picture

The second training picture is a palm-like picture and the main color of it is almost the same as original palm picture. Below is its grayscale picture.

Figure 19 Grayscale palm-like picture

Using the pictures above and train our model using the same k and other parameters. Below is the effect picture.

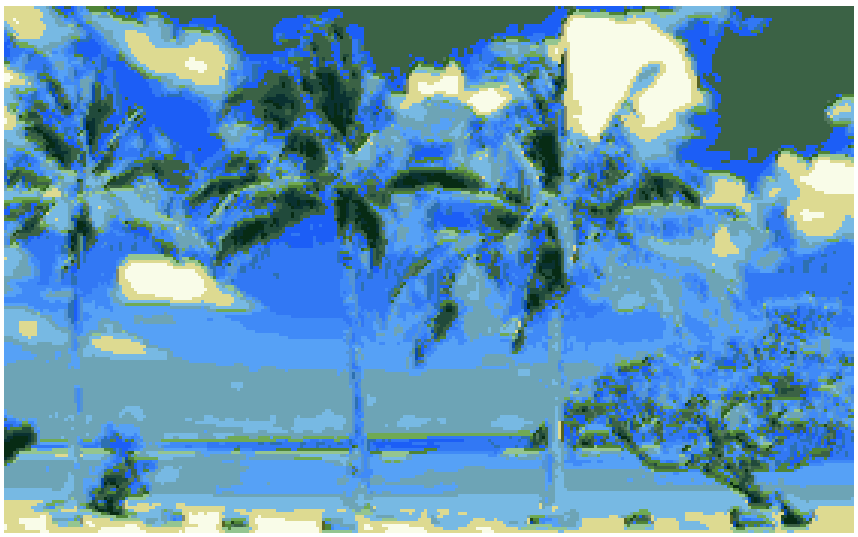

Figure 20 Colored palm picture

Analysis: Well, this time we can see more original colors in above picture. However, we cannot see the separating colors clearly. That's because the structure of new data and original data is not that similar.