CS 520: Assignment 4 - Colorization

Karan Ahuja, Nitin Mali

December 14, 2017

Contributions

- KA - Karan Ahuja
- NM - Nitin Mali

Program design and implementation:
Kmeans Algorithm implementation- KA, NM
Linear Regression Implementation– KA
Analyzing Results - KA, NM
Bonus - KA, NM

Write-up:
Latex formatting - KA, NM
Part 1 Questions:
1 - KA
2 - KA
3 - KA
4 - KA
5 - NM, KA
6- KA
7- KA
8 - NM, KA

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, or color and image on a pixel by pixel basis**
   Considering shade value of integers to be in between 0 and 255, there are 256 unique values for each color. So for Red, Green and Blue there will be a total of 256^3 i.e 16,777,216 possible color values.. There are 256 possible gray values and 256^9 i.e $4.72237 * 10^{21}$ possible 3x3 gray pixel patches.

   Based on working with images of a beach and a tiger for us it is reasonable to conclude that 3x3 pixel is not enough information to guess the center color. A 3x3 pixel patch contains very little information regarding the texture of the color and thus it might not be sufficent to associate it with it's center color. If we are trying to color images with plain colors and not the images which occur naturally trying to color an image on a pixel by pixel basis might suffice. For example if we are trying to color gray scale images of flags of countries like Romania or Ireland we are more concerened with the color and not the entire texture. In such cases coloring pixel by pixel might certainly work out well.

2. **Why might it be a good idea to reduce the color-space or the patch-space, for instance by lumping shades together of green together into color? What should a good algorithm do if it was fairly confident a pixel should be green or red, but certainly not blue?**
   There are several reasons for reducing the color space (Clustering) to be a good idea:

   - As we see that there are 256^3 possible values for Red, Green and Blue, not all these values will be occuring in the natural images or images of interest. Lumping certain shades of green or red reduces a lot of variance in the original data and we end up having what actually matters in the original data.
   - Clustering also helps to filter noise from the data. Certain outliers in the original data would be lumped in under a certain cluster which can help in better classification/prediction.

- Data Balancing: There might be a lot of cases where the data might not be equally balanced. With respect to this assignment suppose we had an image which contained 70% of red of 30% of green, it is more likely that a machine learning algortihm will be prune to predict red most of the time. If we cluster this data (2 clusters, suppose), assign respective centers of the clusters to the data points and use equal amount of samples for both the clusters, the results can significantly improve.

A good algorithm should find out the right combination of green and red, simply averaging won't work. Based on the weights the algortihm as learned and the corresponding input it should try to map it to nearest possile shade.

3. **Decribe your algorithm, and the design or the parameter choices that went into your algorithm. What was your reasoning? What were the diificulties you had to overcome?**
We implemented two algoithms: kmeans algorithm to cluster the data and linear regression for prediction. For the bonus part we modified linear regression to polynomial regression of power 3. **Design/ Parameter Choices**

   - **kmeans algorithm:** Previously we were trying to cluster both the color data and the gray scale data, but after many tests with linear regression we just clustered the color data and mapped normalized grayscale data to cluster center values using linear regression. To cluster the color data we defined the distance as:

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

     To dtermine the optimum number of clusters we clustered with values ranging from 2 to 50 and calcluated how the over all clustering loss perfroms. Loss was nothing but the sum of the mean distance between every data point and its corresponding cluster center to which it belongs. Correctly as the number of clusters increased the overall loss decreased. Using elbow method we found out the optimum number of clusters to be between 12-15.

     To cluster the gray scale data we used the similar distance formula above, but we gave more weight to the center pixel i.e the elements belonging to coloumn 5 of the input data. However as the results did not turn out well, we did not cluster the grayscale data

     For the kmeans algorithm we concluded upon convergence when the data points did not move after recentering which resulted in centers being the same.

   - **Linear Regression:** Based on the problem it was evident that a linear algorithm would just average out the data ang give bad results in terms of actual colors. Hence we introduced non linearity in the data set by using kmeans clustering and explored linear regression to see how it perfroms. We defined the our hypothesis function as:

$$f(x) = \sum_{j=1}^{m} \theta_j x_j$$

     where θ contains the parameters of the model .and m is the number of data points. We defined our loss function as:

$$\text{loss(f(x), y)} = ((f(x) - y)^2)/\text{m}$$

     To compute the parameters of the model we used batch gradient descent instead of stochastic gradient descent as the number of data points were not that huge. We varied the learning from 0.0001 to 0.01 and plotted loss versus number of iterations for gradient descent for each learning rate. We selected that particular learning rate where loss decreased with increasing number of iterations. To sum up we got comparative best results when input was the normalized gray patch, a linear regression model and clustered color centers as output. To perform normalization we directly divided each data point by 256. Mean normalization also could have worked here but we just divided the eah data point in the data set by 256. Our output contained an array of m rows and 3 coloums, so for each coloumn we used linear regression sepearately to minimize the loss and then combined the output.

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 as you can**
We divided the data into training, test and validation. We experimented with various hyperparamters in our case such as number of clusters, type of input (clustered, normalized, one hot encoded), type of output (cluster centers, one hot encoded), learning rate and worked on training and test data. Based on the results we quantified results on the validation data set. Although the data provided is less, it can be still reasonably used to evaluated how our clustering algorithm is.

   - Based on results for the validation data set we get a mean square error of **900** for red color, **147** for green color and **1250** for blue color.

- After the model predicted the colors for the validation data set, we matched the predictions to the nearest cluster indices and calculated the misclassification rate of model. Our model could classify the clusters correctly with an accuracy of **48**% . The number of clusters in our model was 15.
- Using inbuilt functions we calculated the Peak Signal to Noise Ratio on the validation data set which was **4.533**
- On an average our model was off by **25 pixel values** for red, green and blue color. This was caluclated from the plots of loss function versus number of iterations plot.

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**
   The **data.csv** file is submitted along with the other files.

6. **Where does our algorithm perform well, where does it perform poorly? How could you (with greater time and resources) improve its performance? What parameters or design decisions could you tweak? Would more data on more diverse images help or hinder your algorithm?**
   Our algorithm is able to identify clusters accurately and can accurately recover entire image based on the clusters. Also our algorithm is able to colorize a simple image with red and blue color. But when we train it on a image of tiger and test it on another image of tiger, it performs poorly. Although the data is clustered and there is some non linearity induced, the reconsrtucted image based on the response generally seems to have an averaging process. The green shades seem to be distinct in the reconstruced image, but all other colors (orange, white and black) seem to have a single tone. To conclude we can say that our entire model still has some averaging process which increases the mean square error as well. With greater time and resources we would not only like to improve our model, but explore other possiilities as well. To improve our model we would be keen in using polynomial regression rather than just linear regression. For polynomial regression we would tweak our hypothesis which can be something the form of

$$f(x) = \sum_{j=1}^{m} \theta_j x_j + \sum_{k=1}^{m} \theta_k x_k^2$$

   We can increase the order as per the need, and insted of gradient descent we can use stochastic gradient descent if the number of data points are huge. We can still stick to squared error loss and optimize the results. To assess overfitting we can directly work on several test images, drastic change in predicted colors (for example expected color is red but predicted is blue) wo clearly indicate overfitting. More data on more diverse images will definitely help the algorithm as the model can learn all the possible values and amappings. Apart from polynomial regression we would be keen on observing how k nearest neighbours perform. We can find the nearest gray patch value to the current patch and then map it to the respective color. The design parameters we can tweak here is the number of neighbors to look at as well as specifying the distance metric between two patch values. As we mentioned before center pixel in the patch can be assigned more weight with respect to the neighboring pixels.

7. **Do you think this is a reasonable approach to the problem? What would you change?**
   Yes, the current algorithm with the optimizations mentioned in question 6 with in depth analysis could be a decent approach to the problem. We would change the gray patch data. Instead of a 3x3 patch a 5x5 patch or even 7x7 could help capture a lot of texture for the color image and solve the problem in a more better way.

8. **What would happen if you tried to color a grayscale image of a tiger, after training your algorithm on pictures of palm tress?**
   The image of palm trees is certainly likely to contain more green color than any other color. When we train our algorithm on this it would be more biased towards predicting green color (or specifically a shade of green) most of the time. So when we try to color the image of a tiger the colrized image will have a tone of green color which might make the orange and white color patches appear green in color.

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

Figure 1: gray scale image for training

Figure 2: corresponding color image for training

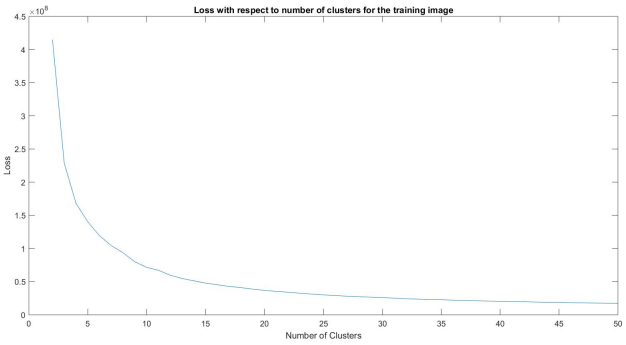

Figure 3: Loss versus number of clusters plot for color data

Figure 4: recovering the image based on cluster centers, (15 Clusters)

Figure 5: Similar gray scale image to test the algorith

Figure 6: Colored image by our algorithm

Figure 7: Actual Color Image (Ground Truth)