Colorization

CS 440

Cesar Herrera, Jae Weon Kim

Spring 2020

List of Figures

The Problem

Consider the problem of converting a picture to black and white.

Figure 1: Training Data - A color image and its corresponding grayscale image

Typically, a color image is represented by a matrix of 3-component vectors, where Image[x][y] = (r,g,b) indicates that the pixel at position (x,y) has color (r,g,b) where r represents the level of red, g of green, and b blue respectively, as values between 0 and 255. A classical color to gray conversion formula is given by

$$Gray(r,g,b) = 0.21r + 0.72g + 0.07b,$$

where the resulting value Gray(r,g,b) is between 0 and 255, representing the corresponding shade of gray (from totally black to completely white.)

Note that converting from color to grayscale is (with come exceptions) *losing* information. For most shades of gray, there will be many (r,g,b) values that correspong to that same shade.

However, by training a model on similar images, we can make contextually-informed guesses at what the shades of grey ought to correspond to. In an extreme case, if a program recognized a black and white image as containing a tiger (and had experience with the coloring of tigers), that would give a lot of information about how to color it realistically.

Figure 2: Trained on the Color/Grayscale image in Fig.1, recovers some green of the trees, and distinguishing blues between sea and sky. But there are definitely some obvious mistakes as well.

For the purpose of this assignment, you are to take a single color image (of reasonable size and interest). By converting this image to black and white, you have useful data capturing the correspondence between color images and black and white images. We will use the left half of

the image as training data, and the right half of the image as testing data. You will implement the basic model described below to try to re-color the right half of the black and white image based on the color/grayscale correspondence of the left half and as usual, try to do something better.

The Basic Coloring Agent

Consider the following basic strategy for coloring an image: while a single gray value does not have enough information to reconstruct the original cluster, considering the surrounding gray pixels might. So given a 3x3 patch of grayscale pixels, we might try to use these 9 values to reconstruct the original (r,g,b) value of the middle pixel. We simplify this further in the following way:

- Instead of considering the full range of possible colors, run k-means clustering on the colors present in your training data to determine the best 5 representative colors. We will color the test data in terms of these 5 colors.
- Re-color the right half of the image by replacing each pixel's true color with the representative color it was clustered with.
- For every 3x3 grayscale pixel patch in the test data (right half of the image), find the six most similar 3x3 grayscale pixel patches in the training data (left half of the image).
- For each of the selected patches in the training data, take the representative color of the middle pixel.
- If there is a majority representative color, take that representative color to be the color of the middle pixel in the test data patch.
- If there is no majority representative color or there is a tie, break ties based on the selected training data patch that is most similar to the test data patch.
- In this way, select a color for the middle pixel of each 3x3 grayscale patch in the test data, and in doing so generate a coloring of the right half of the image.
- The final output should be the original image, the left half done in terms of most similar representative colors to the original image colors, and the right half done in representative colors selected by the above process.

How good is the final result? How could you measure the quality of the final result? Is it numerically satisfying, if not visually?

Bonus: Instead of doing a 5-nearest neighbor based on color selection, what is the best number of representative colors to pick for your data? How did you arrive at that number? Justify yourself and be thorough.

The Improved Agent

In the usual way, we want to build an improved agent that beats the basic agent outlined previously. You have a lot of freedom in how to construct your approach, but follow these guidelines:

- Use the left half of the image as training data (color/grayscale correspondence) and the right half of the image as testing data (having converted it to grayscale).
- The final output should be the original image, with the right half colored according to your model.
- *The use of pre-built ML libraries or objects, or automatic trainers, is strictly forbidden. You can use TensorFlow for instance as an environment to build your model in, as long as you do not make use of layer / model objects or automatic differentiation /training*

Your final writeup should include the following:

- A specification of your solution, including describing your input space, output space, model space, error or loss function, and learning algorithm. (*See **P1** in **Write Up and Analysis Below***)
- How did you choose the parameters (structure, weights, any decisions that needed to be made) for your model? (*See **P2** in **Write Up and Analysis Below***)
- Any pre-processing of the input data or output data that you used. (*See **P3** in **Write Up and Analysis Below***)
- How did you handle training your model? How did you avoid overfitting? (*See **P4** in **Write Up and Analysis Below***)
- An evaluation of the quality of your model compared to the basic agent. How can you quantify and qualify the differences between their performance? How can you make sure that the comparison is 'fair'? (*See **P5** in **Write Up and Analysis Below***)
- How might you improve your model with sufficient time, energy, and resources? (*See **P6** in **Write Up and Analysis Below***)

As usual, be thorough, clear, and precise, with the idea that the grader should be able to understand your process from your writeup and data.

Some Possible Ideas

- The basic agent described previously executed k-NN classification, using the pre-clustered colors. You could also think of this as a regression problem, trying to predict the red/green/blue values of the middle pixel.
- Note that there are things you know in advance about the red/green/blue values, for instance, they are limited to values between 0 and 255. How does this inform your model?
- How could soft classification like logistic regression be used?
- Neural networks are always an option, but how should you choose the architecture?
- The k-NN classification of the basic agent essentially reduces the output space dramatically - how could the input space be similarly reduced?

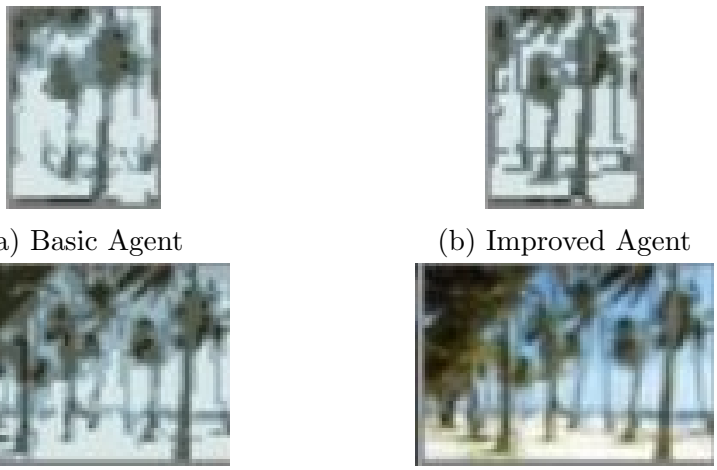
Write Up and Analysis

(a) (b)

(c)

Figure 3: Images Used

Results 1



(a) Basic Agent (b) Improved Agent

(c) Clustered (d) Original Image

Figure 4: Scaling Images To Different Dimensions

Results 2

(a) Basic Agent

(b) Improved Agent

(c) Clustered

(d) Original Image

Figure 5: Scaling Images To Different Dimensions 2

Results 3

(a) Basic Agent

(b) Improved Agent

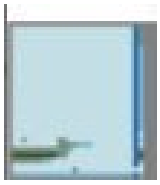
(c) Clustered

(d) Original Image

Figure 6: Scaling Images To Different Dimensions 3

Results 4

(a) Basic Agent



(b) Improved Agent

(c) Clustered

(d) Original Image

Figure 7: Scaling Images To Different Dimensions 4

Results 5

(a) Basic Agent

(b) Improved Agent

(c) Clustered

(d) Original Image

Figure 8: Scaling Images To Different Dimensions 5

P1: A specification of your solution, including describing your input space, output space, model space, error or loss function, and learning algorithm.

Basic Agent

- The basic agent model followed the instructions described in the project description above. The input space consisted of all the pixels from the image. The pixels of the left half of the image were used as training data, and those pixels were used as padding for the edge of the image. The output space was filled with the testing pixels from the right half of the image. The pixels are filled with gray colors and are recolored after the comparison process with the five representing colors from k-means clustering. In our case, the value of k was set to 5. When it came to difficulties, one of the hardest parts for structuring the basic agent was the implementation of the k-nearest neighbors algorithm, due to the pixel sizes. Although we are only recoloring the right half of the image, the model was still required to compute all of the pixels to check the similarity with the original image. Additionally, each of the pixels required O(n*n) time complexity which proved quite difficult to find the best environment setting to get the best result possible.
- Our solution to this problem:
  - We padded the image's patch size to half. This way the agent proceeded to handle the edge pixels of the image quicker than the original environment setting. In general, the 'similarity' for the KNN model is computed by sorting the euclidean distance size in ascending order (*in which the smaller size is considered as a higher 'similarity'*). However, to minimize the computational time complexity, we substituted with finding the mean squared error from the patcher of the image.
- To handle the error or loss function, the first obstacle we encountered was figuring out which of the six representative colors would be patched primarily with what standards. So, there are two standards that we decided on.

1. Finding the size most similar $3x3$ grayscale pixel patches in the training data (*left half of the image*). And then taking those six patches and sorting them in ascending order each according to the risk and error values of said patches.
2. Counting the pixels, representing colors in the list and sorting them in descending order.

Then finally taking the first element from each of those two sorted lists and choosing the one that is the most similar to the current testing pixel.

Improved Agent

- The improved agents input space and output space are relatively similar to the basic agent environment settings mentioned above. Given an image, the left half was used for the training that was divided into patch sizes of three.
- K-means clustering is also used here before implementing the neural network model to extract some representing colors, while the agent computes the similarity or minimizing the loss function.
- The improved agent used a 2-layer neural network model. The training data labels are the representative colors that are extracted from the left half of the image. (e.g At this point, the data shape was something similar to [211,20,52]. So we used the *color_to_label* function (***main.py, line# 34***) to execute a one-hot encoding process for improving the predicition accuracy).
- The model parameters are updated by the calculation of the numerical which uses the training data batch and the test data batch. The interlayer activation function is currently set up as the *sigmoid* function (***common/functions.py, line# 12.*** There are also two other functions *relu* and *step*[***common/functions.py, line# 20 & 8 respectively***] that are available to replace the sigmoid function)
- The output layer uses the *softmax*function (***common/functions.py, line# 30.***) to make use of the representing color index classification.

P2: How did you choose the parameters (structure, weights, any decisions that needed to be made) for your model?

Basic Agent

- The hyperparameters (*K, patch_size, patcher, similar_path_num, in* ***main.py, line#'s 49-53***) are following the environment suggested in the project description above.Additionally, *skip_knn* (***main.py, line# 53***), decides whether the basic agent skips the KNN computation or not.
- Among the possible results that you can get by changing hyperparameters, the major difference will occur when you change the K value and *similiar_patch_num*. The reason for this is because the range for recoloring the pixels becomes more flexible and less limited/restricted, when used for comparison between the best siimilar representing colors and similar path numberfrom the KNN computation.

- **INSERT COMPARISON HERE FOR SIMILARITY MEASURE GRAPHS USING DIFFERENT HYPER PARAMS**

Improved Agent

- To get higher accuracy from the testing data, we experimented with different hyperparameter settings. (Which are the *iters_num, batch_size, learning_rate* and *hidden_size* variables in **main.py, line #'s 53-56**).
- *iters_num* affects the result if the size of the pixel is large enough to process the different classification. The main difference will occur if *batch_size* and *learning_rate* change because those values will directly change the weight values and computations for the error/loss model.
- **SHOW COMPARISONS OF RESULTS W/ DIFF HYPERPARAMETERS FOR IMPROVED AGENT**

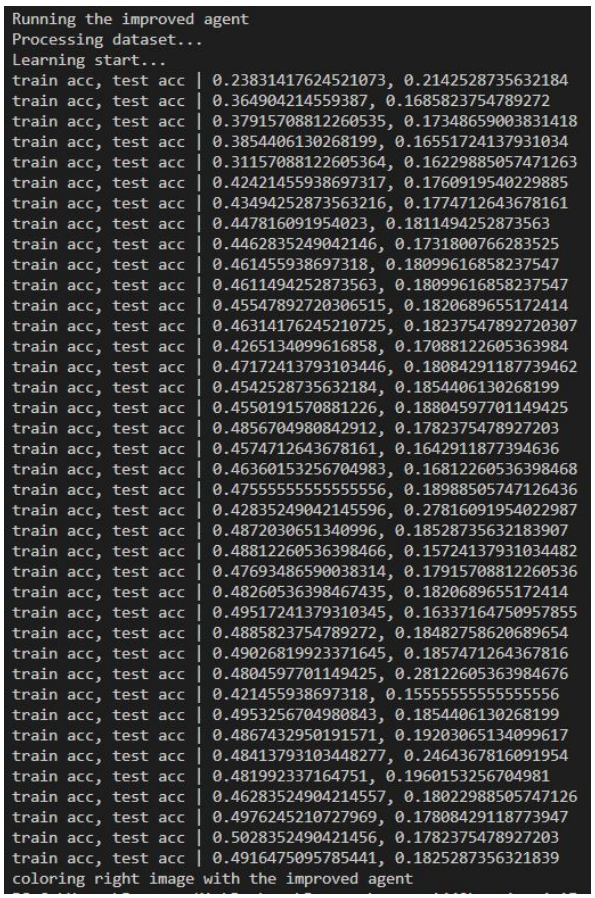
P3: Any pre-processing of the input data or output data that you used.

- There was minor pre-processing of the input data, i.e flattening the image pixels.

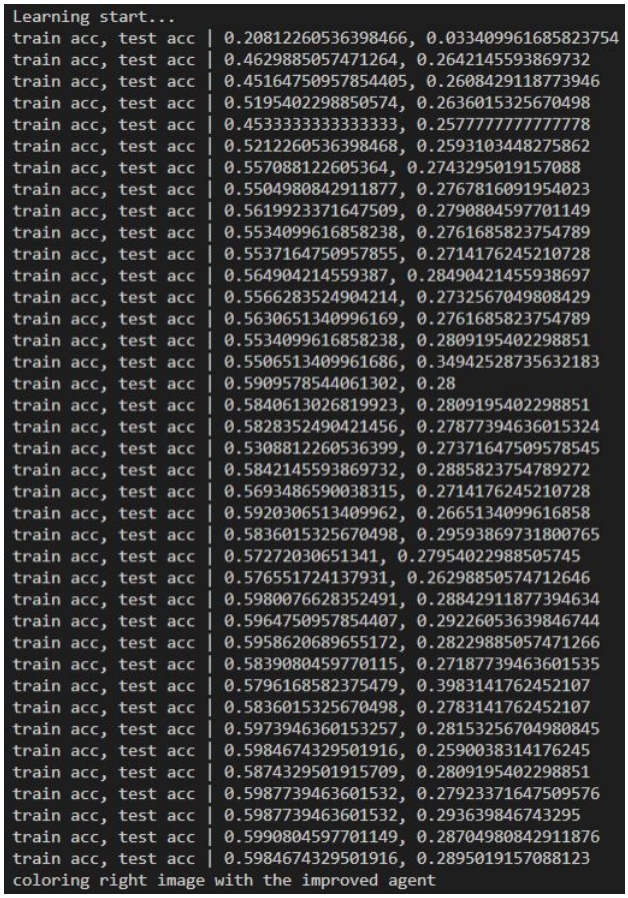
P4: How did you handle training your model? How did you avoid overfitting?

- To handle the training model, the training dataset needs to be flattened so that we can easily compare it with the testing data set. Next, each pixel will be converted into a grayscale that contains red, green, and blue values as a single list/vector. This way, the testing dataset's grayscale pixels can efficiently compare each pixel with a training datatset to find the best representative color from k-clustering.
- Currently, the training accuracy is between fifty-five and seventy-five percent, but the test accuracy lies in between ranges of twenty to thirty. (*See Figure: 9 below.*) We think this is suffienct enough to avoid overfitting. For a more accurate diagnosis of performance improvement, creating the validation set is probably recommended. However, using the validation set is limited to use properly under the situation that we're only training using the left hand side of the image. Therefore, we didn't implement the validation set but incremented the number of data sets through data augmentation to avoid overfitting.

  Moreover, there are other possible methods to avoid overfitting by using methods such as Dropout, Batch normalization, weight decay (L2 regularization), but these methods take a long time to implement.

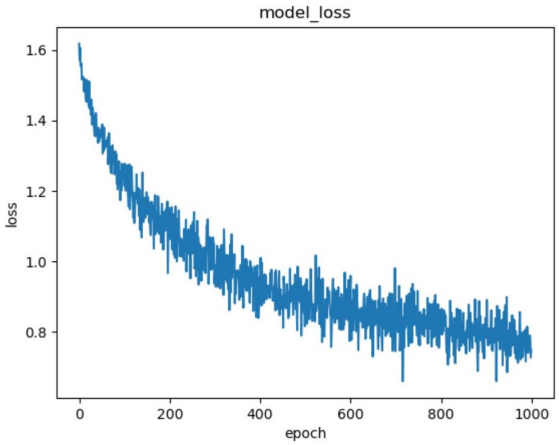


(a) Sample Run 1

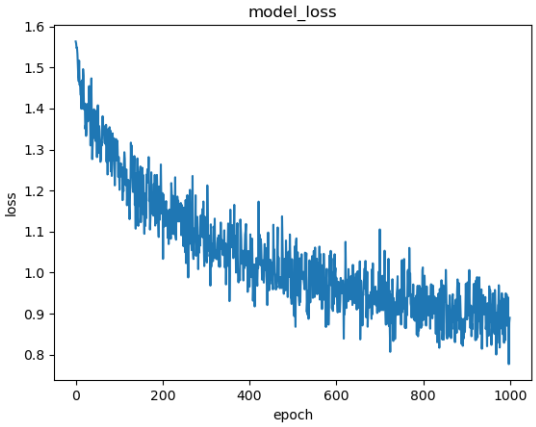


(b) Sample Run 2

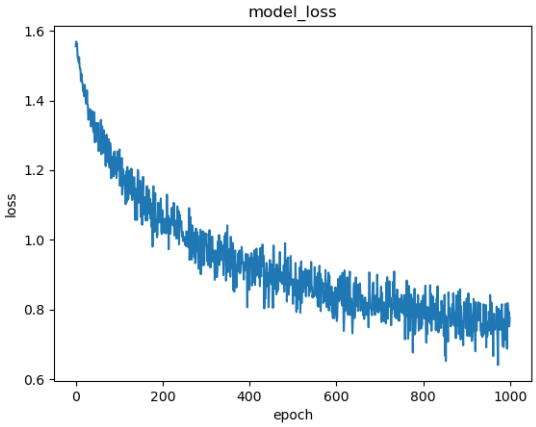
Figure 9: Training and Testing Accuracy
For Two Seperate Runs



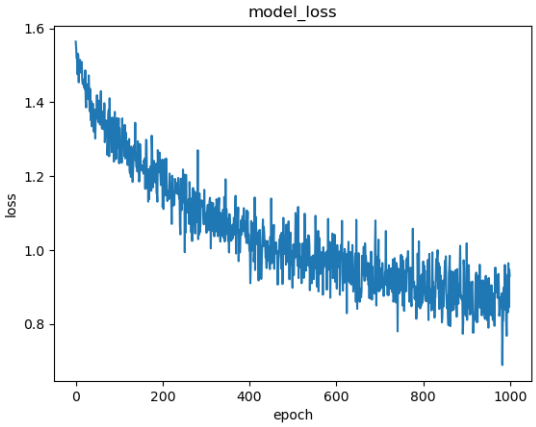
(a) Figure: 4 loss



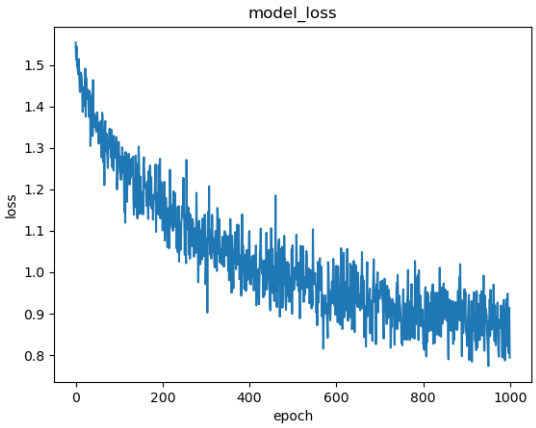
(b) Figure: 5 loss



(c) Figure: 6 loss



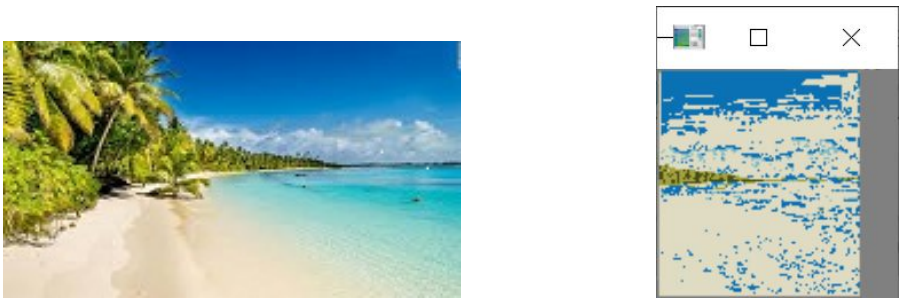
(d) Figure: 7 loss



(e) Figure: 8 loss

Figure 10: Model Loss Plots For Figures: 4 - 8
Data for 1000 epochs

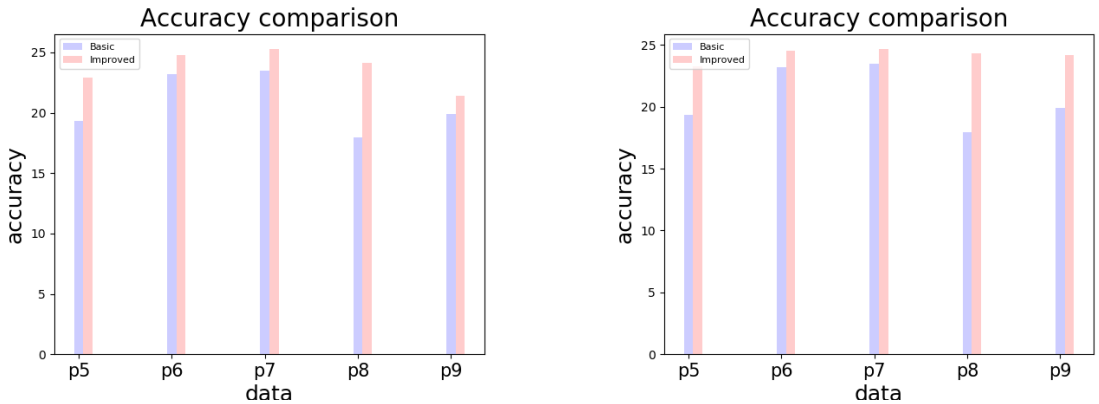
P5: An evaluation of the quality of your model compared to the basic agent. How can you quantify and qualify the differences between their performance? How can you make sure that the comparison is 'fair'?



(a) Original Image (b) Basic Agent

Figure 11: The Original Image Used vs. the Colored Image Using the Basic Agent

- Currently, the basic agent is a solid model for recoloring the testing image (*See Figure:11 above*) and we can quantify the quality by analyzing the similarity measurement plot. In the plot, most similarity errors/losses are formed near the x-axis. This result represents the loss function, which is the quantified difference between each pixels' tested dataset and actual output dataset value.
- Although, the performance for the improved agent doesn't have to be better than the basic agent, we still need to figure out why quantifying the differences between their performance is invalid. To check the experiment result, we should take a look at the loss function of each model to analyze how fast the model converges to higher accuracy. However, the *iters_num* parameter value (*used for the simple classification of the image*) is negligible when we are dealing with the small pixels of an image. Additionally, it takes too much time to experiment with a higher pixel size image than about 300 pixels because the time complexitiy is quite large. Hence, the loss function analysis would be fair if the image comparison is done with the same image object that used the same model in either the basic/improved agent but with different parameter values.



(a) (b)

Figure 12: Basic Agent vs Improved Agent in Terms of Accuracy

P6: How might you improve your model with sufficient time, energy, and resources?

There is an instrinsic problem that occurs in the improved agent:

- **The Problem**: if the sorting order of k-means clustered colors of the training dataset are the complete opposite of the testing dataset, it is highly likely that the right half of the image is colored incorrectly.(*See Figure: 13 below.*)
- **Possible Solution**: There are more efficient models than the one we implemented, to improve the prediction accuracy of image recognition by using four or more layered neural , convolutional, or recurrent neural networks.
- Additionally, using pre-processed training datasets from other pictures, excluding the image that we want to work with, should help the agent 'pick' the best (*the more valid*) colors that are accurate for recolorization, minimizing the chances of using the incorrect color.

(a) Original Image (b) Basic Agent

Figure 13: The Original Image Used vs. the Colored Image Using the Improved Agent
Produced from the two-layered neural network

Contributions

Jae: All code for: *main.py, performance_comparison.py, model.py, functions.py, gradient.py & analysis for report*

Cesar: Creation of LaTex doc