Sunny Cheng, Maxwell Wang
5 May 2021

Colorization

Professor Cowan
CS 440

Intro

Group members: Sunny Cheng (sc2040) and Maxwell Wang (mlw195)
Group contribution: Each group member contributed equally. Maxwell coded the basic agent implementation and utility functions. Sunny coded the improved agent. Both members worked together.
Group statement: All work here is our own, and we did not copy or take code online or from other students.

Basic Agent

The final result of the basic agent clearly resembles the original image. The shape of the original picture's subject has been preserved very well. Some clear disadvantages, however, is how the basic agent limits the color options to 5: as a result, many of the less common colors have been ignored and therefore the final result is a much more simplistic version of the original, like minimalist art. Nevertheless, the basic agent result does a good job at creating a simplified version of the original colored image.

We could loop over all the pixels and compare their values to the original colored image's pixels. The difference between pixels is computed using $|red_{original} - red_{new}| + |green_{original} - green_{new}| + |blue_{original} - blue_{new}|$. The total difference in all pixel values would indicate how much quality was lost in the process. A total difference of 0 would mean that no quality was lost and the newly colored picture looks exactly like the original.

The final result is numerically satisfying in how the 5 colors form cluster-like shapes in the image's pixel values. This makes sense because similarly colored pixels often appear next to each other in a group. This shows that the algorithm correctly detected that groups of pixels with many similar colors could be summarized into 1 of the 5 representative colors that we had selected.

Improved Agent

Our improved agent is called TRACE: Triple Regression Analysis Color Elector.

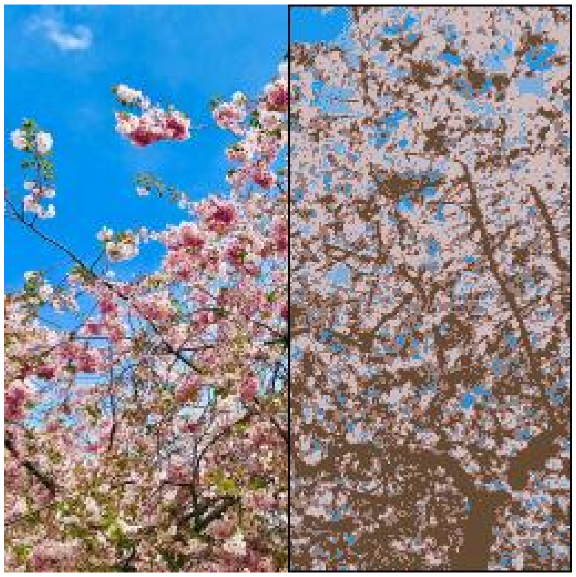
A linear model is a bad idea as linear model indicates that the data points are formed in such a way that the relationship between the grayscale value and the color value is strictly linear, but this is clearly not true because it is possible for colors to be in various different groupings depending upon the picture.

For our improved agent, we split the data set into three different data sets, one for each color (red, green, and blue). Each color would be trained separately, but using the same technique. We preprocess the image data by scaling RGB values, which can range from 0 to 255, down to 0 to 1. Then, once we obtain our output, we scale the values back up. The scaling is strictly linear. Our input space is a set of vectors $v \in [0,1]^{11}$, where the elements in the vector are $1, x_1, ..., x_5, x_1^2, ..., x_5^2$. x_i takes on 5 values: the value of the pixel itself, the pixel above it, the pixel below it, the pixel to the left, and the pixel to the right. By feature mapping the values of the pixels, we are able to ensure we are using a non-linear (quadratic in this case) model to allow for the best possible result. Our output space is a single value $x \in [0,1]$ which represents the red, green, or blue value of some particular pixel. Thus our model space is the set of functions $f : [0,1]^{11} \rightarrow [0,1]$. Our model f is simply $f(w,x) = w \cdot x$, i.e. the dot product of weights and the sample input. For loss, we use the L2 norm. That is, loss $= (f_w(x) - y)^2$ where x is in our input space and y is in our output space. For our learning algorithm, we use stochastic gradient descent. The details of it will be described in the next section.

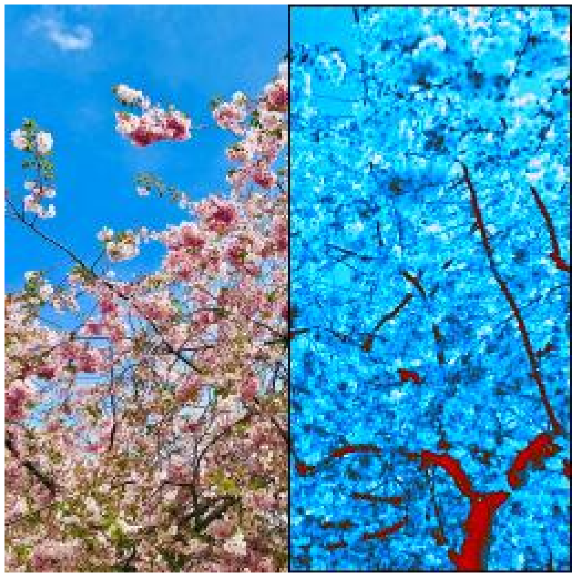
For our model, we decided on scaling between 0 and 1 to keep the processing simple. We decided to use a 5 pixel batch as a trade off between too many pixels with 9 and too little information with less than 5. Moreover, with our feature mapping to the squared values of the pixels, we wanted to keep dimensional at a minimum. We decided to add feature mapping as we knew that a linear model was not optimal for colorization. For weights, we initialize them all to zero at first. To update the weights, we run the learning algorithm. We used stochastic gradient descent, so we would update our weights according to $w' = w - \alpha \nabla Loss(f_w(x) - y)$. Of course, the gradient of this simple function is easy to compute, so we have $w' = w - \alpha(f_w(x) - y)x$, with the constant factor being absorbed within the alpha. We decided on batching, so we used a batch size of 300 samples, and would take 300 data points from our input space and compute the gradient update operation. It is important to note that we do not update w between each update: w stays constant until the end, where it is then updated as w' so all the gradients are computed simultaneously. For our learning rate, we experimented by looking at how fast or slow loss was, and we decided on 0.0003. This was small enough to ensure accuracy to find an maximum without overshooting, and large enough to have the loss decrease over a reasonable time. We then ran 25000 steps of this batching algorithm, as after 25000 steps the loss did not decrease very much.

We trained our model on each of the three colors separately and then combined the data at the end. Overfit becomes an issue if the training data set is too narrow and the model becomes too specified to that particular input, so we made sure to choose varied images that would likely not have this issue. We also tested our model on several different images to ensure that huge overfits did not occur.

We can compare the performance of models in two main ways. First, qualitatively, where we just subjectively look at the images produced and use our own judgement

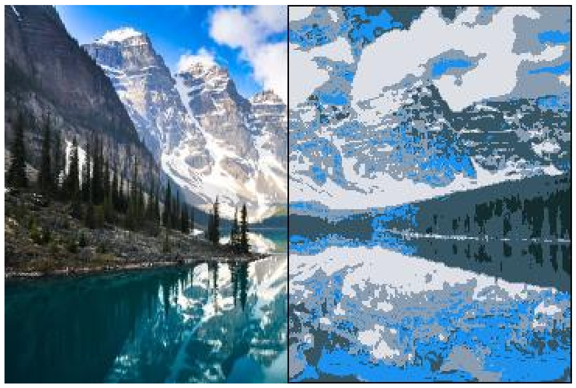


(a) Basic Agent

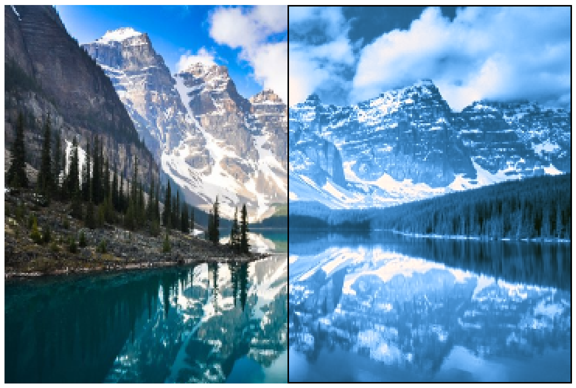


(b) Improved Agent

Figure 1: Cherry blossom comparison



(a) Basic Agent



(b) Improved Agent

Figure 2: Mountain comparison

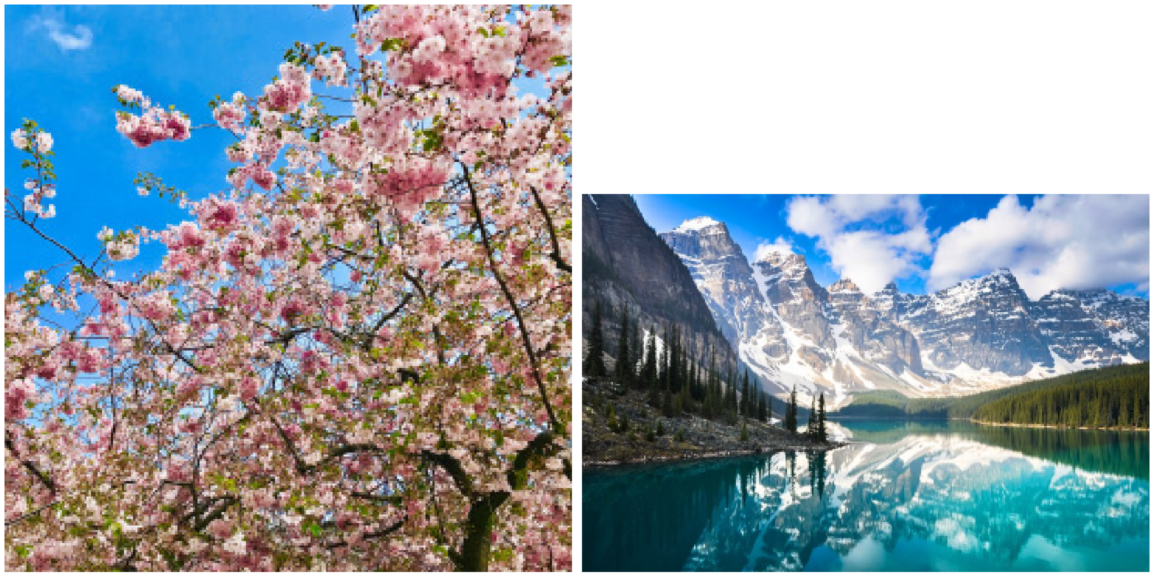

(a) Cherry Blossom (b) Mountain lake

Figure 3: Original images

to decide how well the images were colorized. This relies on human intuition and may be subject to one's individual bias, but is the natural way one would compare images. When qualitatively comparing the images, we find that in the cherry blossom photo the basic agent seems to have better color accuracy than the improved agent, as the improved agent seems to have trouble balancing the red channel with the brown of the branch and the pink of the flowers, but the improved agent is better when it comes to small details, which are smudged or blotted out in the basic agent. However, when it comes to the mountain photo, the basic agent's colors makes it look very unrealistic and incorrect. On the other hand, the improved agent seems to have a slightly different shade, but the details are all still there.
Secondly, we can compare the results quantitatively. While this seems like an objective view, it is still subjective as we have to subjectively decide on some function to quantify how well an image is colorized. For example, we could do the sum over all pixel's of their difference between red, green and blue values. We could also quantify it similar to the way we trained our improved agent: by splitting the results into the three color channels, and then comparing each channel's values directly to the ground truth's greyscale values, and taking the L2 norm over all pixels. Of course, in this formulation, the improved agent will win, as that is exactly how it was trained. Thus, it is extremely difficult to come up with a fair metric of comparison, but we do conclude that both agents have their own strengths and drawbacks

For this particular model, we could attempt to try different feature maps, or try logistic regression, or try to combine this regression with some type of classification to ensure that colors are not muted. But, with even more time and energy, we might try an entirely different model. It is clear that regression seems to have its limits when dealing with colorizing images. Thus, with additional time and energy, we would try implementing a neural net or a convolutional neural network as those are said to perform well for images. Resources could make a huge difference as well: we would be able to vary many of our parameters and try larger batch sizes, smaller learning rates, and train for more steps to maximize the amount of information learned from the data.