CS440 - Spring 2021 - Project 4

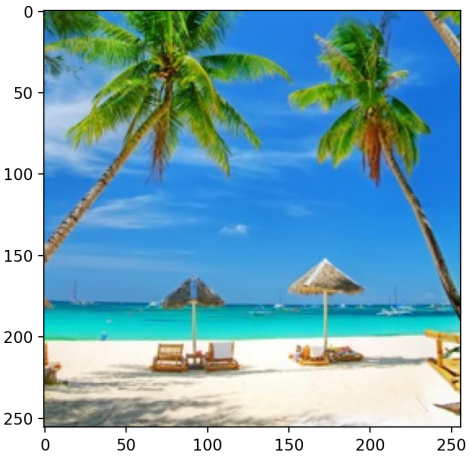
Gal Zandani (gz113) and Vaishnavi Manthena (vm504)

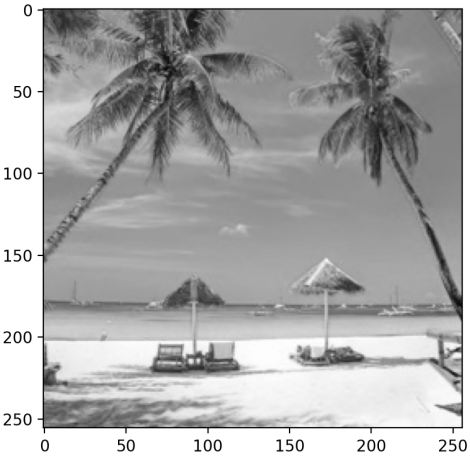
May 5, 2021

The Basic Coloring Agent

Our basic agent is designed following by the specifications in the assignment description and generates a colored image by the steps. We initially used this image

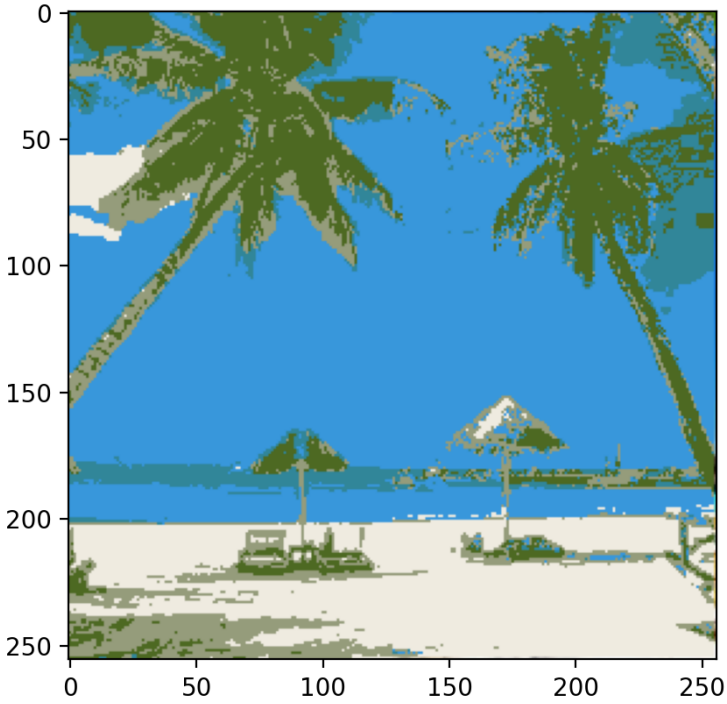
which converted to this one after using the cv2 python library and the function imread() to the colored image shown below. And Our function which transformed this image to black and white returned the image below.





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

This is our final result for the basic agent



This final result looks similar to our original picture and it captures the major details in it.
We measured the quality of the final result by writing a method, detectDiff(), for detecting image differences which returns a numerical value between 0 and 1. This value corresponds to the Structural Similarity Index (SSIM) between the two images and it gives a good indication to the similarity between the two images. The value we got for the original image and this result is 0.6912248256773664.

Bonus: Instead of doing a 5-nearest neighbor based 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 best number of representative colors to pick for the data is generated through plotting a graph of average distance of a point to the nearest cluster and values of k (number of clusters). We can use this elbow method to get the value of 'k' where the graph starts to level off. This is the point where our clustering algorithms starts to show very less improvement on further increasing the value of k.

The Improved Agent

Our improved agent Clever Acronym: Generalization. We chose this acronym because we are attempting stochastic gradient descent which tries to adjust to all the input and data. Also, we were thinking about ways to prevent overfitting so that the model could work well with new data.

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

Input Space: 9 dimensional vectors Xi. The components of each of these vectors are the gray scale values of each pixel in a 3*3 patch of the training gray scale half of the picture. We make these inputs 10 dimensional to add a '1' component at the starting of each vector.

Output Space: This algorithm would produce an integer in the interval [0,255] for every of the three components of the (r, g, b) 3-dimensional tuples. So, $r_i \in [0, 255]$, $g_i \in [0, 255]$, and $b_i \in [0, 255]$ accordingly. These three values together would compose the rgb color of a certain pixel in the picture.

Model Space: The model space we use is $f_w(x) = 255 * \sigma(w \cdot X)$. The sigmoid function is defined as sigmoid(Z) = $1/(1+e^{-Z})$. We chose this model space since it is non linear and produces a value between 0 and 255. We will pick a model from this model space for each of the three color components (red, blue, green).

Error/ Loss Function: The loss function we use here is separated into the three outputs we compute. So, there is a loss function for each of these (r, g, b) values calculated. So, the functions are $(r_i - f_w(x_i))^2$, $(g_i - f_w(x_i))^2$, and $(b_i - f_w(x_i))^2$.
The derivative of the loss functions, considering the one of the red color $L_i = (r_i - f_w(x_i))^2$ as an example, is computed here:
$\Delta w L_i$

$\frac{\delta L_i}{\delta w_j} = 2(r_i - f_w(x_i)) * \frac{\delta}{\delta w_j}(r_i - f_w(x_i))$

$\frac{\delta}{\delta w_j}(r_i - f_w(x_i)) = -\frac{\delta}{\delta w_j} f_w(x_i) = -255\frac{\delta}{\delta w_j}\sigma(w \cdot x_i) = -255\sigma'\frac{\delta}{\delta w_j}(w \cdot x_i)$
$= -255\sigma' {x_i}^j = -255\sigma(w \cdot x_i)(1 - \sigma(w \cdot x_i)){x_i}^j$

$= -2(r_i - f_w(x_i)) * f_w(x_i)(1 - \sigma(w \cdot (x_i))){x_i}^j$

Therefore, $\frac{\delta L_i}{\delta w_j} = -2(r_i - f_w(x_i)) f_w(x_i)(1 - \sigma(w \cdot (x_i))){x_i}^j$.

Learning Algorithm: We use stochastic gradient descent since we have a continuous model space and loss function. Below is the stochastic gradient descent for the color red. The algorithm for other colors is very similar.

Step 1: Initially we start with a random 10 dimensional vector w. The components of w are random real numbers.

Step 2: Pick a random 3*3 patch in the training the training algorithm. Let this be X_i.

Step 3: Update the vector w in the following way:

$w_{t+1} = w_t - \alpha$ * gradient of $L_w(Xi)$ with respect to w.

A calculated above : gradient of $L_w(Xi)$ with respect to
$w = 2 * (f_w(X_i)) * (f_w(X_i) - r_i) * (1 - \sigma(w \cdot x_i)) * X_i$

Here alpha is the learning rate.

Repeat the steps 2, 3 until the difference between w_{t+1} and w_t starts to level off.

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

The weights were actually learned during the stochastic gradient descent. However, while choosing the initial random weight vector, instead of choosing a random real number for each component of the weight we decided to choose a random real number between -1 and +1. This would cause the w to not be too high or too low initially. From this point the learning algorithm makes the required adjustments.

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

The pre-processing of the input data we used is for the 9 dimensional vectors representing the 3x3 patches of the pixels in the image, we added another component as the first element of the vector which is 1.

The pre-processing of the output data we used is combining the three different values that we get from r, g, and b into one 3 dimensional tuple. Since the computations for r_i, g_i, and b_i are done separately, for the output data to be valid we have to compose the corresponding r, g, and b with the same values of i into the same tuple.

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

We thought of two ways that might help prevent overfitting:
1. When we receive a model from the training algorithm (stochastic gradient decsent), we thought of discarding it if the components of the weight vector are really large or there is a lot of variation between the different components of the vector.
2. We also thought of just modifying the loss function by at a term that is the square of the 2-norm of vector 'w.' This process of adding a penalty for w values with large coefficients is the idea of L2 regularization. If we implemented this we planned to modify the gradient calculation (the gradient of the loss function at data point i with respect to w) accordingly.

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'?

We can quantify the differences between the two images resulting from the basic agent and the improved agent by comparing both of them to the original image computing the Structural Similarity Index (SSIM) between the two images. We can also compute this value for comparing between the two results of the basic agent and the improved agent.
We can qualify the differences between the basic agent and the improved agent images by passing through the pixels of the images and coloring in black (or any color that is not one of the 6 representative colors) to visualize which ones are distinct between them. We can also do this approach by first converting the two pictures to gray scale and them the differences between the colors would be more distinct, continuing with the method of coloring the different pixels would be a good visualization of the differences between the two pictures.
These two are fair representations since they approach to these two images in the same way so there is no biases in the results or the input.

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

We could improve our improved agent model with sufficient time, energy, and resources by expressing the agent as a Neural Networks algorithm. We could use non linear activation functions for each layer. The Neural Networks capture the patterns of the elements of the pictures so that we are able to train the algorithm to learn the different color distribution of the different elements in the picture. This would create a more similar picture to the original one after each round of the algorithm with some feedback.

Bonus: why is a linear model a bad idea?

A linear model is a bad idea because it does not reflect the properties of the data. Specifically if we use (W dot product X), the value of this could be anywhere between negative infinity to positive infinity. However the output of say the model for red should be between 0 and 255 because other values will not have any meaning. Hence, using the non linear function will generate meaningful outputs. Moreover linear models may run into the risk of under fitting due to there simplicity whereas non linear models may capture more features of the data.

Bonus

Research a ML framework like scikitlearn or tensorflow, and build a solution to this problem using this framework that beats your improved agent.

After researching the ML framework named scikitlearn, a machine learning library for python, which can help us produce a better agent than our improved agent. We can combine the algorithms for the basic agent and the improved agent into one. One of the significant advantages with with this library is that each operation takes approximately a second.
The functions of scikitlearn can be used in order to find the optimized number of clusters, number of representative colors, and the color quantization using k means. After using this idea from the basic agent 1 (k means), we would have effectively reduced the output space to a more manageable size. From here, we can use our improved agent (stochastic gradient descent) to get the final result.

Summary

We both worked on the logic and code for the problems in this project, we brain stormed ideas, and agreed on the solutions we got it. We met on Zoom to write together the code for this project to generate the image, converting it to gray scale, the basic agents algorithm, and the advanced agent. We collaboratively found solutions to issues and bugs we faced while testing and running the code. We both were running test cases on the code outside of our Zoom meetings (generating gray scale images, running the basic agent with images and checking the results). In this report, we wrote the basic agent and advanced agent sections together. Mostly it was completely collaborative but the below are some individual things we did:

Gal: Got the latex document ready with the relevant sections, debugged the code and found solutions for the issues, and offered ideas or more efficient ways of implementations while working on the code together. Generated image solutions of the basic agent and found a way to compare it to the original one numerically. I wrote the basic agent, some questions of the improved agent, and one of the bonus questions.

Vaishnavi:
Coded for the basic agent along with gal through zoom. Wrote a rough version of the derivation of the gradient of the loss function with respect to w. Went through Gal's answers in the report and added on a few points. Wrote the bonus for the basic agent and the one about why linear models are a bad idea. Also, wrote a few of the improved agent questions.

We both wrote some methods of the code, debugged separately, and contributed to the answers on this report. This write up was written by both Gal and Vaishnavi, each one of us was responsible for some questions and we added on to each other's answers.

I read and abided by the rules laid out, I have not used anyone else's work for our project, my work is only my own and my partner's.
Gal
Vaishnavi