Project 4 Report

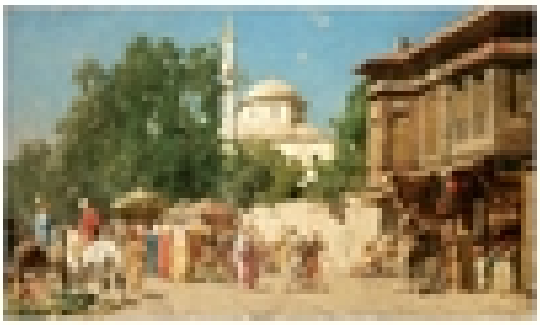
Tristan Kilper (twk28)

5 May 2021

See attached zip file for the code used to solve the problems in this project. All code was written in Python and tested using Visual Studio Code.

1 The Testing Image

The image that was used in the algorithms detailed below is the following:



It is a digital image of Street in Damascus by Alberto Pasini from 1871. Its pixel size was decreased to 118 x 77. I imported and worked with the image in my code using the OpenCV library for Python.

2 The Basic Agent

2.1 Results

I designed the basic agent algorithm as specified by project description. These are results produced by the basic agent program:

(a) Colored with k-means clustering

(b) Colored with basic agent algorithm

Figure 1: BasicAgent coloring results

The image on the left is the left half of the original image colored with the five most representative colors found using k-means clustering. The image on the right is the right half of the original image colored using the basic agent algorithm.

3 The Improved Agent

The name of the Improved Solution is **PICASSO**, which stands for Picture In Color After Separate Sigmoid Origins because the final model is separated into three different equations each utilizing the sigmoid equation.

3.1 Specification

The solution takes the following form:

$$f_c(x) = \sigma(w_c.x) * 255 \tag{1}$$

The solution takes a gray scale image and breaks it down into $3x3$ patches of pixels. The nine gray scale values from the pixels in the patches form a nine dimensional vector that acts as the input to the model, x.

Since the model needs to predict the red, blue, and green values of pixels, it consists of three different equations for $c = r, b, g$. The outputs, $f_c(x)$ are integer values between 0 and 255 which represent the predicted color values of the middle pixel in the $3x3$ input patch.

The loss function is the sum of squares error formula:

$$L = (f_c(x) - y_c)^2 \tag{2}$$

I found this to be the simplest and best formula to use since the output values are integers.

The learning algorithm used in finding the coefficients for the model was stochastic gradient descent to help make for computational efficiency and high accuracy. The details of the training are outlined in the training process section.

3.2 Parameters

I started with a simple, single layer parametric linear model, $f_c(x) = w_c.x$. Since I knew that the color values are between 0 and 255, it made sense to try to find a way to restrict the output values of the model to those constraints instead of needing to deal with values outside of that range later. To accomplish this, I decided to use the sigmoid function as an activation function on $w_c.x$ and multiply that result by 255. The sigmoid function is the following:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

The sigmoid function returns values in between 0 and 1, so by multiplying the result of $\sigma(w_c.x)$ by 255, the final predicted model value will be in between 0 and 255. Additionally, since there are three separate model equations with nine different weight parameters each, there are 27 total weight values in the complete model.

3.3 Input & Output Data

The input image is pre-processed in the following way. It is broken into its $3x3$ patches of pixels similarly to the basic agent process and the value of each pixel is recorded as an entry into a new nine dimensional vector starting with the top left pixel and going left to right. This would make the center pixel value be in the fifth position in the vector and the bottom right pixel value be in the last position. This vector then acts as the direct input to the model.

The three output values are combined into a three dimensional vector that replaces the existing color vector of the middle pixel in the current pixel patch.

3.4 The Training Process

As stated above, the training algorithm that I used for each equation (1) for $c = r, b, g$ was stochastic gradient descent. Hence, I updated the weights at each data point instead of after processing the entire data set. At each data point x, all of the weights w in the current equation were updated in the following way. I first calculated the gradient of the loss function in respect to each weight w_i:

$$\nabla_{w_i} L = \nabla_{w_i} (f_c(x) - y_c)^2 \tag{4}$$

Using the fact that $f_c(x)$ is equal to (1), I use the chain rule to end up with the following computation:

$$\nabla_{w_i} L = 2(255\sigma(w_c.x) - y_c) * 255\sigma^{'}(w_c.x)x_i \tag{5}$$

Where the derivative of the sigmoid function, $\sigma^{'(x)}$ is the following:

$$\sigma^{'}(x) = \frac{e^{-x}}{(1+e^{-x})^{-2}} \tag{6}$$

Then, the step is calculated in the following way:

$$\delta = lr * \nabla_{w_i} L \tag{7}$$

lr is the learning rate or step size. Finally, the weights are updated like so:

$$w_i(new) = w_i(old) - \delta \tag{8}$$

This process is repeated until convergence, which is defined by the absolute difference of the actual and predicted function values being under a given constant tolerance. I initialized the tolerance to 0.0001. The consequence of running this training algorithm on the three model equations $f_c(x)$ for $c = r, b, g$ is generating the best values for the 27 weights.

3.5 Performance vs. the Basic Agent

In the end, I was unable to find optimal weight values for the final model. The weight values would increase or decrease to extremely high or extremely low values very quickly, and I was unable to adjust the learning rate in time to accommodate for the over fitting.

Hence, I was unable to come up with a final improved model to compare against the basic agent. If I had more time, I would probably plot the loss function as I train to help better discover the issue.

In order to compare the performances of the improved and basic agents, I would have measured the euclidean distance between the actual color and predicted colors of each agent's final image's pixels and taken the sum of all of those values. Then, the agent with the lower total distance would be considered more accurate.

3.6 Ideas for Improvement

With sufficient time, energy, and resources, I could improve the improved model in the following ways.

- *Larger Input*: I could increase the amount of information in the input by increasing the dimensions of the pixel patches and corresponding input vectors to $5x5$, $7x7$, etc.
- *More layers*: I could expand the three equations into a neural networks with more than one layer.

Work Contribution and Integrity Statement

All work on this project including coding and preparing the report was done by myself, Tristan Kilper (twk28).

In completing this project I read and abided by the rules of the project. I did not use anyone else's work, the work done in the project is my own.