Rutgers University

Artificial Intelligence
01:198:440:01

Assignment: Colorization

Authors:
Azim Khan
Aditya Anikode
Vrinda Jain
Net ids (auk11,aaa387,vj140)

May 6, 2021

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

1 Abstract

The goal of this assignment was to demonstrate our understanding of the basic techniques used in supervised learning and computer vision. We will be building a model to generate a coloring of a black and white image in Python. We will be building a basic agent that uses a k-means clustering method to determine the best 5 representative colors when converting grayscale images to a colored one. To receive an accurate image, we will make an improved agent that utilizes neural networks.

2 Introduction

Image colorization is the process of taking a grayscale image as input and producing a colorized image as an output. This has been a big topic in the deep learning field with plenty of research going on in order to obtain as realistic colorized images as possible. A deep learning approach to image colorization is through the use of Convolutional Neural Networks (CNN) which analyzes many sets of grayscale images and trains on them to learn. With enough time and data sample, the algorithm is able to produce quite accurate colorized images. In order to get a better understanding on supervised learning and neural networks, we created this project which trains an agent on only one image in an attempt to produce a colorized version of the given image in grayscale. Through the use of one half of the image as training data and the other half as testing to create the most accurate colorized image.

3 Project Approach

This project required a variety of additional knowledge outside of lectures. While during lectures we learned about supervised learning and neural networks however it was mostly mathematical, which was really helpful in understanding these complicated topics but applying these knowledge was more difficult. Therefore, we initially researched about image colorization and how that is done, then looked more into how it is done with AI or through supervised learning. From our research we learned about color perception, which is an algorithm to map the pixel color to its closest original color, we implemented this in our code as colordistance. We also found a very helpful python library skimage, which we used to convert our original image to grayscale using the function, Gray = 0.2125 R + 0.7154 G + 0.0721 B. We also used skit-learn but simply to create the KD tree using NearestNeighbors, where we store the instances of learning data for the agent which is used to create our model. KD Tree picks a dimension, finds the median and splits the data and repeats the process. For a new data point, a region containing the point is found and compared to all points in the region. We also learned that machine learning classification is important and needed to help the computer classify in order for training, so from lecture we learned about one hot encoding which we implemented in our improved agent (further explained in report).

4 Code Explanation

4.1 Basic Agent Class AKA Patrick (Because he is slow but we still love him)

Figure 1: Patrick

The basic agent algorithm uses a five color approach in analyzing a provided input image. Here, a color distance function is implemented which calculates and compares two RGB numbers to find the difference in R, G, and B individual values between the two points. The Five Color function parses through the length and width of the image and uses the color distance to take the minimum distance, using the np.array. We implement a black and white training, black and white prediction, color training, and actual color using the five color image and the rgb2gray built in function. Furthermore, the AverageColor method uses the surrounding 3x3 cells of a point and looks at the red, green, and blue pixels in the area. If the point is within boundaries, the total pixel value is updated and the patch count is incremented, and finally the average of all three (RGB) values is

taken into consideration and returned. The same process is repeated for the AverageBlackAndWhite function. A dictionary of averages where the keys are points and values are the average black and white values for points is done. The RealVsPredicted is then done where we use the numpy function array equal to compare the points of the predicted image values and the actual image values, and this difference is noted as the incorrectness factor, which is the percentage used to analyze the efficiency of our program. For the basic agent function, color is matched and sorted if the distance is less than or equal to 10 and added to the match list. The KD Tree is implemented in this method, where kneighbors and fit are used to assign values to the kdtree. RGB prediction is set as ColorPrediction and returned containing the target red, green, and blue values. Finally, this information is visualized when plotted in the main method, where the recolored, black and white predicted, and color predicted image is displayed.

4.1.1 K Means Class

The K-Means algorithm takes coordinate points from the image and assigns them into clusters. The euclidean distance is used to find the distance between two given points. Furthermore, we are converting the pixels from the image into points using the PixelsToPoints function, which is then implemented in the KMeansClustering class, where the center of a given cluster is found and points are sorted by assigning them to clusters. This sorting is done by using the euclideanDistance function, and then the matchPoints function assigns points to clusters by updating the clusters with any new points that are found to match. Finally, we implement the Colors function, which uses a five color count set to 5 and returns the RGB values, ranging from 0 to 255 for a list of clusters, and displays it in the terminal for reading.

4.2 Improved Agent Class AKA Mega Mind (Machine Educated Graphics Algorithm and Media Intelligent Neural Displayer)

This agent is a combination of the K-Means and neural network file. It uses the 5 colors gathered from K-Means and the training data gathered from the neural network to analyze the image.The one hot code function is used for classification of the colors, in our improved agent we get 5 colors from recolorization which we classify using one hot code which has a list with a length of 5 where each color corresponds to the index on the list. Essentially one hot code converts each specific color from the five colors to one-hot code which is then used for training the agent. Reverse one hot converts the one-hot back to the original color.

Figure 2: Mega Mind

4.2.1 Neural Network Class

Each neuron is characterized by its weight, bias and activation function. After that, an input is given to the input layer; the neurons perform a linear transformation on this input using the weights and biases. Activation function is one of the building blocks on Neural Network and it is applied:

$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

Figure 3: Activation Function Formula

We tested tanh and sigmoid activation function; tanh is very similar to the sigmoid function but the only difference is that it is symmetric around the origin. The range of values in this case is from -1 to 1. Thus the inputs to the next layers will not always be of the same sign. Sigmoid, is not symmetric around zero. So output of all the neurons will be of the same sign. We tried both activation keys and got a similar % error, however using tanh gave us less error compared to sigmoid for certain images, but vice versa for others. Sigmoid can cause a neural network to get "stuck" during training. This is because, if a strongly-negative input is provided

to the logistic sigmoid, it outputs a value, which is very near to zero. Because of this behaviour, updation of weights will be slow and they are less regularly updated. However, tanh is unlikely to get "stuck" during training, so having a lower error for one image using sigmoid compared to tanh did not make sense.

Next we have our model fit method where we input our data, training data, learning rate, and epoch. The epoch is to be done when whole training data is passed through the entire network once. As the number of epochs increases, more number of times the weight is changed in the neural network and the curve goes from underfitting to optimal to overfitting curve. Using 10,000 for the Epoch gave us the least error; to get a much more fit model we would increase that number, but it would slow down the code even more.

We will then have forward and backward propagating through the neural network and update the weights. Lastly, the predict function takes in input data and uses the activation function and weights to return a prediction based on what the neural network has processed.

[2]

Representation

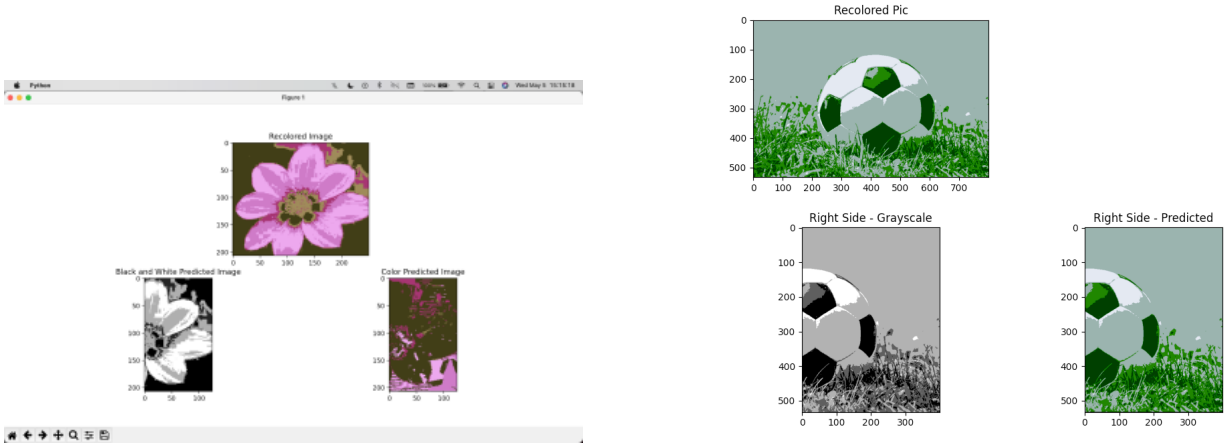

Figure 4: Left is Basic Agent results and Right is Improved Agent Result

Performance

Our graphs agree with our intuition as the improved agent provides us with better accuracy compared to the basic agent

Structure

Weights are learnable parameters inside a network because a neural network is teachable and it can randomize both the weight values before learning begins. As the training continues, the weight is adjusted toward the desired values. The neural network contains hidden layers which apply transformations to the input data. The final layer of the neural network is also known as the output layer which tunes the inputs from the hidden layers to produce the desired results in a specified range. These weights are present in the hidden layers of the network. We had initialized them to small random values based on the number of layers. Some things we had to take into consideration was which Activation Function to use and we chose to use both sigmoid and tanh. Sometimes tanh resulted in better results and sometimes sigmoid gave better results.

Since our basic agent had a long processing time, we used images with lower sizes as our input. We used https://www.reduceimages.com/ to lower the resolution and size. However, for the improved agent we kept the input as the original image.

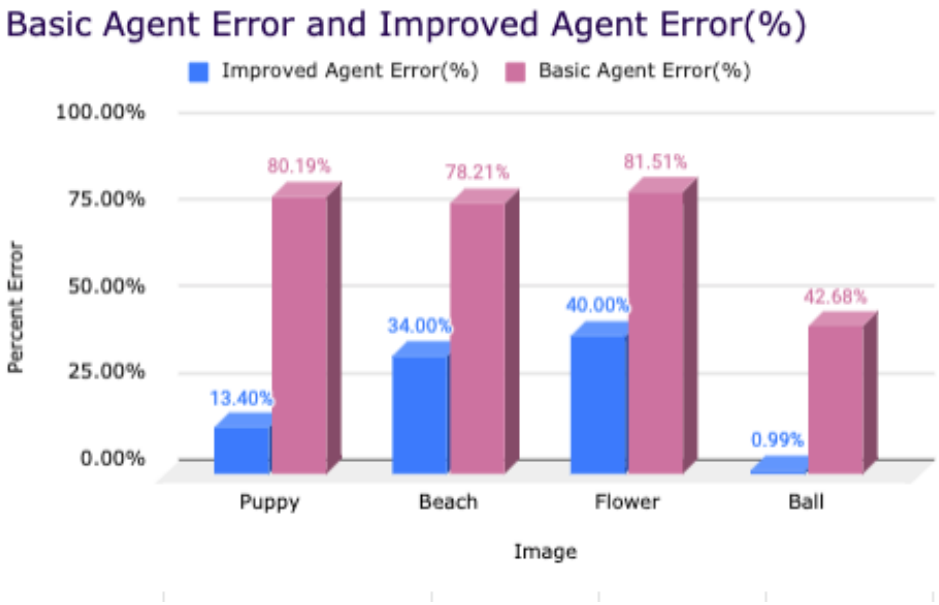

Figure 5: Basic Agent Error and Improved Agent Percent Error

Performance

Our graphs agree with our intuition as the improved agent provides us with better accuracy compared to the basic agent

Modeling/Training

We played with the Epoch values in the neural network function to avoid overfitting. We chose a value that would give us the least error. When we raised it too much to 70000, the percent error started going back up again, so we set 10000 as our threshold value. Altering the learning rate also prevents overfitting. The rate we set it to was 0.1 and it was the best rate to use as it resulted in the lowest error.

In order to calculate the performance of our agent we implemented a pixel to pixel comparison between the original image and predicted image in both of our agents. We created a function called RealVsPredicted which essentially finds the correctness of our prediction by comparing it to the original image pixel value, in other words calculating the percent error/difference. We make a fair judgement for both agents by keeping the scale of performance the same.

Knowing the limitations of the RGB values narrows the range of cluster sorting that needs to take place. Moreover, by using the five most prominent colors, the range of the color spectrum is reduced significantly for efficiency rather than for better image quality.

Our model currently uses square to square comparison or pixel to pixel comparison, however if we tried a different approach and used logistic regression, where we use probabilistic modeling for classification, we might have a different outcome. Logistic regression uses probability to classify instead of predicting it like linear regression. Colors are predictable, for instance grasses are usually green or skies are usually blue, based on that our agent would be able to classify a certain class/object with a certain color (this is my assumption as I am still unsure on a few topics). However that would a deeper dive into logistic regression and machine learning which could be implemented/upgraded in the future.

Bonus

Base your agent on a parametric model, but a non-linear one (+5 bonus points: why is a linear model a bad idea?) In a parametric model we would have a finite number of parameters which will be able to capture all the information and predictions within the set. However if it is a non-parametric model it will add a lot of assumptions which will skew and over simplify the predictions. Linear model is a bad idea since it is very simple and one dimensional which makes it sensitive to outliers.

We attempted to recreate this project using autoencoder, which is a type of unsupervised neural network learning technique that fits a model with X and X, in other words the input is the same as the output. Our Autoencoder trains the model by using an image on top of an image and reconstructing the original image back. In our case we attempted to create a model that takes in a high pixel density image and down scaling to a much smaller pixel size in order to train its weights and adjust the biases and then late upscaling it to the original image. We trained our model with RELU activation function and with the use of tensorflow. Our model was trained using 20 different images and then once the model was trained we input a completely new image for it to colorize. We also trained our model for only 100 epochs since we were short on time as we were nearing the deadline, but even with the short amount of training and little resources, it was able to perform quite efficiently. While we could not generate a performance number for this model to compare it to the performance of the improved agent, However we made a resulting picture and by looking at the picture it is easy to see that the model outperforms our improved agent.

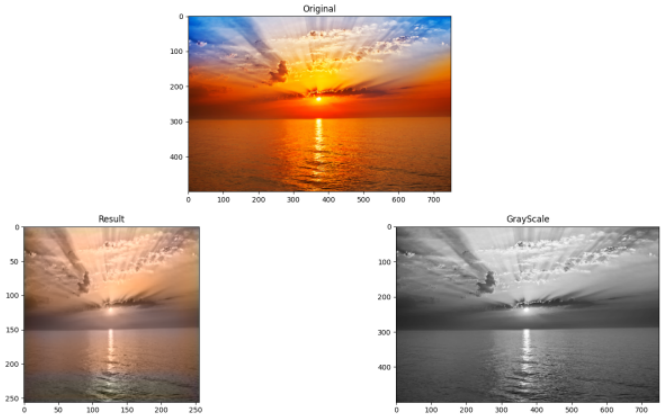

Figure 6: Results from Bonus Algorithm

Conclusion

Neural Network is the core of AI and Machine learning. The main inspiration for neural networks came from the biological neurons within the human body resulting in a related action performed by the body in response, however the right neuron must be triggered for the right action and the accuracy of the action depends on the training. Similarly, the type of neural network architecture used in training for a certain task is also important. We know that Neural networks are used in computational models and can be used here for image colorization, however there are different types of neural networks such as Convolution Neural Networks (CNN), Deep Neural Network(DNN), Multi-Layer Perceptron (MLP) and many others. While each architecture is useful, the application of these architectures is what matters. For example, MLP is a bi-directional meaning it goes back and forth with its inputs and predicts using weights and activation functions in order to self-adjust, it is similar to what we used in our nerual network. However CNN is more efficient with a three-dimensional arrangement, where it understands the image in parts and is able to detect the edges which allows it to make better predictions. So the decision on which architecture to use for which application is quite important as both have complex design and are quite slow. So in the future if we wanted to implement deep learning using AI, we would use CNN which has been found to be the most effective in image processing, image colorization and computer vision.

Contribution statements

In order to divide the work fairly, we decided to make a server on discord where we would meet regularly and work on the project together. We started off by creating pseudo code for each search algorithm and implementing them as we went. Our group as not as proficient as the project needed us to be, so we spent a lot of time learning things on our own in order to write our code.Overall, I believe everyone worked hard and fairly on this project. - Azim Khan

Our group initially struggled with the coding logic for our project. This is because we are all novice programmers and did not have much exposure to Python prior to this project. Therefore, we initially started by making pseudocode for each method and algorithm we were tasked with creating. We spent a lot of time individually learning through youtube tutorials and reading online materials regarding the topics of interest. After individually building our skills, we came together and communicated via discord and messenger video call to bring our ideas to the table. We met regularly, pretty much every day, and diligently worked on the project. Although we were novice coders and struggled initially with the project, we were able to piece it together little by little to form a finished final project.- Aditya Anikode

This project was a challenge. My group met regularly via discord and facebook messenger and collaborated fairly and evenly on this project. We initially struggled with writing the code, so we started by collectively writing pseudo code to better understand the task at hand. Moreover, we spent a lot of time individually learning the concepts and ideas behind each search algorithm, so we could make progress when we met. Overall, I believe everyone put a great deal of effort into this project and the work was divided equally. Although we struggled, a great deal of information was learned. - Vrinda Jain

Honor Code

We have read and abided by the rules of the project, we have not used anyone else's work for the project, and we did all of the work by ourselves.

[5]
[2]
[1]
[3]
[4]

References

[1] sklearn.preprocessing.onehotencoder.

[2] Wesley Cowan. Assignment 4- colorization, 2021.

[3] Dishashree Gupta. Activation functions: Fundamentals of deep learning, Jul 2020.

[4] Dustin Stansbury. Derivation: Derivatives for common neural network activation functions, Jun 2020.

[5] Vasudev. What is one hot encoding? why and when do you have to use it?, Aug 2007.