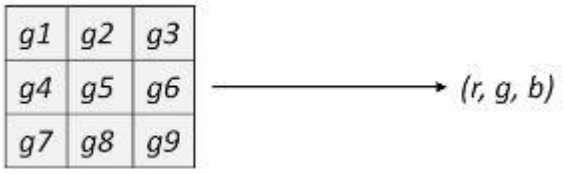
CS 520
Assignment 4
Colorization

Adhish Shrivastava, Hiral Nagda, Yash Nisar

1. Representing the Process:

As we cannot map a single gray value to a corresponding color vector which consists of 3 values (r, g, b). We decided to use a 3 x 3 pixel windows on the grayscale image to represent the center pixel. Consider this window sliding pixel by pixel. These 9 values map a center pixel of which is the color vector.



2. Data:

To begin with, we picked up a dataset which purely consisted of images of apples, in the RGB color system.

In RGB, a color is defined as a mixture of pure red, green, and blue lights of various strengths. Each of the red, green and blue light levels is encoded as a number in the range 0-255, with 0 meaning zero light and 255 meaning maximum light. So for example (red=255, green=100, blue=0) is a color where red is maximum, green is medium, and blue is not present at all, resulting in a shade of orange. In this way, specifying the brightness 0-255 for the red, blue, and green color components of the pixel, any color can be formed. Thus any RGB image is represented as superimposed matrices of red, green, and blue.

We first pre-processed the image by converting from RGB to Grayscale. We could achieve this by using the ITU-R 601-2 luma transform:

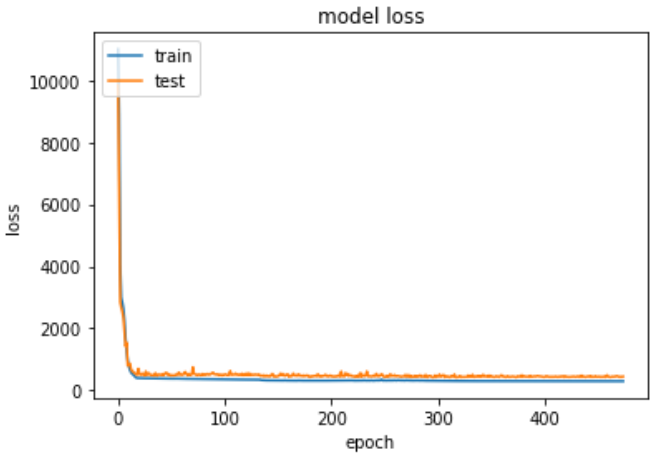
grayscale_value = R * 299/1000 + G * 587/1000 + B * 114/1000

For grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from **0 to 255**. Typically 0 is taken to be black, and 255 is taken to be white.

The images that we've used to train the model are of the dimensions 100 pixels * 100 pixels. So, we have a 2-D matrix of size 10,000 values between 0 - 255 representing the brightness of each pixel. For each of the 10,000 pixel values, we map it to the neighbouring pixel values (that represent the center pixel) and convert it to a CSV file. We do this for all 10 images in the training dataset and append it to the CSV file. Our CSV file would finally consist of 100,000 rows and 9 columns. Since our Artificial Neural Network has 9 input nodes, we feed each line of the CSV to the Neural Network.

3. Evaluating the Model:

We can evaluate how good our model is by plotting model loss vs number of epochs.



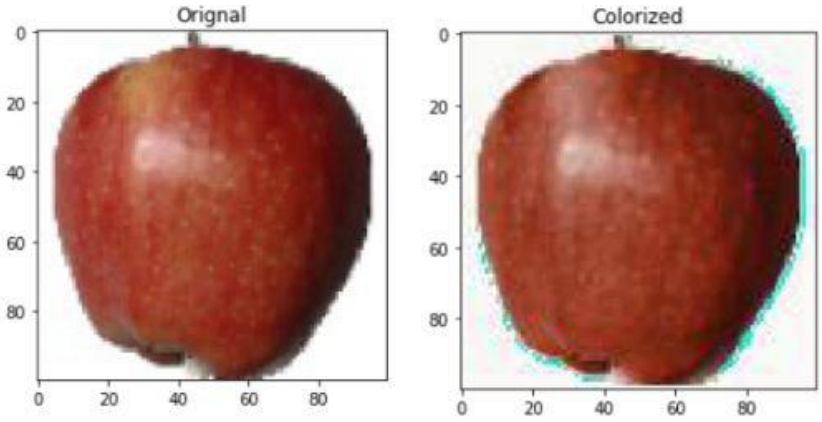
Loss value vs number of epochs

Initially, we randomly assign weights for our Artificial Neural Network. We then feed inputs (9 features) to it and forward propagate layer by layer to obtain the activated product of weights and the previous output (as the new input). We keep on forward propagating till we reach the output layer and obtain 3 distinct values as Red, Green, and Blue.

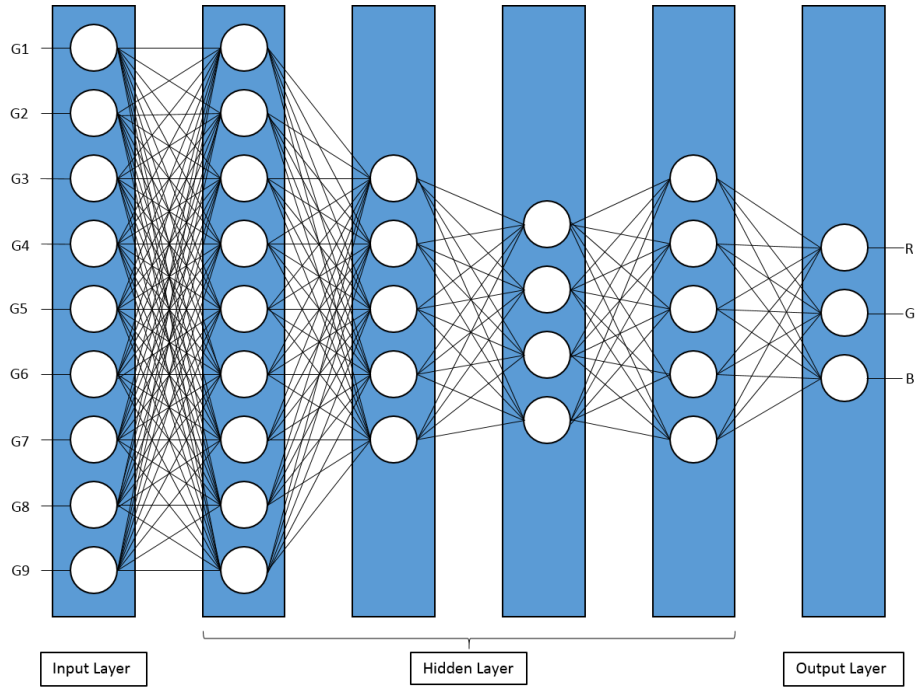
Once we've reached the output layer, we take the difference of the desired output values and the expected output values and back propagate the error and adjust weights accordingly. The way backpropagation works is we apply chain rule to differentiate the cost function with respect to the weights, layer by layer (starting from the output layer) and update weights with the derivative of the loss function. This is how the model learns over time.

The numerical/quantified error can be calculated by the RMS loss function whereas the perceptual error can vary from person to person, but to summarise it is how different the original image and the colorized image appear to be visually. When we tested an image of an apple, the results were very close to the original image visually. However, when we tested our results for an avocado the shape was intact but the color wasn't. As our model is solely trained on images of apples.

Test image for our model:



4. Training the Model:



We have used an Artificial Neural Network to train our model. Our model consists of an input layer, 4 hidden layers, and 1 output layer. Input layer consists of 9 nodes to which we input 9 grey values. The first hidden layer has 9 nodes and we used a linear activation function for its output. The second hidden layer has 5 nodes and activation function used for this layer is reLU. The third hidden layer has 4 nodes with a linear activation function. The fourth hidden layer has 5 nodes with a reLU activation function. We also use L1 regularization to the output of the layer to avoid overfitting. The output layer consists of 3 nodes which correspond to the (r, g, b) values respectively. With a lot of experimentation, trial and errors and hyper parameter tuning, we've come to the conclusion that the given architecture best fits our requirements.

We have used early stopping for convergence with penalty value of 100. That is if there is no improvement for 100 consecutive epochs the training will stop and the best model so far is saved. We've used the RMS loss function and determined convergence when the graph of loss function vs the number of epochs does not fluctuate substantially.

Linear activation function: For this function, the activation is proportional to the input to the node.

$$f(x) = x$$

reLU (Rectified Linear Unit) activation equation: $f(x) = max(0, x)$

L1 Regularization:
We use L1 regularization to reduce overfitting of the data. We used weight regularizes which are used to regularize the weights in the neural network.

$$Loss\ Function = Data\ Loss + Regularization\ Loss$$

$$Regularization\ Loss = \lambda \sum_{i=1}^{k} |W_i|$$

Here lambda is the regularization parameter.

RMSprop Optimizer:
The RMSprop optimizer hinders oscillations in the vertical direction. Hence, by increasing the learning rate the algorithm will take larger steps in the horizontal direction converging rapidly.
The following equations show how the parameters are updated using RMSprop optimizer:
Here, W is the weight parameter and b is the bias value.

$$S_{dW} = \beta S_{dW} + (1-\beta) dW^2$$

$$S_{db} = \beta S_{db} + (1-\beta) db^2$$

$$W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \epsilon}$$

$$b = b - \alpha \frac{db}{\sqrt{S_{db}} + \epsilon}$$

$$\epsilon = 10^{-8}$$

$$\alpha \leftarrow \text{Learning Rate}$$

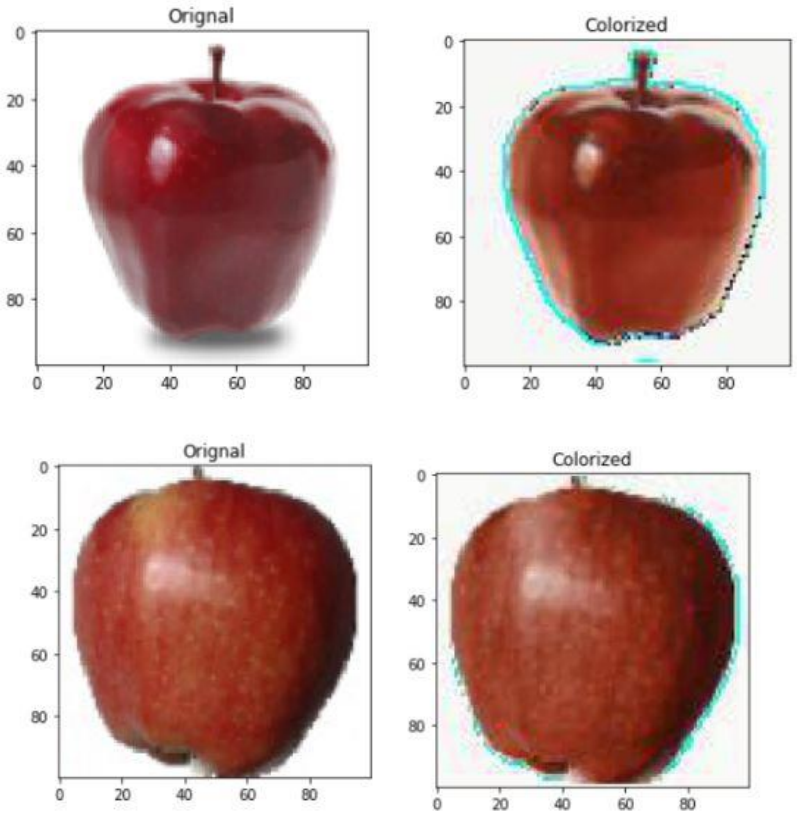
Mean Square Error:
Mean Square Error is the average of squares of error, where the error is the difference between the predicted output and actual output. This error is back propagated to update the weights of the network.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (f_i - y_i)^2$$

Where,
N is the number of data points
f_i is the value returned by the model
y_i is the actual value for data point *i*

5. **Assessing the Final Project:**

Our program is good at colourizing any red apple because it has been trained on images of red apples. We could use only 10 images in our dataset to train the model too because of the limited availability of resources and computing power. Our model does is extremely good at colourizing red apples as shown in the image below:



We validated our model by plotting the loss function vs the number of epochs

One major improvement that we could use trains the data on multiple (about a million) images, instead of just 10 training images, expanding the domain to colourize any grayscale image given to the model, resulting in the prediction of the original picture.
Another improvement can be the usage of a deeper Artificial Neural Network with more hidden layers, and more number of nodes, a better optimizer and fine tuning of hyper parameters.

Finally, one more improvement to increase the efficiency of the model, resulting in greater training speed could be the usage of the LAB colour space where we'll include the L/grayscale image we used for the input and a/b are the green-red and blue-yellow components, thus reducing the number of output nodes to 2, and heavily decreasing the computation time.

Our program does make an error while colourizing other images (e.g. some different fruit) and gives a reddish tint to that fruit, irrespective of whether the image was red or not. We tested it on an Avocado and the result was an expected "Red Avocado". The program's mistakes completely make sense because it won't be able to differentiate since it has been trained only on red apples.

Distribution:
Yash Nisar: Coding
Hiral Nagda: Coding
Adhish Shrivastava: Report