Assignment 4 – Colorization

Di Hao

This aim of this assignment is to build a model to colorize grayscale images, i.e. colorization, which is a computer-assisted process of adding color to a monochrome image or movie.

1. Representing the Process

Because of limited information in a single gray value, mapping from gray to (r, g, b) may not reliably reconstruct the true color of the pixel. Considering the neighbors of a pixel can also reveal some information of the center pixel, I chose to map the context of a pixel, i.e. group of pixels nearby, to a single (r, g, b) color vector. The intuition also comes from that pixels that are close and have similar grayscale value tend to have similar color vectors.

As an example, I could choose all neighboring pixels that are within a certain window size 2 to represent the context of the center pixel. This way, the input space of the center pixel is represented by 25 numerical numbers. Padding is used to take care of the edge cases where the center pixels are close to boarders. The output space is the (r, g, b) color vector.

The model is a neural network model built from scratch. To ensure its correctness, I computed the numerical gradients with respect weights and bias and compared them with the gradients obtained from back propagation. The below ratio of the L^2 norm of the differences to that of the sum of both gradients was computed. With extensive testing, the value is consistently smaller than $1e - 11$, hence I believe the back propagation is implemented correctly.

$$r_{L^2} = \frac{||\boldsymbol{G}^{numerical} - \boldsymbol{G}^{back-propagation}||}{||\boldsymbol{G}^{numerical} + \boldsymbol{G}^{back-propagation}||}$$

To ensure the correctness of the optimization algorithm used, I tested the neural network against the XOR problem with architecture [2, 2, 1], i.e. input dimension is 2, there is only one hidden layer with 2 neurons and 1 output neuron. The program converges in several thousand of iterations and generating testing error smaller than $1e - 6$ (the training set and testing test are the same in this toy test).

The network is implemented in Python without usage of GPU.

2. Data

The network is trained on the CIFAR-100 dataset[1]. Because I do not have too much computation power at my disposal, I only used one of trees from the dataset, which contains 600 32x32 color images in total. 100 of the images were randomly chosen as testing set. Another 100 were randomly chosen as the validation set. The remaining 400 images were the training set.

Pre-processing is necessary in this model. The images were first normalized to have (*r*, *g*, *b*) values lie within [0, 1]. To create the input data, the images are converted to grayscale. Assuming the window size is 2, a length of 25 array is created for each pixel in each image. This gives me 400 * 32 * 32 = 409, 600 data points in total. The corresponding output labels were directly obtained from the (*r*, *g*, *b*) values from the original color image.

3. Evaluating the Model

To evaluate the model, I defined the cost function as the mean square error as below, where $y^{(i,c)}$ is the predicted value of *i-th* pixel and channel *c*, *m* is the number of batch size, λ is the regularization rate and $w^{i,j}$ represents all the weights.

$$J = \frac{1}{2m} \sum_{i=1\ldots m} \sum_{c} (y^{(i,c)} - t^{(i,c)})^2 + \frac{\lambda}{2m} \sum_{i,j} (w^{i,j})^2$$

The cost function can be easily optimized utilizing back propagation. This cost function describes the different between predicted value and truth value on a pixel level for each color channel, hence is used to assess the error the model. This is the numerical / quantified error.

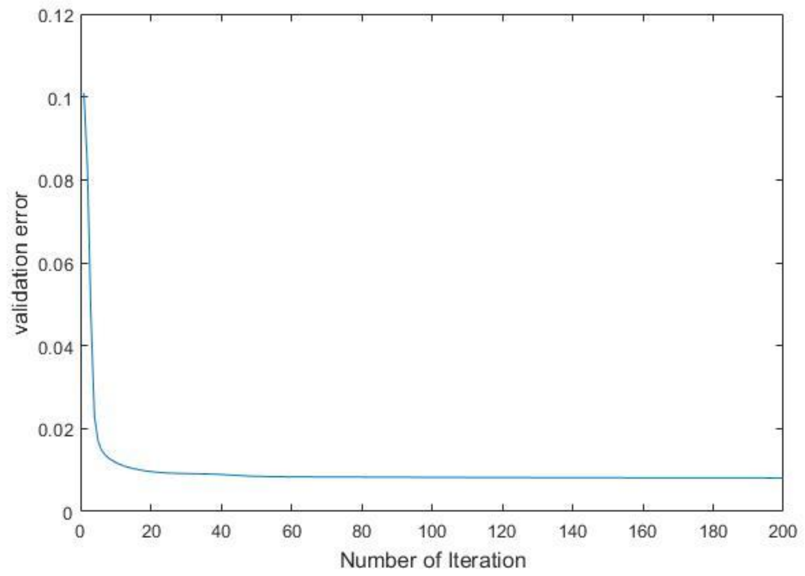
"Colorization Turing test" may be used to assess the perceptual error. However, this is not done because of limited time and man power.

4. Training the Model

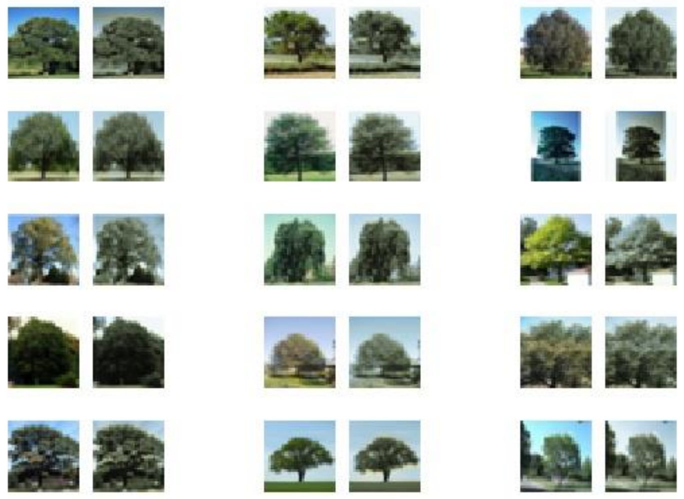
I applied stochastic gradient descent and Adam algorithm, which may be regarded as the best practice in training neural networks, to update the weights and bias of the neural network. The initial parameters of Adam method is chosen as recommend by the original authors[2], i.e. $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 1e - 8$. A regularization rate 0.0001 was chosen as λ, which is used to avoid overfitting. The training epochs are iterated until the validation set stops improving in accuracy for 10 epochs and the optimization process is carried out 6 times using an order of magnitude smaller α (learning rate) each time. After this, the algorithm is regarded to have reached convergence.

5. Assessing the Final Project

To assess the performance of the model, I first plotted the validation errors during the training of the model as shown below. The validation error decrease very fast at the beginning and then slowly improves after. The model converges after 200 (determined by the approach describe in section 4 above).



The result model was saved and tested against the testing dataset. The results are shown below. For each pair, the image on the right side is the original color image and the image on the left is the predicted image given the grayscale of the original color image. One thing to notice is that the model is good at differentiate between tree, sky, ground, and other buildings. Although the exact color may be different from the original tree, but the overall shape / segment is correct.



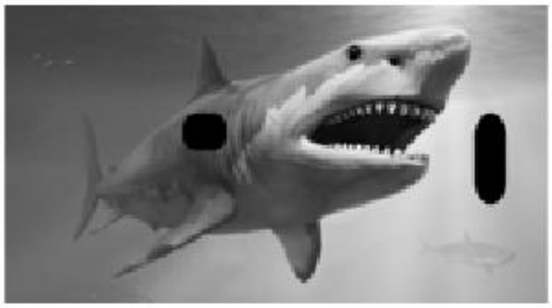
Let’s take the image on the third row and third column for an example. The predicted images have tree with pale color, which is different from the original green. This “mistake” is most likely because the image have some building with white color nearby and the neural net model is tilted towards white. Those mistakes do make sense.

One disadvantage is that the model is trained only on images of trees. Give an image of ship or dog, it can’t predict the correct color accurately. This is also one of the places I would want to improve if I have more time and computing power.
Given more time, I would also try to build a GPU version of this model and train on the whole CIFAR-100 dataset. Another direction I would try is to apply better normalization mechanism, such as batch normalization or drop out, and deeper / larger network with CNN flavor.

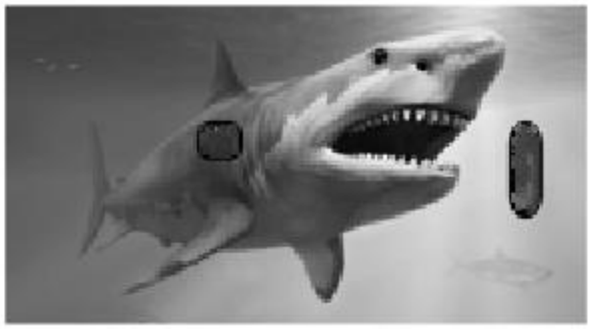
Bonus

This is similar to image inpainting problem. I decided to apply a more “traditional” method, i.e. a non-parametric method for texture synthesis[3]. Simply speaking, this approach by Efros and Leung grows a new image outward from an initial seed, one pixel at a time. The conditional distribution of a pixel given all its neighbors synthesized so far is estimated by querying the sample image and finding all similar neighborhoods. The implementation is in MATLAB by simply following their pseudo-code here[4]. This method does not need training data per se. I created an image with some portion removed and the result from this model is shown below.

Before:



After (the window size is chosen as 15):



This method is slow and not perfect. You could see the flaw in the above image. Maybe again neural network is the way to go here.

Reference:

1. https://www.cs.toronto.edu/~kriz/cifar.html
2. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” CoRR, vol. abs/1412.6980, 2014.
3. Efros, Alexei, Leung, Thomas K, et al. Texture synthesis by non-parametric sampling. In Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, volume 2, pp. 1033–1038. IEEE, 1999.
4. http://graphics.cs.cmu.edu/people/efros/research/NPS/alg.html