Assignment 4 - Report

CPSC 599, Deep Learning For Vision

Deliar Mohammadi, 30072994
John Zheng,30125258
Xinzhou Li, 30066080

Model Architecture

In this project, we borrowed some ideas from the model eccv16 introduced by the article written by Rached. Zhao. Since we used the CIELAB color scale which has two color channels A and B each of them has a range from -128 to 127 and one lightness channel L which is from 0 to 100, since the lightness is directly given by the original input.

Therefore, our model only needs to learn how to predict the proper paired A,B value. There are actually (256*256) / 2 number of combinations of colors. Since we decide to resize all the input data to size 256 * 256, our model first take the convolution on our input images with the kernel size 3*3, down to 32*32*512, every time it has been applied convolution layer, it will be also applied a batchnorm transformation to increase the computation. After convolution, dilation convolution will be applied to the output of the convolution, which will offer a wider field of view, so it can get more info without increasing the size of the kernel. Finally using ConTransposed to map our data to 64*64*313, then use upsample by scale of 4 to get our final result which is 256*256*2 tensors contain predicted A, B values.

The CUnet is a model by Na Wang, Guo-Dong Chen, and Ying Tian. The CUnet is based on the Unet model, which utilizes skip connections over a series of convolutions/downsampling and deconvolutions/upsampling. The skip connections (which copy the output of a downsampling step and concatenate it to an upsampling input) allow low-level features to be accessible during the deconvolution/upsampling steps. The basic Unet uses 4 downsamples and 4 corresponding upsamples, with convolutions in between.

The CUnet replaces the lowest downsample/upsample steps with extended convolution or dilated convolution. This allows the model to use features from other parts of the image without losing spatial resolution as a traditional max pool + convolution does. Using a dilated 3x3 convolution instead of a 5x5 convolution also saves processing time and may prevent overfitting.

The input should be a 256x256 image (or any multiple of 8) with channel, the lightness, and the output will be a 256*256*2 tensors contain predicted A, B values, the same as the eccv16 network.

Model Training

To train our models, we have decided to use the Adam optimizer with the following parameters:

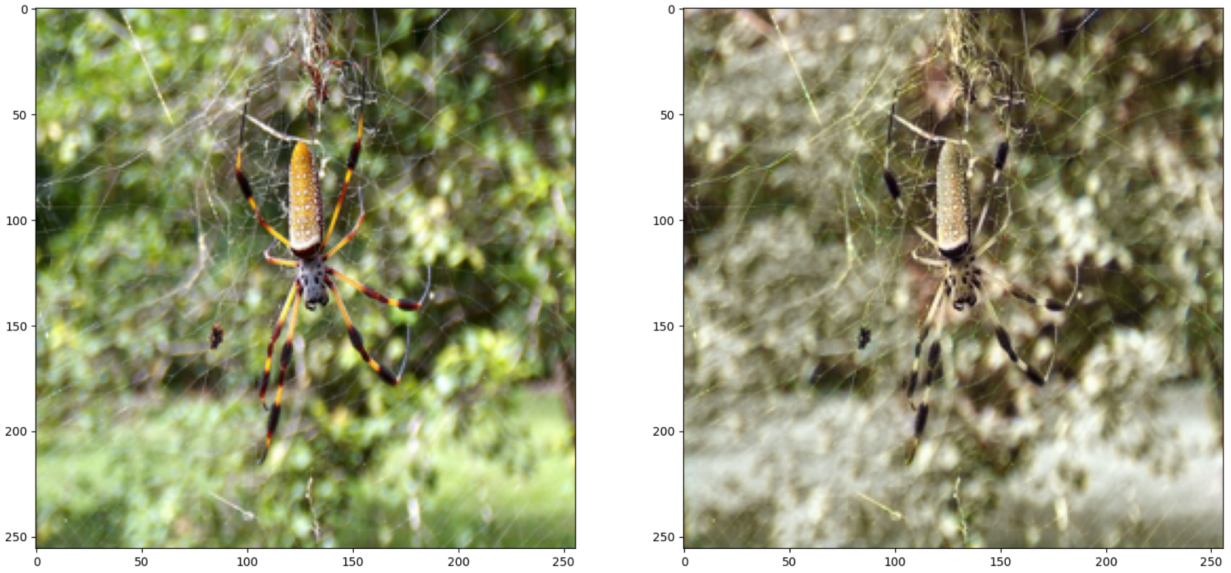
- Learning rate = 0.001 -> 0.0002 -> 0.0001
- Weight_decay = 0.0001 -> 0.00001 (this is L2 regularization)

The reason we chose Adam was because it has worked for us in previous projects and seems to work well generally in vision related applications. During our training if we notice that the accuracy is plateauing we will reduce the learning rate and potentially try a different optimizer altogether. In addition to the optimizer we will train the model for several hundred epochs, we have made a function that saves the model intermittently and loads in the weights of the previous epoch so that we can start where we left off and not lose any training progress. Our batch size is currently set at 32 due to hardware constraints, if we find that the vram can handle

more images we may increase this size to 48 or 64. In order to find the best model, we have implemented our own “stopping mechanism“ as discussed above. It saves the model in a file with the corresponding loss in the file name. This way we can find which model had the smallest loss by simply looking at the saved weights.

Model Evaluation

Since our selection is to colorize the grayscale image, there is no accuracy or F1 score parameter to evaluate our model, but we think using the loss value to determine how far our model deviates from the original value is a proper evaluation of the color and image quality, and then we also will introduce human visual assessment to evaluate whether the color and quality of the image meet the human visual perception. If an image is able to convince a human that it is real and not colored by a machine learning model, then we would consider that a success. Take for instance the Figure below, on the left is the original image from the dataset, and on the right is our colored version of the image. From our perspective, both of these are plausible and hence the model made a good prediction on this image.



Since our model wants to achieve reasonable coloring for all categories of images, we will first implement it to give reasonable and accurate coloring for a particular category of images. Therefore, we do not have any special requirements for the test dataset and do not need to consider the detailed differences between images, so our test dataset will be directly selected from a random part of the dataset.

After the model finished training, there is a printCompersion function that will take 10 pictures at once eval by the model, get the output AB color data and contacts with input scale color data to rebuild the proper image with LAB color scale, then turn it to RGB color scale, and finally print out the original images and colorized images side by side for human version evaluation.

Model Fine-Tuning

In order to improve the performance of the model we implemented a couple of fine tuning techniques. One of these being L2 regularization. In Pytorch this can be achieved fairly easily by setting the *weight_decay* argument in our optimizer to a small value. The figure below demonstrates the additional term in our loss:

$$Cost\ Function = Loss + L2\ Weight\ Penalty$$

$$= \underbrace{\sum_{i=1}^{M}(y_i - \sum_{j=1}^{N} x_{ij}w_j)^2}_{Squared\ Error} + \underbrace{\lambda \sum_{j=1}^{N} w_j^2}_{L2\ Regularization\ Term}$$

In the equation above, the *weight_decay* argument is represented by lambda. This additional term prevents our model from learning hard set patterns from the training set and makes it fit better in the general case.

In addition to using L2 regularization, we also manually reduce the learning rate when the loss of our model plateaus, this way we are able to squeeze more performance out of the model. To begin with we set the learning rate to 0.01, and we reduce this to 1/10th when needed. This helped us to reduce the loss, and although its not a critical difference, it still helped push our results in the right direction.