**CS 520: Assignment 4 – Colorization**

Group members:

|  |  |  |
| --- | --- | --- |
| Name | netID | Work |
| Yan GU | yg369 | Analysis/Data Process/Total collection |
| Yunfan LI | yl1269 | Implementation and training of the CNN |
| Xueyu WU | xw318 | Data collecting/Write up |
| Junlin LU | jl2364 | Implementation of full connected colorization using dataset Cifar10 |

1. **Colorization problem**

For this project, our goal is to produce colored images by given grayscale pictures. The main challenge is that we do not have enough information on a pixel to map from a single grayscale value gray to corresponding color. Which means that a single grayscale image may correspond to many plausible colored pictures.

Neural networks have shown remarkable success in computer vision due to their ability to gain information from whole pictures. So we will use Convolutional Neural Network and aim to infer a full-colored image with 3 values per pixel RGB or LAB from a grayscale image.

1. **Mapping process**
2. **Mapping spaces**

In general, colored images have RGB format where R represents the level of red, G of green and B blue respectively with value between 0 and 255. A classical relation of color and gray pixel is the formula

where the value is also between 0 and 255 which representing the corresponding shade of gray. So we are going to map 1 gray value to 3 values of red, green and blue.

Another format of colored images is LAB color space where L is lightness from 0 to 100, A represents the green-red component and B the blue-yellow in the range of -128 to +127. This color space contains more information than RGB and is easier to handle when colorization because instead of predicting 3 values R G B, we only need to colorize 2 channels A and B for a grayscale image itself is the lightness of the corresponding colored picture.

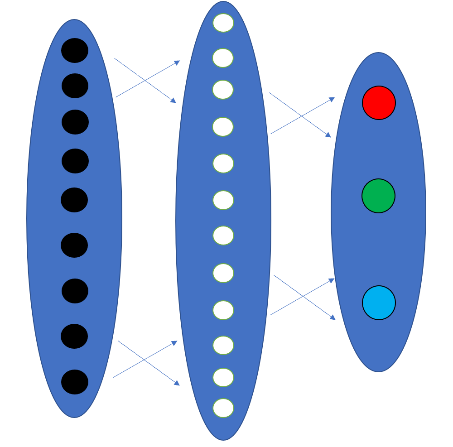
1. **Model structure**

We build 2 models respectively. The first one is a Simple neural network whose role is to test the ability of neural network and make us learn more about the structures and working principles of neural network. The other one is a fully completed auto-encoder structure of convolutional neural network, which is right down image processing and computer vision.

**Simple Neural Network**

One way to approach colorization is to gain additional information from the surrounding of the current pixel. So, we used a 3 x 3 window to scan on the matrix.

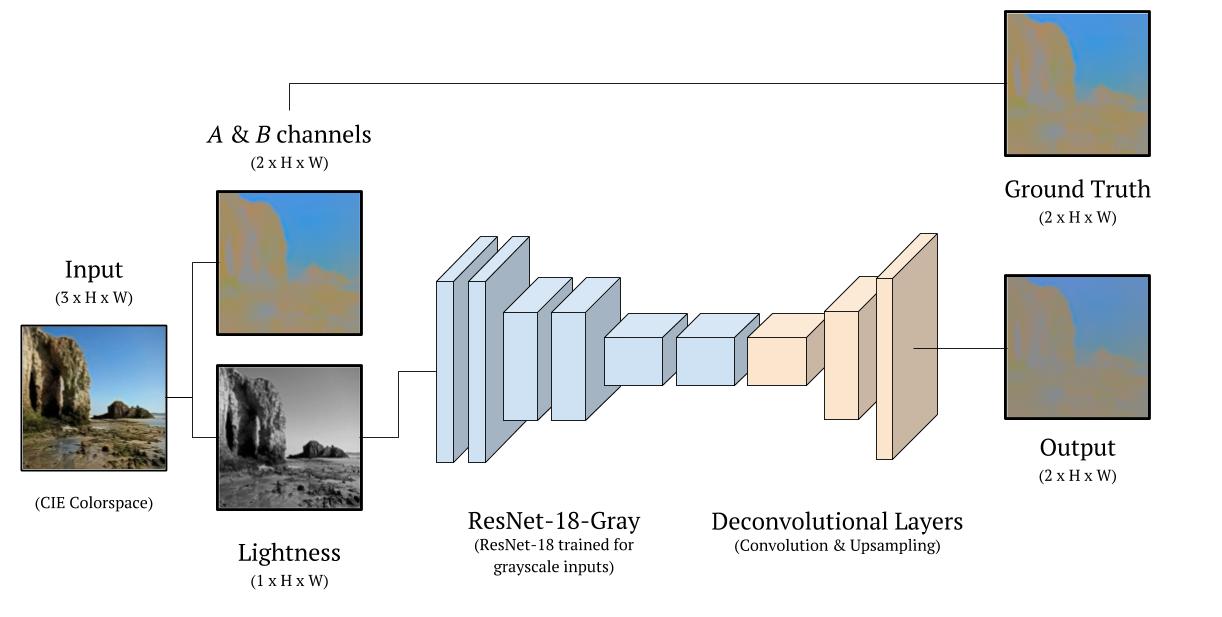
We construct a neural network with an input layer with 9 nodes (possible 4 in the corner, 6 in the border), an output layer with 3 nodes (representing red, green and blue of the center pixel) and one hidden layer with 12 nodes.



*Figure 2-1 nn model*

**Convolutional Neural Network**

In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery. So we are using it to colorize grayscale images. Firstly we apply a number of convolutional layers to extract features from grayscale images, then we build deconvolutional layers for unscaling of our features to reconstruct colored images. More specifically, we use ResNet-18 as the encoder of our CNN model, which is an image classification network with 18 layers and residual connections. As for the decoder, we use a set of convolution and upsampling operations to implement deconvolution.



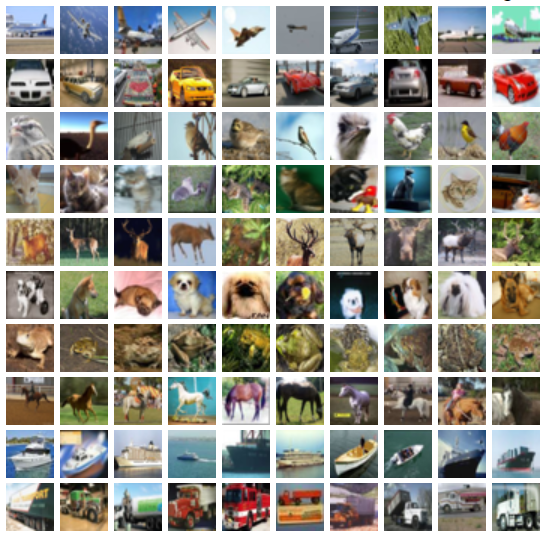
*Figure 2-2 cnn model*

1. **Tools**

For the small neural network, we build and train our model by ourselves with Python. On the other hand, for better training results and the faster training speed, we decide to build and train convolutional neural network with PyTorch. We also use other helpful toolkits like scikit-learn for converting between RGB and LAB colorspces.

1. **Data**
2. **Datasets**

The key to getting better at neural network is practice. Datasets are an integral part of the field of machine learning. There are a lot of free image datasets for computer vision. At first, we consider cifer-10 datasets because of the low memory and compute speed of our laptops. However, there are some disadvantages of them. Firstly they have too many classes of pictures which is difficult for our model to colorize. Secondly, the low revolution results in dramatic changes between neighboring pixels, so it is not good for training colorization.



*Figure 3-1 cifer-10 datasets*

So we use the subset of the MIT Places dataset, which consist of places, landscapes, and buildings with size 256 \* 256 in our CNN model. During the training, we used a subset of 5,000 images as the training set, and another subset of 1,000 images as the validation set.

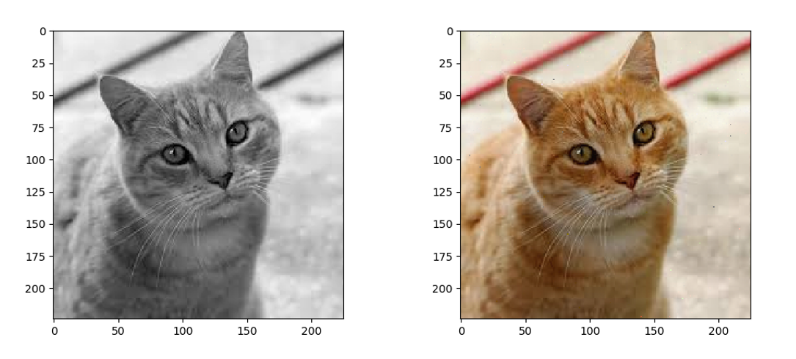
  

*Figure 3-2 MIT Places datasets*

1. **Pre-processing**

For our CNN model, before we train, we define helper functions for tracking the training loss and converting images back to RGB. But it does not need to convert colored images since we have modified the first layer of network in order to accept grayscale images. In the neural network, however, it is necessary to convert RGB colors to gray to get the input. We use a classical conversion formula as we discussed before

to get grayscale images as our input.



*Figure 3-2 conversion*

1. **Evaluating the Model**
2. **Numerical error**

A general way to compute error in vector space, here is color space specifically, is the distance square between each corresponding pixel. So we can quantify our algorithm’s performance by calculating the average error per pixel of our test or validation data set.

where i is the index of each pixel in an image.

1. **Perceptual error**

However, the numerical error we discussed before has problems. One thing it can not achieve is that colorize an image based on interpretation. If there is an grayscale image with apples in it, it makes sense to colorize it green or red for people to observe. But for the model, if the original apples in that image is red, then colorizing green is a big mistake. Another thing is that the model may prefer to choose less bright colors rather than others in order to reduce the error. All in all, generally speaking, the less the loss is, the more correct the output becomes. But sometimes the output also looks like a real image even if its error is big based on distance square.

1. **Training the Model**
2. **Loss function**

Since it is a regression algorithm, we use the mean squared error as our loss function. The basic idea is that we want to minimize the squared distance between true color values and our prediction, so that the output is as close as possible to the original colored image.

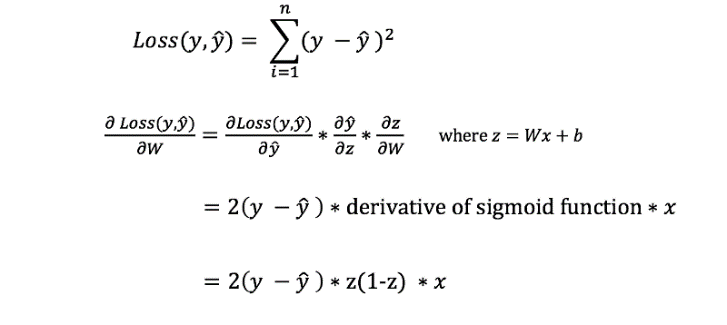
Specifically, in the feed forward function of our simple one hidden layer neural network, we use Sigmoid function as the activation function. For each pixel, we take this pixel and its 8-surrounding pixel, shape them into a 1 x 9 matrix; According to the function, we use a 9 x 12 matrix as weight f, we multiple these two matrix and get a 1 x 12 matrix, process it with the sigmoid function, we get the hidden layer with 12 nodes; then we use a 12 x 3 matrix as weight2, we multiple them and process it with the sigmoid function, the 1 x 3 matrix that we get represents the RGB in this step. The function of the output with regard to the input can be written as follows:

https://cdn-images-1.medium.com/max/800/1*E1_l8PGamc2xTNS87XGNcA.png

As for the CNN model, we use ReLU as the activation function. Since the gradient can propagate and cause gradient explosion in deep networks, the loss function we use in single-layer network does not apply here. Instead, we use Mean Square Error, which is actually quite similar to the previous one:

1. **Backpropagation**

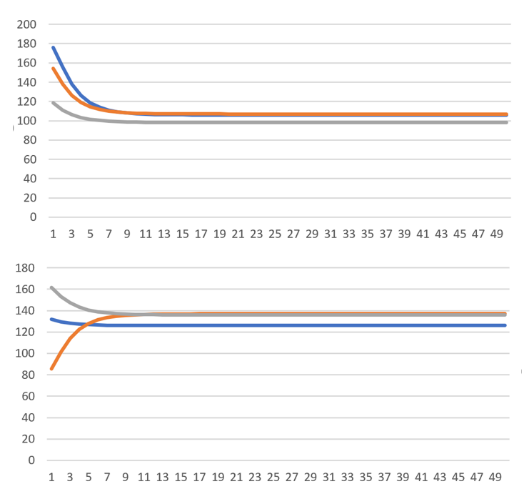
As for the back propagation, we use a simple sum of squares root function as our loss function. Therefore, the chain rule that we use to get the derivative of the loss function can be represented as



For our CNN model, we will optimize our loss function (criterion) with the Adam optimizer. And in training process, we run model and backpropagate using

1. **Convergence**

Observe the loss function, we can see that the loss monotonically decreasing towards a minimum. In respect to the RGB, the reduction of each channel is monotonic. This is consistent with the gradient decent algorithm.



*Figure 5-1 convergence of the single-layer network*

1. **Overfitting**

A general way to avoid overfitting is to add  to Loss function because overfitting happens if the number in parameter matrix W is too big or small.

We are going to minimize the new loss function to deal with overfitting in the future, But right now we want to see what happens and what the influence is if we do not add penalty to our loss function, so we still use the original one as our loss function.

1. **Assessing the Final Project**
2. **Test data and error**

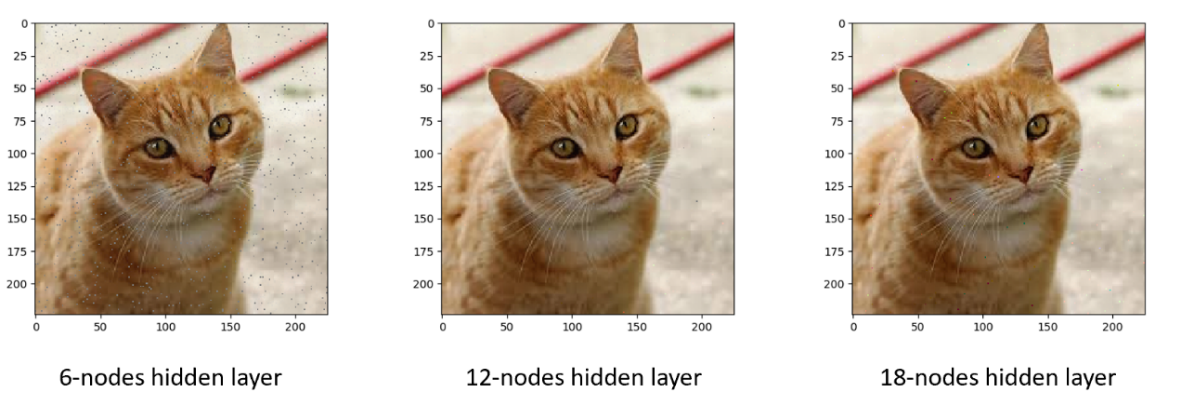
There are 1000 images to see how our trained CNN model's performance on this validation set.

*Figure 6-1 validation dataset (100 epoch)*

For neural network, one thing worth mentioning is that the number of nodes in the hidden layer affects certain number of pixels’ RGB. With a 6-nodes hidden layer, some of the pixels stay gray no matter go through how many iterations, the gradient decent doesn’t work if three channels are all around 127; However, with an 18-nodes hidden layer, some of the pixels are overfitting and restored as the original color.



*Figure 6-2 influence of hidden layers*

1. **Disadvantages**

Because the loss function is too straightforward and does not consider the complexity of colorization, we will get more desaturated colors in our output rather than bright colors which may result in harsh penalty if it is the wrong color our model chooses.

1. **Improvements**

We do not apply any algorithm to prevent overfitting in our model for some practical issues such as running time or convergence. So we can add some strategies as we discussed before to avoid overfitting and compare the difference.

Another thing is to improve the brightness of our output images. This is a direct result of our loss function. So maybe we can add some coefficients to pixels based on some deduction.