Assignment 4:Colorization

Changlin Jiang(CJ360),Ruolin Qu(rq40),Shengdong Liu(sl1563)

1. Introduction

We tried an ANN-based regression approach to colorize grayscale pictures. The input of the network is 25*1, which stands for grayscale value of 5*5 area surrounding one pixel. The output is the a and b value of the pixel (a and b come from a different color metric Lab different from rgb, they will be discussed later) We choose python as the programming language with python interpreter 3.7. The ANN is implemented by ourselves only using Numpy. To read in pixel data from files and show generated pictures, we use 3rd party libraries Image and matplotlib. Beyond that, the package opencv-python helps us resizing and transferring to grayscale image and L*a*b* color space.

The Dataset was collected from Google Image Search include 22 images of the Beaches. Images are resized to 64x64 pixels. You can see the dataset in the folder “Dataset”.

2. Progress Representing

Experiment Setting

We implemented a vanilla NN with numpy. The structure of the model is:

input: (25,1)
fc1: (16,25)
fc2: (2,16)

The input of the model is a 25*1 vector containing the grayscale values of 5*5 area of a pixel. The output is the regression result of ab channel of the pixel. The algorithm is iteratively pass forward on every pixel in the image and do back propagation and SGD on every pixel.

Mathmetics Base of ANN - Backpropagation

To explain the backpropagation mechanism in fixing error for each node. We can illustrate by appreciate this following example:

Assuming we have an ANN with two hidden layers considering we are trying to fix the weight w5. Basically, the first thing we concern is how much influence does this weight have for the output.
From the graph above, the total influence for w5 is:

$$\frac{\partial Etotal}{\partial w5} = \frac{\partial Etotal}{\partial outo1} \times \frac{\partial outo1}{\partial neto1} \times \frac{\partial neto1}{\partial w5}$$

Now since we can calculate each part, for both $\frac{\partial Etotal}{\partial outo1}$, $\frac{\partial outo1}{\partial neto1}$, $\frac{\partial neto1}{\partial w5}$
In the end, we multiply these three items and get $\frac{\partial Etotal}{\partial w5}$.
Another way we calculate $\frac{\partial Etotal}{\partial w5}$ is:

$$\delta o1 = -(target.o1 - out.o1) * out.o1(1 - out.o1) * out.h1$$

Hence, if we use symbol $\delta o1$ stand for the final loss, we can get:

$$\delta o1 = -(target.o1 - out.o1) * out.o1(1 - out.o1)$$

$$\delta o1 = \frac{\partial Etotal}{\partial net.o1} = \frac{\partial Etotal}{\partial out.o1} \times \frac{\partial out.o1}{\partial net.o1} \times \frac{\partial net.o1}{\partial w5}$$

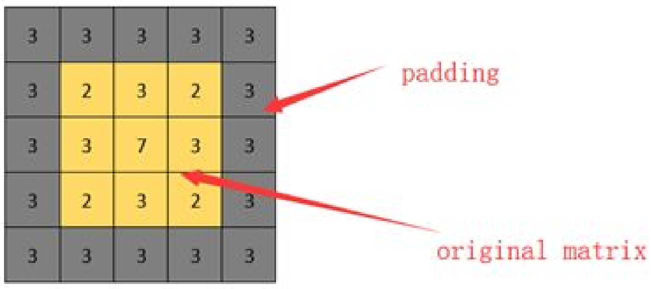
To update the value w5, we can:
w5(new)=w5(old)- $\eta \times \frac{\partial Etotal}{\partial w5}$
η is the learning rate.
Generally, we iteratively update the weight value .

CIE Lab representation of color.

We transfer the color space of original image from RGB to Lab.to where L*,a*,b* stand for L*:Lightness, a*:Red/Green Value, b*:Blue/Yellow Value. L* value is provided by the Gray image. a* and b* will be predicted by the neural network.
Compared with the original RGB color space, we can reduce the output of neural network with LAB color space and fully utilize the information of Gray image.

Pre-processing by image padding

Theoretically, our work is very close to convolution operator but replacing the filter kernel (actually a one-layer vanilla neural network) with a more complicated multi-layer neural network.But for 2D convolution, the pixels on the edges and corners are invalid because some part of the filter area will get out of the boundary. To validate those pixels, average padding is implemented to extend the image boundary by 2 pixels (filled with average value of the whole image), as shown in the figure:



3. Data

Our dataset contains 53 rgb-scale images of beaches, forests and sunrise scenarios downloaded from the Google Image Search and resized all images into (64,64). All the scenarios contains a limited amount of color.

We also convert all images into CIE Lab color space with opencv library. [1]

4. Evaluating the Model

The loss criterion we used is l2-loss:

$$\sqrt{\sum(Original\ A\&B\ value\ Image - A\&B\ value\ in\ ConvImage)^2}$$

This function is used to evaluate the difference between predicted image and the original image. The goal is to minimize the distance between the prediction and the original image.

According to the way we represent the labels, the element in label array for A and B value will represent the loss.

5. Training

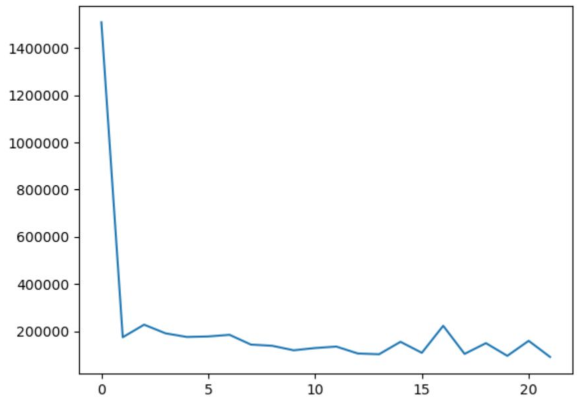
The structure of our neural network is 25 neurons in the input layer, 16 neurons in the hidden layer, and 2 neurons in the output layer. We choose to use the first picture as the test picture and the rest 21 images as the test data. For testing, our program can be easily modified to use another similar dataset as training dataset instead.

We first convert the image in training set to grayscale as the training input. For the training process, we first use cv2.cvtColor(img,cv2.COLOR_RGB2Lab) to calculate the A,B. Those two values will be standard for evaluating the loss in each training round. Then we start training our network, we use a window with size 5*5 and slide it along the grayscale image. The 25 grayscale values within the window, labeled with the A and B values of the central pixel, are flatten into a 25 array and fed in to network. We use hyperparameters such as the number of epochs and learning rate to control the training process and to find the converge. In each training round, we will get two A,B value from the output layer. Based on the backpropagation process we mentioned in the representing section, the algorithm will minimize the loss for A,B by moving the gradient toward the direction which itself drop rapidly. Then we calculate the loss for each round as part of the result.

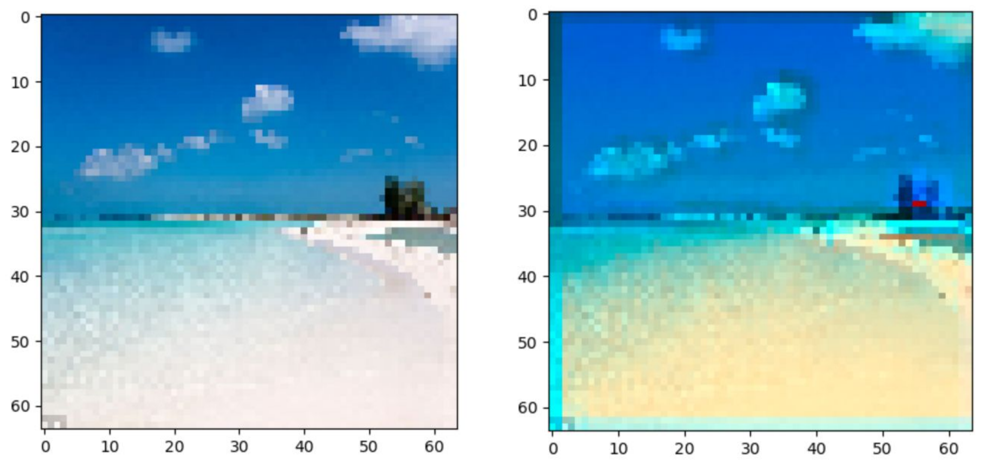
6. Assessing

The first trial - beach only

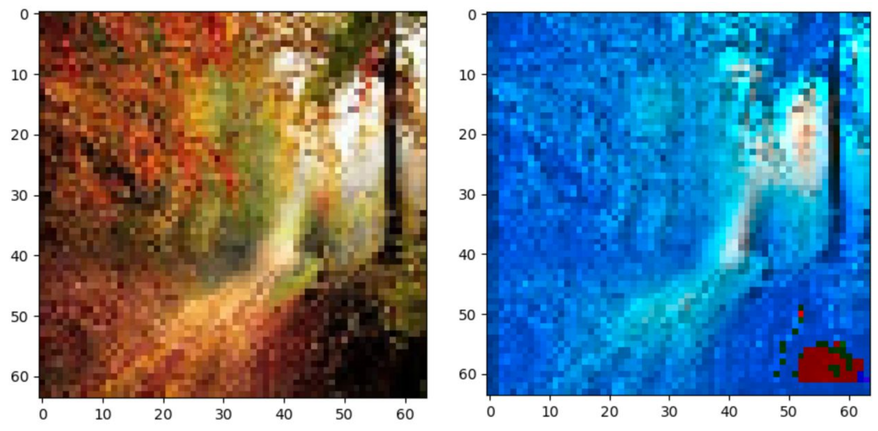
Firstly we trained the model only with 22 images of beach for 1 epoch, and here are the results with these settings.the loss value is reducing fast. the following figure shows the reduction.



the following images are the comparison between the original image and the predicted image. the neural network can deal with the beach image. but we can also find some mistakes in detail. For example, clouds in the top-right is colorized with yellow, which might be considered as sand. and the rocks are colorized with some red. Perceptively, I think this result is decent because the model successfully recovered the color of sky, beach and clouds.



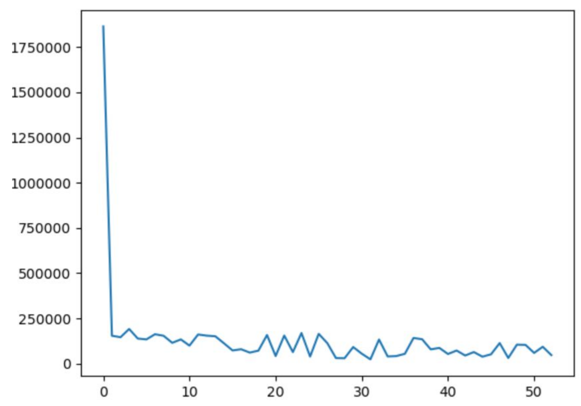
However, based on the same model trained with the dataset of beach image, we tested an image of colorful autumn forest and the result is bad.



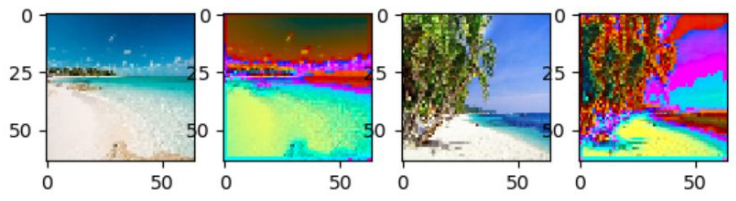
The reason is easy to be found. In the training set, the images are mainly filled with several textures: sky (blue), beach (golden), cloud (white). But forest image with woods and leaves is a different story. So the textures coming up in forest image are never learnt by the network so it did bad job.

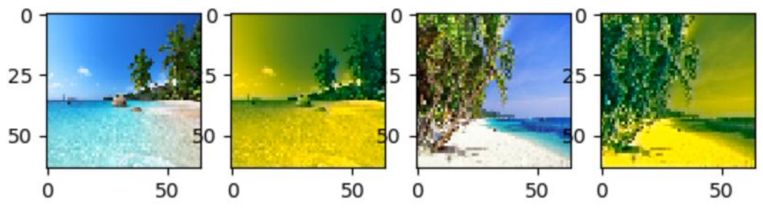
Second Trial: beach, sunrise and forest all included

In our second trial, the dataset includes 22 beach images, 17 forest images and 14 sunrise images, which is 53 images in total. Then we randomly shuffle the order of images and trained the network for 1 epoch with learning rate 0.01..The following figure shows the reduction of loss with these settings.



In this trial, we got very unstable results in different rounds of training and all the results are very bad:





In this round, the result largely depends on the initial state of the network and the random order of dataset.
Here is a simple example to illustrate how the order of dataset influence the result: Assume that after shuffling, the dataset fed into model has 4 sunrise images (with purple & red sky) in a row. During training on those images, the network is likely to converge on sunrise images and tends to fill a smooth, continuous sky-like texture with purple and red color. The loss on sunrise images is getting smaller and smaller during training. Then assume a beach image is coming up whose sky is blue and white. The network will achieve a very high loss after forward pass and change a little bit during SGD. But this change is too small compared with the previous 4 sunrise images in a row so the network still tends to fill the sky texture with red and purple instead of blue and white, or pick a median value between red and blue to achieve medium performance on both sunrise images and beach images.
As a result, the first row of result might had many sunrise images during training so the sky is filled with purple. The second row might be dominated by forest images so the sky was filled with green and yellow.
Furthermore, this result might be caused by structure failure of the network. In our network structure, only 5*5 area around the pixel is inputted into the network and the network knows nothing out of that. So the receptive field of our model is 5*5 so some far-off features for that pixel might be ignored. For example, for one sky pixel in sunrise image, there might be a very large sun far away. If the sun can be covered in

the receptive field of the pixel, the model will be able to know this is a sky pixel in a "sunrise" image which should be colored red and purple. If the sun cannot be covered, the model will be confused because it doesn't know whether the sky pixel is a pixel in beach image (should be colored blue) or a pixel in sunrise image (should be colored red). So to improve the result, the receptive should be extended.
One possible approach is to implement CNN. In CNN, two 5*5 conv2D layer with stride of 4 can achieve 25*25 receptive field (5*5 grid in which every pixel includes information of 5*5 grid of previous layer) so the features covered by the model will be greatly extended. Then the model is likely to achieve better results on this work.

[1] Opencv: Color Conversions,
https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html