project4: Colorization

Bingyu Xin

December 2020

1 Dataset

we select a picture on the website[1], and resize it to 256x256 as shown in Fig. 1. As required by the assignment, we use the left half of the image as training data and the right half of the image as testing data.

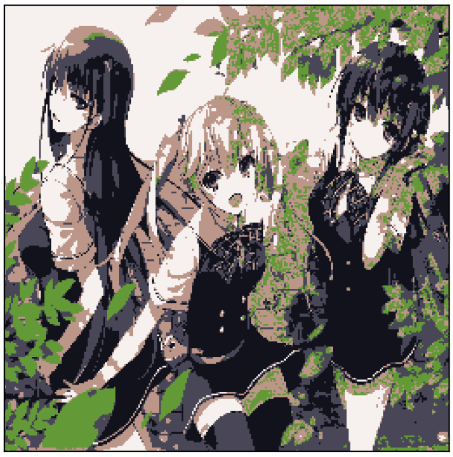
Figure 1: The 256x256 RGB image and its corresponding gray scale image used for training and testing the colorization model

2 The basic Coloring Agent

Following the pipeline from the assignment note, we run k-means clustering on the colors in the training data to find the best 5 representative colors. We color the test data using these 5 colors. To be more specific, for each 3x3 patch in the testing data, we find the most similar 3x3 patch in training data, and use the representative color of the middle pixel in the similar patch to colorize the middle pixel in the testing patch. The result shows in Fig. 2

[1] https://w.wallhaven.cc/full/vm/wallhaven-vmm37p.jpg

(a) 5 representative color



(b) colorized image from gray scale image

Figure 2: Result by the Basic Agent. for (b), the left half of the image is generated by k-means method, the right half is generated by patch method

3 Improved Agent

Since we only have one image for training, we decide to use a simple 4-layer FCN to learn the colorization mapping and avoid overfitting. We regard this as a regression problem, which means we will use a MSE loss to calculate the pixel-wise difference between the result and the ground truth. However, we don't use FCN to generate RGB image directly, we find that it will be easier to learn the mapping from gray scale image to LAB color space. In LAB color space, the L channel equals to the gray scale, so what we need to do is using L channel to predict other two channels in LAB space, rather than to predict three channels in RGB space. The input data will be scaled to [0,1].

The structure of our FCN is shown in Fig. 3. We build all the layer from scratch using only numpy, including forward function, backward function and update function.

Unfortunately, our implementation is computationally terrible and can not converge to sensible results. So, we use pytorch to build the FCN, we set the learning rate to be 0.01, and use only one image to train FCN for 4000 epochs to generate the result, as shown in Fig. 4

4 Evaluation

We can see that the color in improved agent is more smoother than the basic agent, the whole image quality looks more similar to the original image. But improved agent is not perfect, it also exists artifacts and wrong colorization. Besides, we use PSNR and SSIM to evaluate the quality of the result, shown as

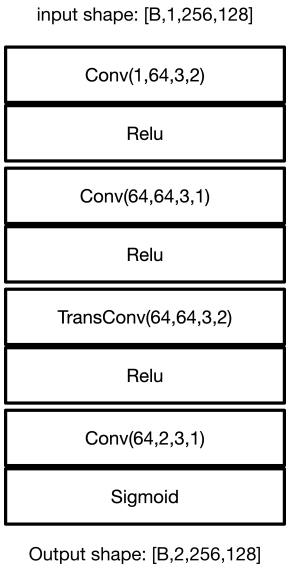


Figure 3: The FCN structure. Conv(1,64,3,2) means Conv(input channel, output channel, kernel size, stride)

Table. 1

To further improve the result, we can extract patches from the training image as training input and add more image augmentation methods to enlarge the dataset, we can also optimize the convolution operation code for higher efficiency.

Table 1: Quantitative analysis of two Agents

method	PSNR	SSIM
Basic Agent	17.06	0.65
Improved Agent	18.99	0.83

Figure 4: The FCN colorization result. The left half is the original image, the right half is the generated image