Introduction to Machine Learning

Final Project Report

Automatic Colorization System

Team 22

Yuanting Song N10731496

Dapeng Liu N12261428

May 14th,2018

1. Introduction

There are many black and white photographs in history. Due to the limitations of the technology at that time, most of them are plain black and white photographs. Now, with the help of digital processing technologies and machine learning, we could colorize these black and white photos to recreate the fresh world.

2. Dataset

In this project, we need to create a dataset of images of only one class. To get the target image dataset, we use the Flickr API to download images from the Flickr database. Thus, our dataset is the image about the same class corresponding to the keyword chosen.

At first, we apply for the keys to use the Flickr API. Then we define the keyword and use the API to download the training and testing images of the same class in the same operation, which makes sure the difference of training data and testing data.

In our dataset, we select the “Mountain” class and define the image size 256x256. Finally, we download 5000 mountain images as training dataset and additional 100 new images for testing.

3. Image Preprocessing

L*a*b image are used as the data prepared for the network, because L*a*b model is more suitable than other models in the model training. About the L*a*b, the L represents lightness which ranges from 0 to 100. Positive and negative “a” values mean amount of “red” and “green”, while positive and negative “b” values mean amount of “yellow” and “blue”.

If we use CIE-Lab color space, we only need to modify gray image input’s pixel value and use modified value as the L value of predicted output, so our network only need to output two dimensions of a pixel, which simplifies the network by reducing the number of required computations.

The process of generating dataset: First, load the RGB images to array X, then divide 255(the range of RGB); Then “ImageDataGenerator” of Keras can help us realize data argumentation reducing the overfitting; Convert the image from RGB to L*a*b through rgb2lab(). In this step, we divide the whole training data after data argumentation into batches. The size of each batch is 10. The X training is the value in “L”, while the y training shows the value in “a” and “b”. Finally, we generate the training dataset matching model input shape.

4. Model training

4.1 Model architecture

The color values of pixels in an image are spatial data in order words one pixel is meaningful to its neighbor pixels. Thus, we decide to build our model using convolutional neural networks, which can get the features of images after training.

The model’s input shape is (256,256,1), which is value in the L color space, and the output shape is (256,256,2) representing the a*b in L*a*b color space. The filter size is 3x3, we set the activation “relu” and padding “same” [1]. The output layer provides the output shape (256,256,2) representing a*b. However, the former layer use “relu” activation making nonnegative output. In the final layer, we use the activation “tanh” to translate the nonnegative input to real values, which represents the value of a*b (may be positive and negative). [2]

The model summary is shown in the following:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 64)      640
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 64)      36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 128, 128, 128)     73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 128)       147584
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 256)       295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 256)       590080
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 512)       1180160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 256)       1179904
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 128)       295040
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 64, 64, 64)        73792
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 128, 128, 32)      18464
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 128, 128, 2)       578
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 256, 256, 2)       0
=================================================================
Total params: 3,892,194
Trainable params: 3,892,194
Non-trainable params: 0
```

4.2 Loss Function

We employ MSE (mean square error) between the predicted value of pixel and the real value of that pixel(y), which can reflect the difference between the colors in a*b space.

$$L(\theta) = \frac{1}{K}\sum_{k}\frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}(y_{i,j,k} - \hat{y}_{i,j,k})^2$$

In the formation, K represents the output level(a*b), $H$and W donate the height and width(256x256). θ means the trainable parameters of network.

$y_{i,j,k}$and $\hat{y}_{i,j,k}$ shows the color value of the pixel(i,j) in a or b of L*a*b color space.

The formation is the loss function of one image input. If the input is multiple images(L). The loss function of model is the average of all images' MSE.

$$L(\theta) = \frac{1}{L}\sum_{l}\frac{1}{K}\sum_{k}\frac{1}{HW}\sum_{i=1}^{H}\sum\nolimits_{j=1}^{W}(y_{l,i,j,k} - \hat{y}_{l.i,j,k})^2$$

4.3 Model training

During the model training, the model parameters are updated to minimize the loss function. To reduce the overfitting and increase the training speed, we also divide the data into batches. The batch size is 10. We choose the "rmsprop" optimizer, which is a mini-batch version of "rprop".

The number of epochs is difficult to be determined, because the efficiency of model cannot directly be shown as the value of loss function. While number of the epochs increases, the loss value keeps decreasing. Thus, we try to set the epoch number 100, which may be modified according to the result.

To optimize the colorization effect, we choose to train and test the image on the same class. For example, in this project we only download the images about "Mountain" and use the "Mountain" images to test the model performance.

4.4 Testing and Image reconstruction

The process of using the model: Load the grayscale photographs(256x256), then do the image preprocessing but only take the L layer as the input of model. Use the trained model to predict the output. After the predicting, we can rescale output and combine the input and output building colored image.

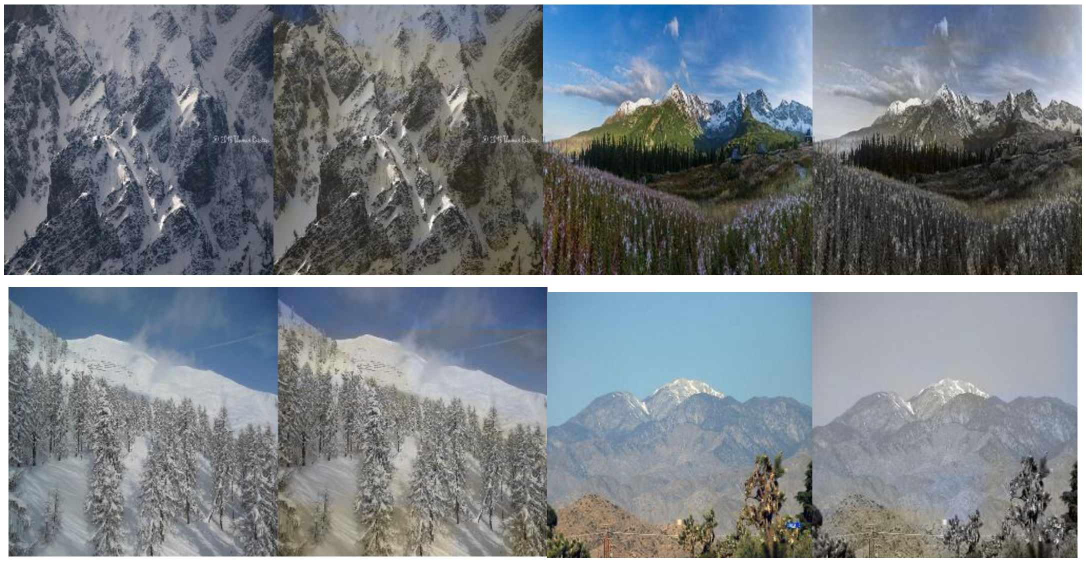
Compare to RGB, the CIE-Lab color space is harder to be recognition by the human, so we convert image back to RGB in the final output. Therefore, we save the output image in the .png. The colored images are saved in the result file.

5. Results Evaluation

The network is trained on the dataset from the Flicker and test the model on the test dataset. The quantitative result is only the value of the loss function, which cannot reflect the performance of network directly. However, we don't find a quantitative function to mapping the performance. We show the performance of model by listing the same image's real version and predicted version.

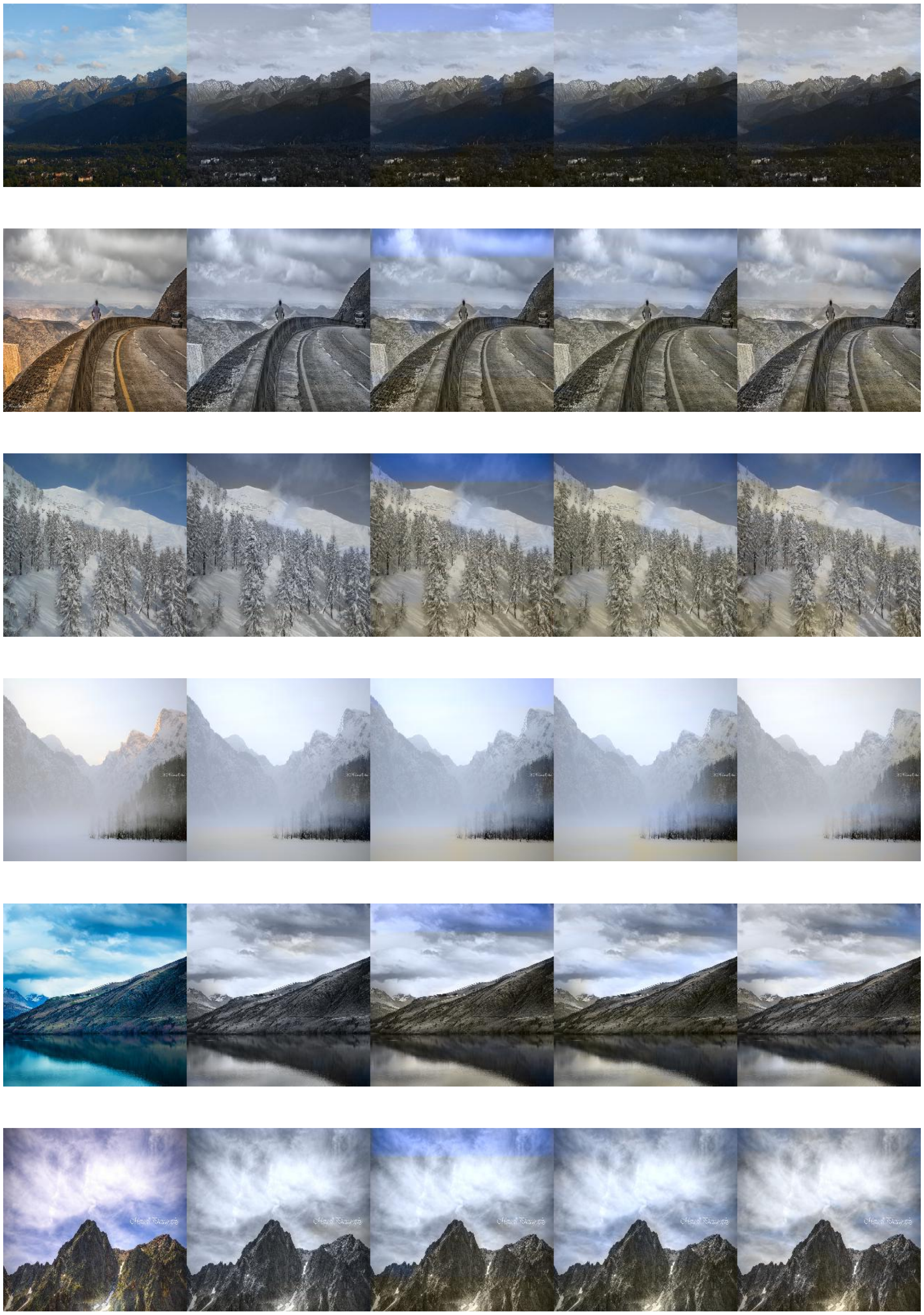
5.1 The performance of colorization

We judge the model performance by our opinion that whether the colorization is logical. The result is the model work well, because most of colorization outputs are suitable. When certain features appear, our model performs better than others. For instance, the tree and snow are well recognized, while the human and other specific objects are in wrong color or even not colored in visually. In addition, our model is not sensitive to the red, the output images never show red at the pixels where are in red in the real version.



5.2 Performance under different epochs number

During the experiment, it is difficult to determine the optimal value of epochs number. Thus, we compare the model's performance under different number of epochs. The number varies from 10 to 150. To observe the results easily, we put the results of same test image into one image. From left to right, the parts of image are real image, 50 epochs model's output, 100 epochs model's output, 150 epochs model's output. 200 epochs model's output.



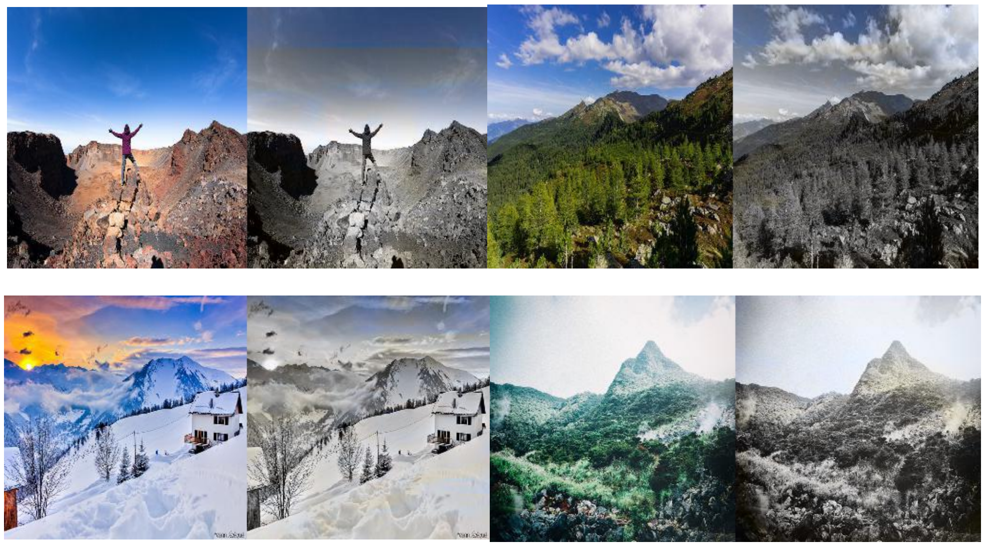
As the number of epochs increasing, the performance of network is better. In the comparison of outputs, when the epoch is 100, we notice the upper part becomes blue, but as the training going on, this problem is solved by the model.

In our opinion, the model can get more features and more color types as the epochs increasing. However, we don’t have the enough time to train the model more times(epochs), our prediction still need to be verified.

5.3 Performance between different images

We compare all 100 testing images, and discover the performance of colorization results are quite different. For example, some grayscale pictures are well colored, while some are still like grayscale pictures.

Among the successfully colored pictures posted former, there are some output pictures which are not obvious. When a picture is very different on colors from other, the model is likely to color it with poor effect.



5 Conclusion and Future works

Depending on the test results, this project shows that the automatic colorization could be suitable for some tasks. The most grayscale images are colored successfully and logically, but the model can still be improved in the following four parts: dataset size, optimizer, epochs and network architecture**.**

In terms of dataset, we only use 5000 images from Flicker for training, so the size of our dataset is not large enough to guarantee the network can get a good performance. Although data argument can extend the size of training data, the number of the data still cannot guarantee the good parameters. To get the useful features, we only choose the training and testing images in the same class ("Mountain"), which can concentrate the feature of the dataset. It is also a way to reduce the request of the dataset size.

About optimizer and number of epochs, it is impossible to make sure optimal optimizer type and number. Because during the process of training, we only print the MSE. Even at the testing part, we don't find a suitable function to evaluate the performance. The evaluation depending on feeling obstructs finding out the optimal value. To solve this problem, we need to induction a performance evaluation function, else traversing all possible values is the only way which wastes a lot time.

The network can be improved in two parts: multiple classes, network architecture. These factors influent the performance of model directly, the optimization of them can make the result better.

In our project, we use the images from same class to concentrate the features. However, the images from same class leads to the lack of diversity. We don't know whether our network works good when the dataset including the images from multiple classes. If the network doesn't perform well on the mixed dataset, there is a solution that training multiple models with different topics, which is the combination of image classification and colorization. If there is an image need to be colored, we use image classification to get the topic of that picture, then choose the corresponding model to predict the output.

In terms of network architecture: According to the relative articles and result evaluation, we realize our network need to enhance the weight of image feature, which can be added to the middle layer. The feature extractor can get the high-level information of the input image that can benefit the colorization. One article puts forward to add an Inception model, then combine the inception model's output with the CNN, which can improve the image features’ affection on the colorization results. [3]

6 Reference

1. Deshpande A, Lu J, Yeh M C, et al. Learning Diverse Image Colorization[J]. 2017:2877-2885.

2. How to colorize black & white photos with just 100 lines of neural network code, https://medium.freecodecamp.org/colorize-b-w-photos-with-a-100-line-neural-network-53d9b4449f8d

3. Baldassarre F, Morín D G, Rodésguirao L. Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2[J]. 2017.