Suyu Huang, Cong Deng, Bowei Feng

CS520 INTRO TO AI

14 December 2018

Colorization

Representing the Process

Gray images can be represented in grid of pixels, for each pixel there is a value that is related with its brightness. The value span from 0 to 255, which means black to white. And the results we want to get are color images consisting of three layers: a red layer, a green layer, and a blue layer. Just like gray images, each layer in a color image has a value from 0 to 255. The value 0 means that it has no color in this layer. If the value is 0 for all color channels, then the image pixel is black.

As we know, a neural network creates a relationship between an input value and output value. As for our project, the network needs to find the features that link gray images with colored ones.

Here, we use an algorithm to change the color channels, from RGB to Lab. L means lightness, and a and b for green-red and blue-yellow. And the value of L is same as the value of gray images, so we just need to predict two channels in our prediction from gray images to colored ones.

So, in our neural network, we include L/grayscale image as our input, which means we turn one layer into two layers, we use convolutional filters. Each filter determines what we see in

a picture. They can highlight or remove something to extract information out of the picture. The network can either create a new image from a filter or combine several filters into one image.

Data

We use a Dogs Dataset from :

http://vision.stanford.edu/aditya86/ImageNetDogs/

We use a Faces Dataset from :

https://github.com/2014mchidamb/DeepColorization/tree/master/face_images

Preprocessing: we use ImageDataGenerator of Keras, which increase our dataset since we only have 400 images in our training dataset. ImageDataGenerator is a method that increases the variations of images in dataset by using horizontal/vertical flips, rotations, variations in brightness of images, horizontal/vertical shifts etc.

Evaluating the Model

We simply use the RGB difference between each image of the image generated by the neural network and the original image as our evaluation method. The specific formula is as follows:

$$\sum_{i=0,j=0}^{n} (O_{i,j,R} - P_{i,j,R}) + (O_{i,j,G} - P_{i,j,G}) + (O_{i,j,B} - P_{i,j,B})$$

Where *n* represents the size of image. In this way, we can evaluate the results of the training and continue to get better training.

Training the Model

The Stanford Dogs dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization.

Representing the problem is one thing, but can you train your model in a computationally tractable manner?

We trace the loss value for each epoch, and use gradient descent to minimize loss function for data in per batch size.

What algorithms did you draw on?

Loss Function:

- The loss function we decided to use is mean square error. Mean Square Error (MSE) is the most commonly used regression loss function. MSE is the sum of squared distances between our target variable and predicted values.The formula is following:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)$$

Optimizer:

- The optimizer we used is Root Mean Square Prop. RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class:

$$E[g_2]_t = 0.9 * E[g_2]_{t-1} + 0.1g_2$$

$$\theta_{t+1} = \theta_t - \eta\sqrt{E[g_2]_t} + \varepsilon g_t$$

Model:

CNN layers that we built is as following:

- 2D Convolution layer with 128 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Convolution layer with 128 filters, 3*3 kernel_sizes, same padding, relu activation, 2 strides
- 2D Convolution layer with 256 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Convolution layer with 256 filters, 3*3 kernel_sizes, same padding, relu activation, 2 strides
- 2D Convolution layer with 512 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Convolution layer with 512 filters, 3*3 kernel_sizes, same padding, relu activation, 2 strides
- 2D Convolution layer with 256 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Convolution layer with 128 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Convolution layer with 64 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Upsampling layer with size (2, 2)
- 2D Convolution layer with 32 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Upsampling layer with size (2, 2)
- 2D Convolution layer with 16 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Convolution layer with 2 filters, 3*3 kernel_sizes, same padding, relu activation
- 2D Upsampling layer with size (2, 2)

Explain:

Our goal is to make our neural network find feature to map grayscale images to their colorful version. First we use a 3*3 filter to scan each image from the top left to bottom right to generate

128 images, which can represent simple features, like edges, dots, lines. To make computer understand images at higher level, we half the size of image by specifying strides with 2.

To colorize the image, location of pixels is very important. As we know, although the max pooling layers can increase the information density, it distort images, which is not desirable. That is why we use 2d Convolution layer by specifying strides with 2 to downsize the image by half, which is a approach to increase the information density without distort images. To keep the image ration, we specify padding with "same".

Since we downsize the image by half 3 times, we built 3 2D upsampling layers with size (2, 2).

How did you determine convergence?

To determine convergence is just observing the loss value for each epoch to see the tendency or the pattern. If the loss value keep oscillating during hundreds epoch without figuring out a way to decrease, then we determine it has converged. In our cases, when we are dealing with face images, the loss keep decreasing, we didn't see any sign of convergence with our limited computational resources and running time.

```
Epoch 600/800
4/4 [==============================] - 3s 629ms/step - loss: 5.5403e-04
Epoch 601/800
4/4 [==============================] - 3s 638ms/step - loss: 4.2048e-04
Epoch 602/800
4/4 [==============================] - 3s 638ms/step - loss: 4.9417e-04
Epoch 603/800
4/4 [==============================] - 3s 630ms/step - loss: 7.4852e-04
Epoch 604/800
4/4 [==============================] - 3s 636ms/step - loss: 4.7662e-04
Epoch 605/800
4/4 [==============================] - 3s 644ms/step - loss: 5.1679e-04
Epoch 606/800
4/4 [==============================] - 3s 627ms/step - loss: 5.2795e-04
Epoch 607/800
4/4 [==============================] - 3s 658ms/step - loss: 3.9260e-04
Epoch 608/800
4/4 [==============================] - 3s 646ms/step - loss: 4.6249e-04
Epoch 609/800
4/4 [==============================] - 3s 654ms/step - loss: 5.6421e-04
```

```
Epoch 790/800
4/4 [==============================] - 3s 629ms/step - loss: 2.8566e-04
Epoch 791/800
4/4 [==============================] - 3s 631ms/step - loss: 3.3898e-04
Epoch 792/800
4/4 [==============================] - 3s 634ms/step - loss: 4.7235e-04
Epoch 793/800
4/4 [==============================] - 3s 657ms/step - loss: 4.5658e-04
Epoch 794/800
4/4 [==============================] - 3s 641ms/step - loss: 3.4253e-04
Epoch 795/800
4/4 [==============================] - 3s 642ms/step - loss: 3.2125e-04
Epoch 796/800
4/4 [==============================] - 3s 647ms/step - loss: 3.2720e-04
Epoch 797/800
4/4 [==============================] - 3s 638ms/step - loss: 3.2969e-04
Epoch 798/800
4/4 [==============================] - 3s 646ms/step - loss: 3.6113e-04
Epoch 799/800
4/4 [==============================] - 3s 634ms/step - loss: 4.3455e-04
Epoch 800/800
4/4 [==============================] - 3s 638ms/step - loss: 2.9717e-04
```

When dealing with dogs dataset, although we found that the rate of decreasing in loss function is slow, it is hard to tell if it has converged without doing more epochs because of our limited computational resources.

```
Epoch 1105/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0021
Epoch 1106/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0021
Epoch 1107/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0019
Epoch 1108/1300
16/16 [==============================] - 6s 366ms/step - loss: 0.0019
Epoch 1109/1300
16/16 [==============================] - 6s 366ms/step - loss: 0.0020
Epoch 1110/1300
16/16 [==============================] - 6s 366ms/step - loss: 0.0020
Epoch 1111/1300
16/16 [==============================] - 6s 366ms/step - loss: 0.0020
Epoch 1112/1300
16/16 [==============================] - 6s 366ms/step - loss: 0.0020
Epoch 1113/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0018
Epoch 1114/1300
16/16 [==============================] - 6s 366ms/step - loss: 0.0020
```

```
Epoch 1290/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0018
Epoch 1291/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0017
Epoch 1292/1300
16/16 [==============================] - 6s 366ms/step - loss: 0.0019
Epoch 1293/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0018
Epoch 1294/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0018
Epoch 1295/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0019
Epoch 1296/1300
16/16 [==============================] - 6s 366ms/step - loss: 0.0018
Epoch 1297/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0016
Epoch 1298/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0018
Epoch 1299/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0017
Epoch 1300/1300
16/16 [==============================] - 6s 365ms/step - loss: 0.0019
```

How did you avoid overfitting?

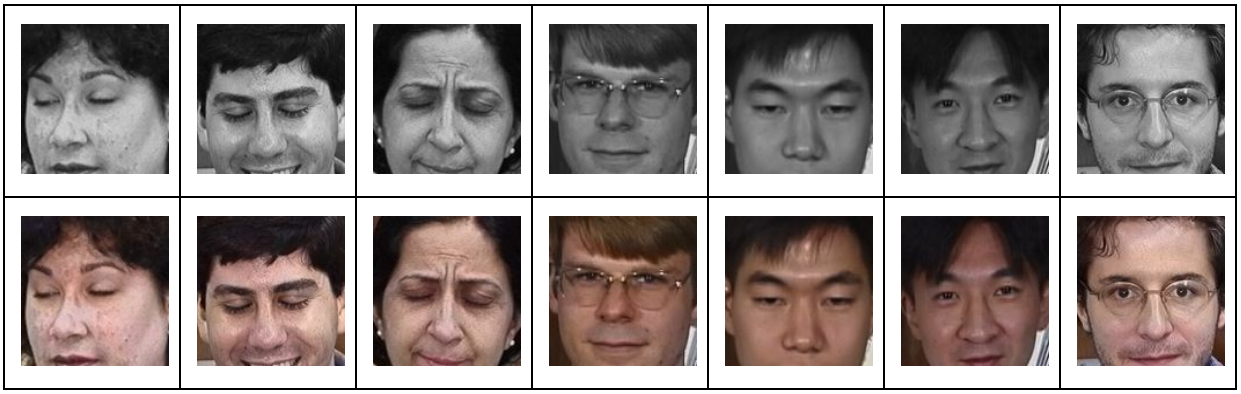
We did following way to avoid overfitting.

- Reduce the dimension of the input and do feature selection is a way to avoid overfitting. It seems that add max pooling layer is a good option. But as mentioned above, max pooling layer distorts images, which is not a good ideal in colorize network. Thus what we did is to specify strides with 2 so that it can reduce the dimension of the input by half without distorting images.

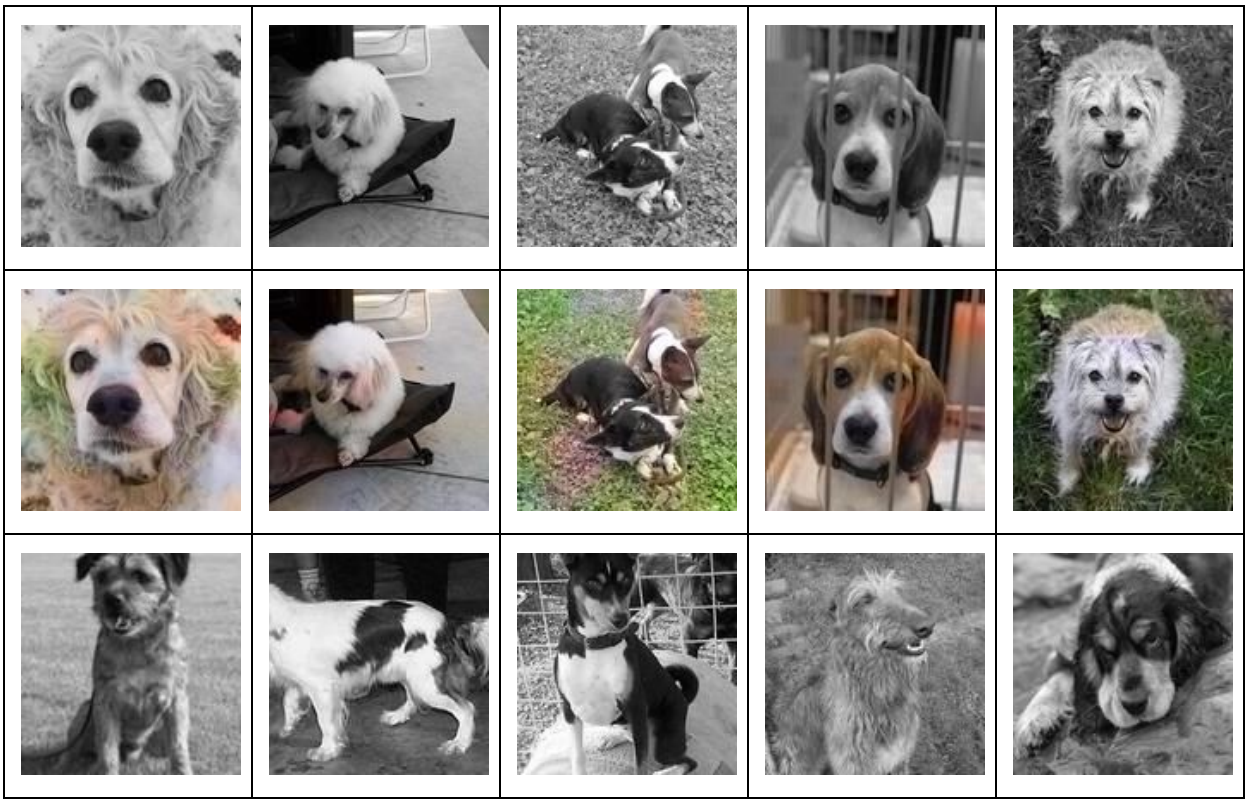
- Another way to avoid overfitting is to early stop training. Since we have very limited computational time, our model stop actually stop learning early.

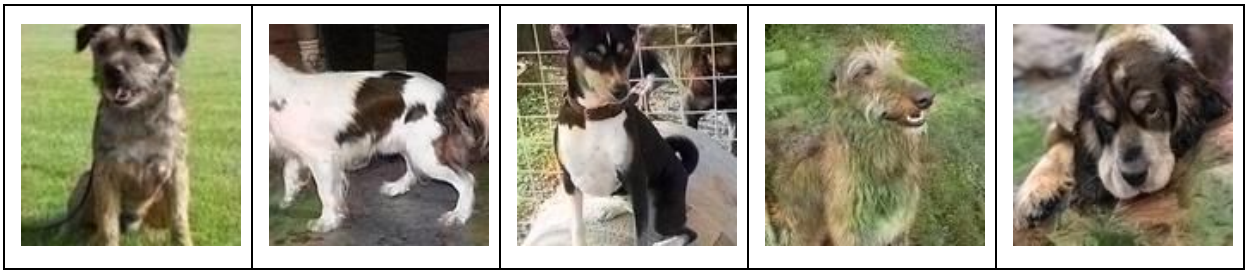
Assessing the Final Project

Test Result (Used 200 face images in dataset, epoch = 800)



Test Result (Used 400 dog images in dataset, epoch = 1300)





How good is your final program, and how can you determine that? How did you validate it?

There is two way to determine how good is our program.

First one is subjective evaluation. Just look at our result above, when dealing with face image, it got decent result, correctly differentiating human's face and hair and dyeing them correctly. When dealing with dogs image, it might sometimes colorize dogs' fur with green mistakenly. However, generally speaking, it is able to differentiate dogs and other objects, like grass, ground, furnitures in people's house and colorize them with reasonable colors.

Second one is objective evaluation. When dealing with face dataset, after 800 epochs, the loss value returned by evaluation method of keras in test mode is 0.0014189. When dealing with dogs dataset, after 1300 epochs, the loss value returned by evaluation method of keras in test mode is 0.0076965.

What is your program good at, and what could use improvement?

My program is good at colorizing images and is able to have decent results when images in training dataset are similar(e.g they are all face images or dogs images). If images in training dataset are diverse, the image our model predict will be brownish. It is because our model doesn't have a strong ability to determine different object in a image. One approach to improve it is to use classifier to equip our model with the ability to differentiate different objects in image,

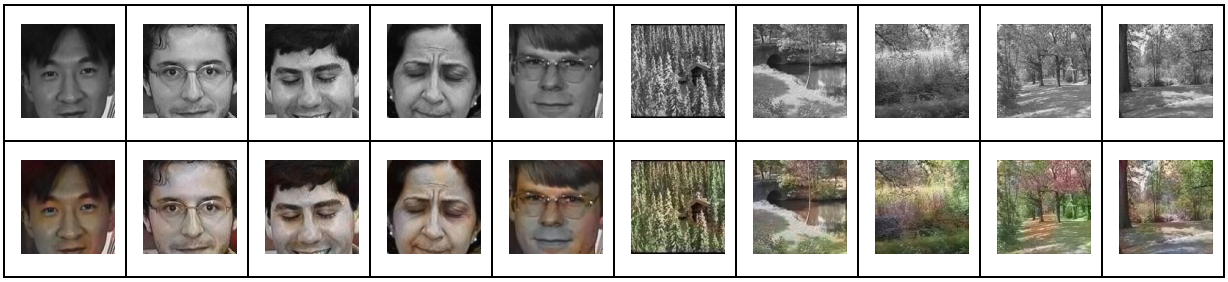
then it can link objects with specific color. In the further study, we can introduce inception resnet v2, a powerful classifiers, to our model, which can be used to extract the classification layer.

Do your program's mistakes 'make sense'?

Yes, it made mistakes in some pictures! Observe the result when using dogs image as training data, we find that our model dye dog's fur green mistakely, which means that it sometimes struggle in differentiating dogs' fur and grass. Besides, there are some cases that our model paint ground green. The reason might be that in our dogs dataset, dogs and grass are very likely appearing in the same pictures.

What happens if you try to color images unlike anything the program was trained on?

To see what happens if we try to color images unseen, just observe following result. Notice that the model has only seen dogs images.



Observe first 5 face images, our model has no ideal how to colorize face correctly.

Observe last 5 scene images, it seems that our model gets better result compared to what it performed in dealing with face images. It dyes tree green or yellow and paint ground gray because it has seen how to colorize them in dogs dataset, therefore it just simply colorize them without generating more nuanced colors.

In general, when our model is dealing with something it didn't see before, it might regard objects mistakenly as other object to which it thought is similar in training data, which leads to dye them incorrectly.

What kind of models and approaches, potential improvements and fixes, might you consider if you had more time and resources?

- Sometimes it is hard for our model to determine two different objects if they are similar, one improvement is to manually draw lines or spots to images so that it can guide our model to recognize them.
- If images in dataset are similar, it gets decent result but has trouble in generalizing. Thus using a more diverse dataset and develop more complex neural network might be helpful in generalizing. One approach to realize this goal could be implementing a weighted classification to classify different objects in images.
- Create a model to the coloring network, which can be an amplifier within RGB color space.
- Just try a larger dataset and epoch more times to see what happen.
- Try smaller batches because a too large batch size might prevent convergence.
- Try different optimizer and tune hyperparameters. It is hard for beginner to determine which optimizer is better, how to determine the hyperparameter. Thus if we have more time and resources, one way to gain a better understanding of neural network is just to do more experiments.