Final Project: Image Colorization Using Convolutional Neuron Network

Pinshuo Ye, Qinyi Liu
N19437116, N12748239

Introduction

Our source code for this research project on GitHub: https://github.com/yepinshuo/Image_Colorization

Problem Description: Recently there are growing demands on coloring various old pictures, because colorizing a photo or a video gives more details, and it could bring us a more realistic sense of what life was like in the past days. Our team is quite interested in this topic, and our aim is to create a colored reproduction of grayscale images, or image colorizing, on our chosen dataset.

Methods and Models: Ideally, the model will have one channel input and three channel output, where though grayscale pictures still have RGB attributes, according to the formula that calculates the brightness, all three values shall be the same, thus the mapping relationship should be L to $\{$R,G,B$\}$. A simpler version of this method is having two channel output, because with L and any two of $\{$R,G,B$\}$, the left one can be calculated.

There is also another model that could simplify the previous method, if the training and calculating power is limited. We can use k-mean clusters to identify k different colors that could best recover the usage of color of the original training set. Then we can implement a classification model to colorize black and white photos. Obviously, this is a faster approach because we "compressed" the output space in k colors, but this may sacrifice the effect and the quality of result images.

The expected output is the colorized picture; but for image colorization problems there are no such results as the "absolute best" colorization, because these result can only be judged by human eyes subjectively. Therefore, in this research, we will try to identify "better" colorization subjectively in multiple training outcomes.

Literature Survey

Based on our previous search, here are some reference that we will possibly use:

- Grayscale Wikipedia: Basic understanding on grayscale images and how to convert color images to grayscale
  https://en.wikipedia.org/wiki/Grayscale
- Automatic Image Colorization Using Machine learning: Provides a basic model using conv-deconv neuron networks on colorizing
  https://xiangyutang2.github.io/auto-colorization-autoencoders/
- Image Colorization: A Survey and Dataset
  https://arxiv.org/abs/2008.10774
- Zhang, Richard and Isola, Phillip and Efros, Alexei A (2016). Colorful Image Colorization: A specific example of using image colorization, and a brief overview on Lab format
  http://richzhang.github.io/colorization/
- Load data from Kaggle to Colab: How to load image data
  https://github.com/prasunroy/cnn-on-degraded-images
- Colorizing B&W Photos with Neural Networks: Another example project on colorization
  https://blog.floydhub.com/colorizing-b-w-photos-with-neural-networks/
- Image Colorization with Deep Convolutional Neural Networks: Some techniques on training convolutional model on image
  http://cs231n.stanford.edu/reports/2016/pdfs/219_Report.pdf
- ColorUNet: A convolutional classification approach to colorization
  https://arxiv.org/pdf/1811.03120.pdf

These references gives us ideas about image colorization on complicated models and detailed images, and we will try to simplify the dataset and model in order to fit our machine's calculation power.

Dataset Sources

We decided our dataset would be chosen from the resources of some professional deep learning websites, and possibly will be some simple-structured and categorized images before.

- Kaggle:
  https://www.kaggle.com/prasunroy/natural-images
- IMAGENET:
  https://image-net.org/download-images.php

- CIFAR10:
  https://www.cs.toronto.edu/ kriz/cifar.html

For example, in Kaggle datasets, we can use fruit, vegetables or animals like cats and dogs, so that our training process could be simplified. If we are able to handle complicated datasets with multiple categories, then we will try to use mixed categories images in our training. As for dataset in ImageNet, it contains image datasets for classification problems, which also could be utilized on colorizing.

Since Kaggle supports directly data loading on Colab, we will use dataset on Kaggle as described at https://github.com/prasunroy/cnn-on-degraded-images. Here are some sample pictures in figure 1.

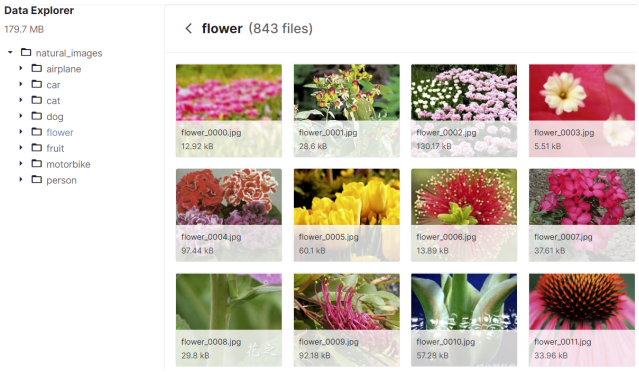


Figure 1: Dataset Sample

Additionally, the original images for training have to be grayscaled, so we will convert the image data into black and white, then use the black and white pictures as our input. It can be done using PIL in python. For each input, we can get a new greyscale image by Image.convert(). Here shows an example in figure 2:

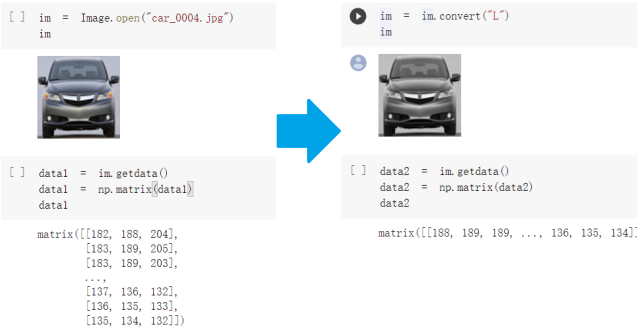


Figure 2: Convert of Original Image

Except for PIL library, there are also rgb2grey available from skimage.color. After using this package, we can convert the results back into 2-dimensional arrays, then we will receive converted black and white images. Here is another example in figure 3:

According to the result above, the matrix size of the original .jpg file is 100*100*3, and after conversion it becomes

Figure 3: Converting Images Using "rgb2grey"

100*100, which is the value of brightness L calculated by {R,G,B} using the formula $L = \frac{1}{1000}(229R+587G+114B)$

Model Description

For image processing problems in deep learning, the most common and efficient way to capture the representation or crucial information from an image is CNN. Our model for image colorization is also a CNN structure, which is shown in the following figure 4:

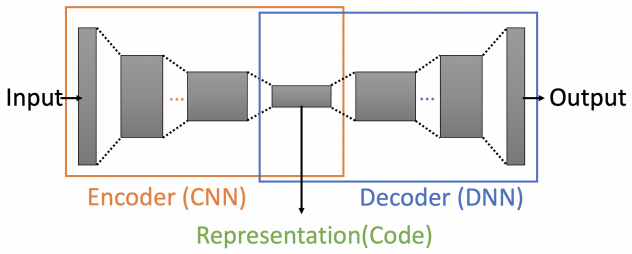


Figure 4: CNN-Encoder-DNN

There is a neuron network architecture combines convolutional neuron network (CNN) with deconvolutional neuron network (DNN), and we call it as "autoencoder", as we shown above. The autoencoder encodes image data using CNN structure, giving us an encoded vector that could extracts the important information, or the features from the input image. After reconstruction through the network, the autoencoder decodes the vector using DNN, regenerating an image, ideally the colorized version of the original image.

The inputs and outputs for our model are both images with the same size, except the output image may contain 3 different layers (R,G,B). Based on our current research, there are two ways of measuring colorized images for our training. One is the traditional RGB images, which measure the brightness of three colors; another one is the Lab images, where L represents brightness, and ab are coordinates in a color map which represents a specific color. Previous researches that using Lab measurement gives very accurate colorized result, and it is possible that Lab could be more efficient. However, we still decide to use RGB as the standard.

Although there are a lot of benefits using Lab, such as it expresses much more colors than RGB does, in our model we only need integer inputs and outputs. Without the float numbers calculation in the final stage, there shall be less error than Lab.

For the second method using k-mean, instead of capture the representation or crucial information from the image, we generate k color using k-mean method that could best recolor the image. After training we should get a batch of k images that size equals to the original image, and we use "softmax" function to decide which color to use on each pixel. Therefore, we could obtain a colored image with the same size. Here is an example model in figure 5:

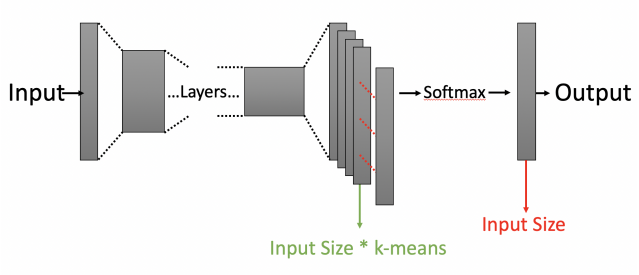


Figure 5: CNN-Encoder-DNN

Instead of providing an RGB in result images, we choose a color range that could best regenerate original pictures using k-means method. This could speed up the process and probably obtain nearly as good quality results as the autoencoder method. However, we still need to pay attention on training multiple-category datasets, because the result highly depends on the diversity of colors of the orginal images, and loss of diversity might cause a relatively poor quality colorization.

As for implementing model, we will refer to the open source models from the researches, adjust the model and apply it to our datasets. The models for training neuron network for image colorization are various, and we may need to adjust the model according to our training results. Other than that, the loss function and gradient descent method might still need adjustment during the training process.

Loss Function / Training Parameters

Because we are currently using RGB form images, a simple loss function would be the sum of pixel-by-pixel losses. To optimize result, we will try to add some other penalties like the variance of the whole image or its difference with the variance of original image.

On the other hand, we use the other method and consider about loss function for this process. In this simpler version of our training process, we could slightly change the model and our loss function in order to save time to test some trivial results. For example, if we are training a single category images, we may use a k-mean clusters method, as we mentioned in the previous sections, to calculate k colors that represents the k-mean colors of the dataset, and we can simply use these k colors to recolorize the black and white images. Then, this colorizing problem is converted from extracting features from the images to choosing the right color for a single pixel from k colors we implied before. Therefore, we can temporarily skip the deconvolutional neuron network, and focusing a classification problem, which also means we could use $softmax$ and **cross-entropy loss** as our loss function of the model. Here is an overlook of our loss functions in our model and some other hyperparameters in figure 6:

```
cnn.compile(loss='mean_squared_error', optimizer='adam', metrics = ['
history = cnn.fit(X_train, y_train, epochs=100, validation_split=0.1)
```

Figure 6: Cross-Entropy Loss and Other Parameters

There are also many modifications possible on the structure of the model or some hyperparameters. First of all, in order to calculate the result as accurate as possible, we may consider adding hidden layers to the basic model structure (which we actually did in next sections), but adding layers is also limited by our computing powers of cloud machines, so we have to decide the rough structure by testing several times. Second, we can change the epochs and the learning rate to adjust the learning time and its accuracy (shown in previous figure 6). And third, for the simper version of our training process, we can adjust the value k if we need more or less colors in our colorized images. Here is an example we can adjust hyperparameter k in our code in figure 7:

```
kmeans = KMeans(n_clusters=32,random_state=0)
kmeans = kmeans.fit(X)
```

Figure 7: k-mean Clusters

There are also some methods like changing the activation functions like relu and tanh to get better image quality. If these extra loss functions and training parameters are useful, we will consider adding them into our model. Ideally, if our model goes well in our training process with proper loss function and training parameters, the outcomes of the model should be clear and colorized properly.

One interesting thing is that we don't expect a specific color of the objects in the picture, which is because there can be several possible colors of one thing, for example, an apple can be either red or yellow or even green. The final result should give us the sense of reality. It is something like Turin Test, and people who test the result can tell the goodness of the training result. Based on previous researches, the use of pixel-by-pixel loss function is relatively conservative on the result of colorized images. That means the model is likely to choose a mixed combination of colors based on the anticipated probabilities, which might lead to a mixed colorization on our result images. According to the paper "Colorful Image Colorization", they trained the network using 1 million pictures, and the loss function is done by asking humans if they can tell whether a photo is true or fake. It is impractical for us to do the survey to get the loss, so we will observe the

results by ourselves to justify the quality of colorized image, as a plus to the previous loss function..

Also, it is likely that some problems might raise during training, such as the important features of the original image is not extracted properly, and the colorized images don't have a clear boundary between objects. Then there would be a lot of noises in the result. Thus choosing a appropriate training model, loss functions and training parameters is significant.

Preliminary Results

In the first phase of our project procedure, as we mentioned before, we use a simple model to test our ideas, which is extracting simple information from our training images. What we did in this part is to load the data, cluster them with k-mean method, change the colored picture into black and white for training, construct a simple model with hidden layers, training the data, observe the training loss and validation loss, and finally predict the result. Here is a summary of our model in figure 8:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        1664

 max_pooling2d (MaxPooling2D  (None, 32, 32, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        102464

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 dense (Dense)               (None, 32, 32, 32)        2080

 dense_1 (Dense)             (None, 32, 32, 4)         132

=================================================================
Total params: 106,340
Trainable params: 106,340
Non-trainable params: 0
```

Figure 8: Model Summary

Because we used cross-entropy loss, and we have both validation and training losses calculated, there are two lines on our loss graph, which shows below in figure 9:

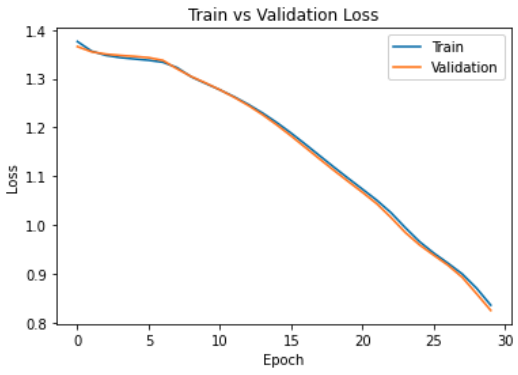


Figure 9: Training & Validation Loss

We used 10% validation to test our training results, and based on the graph above, both the losses are decreasing, and the training loss is very close to the validation loss, so we can conclude that our training process is successful.

After time-consuming training process, we finally can test our result on our test data. There are no definite right or wrong answer on colorizing image, or a measurement to calculate the correctness of colorizing, because colorizing gives just a simple reference to human.

But we can tell if our result extract the original information or not, by observing if the predicted images preserve the rough shape of the original image. Here are two images testing 16 pictures, and we can analyzing these result by human eyes in figure 10 and figure 11:

Figure 10: Original Test Images in B&W Form

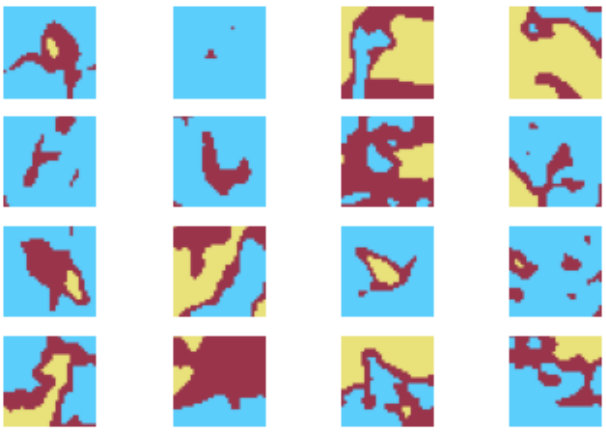


Figure 11: Predicted Test Images

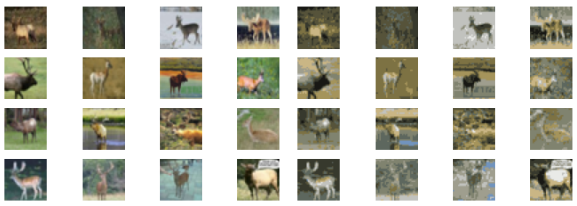
We can see that our model captured features from some of the images, like image 1, 3, 9, 11, although they are only rough shapes. And we actually used 4 colors in k-mean cluster method, so that we only use 4 colors in our result. The bright side of this result is: as long as the items in our original images could be distinguished from the background, our

model can detect the rough image shape, that means with proper longer training process, we can detect more and more items in the pictures. Plus, if we want to recolorize our images with more color, we can simply change the hyperparameters of k-mean method.

As a temporary conclusion, although our results still have a large percentage of pictures that features are not captured, we can see that we are working in the right track. As long as we make adjustment on some parameters, and running a more complicated model with longer training process, we can improve the results.

Model Training

After having some example results, we tried to do the expansion for what we have obtained. There are two steps, the first is to add more colors, which means we need more labels for the model to do the classification. When thinking about how do people usually paint something that looks real, where painters can create paintings with limited number of pigments, we conclude that with several colors a grayscale figure can also be colorized as well. Then our remaining work is simply trying to add as many colors as possible in our classification model, and make sure the training results are reasonable and training procedure is effective. Here are a comparison of original images and k-mean clustered images in figure 12:



(a) Original figures (b) Clustered figures

Figure 12: Clustering of each pixel

However, when modifying parameters it takes a lot of time to verify correctness of the adjustment. We referred to several existing grayscale colorizing models, and the final network structure details is shown as below in figure 13. It includes 5 CNN layers and 3 dense layers, and the output layer using softmax to choose from k colors, giving the classification result.

We accomplished three types of training, with three models trained with single class figures, including cattle, deer and watercraft, giving out 12-color, 24-color and 32-color results, and mixed class figures giving out 32-color results.

Final Results

The feedback of our first test is quite good, where we choose a 12-color classification with label-4 only(figures of deer), and the accuracy turned out to reach around 85%. It convinced us that our neural network structure worked well, and for the other two test models also the final model, we let the network stay unaltered.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        640

conv2d_1 (Conv2D)            (None, 32, 32, 128)       73856

conv2d_2 (Conv2D)            (None, 32, 32, 128)       147584

conv2d_3 (Conv2D)            (None, 32, 32, 128)       147584

conv2d_4 (Conv2D)            (None, 32, 32, 64)        73792

dense (Dense)                (None, 32, 32, 64)        4160

dense_1 (Dense)              (None, 32, 32, 32)        2080

dense_2 (Dense)              (None, 32, 32, 32)        1056

=================================================================
Total params: 450,752
Trainable params: 450,752
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: Network Structure

Here are the results of 12-color model, in figure 14 and figure 15.

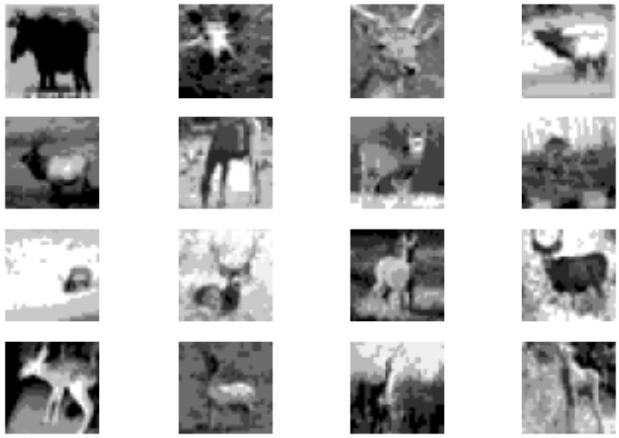
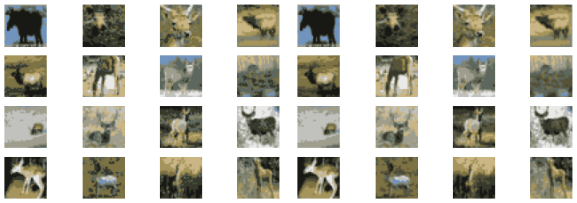


Figure 14: 12-Col Test Input



(a) Model Output (b) Test Output

Figure 15: 12-Color Result

When using only 12 colors there are not much difference between the test outputs and model outputs, but according to Figure 12, which is also clustered using 12 colors, it is obvious that the figures can loss a lot of information, because

the k-means clustered result decides the upper limitation of our training results. To get vivid pictures, more colors are needed.

Next model is trained with 24 clustered colors, and Figure 16 and 17 show the training results.

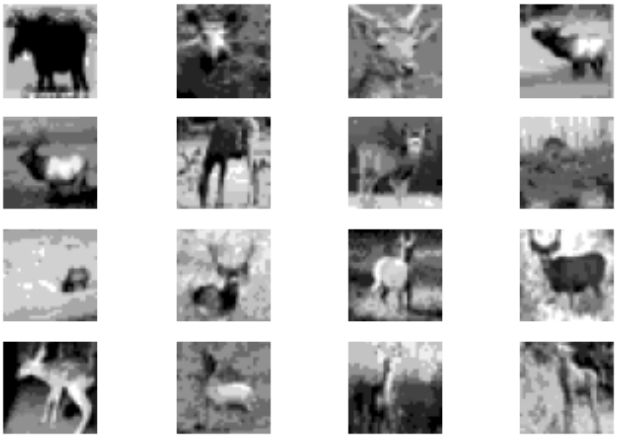


Figure 16: 24-Col Test Input

(a) Model Output (b) Test Output

Figure 17: 24-Color Result

Now the results look much better, although for several images it tends to predict the background as green rather than blue, things are more colorful, because we have more colors available in our predicted results. However, more colors in this case does not equal to more precision, we still need to justify other possible factors. From the previous results of 12 colors and 24 colors using k-mean, we can change our k value for our k-mean method bases on following principles: the size of the output, the approximate colors needed for result output, and a maximum color allowed or an upper limit for number of colors we use.

Finally, based on these principles and the computing power of our machines, we choose 32 colors, because for all test cases, the most time consuming function is fitting k-means method, and with more labels(colors) the fitting time grows linearly. This time we change the training object into watercraft, Figure 18 and 19 are the training results. We can see that with 32 different labels(colors) available in this training result, we can get better result even if the loss function is not going down very smoothly.

With Adam instead of SGD as before, the loss does not move smoothly, but it converges to a lower level and keeps its stability.

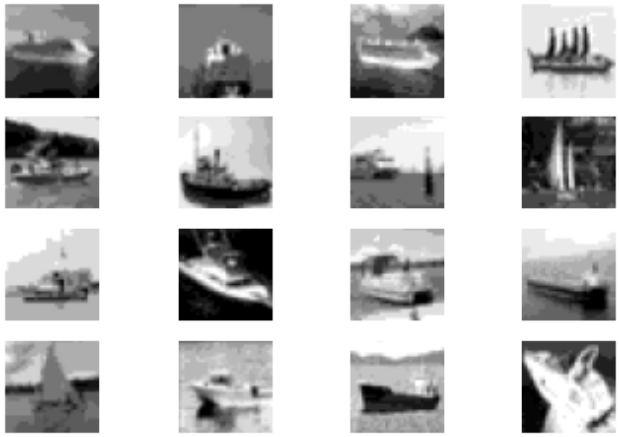
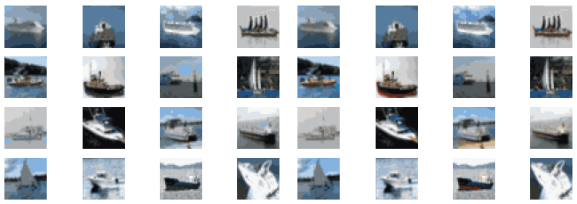


Figure 18: 32-Col Test Input



(a) Model Output (b) Test Output

Figure 19: 32-Color Test

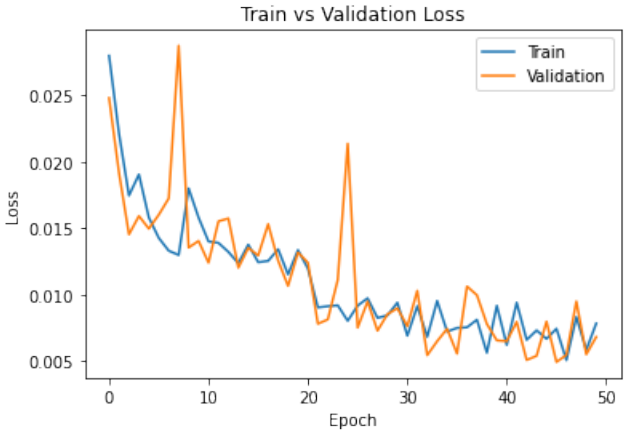
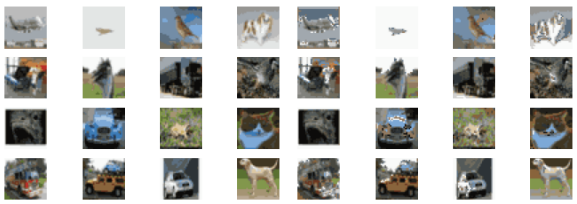


Figure 20: 32-Col Training and Validation Loss

Then we tried to train the network with mixed types of objects, while the result could not reach its quality like before. Lots of noises occur as shown in Figure 22, and the loss is hard to converge shows in figure 21, although we received a fair result output.

Figure 21: 32-Col Mixed Training and Validation Loss



(a) Test Output (b) Model Output

Figure 22: 32-Col Mixed Result

Conclusion

Again, our source code for this research project on GitHub: https://github.com/yepinshuo/Image_Colorization

What we intended to do: Originally we derived two different models to achieve image colorization, using either autoencoder CNN method or k-mean CNN method. Ideally, what we intended to do is to try different models and comparing the result, if the computing power is sufficient; or we can apply two models individually on different datasets. However, limited by our computing power and the complicated model, we had a hard time produce a meaningful result using autoencoder. We then changed strategy on k-mean method, changing different parameters to approach the colorizing effect like autoencoder.

If we have much time on this project, we can change the diversity of our datasets, since we are currently using datasets with images all same size; we can come up with a solution to scale all other datasets so that all images are in the same size, then we can have more datasets to train in this project.

What we have achieved: Other than the previous problems that we had, we still managed to solve rest of the problems. We proved our models are executable and it could produce proper results; we solved some small problems like loading images and justify images using outside references; we also find proper datasets and adjust our models so that our code could run these datasets; and finally we have loss functions, parameters, and test results that are acceptable.

To conclude, our method of colorization of grayscale figures is mainly accomplished by doing the classification for each pixel. The result shows that for same label objects such as deer or cattle, colorization model works good. But when comes to mixed categories of objects, figures are not painted as what we expect before that it should look like a real photo.

A simple way to improve this neural network's behavior is adding the number of colors. In some way if color number is extended to 2^{24}, it would be no longer a cluster model but a regression model, because it expresses the whole integer variants in format RGB. But as mentioned before, time spent on initializing the model is a linear function with the number of colors, which would be unaffordable when it grows too large.

Another way is to add a classification model in the training model, and with a combined label, network we are using now can do the colorizing much easier. Of course more layers in the network is needed, then with label information, things might be easier for the model.