# Colorization

Team member:
Weikang Li
Ningyuan Zhang
Linchen Xie
Shenao Yan

In this project, we want to colorize gray by using supervised learning techniques. As we all know,  a color image is represented by a matrix of 3-component vectors typically, where Image[x][y] = (r, g, b) indicates that the pixel position (x, y) has color (r, g, b) where r represents the lever of red, g of green, and b blue respectively, as values between 0 and 255. A classical color to gray conversion formula is given by:

$$Gray(r, g, b) = 0.21r + 0.72g + 0.007b$$

We can know from this formula that Gray(r, g, b) is between 0 and 255. Note that we are losing information when converting a color image to gray image. For most shades of gray, there will be many (r, g, b) values that correspond to that same shade. Therefore we need to get more information when we colorize a gray image.

By training a model on similar images, we can get more information about its context, so we can make reasonable guesses when colorizing. In this project, we decide to use Convolutional Neural Network(CNN) to train a model which can colorize images that are similar to training samples reasonably.

# 1. Data

Because we do not have enough time and computing resource, we just choose few simple images as our data set,  as shown below, which is a cartoon character, Bika Qiu.

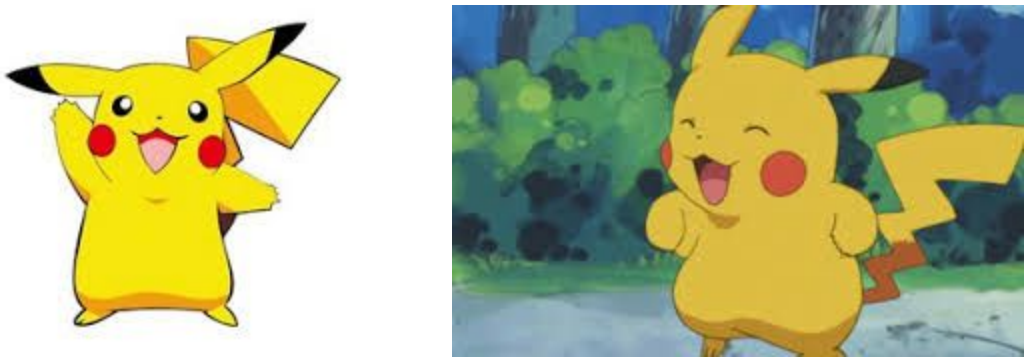

Figure 1: sample of data set

We get these raw color images from the Internet.

## 1.1 resize the data set

These raw data have different size, so we first fill the rectangular images into squares with (255, 255, 255). After visual inspection of the images, we determined 64x64 to be a proper size, and then resize all images to size of 64x64.

## 1.2 compute gray images

Secondly, we compute gray image of each images using the formula:

$$Gray(r, g, b) = 0.21r + 0.72g + 0.007b$$

Then, we divide gray image by 255 to rescale input in range [0, 1], which makes CNN is easy to train.

## 1.3 K-means

Most importantly, because we want to treat this colorization problem as a classification problem, so need find a mapping from grayscale value to (red, blue, green), each value between 0 and 255. This gives a total 16,777,216 possible colors. The thing is that we do not even need that much colors, also it is very hard to find a good mapping from a grayscale value that between 0 and 255 to all 16,777,216 classifications.

To simplify the problem, we use clustering method to find several colors that can represent original images perfectly. Specifically we use K-means algorithms to cluster 32 colors, which is enough to represent original images. By using K-means, we can minimize the amount of subjective decisions that must be made before.

Instead of realizing K-means algorithm, we decide to use sklearn.cluster.k-means after understanding this algorithm:

| k-Means Clustering Algorithm |
| --- |
| 1.Start with initial guesses for cluster centers<br>2.For each data point, find closest cluster center<br>3.Replace each center by average of data points in its partition<br>4.Repeat step 2 and step 3 until convergence |

After get the 32 colors by K-means on training set, we transform each (r, g, b) of original images to a scalar that between 0 and 31 according to these 32 clustered colors. So, images become 2D matrices.


Figure 2. Gray image, original image and image using clustered 32 colors (from left to right)

Now we just need to find a mapping from  scalars of [0, 1] to scalars of [0, 31], which is much simpler than before.

### 1.4 one-hot encoding

In order to get  a good CNN model, we need to apply one-hot encoding to labels.

At first, we did not use one-hot encoding for our labels. So when we train our model with data, we find that it is hard to converge.

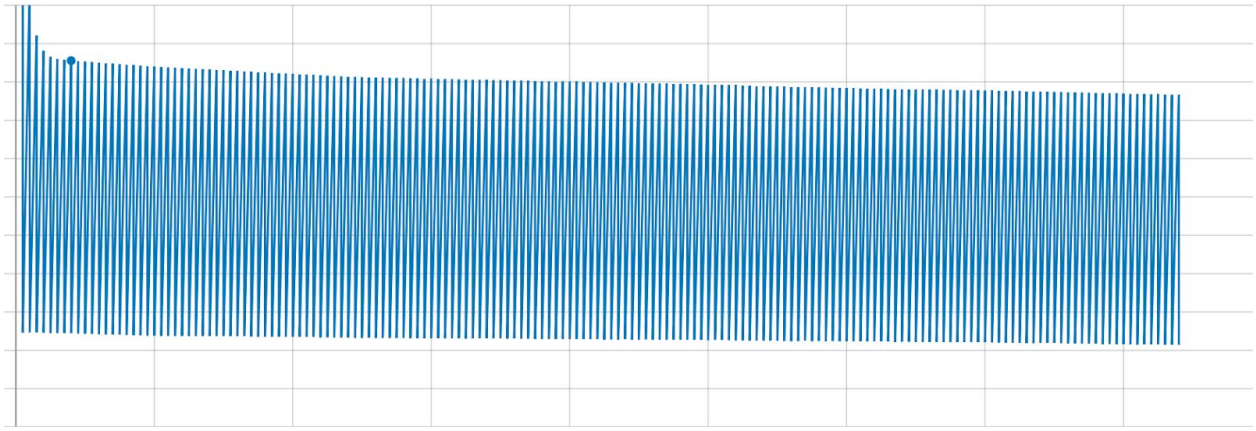

Figure 3. Loss(the model is hard to converge without using one-hot label)

The reason why we got this result is that we did not design loss function properly. Before using one hot labels, our loss function is mean of absolute value of difference between each labels and outputs.

$$Loss = Mean(\sum ABS(label - int( 31 * sigmoid(output))))$$

By using sigmod(output)*31, we can get output in range (0, 31), then round off the fractional part ,we get output in range [0,31] . But the problem is that this loss is not reasonable. For example, if a pixel should be labeled with 3, the the loss of mislabel it as 7 is 4. If the model mislabel it as 30, then loss is 26. However, we know that mislabel it as 7 or 30 should have same penalty.

After finding this problem, we use one-hot encoding to get a new format of labels.

## 1.5 Splitting data into training set and test set

We split total data set into training set and test set with 70% and 30% respectively.

# 2 Approach

## 2. 1 Architecture of CNN

Our CNN has 3 layers.

The first layer of our CNN is a convolutional layer, the size of each kernel is 3x3, 32 kernels totally, with a stride of 1. Because we need to find a mapping from a gray image to a color image, we have to get some context information. Without getting extra information, we are not able to find a good mapping, it just like a random guess instead. This convolutional layer can help us to get some contextual information. The activation function is Relu.

The second layer of out CNN is a fully connected layer, with 64x64 neural nets. The activation function is Relu.

The output layer has 64 x 64 x 32 neural nets. The activation function is softmax(we compute softmax of each 32 output, which represent label of a pixel).

Loss function is mean of absolute value of difference between each labels and outputs.

$$Loss = Mean(\sum ABS(label - int( 31 * sigmoid(output))))$$

The optimizer we use is Adadelta Optimizer.

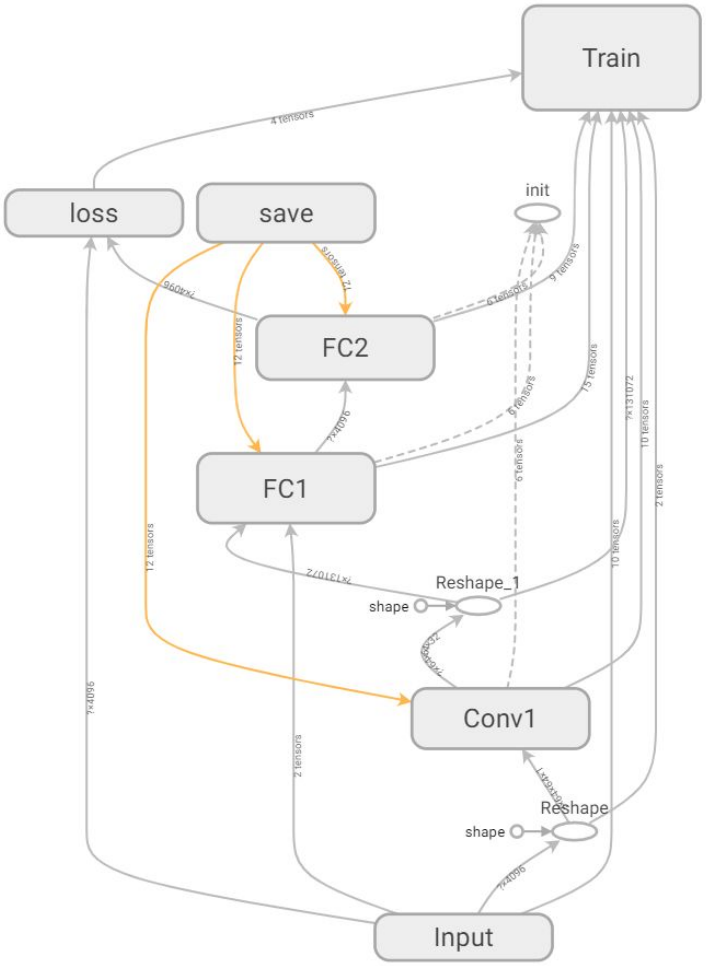


Figure 4. The architecture of CNN.

## 2.1 Implementation

We all know that we should realize these algorithm step by step, such as forward propagation, back propagation, and so on. Unfortunately, we do not have enough time to do that during this period, too much exams and projects. SO we decide to use Tensorflow to implement this CNN.

## 2.2 Evaluating the model

We know that it's hard to compute accuracy of this model.

Because even the some gray pixels are mislabeled by this model, we sometimes still think the colorization is real enough, or sometimes those mistakes surprise us. On the

other hand, sometimes a small terrible mistake made by the model can destroy total colorization. So we can not find a reasonable way to compute its accuracy.

Another thing is that we do not have a large data set, so we can just evaluate the model by perceptual error(how good do humans find the result of the model).

## 2.3. Overfitting

Because we do not have a large data set, we need to find a way to deal with overfitting.

In this project, we used “Dropout” method. Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

# 3 Result

The model trained by our data set is relatively reasonable. There is a example below:

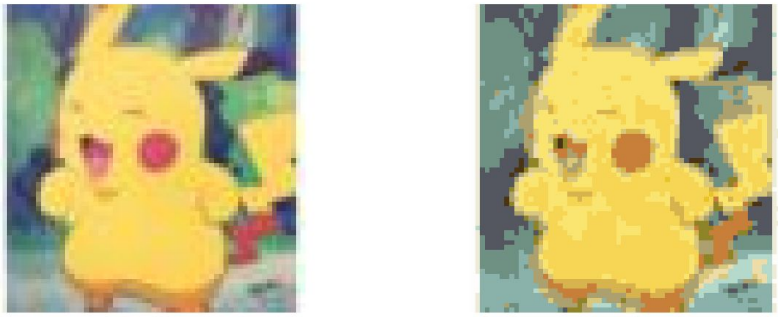

Figure 5. Original color image(left) and colorized image(right)

We think this colorization is reasonable, even it looks not good enough.

Firstly, the model can just choose 32 colors to colorize the gray image. Moveover, these 32 colors are obtained from training set, not from test set. So the model can just choose from 32 colors that get from training set, which may not be suitable for new images.

If we want this model to colorize a picture that different from training images, it will disappoint us. Because firstly, the model can just use colors that we compute based on training set to colorize the new image. Secondly, shapes, context of the new image may not be understood by our model, it just likes a random guess when choosing colors. Therefore, the model will perform very bad on new images.