CS 520: Assignment 4 - Colorization

Haoyang Zhang, Han Wu, Shengjie Li, Zhichao Xu

December 15, 2018

1 Introduction, group members and division of workload

In this group project, we implemented a image colorizer which can generate satisfying color images given grayscale images.

Name RUID	Workload
Haoyang Zhang 188008687	.
Han Wu 189008460	Wrote part of the report and participated in the discussion. Analyzed the outcome of our colorizer.
Shengjie Li 188008047	Implemented the colorizer. Trained the model. Wrote part of the report. Finished the format design of the report using LaTeX.
Zhichao Xu 188008912	Wrote part of the report. Came up with the original structure of network.

2 The Problem

- **Representing the Process:**
  1. **How can you represent the coloring process in a way that a computer can handle?** A color image can be interpreted as an array with 3 channels – channel R, channel G, channel B. A grayscale image can be interpreted as a color image being compressed to having only one channel gray, by the formula $Gray(r, g, b) = 0.21r + 0.72g + 0.07b$. So the coloring process can be seen as: given a one-channel image, produce a three-channel image. To achieve this, we are using a CNN with autoencoder.
  2. **What spaces are you mapping between? What maps do you want to consider?** Note that mapping from a single grayscale value gray to a corresponding color (r, g, b) on a pixel by pixel basis, you do not have enough information in a single gray value to reconstruct the correct color (usually).
- **Data:**
  1. **Where are you getting your data from to train/build your model?**
     We are using the The CIFAR-10 dataset.

     This dataset consists of 60,000 32×32 colour images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images in total. The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 1,000 randomly-selected images from each class.

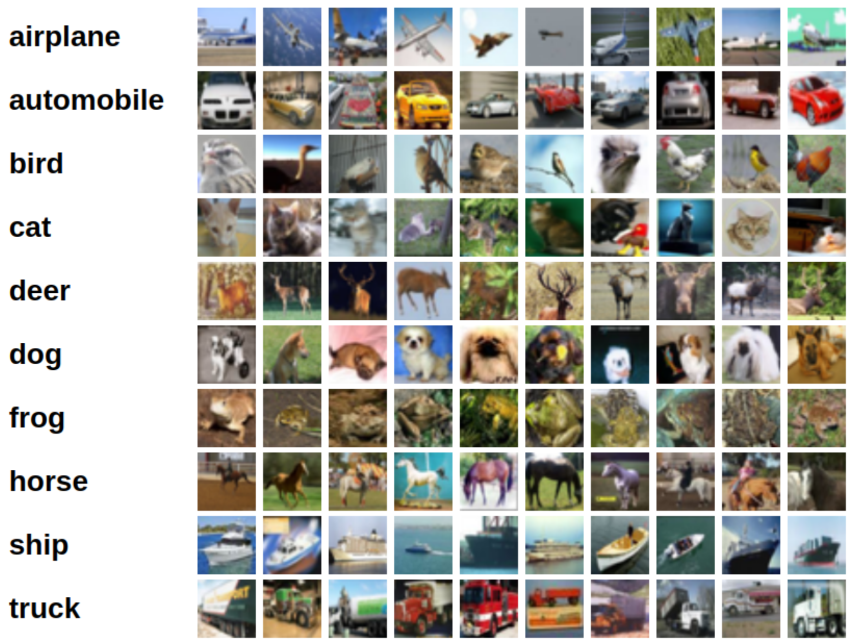

Figure 1: The CIFAR-10 dataset

2. **What kind of pre-processing might you consider doing?**
   Each training batch is a 10000 $\times$ 3072 numpy array. Each row of the array stores a 32 $\times$ 32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The value of each channel is between $[0, 255]$. We are going to scale the RGB channel from $[0, 255]$ to $[0, 1]$. By standardizing the data, we could train faster and reduce the chance of getting stuck in a local optima.

- **Our Model:**

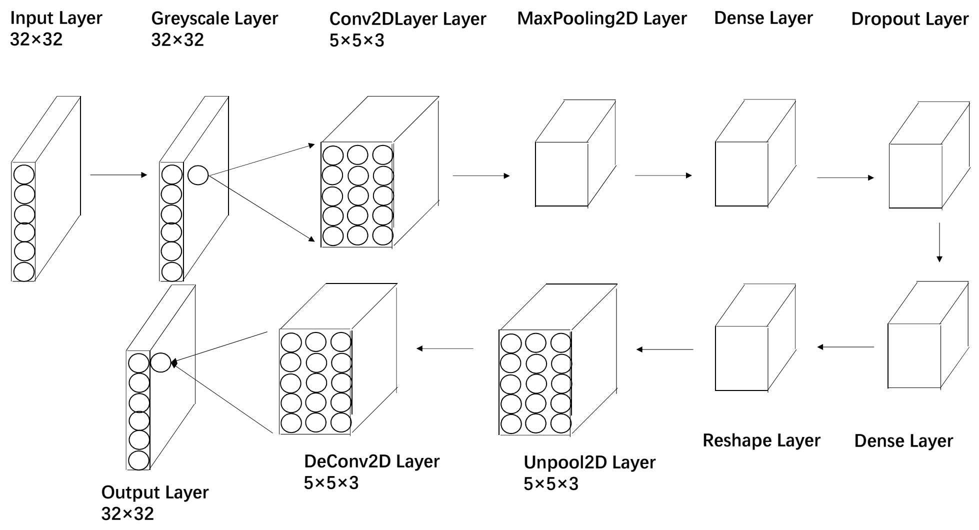

Figure 2: Our model

1. Input layer, use 4d-tensor to represent the input image.

2. Greyscale layer, calculates the greyscale of the input colored image.

3. Conv2D layer, in this layer we are using a filter of size 5*5*3(5 pixels width an height, and 3 because images have depth 3, the color channels). During the forward pass, we convolve each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. In this process, we will produce a 2-d activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual features, such as an edge of some orientation or a blotch of some color on the first layer, or eventually some wheel-like patterns on higher layers of the network. We will stack these activation maps along the depth dimension and produce the output volume.

4. MaxPool2DLayer, this layer's function is to progressively reduce the spatial size of the representation to reduce the amount if parameters and computation in the network, and hence to also control overfitting. This layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.

5. Dense layer, in this layer the neurons have full connections to all activations in the previous layer. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

6. Dropout layer, this layer is a regularizer that randomly sets input values to zero. It can be used to prevent overfitting.

7. A Dense layer and a Reshape layer, these two layers serve as preparation for the Unpool2D layer.

8. Unpool2D Layer, this layer performs unpooling over the last two dimensions of a 4D tensor. This layer is essential if we want to reconstruct the image.

9. Deconvolution layer. This layer is used for upsampling (backwards strided convolution). It can coarse outputs to dense pixels by interpolation. In-network upsampling is fast and effective for learning dense prediction.

10. Output layer, this layer reshapes the network and outputs the generated image.

- **Evaluating the Model:**

  1. **Given a model for moving from grayscale images to color images (whatever spaces you are mapping between), how can you evaluate how good your model is?**
     For the evaluation part, we used the squared loss as the criteria.

     $$\mathcal{L} = ||output - input||^2$$

     When the loss is small enough, we can know our model is performing good.

  2. **How can you assess the error of your model (hopefully in a way that can be learned from)?**
     - From the numerical perspective, the mean squared validation loss is, the better the model is performing.
     - From the perceptual perspective, the closer the color is to the original picture, the better the model is performing.

- **Training the Model:**

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

     It is actually quite easy to train our model because of the small size of the input. Generally, for each iteration, we would need to train for 40 seconds using a GTX 1070ti, for 25 seconds using a GTX 1080ti. We could achieve a good result in less than 100 iterations of training most of the time.

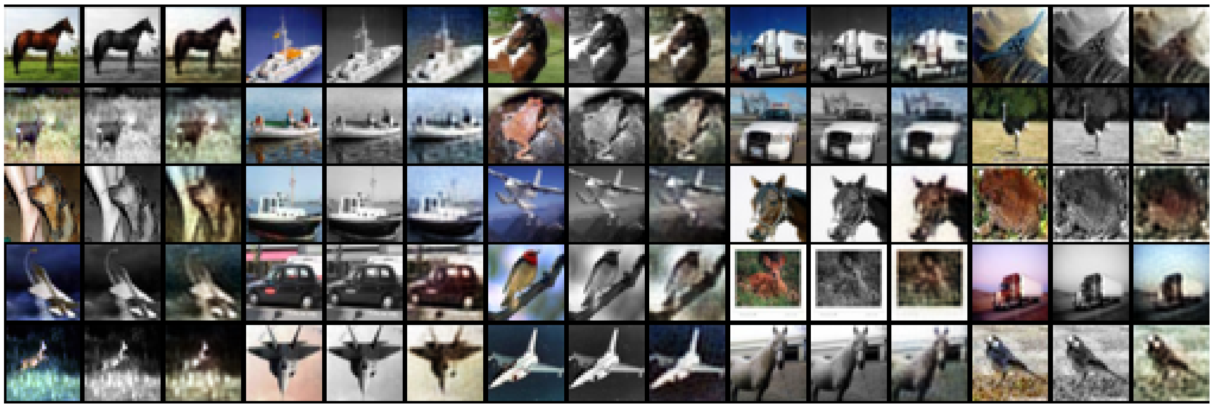

Figure 3: Some results

2. What algorithms did you draw on?

   We are using Stochastic Gradient Descent algorithm with Nestrerov momentum, and we are using back propagation to update the gradient.

3. How did you determine convergence?

   We validate the model and print out the validation loss every 10 iterations. When the validation loss stabilizes and even starts to rise, we stop our training and call it a convergence.

4. How did you avoid overfitting?

   We are using Max Pooling layers and Dropout Layers in our model to prevent overfitting.

   – Max pooling is basically selecting a subset of features, so we are less likely to be always training on false patterns.
   – Dropout layers randomly drop some components of our neural network with some probability, which actually creates a different network. Thus after we finished the training, we actually have an ensemble of models. The probability is typically 50%, but after our experiments, we found 30% is better.

- **Assessing the Final Project:**

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

     Our final model is satisfying as Fig 3 shows.

     We use our validation dataset to calculate the validation loss. The validation loss of our final model achieved 20.928495, which is reasonable to us.

  2. What is your program good at, and what could use improvement?
  3. Do your program's mistakes 'make sense'?

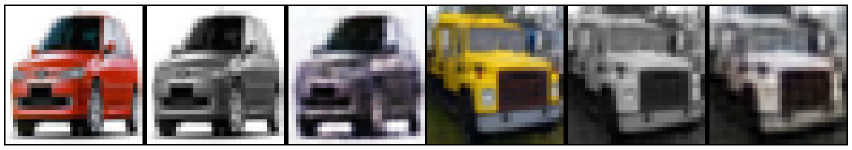

Figure 4: Some results

The colorizer can totally mess up blue and red or white and yellow as Fig 4 shows. But to be honest, we cannot tell the color of the car by just looking at its grayscale image. We think this kind of mistakes are quite resonable.

  4. What happens if you try to color images unlike anything the program was trained on?
  5. What kind of models and approaches, potential improvements and fixes, might you consider if you had more time and resources?

3 Our understanding towards CNN

Please refer to docs/XXXXX.html