Deep Colorization with CNNs

Shantanu Ghosh (4311-4360) Kalpak Seal (8241-7219)

1 Steps to reproduce the output:

Download the file from the link **UF OneDrive** (https://tinyurl.com/yyuxne2z) shared and unzip the file **DeepColor.zip** and execute the following command:
python3 main.py

A detailed description of the project is available at the **README.md** file. We have also included a **run_regressor.sh** file which will run the code in the hypergator.

2 Architecture

The **Colorizer** Network is defined in **Colorizer.py** which internally invokes the Regressor model written in **Regressor.py**, responsible for extracting the features from the batch of images.

ColorizerManager.py has the train and test method which is responsible for training and testing the models from the respective train and test datasets provided.

First, it receives the data in the defined batch-size from the **DataLoader** , retreives the hyperparameters from **Constants.py** and then trains the networks for the given number of epochs.

Once the network is trained, we test the model by loading the test data and subsequently store the outputs under the **outputs_sigmoid** and **outputs_tanh** folders, for the respective runs, as per requirement.

For the **Regressor**, there is a separate **Regressor_Manager.py** that trains and tests the model to generate the required performance and plots the Loss function over the epochs. The plot of the same is stored under **Plots** → **Regressor**.

3 Hyperparameters:

All the hyperparameters have been defined inside the **get_hyperparameters()** function within the **Utils** class in **utils.py** file. The parameters are defined as a dictionary and the function returns a list having the list of parameters respectively. Following this, during runtime, using **itertools.product**, we generate a cartesian product of the list of hyperparameters, the training has been done for each set of hyperparameters. If anyone wishes to add more hyperparameters, he / she can add it to **get_hyperparameters()** function within the **Utils** class in **utils.py** file. The hyperparameters, using which we have tested our code is given in table 1.

Parameter name	**Value**
Learning rate	[0.001, 0.0001, 0.1]
Epoch	[100, 200, 300]
Weight decay	[0, 1e-5, 1e-6]

Table 1: Hyperparameters (**extra credit**) for tanh and sigmoid activations

4 Best Hyperparameters

Our experiments shows that the best results were obtained from the following set of hyperparameters: **Learning rate: 0.001; Epoch: 100; Weight decay: 1e-5**.

5 Downsampling - Upsampling

We designed our model such that the size of the image will go down through the regressor network as follows - **128, 64, 32, 16, 8, 4, 2** and then will go up though the colorizer as follows - **2, 4, 8, 16, 32, 64 128**

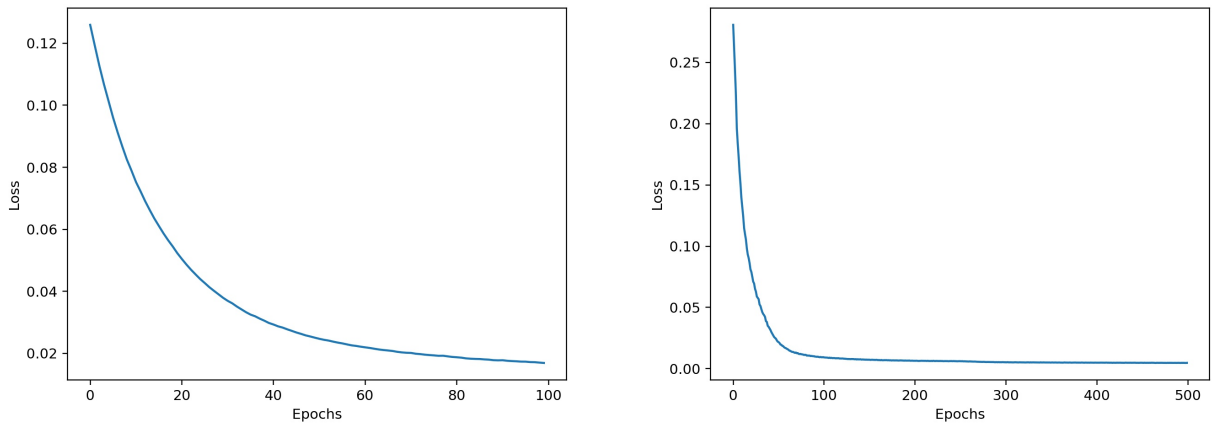

Figure 1: The plot of Colorizer(left) and Regressor(right) Network

6 GPU

The code extracts the device at the start of execution using **torch.cuda.is_available()** and loads the tensors accordingly, leveraging the compute based on availability.

7 Loss Function plot

The loss function for the colorizer and the regressor network is plotted in the figure 1.

8 Output

8.1 Regressor

The regressor outputs two scalar values which are the average of the a and b channel for each of the images in the console.

8.2 Colorizer

The colorizer output color images and stores the same in the respective **outputs_sigmoid** and **outputs_tanh** folder based on the activation function sigmoid and tanh respectively. The grayscale and colorized images are stored under the gray folder and color folder within **outputs_sigmoid** and **outputs_tanh** folders respectively.

9 Optional Credit - MSE of a channel and b channel

We varied the number of feature maps from 3 to 9, incremented by one at a time and tried to see the MSE between the true and predicted value of a and b channel and find out the optimal number of feature map as 3 and the MSE computed as **0.0002632503402417952**.

10 Optional Credit - tanh as activation function

For **normal credit**, we used sigmoid as the activation function for the final layer of the colorizer network and tanh for the **extra credit**. All the outputs for each run of each hyperparameter combination for **sigmoid** activation function is shown in **Figure 2 - 5**. The outputs with **tanh** activation function with best hyperparameter is shown in **Figure 6**.

Figure 2: Generated images for Run1(epoch: 100 learning rate: 0.001 weight decay: 0) with activation function as **sigmoid**. For every triplet, from left to right are gray scale, original and colored reconstructed images

Figure 3: Generated images for Run1(epoch: 100 learning rate: 0.001 weight decay: 0.000005) with activation function as **sigmoid**. For every triplet, from left to right are gray scale, original and colored reconstructed images

Figure 4: Generated images for Run1(epoch: 100 learning rate: 0.0001 weight decay: 0) with activation function as **sigmoid**. For every triplet, from left to right are gray scale, original and colored reconstructed images

Figure 5: Generated images for Run1(epoch: 100 learning rate: 0.0001 weight decay: 0.000005) with activation function as **sigmoid**. For every triplet, from left to right are gray scale, original and colored reconstructed images

Figure 6: Generated images for Run1(epoch: 100 learning rate: 0.001 weight decay: 0) with activation function as **tanh (extra credit)**. For every triplet, from left to right are gray scale, original and colored reconstructed images