CVPR
#*****

Generative Adversarial Network based Image Colorization

Shaochun Zheng Shuyuan Yang Minjie Mao Chengfan Li Zeyu Yang

1. Introduction

Image colorization is a challenging task that converts grayscale images to colorful images. It plays an important role in restoring historical photos and videos and helping artists create their masterworks. Image colorization makes it possible to generate a image that approaches the authentic one with limited or contaminated channel information. Since coloring images does not require labeled images, it can be used to preprocess image data in machine learning and any other fields of the image processing. Another application of image colorization technique is image compression. With the method to restore the authentic images, we can store only one channel of a colorful image, and other channels can be generated through it. Therefore, it will greatly decrease the storage space of photos and videos.

From the perspective of computer vision, the colorization problem is an image-to-image mapping problem. It is the process of predicting extra channels of the color space based on the input of an image with a single channel. The Y channel of the YUV color space is often used as the input, and the algorithm will generate the rest two channels. The basic idea is to use convolutional neural network to implement the colorization function. The network is able to present vivid results. However, the difficulty is that there are many possible coloring schemes for one image. For example, an apple can be colored either red or green, both of which are acceptable. Different colorization schemes contribute to uncertainties that would make the algorithm unreliable when restoring historical photos.

Recent learning-based implementation of image colorization includes Pixel Recursive Colorization [4], Colorization for Image Compression [2], Deep Colorization [3], and Colorization with Classification [5]. Deep Colorization [3] presents a fundamental idea of using convolutional neural network for colorization. The Pixel Recursive Colorization method constructs its model based on classic PixelCNN probabilistic model and makes some refinements to deal with high-resolution imagesprc.In the article Colorization for Image Compression [2], a multi-branch model is proposed with a characteristic loss function that allows one image to grow into differently colored images. Then the result image is obtained from another well-trained classification neural network. This method simulates different scenarios of coloring schemes and picks the best one. Colorization with Classification [5] applies the similar idea from object classification and colors the image based on its classification.

A survey [1] about image colorization summarizes a variety of outstanding works that have been done on image colorization, which gives us inspiration on developing our own learning model. The overview of our project is to use GAN to implement the image colorization. We use convolutional layers and fully connection layers to construct the generator in our model. A discriminator is designed for adversarial training to improve the performance of the generator.

2. Approach

As shown in Figure 1, the architecture of our model is composed of three parts: data preprocessing, which prepares training sets and validation sets; generator and discriminator pre-training, where the generator and the discriminator are trained separately; and adversarial training, which optimizes the generator based on the feedback of discriminator. Eventually, we send grayscale images as input to the generator and get colorized output images.

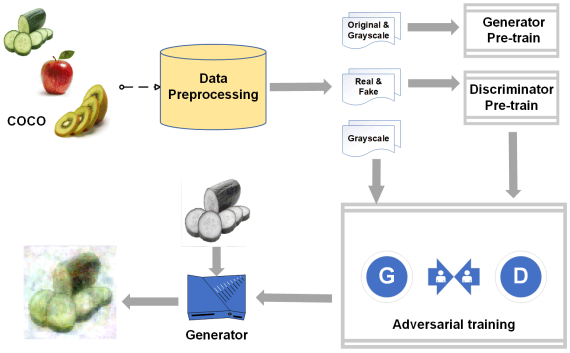

Figure 1. Model Overview

2.1. Data Preprocessing

Since images from the dataset are of different sizes and all colorful, we first resize the images to 256×256, a size

CVPR
#*****

friendly to deep learning models. For the generator pre-training, we convert the colorful images to grayscale images as input, and use the original images as ground-truth labels. For the discriminator pre-training, we label the original images as true and the randomly re-colorized images as false. We construct one data loaders for generator pre-training, discriminator pre-training, and adversarial training each. Furthermore, in order to avoid overfitting the training data, we augment the dataset by rotating and flipping the images.

2.2. Generator Pre-training

Our generator model takes grayscale images as input and generates images with RGB channels as output. In our model, the first two convolutional blocks extract features and downsamples image, followed by a fully connected block and am upsample layer, as shown in Figure Fig. 2 and Table Tab. 1. Each of the convolutional block consists of two convolutional layers and a non-linear activation layer (ReLU), and the fully connected block consists of three fully connected layers. the The mean squared error (MSE) between the generated image and the original image is used as the pre-training loss.

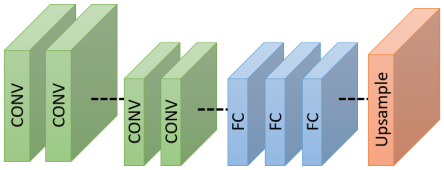

Figure 2. Generator Pipeline

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 6, 128, 128]	60
Conv2d-2	[-1, 6, 128, 128]	330
ReLU-3	[-1, 6, 128, 128]	0
Conv2d-4	[-1, 6, 64, 64]	330
Conv2d-5	[-1, 3, 64, 64]	165
ReLU-6	[-1, 3, 64, 64]	0
Flatten-7	[-1, 12288]	0
Linear-8	[-1, 1024]	12,583,936
Linear-9	[-1, 1024]	1,049,600
Linear-10	[-1, 12288]	12,595,200
Upsample-11	[-1, 3, 256, 256]	0

Total params: 26,229,621
Trainable params: 26,229,621
Non-trainable params: 0

Table 1. Generator Architecture

2.3. Discriminator Pre-training

To find the best model for discriminator, we test convolutional layers with different activation layers and various values for parameters such as learning rate, batch size, and weight decay. We also use lr_scheduler to reduce the learning rate based on number of epochs, which helps to approach results of higher accuracy and stability. As is shown in Figure Fig. 3 and Table Tab. 2 below, our discriminator model utilizes BatchNorm2d layers, which accelerates network training to the convolutional output before using LeakyReLU as the activation function. Afterwards, two fully-connect layers are used to generate a final output. The binary cross-entropy (BCE) loss is used as a metric for the discriminator.

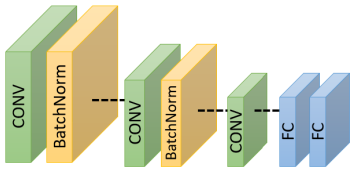

Figure 3. Discriminator Pipeline

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 128, 128]	3,072
BatchNorm2d-2	[-1, 64, 128, 128]	128
LeakyReLU-3	[-1, 64, 128, 128]	0
Conv2d-4	[-1, 128, 64, 64]	294,912
BatchNorm2d-5	[-1, 128, 64, 64]	256
LeakyReLU-6	[-1, 128, 64, 64]	0
Conv2d-7	[-1, 1, 64, 64]	1,152
Sigmoid-8	[-1, 1, 64, 64]	0
Linear-9	[-1, 1, 64, 1]	65
Linear-10	[-1, 1, 1, 1]	65

Total params: 299,650
Trainable params: 299,650
Non-trainable params: 0

Table 2. Discriminator Architecture

2.4. Adversarial Training

Next, we enter the adversarial training stage. The generator receives a greyscale image as input, and generates a colorized image, which is then handed over to the discriminator to determine whether the color of image is artificial or authentic.

Let $D(x)$ denote the output of the discriminator, the probability that the instance x is real. Let $G(z)$ denote the output of the generator with input z. The minimax loss is given by

CVPR
#*****

$$\text{Loss}_{\text{mm}} = \log[D(x)] + \log[1 - D(G(z))]$$

The goal of generator is to minimize the loss and the discriminator tries to maximize it. This loss function better considers the consequences of the adversarial results between the two, and dynamically links them.

The loss is then back-propagated to update the parameters of generator and discriminator. Note that since the generator's output is connected to the discriminator's input at this stage, the discriminator parameters should not be updated when back-propagating the generator loss but should be updated when back-propagating the discriminator loss. Repeat this process, training both alternately until convergence.

3. Experiment

3.1. Dataset

Our dataset is Fruits and Vegetables Image Recognition Dataset from Kaggle [6]. The dataset contains 4,3200 categorized images of fruits and vegetables. Moreover, the images are divided into training, validation, and test parts for the convenience of training and testing.

Considering the nature of the colorization task, we choose images of fruits and vegetables whose color is consistent and distinct. Therefore, we choose four representative categories, apple, cucumber, kiwi, and orange. The number of images in each category we pick are the same.

In the generator pre-train process, grayscale images are set as input and the resized images with original color are set as labels. While in the discriminator pre-train process, images with size $3 \times 256 \times 256$ are set as input and whether the image is true-colored or fake-colored as label $\{1, 0\}$. The ratio of train and validation datasets is 10:1 for both generator pre-train process and discriminator pre-train process.

3.2. Hyperparameter tuning

Due to the complex architecture and training process, GAN has a large number of hyperparameters that can be tuned.

Name	value	Description
gpre_epoch	20	generator pre-train epochs
dpre_epoch	40	discriminator pre-train epochs
pre_batch	64	pre-train batch size
s_n	256	pre-train input image size
optim	adam	optimizer
pre_lr	0.001	pretrain learning rate
pre_wd	0.00001	pretrain weight decay
epoch	5	final training epochs

Moreover, we use mean square loss for pre-training Generator and pre-training Discriminator and minimax loss for adversarial training stage.

3.3. Pre-train Results

Figure Fig. 4 shows the pre-train result of generator. We train the generator for 20 epochs to make sure that the generator can generate some basic images instead of randomly generating noise. From the graph we can observe that the loss keeps decreasing on both training set and validation set.

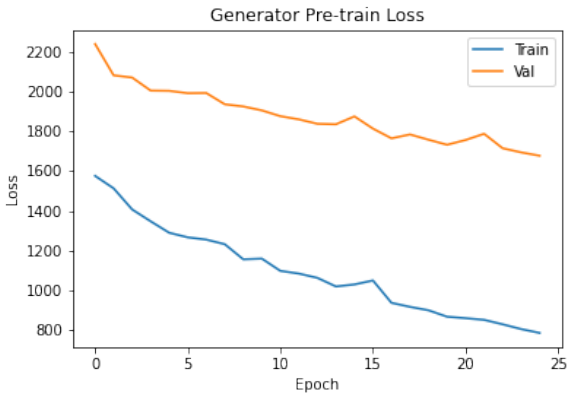

Figure 4. Generator Pre-train Performance

Figure 5 shows the pre-train accuracy of discriminator. We can observe that the discriminator reaches a good accuracy on an early stage.

From Figure 5, although the trend of validation and train accuracy make sense, it is weird that validation accuracy stays higher than training accuracy. This may be due to the small sample size of the validation set, which contains 1000 images, while the train set contains 10000 images. Additionally, how we generate images with fake color for discriminator pre-training may also cause a problem, which is discussed in section 3.5.

3.4. Adversarial Training Result

Then, we enter the adversarial training stage. In this stage, we pass a batch of images to the generator to generate new images with color, and then use the discriminator to identify whether the generated images are real. After calculating the loss generated by the current batch, back-propagation is used to update the parameters. The colorized images are regenerated using the finally obtained generator.

After another 5 epochs of adversarial training, we found that the performance of the generator were not getting better, but were getting worse, so we early stopped the GAN. Figure 6 shows the final colorization result. We can observe that although our network does not perfectly colorize the image, we can still recognize items and their approximate colors.

CVPR
#*****

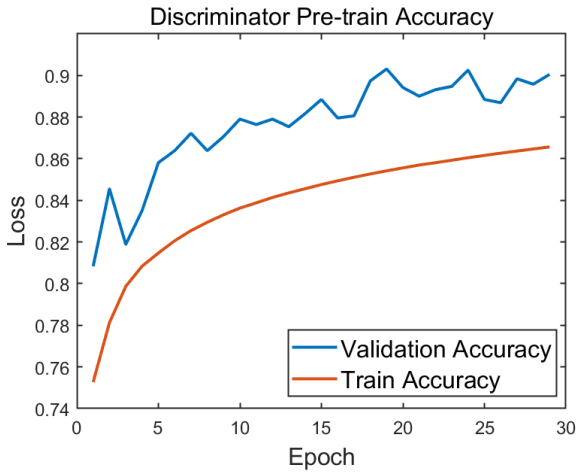

Figure 5. Discriminator Pre-train Performance

Figure 6. Example Results
(Left: input, C: output, R: original)

3.5. Discussion

During the experiment, we found that after the adversarial training, the loss increased slightly and the model early stopped in epoch 5. There are several reasons accounting for this phenomenon.

First of all, When training the discriminator, it is difficult for us to find image datasets with fake colors, so we generate image datasets with fake colors by randomly modifying the original image channels. However, this method is flawed since the resulting images are messy and do not match the generator mechanism, which will lead to poor adversarial training. We have tried to build a dataset of images with fake colors by ourselves, but obviously, we cannot use our own generator to build such a dataset to train the discriminator, we need to construct a new generator to build it, but due to time constraints we did not did not accomplish this goal.

Moreover, the time left for our adversarial training is too limited. In the stage of pre-training the generator and discriminator, the reading of the paper, the construction of the model, and the trial and comparison of different methods is a long process that consumes a lot of time and effort. If there is further opportunity, we will complete the final part.

4. Implementation

In this project, we utilize the idea of GAN to construct and train our model, which is inspired by the work of Saeed Anwar et al [1]. Our generator structure is adapted from Deep colorization proposed in 2016 [3]. And the discriminator is built entirely by ourselves. Moreover, we implement a series of data collection, classification, cleaning work.

In the process of model implementation, we tried many different methods, including but not limited to Instance aware image colorization proposed in 2020 [7] , Let there be color proposed in 2014 [8].

References

[1] Saeed Anwar, Muhammad Tahir, Chongyi Li, Ajmal Mian, Fahad Shahbaz Khan, and Abdul Wahab Muzaffar. Image colorization: A survey and dataset. *CoRR*, abs/2008.10774, 2020.

[2] Mohammad Haris Baig and Lorenzo Torresani. Colorization for image compression. 2016.

[3] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. Deep colorization. *CoRR*, abs/1605.00075, 2016.

[4] Sergio Guadarrama, Ryan Dahl, David Bieber, Mohammad Norouzi, Jonathon Shlens, and Kevin Murphy. Pixcolor: Pixel recursive colorization. 2017.

[5] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)*, 35(4):110:1–110:11, 2016.

[6] Kritik Seth. Fruits and vegetables image recognition dataset. https://www.kaggle.com/kritikseth/fruit-and-vegetable-image-recognition. Accessed: 2022-04-20.

[7] Jheng-Wei Su, Hung-Kuo Chu, and Jia-Bin Huang. Instance-aware image colorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7968–7977, 2020.

[8] Michael Waechter, Nils Moehrle, and Michael Goesele. Let there be color! large-scale texturing of 3d reconstructions. In *European conference on computer vision*, pages 836–850. Springer, 2014.