CSL4020: Deep Learning
Project: Sketch and GrayScale Image Colorization
Instructor: Dr. Mayank Vatsa

Ajay Nirmal(B20AI002), Tushar Kumar(B20AI048), Saksham Singh(B20AI064)

September 13, 2023

1 Introduction

Colorizing black-and-white photos is one of the most fascinating uses of deep learning. Several years ago, this task required a lot of human input and hard coding, but now the entire process can be done end-to-end using the power of AI and deep learning. Generally, you need a large amount of data or long training times to train the model from scratch for this task, but we have tried to work on a subset of the big datasets to decrease the training time and try different model architectures, loss functions, and training strategies. Eventually, we developed an efficient strategy to train such a model, using the latest advances in deep learning, on a relatively small dataset and with extremely short training times.

2 Motivation

To train the models, people utilize L*a*b color space rather than RGB in all papers read and every code looked at by us on colorization on GitHub. There are some reasons behind this decision. To train a colorization model, we should feed it a grayscale image and expect that it will colorize it. When utilizing L*a*b, we may feed the model the L channel (which is the grayscale image) and ask it to forecast the other two channels (*a, *b), and then concatenate all the channels to produce our colorful image. However, if you use RGB, you must first convert the image to grayscale, then feed the grayscale image to the model and hope it predicts three numbers for you, which is a far more difficult and unstable process due to the many more possible combinations of three numbers than two numbers. Figure 1 shows a comparison between RGB vs L*a*b.

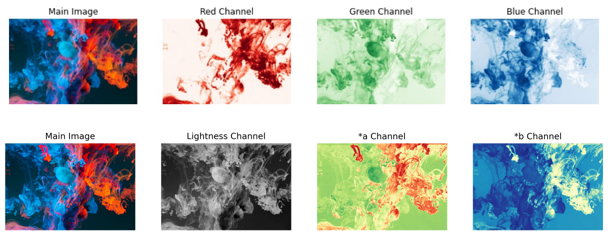

Figure 1: RGB vs L*a*b

3 Literature Review and Solution

Many deep learning-based colorization techniques have been developed in recent years. The Colourful Image Colorization [1] paper approached the problem as a classification task, and they also considered the problem's uncertainty. For eg. an image can take different and valid colors, but being certain about any color about it is an issue. However, another paper approached the problem as a regression task. Each approach has advantages and disadvantages, but in this project, we will take a different approach.

The study Image-to-Image Translation with Conditional Adversarial Networks [2], often known as pix2pix, provided a general solution to numerous image-to-image tasks in deep learning, one of which is colorization. This method employs two losses: an L1 loss, which turns the task into a regression task, and an adversarial (GAN) loss, which aids in solving the problem unsupervised by assigning the outputs a value indicating how "real" they appear. We have followed a similar approach where we use conditional GAN and an extra loss function L1 loss with the initial loss function used in the paper to perform the greyscale image colorization.

4 Methodology

Below shown are the steps used to perform greyscale colorization highlighting the key components.

- Formation of dataset and dataloaders
- Model initialization
- Defining the custom loss function
- Training the model and testing the model on random images apart from the dataset

4.1 Dataset used

COCO dataset [3] is a large dataset that consists of more than 300K labeled images of objects and people. We have used a smaller subset of this dataset using the Fast AI library. Figure 2 shows some sample images of the dataset used.

Data distribution in the dataset used created is as follows:

- **Trainset**: 8000
- **Testset**: 2000

Figure 2: Sample images from the dataset

4.2 Model used: U-Net and conditional GAN

The GAN model used by us in the project makes use of U-Net(fully convolutional network) as the generator and the discriminator. Figure 3 shows the architecture of U-Net. The U-Net architecture consists of two major components: an encoder and a decoder. The encoder is composed of a sequence of convolutional and max-pooling layers that degrade the spatial resolution of the input image while increasing the number of feature mappings helping in the extraction of high-level features from the image. In contrast, the decoder is a series of upsampling and convolutional layers that steadily raise the spatial resolution of the feature maps while decreasing the number of channels. The skip connections between the encoder and decoder layers assist in spatial information preservation and enable the network to retrieve fine-grained details.

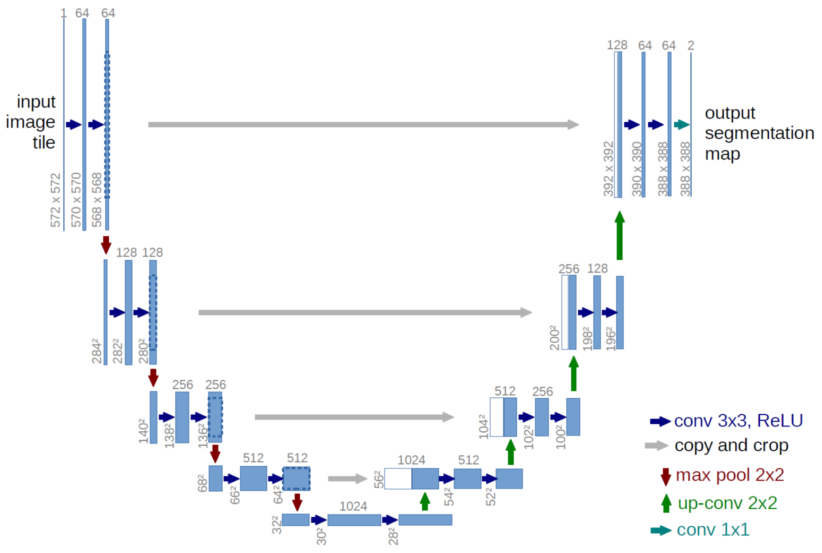

Figure 3: U-Net architecture

Further, the discriminator model gives one number as output for each patch of the input image and assesses if it is fake or not independently. Using such a model for colorization is logical to us because the local modifications that the model must make are critical, maybe deciding on the entire image. The model's output shape is 30 by 30.

4.3 Custom loss function

The generator and discriminator use two different loss functions. Here the GAN_loss is referred to as BCEWithLogitsLoss loss. We use a combination of two loss functions in the generator model present in figure 4 where λ is a coefficient for balancing the contribution of the two losses.

- **discriminator**: GAN_loss (0.5*(real +fake))
- **generator**: GAN_loss + λL1 loss

If the L1 loss is used alone, in the process of learning to colorize images, it will be conservative and will most of the time use colors like "grey" or "brown" because, in times of uncertainty(which color is best), it takes the average and uses these colors to reduce the L1 loss as much as possible (similar to the blurring effect of L1 or L2 loss).

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

Figure 4: Custom loss function

5 Results

5.1 Hyperparemeters used for training

- Learning rate: 0.0001
- Batch size: 32
- Number of epochs: 50

5.2 Evaluation using standard metrics

The loss values for both the generator and discriminator have been mentioned in Table 1 in the form of the table. The metrics clearly depict the good performance of the model. Further, the loss graphs for both have shown in figure 5

Losses	loss_D_fake	loss_D_real	Discriminator_Loss	loss_G_GAN	loss_G_L1	Generator_loss
Values	0.548	0.544	0.551	1.165	6.582	7.748

Table 1: Results on train data

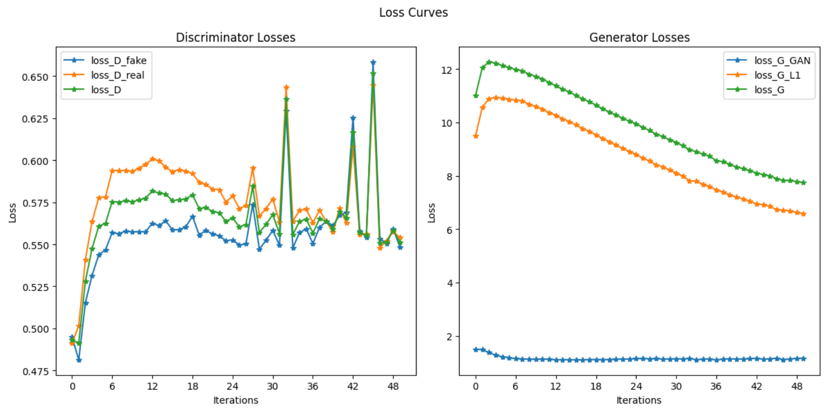

Figure 5: Discriminator and Generator loss curves

6 Conclusion

The task for generating the color images from the greyscale images and sketches has been successfully achieved, a sample of which can be seen in figure6 on random images apart from the dataset. The key DL part in the project used is the combined use of a generator and discriminator together to colorize the images using custom loss functions for each. Also, here is the link to our Kaggle repository for the project. [4]

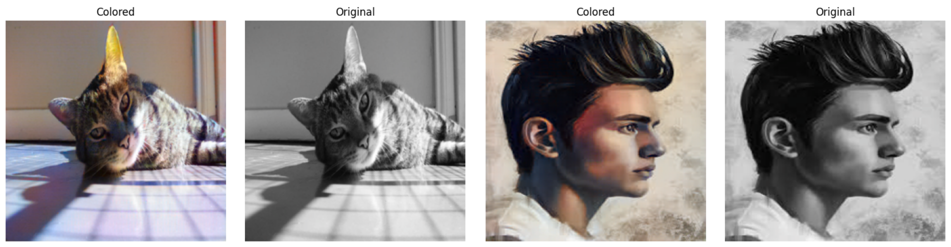

Figure 6: Test results showing Greyscale image and sketch colorization

References

[1] Colorful Image Colorization — Richard Zhang, Phillip Isola, Alexei A. Efros

[2] Image-to-Image Translation with Conditional Adversarial Networks — Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros

[3] COCO dataset

[4] Kaggle repository