Comparing Regression, Autoencoder and Generative Adversarial Network (GAN) for Image Colorization

Vikram Bharadwaj, Sudhanva Narayana
CS6140 Machine Learning
Professor: Prof. Ehsan Elhamifar

May 5, 2022

Abstract

Traditional image colorization methods are based on manual colorization methods such as Photoshop or GIMP. However, these methods are not robust to the changes in the lighting conditions of the scene. Recent advances in deep learning have made it possible to learn a colorization model from grayscale images, where the input for the networks are a set of grayscale images and the output the network learns is the colorized image. In this work, we present three methods for colorization: regression, autoencoder and generative adversarial network (GAN).

1 Introduction

Every color image is a combination of 3 channels, namely: R, G, and B. Another way used to represent images is called the L, *a, *b (CIELAB) colorspace. The idea is to use grayscale images for complex image processing. The goal of this study is to develop a method for colorizing grayscale images using the three aforementioned methods:

- **Convolutional Neural Network + Regression:** Loss between actual values of *a and *b channels and predicted *a and *b channels using vanilla CNN, followed by regression.

- **Convnet Autoencoder:** Uses CNN in an encoder-decoder fashion, where the encoder represents the grayscale image as a latent representation, which the decoder then converts to a 3-channel color image.

- **GAN:** Uses a custom minimax log-loss function with a generator and a discriminator. The generator takes a grayscale image as an input, which it tries to convert to a 3-channel color image. The discriminator tries to tell 2 sets of images apart, i.e., (grayscale, original-color) and (grayscale, generator-color-image).

2 Technical Details

This task can be formulated as a problem of data imputation in which the color values of ***a** and ***b** values are "hallucinated". The mapping from a grayscale channel to color channels is a non-deterministic problem. The range of values that each grayscale pixel can predict is exponentially large and the mapping is non-linear.

2.1 Method 1: Regression

Several convolutional layers extract semantic information from the input images and then apply deconvolutional layers to upscale the extracted information. Here, we are using pure upsample, where there are no parameters to be learned. Specifically, the beginning of our model is a ResNet-18, an image classification network with 18 layers and residual connections. Modify the first layer of the network to accept grayscale input images and cut it off after the 6th set of layers. It predicts a two-element vector (AB channel) for each pixel of the image at the end of the network, as shown in the figure below. We want to minimize the Euclidean error between the AB channel we estimate and the ground truth.

However, this loss function is problematic for colorization due to the multimodality of the problem since there may be multiple plausible solutions. As a result, our model will usually choose desaturated colors which are less likely to be correct, and the model prefers more desaturated colors than running the risk of choosing colors that are too bright.

* All layers are two-dimensional convolutional layers.

Input Size	Output Size	Kernel Size	Function	Upsample
512	128	3	ReLU	Scale Factor = 2
128	64	3	ReLU	-
64	64	3	ReLU	-
64	32	3	ReLU	Scale Factor = 2
32	2	3	ReLU	Scale Factor = 2

2.1.1 Loss Function: Regression

The loss function for this problem is the Euclidean distance between the predicted AB channel and the ground truth and can be written as:

$$\ell(x, y) = \sum_{i=1}^{n} (x_{AB} - y_{AB})^2 \tag{1}$$

2.1.2 Architecture: Regression

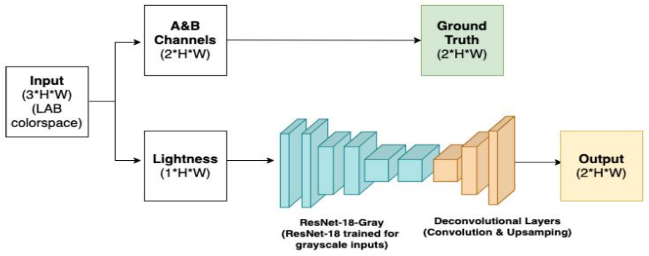
Input (3*H*W) (LAB colorspace)
A&B Channels (2*H*W)
Ground Truth (2*H*W)
Lightness (1*H*W)
ResNet-18-Gray (ResNet-18 trained for grayscale inputs)
Deconvolutional Layers (Convolution & Upsamping)
Output (2*H*W)

Figure 1: CNN + Regression architecture

2.2 Method 2: Autoencoder

In this method, we use a convolutional autoencoder to learn a mapping from grayscale to color. The encoder is a convolutional neural network, which reduces the given grayscale image to a dense latent representation, from which the decoder reconstructs the AB channels of the image. Therefore, in this method, we have two CNNs, one for the encoder and one for the decoder. The upsampling layers in the decoder are transposed convolutional layers, where there are a lot of parameters to be learned by the model while reconstructing the AB channels of the image. The architecture of the entire network is as shown in the figure below.

One approach is to make two copies of the image, one to be a grayscale image and it will act as your input to the encoder which is responsible for extracting the features of the image "Latent Space Representations" that can be used as input for the decoder to reconstruct the image, the other copy will be the same image but colorized as your target to the decoder (Supervised Learning) so that it can minimize the error between the original colored image and the generated one.

2.2.1 Loss Function: Autoencoder

The loss function for this method is the Euclidean distance between the predicted AB channel and the ground truth and is the same as (1).

2.2.2 Architecture: Autoencoder

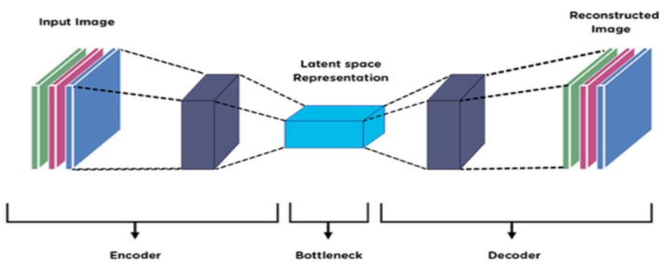

Figure 2: Autoencoder architecture

2.2.3 Encoder

* All layers are two-dimensional convolutional layers.

Input Size	Output Size	Kernel Size	Function
1	32	3	ReLU
32	64	3	ReLU
64	128	3	ReLU
128	256	3	ReLU

2.2.4 Fusion

* All layers are linear layers.

Input Size	Output Size
256 * 14 * 14	56
256	256 * 14 * 14

2.2.5 Decoder

* All layers here are two dimensional transposed convolutional layers.

Input Size	Output Size	Kernel Size	Function	Upsample
256	256	3	ReLU	ConvTranspose2D
256	128	3	ReLU	ConvTranspose2D
128	64	3	ReLU	ConvTranspose2D
64	2	3	ReLU	ConvTranspose2D

Final activation function is TanH [-1, 1] for the output of the decoder.

2.3 Method 3: Generative Adversarial Network

Unlike the previous two models, this model contains two neural networks: a generator and a discriminator. The generator works like previous models, takes in a black-white image and outputs the predicted colored image. The discriminator is trained on two sets of images, to classify the predicted color image as false and the ground-truth color image as true. The pipeline of the model on a black-white image and the corresponding colored image is shown in the figure below.

In terms of loss function, there are two main loss functions applied here. First, the loss of the discriminator is computed as the average of its loss on a true image and its loss on a fake image. Each loss is a MSE loss for each pixel on the image (labeled 0 or 1). This loss could effectively capture whether this discriminator could distinguish between ground-truth image and generated image. Second, the loss of the generator is λ times L1 loss between predicted image and true image, plus discriminator loss on predicted image with label true. This loss could make the generator generate images that have less difference with true image and less loss from the discriminator.

2.4 Loss Function

The loss function in a GAN is the following:

$$\ell(x,y) = \frac{1}{2}\left(\sum_{i=1}^{n}(x_{AB}-y_{AB})^2 + \lambda\sum_{i=1}^{n}(x_{AB}-y_{AB})^2\right) \tag{2}$$

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[1-\log D(G(z))] \tag{3}$$

2.4.1 Architecture: GAN

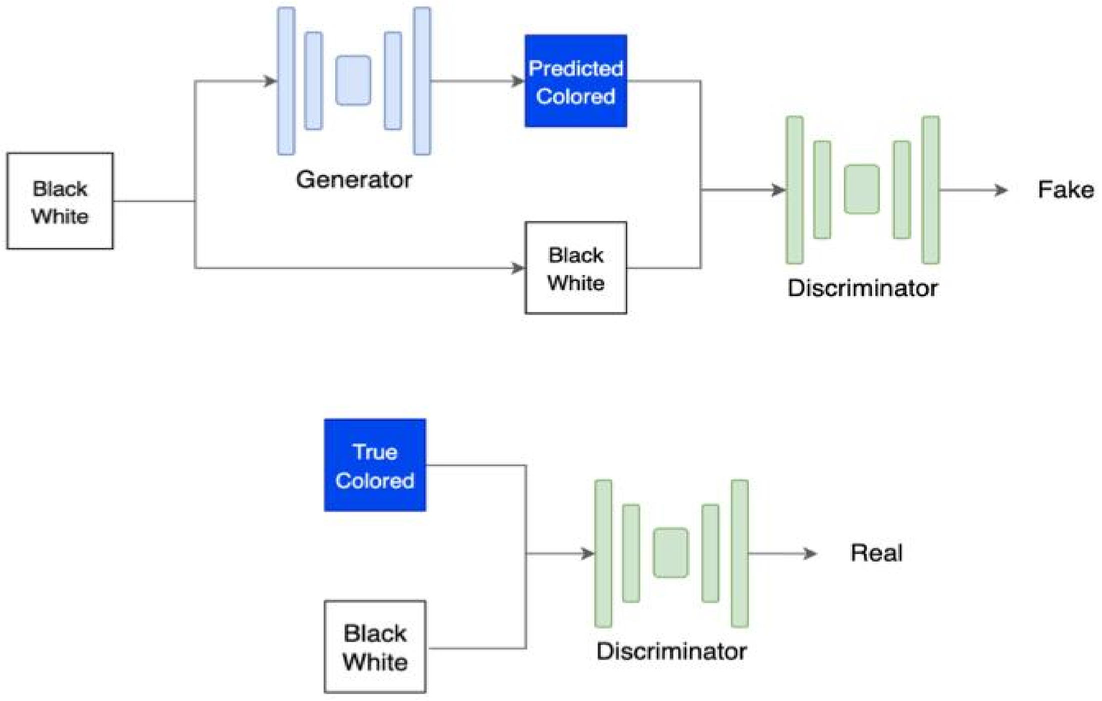

Figure 3: Autoencoder architecture

3 Experimental Results

3.1 Dataset

Our training, validation and test datasets are picked from the validation set pof MIT's Places 365 [7] . The dataset is from the CSAILVision Lab and can be accessed at places2.csail.mit.edu. All the images are in RGB format, and are 256x256 pixels in size. All channel values are in the range [0, 255]. Since there are more than 30,000 images, due to time and compute constraints, we only use a subset of the dataset of around 2,500 images for training and an additional 500 images for validation.

3.2 Preprocessing

To reduce the complexity of the entire process, we use grayscale images by first converting the RGB imagesto the L*a*b color space. The L channel is the grayscale image, and the a and b channels are the color channels. Our models are now trained on this L channel to predict the values of the a and b channels.

3.2.1 Pre-processing for Regression and Autoencoder

- All images are first converted to L*a*b channel image from RGB channel.
- The L channel first extracted from the L*a*b channel image and this serves as the input to the regression model.
- The *a and *b channels are used as the tragets for the regression model.
- The images are then resized to a fixed size of 224x224.
- The images are then normalized in the range of [-1, 1] for *a and *b channels.
- The training set is then used to train the model.

3.2.2 Pre-processing for Convnet based encoder-decoder and GAN

Here, we will use the same pre-processing steps as for the regression method. The only difference is that we will use a CNN in an encoder-decoder fashion for the second method and a GAN for the third, as shown below:

3.3 Implementation

We have used ResNet-18 as the encoder and decoder for the first two methods. The encoder is trained on the training set and the decoder is trained on the validation set. The encoder is used to generate the latent space vector z. The decoder is used to generate the predicted image from the latent space vector z. The generator is used to generate the predicted image from the latent space vector z.

3.4 Results

Methods	**MSE Loss**
Regression	col2
Autoencoder	cell2
GAN	cell6

3.4.1 Regression Loss Plot

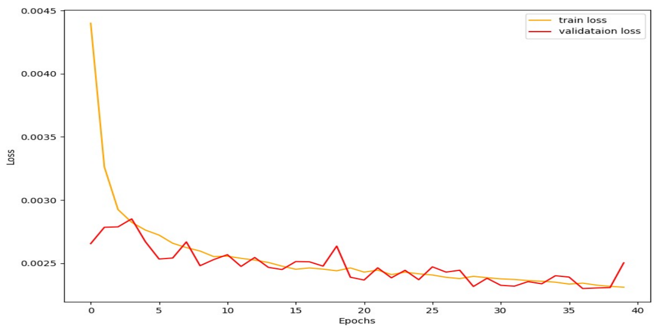

Figure 4: Caption

3.4.2 Autoencoder Loss Plot

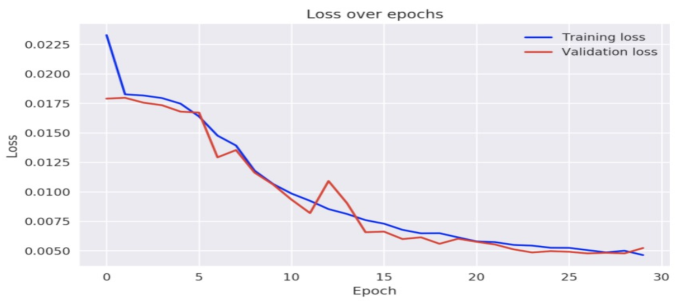

Figure 5: Caption

3.4.3 GAN Loss Plot

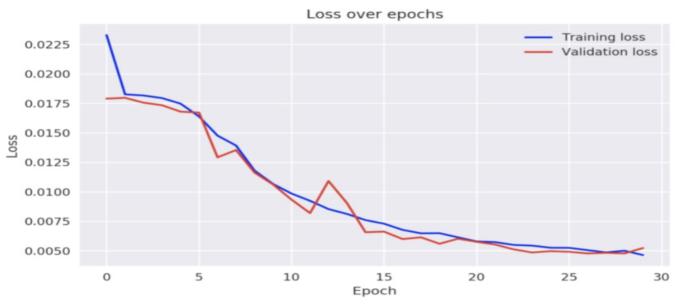

Figure 6: Caption

4 Limitations and Further Work

The idea is to use grayscale images for complex image processing. The goal of this study is to develop a method for colorizing grayscale images using Regression and compare how the model performs against the use of an autoencoder and a GAN.

5 Contribution

The code and instructions to use or run it are available at github.com/nsudhanva/image-colorization. The project was built with both Vikram and Sudhanva as collaborators in the same repository with each of them being collaborative on code, reports and training.

Sudhanva Narayana implemented:

- GAN architecture, code, tests and experiments.
- Trained the models on the dataset on AWS/Colab/GPU.
- Model inference and reporting.

Vikram Bharadwaj implemented:

- Regression architecture, code, tests and experiments.
- Autoencoder architecture, code, tests and experiments.
- Model tuning and reporting.

[1] [2] [3] [5] [6] [7] [4]

References

[1] Federico Baldassarre, Diego González Morín, and Lucas Rodés-Guirao. Deep koalarization: Image colorization using cnns and inception-resnet-v2. *CoRR*, abs/1712.03400, 2017.

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018.

[3] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. *CoRR*, abs/1603.06668, 2016.

[4] Cuong V. Nguyen, Tien-Dung Cao, Tram Truong Huu, Khanh N. Pham, and Binh T. Nguyen. An empirical study on gans with margin cosine loss and relativistic discriminator. *CoRR*, abs/2110.11293, 2021.

[5] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. *CoRR*, abs/1604.08610, 2016.

[6] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. *CoRR*, abs/1603.08511, 2016.

[7] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.