
Image Colorization using Deep Convolutional Networks and Generative Adversarial Networks

Anirudh Mukundan Raghavan*
Department of Computer Science
University of Florida
araghavan@ufl.edu

Prajwal Dondiganahalli Prakash†
Department of Computer Science
University of Florida
prajwal.dondigan@ufl.edu

Abstract

In this project, we address the problem of automatically generating a plausible colored photograph given a grayscale image using Convolutional Neural Networks. We explore two methods - conventional supervised machine learning and generative modeling using Generative Adversarial Networks. In the first method, we frame the problem as an optimization problem to generate a mapping between the pixels in the input grayscale scale and the target color image. In the second method, we use a generator-discriminator model pair that is adversarially trained so that the generator produces outputs indistinguishable from the actual color images. We explore several architectures/models and loss functions to obtain the most aesthetically pleasing colored images. We also discuss state-of-the-art enhancements to our two models that could improve their performances. We provide a quantitative analysis of our results and discuss areas for further research.

1 Introduction

1.1 Background

Image colorization is a fascinating topic to study in deep learning. It involves computer vision techniques, such as image classification, and image segmentation working together. Traditionally, to colorize an image, a model must first classify the various objects present in the image. Next, the model must locate the object in the image. Last, the model must apply the correct color to all the objects present in the image. One might think that there are too many choices for color, a shirt may be of any color or shade in the RGB spectrum, a wall may be painted in any color and a car may have any paint job. Thus, our problem is multimodal, where we may have multiple valid predictions for a single image. This is partially true - while there exist many choices for common objects, most objects have a specific choice of colors. For example, the sky is either blue or black depending on the time of day, the sun is always yellow, the moon is always white, etc. A common problem faced while training models is the lack of data. This is not a problem for this task, as we can simply convert existing images to grayscale and pair them with their color versions to train the network. Thus, predicting color is more tractable than we think, especially using Deep Learning.

Two popular methods exist to colorize grayscale images using machine learning – user-guided, and automatic colorization. In the first method, a human provides additional semantic input, usually by drawing strokes over the grayscale image, providing hints to the model. This method has historically provided accurate results but requires intensive user interaction. The second model works either by matching a picture to its colored version and learning parametric mappings from grayscale to color from large-scale image data. This is the method we will be using to build our model.

*equal contribution

†equal contribution.

We will use a Convolutional Neural Network and a Generative Adversarial Network to colorize images and then compare the results from both networks.

1.2 Problem Statement

Image colorization is a translation problem, where we try to map the input image to an output image, where the images are represented as high dimensional arrays of numbers. It can be seen as a regression problem on individual pixels, where we try to predict the values of each pixel in the output based on the pixel value in the input. Our network will try to generate an output but with different values for the pixels.

Assuming each input image \mathbf{X} has the dimensions $h * w$, and is encoded in a 3-channel colorspace (e.g. RGB colorspace), then $\mathbf{X} \in \mathbb{R}^{h*w*3}$. $\Phi \in \mathbb{R}^{H*W*3}$ is a pixel-to-pixel transformation (e.g. normalization) on \mathbf{X} . $\mathbf{Y} \in \mathbb{R}^{h*w*3}$ is the target output image in the dataset and $\hat{\mathbf{Y}} \in \mathbb{R}^{h*w*3}$ is the image generated from our model.

$$\hat{\mathbf{Y}} = f(\Phi) \quad (1)$$

$$\text{minimise } L(\mathbf{Y}, \hat{\mathbf{Y}}) = \sum_{h,w} l(y_{h,w}, \hat{y}_{h,w}) \quad (2)$$

Our primary objective is then defined by equation 2, where l is a function that defines the distance between two pixel values.

Since there are very few constraints for the values the pixels can take, there are no right or wrong values for the pixels. Each image can be colored in multiple ways. This leads to an interesting challenge, where we experiment with different loss functions and network architectures to see which one gives us aesthetically pleasing images.

1.3 Algorithms

1.3.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a neural network, i.e., it is made up of layers of perceptrons that are connected to each other through connections with learnable weights. Each of these perceptrons receive an input and computes a dot product with its weights. Optionally, it may apply a non-linear function to the output. The difference in a CNN is that the layers typically are built to deal with high-dimensional data, such as images, where filters need to be learned to extract features from specific parts of the high-dimensional data. They are composed of the following layers:

1.3.2 Convolutional Layer

A convolutional layer

- Accepts a volume of size $\mathbf{W}_1 * \mathbf{H}_1 * \mathbf{D}_1$
- Requires four hyperparameters: number of filters K , their spatial extent F , the stride S , the amount of zero padding P .
- Produces a volume of size $\mathbf{W}_2 * \mathbf{H}_2 * \mathbf{D}_2$ where:

$$\begin{aligned} \mathbf{W}_2 &= \frac{\mathbf{W}_1 - F + 2P}{S + 1} \\ \mathbf{H}_2 &= \frac{\mathbf{H}_1 - F + 2P}{S + 1} \\ \mathbf{D}_2 &= K \end{aligned}$$

In the output volume, the d -th depth slice (of size $\mathbf{W}_2 * \mathbf{H}_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by d -th bias.

1.3.3 Pooling Layer

The function of the pooling layer is to reduce the size of the output of a convolutional layer, to reduce the number of parameters in the network. This reduces the amount of computation required to train the network. It also helps in controlling overfitting. The most common pooling layer is with filters of size 2×2 .

- Accept a volume of size $\mathbf{W}_1 * \mathbf{H}_1 * \mathbf{D}_1$
- Requires two hyperparameters: their spatial extent F , the stride S
- Produces a volume of size $\mathbf{W}_2 * \mathbf{H}_2 * \mathbf{D}_2$.

$$\begin{aligned}\mathbf{W}_2 &= \frac{\mathbf{W}_1 - F}{S + 1} \\ \mathbf{H}_2 &= \frac{\mathbf{H}_1 - F}{S + 1} \\ \mathbf{D}_2 &= \mathbf{D}_1\end{aligned}$$

1.3.4 Fully Connected Layer

The neurons in this layer are connected to all the activations of the previous layer i.e. they maintain the shape of the previous layer. These layers are similar to the ones used in ordinary neural networks. They accept an input x , compute a matrix multiplication with their weights, and add a bias. Additionally, they can also apply a non-linearity such as ReLU, sigmoid to the output.

1.3.5 Generative Adversarial Networks

Generative models are unsupervised learning models that model the patterns and irregularities in the input data and can generate new examples that are indistinguishable from the ones in the input data. A Generative Adversarial Network (GAN) is a method to train such generative models by pairing them with a discriminative model. Hence, a GAN has two sub-models, a generator model that generates new examples, and a discriminator model that classifies whether the examples are real (from the input data) or generated by the generator model. The two sub-models are trained together in a zero-sum game until the generator model can fool the discriminator model about half the time, implying that the generator model generates examples that are almost indistinguishable from the (real) ones in the input data. The term adversarial implies that the model rewards conflict between the generator model and the discriminator model. There are many applications of such generator-discriminator networks, including generating photorealistic images of people, objects, or scenes.

2 Dataset

We used the CIFAR-10 dataset to train both our models. The images in this dataset have the dimensions 32×32 and is encoded in the RGB colorspace. While these dimensions do not translate to a model that could be used to colorize real-world images, the small dimensions allowed us to deal with memory and computation constraints. We did not focus on a specific category and instead trained our models on all the 50,000 images in the training set to build a more generalized model.

2.1 Preprocessing

The $32 \times 32 \times 3$ images in the RGB colorspace were converted into the CIE L^*a^*b colorspace. The l (luminance) channel is sufficient to represent the grayscale image and the all the color information is encoded into the a and b channels. Thus we could generate a dataset with input dimensions $32 \times 32 \times 1$, and output dimensions $32 \times 32 \times 2$ instead of dealing with $32 \times 32 \times 3$ images as the input and output images. To get the final image from the output, the l channel is concatenated with the a and b and converted back to the RGB colorspace.

3 Implementation

3.1 Convolutional Neural Networks

We frame the problem as a regression problem - given an input lightness channel $\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$, our objective is to learn a mapping $\hat{\mathbf{Y}} = f(\mathbf{X})$ to the two associated color channels $\mathbf{Y} \in \mathbb{R}^{H \times W \times 2}$.

$$L_2(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{2} \|\mathbf{Y} - \hat{\mathbf{Y}}\|_2^2 = \frac{1}{2} \sum_{h,w} (y_{h,w} - \hat{y}_{h,w})^2 \quad (3)$$

3.1.1 Network Architecture

We used the architecture as described in figure 1. Each convolution layer uses *ReLU* as the activation function. Experimenting with *tanh* as the activation function for just the final convolution layer gave us better results. We used a stride length of 2x2 in the 1st, 4th and 6th convolution layers, while left it at 1x1 for the remaining layers.

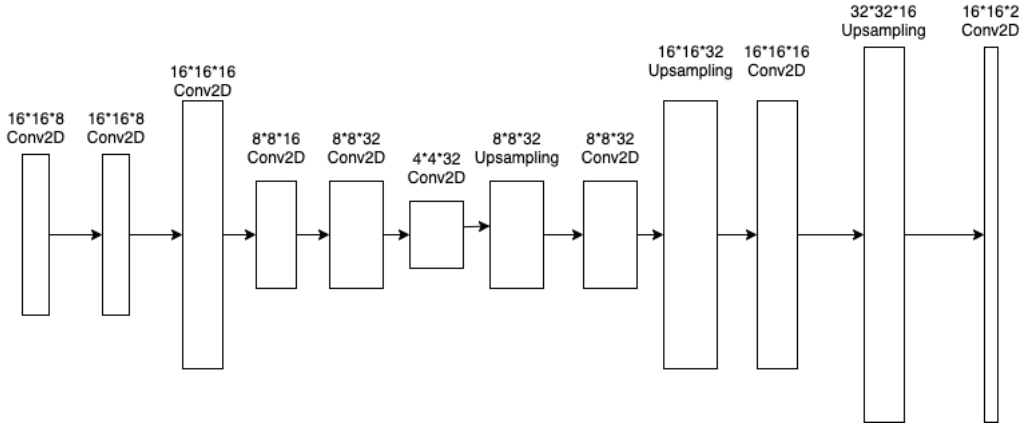


Figure 1: Model of the Convolutional Neural Network

3.1.2 Training

We used stochastic gradient descent to train the network, with the mean squared error loss function. We start with a learning rate of 1e-2. When the loss plateaus and stays the same for more than two iterations, we decrease the learning rate by a factor of 2e-2. We trained our model for 50 epochs.

3.2 Generative Adversarial Networks

Typically, the input to the generator model is pseudorandomly produced noise. In our scenario, the input to the generator is the grayscale image, and the output we'd like it to generate is a colorized version of the grayscale image. Since an additional input is provided (instead of a latent noise vector), our model is a conditional GAN. In effect, we'd like the network to learn a transformation from the input image to a target image. Our implementation is loosely based on the Pix2Pix conditional GAN model. Here, the generator applies some function (that is learned via training) to the input grayscale image producing an image that is intended to be the colorized version of the input image. The discriminator compares this output image and the corresponding color image (the ground truth) from the dataset and attempts to classify the image as real or generated.

In the Pix2Pix model, Isola et al. [2016] use a 3-channel input to both, the generator, and the discriminator, producing a 3-channel output. We use the CIELAB color space to represent images in our implementation, where 1 channel is enough to represent the grayscale image, and 2 channels are enough to represent all the color information.

3.2.1 Generator Architecture

The generator model in our implementation follows the encoder-decoder pattern as in the figure. The input is a grayscale 32-by-32-pixel image represented by the l -channel of the CIE $L^*a^*b^*$ colorspace. The output is a 32-by-32-pixel image represented by the a and b channels of the CIE $L^*a^*b^*$ colorspace. The encoder layers of the generator downsample the input image with a series of encoders into a much smaller higher-level representation of the image features. Each encoder comprises of a convolutional layer, followed by a batch normalization layer and an activation function. The decoder stack upsamples this high-level feature representation and reverse the action of the encoder layers. Each decoder comprises of a deconvolution (the transpose of a convolution operation), followed by batch normalization and an activation function. The activation function used in the encoder is *Leaky ReLU*. Batch normalization is applied to the transposed convolution in the decoder followed by dropout (for the first three blocks). The activation function used in the decoder blocks is *ReLU*.

While the implementation of the generator in the Pix2Pix model uses a U-Net instead of a conventional stack of encoders and decoders, our implementation imitates the same behavior with the use of "skip connections" which directly connect upsampling layers to downsampling layers as in the figure. This effectively gives the network the option of skipping an upsampling layer-downsampling layer pair if it doesn't have a use for it.

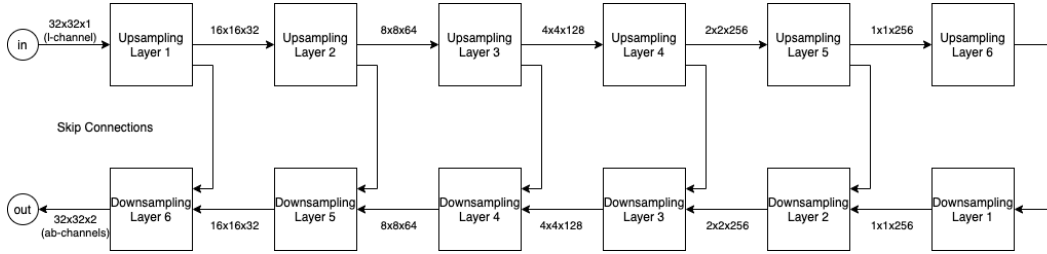


Figure 2: Architecture of the Generator

3.2.2 Generator Hyperparameters and Loss

The generator loss is based on sigmoid cross entropy of the generated images and an array of ones. Isola et al. [2016] also include a L1 loss which is the mean absolute error between the generated image and the target image. We use the same generator loss function as described by Isola et al. [2016], which is defined in equation 4

$$\text{Generator Loss} = \text{sigmoid-loss} + \lambda * \text{L1-loss} \quad (4)$$

The optimizer function used was Adam, which is based on stochastic gradient descent. The learning rate was set to $2e-4$, and the exponential decay rates for the first moment estimates was set to 0.5. λ was set to 100 as in the paper.

3.2.3 Discriminator Architecture

The discriminator in the original Pix2Pix model is a PatchGAN. In our implementation, the discriminator is a more conventional classifier as shown in figure 3. It takes in the l -channel from the input data stacked with the ab channels from the generator output or the input data. It returns the probability of the image being generated by the generator.

3.2.4 Discriminator Hyperparameters and Loss

The total discriminator loss is the sum of two components - a real loss component which is the sigmoid cross-entropy loss of the real images (lab from the input dataset) and an array of ones, and a generated loss which is the sigmoid cross-entropy loss of the generated images (l from the input dataset concatenated with the ab output from the generator) and an array of zeros.

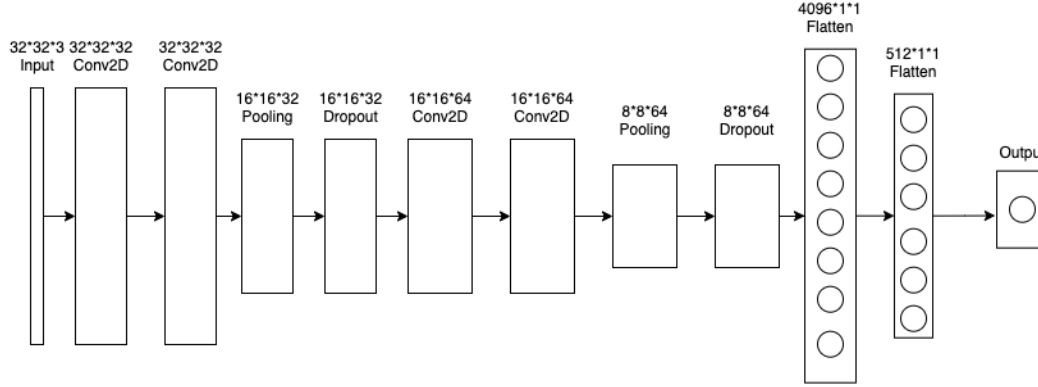


Figure 3: The Discriminator Model

The optimizer function used was Adam, which is based on stochastic gradient descent. The learning rate was set to $2e-4$, and the exponential decay rates for the first moment estimates were set to 0.5.

3.2.5 Training

Our implementation trains the discriminator model and the generator model in two alternating steps. This process is illustrated in figure 4.

- The generator produces an initial output image from the input grayscale image.
- The discriminator classifies this generated output by concatenating it with the input and comparing it with the target concatenated with the input.
- The parameters of the discriminator are updated through the discriminator optimizer function based on the classification error.
- The parameters of the generator are also adjusted through the generator optimizer function based on the classification error by the discriminator.

We performed the training for about 350 epochs, and the metrics of the models still seemed to improve with further epochs. Thus giving the model some more epochs should further improve its performance.

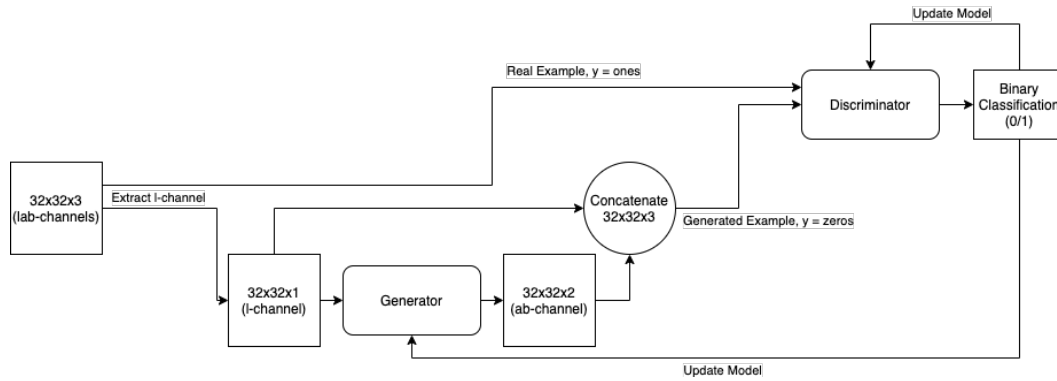
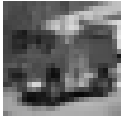









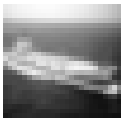

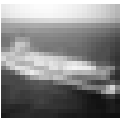


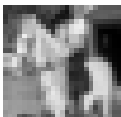





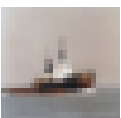










Figure 4: The Training Process

4 Results

The results of our models on some of the samples from the training set are in tables 1, 2, and 3, 4. Table 1 displays the images, whose colorized versions were able to be replicated closely by all our

Table 1: Results - Successful original image replications by both models

Grayscale Image	Ground Truth	CNN (ReLU)	CNN (tanh)	cGAN
				
				
				
				
				
				

models. Table 2 displays the images, where the GAN model was able to perform better on. Table 3 displays some images which do not match the ground-truth images but are realistic nevertheless. Table 4 displays the images which all our implementations performed poorly on.

It is clear from the results that our GAN implementation performs the best consistently over all the samples in the test set. The L2 loss regression also performs well, except for the sepia tones that result from the same pattern (e.g. a car) taking on multiple colors in the ground-truth images causing averaging. It can also be seen that using the *tanh* activation function generates better results than the *ReLU* activation function.

4.1 Quantitative Metrics

Agrawal and Sawhney use the idea of closeness as a rough measure of accuracy and acknowledge that it may not be a good metric to measure our models due to the objective nature of the problem.

5 Conclusion and Future Work

Based on our experiments, we find that convolutional neural networks can be used to colorize images to an acceptable degree of realism. However, there are several enhancements that we could make to both our implementations.

Table 2: Results - Successful original image replications by the GAN

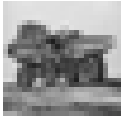




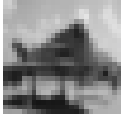




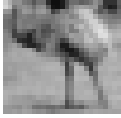















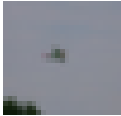


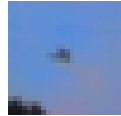
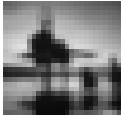




Grayscale Image	Ground Truth	CNN (ReLU)	CNN (tanh)	cGAN
				
				
				
				

Table 3: Results - Realistic Colorizations, not necessarily similar to the input image

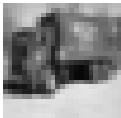




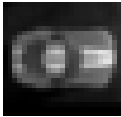




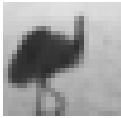




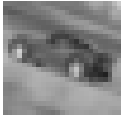




Grayscale Image	Ground Truth	CNN (ReLU)	CNN (tanh)	cGAN
				
				
				

5.1 Classification Instead of Regression

Nazeri et al. [2018] conclude that while L2 loss can work well, it often leads to desaturated sepia tones in the output images, as observed by us. The problem can be reframed as a classification problem where the a and b space can be quantised into bins. Nazeri et al. [2018] use 10 bins corresponding to a range of 20 pixels (the a and b channels have values from -100 to 100). This preprocessing coupled with a multinomial crossentropy loss function can output a class and the colorized image can be reconstructed by replacing the class with the average ab value of that class.

A similar approach is followed by Zhang et al. [2016] to quantize the ab output space into bins with grid size 10. Using the number of quantized classes $Q = 313$ keeps all the pixel values in-gamut

Table 4: Results - Failed Samples

Grayscale Image	Ground Truth	CNN - ReLU	CNN - tanh	cGAN
				
				
				
				

and hence was the number of classes used. Zhang et al. [2016] also implement color rebalancing by reweighing the loss of each pixel during training based on pixel color rarity. This technique could also improve the performance of our CNN implementation.

5.2 Cycle GANs

Cycle GANs Zhu et al. [2017] used CycleGANs for image translation between several domains, famously, including artists' styles and photos, apples and oranges, and zebras and horses. We would also like to further develop our cGAN implementation to incorporate this idea to deal with situations with no paired data for training.

Finally, extending our model to gifs and videos using a combination of our implementations and recurring neural networks is an interesting project to take up.

References

- Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016. URL <http://arxiv.org/abs/1611.07004>.
- Maresh Agrawal and Kartik Sawhney. Exploring convolutional neural networks for automatic image colorization. Technical report.
- Kamyar Nazeri, Eric Ng, and Mehran Ebrahimi. Image colorization using generative adversarial networks. *Lecture Notes in Computer Science*, page 85–94, 2018. ISSN 1611-3349. doi: 10.1007/978-3-319-94544-6_9. URL http://dx.doi.org/10.1007/978-3-319-94544-6_9.
- Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. *CoRR*, abs/1603.08511, 2016. URL <http://arxiv.org/abs/1603.08511>.
- Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. URL <http://arxiv.org/abs/1703.10593>.