Image Colorization using Generative Adversarial Networks

Ramgopal Reddy Putta, Swathi Kolar Ravikumar, and Carlo Provenzani

Seattle University, Spring 2022, DATA-5322 Statistical Machine Learning 2

June 8, 2022

Abstract

This study covers the exploration and utilization of Generative Adversarial Networks (GANs) to colorize grayscale images of various birds common in Western Washington and the Seattle Area. This work goes over different stages from image augmentation and preparation, GAN model construction, and various hyper-parameter tuning to help propound the colored image quality result from the trained network. The study will briefly explore the methodology applied, the theory behind Generative Adversarial Networks, and their utilization to achieve decent models for the selected dataset. The paper also tries to shortly analyze outcomes and choices taken and the quality of the resulting models.

1 Introduction and Overview

Bird Watching is a very profound and grossing outdoor activity around the world. It has a significant impact on stimulating sight and hearing senses by encouraging attentiveness to details, given that there are thousands of bird species throughout the biosphere. The dataset was selected as bird species tend to be highly varying in color and therefore would be interesting to attempt to use for this image colorization project. Realizing that advanced algorithms today can autonomously accomplish arduous tasks like Image colorization with superb precision and realism is incredibly inspiring. Manually performing image colorization is complex and time-consuming; even using state-of-the-art graphics editors would still require tremendous artistic talent from an experienced individual. This study will examine GANs, a reasonably modern Machine Learning methodology first introduced in 2014 to explore and train a model for coloring images of produce. The dataset used includes images of 18 different bird species that, for simplicity, are pre-cropped to ensure that only one bird appears per image and the bird occupies at least 50% of the pixels in the picture.

2 Theoretical Background

2.1 Generative Adversarial Networks

Generative Adversarial Networks are mostly unsupervised machine learning methods with the pursuit of generating new synthetic data that is indistinguishable from training data. The motivation behind Generative Adversarial Networks is to give computers an understanding of what real-world data looks like and train a model in such a way that it can generate data that looks like it's real, even if it is not. A GAN is not precisely similar to classic supervised deep learning, and it is comprised of two major components. The first one is a Network created solely to generate data that tries to mimic actual real data as training of this GAN system is conducted. This Network is called the Generator and the architecture of this network is called a U-Net, which will be covered later on in the study. The second component of GANs is Convolutional Neural network that is indispensable for training the Generator, this Network is called the Discriminator. The Discriminator is responsible for detecting whether an image is authentic, meaning it comes from the actual train dataset, or fake, meaning the Generator generated it. So the Discriminator is, in essence, just a binary classification model that outputs one if it thinks the image is real or zero if the image is false.

Now, since GANs involve two totally different Networks with their own separate purpose, they require two loss functions. The Discriminator is essentially just like any other binary classification problem, so the issue to solve is attempting to minimize something very similar to binary cross entropy(log loss). The regular binary cross entropy function is depicted below.

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(p(y_i)) + (1 - y_i) * log(1 - p(y_i))$$

Where y_i is the actual label(0 or 1) of the data sample, and $p(y_i)$ is the predicted probability of model. So the implication here is that the loss penalizes both false positive and false negative results. Conceptually the loss for the Discriminator is the same and can be viewed readily as

$$DiscriminatorLoss = -\frac{1}{N_{real}}\sum_{i=1}^{N_{real}} \log(D(x_i)) - \frac{1}{N_{generated}}\sum_{i=1}^{N_{generated}} \log(1 - D(G(z_i)))$$

Where $D(x_i)$ is the output of the Discriminator of a real image, and $D(G(z_i))$ is the output of the Discriminator of a generated image from the Generator.

Variable z_i in the above equation represents a vector of random inputs. For the Generator to always create random nonexistent fake data that will ultimately learn to produce them to look more realistic, the input of the Generator is just random noise. The neural network then uses this random noise to mold the Generator's output into something significant.

Where $D(x_i)$ is the output of the Discriminator of a real image, and $D(G(z_i))$ is the output of the Discriminator of a generated image from the Generator.

Variable z_i in the above equation represents a vector of random inputs. For the Generator to always create random nonexistent fake data that will ultimately learn to produce them to look more realistic, the input of the Generator is just random noise. The neural network then uses this random noise to mold the Generator's output into something significant.

For the Generator, however, the cost function is as follows:

$$GeneratorLoss = \frac{1}{N_{generated}}\sum_{i=1}^{N_{generated}} \log(1 - D(G(z_i))$$

This cost function essentially implies that we want the Discriminator to classify images created from the Generator as "one" for generated images. Meaning it is trying to fool the Discriminator by creating natural-looking images using the Generator. However, there's a caveat. The Generator can only learn to make authentic images if the Discriminator is able to distinguish between real and fake photos, so if the Discriminator does a poor job, the Generator won't learn how to create realistic data. Ideally, the two networks would improve together, which is the trickiest part to accomplish in GANs.

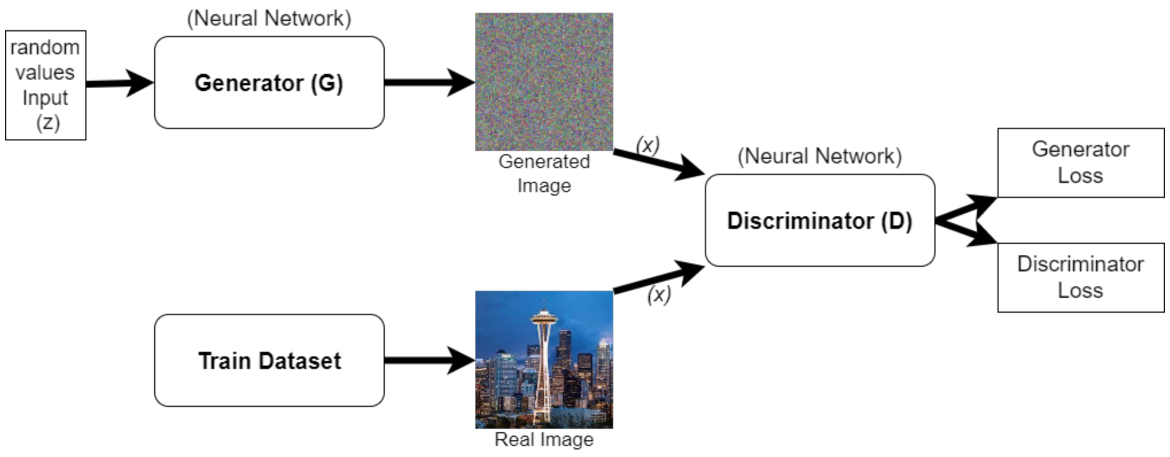


Figure 1: GAN Architecture

2.2 Conditional Generative Adversarial Networks

Conditional Generative Adversarial Networks, or CGANs, are intuitively similar to GANs. Here, the leading idea is that valuable information is handed to the GAN model in order to train the generator based on decided constraints. In order to do this, not only does the Generator need this additional information, which in the case of this study, is a grayscale image. But the Discriminator also needs that same information in order to evaluate what the generator created. So the grayscale image constrains both the Generator and the Discriminator. By adding additional knowledge to the networks, both the Generator and the Discriminator learn to operate in specific ways. So intuitively, for this study, grayscale images will be part of the input for the Generator and part of the Discriminator's input, along with either the natural or generated image.

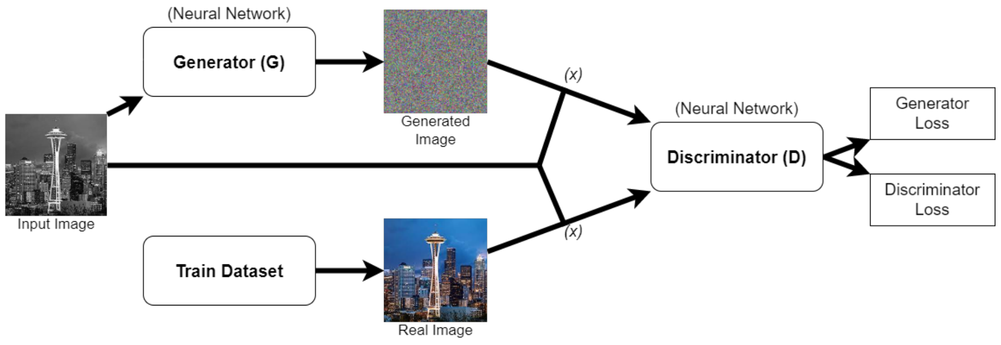


Figure 2: CGAN Architecture

2.3 U-Net

U-net is an architecture for fully convolutional neural networks specializing in image segmentation. Image segmentation, also called semantic segmentation, is a procedure that, for example, predicts whether something is on an image and can create a "mask" that shows where on the image that specific object is located and its dimensions. Semantic segmentation is a variation of classification where every pixel in the picture gets assigned a class that it belongs to. They are then grouped to form a mask. It is fully convolutional as the network does not contain any fully connected layers. The U-Net is the primary structure of generative models. The U-net architecture consists of two parts: the encoder/downsampling and the decoder/upsampling parts. The encoder, which comes before upsampling, resembles what regular CNNs perform and is constituted by the general convolutional process involving convolutional layers followed by max pooling operations and activation functions. Essentially trying to understand information about the input. In the decoder, it is the opposite. The data is upsampled, in most cases, back to the original size of the input. It increases the size of the information while reducing the number of channels by performing something called the transpose convolution.

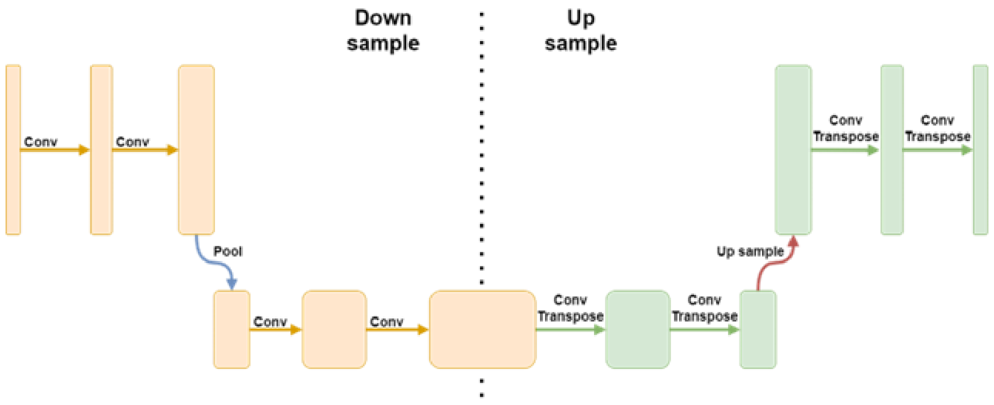


Figure 3: U-Net Architecture

3 Methodology

The Bird Image dataset comprises 1800 224x224 RBG images, 100 for each bird type. The initial task was to determine whether the dataset needed any modification before training to reduce the amount of training required. GANs usually take a considerable amount to train. The size of the images was firstly reduced by an important factor, to 128x128 pixels down from 224x224, as this would be better suited for time constraints and a less complex model. Preparing the dataset involved converting the newly resized RGB images to one-channel grayscale images as they would serve as input parameters to both the generator and discriminator model. The dataset of the 1800 colored bird images with their corresponding grayscale image was then split into a 95/5 train and test set, resulting in a 1,710 train subset and 90 test subset. The CGANs for this study were built and trained using Tensorflow. Proceeding by some initial trial and error, a relatively simple promising U-Net generator consisted of six downsampling layers and six upsampling layers. The entire generator network used a kernel size of 4 by 4 pixels. Instead of pooling, the model used a stride of 2 in convolutions. This would make the downsampling nicely turn the 128 by 128 by 1 into a 1 by 1 by "number of filters" after seven layers. The number of filters used for each layer consisted of 64, 128, 256, 512, 512, 512, and 512. This would turn the input into a 1x1x512 shape tensor after downsampling. The upsampling consisted of these same filters in reverse using the TensorFlow Conv2dTranspose function to convert the shape back to its original form, except for the last layer, which consisted of adding three channels with an outcome of 128x128x3 to the network instead of converting it back to 128x128x1. For the discriminator, a relatively straightforward classifier was assembled, although instead of just one image being passed as input, the model would take the grayscale image and either the original image or the generated image as one concatenated input to the discriminator network. The number of filters for each convolutional layer increased incrementally, resulting in layers with 64, 128, 256, and 512. The kernel size of 4 for convolutions. The GAN network was then trained in batches of 32 for 400 epochs, as GANs generally take a long time before producing something meaningful. This roughly took about two hours using a NVIDIA GeForce RTX 3050 Ti. The generator did a satisfactory job at coloring the images, but not great. The loss interpretation was somewhat strange after training. After training, the discriminator loss appeared to converge to zero, which most likely means that the generator isn't doing a decent enough job for the discriminator to be unable to distinguish authentic and generated images.

After doing further research to try to mend the present outcome situation, The generator was changed in an attempt to test whether skip connections would help. Skip connections bypass layers and feed the output of one layer as the input of another layer further down the network.In the case of U-Nets, skip connections save the outputs of the downsampling convolutional operations and then congregate them with the transpose convolutions in upsampling. This idea is that it forces details learned in each layer of downsampling to construct the output more suitable while upsampling.

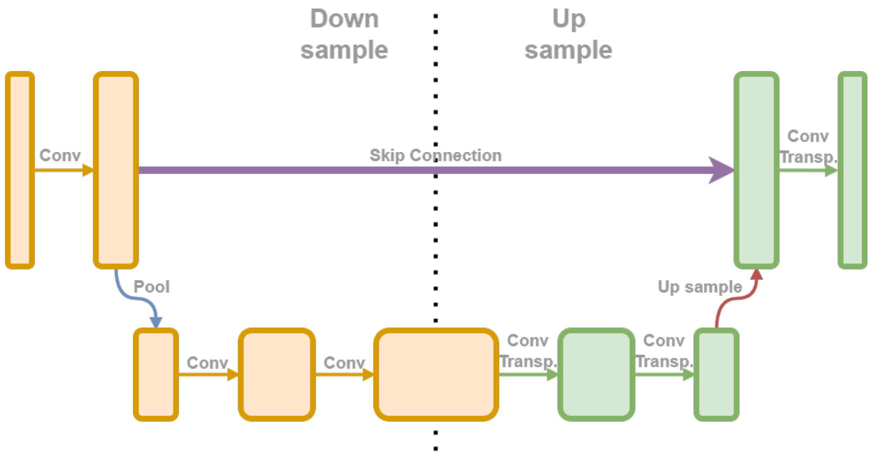


Figure 4: U-Net Skip Connrctions

Apart from adding skip connections to the generator, both generator and discriminator remained unchanged. Before retraining, some straightforward image augmentation was also performed to create more data per bird specie. The training dataset was flipped along the x axis, rotated by ninety degrees clockwise and counterclockwise. This resulted in the training data having 6,840 images.

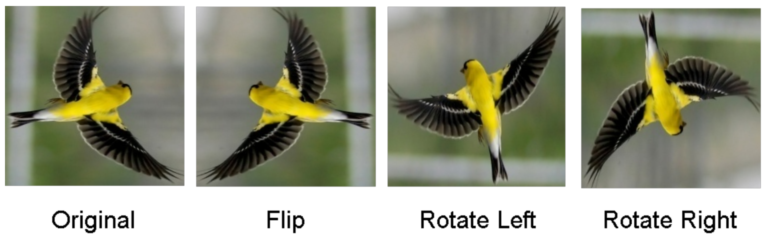


Figure 5: Inage augmentation

After training with the generator having skip connections, surprisingly, the generated images resulted considerably more accurately in terms of actual image resemblance. Also, the discriminator loss luckily did not converge to zero this time. The model was trained for 50 with the added augmented images in the training dataset. The results from the generator seem to be having very minimal changes around that point 50 epoch mark.

4 Comutational Results

The training results from the first attempted model seemed considerably adequate, as illustrated in Figure 3. The model managed to grasp the basic features of the bird. After the first 80 epochs, the generator model already outputted vague resemblances of birds. Over the following few hundred epochs, it refined a few more minor bird details, resulting in increasingly better quality. The results after 400 epochs, although still a bit blurry looking, got reasonably close to the actual image.

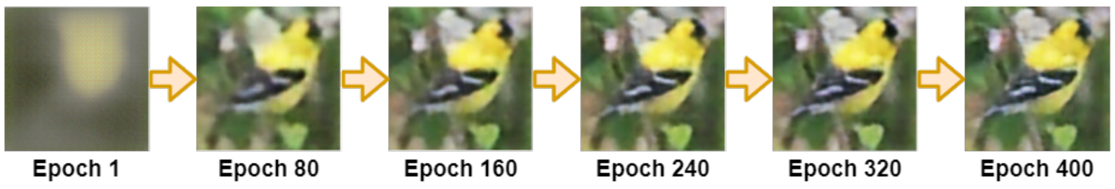


Figure 6: First Model Training Results

Although the first model managed to create decent looking images after 400 epochs, it was not good enough to fool the discriminator, so the loss of the discriminator managed to reach throughout almost the entire training as shown in figure 7.

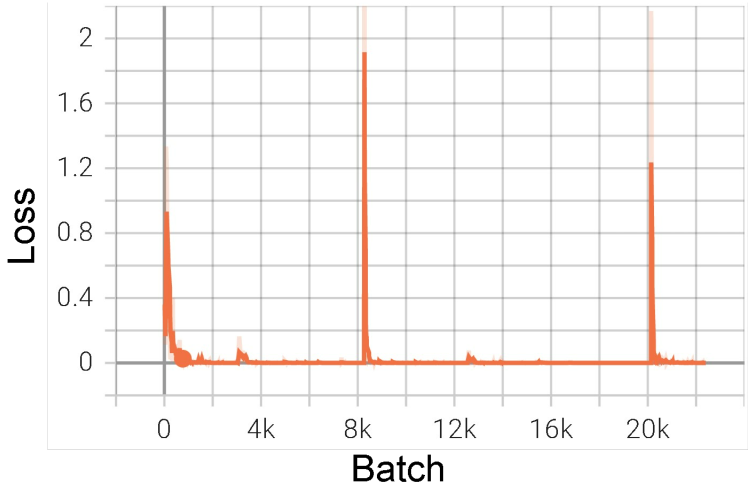


Figure 7: First Model Discriminator Loss

The first model, although not doing good in regards to the discriminator, also distinguished between bird species and the surrounding backgrounds of the images like skies, terrains, and tree branches. It managed to reasonably color both bird species and surroundings correctly, as shown in Figure 4. As before, it did not manage to learn small details, and the results illustrated appear a bit unclear.

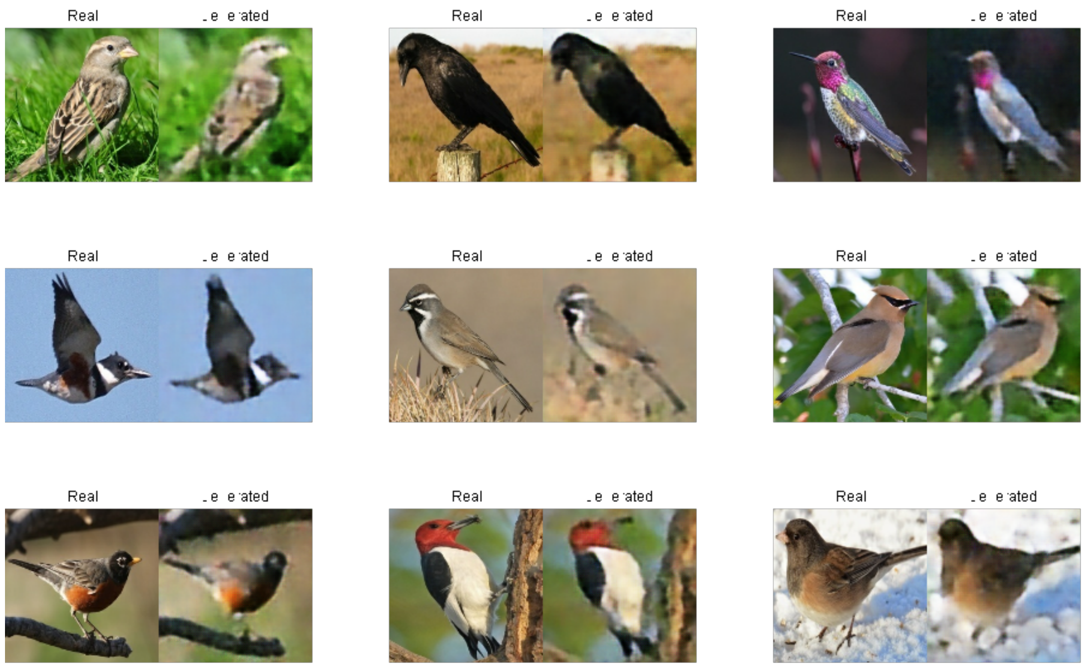


Figure 8: First Model Training Results

The new model, where the generator contained skip connections, did tremendously better over the 50 epochs it was trained, where the data contained augmented images. The images resulted very realistic starting at epoch 20, it then refined very little colored details from that point forward as shown in Figure 9.

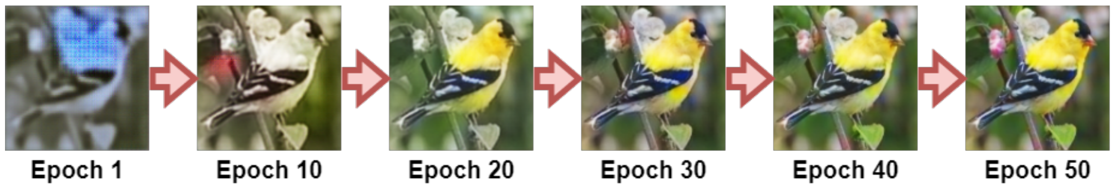
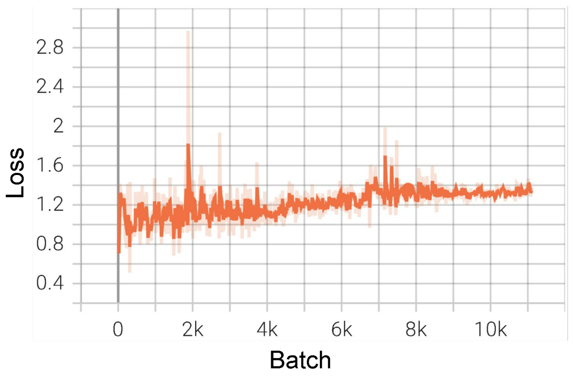
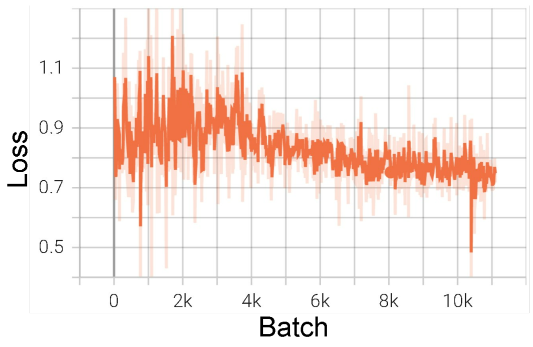


Figure 9: Second Model Training Results

From the losses of the second model, it is essential to note that neither the generator nor the discriminator has prevailed. Since neither the generator and discriminator got very low loss levels(making the other high), as shown in Figure 10, it is an indicator that neither model is dominating the other.



(a) Discriminator Loss



(b) Generator Loss

Figure 10: Second Model Losses

Results from some testing data for the second model show much better performance and clarity of the colored output. As shown in Figure 11, the predicted RGB images look almost indistinguishable from the authentic images.

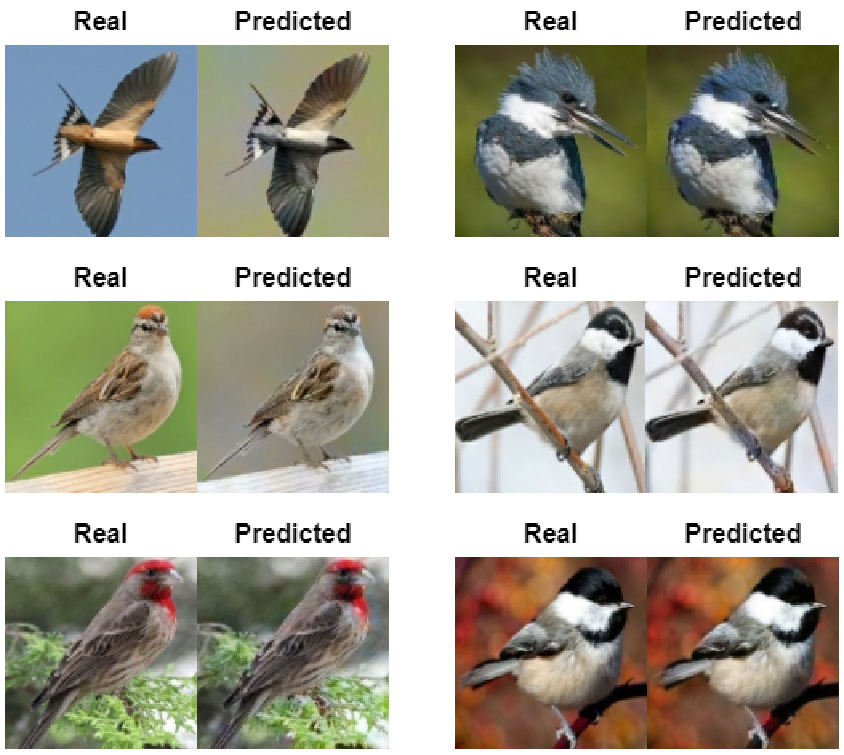


Figure 11: Training Results

The second model was also used to predict colored images of random colorful bird species not found in the training dataset for curiosity, as shown in Figure 12. As expected, the results were not very impressive.

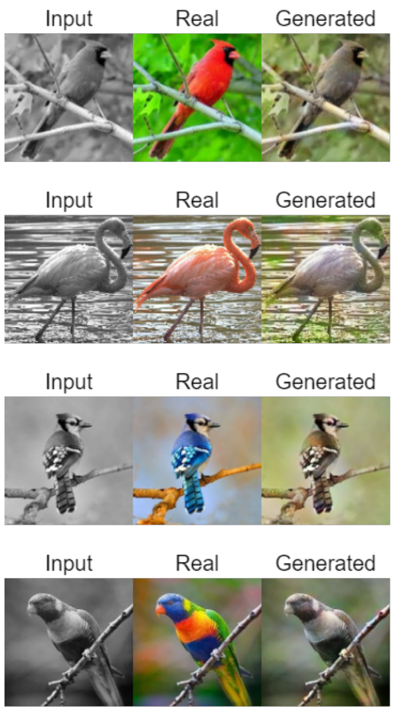


Figure 12: Training Results

5 Summary

This study aimed to explore and utilize Conditional Generative Adversarial Networks to colorize grayscale images of various birds common in Western Washington and the Seattle Area. Luckily, building the models was fairly intuitive using Tensorflow, although learning and figuring out the architecture before constructing it and arbitrarily choosing various hyperparameters was somewhat challenging. The dataset did not require much preparation for the goal of this study. And the model results managed to be reasonably triumphant based on simple standards. In summary, this study confirmed that Neural Networks achieve good identification results relatively quickly and in little time. They are also flexible to customized utilization and can become accurate if adequately set and tuned.