Bird Generation with DCGAN

Zhiling Huang
Department of Computer Science
Stanford University
Stanford, 94305
zhiling@stanford.edu

Abstract

In this paper, I used unsupervised Deep Convolutional Generative Adversarial Network (DCGAN) to learn how to generate images of birds. Unsupervised representation learning is notoriously hard to train and this project is even more challenging due to the diverse looks of different birds, especially their different physical shapes, different body movements and different backgrounds.

1 Intro

The goal of the project is to learn how to generate images of birds, through unsupervised Deep Convolutional Generative Adversarial Network (Radford et al. [2015]). The model is composed of a discriminator and a generator. The discriminator is a classifier that predicts whether an image is real or not. The generator generates images to try to fool the discriminator. Both are learned and updated in each iteration of learning.

2 Data & Data Preprocessing

I am using Caltech-UCSD Birds 200 (Welinder et al. [2010]), which includes 11776 images of 200 types of birds. The images are of different sizes. To reduce the noise of background and make it easier to focus only on the bird itself not background, I first used the bounding box to crop out just the bird and then reshape the image to 64*64.

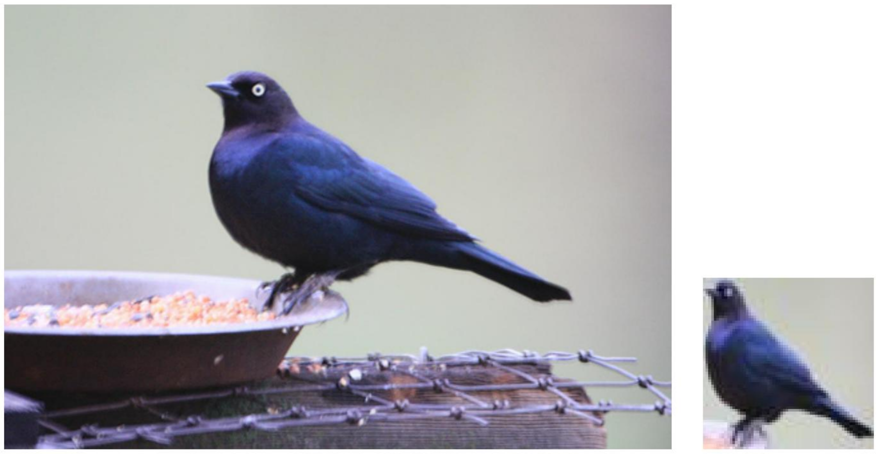

Figure 1: Left, Original Image of Brewer Blackbird; Right, Preprocessed Image

For example, in the original image, the brewer blackbird is standing on a bowl of bird food, which is irrelevant to representation learning of bird. I cropped out the irrelevant part and leaves only the bird part and then reshapes the remaining image to 64*64. By cropping out the background, the model will not be spending unnecessary time on learning the representation of irrelevant part like the bowl of food in this image.

3 Architecture

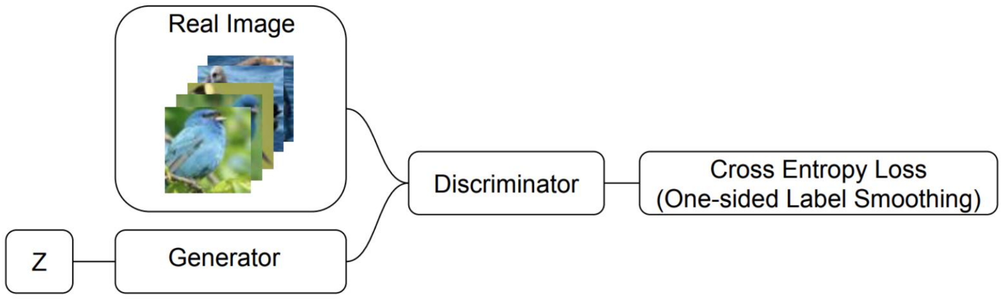

Figure 2: Architecture

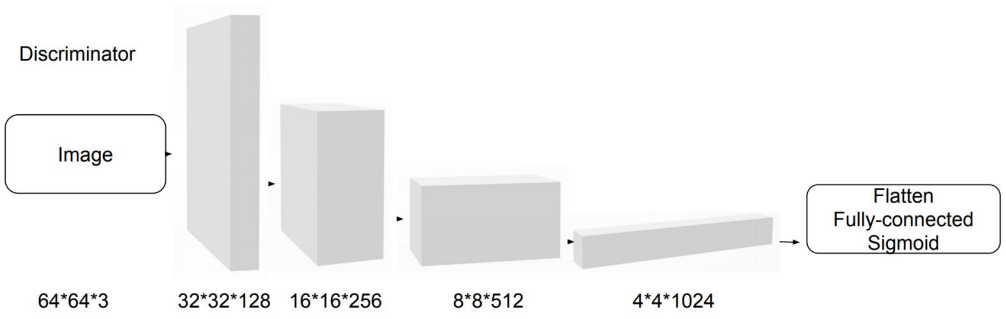

Figure 3: Discriminator

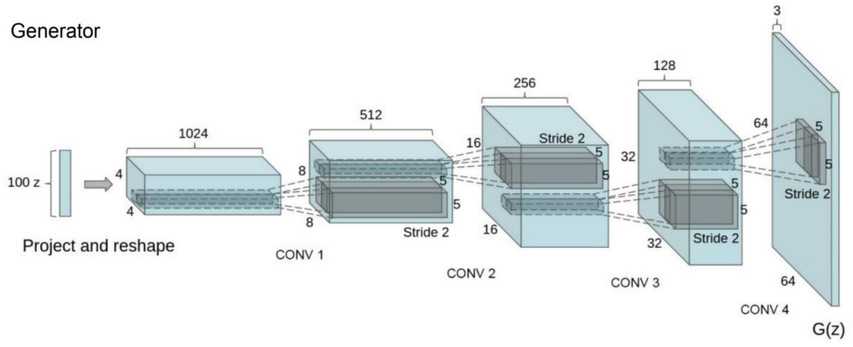

Figure 4: Generator

The discriminator takes in a 64*64 image and passes it through 4 convolution layers. The output of the 4th convolution layer is flattened and then followed by a fully connected layer. Lastly the output layer is a sigmoid function.

The generator is almost the exact reverse of the discriminator. It takes in a Z vector of length 100. Z vector is passed in through a fully connected layer, whose output is of length 16384. This output is then reshaped to be 4*4*1024. Then there are 4 deconvolution layers and the output of the 4th layer is 64*64*3, which is the generated image.

Hyperparameter Tuning & Architecture Hacks (Chintala et al. [2016]):

- Use non-saturated cost function for generator, so that the generator learns faster.
- 2 updates for discriminator/1 update for generator. I tried several other ratios (3:1, 4:1, 1:2 etc), and also strategy that keeps updating generator and discriminator separately until loss is smaller than some threshold. But 2:1 ratio produces most realistic images.
- Virtual batch normalization for generator and regular batch normalization for discriminator. Virtual batch normalization ensures that generated images in one batch are not correlated.
- One-sided label smoothing, by changing the ground truth label to 0.9 for discriminator.
- Use normal distribution (mean 0, variance 1), not uniform distribution for Z.
- LeakyReLU for both discriminator and generator to avoid sparse gradients.

4 Result

4.1 Sample Generated Images

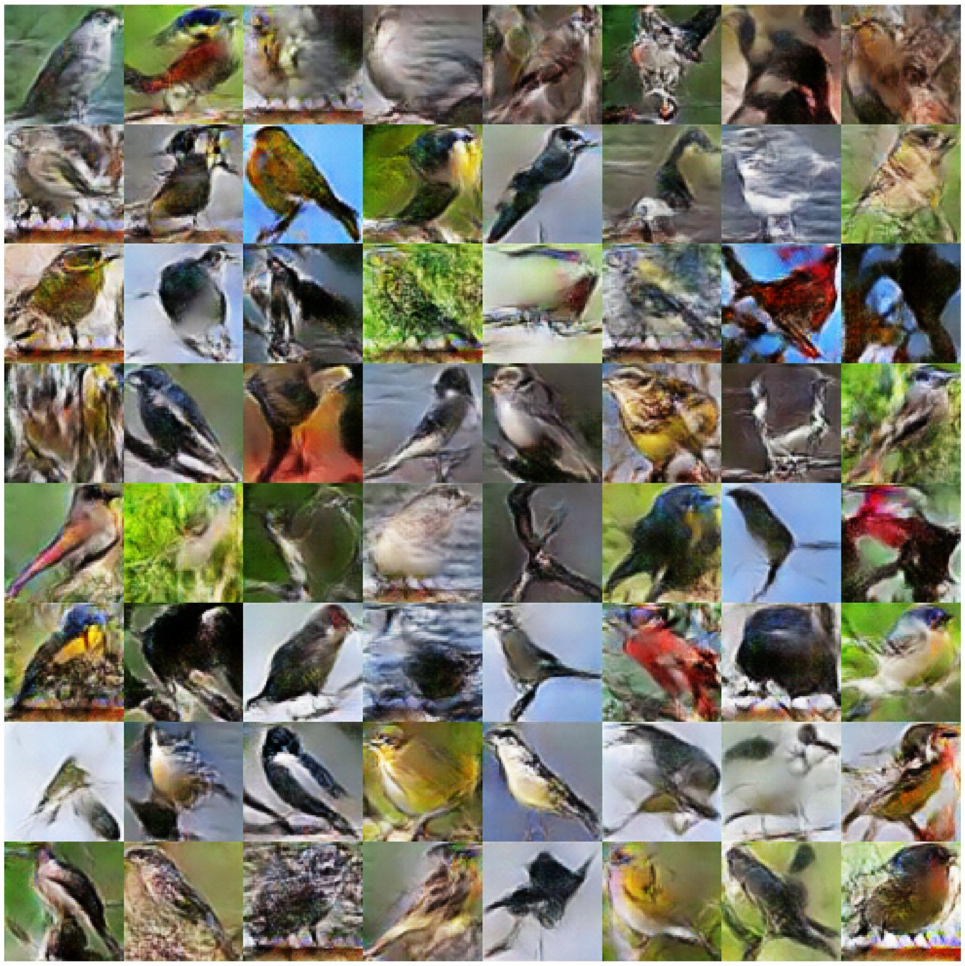

Figure 5: 64 Sample Generated Images

4.2 Incrementing Entries of Z Vectors

In this section, I explore with incrementing any one entry of Z vector while keeping all other entries of Z the same. That way I know what effect the entry has. First I construct Z, with 100 samples from a normal distribution with mean 0 and variance 1. Secondly I copy Z to have in total 8 same Z. And then for entry i (any number from 1 to 100), I set the ith entry in 8 copies to be -2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5. Lastly, I used these newly generated 8 Z vectors to generate 8 images and see what effect incrementing the ith entry will have in the result images. I repeated the process several times (testing with different Z vectors and different index).

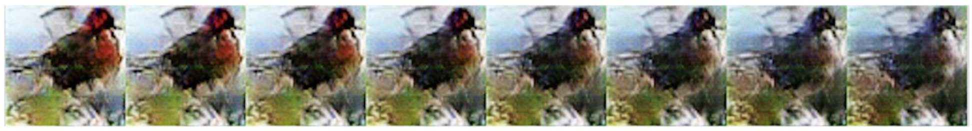

Figure 6: Example 1 Increment Z

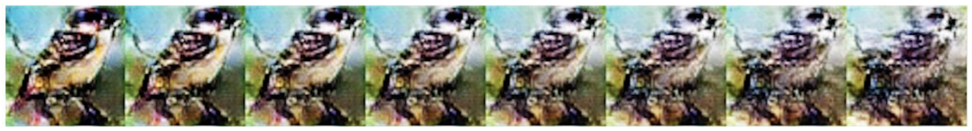

Figure 7: Example 2 Increment Z

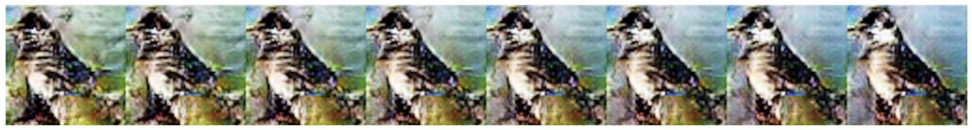

Figure 8: Example 3 Increment Z

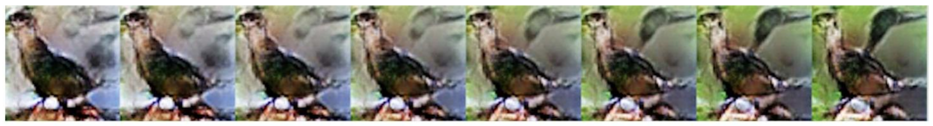

Figure 9: Example 4 Increment Z

We can see that some entries of Z vector shift color of bird from red to white, background color from green to blue; some even change the shape of the bird (example 2); others might add a duck head (example 4).

4.3 Average of Two Z Vectors

In this section, I tested with averaging two Z vectors and see whether the image generated with the averaged Z vector is a combination of features in images generated by the two Z vectors separately. Specifically I first generate vector Z1 and Z2 independently. Z3 is (Z1 + Z2)/2. Then I generate images with Z1, Z2 and Z3.

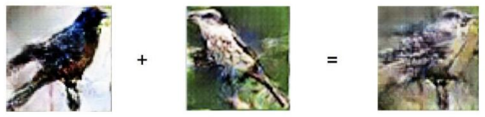

Figure 10: Example 1 Average Z vectors

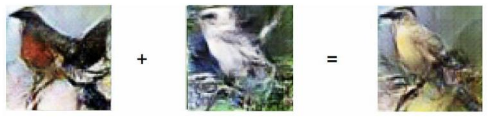

Figure 11: Example 2 Average Z vectors

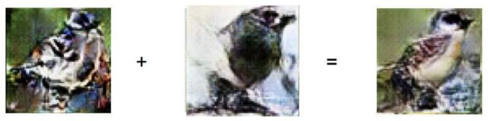

Figure 12: Example 3 Average Z vectors

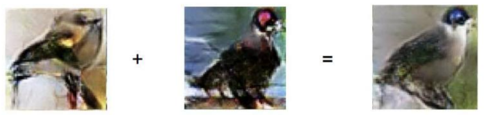

Figure 13: Example 4 Average Z vectors

On the whole, the effect of averaging Z vectors is not easy to grasp. In some cases, the effect is straightforward: for instance, in example 1, the average of a black bird and a white bird is a gray bird; in example 2, the average of a red and black bird and a white bird is a yellow bird. In other cases, the effect of averaging is harder to measure, such as example 3. In example 3, dark green background color merged with white background color becomes light green background color, which is as expected. But it's harder to understand why in example 3, the average of two birds with dark black belly has a white belly.

5 Conclusion & Future Steps

With 11776 images, I am able to generate images of birds vaguely and see some interesting results by incrementing any one entry of Z vectors and averaging two Z vectors.
However, I am short of training images. This problem is worse due to the diversity of birds in the corpus. There are in total 200 distinct types of birds, all with different outlooks. The hundreds of types of birds also come with hundreds of different backgrounds, sandy rocks, leafy branches, water, or the sky. What's more their body movements also pose a challenge for representation learning. Some birds are flying, some are standing on a branch or a stone, while some others are floating on water. Lastly, the images are taken from various angles. Even the same bird, doing the same thing, in the same background will result in different visual appearances due to the countless different angles the image can possibly be taken.
One other thing I noticed is that generated birds are fat. This is due to resizing the bounding box during preprocessing. Cropping in the center instead of resizing might work better.
It would also be interesting to see, for what range of Z, the result image is most realistic.
As with evaluation metric, I can use a pre-trained bird detector to check the percentage of 1000 generated images that are classified as bird. So that I have a more quantifiable metric.

References

Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. How to train a gan? tips and tricks to make gans work. 2016. URL `https://github.com/soumith/ganhacks`.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL `http://arxiv.org/abs/1511.06434`.

P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.