Drone image generation using DCGAN

CS 445 Spring Term, 2019

Tamotsu Tanabe, ID: 944654432

Introduction

In our company, we use NN (Caffe) based Classifier to classify drones from other objects in video stream in real time. The classifier works relatively well, considering the limited number of image set we have. If we have more images, presumably the accuracy should improve.

We have quite a few video footages from customers but those are usually confidential and not allowed to be used directly in our product.

In this project, I try to generate images of drones using GAN (Generative Adversarial Networks). By using large number of synthesized images, we will be able to train the classifier better, and also confidentiality issues of videos can be avoided.

Methods

How GAN works

GAN [1] is one of the techniques to generate synthetic images out of training images, and it has gained lots of popularities in recent years.

GAN consists of two neural networks, one is called *Generator*, the other is *Discriminator* (as shown in the figure below).

The Generator creates fake images out of random noise. The Discriminator takes in an image from both real and fake images and judge if the image is real or fake. As the training goes by, the Generator creates better images, and at the same time, the Discriminator is getting better at judging fake images from real ones.

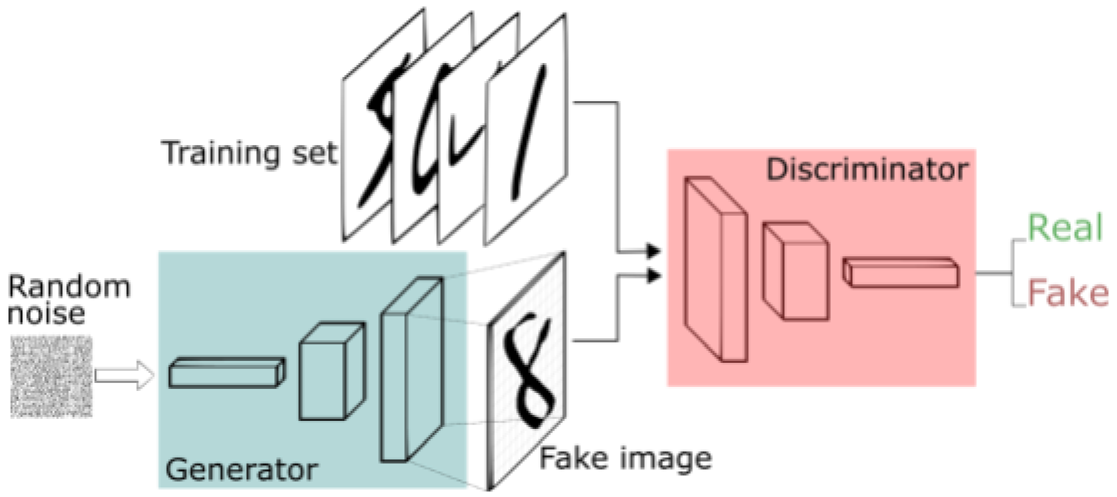


Figure 1. GAN architecture [2]

DCGAN

There are lots of variations of GANs. In this project, I choose *DCGAN* [3] (Deep Convolutional GAN). DCGAN replaces the NN layers in GAN with Convolutional NN (or CNN). The original GAN doesn't take the relationships between neighboring pixels into consideration, and which usually results into images with small pixel noises. Using the convolution layer in DCGAN seems to be a natural extension to the original GAN considering that the convolution layer takes care of the neighboring pixel relationships.

Training Image Data Set

I used this program [4] to collect drone images from web. I manually excluded images which had lots of noise or other objects in the scene, or texts overlays on top of drones, etc. Images are then resized to 128x128.

After those pre-processing, there are total about 800 images. 800 images seem too small for a training set. There are techniques to increase the number of samples by slightly modifying the original images, such as slightly rotating the image, or adding some noise to it. Due to the limited time, I didn't try those techniques.

First Implementation

I used Python/Keras with TensorFlow backend since it seemed to be one of the most popular framework nowadays.

There are number of implementations available online, and I started with this implementation [5] as a reference because it seems to be closer to the algorithm proposed in the original DCGAN paper [1].

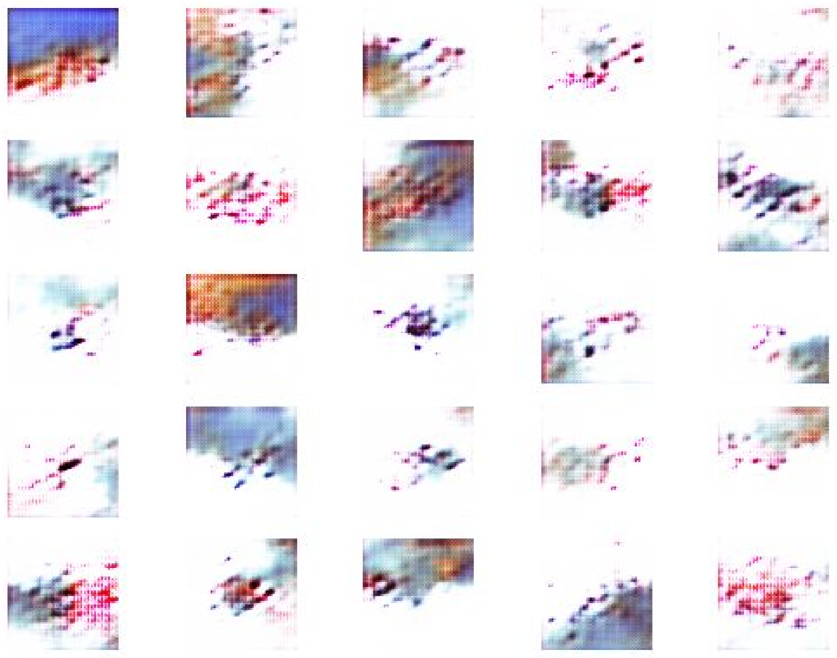
Below is the Generator part of my first implementation.

It consists of 4 convolution layers. The output image size is 128 x 128 x 3 (color image). The input noize is a 100 dimension 1D array, which seems to be very common in DCGAN implementations.

```
class DCGAN(): # 1st implementation
    ...
    def build_generator(self):
        model = Sequential()
        model.add(Dense(256 * 16 * 16, activation="relu", input_dim=self.latent_dim))
        model.add(Reshape((16, 16, 256)))
        model.add(Conv2DTranspose(256, kernel_size = 5, strides = 2, padding='same'))
        model.add(BatchNormalization())
        model.add(LeakyReLU(alpha=0.01))
        model.add(Conv2DTranspose(128, kernel_size = 5, strides = 2, padding='same'))
        model.add(BatchNormalization())
        model.add(LeakyReLU(alpha=0.01))
        model.add(Conv2DTranspose(64, kernel_size=5, strides=1, padding='same'))
        model.add(BatchNormalization())
        model.add(LeakyReLU(alpha=0.01))
        model.add(Conv2DTranspose(self.channels, kernel_size=5,strides=2,padding='same'))
        model.add(Activation("tanh"))
        z = Input(shape=(self.latent_dim,))
        img = model(z)
        return Model(z, img)
```

The figure below are the generated images after ~3,000 iterations. There is a code by the authors of the original paper, but it is not using Keras. I tried to mimic the original implementation as close as possible, but since I am not very familiar with Keras or TensorFlow, I must have had something wrong. The code didn't work well.

Figure: Images from first implementation (~3,000 iterations)



Second Implementation

After lots of try & errors and looking up other implementations (mostly in these sites [6][7][8]), below is an updated implementation:

```
    def build_generator(self):
        model = Sequential()
        model.add(Dense(128 * 32 * 32, activation="relu", input_dim=self.latent_dim))
        model.add(Reshape((32, 32, 128)))
        model.add(UpSampling2D())
        model.add(Conv2D(128, kernel_size=3, padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))
        model.add(UpSampling2D())
        model.add(Conv2D(64, kernel_size=3, padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))
        model.add(Conv2D(self.channels, kernel_size=3, padding="same"))
        model.add(Activation("tanh"))
        noise = Input(shape=(self.latent_dim,))
        img = model(noise)
        return Model(noise, img)
```

Major differences from the first one are:

- A: Number of convolution layers are reduced from 4 to 3.
- B: Using {UpSampling2D + Conv2D} Instead of Conv2DTranspose.
- C: Activation function is changed from LeakyRdLU to relu.

Not quite sure which one really made a difference, but this implementation worked much better. After 3,000 iterations, the images already kind of look like drones.

Below are some of the images generated by this implementation:

Figure: Images from second implementation (3,000 iterations)

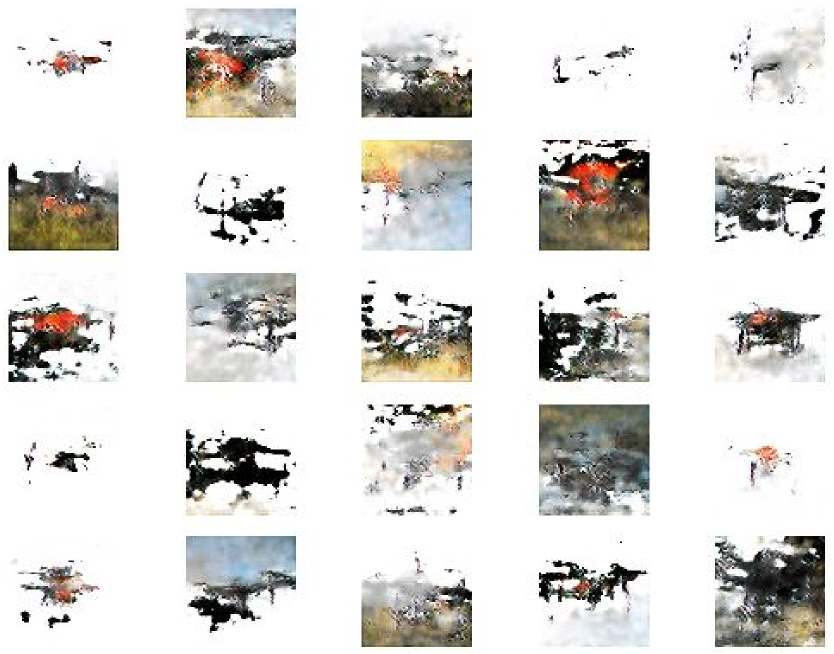


Figure: Images from second implementation (10,000 iterations)

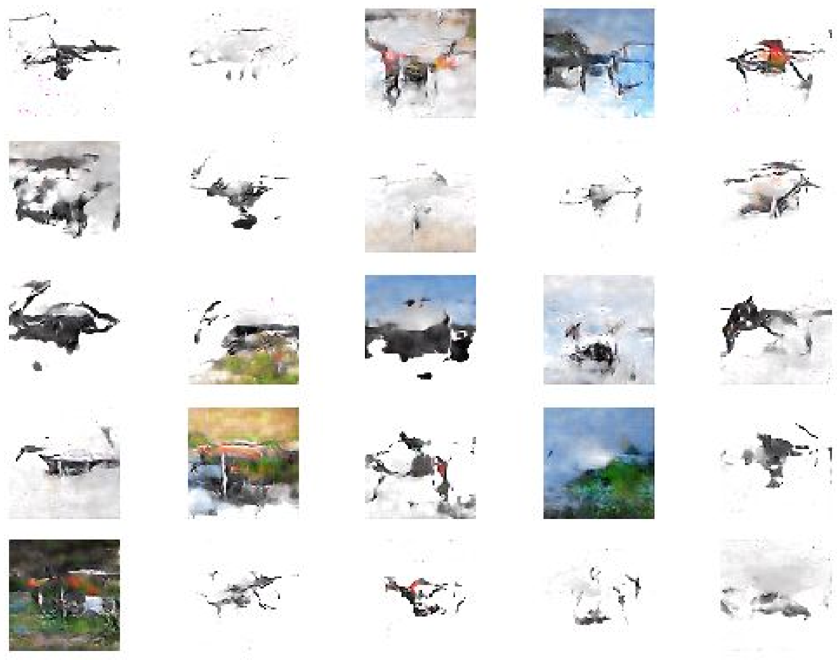


Figure: Images from second implementation (20,000 iterations)

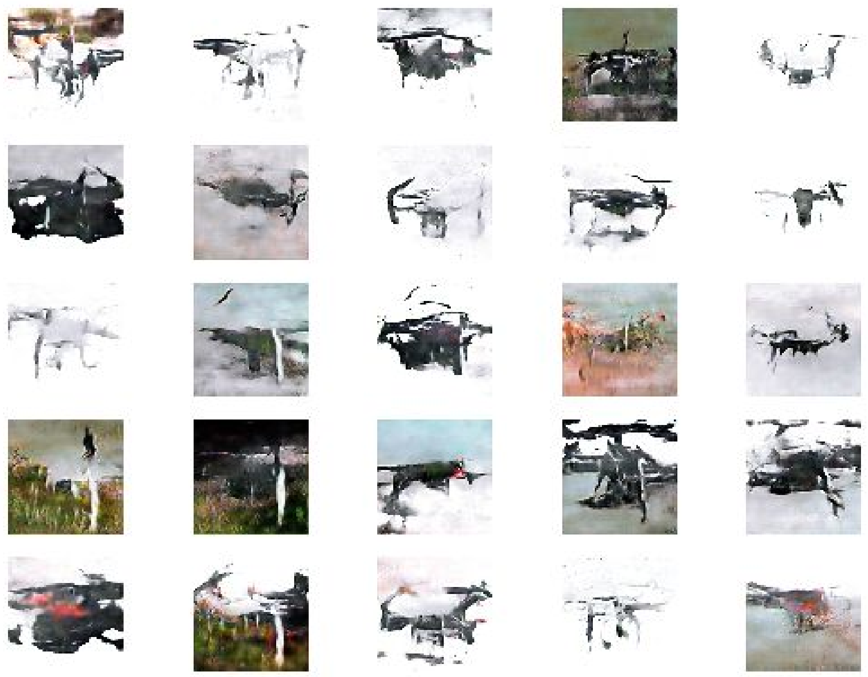
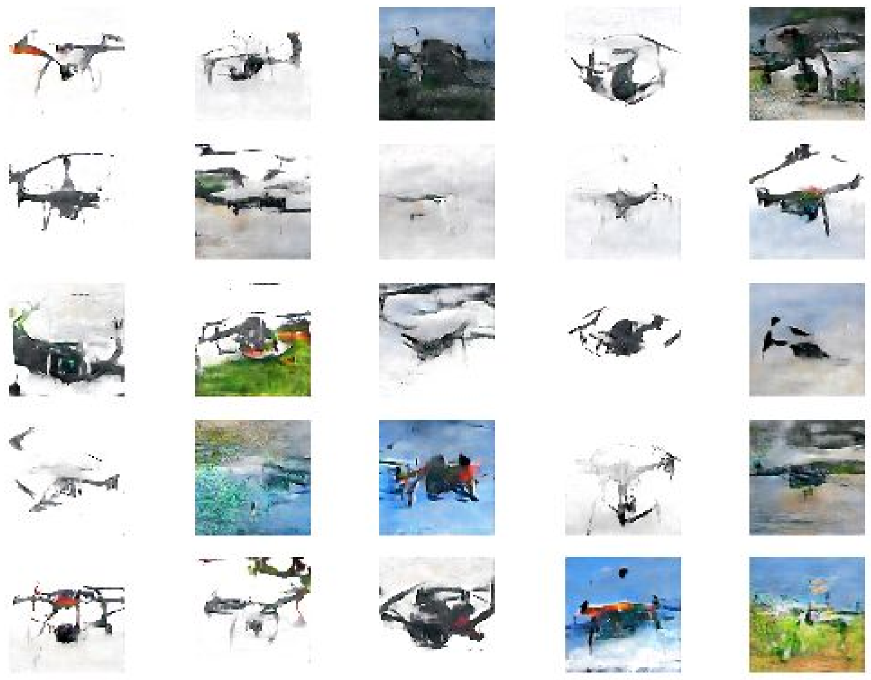
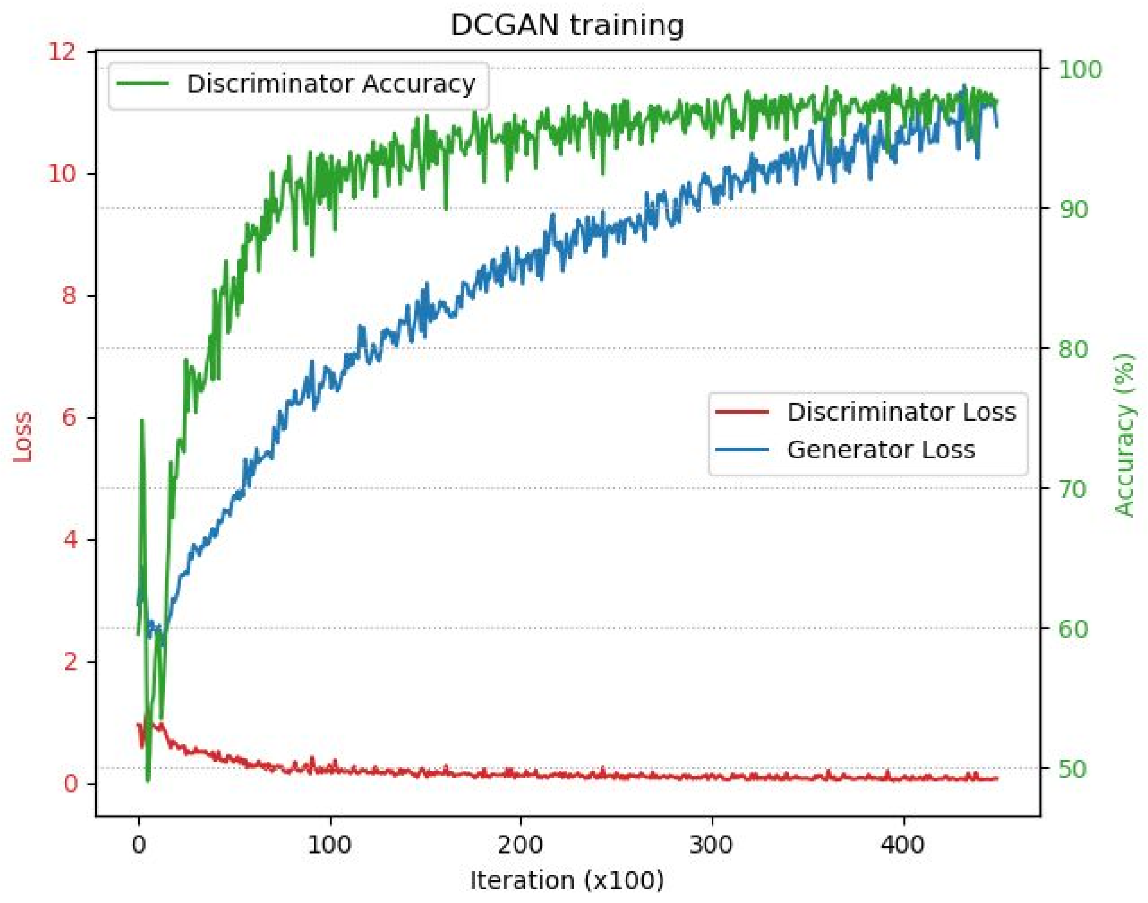


Figure: Images from second implementation (40,000 iterations)



Results of Experiments

The figure below shows the Loss values from Discriminator and Generator, and accuracy from Discriminator. In this experiment, the graph looks like it converged well. Unlike usual NN training process, however, even if the graph looked like converged, it doesn't necessarily mean you get good quality images. I had a few cases where the graph looked like this but the images were just really bad.



NOTE: the current implementation does not implement epoch correctly (even though the variable name is "epoch"). The program just iterates the loop, with a batch size of 16. The total training images are 784, so 1 epoch ~= 784/16=49 iterations (but the program randomly samples 16, so it is not epoch).

Discussion

Difficulty of GAN

GAN is really interesting and fun, but there are a lot of difficulties:

- Training takes very long time and also takes lots of memory. Having a good GPU would be nice.
- It is difficult to predict the end results.

Quality of images

- Generated images look like drones from distance but not really.
- They are strangely mixed images -- in some case, some portion of the image is from one type of drone and another part is apparently from different type of drone.

Possible Improvements

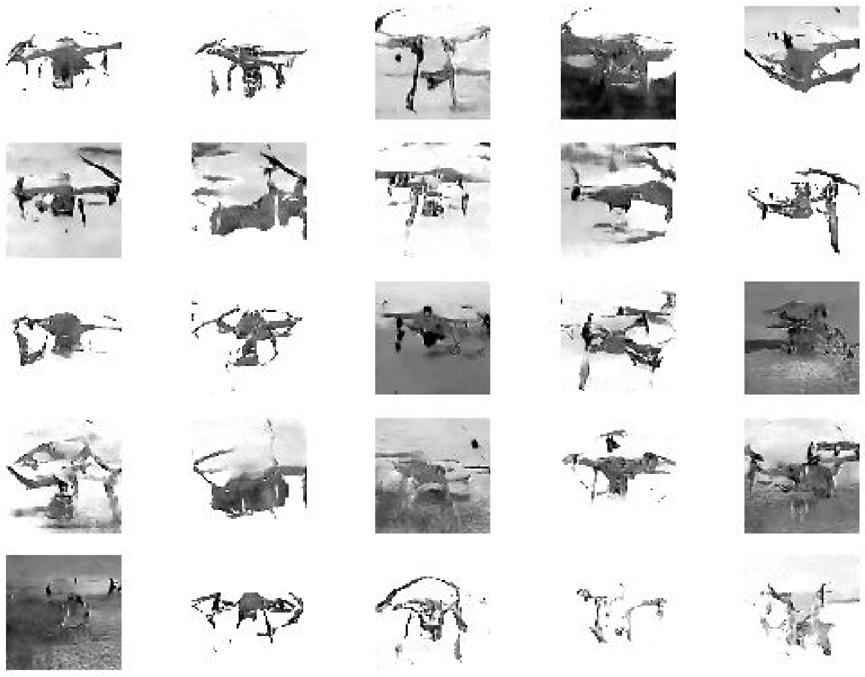
- The number of training images was really small. Having more images will improve the result.
- The training images are very random. It will be better to run some pre-processing of images -- such as, classify images by the size of drones, colors, etc. Also, background should be removed.
- For improving speed, techniques such as "*pyramid-image-search*" might be useful.
- Also, using gray scale image might be a good idea. It is faster to process and there is another kind of GAN to colorize images [9]. Instead of generating color images directly, two-step approach might work better in some cases. In the appendix at the end of this report, I added sample of gray images using the same program.

Conclusions and Future Work

In this project, I generated drone images using DCGAN. There are lots of rooms for improvements. Also newer GAN algorithms have been announced and some of them generates photo realistic images [10][11]. I would like to try those and ultimately test them out by using those images as the training set for our company's Classifier product.

Appendix.1

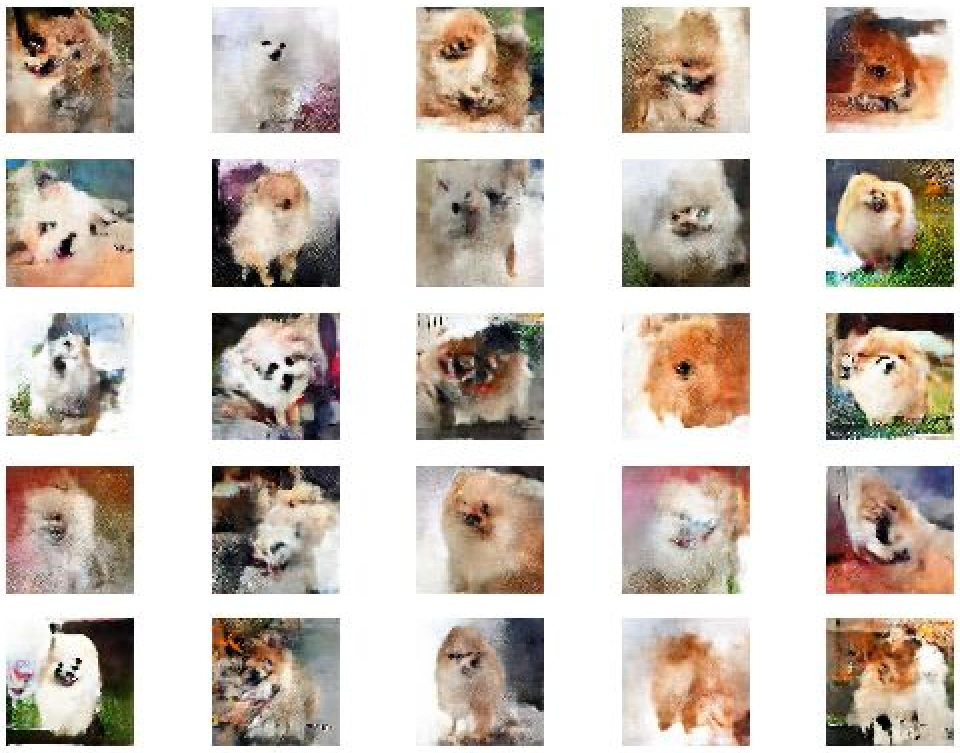
Below are the images generated with the same code but using gray scale. It is difficult to say if these are better than the color version. Color version has mixed colors from different training images and that made them look bad. Using the gray scale image, then colorize afterwards might look better.



Appendix. 2

Drones consist of artificial straight lines and curves, and that might be actually difficult for GAN to generate. So I wanted to apply the same code for different kind of image set, something more fuzzy and obscure. Here, images generated using Pomeranian images (including my dog). Arguably, they look more realistic than drones.

Figure: pomeranians (~70,000 iterations)



References

[1] Generative Adversarial Networks

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio

[2] Generative Adversarial Networks (GANs) — A Beginner's Guide

https://towardsdatascience.com/generative-adversarial-networks-gans-a-beginners-guide-5b38eceece24

[3] Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

Alec Radford, Luke Metz, Soumith Chintala

[4] Image Collector https://github.com/nazboost/image-collector/blob/master/image_collector.py

[5] Deep Learning with Python https://github.com/FelixMohr/Deep-learning-with-Python

[6] Goldesel23 / DCGAN-for-Bird-Generation

https://github.com/Goldesel23/DCGAN-for-Bird-Generation

[7] DCGAN https://github.com/triwave33/GAN/tree/master/GAN/dcgan

[8] Killme_DCGAN https://github.com/taku-buntu/Keras-DCGAN-killmebaby

[9] Image Colorization with Generative Adversarial Networks

https://github.com/ImagingLab/Colorizing-with-GANs

[10] Progressive Growing of GANs for Improved Quality, Stability, and Variation

https://research.nvidia.com/publication/2017-10_Progressive-Growing-of

[11] StyleGAN — Official TensorFlow Implementation https://github.com/NVlabs/stylegan