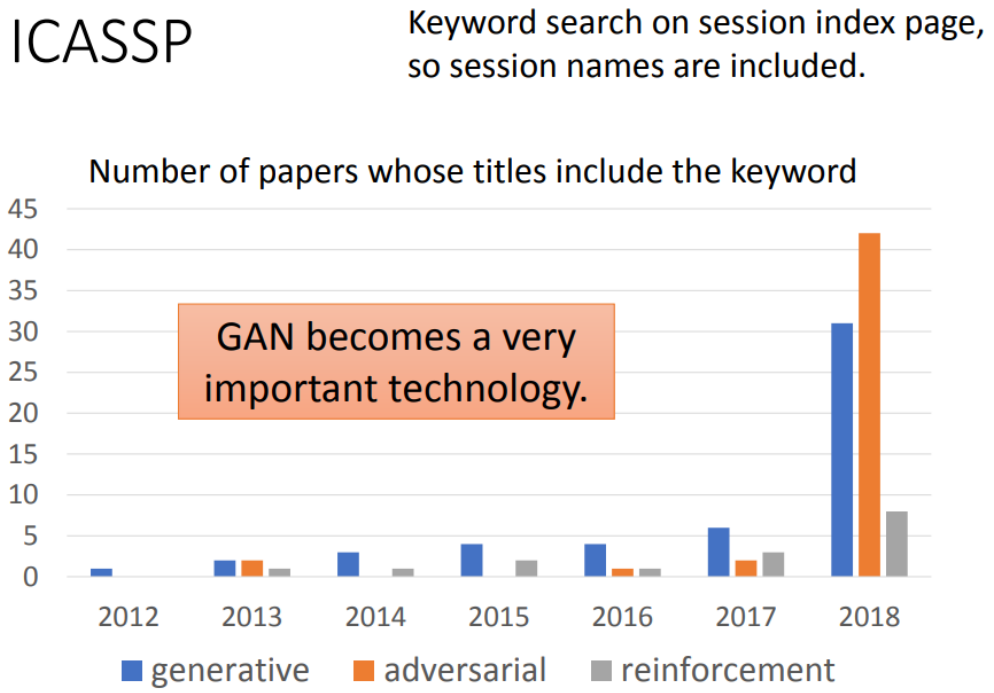
# Individual report

By Gaofeng Huang

## Introduction

Generative Adversarial Networks (GAN) is a cutting-edge technique of deep neural networks, which was first come up by Ian Goodfellow in 2014. In 2016, Yann LeCun, who is one of the leading scientists in AI, described GAN as “the coolest idea in machine learning in the last twenty years.”




GAN is a very new stuff and has a promising future. Especially in last year (2018), GAN was developed with an exponential increment. In other words, it is almost an infant technology. Although it is really new, there are bunch of models named with the suffix __GAN, such as Conditional GAN, DCGAN, Cycle GAN, Stack GAN. Fortunately, we can catch up the development of GAN now. Actually, we’ve learned every single component in GAN if I break down it.

## Individual work

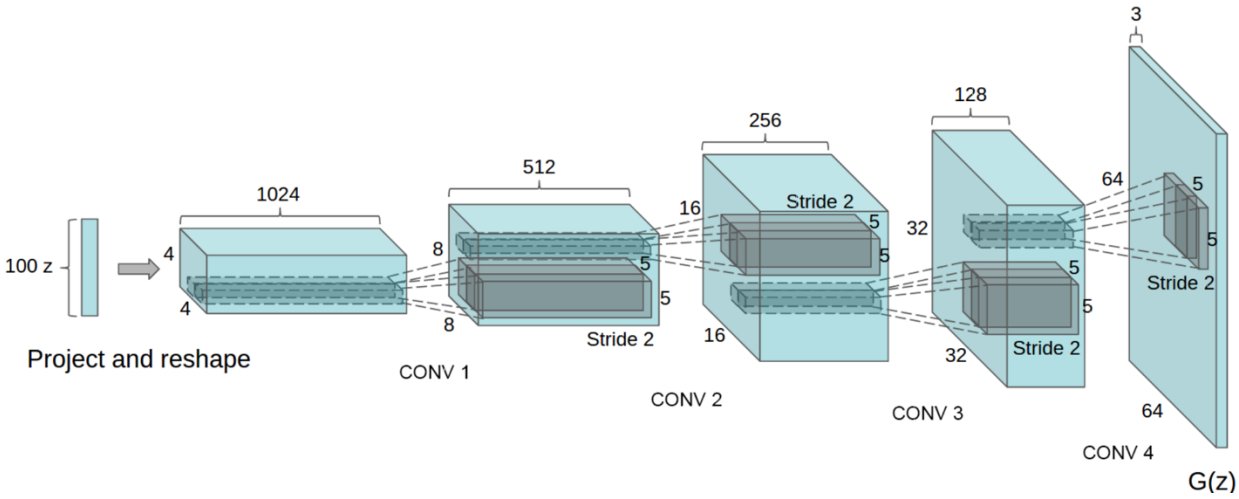
I’ve worked on

1. Searching datasets and topics.
2. GAN (DCGAN) model construction.
3. CGAN model improvement.

# 4.1, GAN

## 4.1.1, Generator

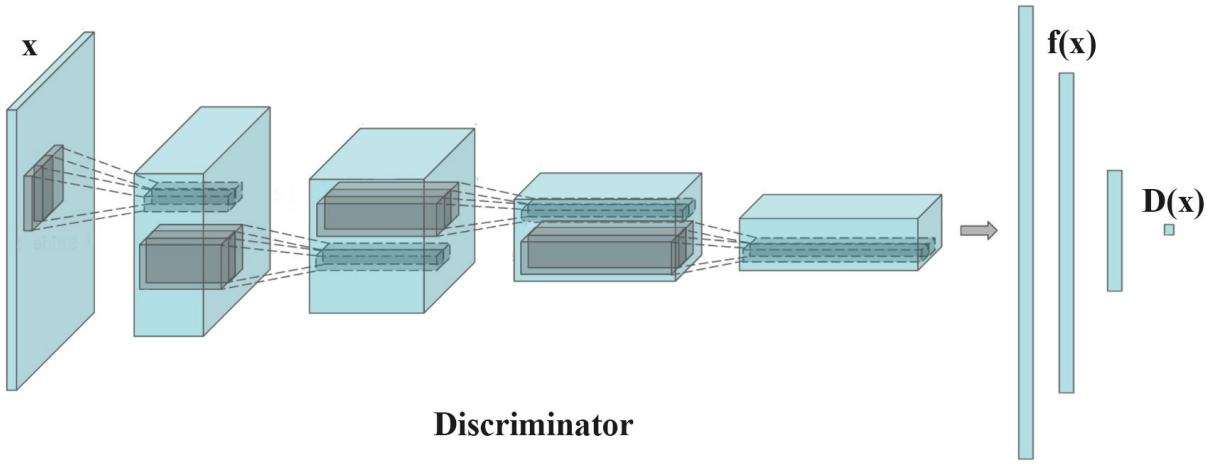
Similar to the Variational Auto Encoder (VAE), Generator is more like the decoder part. As you feed the generator with a vector or matrix, the Generator will return a new image. From a vector to an image, upsampling methods are always applied into the Generator.




(Source: ICLR 2016)

## 4.1.2, Discriminator

Similar to the regular neuron networks (MLP, CNN), Discriminator works as a classifier which only needs to distinguish an image whether it is real or fake. Downsampling methods need to be applied into the Discriminator.




### 4.1.3, Training step

Training a GAN cannot be shown obviously in this flow chart. Remember this network is a minimax game. There will be a competition between Generator and Discriminator.

### 4.1.4, Training Discriminator

First, initializing all weights in this network. Second, using initialized Generator to get a group of fake images by some random noises. Third, the real images randomly chosen from the original dataset are labelled as class 1 and fake images created by Generator are labelled as class 0. Fourth, training Discriminator to classify these combined images and updating the weights of Discriminator.

### 4.1.5, Training Generator

Before training Generator, let the weights fixed in Discriminator. First, put a group of random noises into Generator to get fake images. Second, put these fake images into the trained Discriminator to see the probability they are real images. Third, to train the Generator, the target of input noise should be class 1. Thus, using the target 1 to calculate loss and gradient so that the weights of Generator will be trained.

# 5, Results

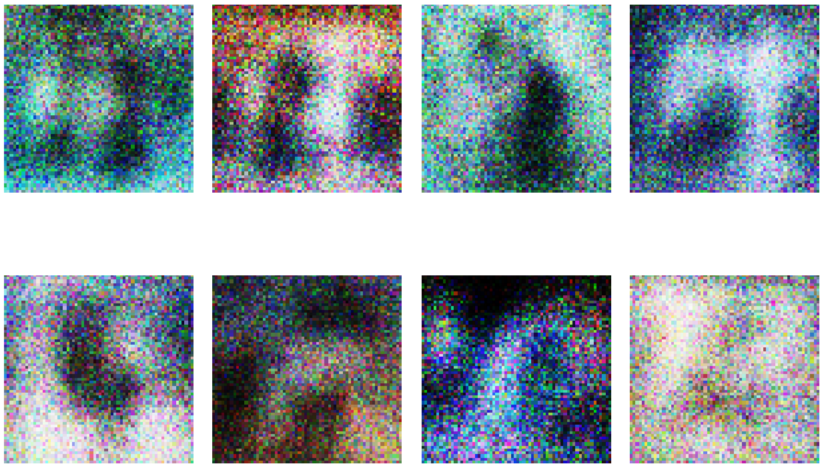
## 5.1 Simple GAN

### 5.1.1, Simple GAN with MLP

At the beginning, we play with the MLP-like networks. The idea of upsampling and downsampling is quite simple, where using increasing number of neurons in Generator and decreasing number of neurons in Discriminator.
Here is the architecture summary.

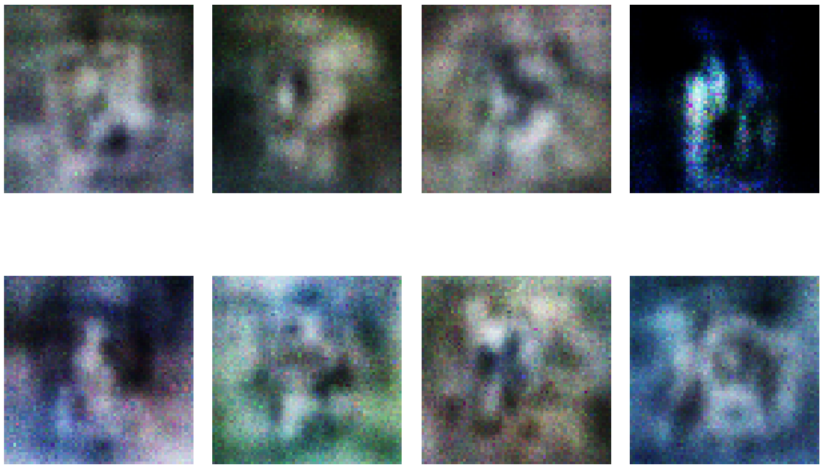
| **Input noise vector z** $\in \mathbb{R}^{100}$ | **Input RGB image** $\in \mathbb{R}^{64\times64\times3}$ |
|---|---|
| Dense, 256 + BN + LReLU | Flatten + Dense, 512 + LReLU |
| Dense, 512 + BN + LReLU | Dense, 256 + LReLU |
| Dense, 1024 + BN + LReLU | Dense, 128 + LReLU |
| Dense + Reshape, 64 x 64 x 3 + Tanh | Dense, 1 + Sigmoid |
| Output fake images $\in \mathbb{R}^{64\times64\times3}$ | Output probability $\in \mathbb{R}^{1}$ |
| **Generator** | **Discriminator** |

To be noted, the activation function on the output layer of Generator is hyperbolic tangent function, which can deal with gradient vanishing to some extent. The activation function on the output layer of Discriminator is sigmoid function, which converts the value into a probability.
Here are some results by this simple constructed GAN.



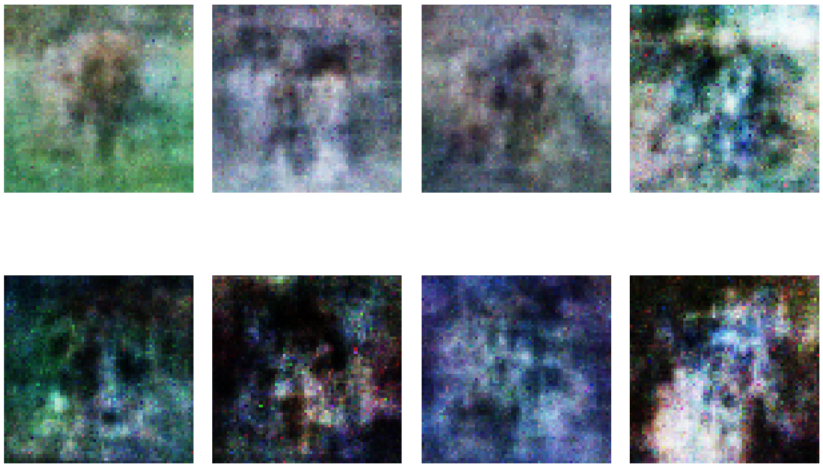
After 1000 epochs

From the early result of this GAN, it can sketch a rough shape at the center of the image. However, it is obvious that there are a lot of noises.



After 3000 epochs

After 3000 epochs, it learns to deal with the problem of noise. However, the generated images are still very blurred.



After 10000 epochs

10000-epoch is a long period for this GAN to improve its performance. As shown above, the result looks like the previous 3000-epoch result. It means this GAN cannot learn anymore from this dataset. If we look at the losses of Generator and Discriminator, the Discriminator has lower loss and it can distinguish the generated images as fake easily. Actually, the Generator is almost stop learning after 3500 epochs.

```
Epoch (80 / 100): [Loss_D_real: 0.299854, Loss_D_fake: 0.289743, acc: 88.28%] [Loss_G: 1.493132]
Epoch (90 / 100): [Loss_D_real: 0.373671, Loss_D_fake: 0.343164, acc: 90.62%] [Loss_G: 1.527919]
Epoch (100 / 100): [Loss_D_real: 0.322785, Loss_D_fake: 0.262252, acc: 89.06%] [Loss_G: 1.751898]
LEARNING STEP #  39 ------------------------------------------------
Epoch (10 / 100): [Loss_D_real: 0.273977, Loss_D_fake: 0.295783, acc: 92.97%] [Loss_G: 1.745354]
Epoch (20 / 100): [Loss_D_real: 0.303272, Loss_D_fake: 0.337140, acc: 89.06%] [Loss_G: 1.479339]
Epoch (30 / 100): [Loss_D_real: 0.316251, Loss_D_fake: 0.365537, acc: 89.06%] [Loss_G: 1.597144]
```

Generator does not learn so much

## 5.1.2, Problems in training and potential solutions

The rationale behind the GAN is easy to understand. However, in practice, GANs are always unstable to train. In order to stabilize the training step of GANs, this part will introduce some techniques proposed in recent work.

**Problems**

1. *Imbalance between Generator and Discriminator.* Generator always cannot compete with Discriminator. Intuitively, creativity is more difficult than criticism. In fact, it tends to be easy to distinguish an artwork is real or fake. However, without seeing the real artwork, it is really hard to create a fake artwork which looks just like the real one. Mathematically, the gradient descent of Generator will be vanishing.

2. *Mode collapse.* Generator only produces low-diversity outputs. A complete mode collapse, which is not common, means the Generator just makes a trick to create only one type of image to fool the Discriminator. A partial mode collapse, which always happens, is a hard problem in GAN to solve.
3. *High sensitivity to hyperparameter.* Tuning a GAN should be very patient.
4. *Non-convergence for training.* The performance of Generator and Discriminator are oscillated. Harder to tune the model.

**Solutions**

1. For *Problem 1*. A trivial approach is adjusting training times of Generator and Discriminator separately. In practice, it helps to some extent, but it also makes the training process more unstable. Besides, we can create a deeper Generator structure (DCGAN). In this project, we've tried a deeper Generator by DCGAN and got some significant improvement. There are also some other approaches we did not discuss in this project, such as spectral normalization, finding a loss function and an activation function with stable and non-vanishing gradients.
2. For *Problem 2*. Mode collapse is still a difficult problem in most GANs. Nevertheless, there are some tricky ways to disperse kind of collapse like Conditional GAN (CGAN). CGAN just uses the label of the real data. Here, our dog dataset has 120 kind of dogs. At least, we can get 120 generated dogs in CGAN.
3. For *Problem 3*. Every GAN model has its own preference to tune its parameters. Up to now, there is no way to get a specific model without so much tuning. Besides, looking at the gradients and loss changes is the most efficient way to help with tuning. After that, the only thing we should keep is our patience.
4. For *Problem 4*. In nature, this problem is similar to *Problem 1*. Using soft and noisy labels can help GANs to be stable a lot. Without such changes, our model cannot create a clear image. Actually, soft and noisy labels let Discriminator learn in the correct direction so that Discriminator can teach Generator to learn in the correct direction. Smoothing the positive labels (like 0.9-1.0). To be noticed, we should only smooth the one-sided label, particularly the positive labels. Here is the quoted explanation from Goodfellow at NIPS 2016 Tutorial.
    a. "It is important to not smooth the labels for the fake samples. Suppose we use a target of $1 - \alpha$ for the real data and a target of $0 + \beta$ for the fake samples. When $\beta$ is zero, then smoothing by $\alpha$ does nothing but scale down the optimal value of

the discriminator. When β is nonzero, the shape of the optimal discriminator function changes. The discriminator will thus reinforce incorrect behavior in the generator; the generator will be trained either to produce samples that resemble the data or to produce samples that resemble the samples it already makes." (Goodfellow, 2016)

The simple GAN model can only perform good in simple and well-preprocessed datasets, such as Mnist dataset and FashionMnist dataset. However, our dataset has RGB three channels and non-trivial background. In this project, we will use two improved GAN models, DCGAN and CGAN.

# 5.2, DCGAN

## 5.2.1, Model construction

Construction of DCGAN is not just simply using CNN in both networks. Performance of GANs are sensitive to the architecture and hyperparameters. Based on the exploration of past work, here are some guidelines when constructing a DCGAN.

1. Replace all max pooling with convolutional stride.
2. Use transposed convolution for upsampling.
    a. As we know, convolution operation is a downsampling approach. On the contrary, transposed convolution operation is a reasonable upsampling approach.
3. Use Batch normalization except the output layer for the Generator and the input layer of the discriminator.
    a. BN can deal with the mode collapse problem and help to create a deeper network.
    b. BN can deal with poor initialization of parameters.
    c. (In training, BN helps to create sharper images only when other tuning steps are done correctly. Otherwise, BN will lead to a bad result.)
4. Use LeakyReLU instead of ReLU except for the output which uses tanh (Generator), sigmoid (Discriminator).
    a. LeakyReLU is an advanced ReLU activation, which allows the pass of negative cases with a small value.

b. In backpropagation, Discriminator flows stronger gradients (with negative gradients) to Generator.

Here are also some tips for tuning from the DCGAN paper.

1. Trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128.
2. All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02.
3. In the LeakyReLU, the slope of the leak was set to 0.2 in all models.
4. If we want to use momentum to accelerate training. Using Adam(lr=0.0002, β1=0.5) is better for most cases.

Here is the **final architecture** summary.

| **Input noise vector z** $\in \mathbb{R}^{100}$ | **Out size** |
|---|---|
| Dense + Reshape, 4 x 4 x 128 + LReLU | 4 x 4 x 128 |
| 4 x 4 TranspConv + Stride(2, 2), 128 + LReLU | 8 x 8 x 128 |
| 4 x 4 TranspConv + Stride(2, 2), 128 + LReLU | 16 x 16 x 128 |
| 4 x 4 TranspConv + Stride(2, 2), 128 + LReLU | 32 x 32 x 128 |
| 4 x 4 TranspConv + Stride(2, 2), 128 + LReLU | 64 x 64 x 128 |
| 8 x 8 Conv, 64 x 64 x 3 + Tanh | 64 x 64 x 3 |
| Output fake images $\in \mathbb{R}^{64\times64\times3}$ | |

**Generator**

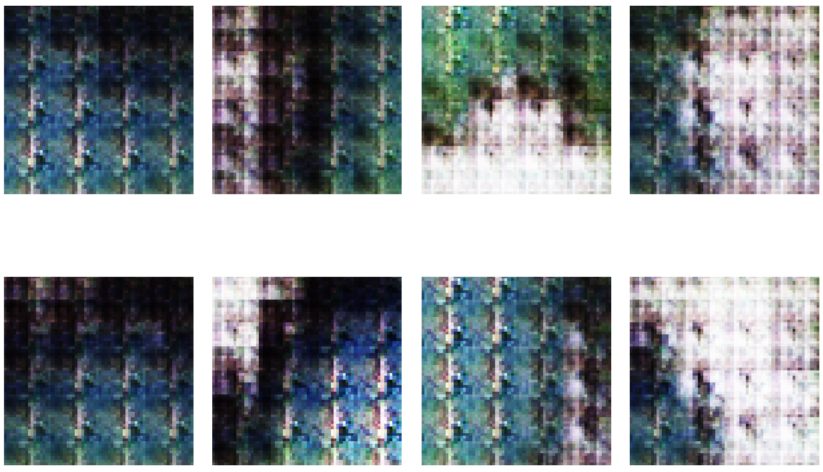
| **Input RGB image** $\in \mathbb{R}^{64\times64\times3}$ | **Out size** |
|---|---|
| 3 x 3 Conv + Stride(2, 2), 128 + LReLU | 32 x 32 x 128 |
| 3 x 3 Conv + Stride(2, 2), 128 + LReLU | 16 x 16 x 128 |

| | |
|---|---|
| 3 x 3 Conv + Stride(2, 2), 128 + LReLU | 8 x 8 x 128 |
| 3 x 3 Conv + Stride(2, 2), 128 + LReLU | 4 x 4 x 128 |
| 3 x 3 Conv + Stride(2, 2), 128 + LReLU | 2 x 2 x 128 |
| Flatten + Dropout | 512 |
| Dense, 1 + Sigmoid | 1 |
| Output probability ∈ $\mathbb{R}^1$ | |

**Discriminator**

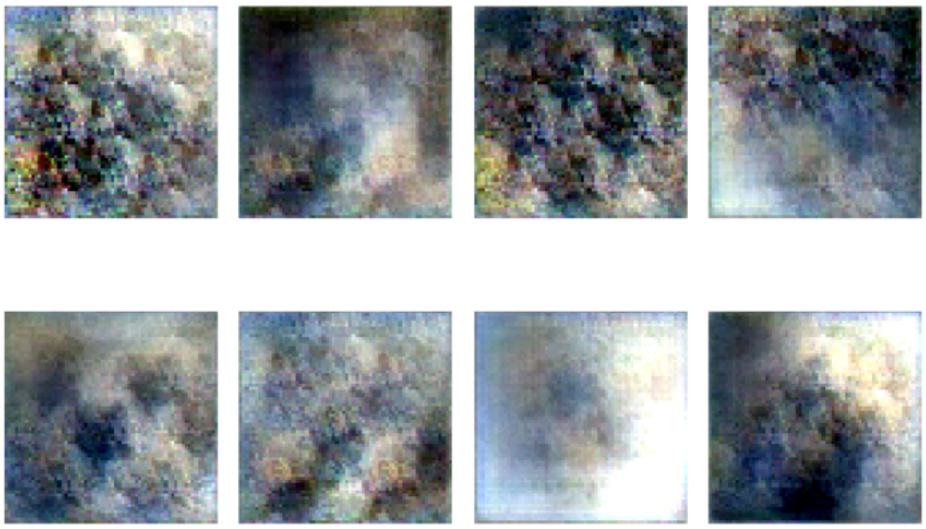
## 5.2.2, Model Performance

**Previous results (not the final structure)**



CNN creates blocks in images

Comparing with MLP structure, a well-tuned deep convolution structure helps the networks recognize the more details inside an image. However, if the convolution structure is not tuned well, the result will be worse than simple GAN with MLP. With the operation of convolution, the generated images may be cut in blocks.



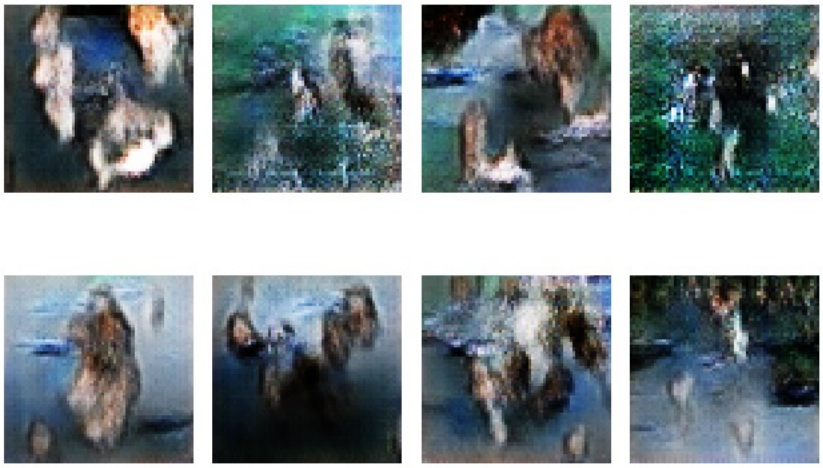
Fail to learn to generate dog-like images

Even if the blocks problem is mitigated, DCGAN may fail to generate target-like images. It means both of Discriminator and Generator are learning in the wrong directions. Although the image quality is improved in some cases, the mode collapse problem is still existed in DCGAN.

Extreme case

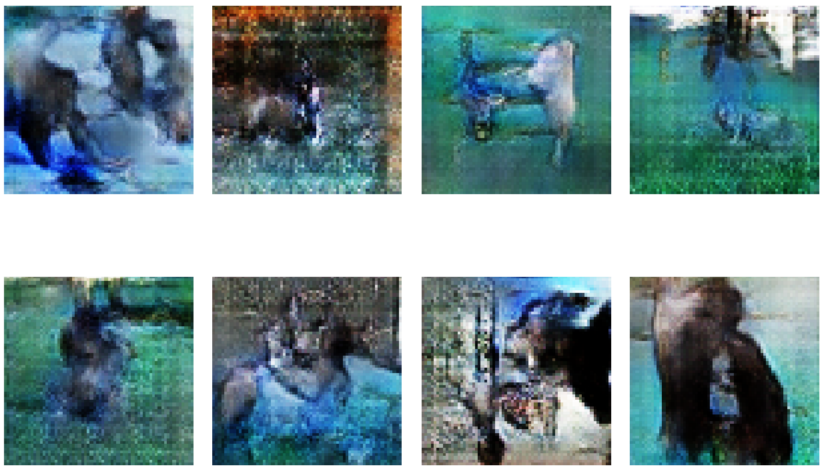
Tuning a DCGAN is much more complicated than simple GAN with MLP. There are two deep CNN participant this competition. Therefore, every small change of one network will affect the performance of another network and then influence the final result of the whole model. Indeed, it took 90% work of this project to tune the GANs. Some ways suggested form other papers or projects always cannot help to improve the GAN model in our project.

**Final tuned model results**



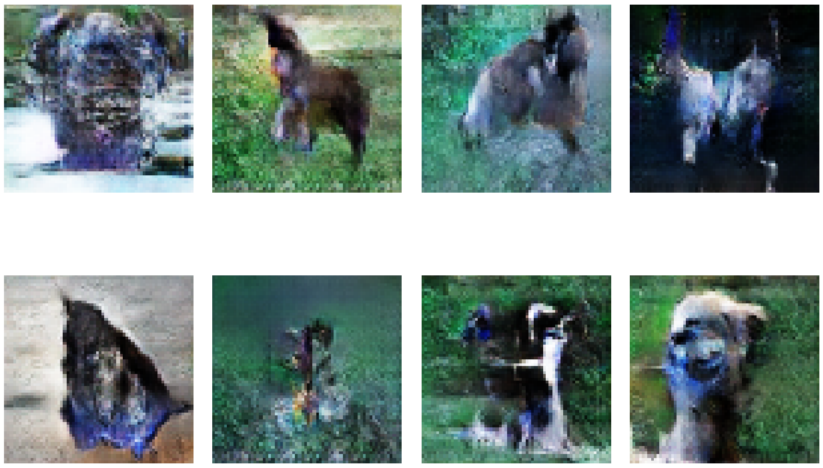
After 5000 epochs

In the early training step, it can generate some clear image but not dog-like.

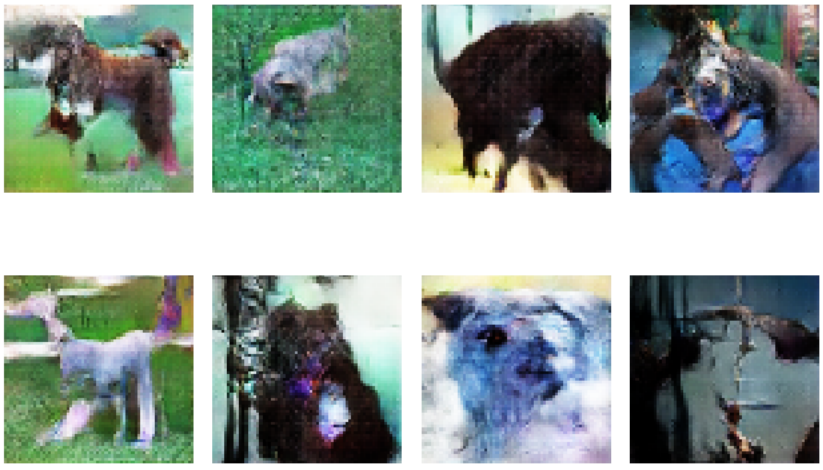


After 10000 epochs

Some particular images generated look like a dog, but most of them are still not dog-like.

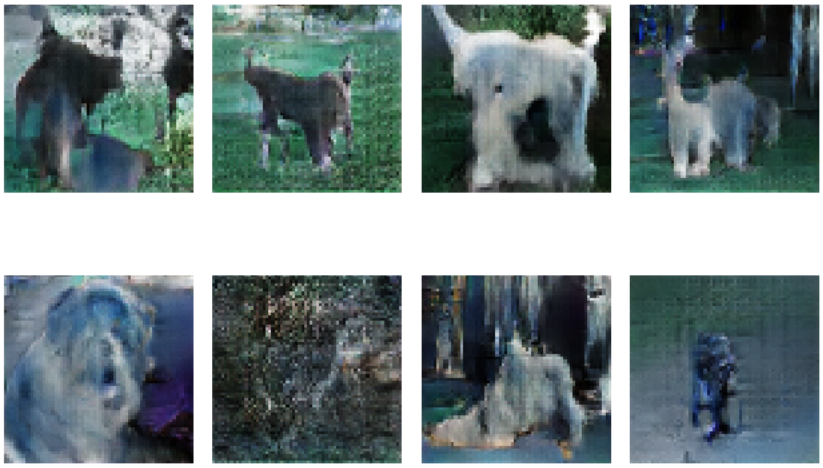


After 20000 epochs



After 40000 epochs

After 40000 epochs, most of generated images look like a dog. Also, mode is not collapsed into a few types of dog. It is not easy to generate clear dog-like images without mode collapse.



After 60000 epochs

After 60000 epochs, the improvement is not significant. Hence, this DCGAN model almost achieves its maximum performance. Up to now, this is our best DCGAN model. However, imaging in the performance surface, our model only stays at the local-best performance. Therefore, it still needs to be tuned with much patience.

## Code

All version of Dog GAN, one version of CGAN.
Over 90% codes are written by myself.

## Summary

At the beginning, we all thought it is impossible to create a GAN model. After learning the structure of GAN and the example codes, I was trying to construct our own model. The model construction just took several days. However, tuning the model is much harder than our expectation. Up to now, my final model is still not very satisfied. One thing I am curious that most tuning methods cannot work in my model.
After this tempt of GAN model, I have a huge interest in this kind of technique. Simple GAN or CGAN model are far from using in real life. Next step, I will try Cycle GAN to generate stuff from domain X to stuffs in domain Y. For example, transforming a cat to a dog and generating paintings just like Van Gogh draws. I am also so proud that I could construct GAN models after this project.

# Reference Links

Transposed convolutional layer
https://datascience.stackexchange.com/questions/6107/what-are-deconvolutional-layers

Upsampling, transposed conv2D
https://medium.com/activating-robotic-minds/up-sampling-with-transposed-convolution-9ae4f2df52d0

GAN-intro
https://www.kaggle.com/jesucristo/gan-introduction

Dog memorizer GAN
https://www.kaggle.com/cdeotte/dog-memorizer-gan

Dog VAE
https://www.kaggle.com/cdeotte/dog-autoencoder

VAE PyTorch
https://towardsdatascience.com/vaes-generating-images-with-tensorflow-61de08e82f1f

Beginner GAN
https://skymind.ai/wiki/generative-adversarial-network-gan

DC GAN
https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

Dog BigGAN
https://www.kaggle.com/tikutiku/gan-dogs-starter-biggan

CNN + Pretrained + Augmentation
https://www.kaggle.com/ignotus09/image-classification-dogs-vs-cats

Why GAN is hard to train
https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

How to improve GAN
https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

How to tune a DCGAN

https://medium.com/@utk.is.here/keep-calm-and-train-a-gan-pitfalls-and-tips-on-training-generative-adversarial-networks-edd529764aa9

Conditional GAN (CGAN)
https://medium.com/@jonathan_hui/gan-cgan-infogan-using-labels-to-improve-gan-8ba4de5f9c3d

PyTorch GAN
https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html