Individual Final Report

1. Introduction

We will first introduce the data, then concepts, algorithms, and structures of our models. We hope to give you a macro understanding of what theories and materials we are based on to complete this project. Then well explain each step of implement, the problem we encountered, and how to solve it. Finally, we will explain the reasons for the deficiency and the ideas for future improvement.
CGAN was first described by Mehdi Mirza and Simon Osindero in their 2014 paper titled "Conditional Generative Adversarial Nets." In this paper, the authors inspire the method based on the desire to guide the image generation process of the generator model.

"by conditioning the model on additional information, it is possible to direct the data generation process. Such conditioning could be based on class labels"
— Conditional Generative Adversarial Nets, 2014.

2. Individual work

1. Construction of CGAN model
2. Improvement of CGAN model

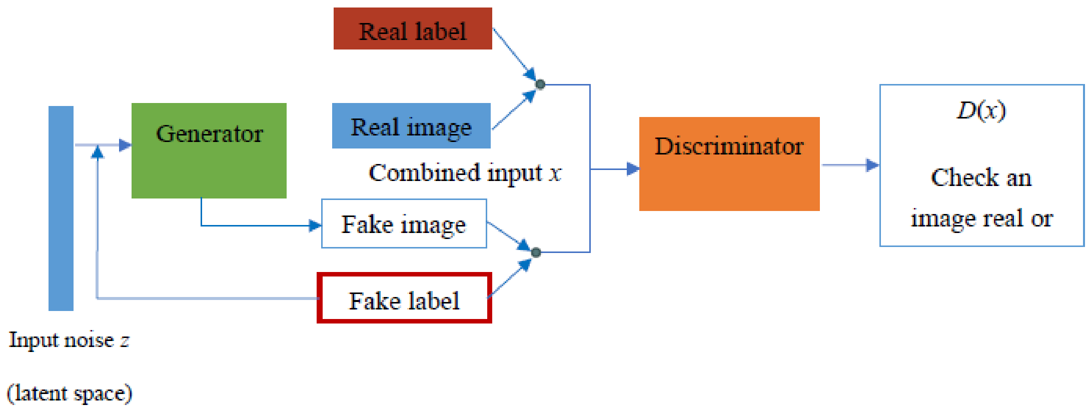
3. CGAN

3.1 Structure of CGAN

The cost function for CGAN is the same as GAN.

$$min_D max_G V(G,D) = \mathbb{E}_x \log\big(D(x,y)\big) + \mathbb{E}_z \log\big(1 - D(G(z,y_z),y_z)\big)$$

$D(x,y)$ and $G(z,y_z)$ indicate that we are distinguishing the label y and generate a picture given by the label y_z.

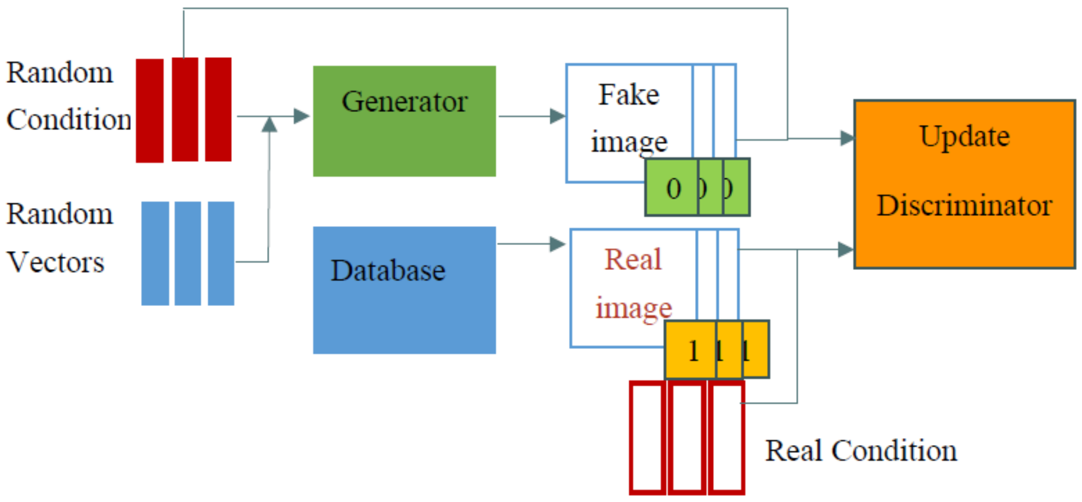


Conditional Generative Adversarial Networks (CGAN), an extension of the GAN, allows you to generate images with specific conditions or attributes. Same as GAN, CGAN also has a generator and a discriminator. However, the difference is that both the generator and the discriminator of CGAN receive some extra conditional information, such as the class of the image, a graph, some words, or a sentence. Because of that, CGAN can make generator to generate different types of images, which will prevent generator from generating similar images after multiple trainings. Also, we can control the generator to generate an image which will have some properties we want.

3.2 Algorithm

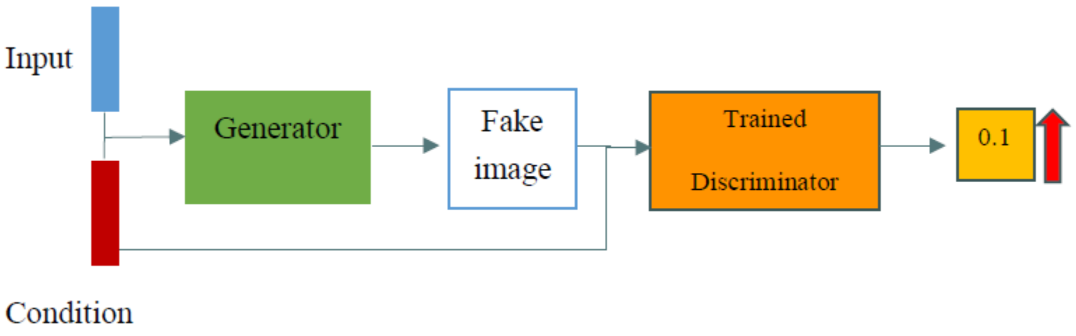
- Initialize the Generator and Discriminator.
- Single train iteration:

1. Fix the generator, then input the random vectors and random condition (category, image or description) into the generator to get the generated images. In order to train the generator to generate the specific images we want; we need to pull the sample images from the database. Then update the discriminator.
2. To update the discriminator, we have to adjust the parameter of it. For example, we can label the real images as 1, and generated images as 0. In addition, we also need to input the condition of the same attribute of images when we input the real images to discriminator. For fake images, we need to input the same condition used when generating them. After updating, the discriminator is supposed to classify the real and fake images well. We can regard this problem as a classification problem and training the discriminator as training a classifier.



3. After training discriminator, we fix it, and adjust the parameters of generator. The goal of generator training is to generate the image that

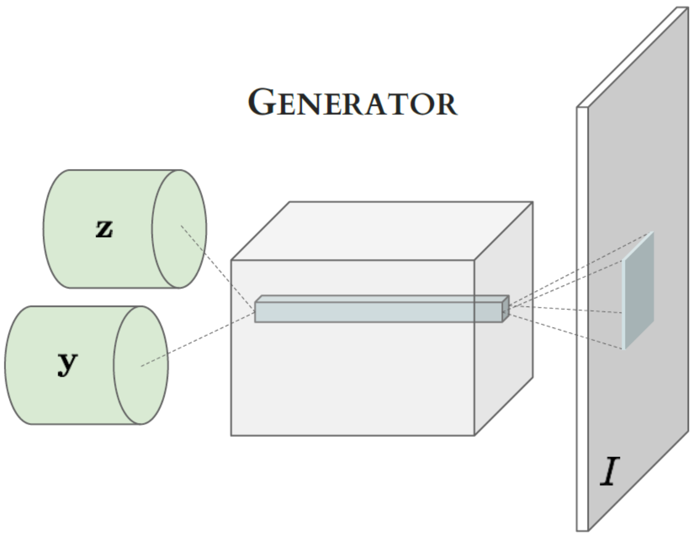
discriminator can grade it close to 1. This step is similar to the optimizer in the neural network that we learned about, but its a gradient ascent process.



4. When we put these two processes together, its a whole big network. As what we showed at the structure introduction. We put in the vector and the condition, then we end up with a number. But if we extract the output from the hidden layer in the middle, we get a complete image.

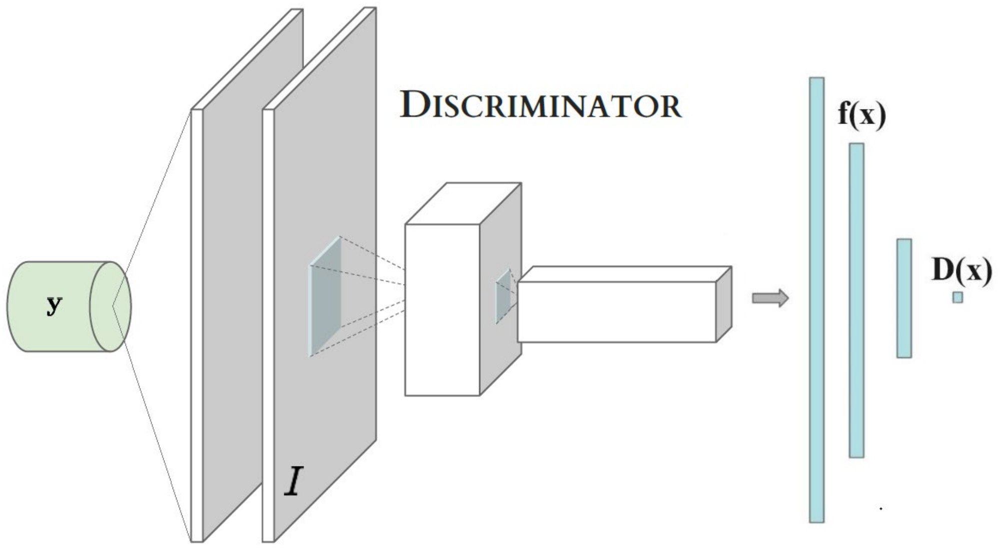
3.3 Generator

Similar to the DCGAN, the Generator will return an image when you feed the generator with a vector or matrix which combine with a condition information.



3.4 Discriminator

The CGAN Discriminator is similar to the DCGAN Discriminator. However, this Discriminator expands the condition to the size of the images and then merges it with the images to form the input. It is only necessary to distinguish whether the image is real or fake.



3.5Training

The training steps, Generator training, and Discriminator training are the same as DCGAN. Only in the input, the condition needs to be merged into Generator or Discriminator.

4. Result

4.1 Model construction

The construction of CGAN is roughly the same as the construction of DCGAN, so only the differences will be explained here.

1. The input of Generator: we have to merge the class and the noise. Thus, we embed the class, and then reshape it to the same dimension as noise.
2. The input of Discriminator: we have to merge the class with the image. Similar to the input of Generator, we embed the class, and then reshape it to the size of image.

Here is the **final architecture** summary.

Input noise vector z $\in \mathbb{R}^{100}$	**Out size**	**Input label (0-119)**	**Out size**
Dense + Reshape,4 x 4 x 256 + LReLU	4 x 4 x 256	Embedding + Dense + Reshape, 4 x 4	4 x 4 x 1
Merge input noise vector and input label			**Out size**
Concatenate			4 x 4 x 257
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

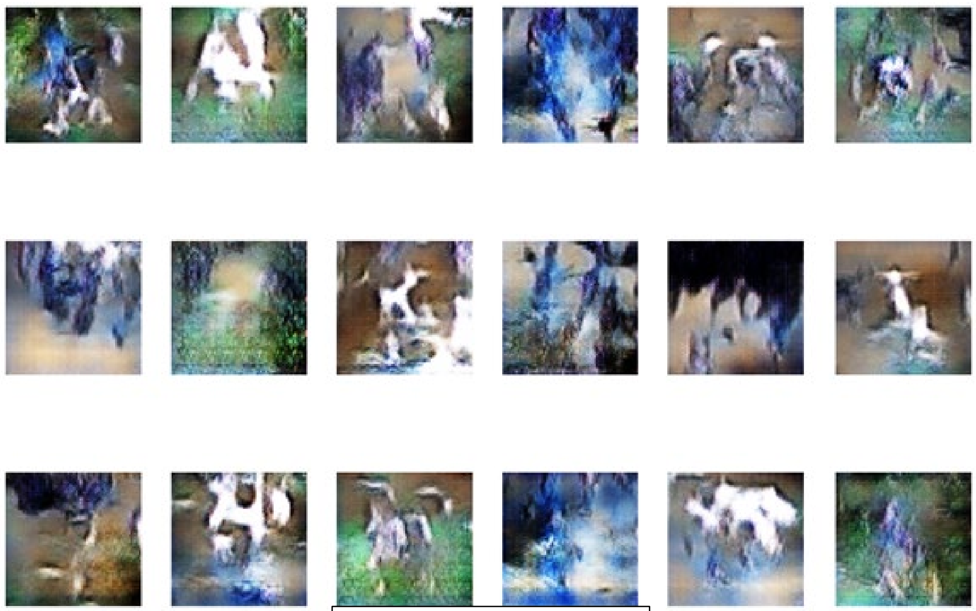
Input RGB image $\in \mathbb{R}^{64\times64\times3}$	**Out size**	**Input label (0-119)**	**Out size**
-	64 x 64 x 3	Embedding + Dense + Reshape, 64 x 64	64 x 64 x 1
Merge input RGB image and input label			**Out size**
Concatenate			64 x 64 x 4
3 x 3 Conv + Stride(2, 2), 128 + LReLU			32 x 32 x 128
3 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 $\in \mathbb{R}^1$	

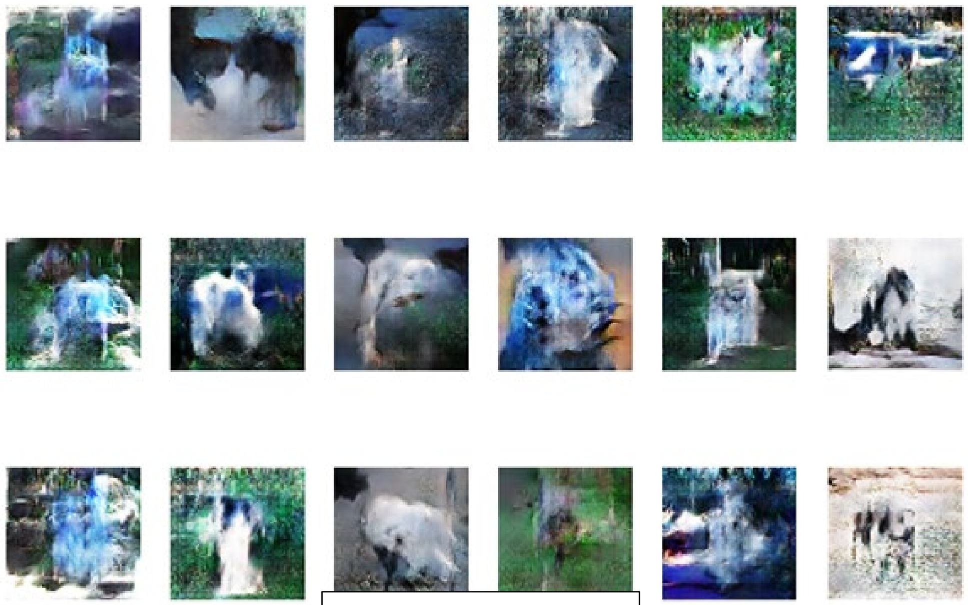
Discriminator

4.2 Model Performance

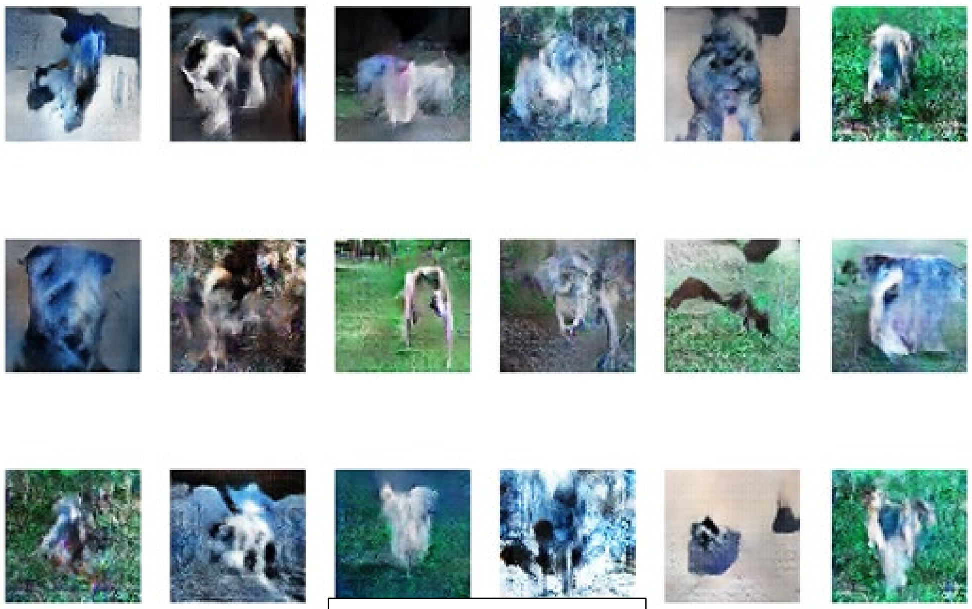
In the early training step, it can generate some clear image but not dog-like and no type.

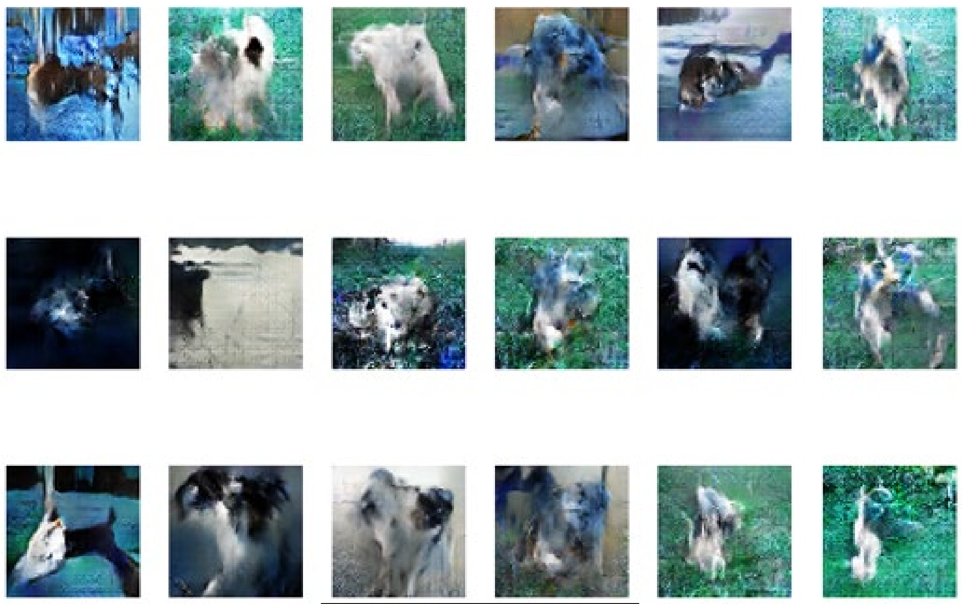

6,000 Epochs

Although still a bunch of things, some have the shape of a dog. In some columns, there are similar types, like column 3 and 4.

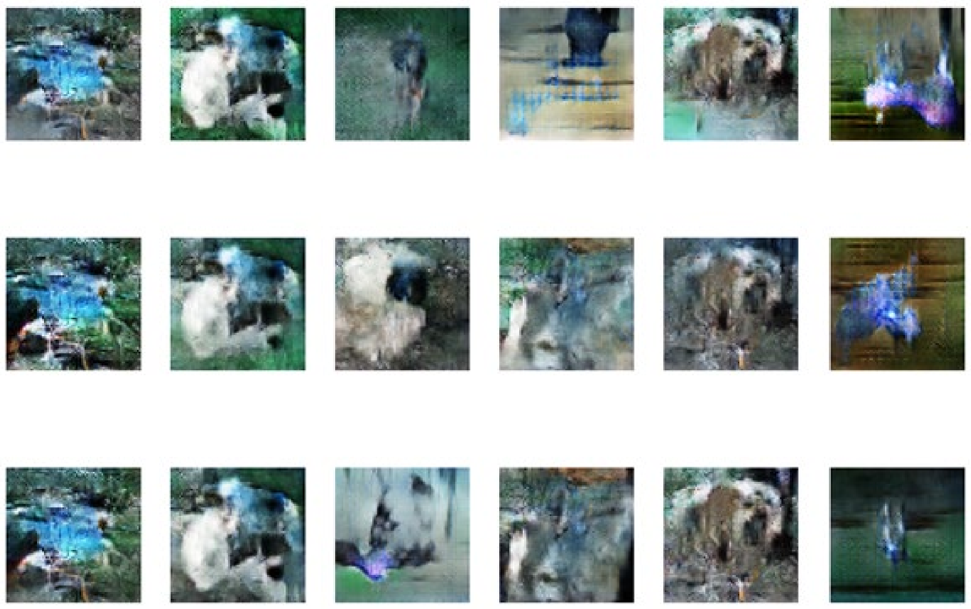

15,000 Epochs

We find that the performance between 30,000 and 45,000 are much better. Most of images look like a dog. Each column roughly has its own type. However, there are some signs of mode collapse, such as columns 4 and 6 of 45,000 epochs.


30,000 Epochs


45,000 Epochs

After 60,000 epochs, the generator cannot further generate a more dog-like image. There is mode collapse of almost each column.


60,000 Epochs

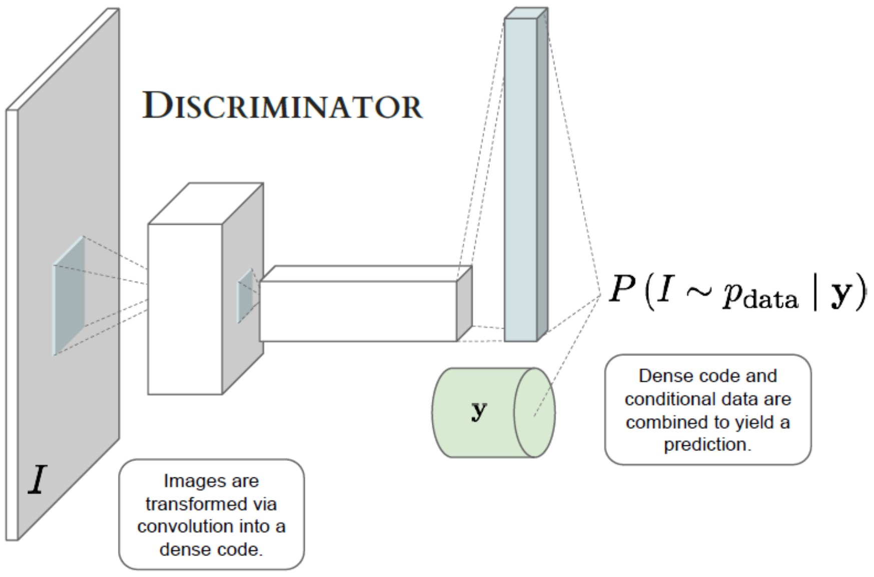
4.2 Some problem of GAN

Issue: 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.

Solutions: 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 inputs the label of one or more real data into the model as a condition, so that the model is affected by the label. Here, our dog dataset contains 120 kinds of dogs. Therefore, we try to use CGAN to solve the problem of mode collapse.

5. Summary and conclusions

In CGAN, we can extend this mechanism to include other labels that the training dataset may provide.
Through this project, I learn the contracture of GAN, and how to build a GAN model. About the CGAN model we built, we reshape the condition and add it as the fourth dimension of the image. However, the discriminator below input conditional data combine with images which via convolution into a dense code. The combined data yield a prediction of each class possibility. So, it might be interesting to try this discriminator in the future.


DISCRIMINATOR
$P(I \sim p_{\text{data}} \mid \mathbf{y})$
Dense code and conditional data are combined to yield a prediction.
y
I
Images are transformed via convolution into a dense code.

6. Calculate

The percentage of the code that you found or copied from the internet is 20%.

7. References

Avinash H. (2017). *The GAN Zoo*. GitHub. https://github.com/hindupuravinash/the-gan-zoo

Hongyi, L. (2018). *GAN Lecture 1: Introduction*. YouTube. https://www.youtube.com/watch?v=DQNNMiAP5lw&list=PLJV_el3uVTsMq6JEFPW35BCiOQTsoqwNw&index=1

Jason B. (2019). *How to Develop a Conditional GAN (cGAN) From Scratch*. Machine Learning Mastery. https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/

Jonathan H. (2018). *GAN — Ways to improve GAN performance.* Towards Data Science. https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

Jonathan H. (2018). *GAN — CGAN & InfoGAN (using labels to improve GAN).* Medium. https://medium.com/@jonathan_hui/gan-cgan-infogan-using-labels-to-improve-gan-8ba4de5f9c3d

Jonathan H. (2018). *GAN — Why it is so hard to train Generative Adversarial Networks!* Medium. https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

Jonathan H. (2018). *GAN — DCGAN (Deep convolutional generative adversarial networks)*
. Medium. https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

Jon G. (2019). *Conditional generative adversarial nets for convolutional face generation.* Stanford University.

Kaggle Competition. (2019). *Generative Dog Images*. Kaggle. https://www.kaggle.com/c/generative-dog-images

Naoki S. (2017). *Up-sampling with Transposed Convolution.* Medium. https://medium.com/activating-robotic-minds/up-sampling-with-transposed-convolution-9ae4f2df52d0

Utkarsh D. (2018). *Keep Calm and train a GAN. Pitfalls and Tips on training Generative Adversarial Networks*. Medium. https://medium.com/@utk.is.here/keep-calm-and-train-a-gan-pitfalls-and-tips-on-training-generative-adversarial-networks-edd529764aa9