Machine Learning II Project

Generate Dog Images with Generative Adversarial Networks (GANs)

Group 1: Gaofeng Huang, Jun Ying, Xi Zhang

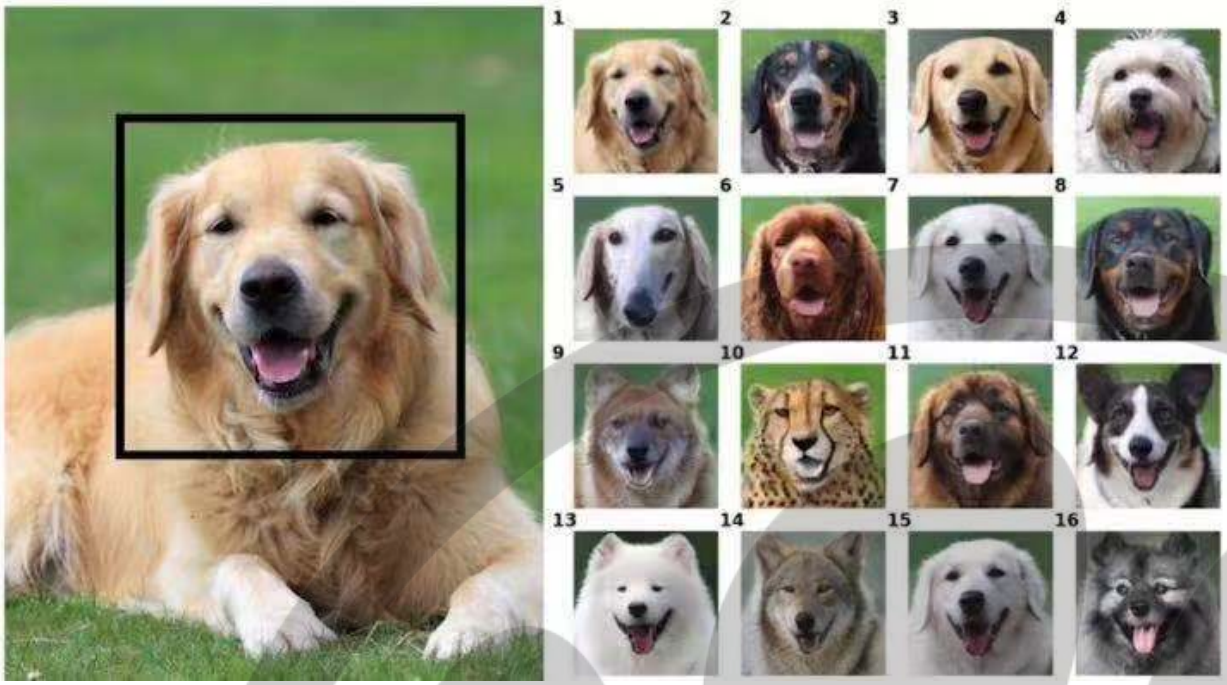
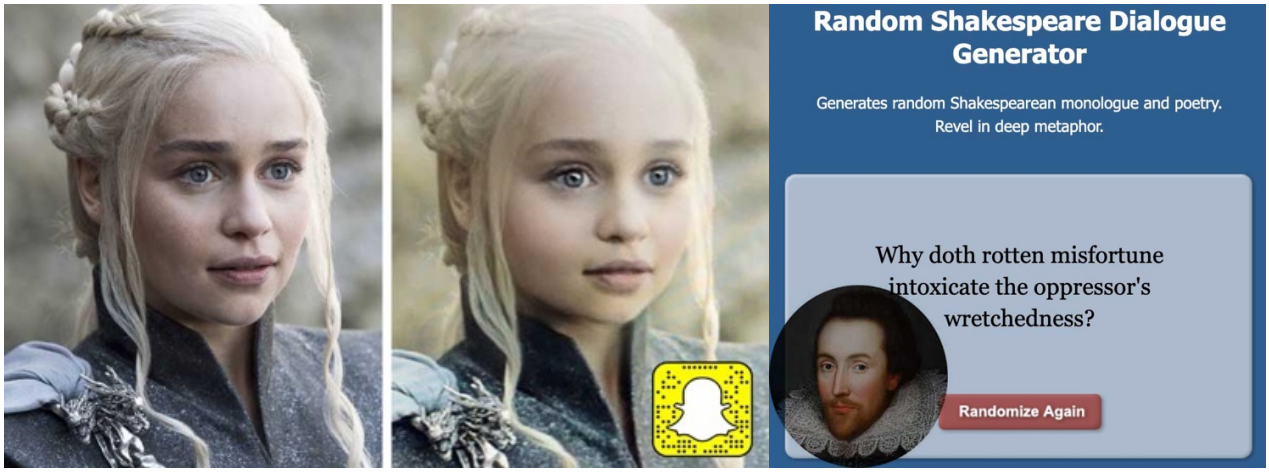


Table of Contents

1, Introduction

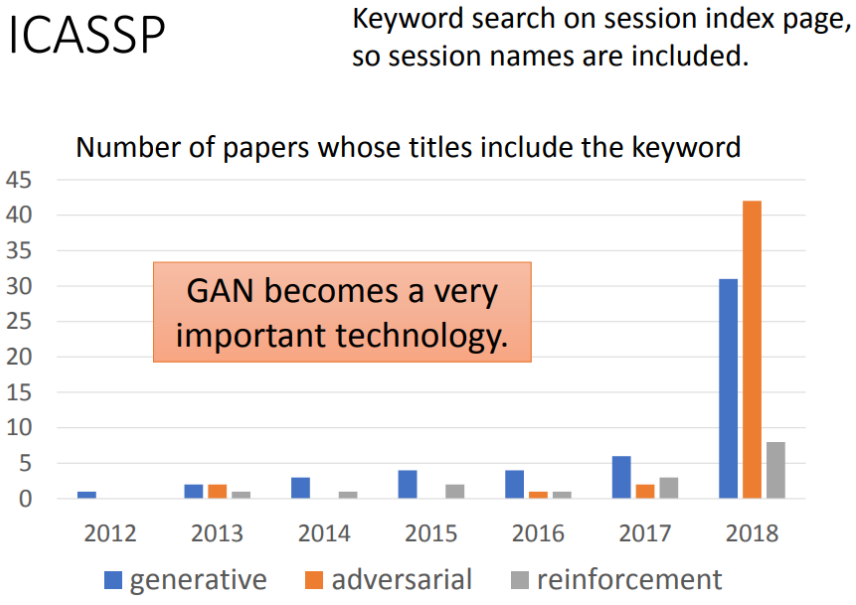
Combined with the knowledge of neural network learned in Machine Learning II, we hope to learn more in-depth and interesting knowledge on this basis. After investigation, Generative Adversarial Networks completely meets our requirements. As a new technology, GAN has been developing vigorously in recent two years. It is based on our familiar neural network such as multilayer perceptron (MLP) and convolutional neural network (CNN), but brings new vitality to machine learning.



Snapchat babyface filter and Shakespearean poetry generator, which only became popular in the last two years, also benefited from GAN's development.

1. 1, Background of GAN

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.

1.2, Outline of Our Project

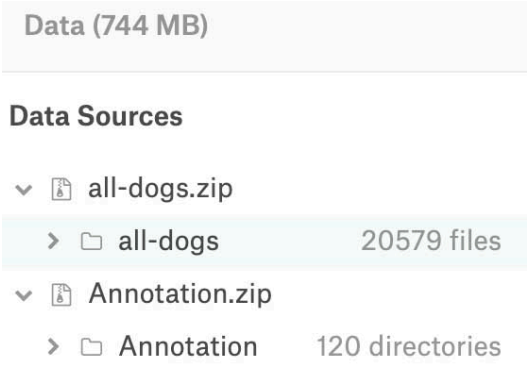
We will first introduce the data, then concepts, algorithms, and structures of our models. We hope to give you a macro understanding of what theories and materials we are based on to complete this project. Then we'll explain each step of implement, the problem we encountered, and how to solve it. Finally, we will explain the reasons for the deficiency and the ideas for future improvement.

2, Description of the Data Set

2.1, Resource

https://www.kaggle.com/c/generative-dog-images

Our data comes from the Kaggle code competition, which was released in June 2019. Since the game has ended in August, the ranking of the leaderboard also has certain reference value for our own model progress.



The open source data consists of the image archive and the annotation archive. After further study, we believe that the subfolder name of the picture package can fully perform the task of label, so our model only USES the picture compression package.

2.2, Overview of the Data Set

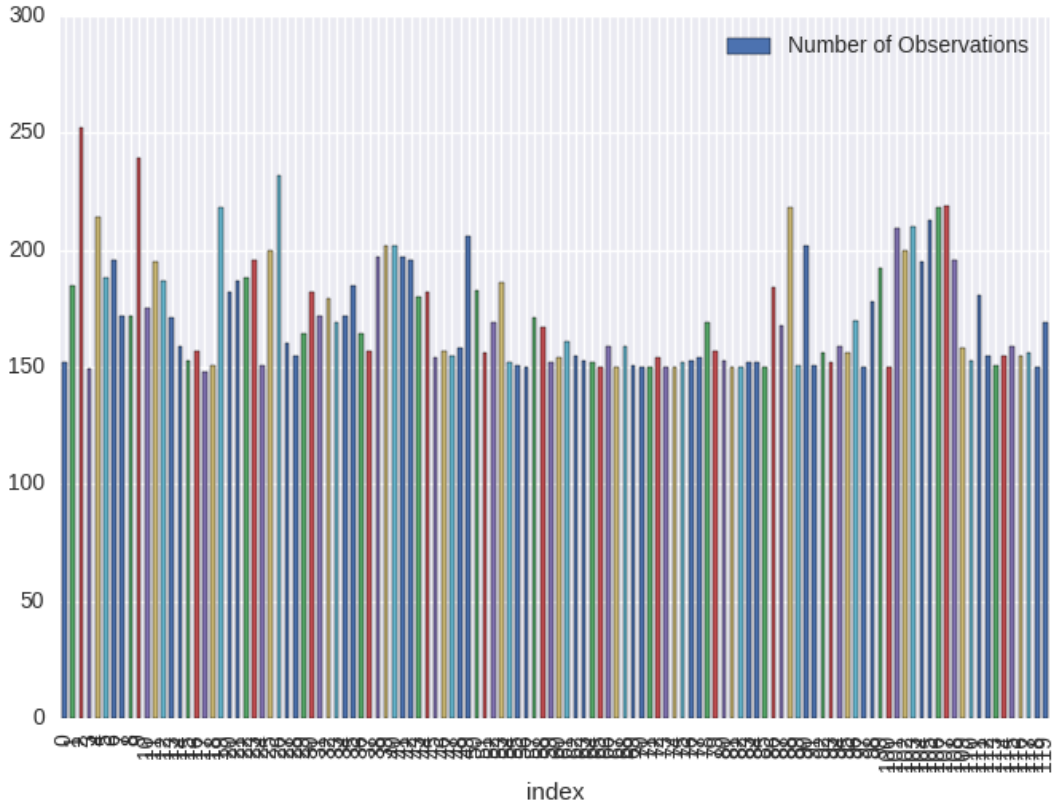
Image/label illustration; shape; balance; resized images

	Dog Breed	Number of Observations
0	n02085620-Chihuahua	152
1	n02085782-Japanese_spaniel	185
2	n02085936-Maltese_dog	252
3	n02086079-Pekinese	149
4	n02086240-Shih-Tzu	214
.	.	.
.	.	.
.	.	.
116	n02113978-Mexican_hairless	155
117	n02115641-dingo	156
118	n02115913-dhole	150
119	n02116738-African_hunting_dog	169

Our dataset contains 20,579 images belonging to 120 dog breeds.

All images are colored with RGB channels, but the size is not fixed, and the proportion of dogs (main features) in the picture is not even. We can see that there are many pictures in which the complex background is the main body. And all we want to do is generate pictures of dogs. A large number of interference features will be the challenge of our study. While the amount of data is not too small, there are too many labels, and the data for each category is not significant when evenly distributed.

To visualize the data balance, we drew a histogram to show the data distribution.



You can see that in the distribution of 120 types of data, the maximum and minimum differences are generally within 100. We do not need to worry about over-sampling or under-sampling resulting in imbalanced data.

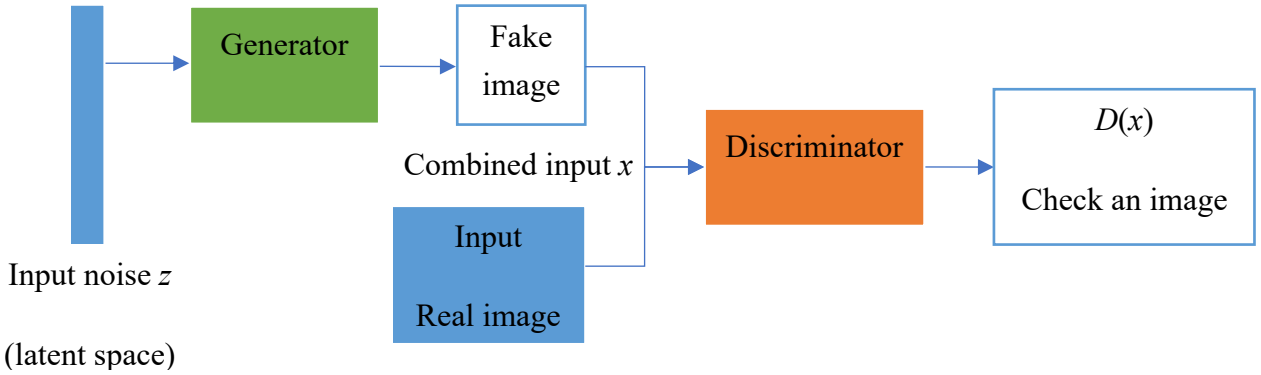
3, Description of Models

We implement GAN as our main network and try CGAN for promoting the results.

3.1, GAN

3.1.1, Structure of GAN

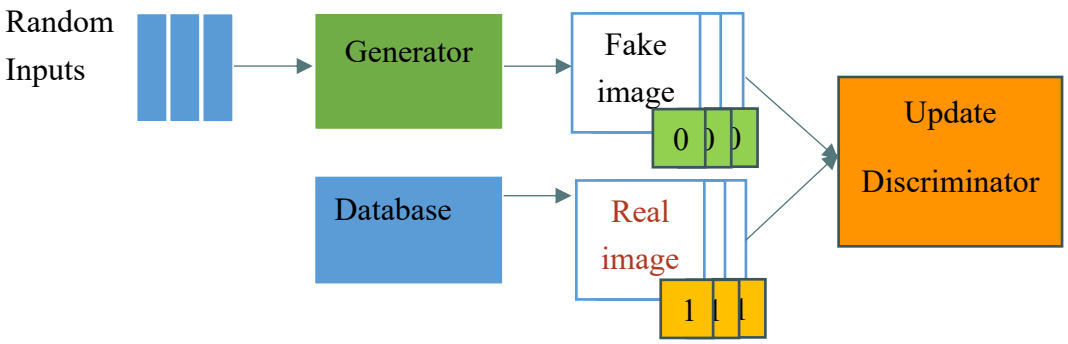
$$min_D max_G V(G,D) = \mathbb{E}_x \log\big(D(x)\big) + \mathbb{E}_z \log\Big(D\big(G(z)\big)\Big)$$



GAN is a minimax problem, which is one of zero-sum non-cooperative games. Generator wants to maximize its performance, which works to generate images as real as possible to confuse the Discriminator. Discriminator wants to distinguish a mixture of original and generated images whether real or fake. In this game, zero-sum means if Generator is improved, then there must be an increased loss of Discriminator. Our aim is to find the lowest aggregate loss of them, where there is a Nash Equilibrium.

3.1.2, Algorithm

- Initialize the Generator and Discriminator.
- Single train iteration:
    1. Fix the generator, then input the random vectors 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. 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 it's a gradient ascent process.

    4. When we put these two processes together, it's a whole big network. As what we showed at the structure introduction. We put in the vector, and we end up with a number. But if we extract the output from the hidden layer in the middle, we get a complete image. Details will be shown in an implement below.

3.1.3, Algorithms for Mathematical Contents

Initialize parameter θ_d for discriminator and parameter θ_g for generator.

In single training iteration:

Training Discriminator:

- Sample n images from database: $\{\{x^1, x^2, \dots, x^n\}\}$. 'n' is the batch size in the model.
- Sample n noise samples from a distribution: $\{\{z^1, z^2, \dots, z^n\}\}$.
- Obtain generated data $\{\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^n\}\}$, $\tilde{x}^i = G(z^i)$.
- Update discriminator parameters θ_d

  $\tilde{V} = \frac{1}{n}\sum_{i=1}^{n} logD(x^i) + \frac{1}{n}\sum_{i=1}^{n} log(1 - D(\tilde{x}^i))$

  In order to maximize the objective function, we need to maximize the $D(x^i)$. It means we need to train the discriminator to give the real image a large value. Meanwhile, we want to minimize the $(D(\tilde{x}^i))$, which means we want the discriminator to give the fake image a small value.

  $\theta_d \leftarrow \theta_d + \eta\tilde{V}(\theta_d)$ * η: $Learning\ Rate$

  We use assent gradient to update the parameter of discriminator.

Training Generator:

- Sample n noise samples from a distribution: $\{\{z^1, z^2, \dots, z^n\}\}$.
- Update discriminator parameters θ_g

  $\tilde{V} = \frac{1}{n}\sum_{i=1}^{n} \log(D(G(z^i)))$

  When we input a random vector into the generator, it generates an image. This result will be graded by the trained discriminator. Our goal is to maximize the objective function.

  $\theta_g \leftarrow \theta_g + \eta\tilde{V}(\theta_g)$

  We use assent gradient to update the parameter of generator.

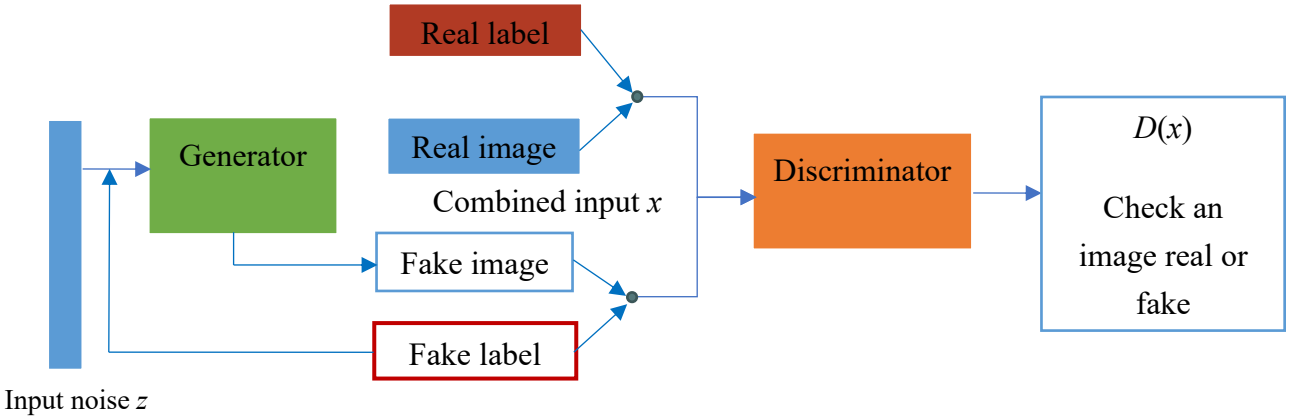
3.2, CGAN

3.2.1, 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.

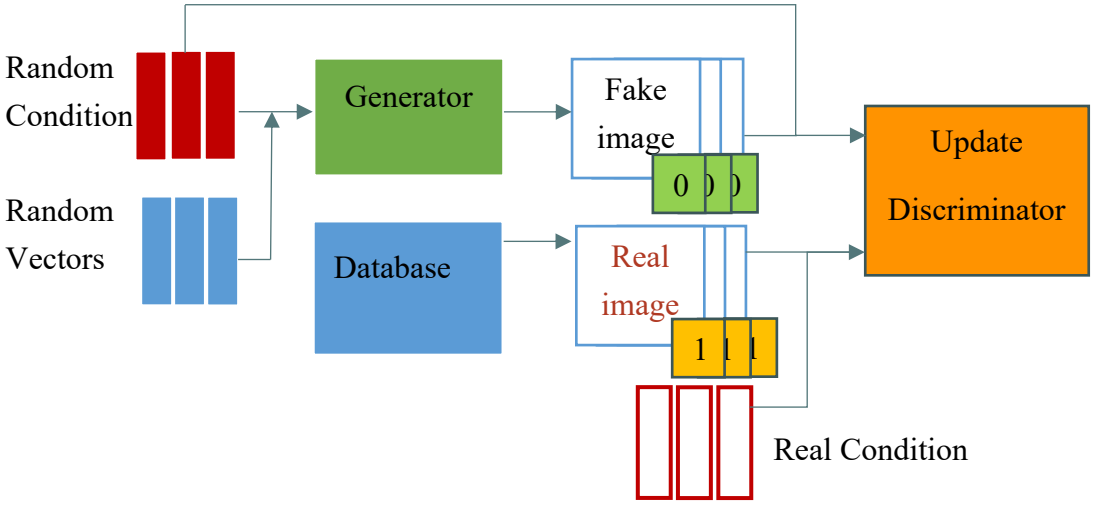


Conditional Generative Adversarial Networks (CGAN), an extension of the GAN, allows you to generate images with specific conditions or attributes. Same as GAN, CGAN also has a generator and a discriminator. However, the difference is that both the generator and the discriminator of CGAN receive some 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.

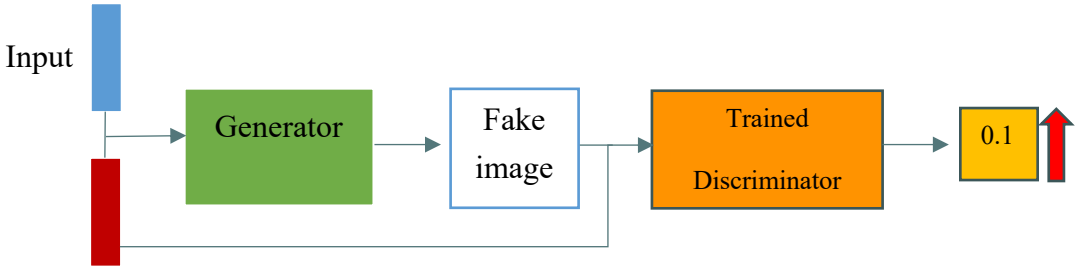
3.2.2, Algorithm

- 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 it's a gradient ascent process.



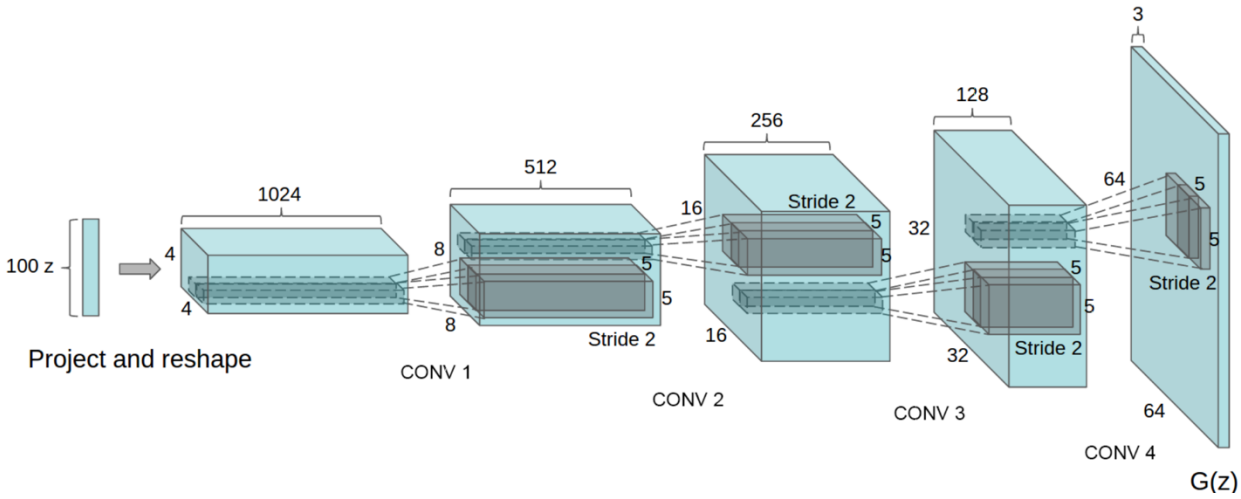
4. When we put these two processes together, it's a whole big network. As what we showed at the structure introduction. We put in the vector and the condition, then we end up with a number. But if we extract the output from the hidden layer in the middle, we get a complete image.

4, Experimental Setup

4.1, GAN

4.1.1, Generator

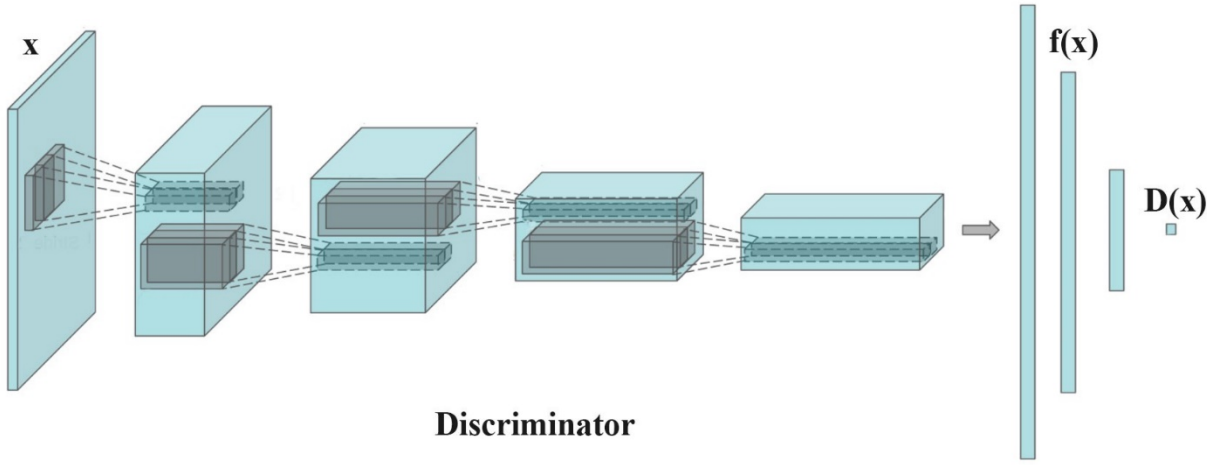
Similar to the Variational Auto Encoder (VAE), Generator is more like the decoder part. As you feed the generator with a vector or matrix, the Generator will return a new image. From a vector to an image, upsampling methods are always applied into the Generator.



(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.

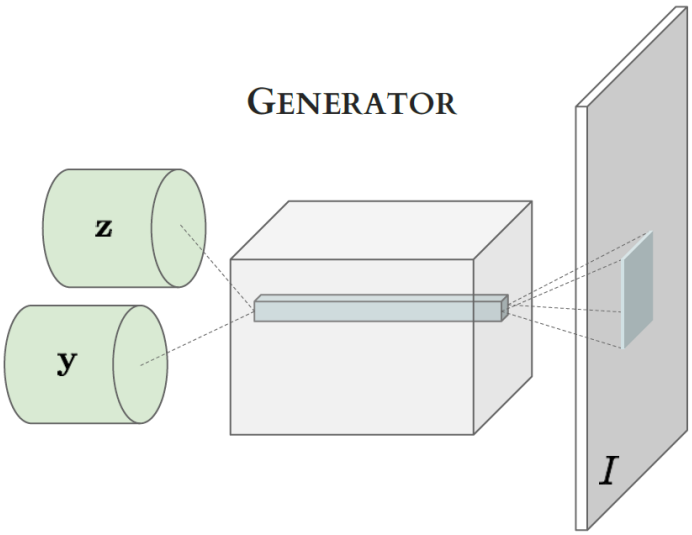
4.1.5, Training Generator

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.

4.2, CGAN

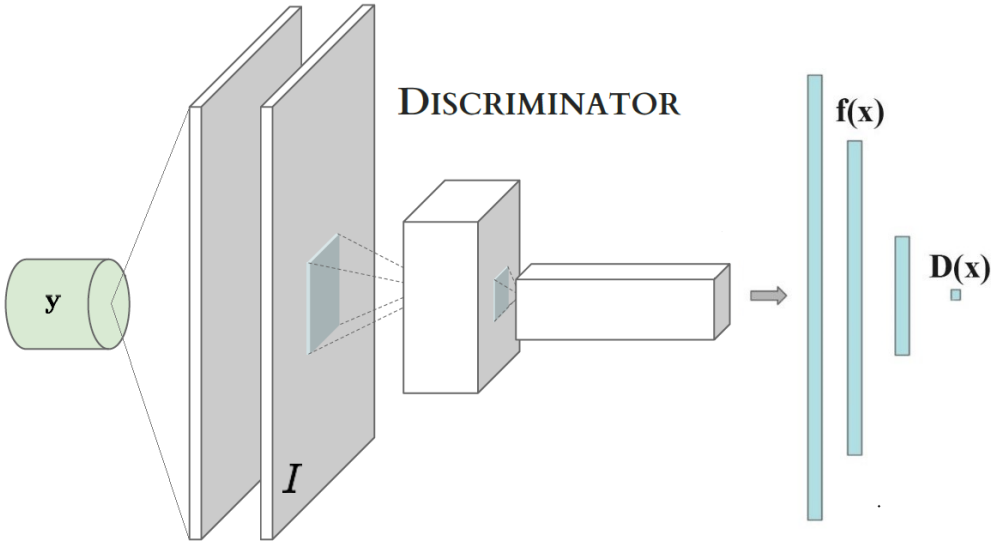
4.2.1, Generator

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.



4.2.2, 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.



4.2.3, Training

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.

5, Results

5.1 Simple GAN

5.1.1, Simple GAN with MLP

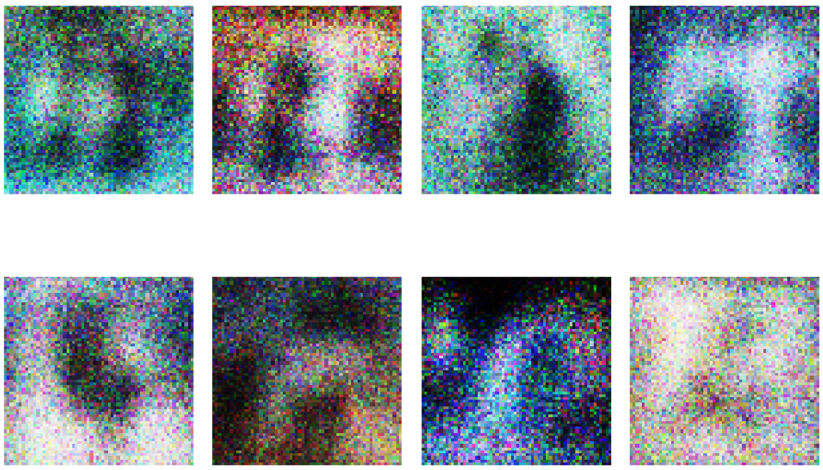
At the beginning, we play with the MLP-like networks. The idea of upsampling and downsampling is quite simple, where using increasing number of neurons in Generator and decreasing number of neurons in Discriminator.

Here is the architecture summary.

Input noise vector z ∈ $\mathbb{R}^{100}$	**Input RGB image ∈ $\mathbb{R}^{64\times64\times3}$**
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 ∈ $\mathbb{R}^{64\times64\times3}$	Output probability ∈ $\mathbb{R}^{1}$
Generator	**Discriminator**

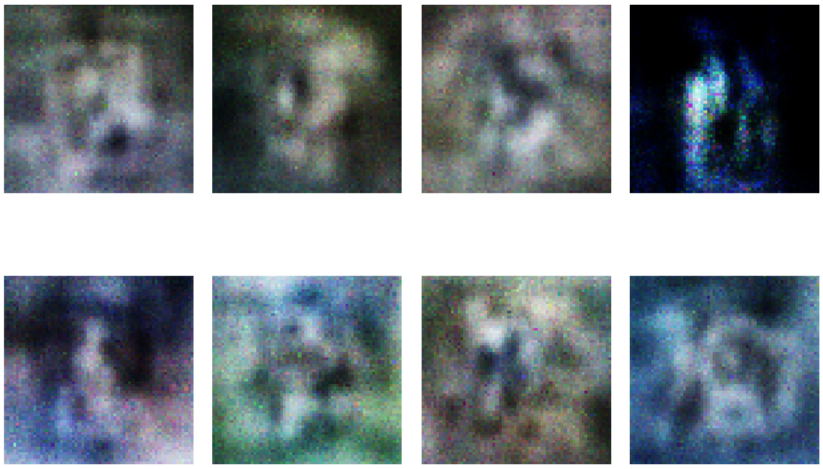
To be noted, the activation function on the output layer of Generator is hyperbolic tangent function, which can deal with gradient vanishing to some extent. The activation function on the output layer of Discriminator is sigmoid function, which converts the value into a probability.

Here are some results by this simple constructed GAN.



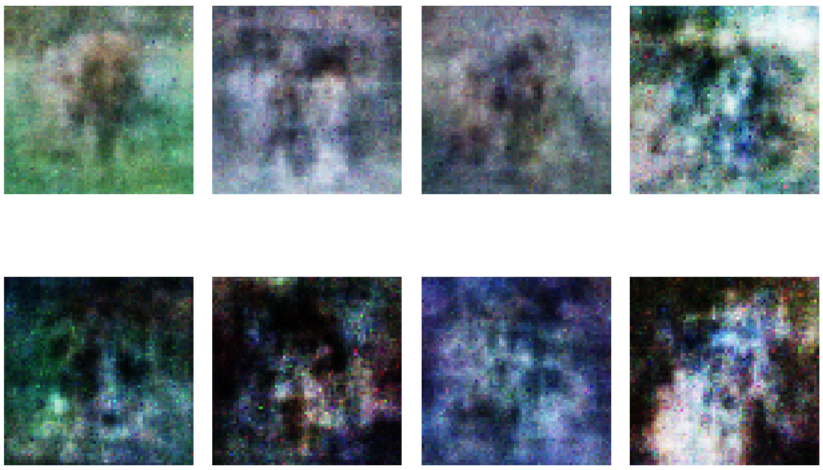
After 1000 epochs

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 β is zero, then smoothing by α 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 ∈ $\mathbb{R}^{100}$	**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	
Output fake images ∈ $\mathbb{R}^{64\times64\times3}$	

Generator

Discriminator

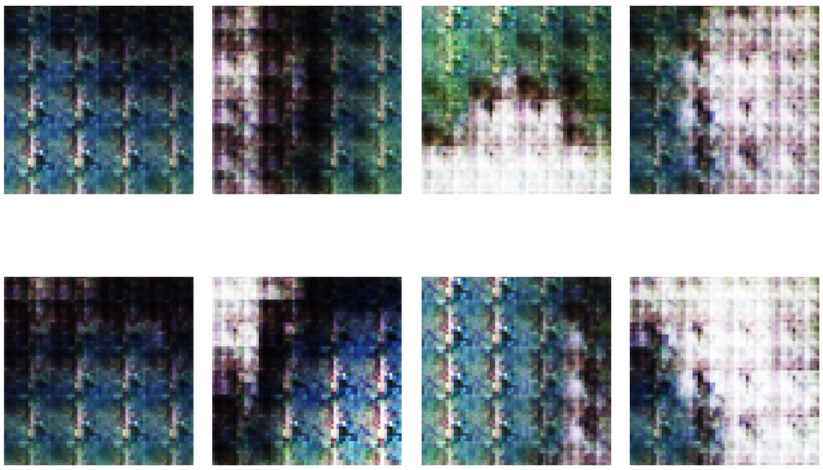
Input RGB image ∈ $\mathbb{R}^{64\times64\times3}$	**Out size**

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 $\in \mathbb{R}^1$	

Discriminator

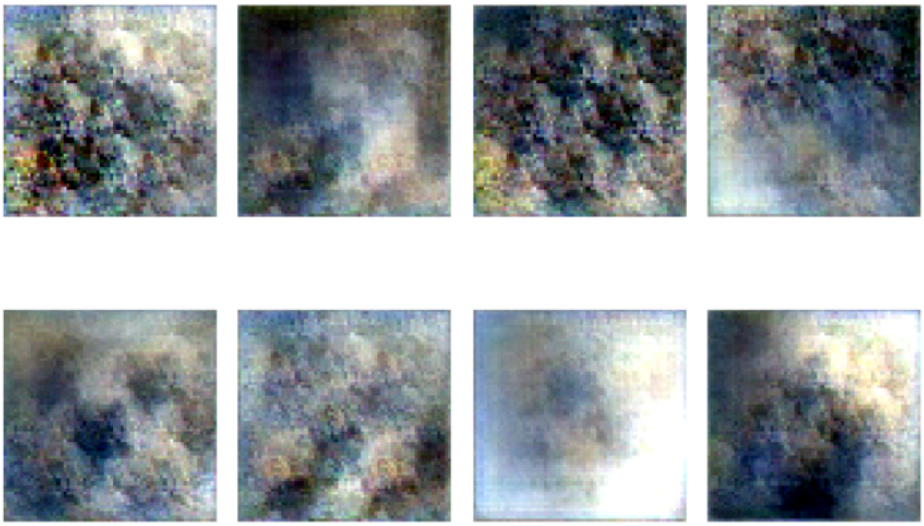
5.2.2, Model Performance

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.



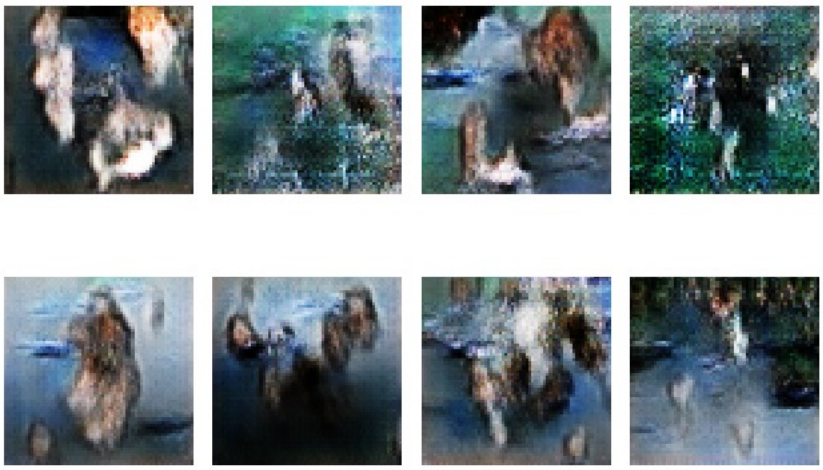
Fail to learn to generate dog-like images

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

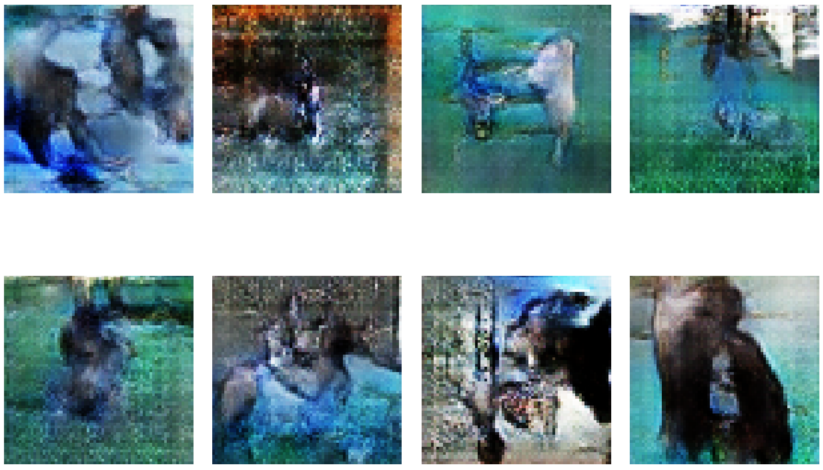
Tuning a DCGAN is much more complicated than simple GAN with MLP. There are two deep CNN participant this competition. Therefore, every small change of one network will affect the performance of another network and then influence the final result of the whole model. Indeed, it took 90% work of this project 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



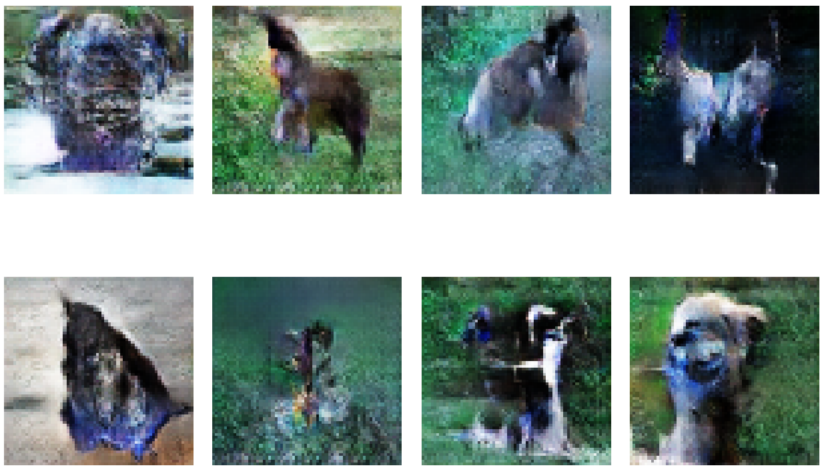
After 5000 epochs

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

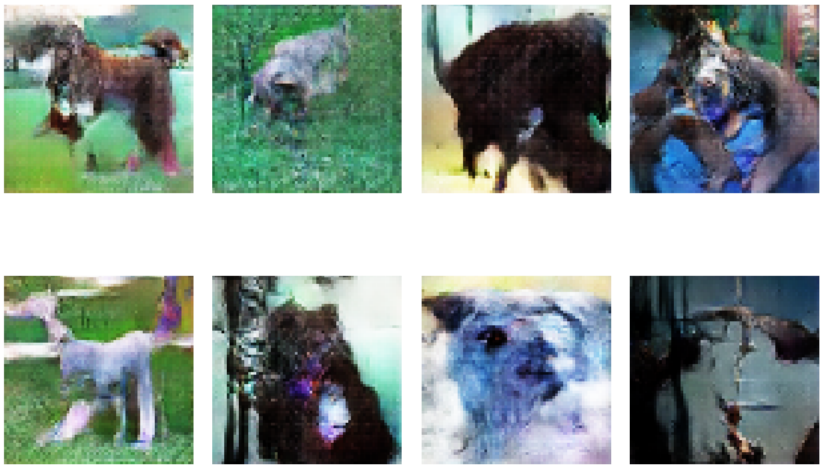


After 10000 epochs

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

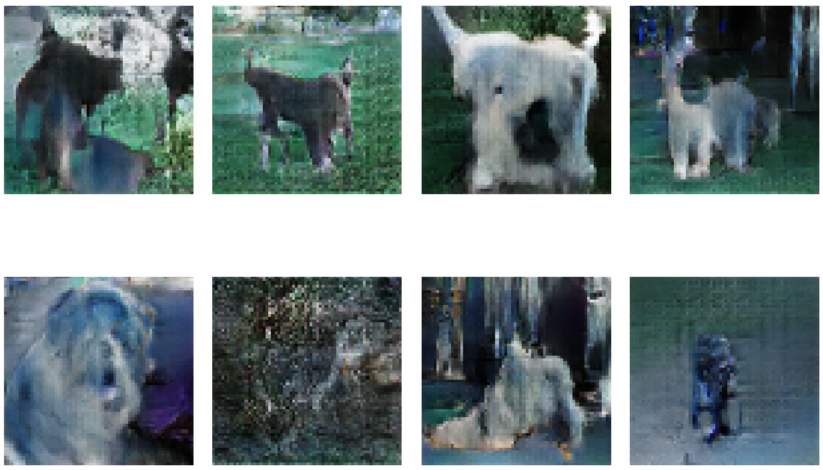


After 20000 epochs



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.

5.3, CGAN

5.3.1, 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 ∈ $\mathbb{R}^{100}$	**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 ∈ $\mathbb{R}^{64\times64\times3}$			

Generator

Input RGB image ∈ $\mathbb{R}^{64\times64\times3}$	**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 ∈ $\mathbb{R}^{1}$			

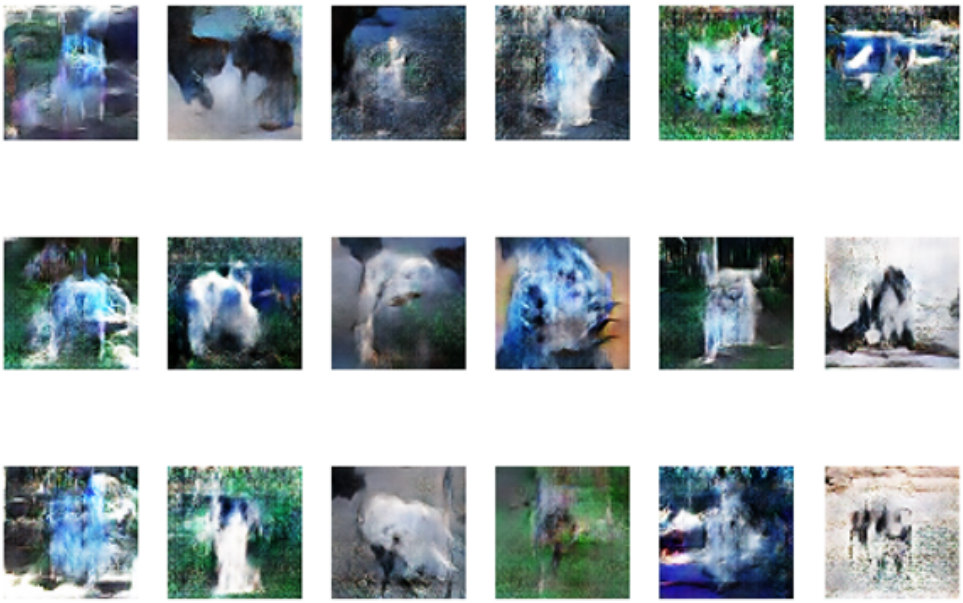
Discriminator

5.3.2, Model Performance

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

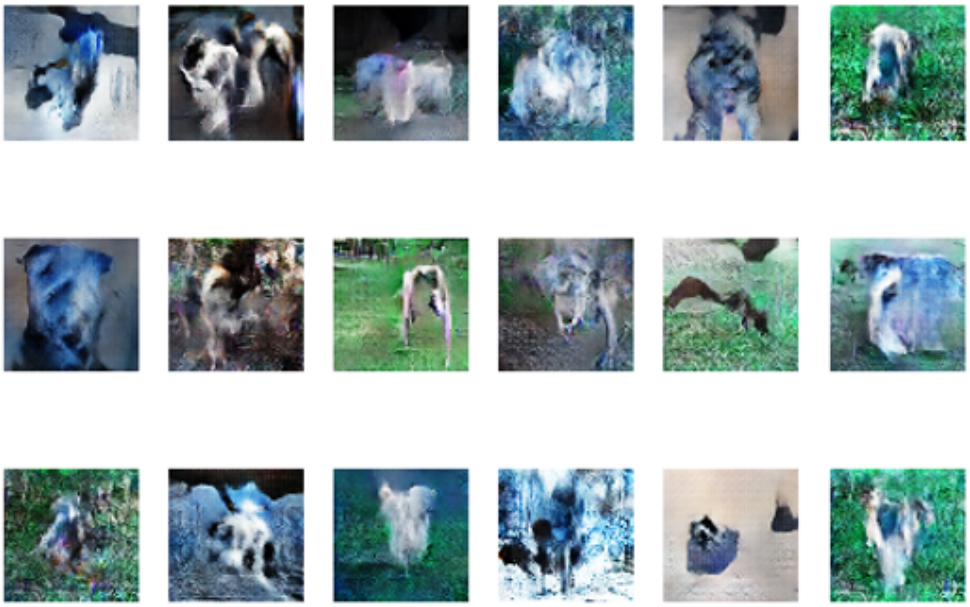
6,000 Epochs

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.

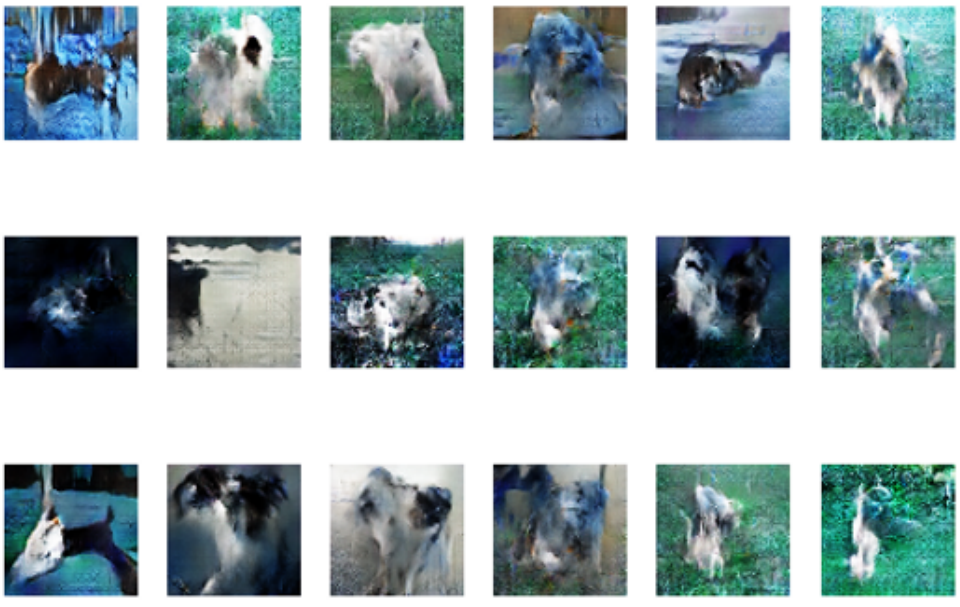


15,000 Epochs

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.



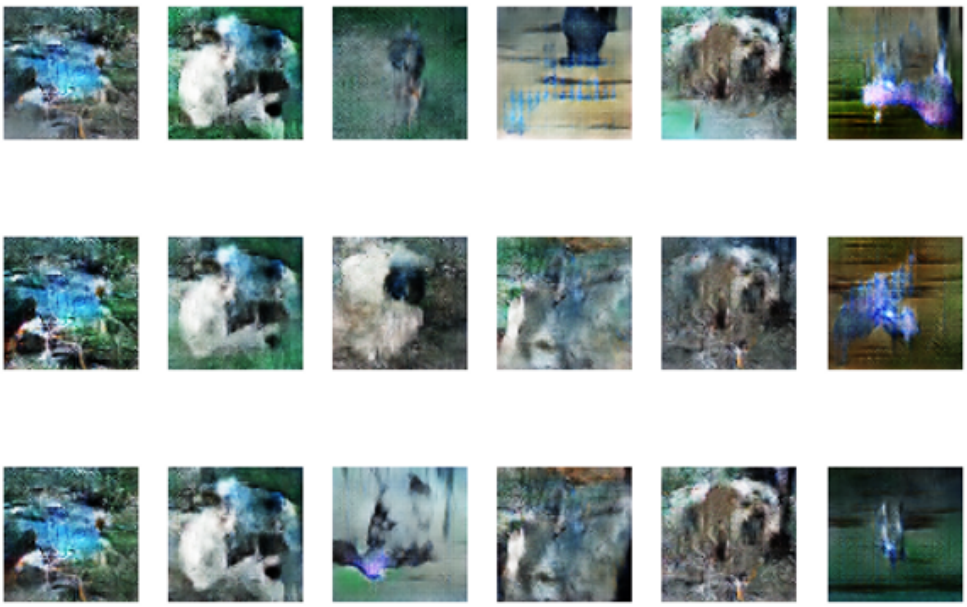
30,000 Epochs



45,000 Epochs

After 60,000 epochs, the generator cannot further generate a more dog-like image.

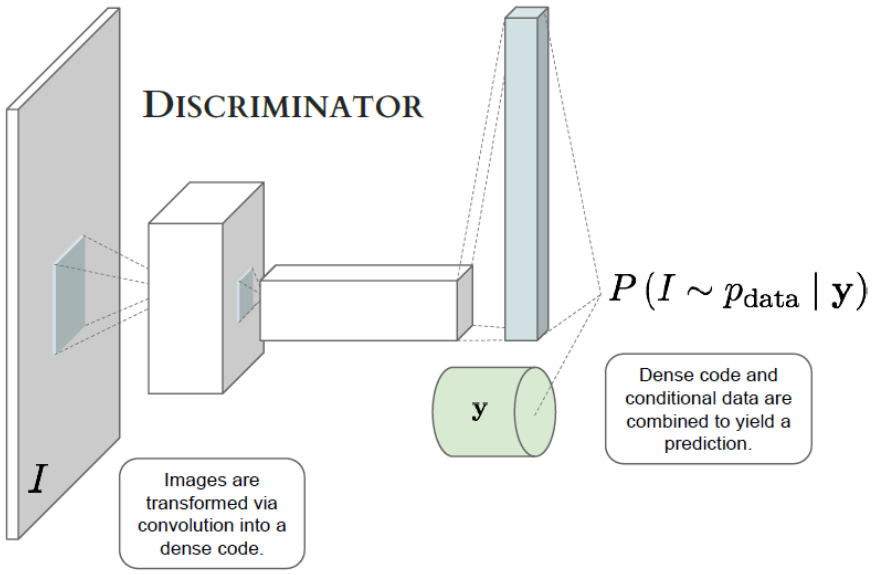
There is mode collapse of almost each column.



60,000 Epochs

5.3.3, Future Improvement

About the CGAN model we built, we reshape the condition and add it as the fourth dimension of the image. However, this discriminator input conditional data combine with images which via convolution into a dense code. The combined data yield a prediction of each class possibility. We think it maybe more focuses on the class label, because it classifies each kind.



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

Appendix

	DOG BREED	NUMBER OF OBSERVATIONS
0	n02085620-Chihuahua	152
1	n02085782-Japanese_spaniel	185
2	n02085936-Maltese_dog	252
3	n02086079-Pekinese	149
4	n02086240-Shih-Tzu	214
5	n02086646-Blenheim_spaniel	188
6	n02086910-papillon	196
7	n02087046-toy_terrier	172
8	n02087394-Rhodesian_ridgeback	172
9	n02088094-Afghan_hound	239
10	n02088238-basset	175
11	n02088364-beagle	195
12	n02088466-bloodhound	187
13	n02088632-bluetick	171
14	n02089078-black-and-tan_coonhound	159
15	n02089867-Walker_hound	153
16	n02089973-English_foxhound	157
17	n02090379-redbone	148
18	n02090622-borzoi	151
19	n02090721-Irish_wolfhound	218
20	n02091032-Italian_greyhound	182
21	n02091134-whippet	187
22	n02091244-Ibizan_hound	188
23	n02091467-Norwegian_elkhound	196
24	n02091635-otterhound	151
25	n02091831-Saluki	200
26	n02092002-Scottish_deerhound	232
27	n02092339-Weimaraner	160
28	n02093256-Staffordshire_bullterrier	155
29	n02093428-American_Staffordshire_terrier	164
30	n02093647-Bedlington_terrier	182
31	n02093754-Border_terrier	172
32	n02093859-Kerry_blue_terrier	179
33	n02093991-Irish_terrier	169
34	n02094114-Norfolk_terrier	172
35	n02094258-Norwich_terrier	185
36	n02094433-Yorkshire_terrier	164

37	n02095314-wire-haired_fox_terrier	157
38	n02095570-Lakeland_terrier	197
39	n02095889-Sealyham_terrier	202
40	n02096051-Airedale	202
41	n02096177-cairn	197
42	n02096294-Australian_terrier	196
43	n02096437-Dandie_Dinmont	180
44	n02096585-Boston_bull	182
45	n02097047-miniature_schnauzer	154
46	n02097130-giant_schnauzer	157
47	n02097209-standard_schnauzer	155
48	n02097298-Scotch_terrier	158
49	n02097474-Tibetan_terrier	206
50	n02097658-silky_terrier	183
51	n02098105-soft-coated_wheaten_terrier	156
52	n02098286-West_Highland_white_terrier	169
53	n02098413-Lhasa	186
54	n02099267-flat-coated_retriever	152
55	n02099429-curly-coated_retriever	151
56	n02099601-golden_retriever	150
57	n02099712-Labrador_retriever	171
58	n02099849-Chesapeake_Bay_retriever	167
59	n02100236-German_short-haired_pointer	152
60	n02100583-vizsla	154
61	n02100735-English_setter	161
62	n02100877-Irish_setter	155
63	n02101006-Gordon_setter	153
64	n02101388-Brittany_spaniel	152
65	n02101556-clumber	150
66	n02102040-English_springer	159
67	n02102177-Welsh_springer_spaniel	150
68	n02102318-cocker_spaniel	159
69	n02102480-Sussex_spaniel	151
70	n02102973-Irish_water_spaniel	150
71	n02104029-kuvasz	150
72	n02104365-schipperke	154
73	n02105056-groenendael	150
74	n02105162-malinois	150
75	n02105251-briard	152
76	n02105412-kelpie	153
77	n02105505-komondor	154

78	n02105641-Old_English_sheepdog	169
79	n02105855-Shetland_sheepdog	157
80	n02106030-collie	153
81	n02106166-Border_collie	150
82	n02106382-Bouvier_des_Flandres	150
83	n02106550-Rottweiler	152
84	n02106662-German_shepherd	152
85	n02107142-Doberman	150
86	n02107312-miniature_pinscher	184
87	n02107574-Greater_Swiss_Mountain_dog	168
88	n02107683-Bernese_mountain_dog	218
89	n02107908-Appenzeller	151
90	n02108000-EntleBucher	202
91	n02108089-boxer	151
92	n02108422-bull_mastiff	156
93	n02108551-Tibetan_mastiff	152
94	n02108915-French_bulldog	159
95	n02109047-Great_Dane	156
96	n02109525-Saint_Bernard	170
97	n02109961-Eskimo_dog	150
98	n02110063-malamute	178
99	n02110185-Siberian_husky	192
100	n02110627-affenpinscher	150
101	n02110806-basenji	209
102	n02110958-pug	200
103	n02111129-Leonberg	210
104	n02111277-Newfoundland	195
105	n02111500-Great_Pyrenees	213
106	n02111889-Samoyed	218
107	n02112018-Pomeranian	219
108	n02112137-chow	196
109	n02112350-keeshond	158
110	n02112706-Brabancon_griffon	153
111	n02113023-Pembroke	181
112	n02113186-Cardigan	155
113	n02113624-toy_poodle	151
114	n02113712-miniature_poodle	155
115	n02113799-standard_poodle	159
116	n02113978-Mexican_hairless	155
117	n02115641-dingo	156
118	n02115913-dhole	150

119	n02116738-African_hunting_dog	169