Final Group Project Report
Anime Face Generator using GAN

Group 2 Team Member: Chen Chen, Hao Ning, Hungchun Lin

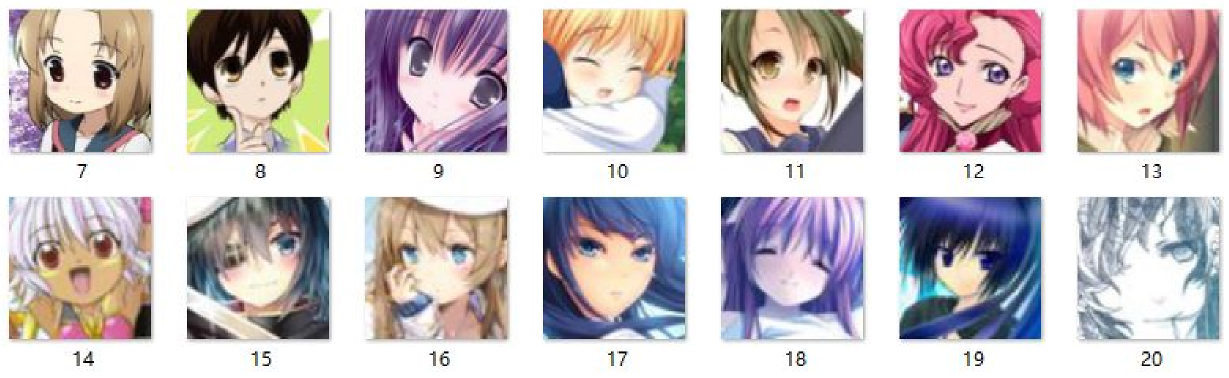
Introduction

Generative Adversarial Network(GAN) is a hot topic in machine learning in recent years, it was invented by Ian Goodfellow and his colleagues in 2014[1]. It has many real life applications in Fashion, Art, Advertising, Science,Video games[2] (picture generation, image conversion, picture repair, video prediction, etc). Image generation is one of the amazing practices of GAN. In this project, a generator that produce various types of anime faces will be developed using Deep convolutional GAN, then the performance of the generator is optimized by Wasserstein GAN (WGAN) and the improved WGAN, which is the WGAN with Gradient Penalty (WGAN-GP) .

Description of the Data Set

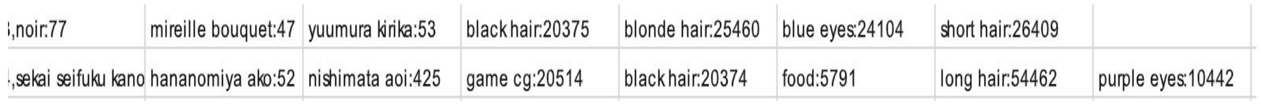
There is an image archive and a tag archive in the original source. Image data and tag data are collected by students from NTU, downloaded from google drive, each consists of 33431 instances. Also, extra_data is used from this google drive, which consist of 36740 instances.

Animation images example:



All images are colored animation portraits with the same size of 96x96.

Character tags example:



;,noir:77	mireille bouquet:47	yuumura kirika:53	black hair:20375	blonde hair:25460	blue eyes:24104	short hair:26409	
,sekai seifuku kano	hananomiya ako:52	nishimata aoi:425	game cg:20514	black hair:20374	food:5791	long hair:54462	purple eyes:10442

We will not use tags in this project. But it will be used in our future work in CGAN.

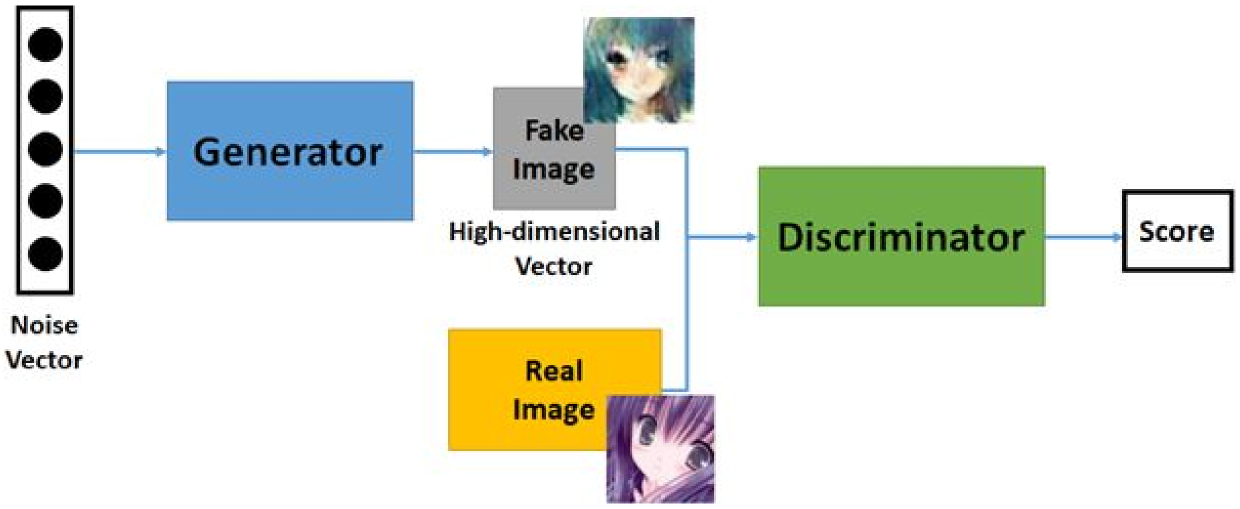
[1] Goodfellow, Ian, etc. (2014). *Generative Adversarial Networks*

[2] Generative adversarial network wikipedia, https://en.wikipedia.org/wiki/Generative_adversarial_network

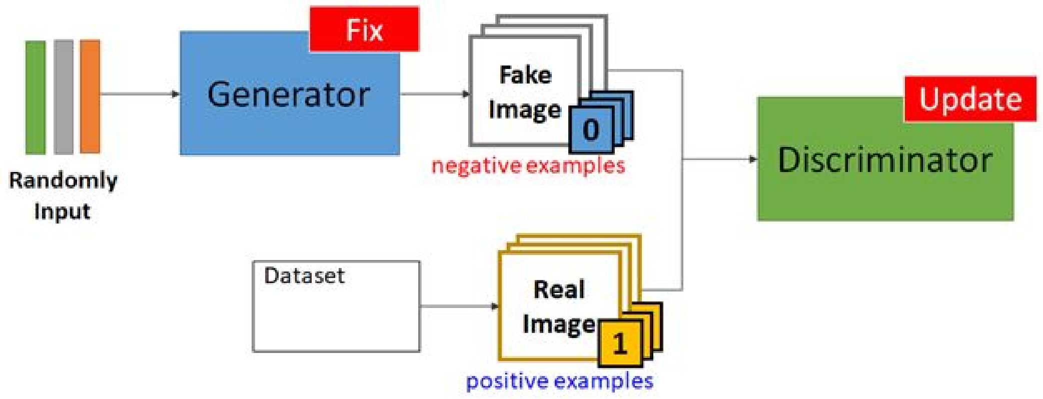
Deep Network and Training Algorithm

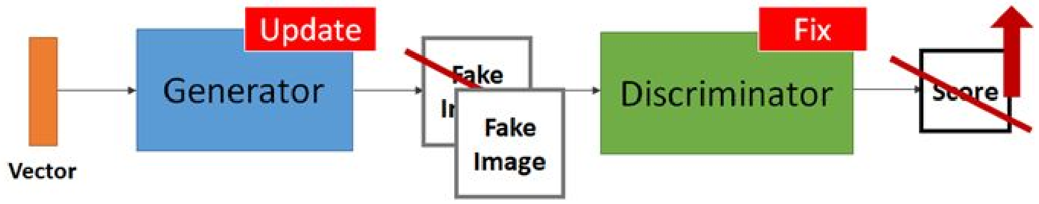
A generative adversarial network (GAN), is an architecture for training deep learning-based generative models. The architecture of GAN consists of a generator and a discriminator model.
Based on the original GAN, the original GAN can be improved by changing the loss function. The Wasserstein GAN (WGAN), was introduced by Martin Arjovsky, et al., which is an extension of the original GAN that both improves the stability when training the model and proposes a new loss function that correlates with the quality of generated images. Based on the original WGAN, a paper "Improved Training of Wasserstein GANs" written by Ishaan Gulrajani, et al. proposed to expose penalty on the norm of weights from the critic network, which is the WGAN with gradient penalty(WGAN-GP) in this project.

GAN Theory



GANs are basically made up of a system of two competing neural network models that compete with each other. The Generator generates fake samples of data, and tries to fool the Discriminator. The Discriminator tries to distinguish between the real and fake samples. The steps are repeated several times and in this, the Generator and Discriminator get better and better in their respective jobs after each repetition.





After initializing the generator and discriminator, we fix the generator first, and update the discriminator. Discriminator learns to assign high scores to real objects and low score to generated objects. Then we fix the discriminator and update the generator. Generator learns to “fool” the discriminator.

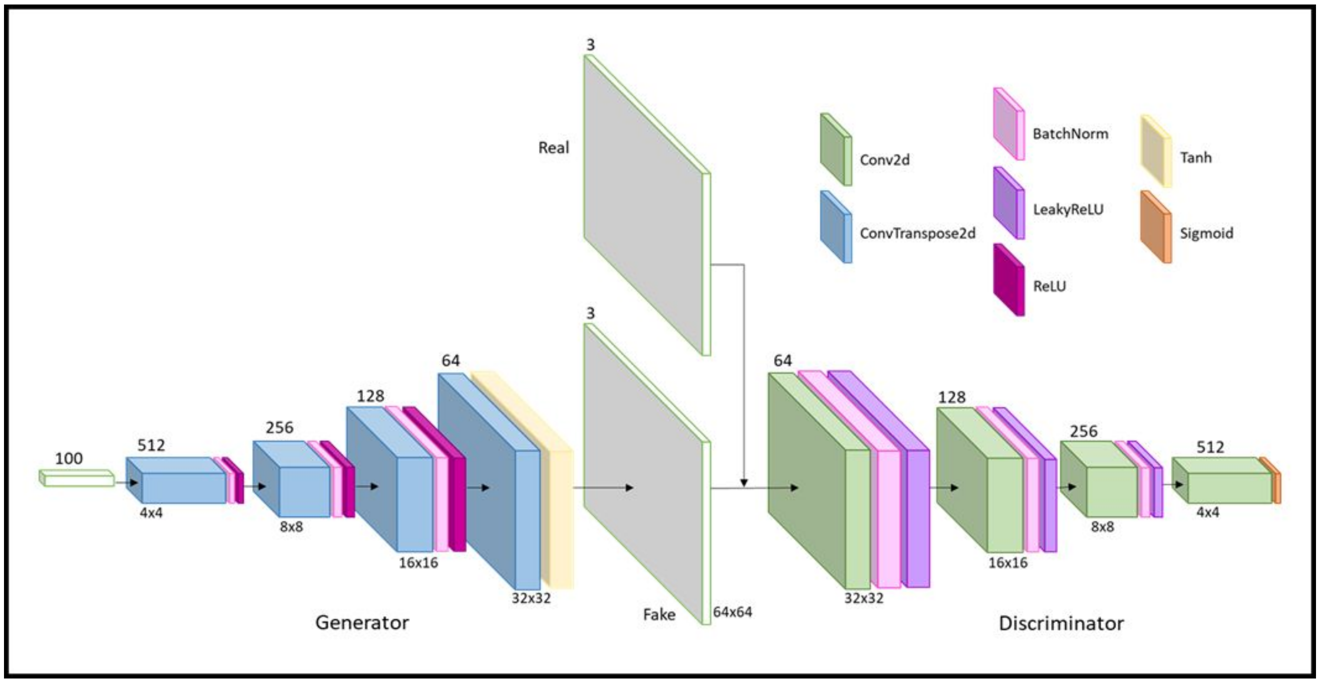
DCGAN

Algorithm

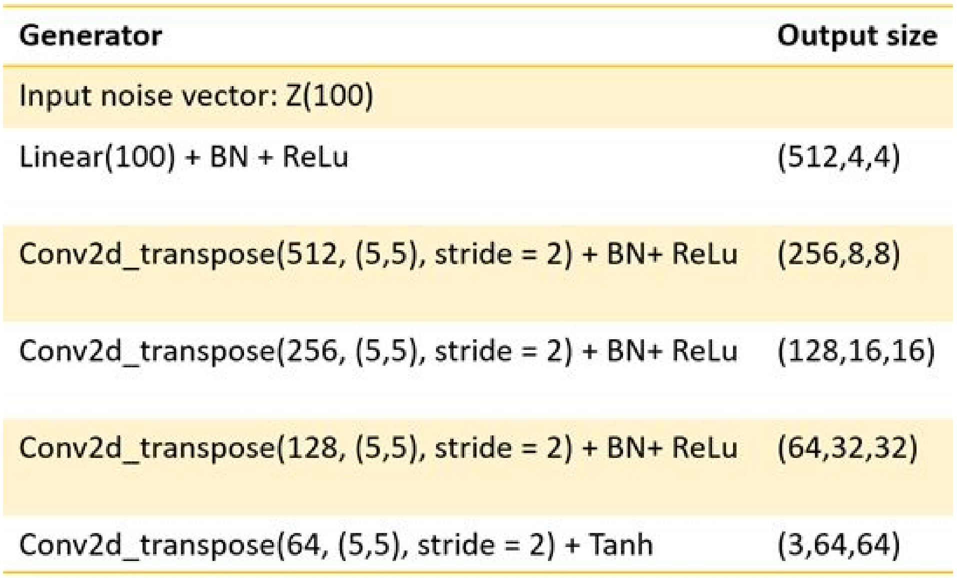
Deep Convolutional GAN (DCGAN) is one of the most popular GAN. It is composed of convolution networks in place of multi-layer perceptrons. DCGAN uses a standard CNN architecture on the discriminative model. For the generator, convolutions are replaced with upconvolutions, so the representation at each layer of the generator is actually successively larger, as it makes from a low-dimensional latent vector onto a high-dimensional image. The convolution networks implemented without max pooling, which is replaced by convolutional stride. And for the DCGAN, it utilized leaky ReLU for discriminator, because Leaky ReLU allows the pass of a small gradient signal for negative values, it makes the gradients from the discriminator flows stronger into the generator.

- x is original data, z is noise
  - Sample m examples $\{x^1, x^2, \dots, x^m\}$ from database
  - Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
  - Obtaining generated data $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = \mathrm{G}(z^i)$
  - Update discriminator parameters θ_d to maximize
    - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} logD(x^i) + \sum_{i=1}^{m}(1 - logD\left(\mathrm{G}(z^i)\right))$
    - $\theta_d \leftarrow \theta_d + \eta\nabla\tilde{V}(\theta_d)$
  - Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
  - Update generator parameters θ_g to minimize
    - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} \log(1 - D\left(G(z^i)\right))$
    - $\theta_g \leftarrow \theta_g - \eta\nabla\tilde{V}(\theta_g)$

__Model Setup__

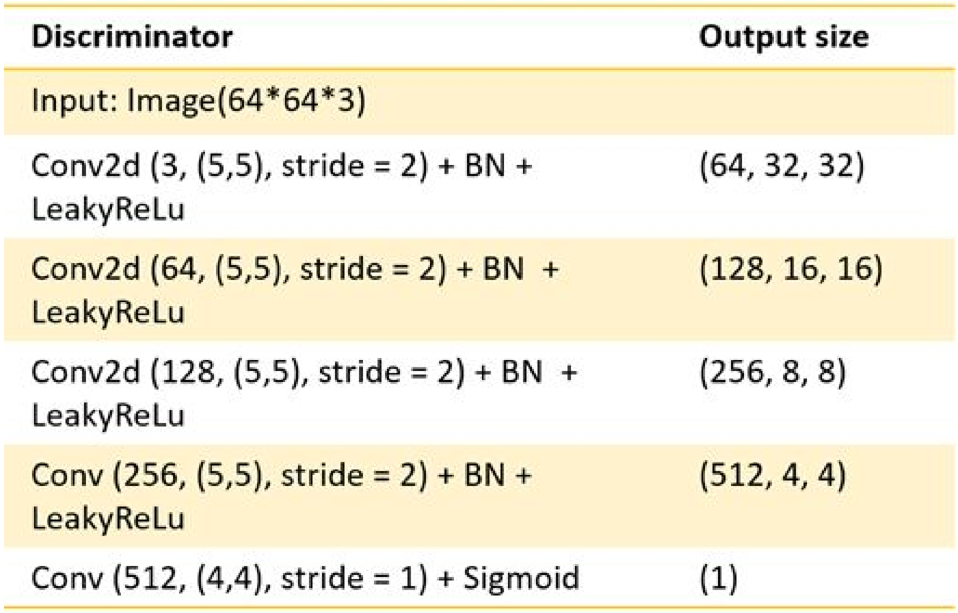


Generator Structure



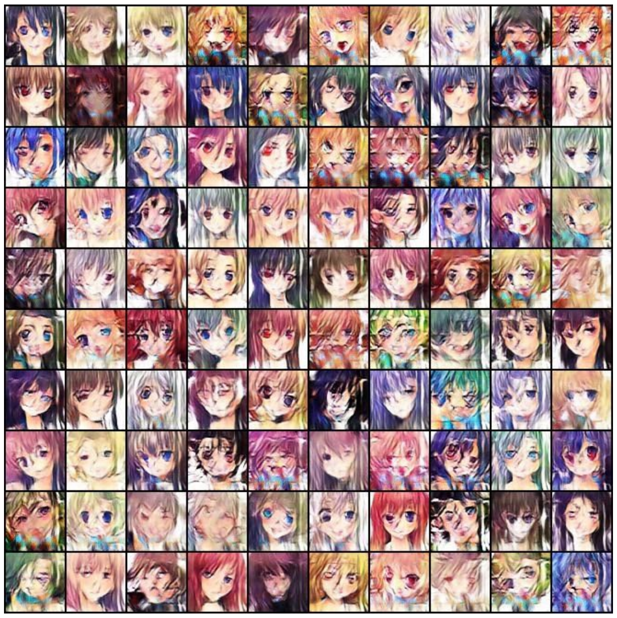
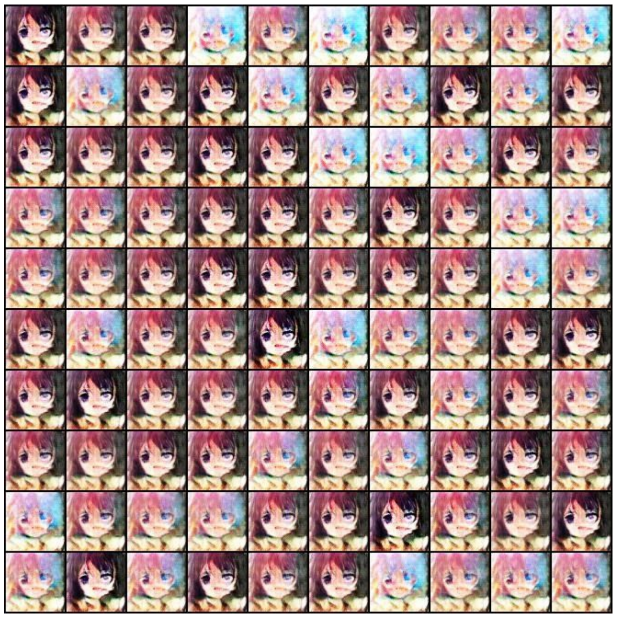
Generator	Output size
Input noise vector: Z(100)	
Linear(100) + BN + ReLu	(512,4,4)
Conv2d_transpose(512, (5,5), stride = 2) + BN+ ReLu	(256,8,8)
Conv2d_transpose(256, (5,5), stride = 2) + BN+ ReLu	(128,16,16)
Conv2d_transpose(128, (5,5), stride = 2) + BN+ ReLu	(64,32,32)
Conv2d_transpose(64, (5,5), stride = 2) + Tanh	(3,64,64)

Discriminator Structure



Discriminator	Output size
Input: Image(64*64*3)	
Conv2d (3, (5,5), stride = 2) + BN + LeakyReLu	(64, 32, 32)
Conv2d (64, (5,5), stride = 2) + BN + LeakyReLu	(128, 16, 16)
Conv2d (128, (5,5), stride = 2) + BN + LeakyReLu	(256, 8, 8)
Conv (256, (5,5), stride = 2) + BN + LeakyReLu	(512, 4, 4)
Conv (512, (4,4), stride = 1) + Sigmoid	(1)

Mode Performance

Left: DCGAN epoch 25; Right: DCGAN epoch 50 (Mode Collapse)

DCGAN Summary

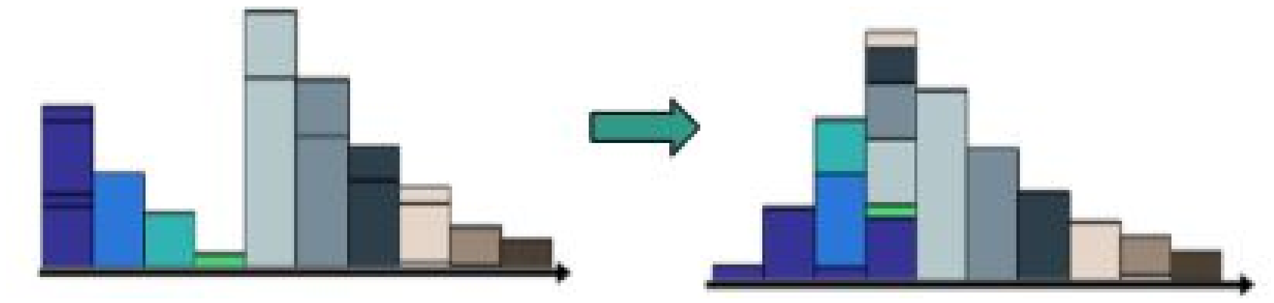
In this project, while the epoch went over 30 epoch, the mode collapse happens frequently. Mode collapse is when the generator generates a low-diversity of samples, or even the same sample, no matter how the different inputs, the output does not change. In real life, the data should be multimodal, but when the mode collapse happens, it only generates one mode of data. Mode collapse is a har problem for training GAN, it may happen only partially or not at all. The reason for mode collapse is that a generated image converges to a x*, that fool the discriminator the most. In this situation, x* will be independent of the input.

WGAN

In GAN, the loss actually measures how well the generator fools the discriminator (since the output is a probability) rather than the measurement of image quality. The generator loss in GAN does not drop even when the image quality improves.

Wasserstein Distance

Wasserstein Distance (or Earth Mover’s distance) is a measure of the distance between two probability distributions. The best ‘moving plan’ is the minimum "cost" of turning one pile into the other between 2 distributions, as shown below.



JS Convergence

Loss function of GAN is equivalent to JS convergence (when D at optimality). The problems in GAN can be explained with math.

First, what is the best value for D:

$$L(G,D) = \int_x \Big(p_r(x)\log(D(x)) + p_g(x)\log(1-D(x)) \Big) dx$$

Let

$$\tilde{x} = D(x), A = p_r(x), B = p_g(x)$$

$$\begin{aligned} f(\tilde{x}) &= Alog\tilde{x} + Blog(1-\tilde{x}) \\ \frac{df(\tilde{x})}{d\tilde{x}} &= A\frac{1}{ln10}\frac{1}{\tilde{x}} - B\frac{1}{ln10}\frac{1}{1-\tilde{x}} \\ &= \frac{1}{ln10}(\frac{A}{\tilde{x}} - \frac{B}{1-\tilde{x}}) \\ &= \frac{1}{ln10}\frac{A-(A+B)\tilde{x}}{\tilde{x}(1-\tilde{x})} \end{aligned}$$

The best value of the discriminator is calculated by set the derivative to 0 :

$$D^*(x) = \tilde{x}^* = \frac{A}{A+B} = \frac{p_r(x)}{p_r(x)+p_g(x)} \in [0,1].$$

Once G is trained to optimality, Pg = Pr, thus, D = 0.5!

In other words, assuming that both D and G are optimized, discriminator can only flip a coin to tell the performance of the generator, although it's already generating samples as good as real ones.

In reality, Pg and Pr are always not overlapping for the following reasons:

1. Both Pg and Pr are low-dimensional manifold in high-dimensional space
2. Sampling

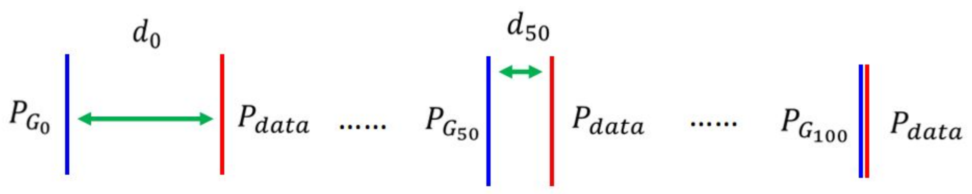
Now, the problem of JS divergence can be explained mathematically.

When 2 distribution are not overlapping:

$$KL(P_1||P_2) = \mathbb{E}_{x\sim P_1} \log \frac{P_1}{P_2} \qquad JS(P_1||P_2) = \frac{1}{2}KL(P_1||\frac{P_1+P_2}{2}) + \frac{1}{2}KL(P_2||\frac{P_1+P_2}{2})$$

$$DJS\,(\mathrm{P_r}, \mathrm{P_g}) = 1/2(\sum_{x=0,\, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0,\, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}) = \log 2$$

The JS convergence is always log2, so there will be no gradient for optimization.
Comparing Wasserstein distance to JS convergence:



$$JS(P_{G_0}, P_{data}) = log2 \qquad JS(P_{G_{50}}, P_{data}) = log2 \qquad JS(P_{G_{100}}, P_{data}) = 0$$

$$W(P_{G_0}, P_{data}) = d_0 \qquad W(P_{G_{50}}, P_{data}) = d_{50} \qquad W(P_{G_{100}}, P_{data}) = 0$$

Wasserstein distance can effectively calculate the difference between Pg and Pr even when they are not overlapping. To get Wasserstein distance, the objective function will be

$$V(G, D) \;=\; \max_{D \in 1-Lipschitz} \{E_{x\sim P_{data}}[D(x)] - E_{x\sim P_G}[D(x)]\}$$

1-Lipschitz function means:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|$$

For 1-Lipschitz function, it should be everywhere continuously differentiable. The output change should be less or equal to the input change. As shown bottom left, the green line is a smooth function, while the blue line is not.
Since D has to be a smooth function, WGAN proposed that using weight clipping (force the parameters w between c and -c) to provide a constraint, otherwise, as shown bottom right, the output of real data will become +infinity, the generated data will be going -infinity, then D will not converge. Note that D with weight clipping here is not a strict 1-Lipschitz function, it’s an ‘engineering’ approach (also for WGAN-GP).

Algorithm[3] and Implementation

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
    $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
 1: while $\theta$ has not converged do
 2:     for $t = 0, \ldots, n_{\text{critic}}$ do
 3:         Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
 4:         Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
 5:         $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
 6:         $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
 7:         $w \leftarrow \text{clip}(w, -c, c)$
 8:     end for
 9:     Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
10:     $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
11:     $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: end while
```

Two significant improvement of WGAN:

- No sign of mode collapse in experiments
- The generator can still learn when the critic performs well

WGAN is still very similar to the discriminator D in DCGAN, but with a few adjustments. There's no sigmoid function at output of the discriminator, thus the output is a scalar score, that can be interpreted as how real the input images are, rather than a probability. Regarding the generator G, we want to maximize the D's output for its fake instances. The changes are summarized as follows:

- Loss function, no log
- D: no sigmoid
- Clip the weight of D (-c,c), if w>c, w=c, if w<-c, w=-c
- Use RMSProp
- Train D more than G

	Discriminator	Generator
DCGAN	$\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m}(1 - \log D(G(z^i)))$	$\tilde{V} = \frac{1}{m}\sum_{i=1}^{m}(1 - \log D(G(z^i)))$
WGAN	$\tilde{V} = \frac{1}{m}\sum_{i=1}^{m}(D(x^i) - D(G(z^i)))$	$\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} D(G(z^i))$

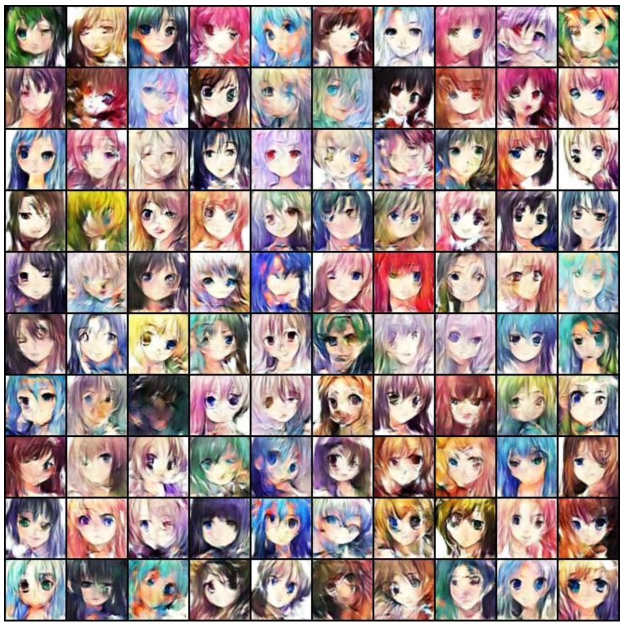
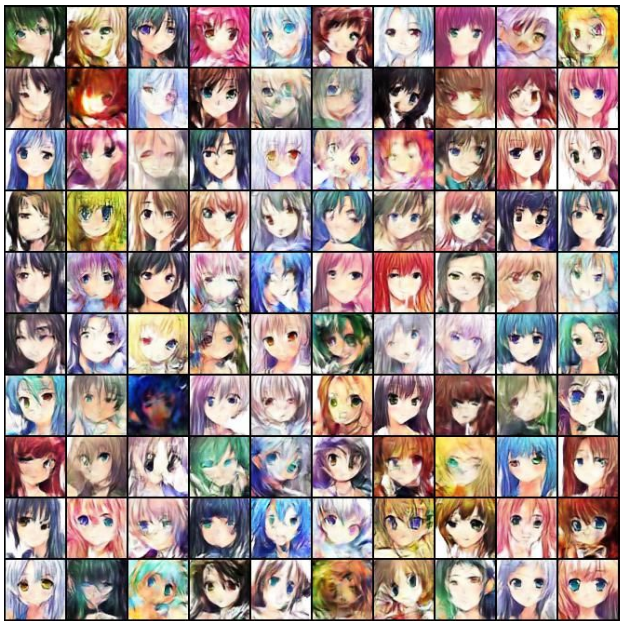
[3] https://arxiv.org/pdf/1701.07875.pdf

Hyperparameters:

- Clip Weight at (-0.01,0.01)
- Batch size 128
- Learning rate 0.0001

The code execution time is 4-10 hours depending on D training times (for each epoch, train D once/twice/4 times, G only once).

Model Performance

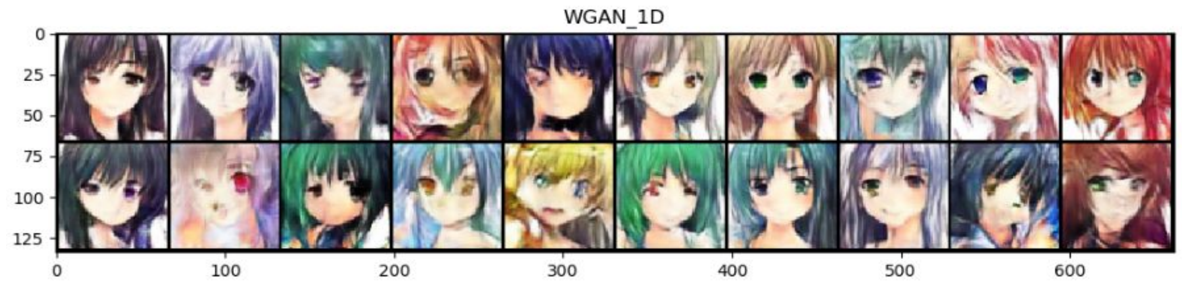



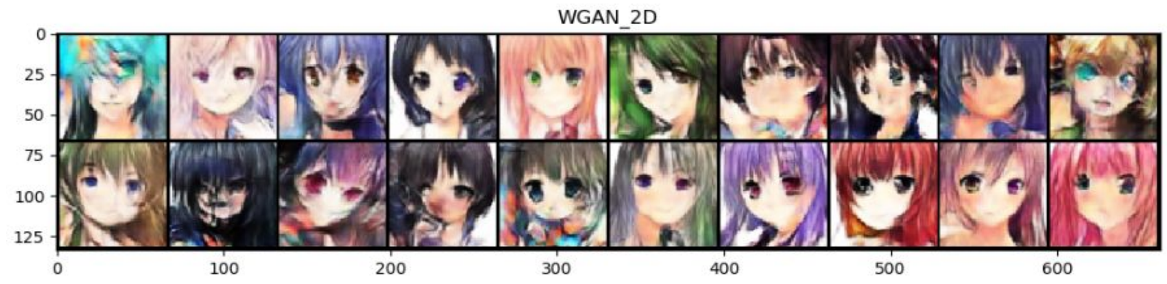
Left: Epoch 10 Right: Epoch 50

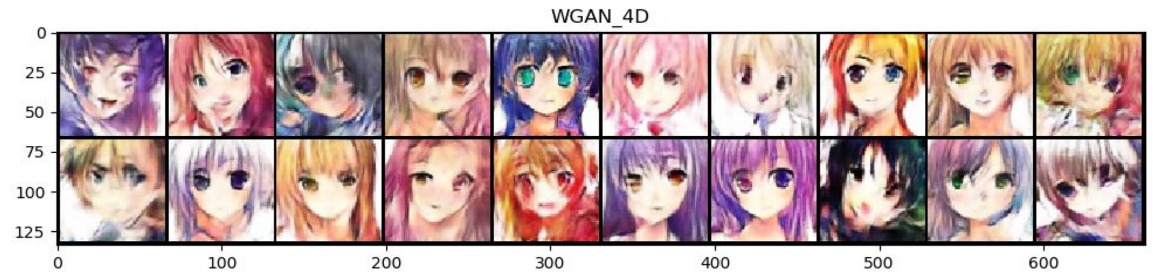
The no mode collapse observed and image quality is improved, such as the detail of hairs, eyes and the overall portrait. However, there're still some very bad quality images, probably due to the effectiveness of the weight clipping in this method.

Comparison of Train D more than G

From our observation for WGAN, there's no obvious differences when train D more than G, perhaps 4D is slightly better. This makes sense since weight clipping parameters have a great impact on the model performance.







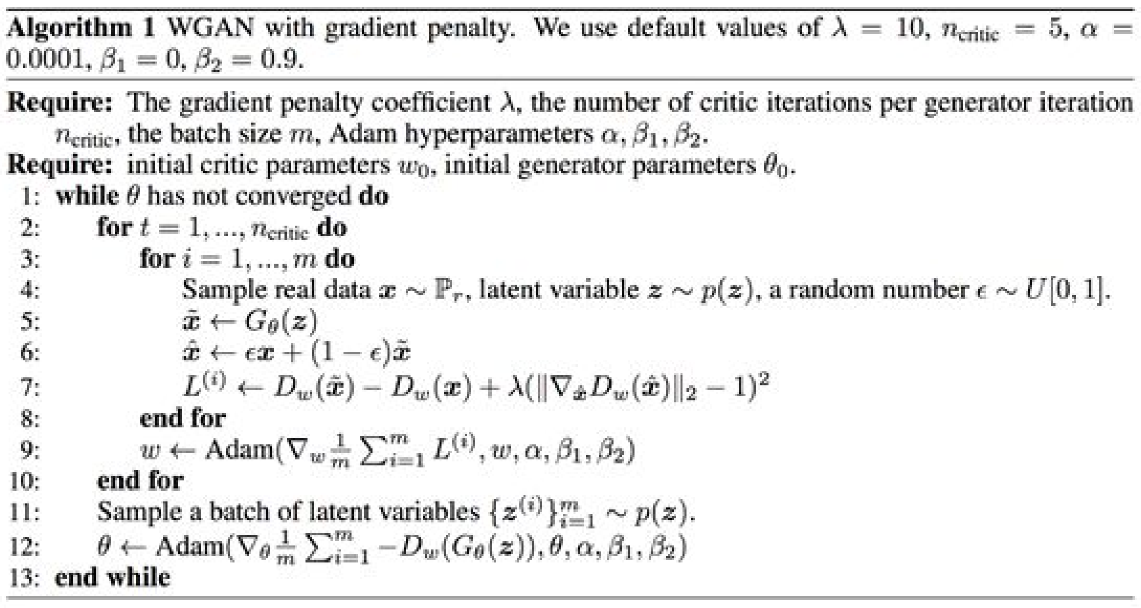
WGAN Summary

WGAN solved the mode collapse problem in GAN, the WGAN generator is able to produce some high quality anime faces. However, bad samples are still being produced thus the total yield is not at optimality. This is because weight clipping is an ‘engineering’ of constraining the objective function. WGAN-GP next will further improve the performance.

WGAN-GP

WGAN-GP is an improved method of WGAN, which also uses the Wasserstein distance to measure the divergence between generated image and real image. The difference between WGAN-GP and WGAN is the weight control method. In WGAN-GP, gradient penalty is used instead of weight clipping.

Algorithm[4]



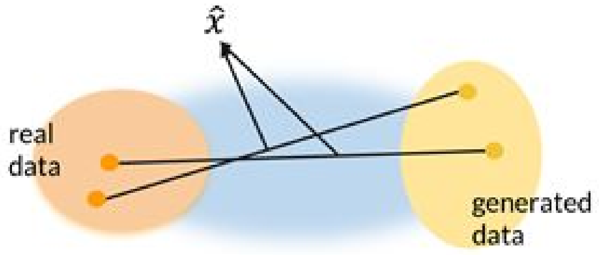
Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

```
Require: The gradient penalty coefficient $\lambda$, the number of critic iterations per generator iteration
    $n_{\text{critic}}$, the batch size $m$, Adam hyperparameters $\alpha, \beta_1, \beta_2$.
Require: initial critic parameters $w_0$, initial generator parameters $\theta_0$.
 1: while $\theta$ has not converged do
 2:     for $t = 1, ..., n_{\text{critic}}$ do
 3:         for $i = 1, ..., m$ do
 4:             Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
 5:             $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
 6:             $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
 7:             $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
 8:         end for
 9:         $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
10:     end for
11:     Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
12:     $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
13: end while
```

As mentioned in WGAN, 1-Lipschitz is used to control the discriminator's weight. And 1-Lipschitz function should be everywhere continuously differentiable. So, if the discriminator is 1-Lipschitz, it should have gradients with norm at most 1 everywhere under the generated image and real image. This is approved by the WGAN-GP author in the paper[5].

Gradient penalty: $$\lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$$

So points interpolated between the real and generated data should have a gradient norm of 1 for the discriminator. The model will be penalized if the gradient norm moves away from its target norm value 1.

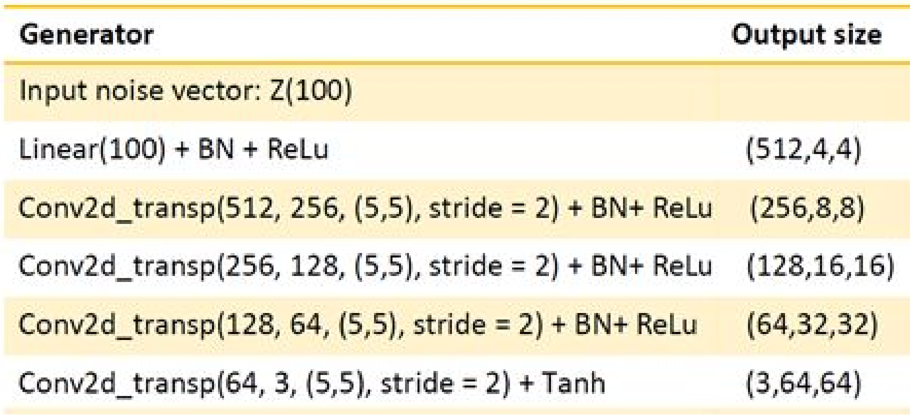


[4] https://arxiv.org/pdf/1701.07875.pdf
[5] https://arxiv.org/pdf/1701.07875.pdf

<u>Model Setup</u>

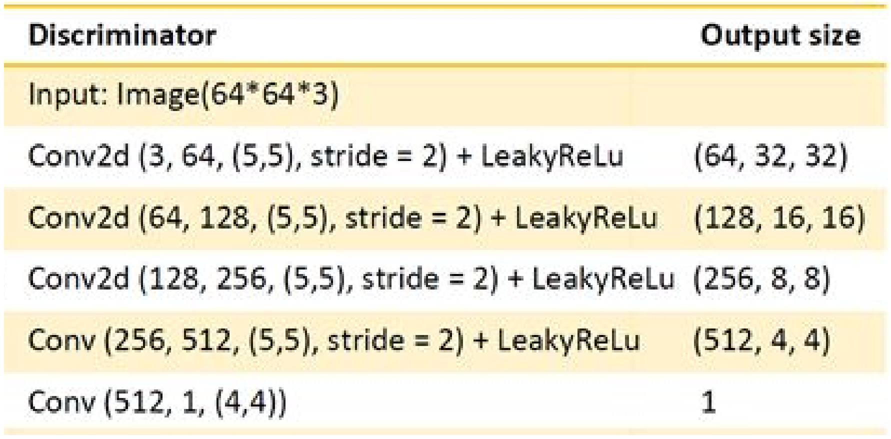
The model structure is almost the same as DCGAN. But here we remove the sigmoid activation at the last layer in discriminator. Besides, batch normalization in discriminator is also avoided, because it creates correlations between samples in the same batch, which may impact the effectiveness of the gradient penalty. According to the WGAN-GP author, Adam optimizer is used in this model.

Generator Structure



Generator	Output size
Input noise vector: Z(100)	
Linear(100) + BN + ReLu	(512,4,4)
Conv2d_transp(512, 256, (5,5), stride = 2) + BN+ ReLu	(256,8,8)
Conv2d_transp(256, 128, (5,5), stride = 2) + BN+ ReLu	(128,16,16)
Conv2d_transp(128, 64, (5,5), stride = 2) + BN+ ReLu	(64,32,32)
Conv2d_transp(64, 3, (5,5), stride = 2) + Tanh	(3,64,64)

Discriminator Structure



Discriminator	Output size
Input: Image(64*64*3)	
Conv2d (3, 64, (5,5), stride = 2) + LeakyReLu	(64, 32, 32)
Conv2d (64, 128, (5,5), stride = 2) + LeakyReLu	(128, 16, 16)
Conv2d (128, 256, (5,5), stride = 2) + LeakyReLu	(256, 8, 8)
Conv (256, 512, (5,5), stride = 2) + LeakyReLu	(512, 4, 4)
Conv (512, 1, (4,4))	1

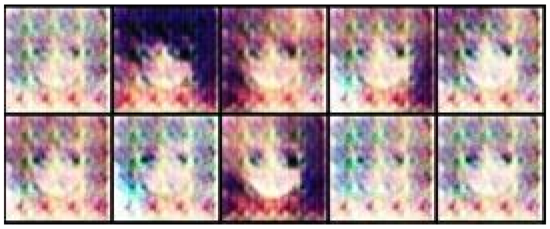
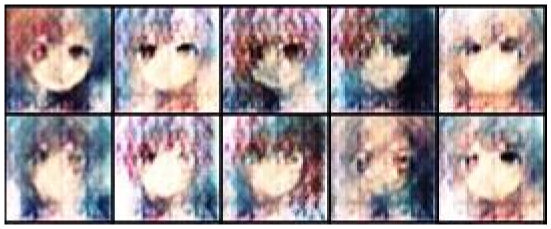
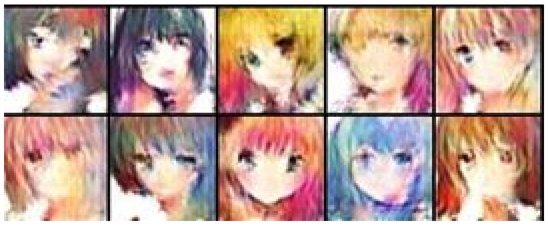
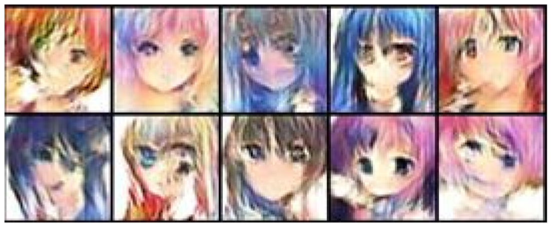
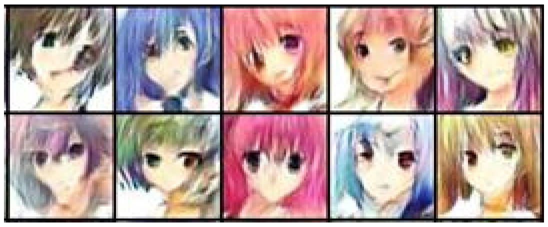
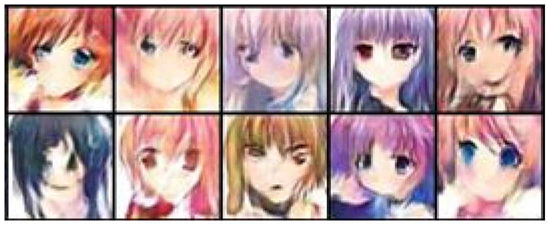
Tips for Tuning the Model

- Increase lambda(coefficient of gradient penalty)
- Train discriminator more than generator

<u>Model performance</u>

Tip 1:

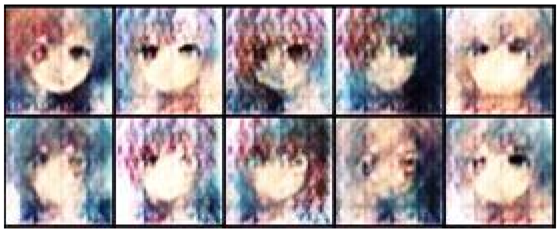
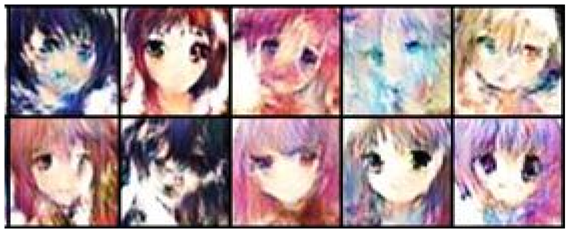
Batch number: 128 Learning rate: 0.0001 Noise:100 Train D once and G once

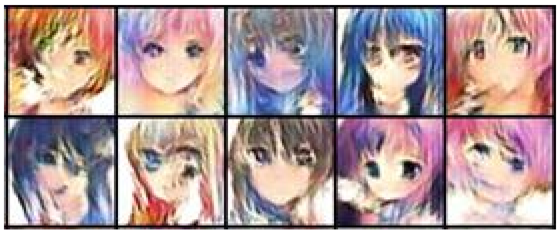
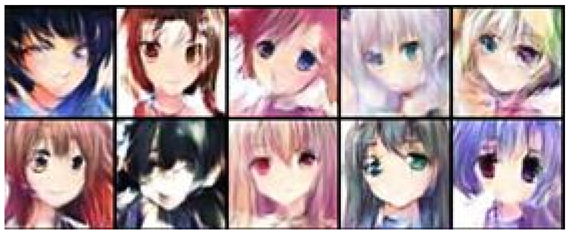
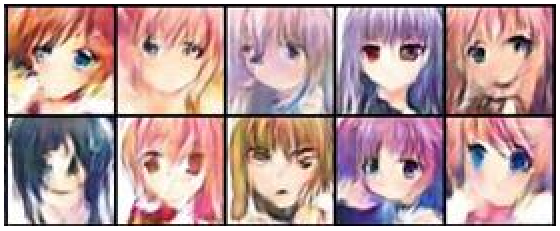
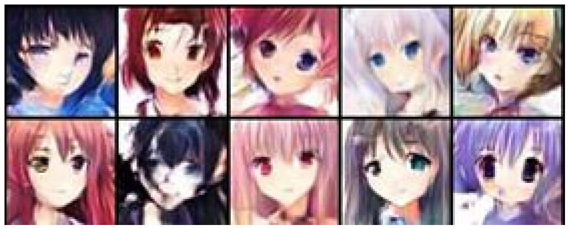
Epoch	**Lambda = 0.5**	**Lambda = 5**
10		
30		
50		

Results: By increasing lambda, the model converges faster, especially at the early stages.

Tip2:

Batch number: 128 Learning rate: 0.0001 Noise:100 Lambda = 5

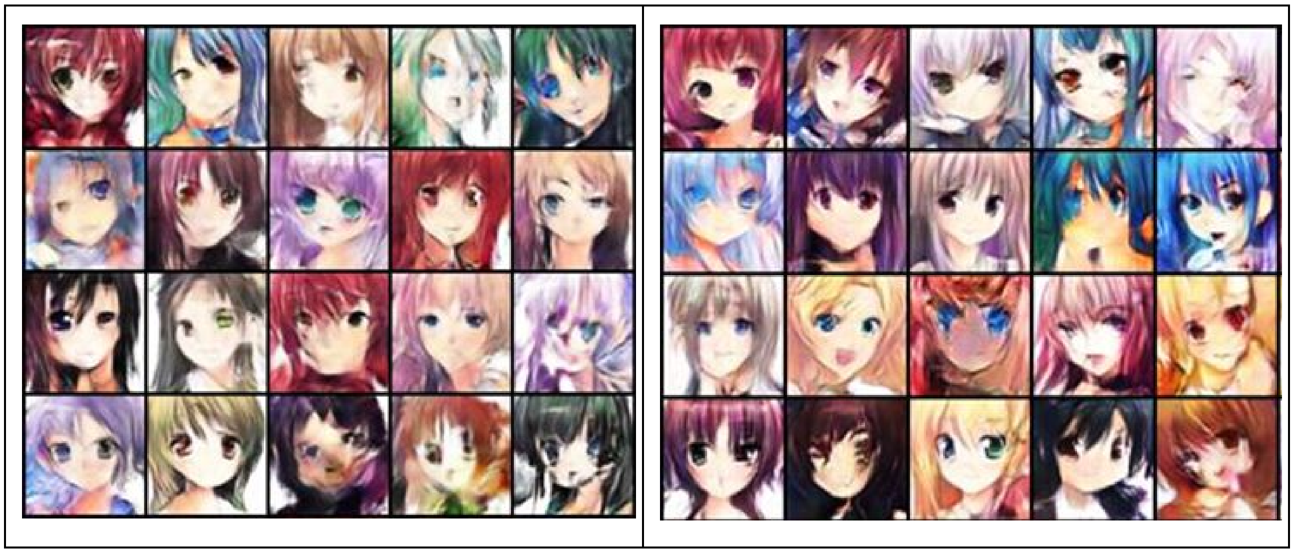
Epoch	**Train D once**	**Train D twice**
10		

30		
50		

Results: By train Discriminator twice, the model improved again. At epoch 50, faces are generated with mouth on the right hand side, but on the left hand side few have mouth.

Model Comparison

WGAN	**WGAN-GP**
Batch number: 128 Learning rate: 0.0001 Noise:100 Epoch: 50 Train D twice G once Weight clipping: (-0.01, 0.01)	Batch number: 128 Learning rate: 0.0001 Noise:100 Epoch: 50 Train D twice G once Lambda = 5



Results: Both WGAN and WGAN-GP are using the same parameters. WGAN-GP has a higher yield rate than WGAN, and faces are more clear. So WGAN-GP performance is better than WGAN.

<u>WGAN-GP Summary</u>

Using WGAN-GP, the quality of the generated images has been improved, the difference between good and bad images is not that big. However, there are still some faces not generated successfully. This is due to the gradient penalty is still not a strict 1-Lipschitz function, but it works better than the weight clipping method.

Project Summary

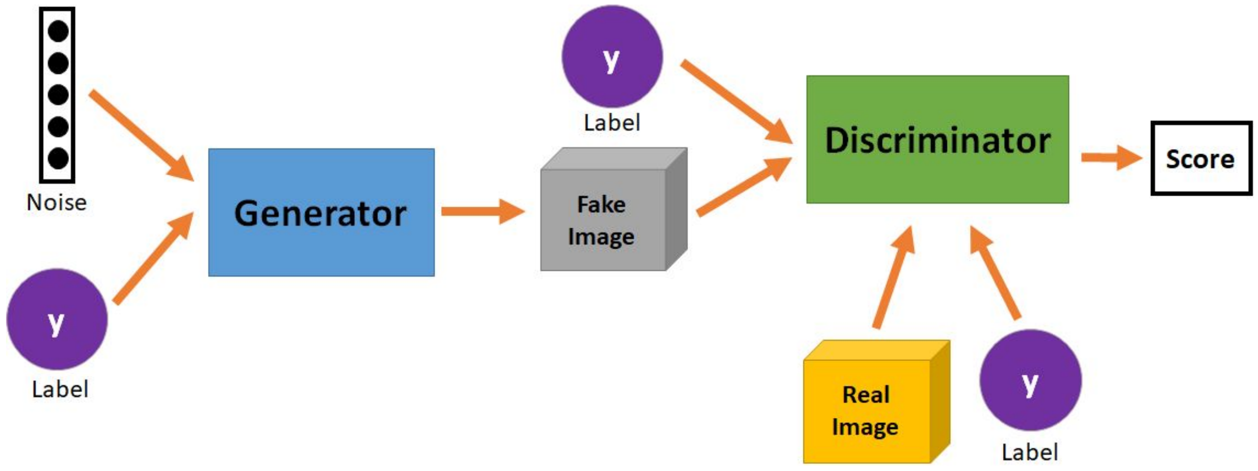
This project provides a detailed demonstration of the theory, math, architecture and implementation of Generative Adversarial Network. The problems of original DCGAN such as mode collapse and gradient vanishing is solved by using WGAN and WGAN-GP with modified loss function. Finally, we are able to train a generator that produces high quality anime faces with a decent yield using the WGAN-GP model.

Future Work

Further Improve Loss Function

Use improved loss function such as spectral normalization GAN[6] to fulfill 1-Lipschitz function .

Conditional GAN



The input is combined by some conditional parameters (y), such as hair colors, eye colors. Parameter y is added to both the Generator and Discriminator.
The discriminator will not only score the quality of the images, but also check whether the label is matching or not. Finally, we will have a generator that will produce images based on given input text, such as green hair, red eyes.

[6] Spectral Normalization for Generative Adversarial Networks https://arxiv.org/abs/1802.05957

References

1. Hongyi Lee, GAN 2018 https://www.youtube.com/watch?v=DQNNMiAP5lw&list=PLJV_el3uVTsMq6JEFPW35BCiOQTsoqwNw
2. DCGAN https://github.com/carpedm20/DCGAN-tensorflow
3. PyTorch-GAN https://github.com/eriklindernoren/PyTorch-GAN
4. GANHACKS https://github.com/soumith/ganhacks
5. Wasserstein GAN https://arxiv.org/abs/1701.07875
6. Improved Training of Wasserstein GANs https://arxiv.org/abs/1704.00028
7. GAN — Wasserstein GAN & WGAN-GP https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490
8. From GAN to WGAN https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html
9. Machine Learning Mastery https://machinelearningmastery.com/category/generative-adversarial-networks/