IMPLEMENTATION OF DCGAN AND WGAN-GP ON EMOJIGAN

NGUYEN, Thanh Quoc Bao – NGUYEN, Canh Huy – NGUYEN, Luong Truong Thinh

I. INTRODUCTION

Expressing emotions is essential in our everyday lives especially ones with heavy workloads and limited time for social communication. Therefore, emojis or visual representations of emotions will come in handy when we have a short time for texting and find it hard to express emotions through words. This project aims to implement EmojiGAN using DCGAN and WGAN. We will implement the different source codes and critically analyze their effectiveness on different results. The environment that supports our project is Google Colaboratory since it is easy to use and has several built-in libraries.

II. BACKGROUND INFORMATION

1. What is GAN?

GAN (short for Generative Adversarial Network) is an unsupervised machine learning method used for generating fake images which resemble real images to a surprising degree. It was first developed by Ian Goodfellow in 2014 and is gaining continuous attention in various fields (Giles, 2018). Some applications of GAN are generating realistic photographs of human faces and enhancing video resolution (Brownlee, 2019).

2. How does GAN work?

GAN contains two separate networks that continuously compete with each other, namely the Discriminator and the Generator. The Generator has the purpose of creating fake images whereas the Discriminator's goal is to distinguish between real images and fake ones that the Generator created. This helps to improve the Generator's performance when trained in the next epoch or iteration (Brownlee, 2019).

In mathematical terms, a combined loss function is defined to assess the performance of the Discriminator and the Generator (Anon., 2017). It is written as:

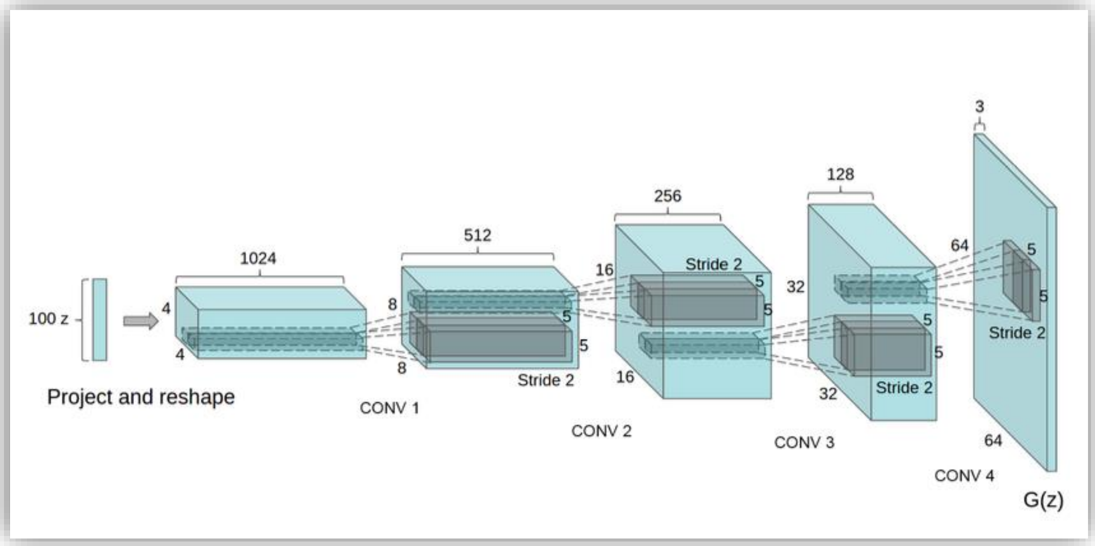
$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

Where in the first part, the Discriminator which is defined as D(x) takes in real images and outputs zero as fake and one as real. The loss function is maximized when D(x) takes in real images and must output one, hence log (1) is 0.

The Generator G(z), on the other hand, takes in some random noise and outputs fake images. The Generator's goal is to fool the Discriminator, therefore has to minimize the loss function. Whereby if G(z) successfully fools the Discriminator, the value D(G(z)) is going to be something close to one. 1 – D(G(z)) is going to be something close to zero and the log of it is some negative value (i.e.: - 0.237). Hence, the loss function is minimized.

3. What is DCGAN?

DCGAN (short for Deep Convolutional Generative Adversarial Networks), which was proposed by Alec Radford & Luke Metz in 2016, uses deep convolutional neural networks rather than the fully connected layer (Radford, et al., 2016). DCGAN is said to be more stable and generate higher quality images than normal GAN without the convoluted networks (Wolf, 2018).



The figure shows the different convolutional layers used by the generator. It takes in a random noise vector of size 100 and outputs a 64x64 image with 3 RGB channels. The Discriminator also uses the same layers but instead goes the opposite way (Shorten, 2018).

4. From GAN to WGAN

The problems with normal GAN (GAN with convoluted nets) or DCGAN are that they encounter **vanishing gradients and mode collapse**. These are the two most common problems that lead to unstable and slow training (Weng, 2017). Hence, disrupting the goal of generating realistic-looking images.

Vanishing gradients refers to a situation where the gradients of the images created by the Generator drop quickly to a point of stability. This happens because the Discriminator has been trained so well that the Generator no longer receives updates for training from the Discriminator. At this point, the loss function gravitates towards zero, thus resulting in minimal changes in updating the parameters for the next batch. As a result, the learning rate will decrease dramatically or even get jammed (Hossein, 2017).

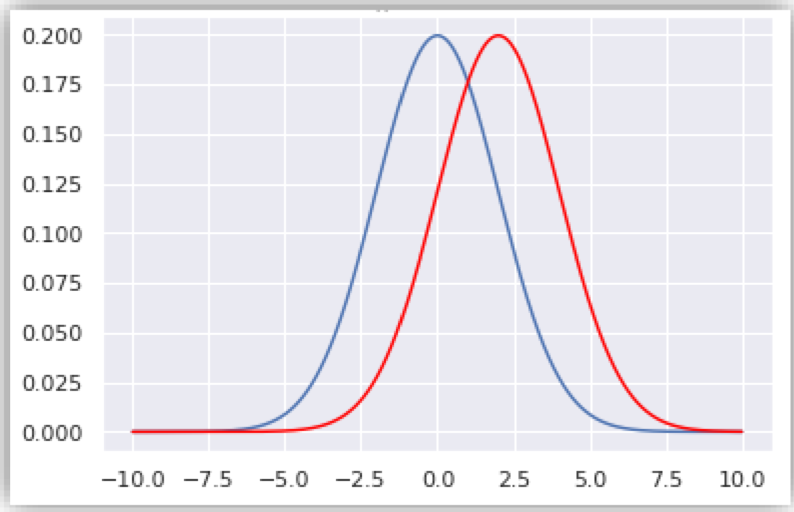
$$w^{new} := w^{old} - \eta \frac{\partial L}{\partial w}$$

The expression shows the relationship between gradients of the images created by the Generator (∂ L/ ∂ w) and the learning rate (η). When Loss (L) approximates zero, the product between the gradients and the learning rate approximates zero. This means the new weights and old weights have no large difference.

In the second problem **mode collapse**, this refers to a situation where the Generator re-creates some similar type of outputs (Shen, 2018). It is said to over-optimize the Discriminator in those images that Discriminator can never distinguish between real and fake (Anon., 2020). Hence, the output results contain repeated images that lack variety.

To prevent these kinds of problems, WGAN has been introduced to stabilize the loss and giving definition to it (for example, having termination criteria) rather than just some plain squiggly lines on a graph. WGAN (short for Wasserstein GAN), was published in a 2017 paper by Martin Arjovsky. It is a type of GAN that *"seeks an alternate way of training the generator model to better approximate the distribution of data observed in a given training dataset."* (Brownlee, 2019).

With regards to probability theory and statistics, WGAN is said to use the Wasserstein distance rather than the Jensen–Shannon or JS Divergence distance that normal GAN or DCGAN often uses. The distance here relates to two probability distributions moving towards one another (Maklin, 2019).



The figure shows two probability distributions Pθ and Pr. Where Pθ is the distribution that comes out from the Generator represented by the blue curve and Pr is the distribution from the real images. The goal is to equalize these distributions so the Generator will create realistic-looking images. In graphical terms, the closer Pg moves towards Pr, the more realistic images will be (Allingham, 2019).

In the paper, Wasserstein distance is defined as

$$W(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_\theta}[f(x)]$$

Where in the first part, x is some real images coming from the distribution Pr which are fed into the function f. In this case, f is the Discriminator whose name was changed to Critic because it no longer used a sigmoid activation function at the output layer as like DCGAN, which will not limit the output to be between 0 and 1.

In the second part, x is some fake images coming from the distribution Pθ. Before then, x was created by feeding some latent noise into the Generator and was also later fed into the Critic (Irpan, 2017).

A condition that must be met is that the Critic has to be constrained within one Lipschitz continuous. This means that the gradient norm of the Critic must be less than or equal to 1.

As referred to in the graph, the Critic wants to separate Pθ and Pr as much as possible since it wants to distinguish between real and fake images. Hence, it wants to maximize the Wasserstein distance. The Generator, on the other hand, wants to put together Pθ and Pr as close as possible since it wants to fool the Critic. Therefore, it must minimize the Wasserstein distance (Irpan, 2017).

After the course of training, the Wasserstein distance will come to the point where it becomes zero. This is the termination criteria showing that Wasserstein loss or distance has a limit and can be controlled (Irpan, 2017). Wasserstein distance equals zero means that there is approximately no difference between the real and fake images. The Generator has done a good job and was able to put the Critic in a dilemma where it is forced to choose whether an image is real, given that its probability distribution is 0.5 (Irpan, 2017).

5. From WGAN to WGAN-GP

As aforementioned, the Critic must be kept under one Lipschitz constraint, meaning that its gradient norm is less than or equal to one. To do so, a method called "weight clipping" was introduced in the WGAN original paper to keep the gradient to be one at most. This means that if the gradients rose too high resulting in accelerated weights or exploding gradients problem. The weights will be clipped to remain below one. However, not only high weights are clipped but all weights are. Therefore, relatively low weights are clipped to even under their initialization resulting in small or close to zero gradients. In other words, vanishing gradients problem (Naidoo, 2019).

Due to the concurrent problems, the "weight clipping" technique finds it hard to balance between clipping the high gradients and not making the already low gradients fall below the limit. As a savior to this dilemma, not so long since the WGAN paper was published, an improvement of WGAN was proposed with a new concept of "gradient penalty". In other words, WGAN-GP. The paper states that the gradient that rose above or under the limit would be penalized according to this expression:

$$(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\| - 1)^2$$

Which is expressed in words as *"the squared difference from norm 1"* (Gulrajani, et al., 2017).

Where x_head is *"a randomly weighted average between a real and generated sample"* (Leinonen, 2019), calculated using this formula:

$$\hat{\mathbf{x}} = \epsilon \mathbf{x} + (1 - \epsilon) G(\mathbf{z})$$

And *"ϵ is selected randomly between 0 and 1"* (Leinonen, 2019).

Thus, a new loss function is deduced for both the Critic and the Generator:

$$L_D = D(\mathbf{x}) - D(G(\mathbf{z})) + \gamma(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\| - 1)^2$$

Where LG=D(G(z)) is the Generator loss.

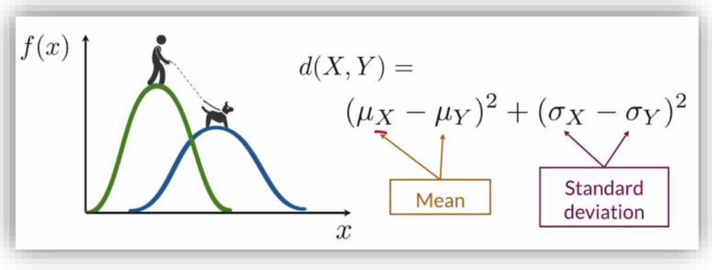
6. GAN evaluation method

It is said that GAN models have no objective methods of evaluation compared to other models. The reason is that there are two separate models, namely the Discriminator and the Generator, which are trained concurrently and the result of one supports the performance of the other to maintain an equilibrium of the loss (Brownlee, 2019).

Therefore, there are both quantitative and qualitative methods of evaluating GAN. One way of qualitative measure is just to simply look at the fake images and to see how much it resembles the real ones in terms of shape, resolution, or noise distribution. However, this is rather subjective and cannot apply to large and complicated datasets such as images in the medical field.

The two most common ways of quantitative measures seen for GAN models are the Inception Score and the Frechet Inception Distance (Brownlee, 2019), which will be used for accessing our model's performance.

FID (short for Frechet Inception Distance), named after the French Mathematician Maurice René Fréchet, is a metric to measure the distance between curves and to be extended to comparing distributions. Concretely, the Frechet Distance is the minimum least length to walk the curves from beginning to end. A formula to calculate the Frechet Distance between two single-dimensional normal distribution is (Coursera Company, Unknown publication date):



In simple words, a low FID score indicates more similarity between two groups of images. This means that the distribution curves of real and fake images are closer together. A high FID score, on the other hand, shows that the two image groups are far less similar since their curves are far away (Brownlee, 2019).

One disadvantage of the FID score is said to be dependent on the sample size. A large sample size is said to reduce the noise of the images, hence increased similarity, and lower score. Another disadvantage is FID uses the pre-trained Inception model which was previously trained on natural images of animals and objects. Therefore, it is a bit absurd to predict on pictured datasets such as emojis (Coursera Company, Unknown publication date).

The second metric, IS (short for Inception Score), measures how well a generated image looks like a group of classified images and how diverse images are generated from a dataset or just a few. In other words, the IS assesses the generated images in terms of their quality and diversity. In simple words, a low Inception Score indicates poor results with generated images that look a lot less like any group of generated images and are re-generated. With regards to the model's fault, it refers to mode collapse. A high Inception Score, on the other hand, indicates better quality and diversity (Brownlee, 2019).

The Inception Score method is said to be developed before FID and was widely used as an evaluation metric. However, it is largely replaced by FID in recent years because of several drawbacks. A major one was that only generated images are taken in to calculate Inception Score. This might seem a lot unrealistic and far-off since there was no literal comparison between the real images and their generated ones (Coursera Company, Unknown publication date).

III. TESTING AND EVALUATION

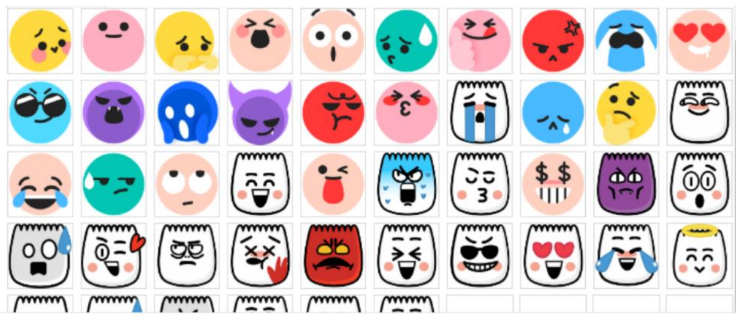
Source code No.1: Tomhata's GitHub repository.

Type: EmojiGAN using DCGAN in Keras.

What is special in this source code is that Tomhata has a part where he cleans up imported data. Since imported images of emojis are often varying in size and most have transparent backgrounds while others have white. Another modification is the usual images of 4RGBA channels will be converted to 3 RGB channels. The reason for this modification Tomhata said it was because *"computer vision usually does not play well with alpha channels"* (Tomhata, n.d.)

Three times of increasing epochs were applied on each dataset to test the performance of the source code. The emojis are seen to be clearer as the epoch increases. The Generator loss fluctuated widely as expected while the Discriminator loss kept itself within the range [0,1]. Training time was measured to be increased over the three epochs. A quantitative record of FID and Inception Score was noted as performance proof. FID score went down, and the Inception Score went up as the epochs increased (and as the dataset grows, only true to the first two).

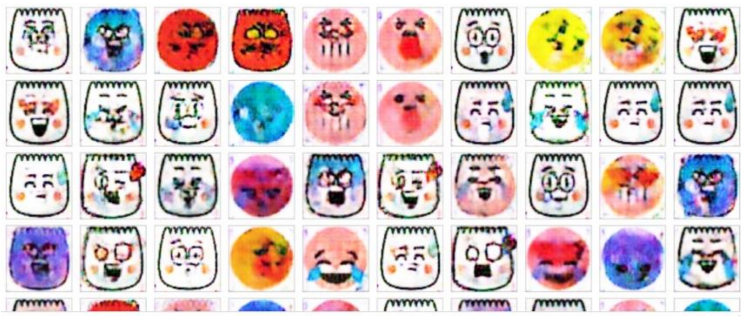
Dataset	Num_epochs	FID	IS (mean, std)
Tiktok (46 images)	5000	345.6597	1.0463, 0.0237
	10000	326.8227	1.0765, 0.0219
	20000	321.7195	1.0624, 0.0173
Docomo (526 images)	5000	267.7497	1.0121, 0.0040
	10000	229.7662	1.0148, 0.0041
	20000	213.2561	1.0150, 0.0036
Joypixels (2448 images)	5000	378.3458	1.0101, 0.0035
	10000	323.9033	1.0170, 0.0055
	20000	301.3094	1.0150, 0.0040



Tiktok emoji dataset. Tiktok is a Chinese-own social media platform that is widely used in the recent years.

Contains 46 images.

Retrieved from: https://emojipedia.org/tiktok/



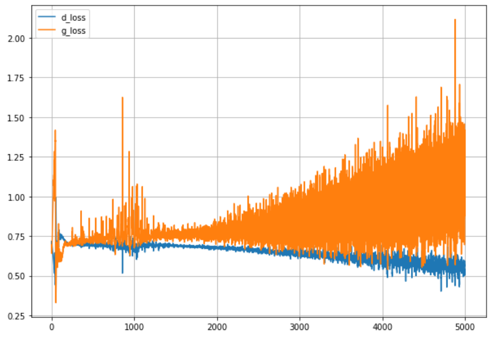
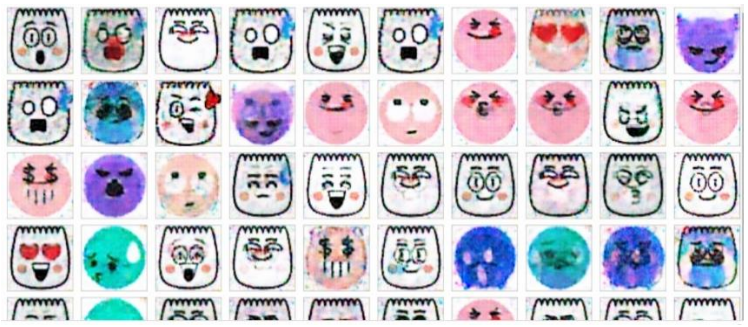


Figure 1: *5000 epochs, training complete in 42m 25s. FID = 345.6597, IS (mean: 1.0463, std: 0.0237)*



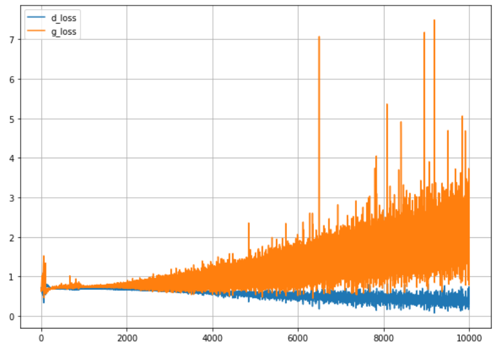
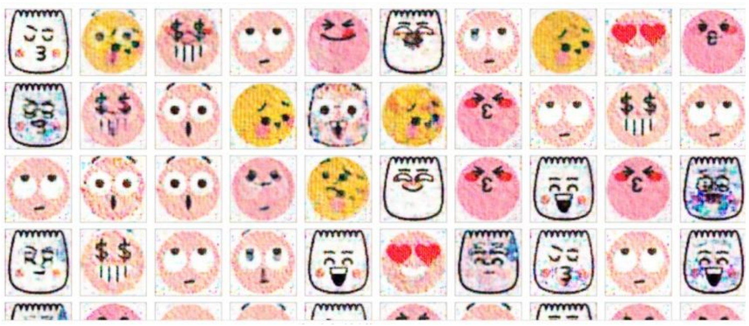


Figure 2: *10000 epochs, training complete in 1h 26m 19s. FID = 326.8227, IS (mean: 1.0765, std:0.0219)*



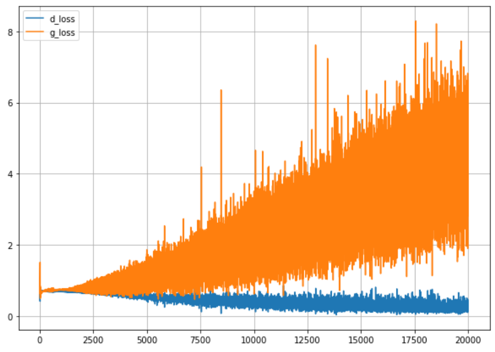
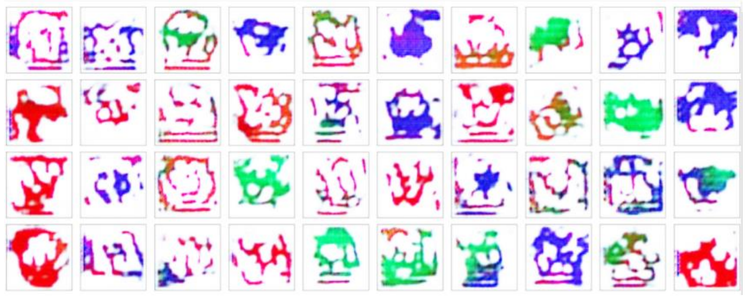


Figure 3: *20000 epochs, training complete in 2h 58m 28s. FID = 321.7195. IS (mean: 1.0624, std:0.0173)*

Docomo 2013 emojis dataset. Developed by Japansese phone producer NTT Docomo Inc.

Contains: 526 images

Retrieved from https://emojipedia.org/docomo/



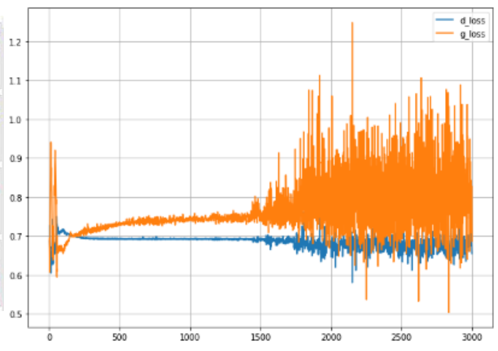
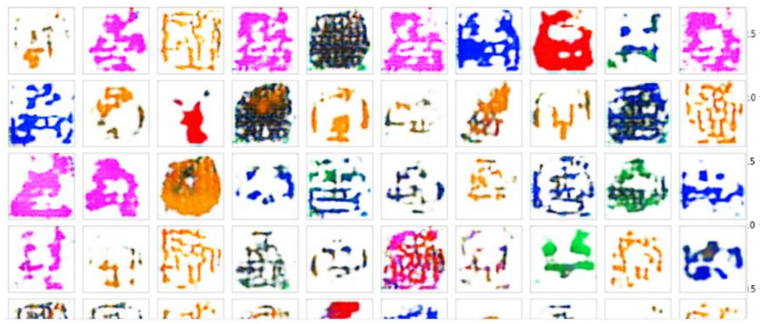


Figure 1: *3000 epochs, training complete in 25m 38s. FID = 267.7497, IS (mean: 1.0121, std: 0.0040)*



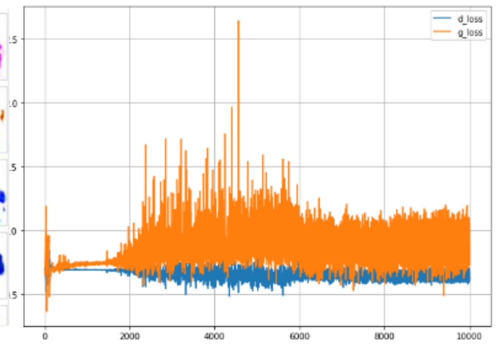
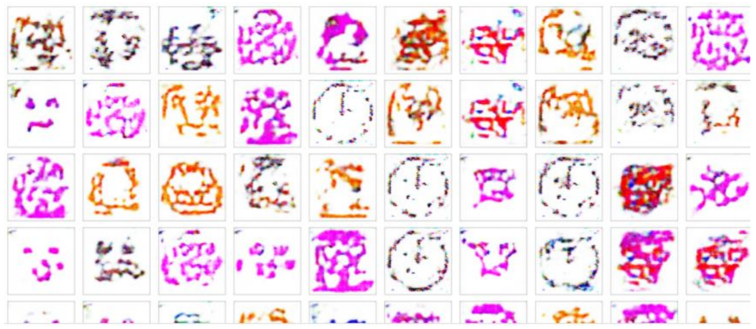


Figure 2: *10000 epochs, training complete in 2h 56m 51s. FID = 229.7662, IS (mean:1.0148, std:0.0041)*



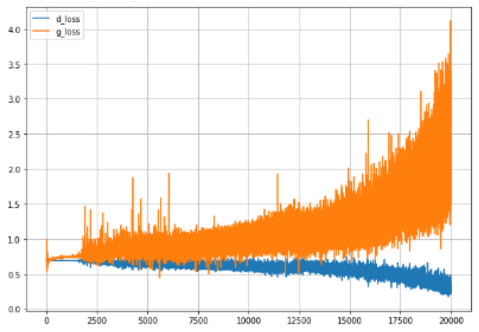
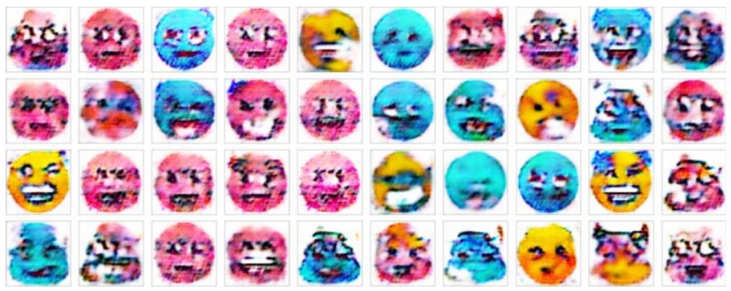


Figure 3*: 20000 epochs, training complete in 2h 46m 56s. FID = 213.2561, IS (mean: 1.0150, std: 0.0036)*

Joypixels emoji dataset

Contains 2448 images.

Retrieved from: https://www.joypixels.com/



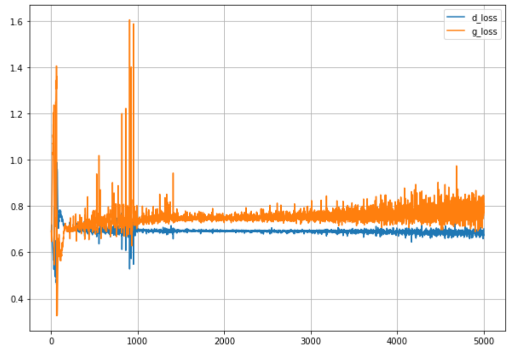
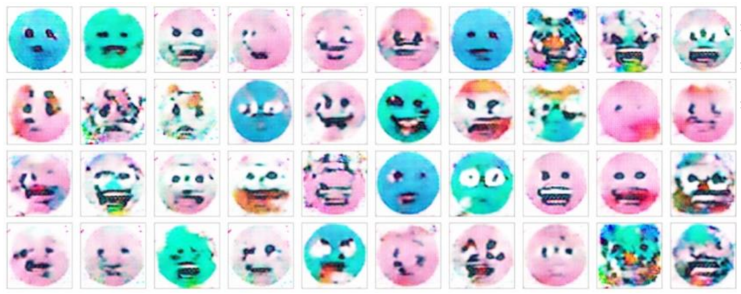


Figure 1: *5000 epochs, training completes in 43m 56s. FID score = 378.3458, IS (mean: 1.0101, std: 0.0035)*



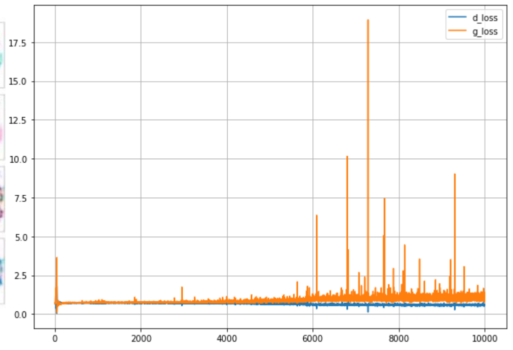
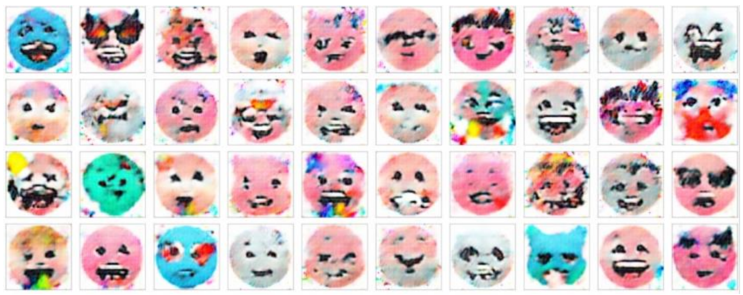


Figure 2: *10000 epochs, training completes in 1h 28m 44s. FID score = 323.9033, IS (mean: 1.0170, std: 0.0055)*



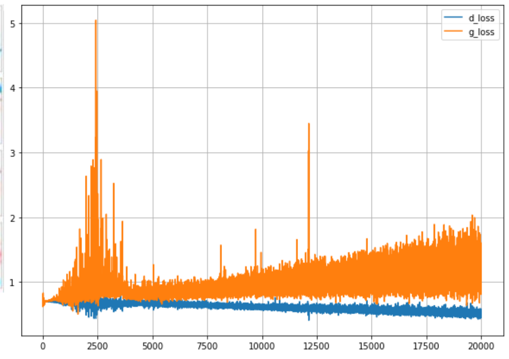


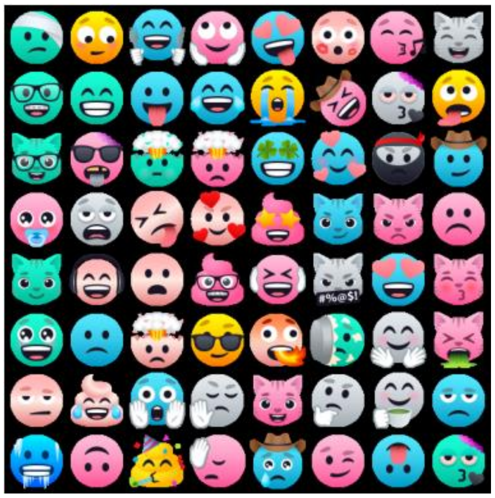
Figure 3: *20000 epochs, training completes in 2h 46m 21s. FID = 301.3094, IS (mean: 1.0150, std: 0.0040)*

The Generator loss is observed to fluctuate widely, and the Discriminator loss tends towards zero. This means that the model might encounter vanishing gradients when continues to train on larger epochs. Therefore, we have decided to use pure DCGAN source code to see if it works on emoji images.

Source code No.2: DCGAN implemented in Pytorch.

The Discriminator and Generator loss looks like they support each other since one does not spread into two different directions as in the Keras source code. However, there emerges a period where the two losses underwent stability, and one separates the other. In one case where the Discriminator loss was seen as zero and the Generator loss continues at its own rate but idle. This might indicate gradients vanishing since the Generator received very little or no feedback from the Discriminator. The FID and IS scores were also seen absurd as one increased and the other decreased as the epochs accelerated. On a brighter note, the generated images seem to become clearer and less noise on larger epochs. But the problem of mode collapse remains since a lot of images in the output were re-generated. When using a pre-trained model or a checkpoint for resuming training, the result was much better.

Dataset (Joypixels)	Num_epochs	FID	IS (mean, std)
Emoji_01_02 (2448 images)	1000	324.1234	1.0000, 0.0000
	2000	306.6026	1.0000, 0.0000
Emoji_01_02 (2448 images) has checkpoint	1000	272.6589	1.0522, 0.0142
	2000	265.8675	1.1065, 0.0509
Emoji_03 (446 images) has checkpoint	1000	438.8504	nan, nan
	2000	429.9377	nan, nan
	2000	424.2836	nan, nan



Real Images from Joypixels emoji dataset. Contains 2448 images. Retrieved from: https://www.joypixels.com/

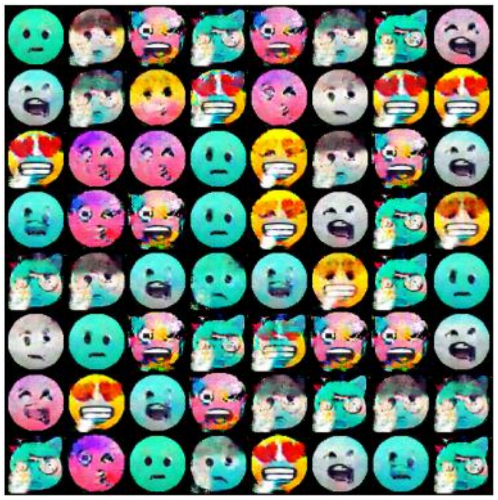


Figure 1: 1000 epochs

training completes in 1h 49m 19s

FID = 324.1234

IS (mean: 1.0000, std: 0.0000)

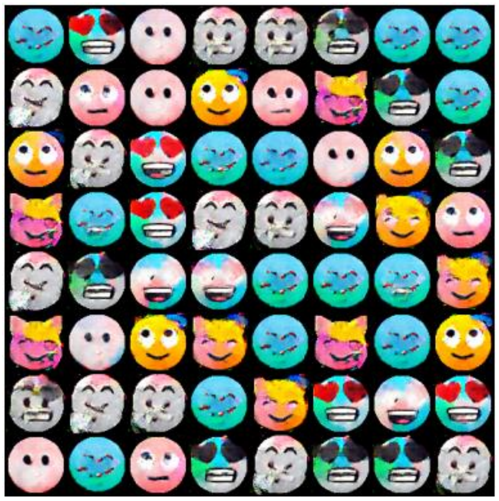
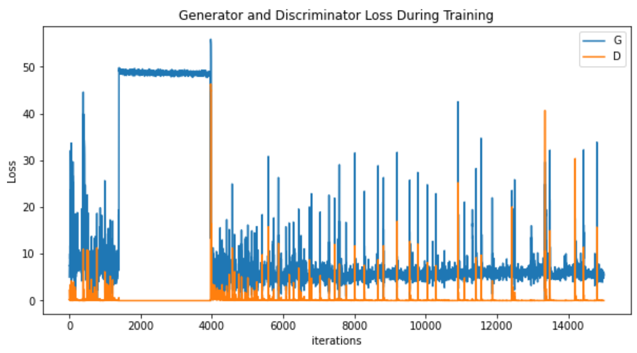


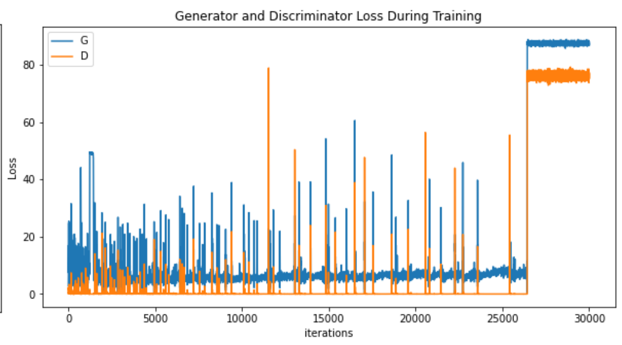
Figure 2: 2000 epochs

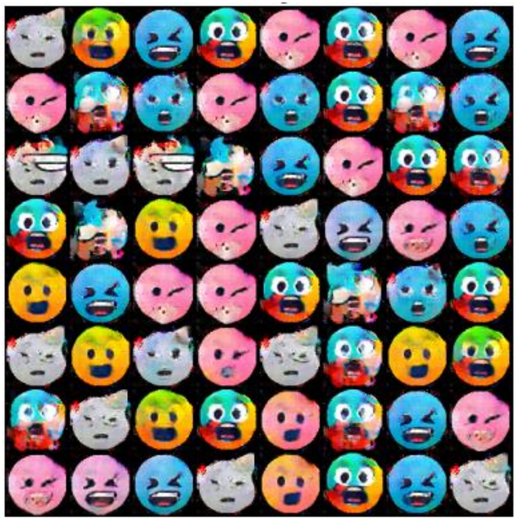
training completes in 3h 10m 25s

FID score: 306.6026

IS (mean: 1.0000, std: 0.0000)







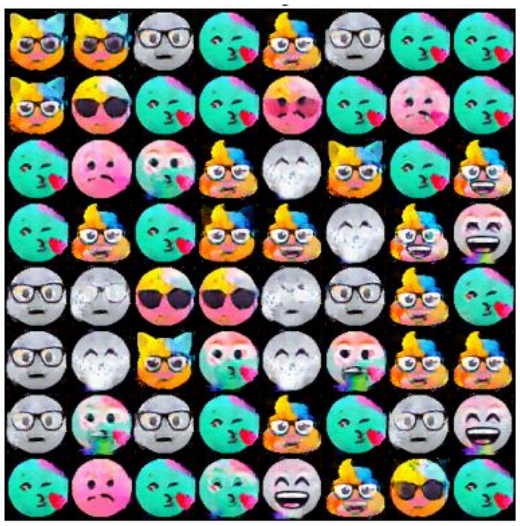


Figure 1: 1000 epochs

training completes in 1h 39m 18s

FID = 272.6589

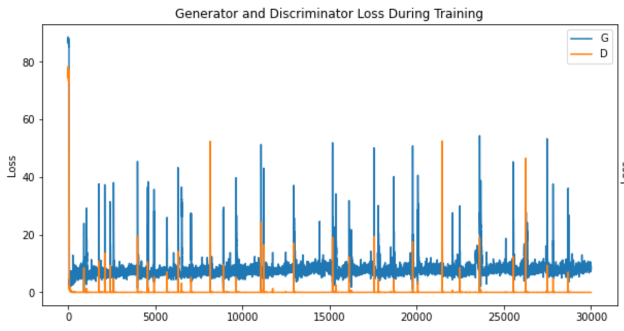
IS (mean: 1.0522, std: 0.0142)

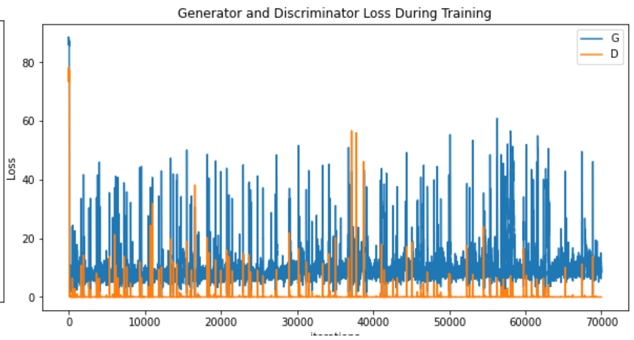
Figure 2: 2000 epochs

training completes in 3h 55m 48s

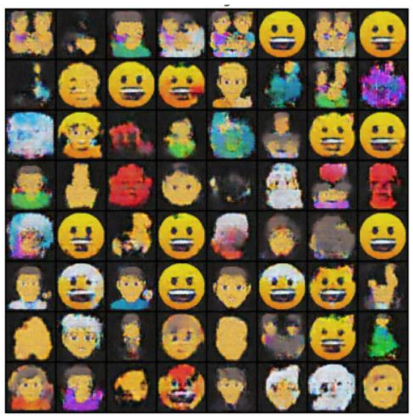
FID = 265.8675

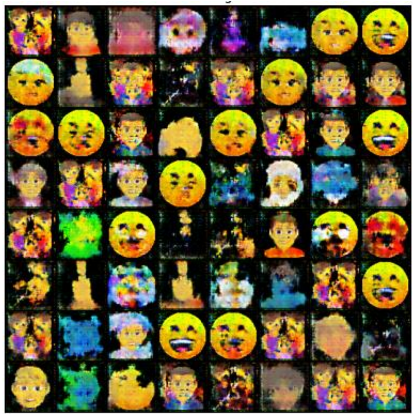
IS (mean: 1.1065, std: 0.0509)





Real Images from Joypixels emoji dataset. Contains 466 images. Retrieved from: https://www.joypixels.com/





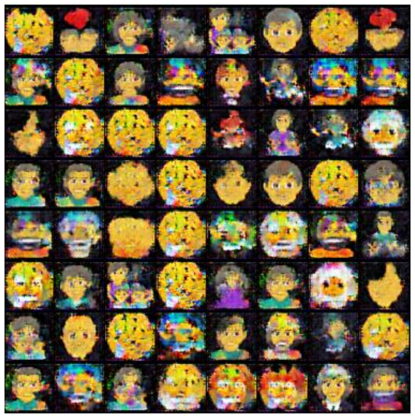


Figure 1: 1000 epochs

training completes in 43m 9s

FID = 438.8504

Inception Score: nan, nan

Figure 2: 2000 epochs

training completes in 42m 42s

without pretrained

FID = 429.9377

Inception Score: nan, nan

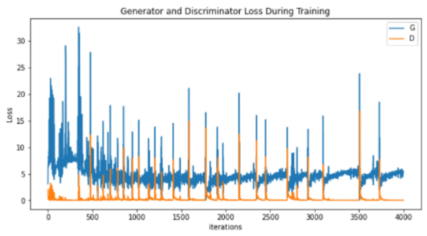
Figure 3: 2000 epochs

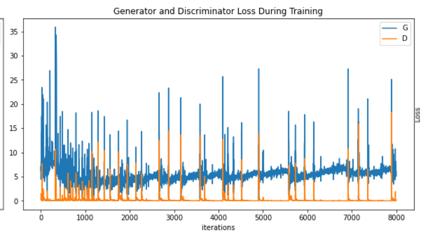
training completes in 42m 32s

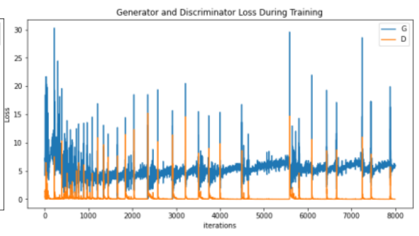
with pretrained

FID = 424.2836

Inception Score: nan, nan







To prevent mode collapse, gradients vanishing, and to improve the quality of images, we have shifted to WGAN.

Source code No.3: EmilienDupont's GitHub repository.

Type: WGAN-GP implemented on Pytorch framework.

The results show that the Critic and the Generator loss differed greatly from one another. The Critic loss only accounted for a very little percentage on the graph while the Generator loss rampages widely. However, its rise and fall look more stable and more densely crafted than it was on DCGAN. The gradient norm was proven to be steadily less than or equal to one since it must follow one Lipschitz constraint. The gradient penalty clearly indicates the Generator loss was penalized if it felt below zero. Although the parameters have proven to be indicators of a WGAN, the generated images do not look like expected. As a result, the FID score is relatively high indicating generated images look a lot less like their originals. Besides, even though the training is expected to take a lot more time than DCGAN, the results came as a disappointment.

Dataset	Num_epochs	FID	IS (mean, std)
Tiktok emojis (46 images)	200	476.1332	1.0015, 0.0007
Joypixel Emoji_03 (466 images)	200	401.6556	1.0037, 0.0005
	1000	406.2647	1.0008, 0.0001

This leads us to a new source code for WGAN which is also implemented in Pytorch.

Source code No.4: Aladdin Persson's GitHub repository.

This source code has proven to be better as the images can now be made out. Training time is still long as expected for WGAN. However, the Critic and Generator loss have displayed some strange shapes. The Critic loss dived steadily at the beginning but soon picked up as a curve and remained stable at some negative value below zero.

Source code No.3: EmilienDupont's GitHub repository.

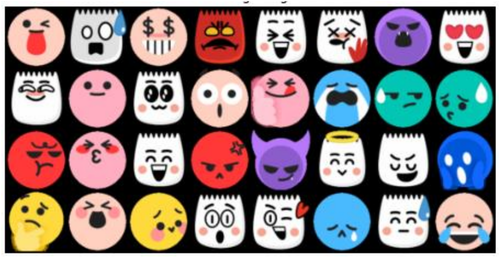


Figure 1: Tiktok emoji dataset (46 images). Real.

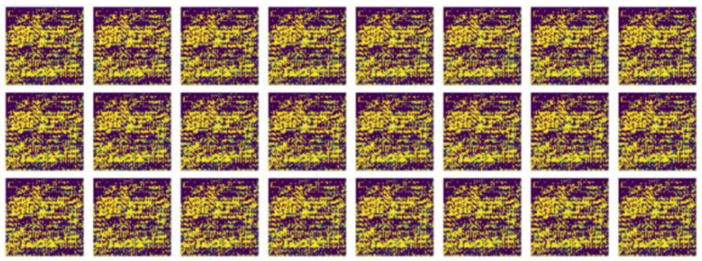


Figure 2: Generated images on 500 epochs.

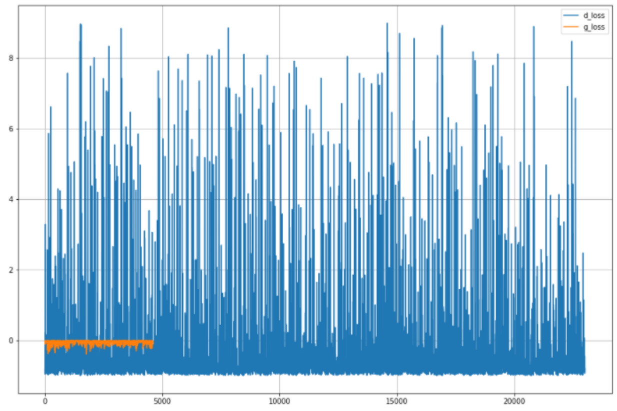


Figure 3: Critic loss vs Generator loss.

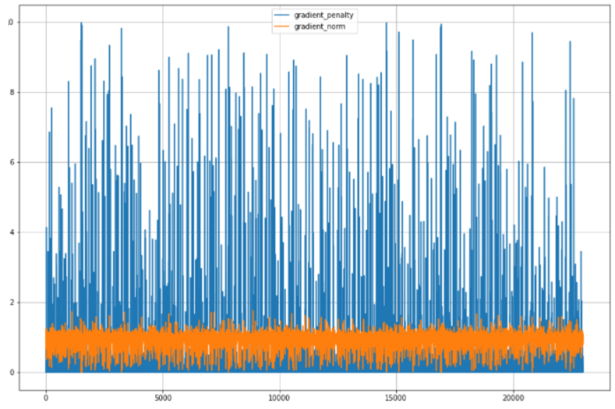


Figure 4: Gradient penalty vs Gradient norm

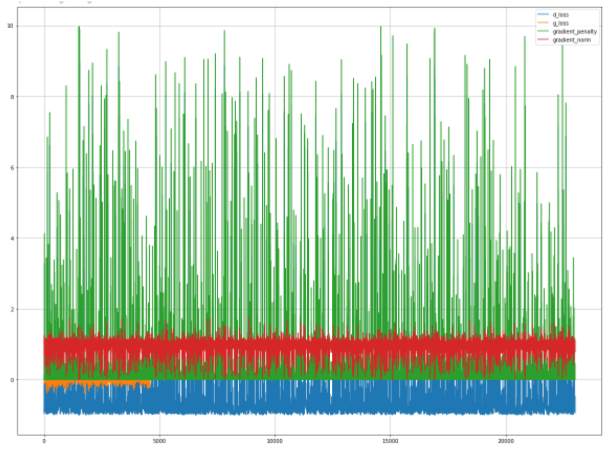


Figure 5: Critic loss (blue), Generator loss (orange) Gradient penalty (green), Gradient norm (red).

Source code No.3: EmilienDupont’s GitHub repository.

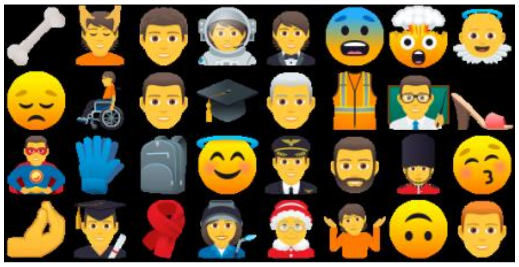


Figure 1: Joypixel emoji dataset (466 images). Real.

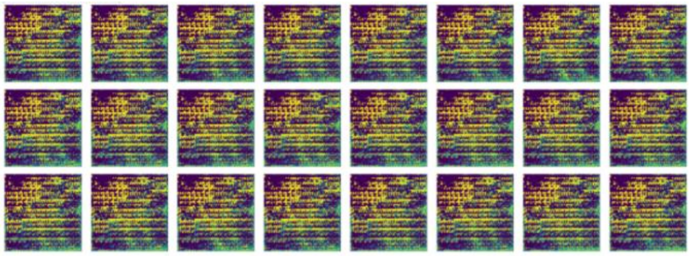


Figure 2: Generated images on 200 epochs, completes after 42m 27s.

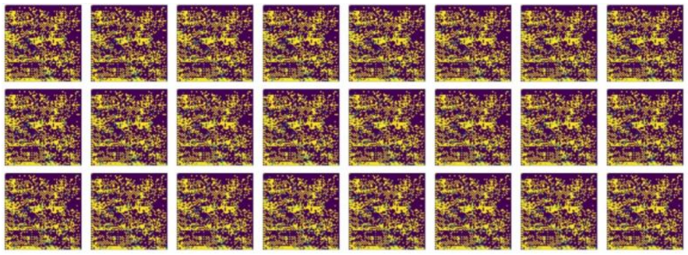
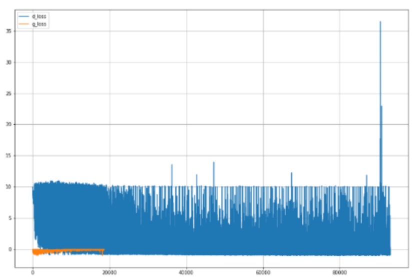
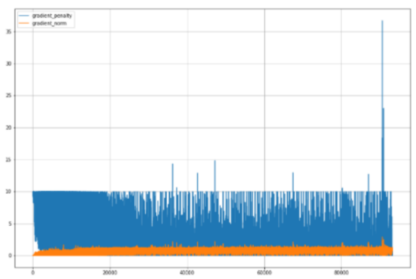


Figure 3: Generated images on 1000 epochs, completes after 3h 27m 6s

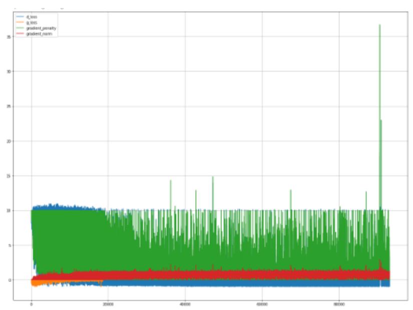
Critic vs Gen loss (200 epochs)



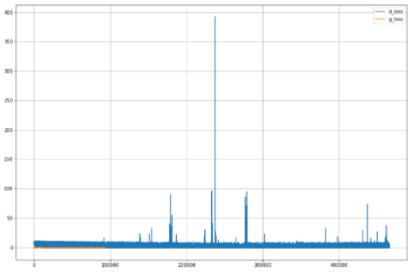
Gradient Pen vs Norm (200 epochs)



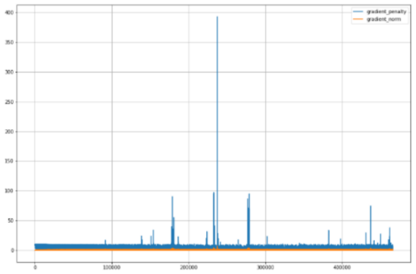
Combined (200 epochs)



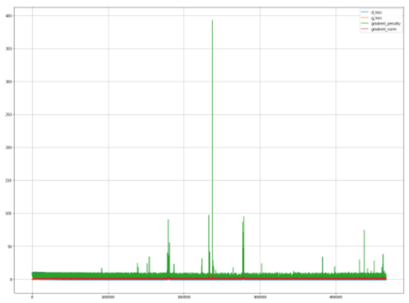
Critic vs Gen loss (1000 epochs)



Gradient Pen vs Norm (1000 epochs)



Combined (1000 epochs)



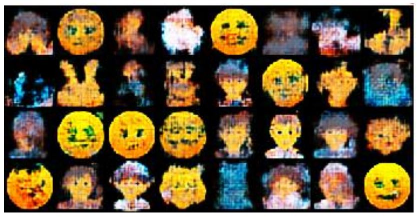
The Generator loss, on the other hand, has soon risen to relatively high values, adopted a similar stable yet slightly fluctuated trend than those for the Critic.

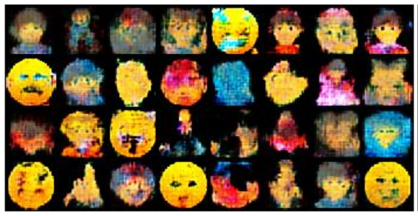
Dataset (Joypixel)	**Num_epochs**	**FID**	**IS (mean, std)**
Emoji_03 (466 images)	200	353.9229	1.0000, 0.0000
	500	323.0977	1.0000, 0.0000
	1000	297.6788	1.0000, 0.0000
Emoji_01_02 (2448 images)	200	302.9240	1.0942, 0.0357
	400	302.6925	1.1104, 0.0455

Compare and contrast.

Source code and Attribute	**Time trained**	**No. of image**	**No. of epoch**	**Image Quality**
No.1	5 secs	Not affected.	10 epochs	A lot of images have noise and re-generated.
No.2	1 min	Depends. Large datasets result in slow training.	10 epochs	Some images have noise and re-generated.
Between the two DCGAN source codes (No.1 and No.2), **No.1 has proven to better.** The reason is that images of any size and background are formatted before training, so **inputs were coherent, which resulted in faster training**. Whereas No.2 could not work with varying size inputs.				
No.3	2 min	Depends. Large datasets result in slow training.	10 epochs	Poor results. Shapes are not even made out. All noise!
No.4	48 secs	Depends. Large datasets result in slow training.	10 epochs	Still has a lot of noise but can be improved on larger epochs.
Between the two DCGAN source codes (No.3 and No.4), **No.4 has proven to better.** The images can be made out and its FID score is lower. For source code No.3, the FID score did not go over the limit indicating some degree of similarity. It was probably not trained long enough.				

Real Images from Joypixels emoji dataset. Contains 466 images. Retrieved from: https://www.joypixels.com/





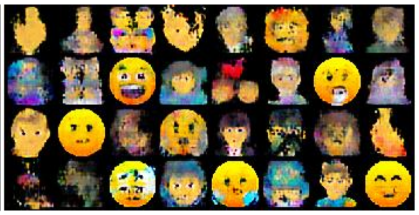


Figure 1: 200 epochs

completes after 16m 7s

FID = 353.9229

IS (mean: 1.0000, std: 0.0000)

Figure 2: 500 epochs

completes after 40m 18s

FID = 323.0977

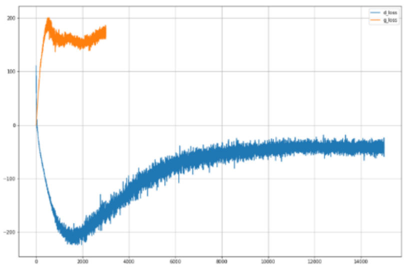
IS (mean: 1.0000, std: 0.0000)

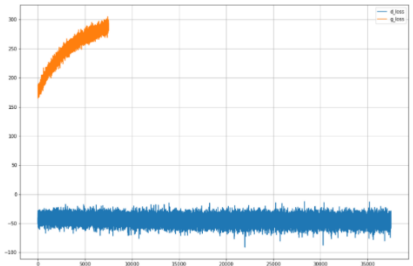
Figure 3: 1000 epochs

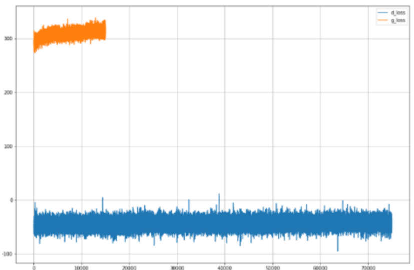
completes after 1h 18m 1s

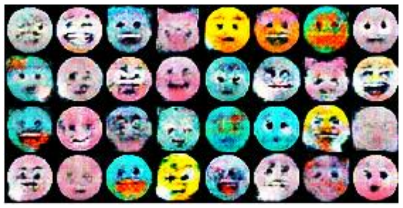
FID = 297.6788

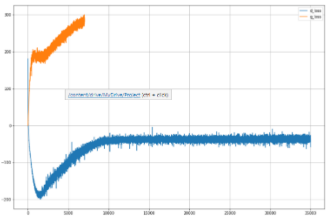
IS(mean: 1.0000, std: 0.0000)







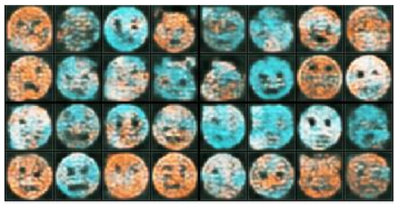


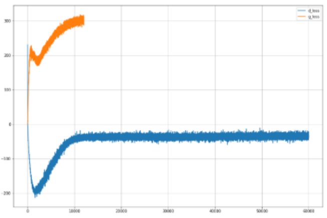


Emoji_01_02 (466 images) Real.

200 epochs. 1h 2m 12s. Fake.

400 epochs. 1h 40m 20s. Fake





IV. CONCLUSION

In conclusion, the project implements a total of 4 source codes, one in Keras and three in Pytorch. Two in DCGAN and two in WGAN. Evaluation and comparison between them have been drawn. It cannot be denied that each source code has its own strengths and weaknesses. For example, the one in Keras was designed to implement on emoji images so it has pre-processing input images of any size. In terms of image quality, it is feasible to say that neither DCGAN nor WGAN is better than the other for a reason that training might not be done long enough. FID and IS metrics are just two popular methods of evaluating GAN, thus cannot be made to conclusion as one model is worse than the other. Training GAN is hard as it must take into consideration many factors to make it work well. Thanks to Google Collaboratory to have supported our project. It has been an exhausting yet meaningful journey.

V. REFERENCE

Scholar paper

Gulrajani, I. et al., 2017. *Improved Training of Wasserstein GANs.* [Online] Available at: https://arxiv.org/abs/1704.00028 [Accessed 18 June 2021]

Radford, A., Metz, L. & Chintala, S., 2016. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* [Online]
Available at: https://arxiv.org/abs/1511.06434 [Accessed 22 June 2021]

Weng, L., 2017. *From GAN to WGAN.* [Online]
Available at: https://arxiv.org/abs/1904.08994 [Accessed 15 June 2021]

GitHub repository

Tomhata, n.d. *Tomhata Emojigan.* [Online]
Available at: https://github.com/tomhata/emojigan [Accessed 15 June 2021]

EmilientDupont, 2017. *GitHub.* [Online]
Available at: https://github.com/EmilienDupont/wgan-gp [Accessed June 2021].

Persson, A., 2020. *GitHub.* [Online]
Available at: https://github.com/aladdinpersson/Machine-LearningCollection/tree/master/ML/Pytorch/GANs/4.%20WGAN-GP [Accessed 24 June 2021]

Website

Allingham, J., 2019. *Wasserstein GAN.* [Online]
Available at: https://www.depthfirstlearning.com/2019/WassersteinGAN [Accessed 15 June 2021]

Anon., 2017. *Machine Learning Crash Course - Loss Functions.* [Online]
Available at: https://developers.google.com/machine-learning/crash-course/prereqs-and-prework
[Accessed 21 June 2021]

Anon., 2020. *Machine Learning Crash Course - GAN - Common Problems.* [Online]
Available at: https://developers.google.com/machine-learning/gan/problems
[Accessed 12 June 2021]

Brownlee, J., 2019. *18 Impressive Applications of Generative Adversarial Networks (GANs).* [Online]
Available at: https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/ [Accessed 15 June 2021]

Brownlee, J., 2019. *A Gentle Introduction to Generative Adversarial Networks (GANs).* [Online]
Available at: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
[Accessed 23 June 2021]

Brownlee, J., 2019. *How to Evaluate Generative Adversarial Networks.* [Online]
Available at: https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/
[Accessed 19 June 2021]

Brownlee, J., 2019. *How to Implement the Inception Score (IS) for Evaluating GANs.* [Online]
Available at: https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/ [Accessed 22 June 2021]

Brownlee, J., 2019. *How to Implement Wasserstein Loss for Generative Adversarial Networks.* [Online]
Available at: https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks/ [Accessed 16 June 2021]

Coursera Company, Unknown publication date. *Build Better Generative Adversarial Networks (GANs).* [Online]
Available at: https://www.coursera.org/learn/build-better-generative-adversarial-networks-gans#faq
[Accessed 23 June 2021].

Giles, M., 2018. *The GANfather: The man who's given machines the gift of imagination.* [Online]
Available at: https://www.technologyreview.com/2018/02/21/145289/the-ganfather-the-man-whos-given-machines-the-gift-of-imagination/ [Accessed 14 June 2021]

Hossein, 2017. *What is vanishing gradient?.* [Online]
Available at: https://stats.stackexchange.com/q/301752 [Accessed 23 June 2021]

Irpan, A., 2017. *Read-through: Wasserstein GAN*. [Online]
Available at: https://www.alexirpan.com/2017/02/22/wasserstein-gan.html [Accessed 17 June 2021]

Leinonen, J., 2019. *The Elements of GANs, Part 2: Wasserstein GANs and the Gradient Penalty.* [Online]
Available at: https://jleinonen.github.io/2019/11/07/gan-elements-2.html [Accessed 15 June 2021]

Maklin, C., 2019. *KL Divergence Python Example.* [Online]
Available at: https://towardsdatascience.com/kl-divergence-python-example-b87069e4b810
[Accessed 23 June 2021]

Naidoo, S., 2019. *Wasserstein GAN Notes - Week 5.* [Online]
Available at: https://www.depthfirstlearning.com/assets/wgan_notes/week5.pdf
[Accessed 18 June 2021]

Shen, K., 2018. *Measuring Mode Collapse in GANs.* [Online]
Available at: https://wandb.ai/authors/DCGAN-ndb-test/reports/Measuring-Mode-Collapse-in-GANs--VmlldzoxNzg5MDk [Accessed 20 June 2021]

Shorten, C., 2018. *DCGANs (Deep Convolutional Generative Adversarial Networks).* [Online]
Available at: https://towardsdatascience.com/dcgans-deep-convolutional-generative-adversarial-networks-c7f392c2c8f8 [Accessed 16 June 2021]

Wolf, S., 2018. *ProGAN: How NVIDIA Generated Images of Unprecedented Quality*. [Online]
Available at: https://towardsdatascience.com/progan-how-nvidia-generated-images-of-unprecedented-quality-51c98ec2cbd2 [Accessed 21 June 2021]