POKEGAN Experiments

2.1 Model

WGAN: Wasserstein GAN is intended to improve GANs' training by adopting a smooth metric for measuring the distance between two probability distributions, called "Wasserstein Distance". In this model, they show that the stability of learning can be improve and one can get rid of problems like mode collapse also providing meaningful learning curves useful for debugging and hyperparameter searches. It is much more robust than standard GAN.[1]

Compared to the original GAN algorithm, the WGAN undertakes the following changes:

- After every gradient update on the critic function, clamp the weights to a small fixed range, [−c,c][−c,c].
- Use a new loss function derived from the Wasserstein distance, no logarithm anymore. The "discriminator" model does not play as a direct critic but a helper for estimating the Wasserstein metric between real and generated data distribution.
- Empirically the authors recommended RMSProp optimizer on the critic, rather than a momentum based optimizer such as Adam which could cause instability in the model training. I haven't seen clear theoretical explanation on this point through.
- Also after choosing WGAN I realized that learning rate is much slower than regular ones, thus training could be slow.

2.2 Dataset

Dataset consists of generation 2 type pokemons[2] which is a total of 251 pokemons. I hoped that trying to work with pokemons would be much more faster than training a model with photographs; since pokemons have less details, less colour and less number of pixels. But I ended up with a very long training anyway.

2.3 Task

The purpose is to create new pokemons by using only 251 generation-2 pokemons as a real image input to discriminator.

2.4 Experiments

First Experiment

First thing with the experiment is that it was so slow that I could not be able to understand whether it will converge or not. I leave it for almost 20 hours and it was only at epoch #820. Since the original code is up to 5000 epochs, I did not want to interrupt the training in case it converges. I could not be able to restore a pre-saved model to keep training continue, so I did not want to risk it. I ended this trial at epoch #820, below there are some examples.

Generated images seems to have a purpose but 820 epochs were not be enough to see that

[1] Martin Arjovsky, Soumith Chintala and Léon Bottou: Wasserstein Generative Adversarial Networks, Proceedings of the 34nd International Conference on Machine Learning

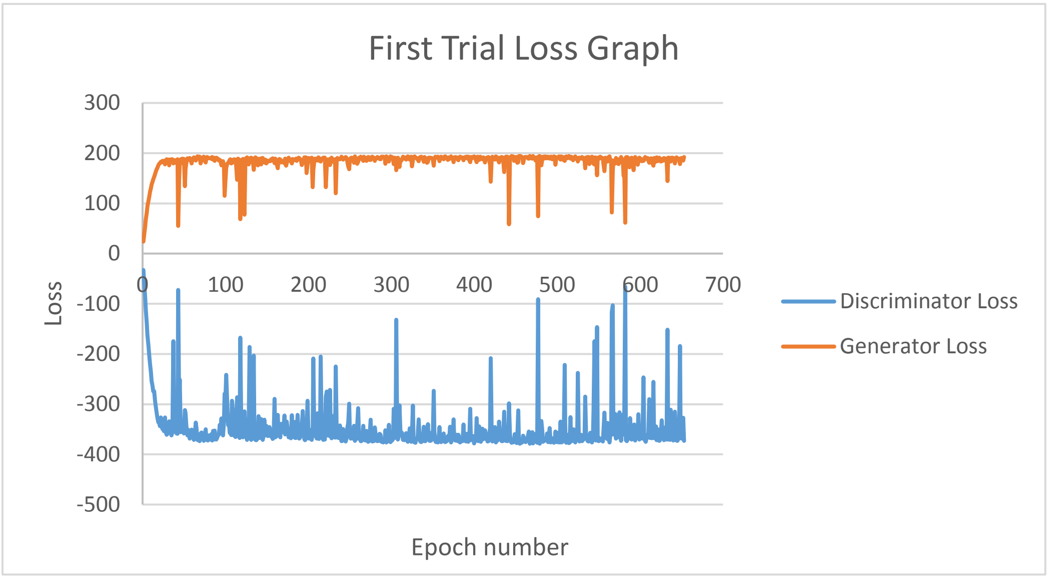
[2] https://bulbapedia.bulbagarden.net/wiki/List_of_Pokémon_by_National_Pokédex_number

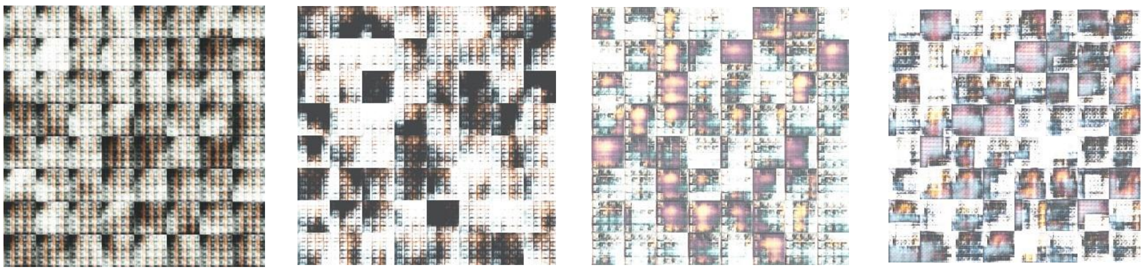
- Batch Size : 64, Learning Rate : 0.0001

* "first_exp" folder contains the model, ipynb with loses; given at my google drive link[3]

After examining the generator/discriminator loss values, I decided to cut it off because of having fluctuations only at same high loss values and not converging anywhere else. The losses were on the order of 300's and -190's for the discriminator and the generator, respectively.

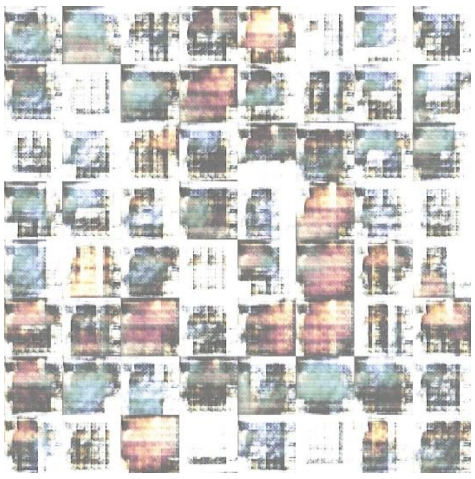
When I drew the graph losses (excel), not converging behavior has shown more easily :





#150 #350 #550 #650

[3] https://drive.google.com/drive/folders/1NxYMeI2kq69vXaIZFl41BcHBwCoUVQ6P?usp=sharing

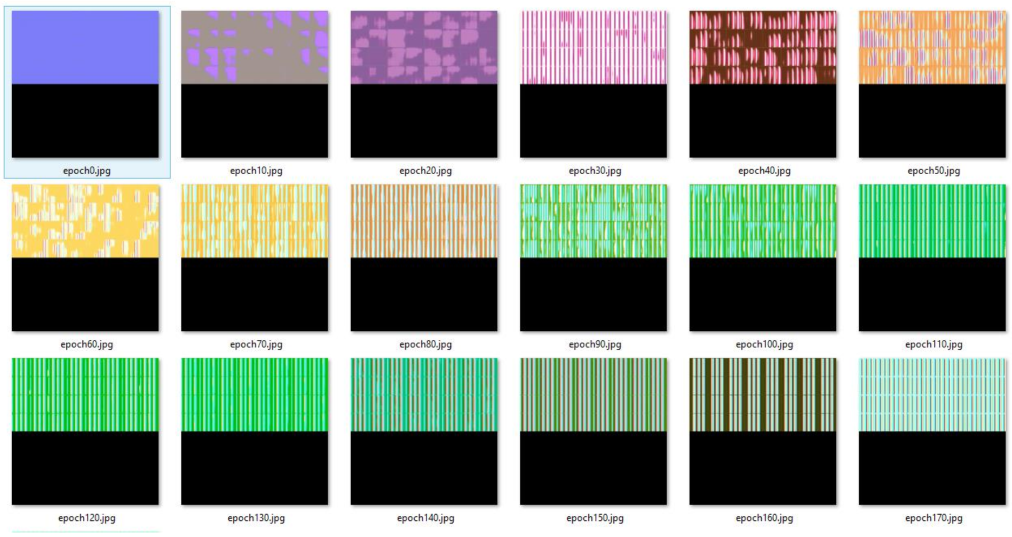


#810

Second Experiment

Secondly, learning rate is increased. I know that GANs, especially WGANs should have low learning rates but since my code was slow, I want to try that and see the effect. As you said in today’s lesson, I realized that I changed only the generator’s learning rate so it resulted in a disconnection between the two (g and d). Discriminator never created good results and generator has never managed to produce good results :

- Batch Size : 32, Learning Rate : 0.01



* “second_exp” folder contains related images and losses.

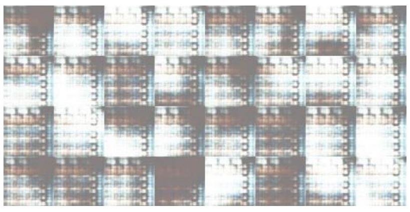
I did not include losses here, since it was very fluctuated and unrelated as also can be seen in produced images above.

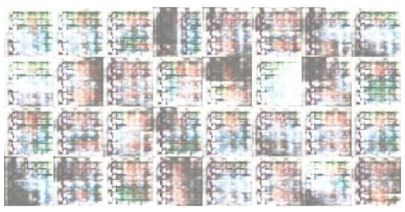
Third Experiment

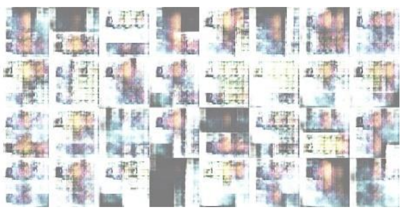
To decrease the computational load, I decreased the batch size to 32 (from 64) and run the training again (like the first experiment). Although resulted generated images seems to trying to reach a point more quickly, the loss results were the same. After training 5 hours, cut it at epoch #410.

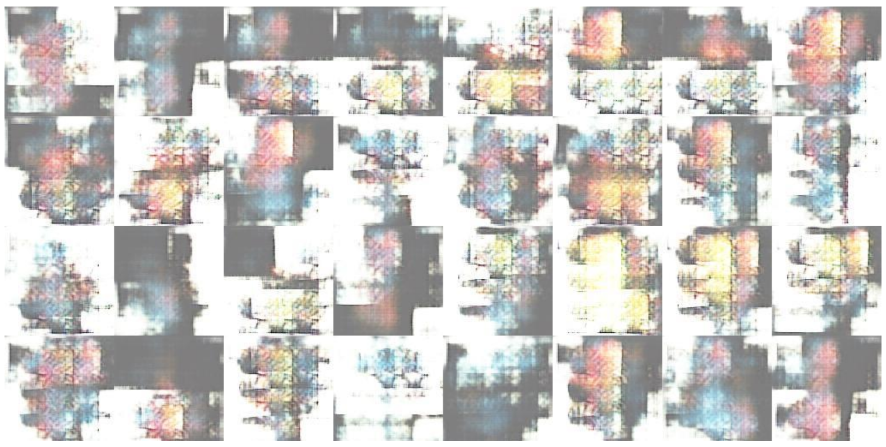
- Batch Size : 32, Learning Rate : 0.0001

* “third_exp” folder contains related images and losses.


#100


#200


#300


#410

Last Experiment and Comments

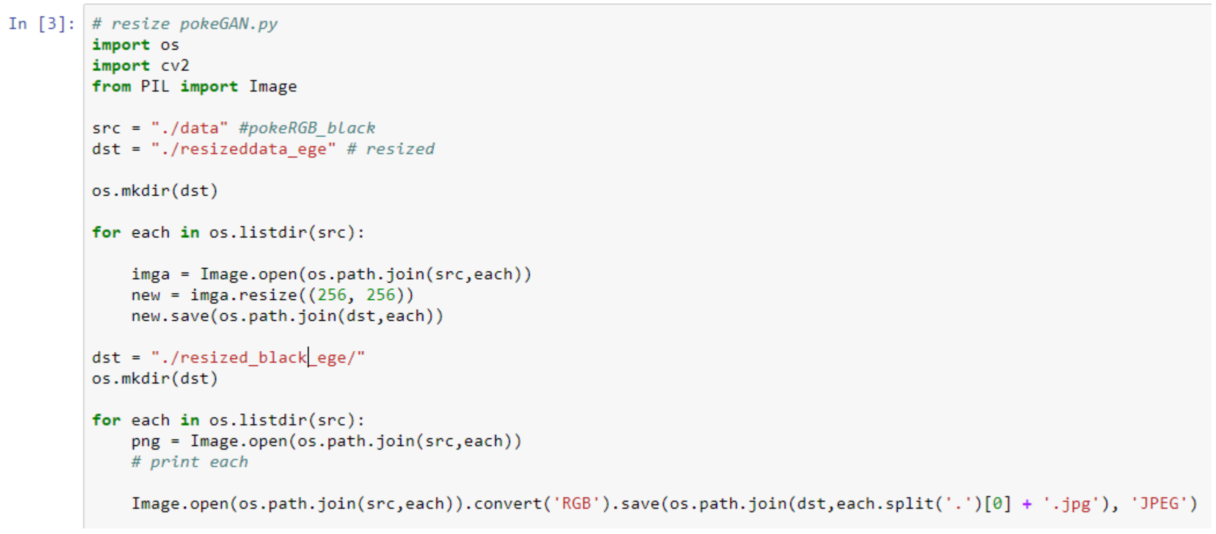
I checked my gpu usage multiple times thinking this much slow training cannot be true with using gpu, but the function “tf.test.gpu_device_name()” gave me an output of : /gpu:0, so I think these hours may be ordinary for GAN trainings.

I wish I can do the whole code by myself from scratch but even in this case, it have been a nice experience seeing all these trial and errors. I think this assignment took minimum 20-25 hours of me.

After some final investigation, I discover that while converting raw pokemon picture data (most of them like 1280*1280 pix) to smaller size images with “resize.py”, besides making the data small ; there are some distortions in the image. Making the background white may decreasing the size of the dataset but there are also distortions at the black parts :

So not all the images are clean like the bulbasaur:

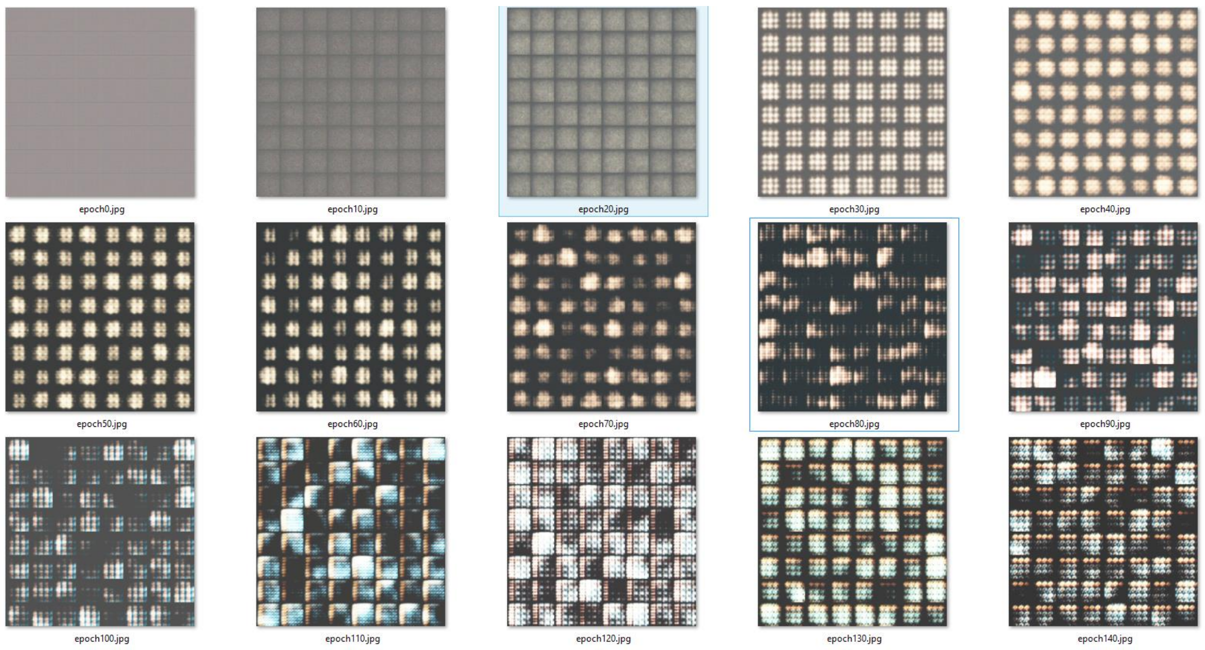
Thus, I changed the preprocessing codes and rewrote it. This time using PIL, pictures are resized to 256*256, and then later converted to jpeg for further compression and memory gain.



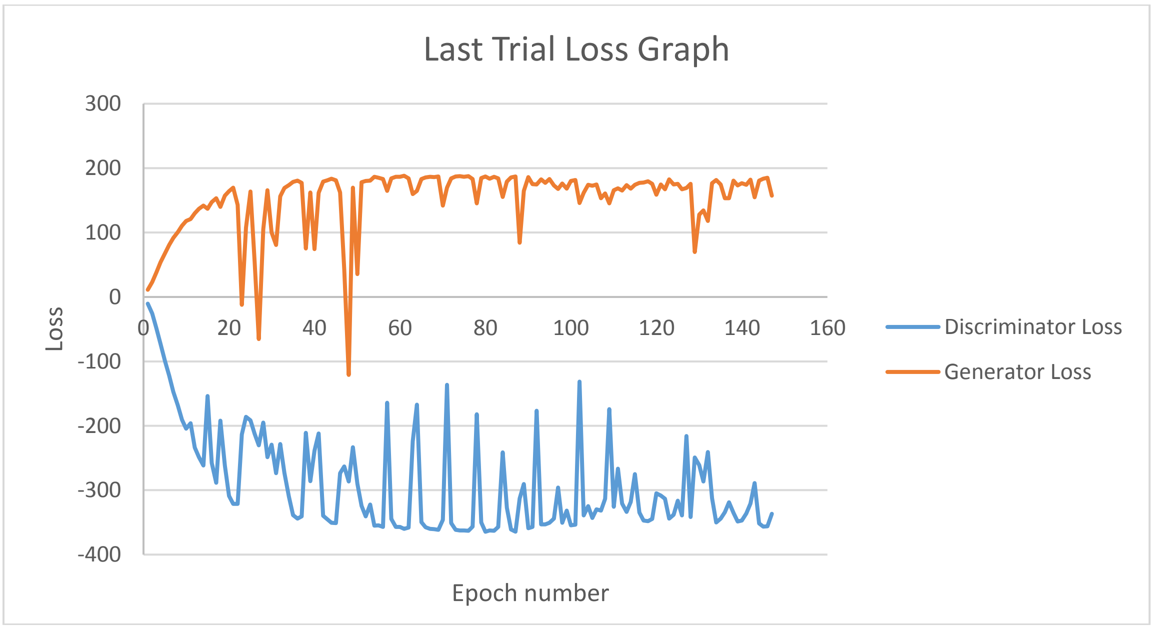
```
In [3]: # resize pokeGAN.py
        import os
        import cv2
        from PIL import Image

        src = "./data" #pokeRGB_black
        dst = "./resizeddata_ege" # resized

        os.mkdir(dst)

        for each in os.listdir(src):

            imga = Image.open(os.path.join(src,each))
            new = imga.resize((256, 256))
            new.save(os.path.join(dst,each))

        dst = "./resized_black_ege/"
        os.mkdir(dst)

        for each in os.listdir(src):
            png = Image.open(os.path.join(src,each))
            # print each

            Image.open(os.path.join(src,each)).convert('RGB').save(os.path.join(dst,each.split('.')[0] + '.jpg'), 'JPEG')
```

So that I could see the effect of input data/size/quality.

- Learning rate and batch size is still 0.0001 and 64.



I could not evaluate the time passed for the task objectively but it is at epoch #150, and couple of hours passed. It looks like following the same path as the first experiment. Loss graph is below.



(Related files are at the google drive link shared at page 2, and original github repo is here : *https://github.com/llSourcell/Pokemon_GAN*

Some of the Web References

https://github.com/llSourcell/Pokemon_GAN/blob/master/Generative%20Adversarial%20Networks.ipynb

Goodfellow, Ian. "Slides: NIPS 2016 Tutorial: Generative Adversarial Networks." http://www.foldl.me/uploads/2015/conditional-gans-face-generation/paper.pdf

Gauthier, J. “Conditional generative adversarial nets for convolutional face generation”