**Pokemon Characters Creation with Generative Adversarial Networks**

**Yek Yi Wei**

**School of Computing**

**Singapore Polytechnic**

**yekluke@gmail.com**

**Abstract**

Automatic generation of facial images has been well studied after the Generative Adversarial Network (GAN) came out. There exists some attempts applying the GAN model to generate Pokemon characters, but none of the existing work gives a promising result. In this work, I would like to explore the training of GAN models to generate Pokemon characters. It is hope that this work serves as base model for further model improvement.

**1 Introduction**

The first generation (Generation I) of the [*Pokémon* franchise](https://en.wikipedia.org/wiki/Pok%C3%A9mon) features the original 151 fictional species of creatures introduced to the [core video game series](https://en.wikipedia.org/wiki/Pok%C3%A9mon_(video_game_series)) in the 1996 [Game Boy](https://en.wikipedia.org/wiki/Game_Boy). Pokemon characters gained their popularity when the augmented reality mobile game Pokemon Go was released in 2016. Since then, we see Pokemon characters everywhere, in advertisement, educational books, toys and etc. Pokémon was a key part of my childhood. Young and old tempted to draw and create the custom ones. However, it takes tremendous efforts to master the skill of drawing, after which we are ﬁrst capable of designing our own characters. To bridge this gap, the generation of Pokemon characters not only offers an opportunity to bring a custom character into existence without professional skill but also benefit a professional creator in designing the anime and new characters. Generative Adversarial Networks, or GANs for short, is one of the newest fields in machine learning. One of its applications is image generation. a number of people have tried to generate Pokemon characters with limited success. This study attempt to explore the generation of Pokemon characters using GANs.

**2. Related Works**

**2.1 Applications of GANs**

Machine learning has become one of the most important tools in data science, and generative adversarial network (GAN) learning is one of the newest fields in machine learning. GANs are an exciting and rapidly changing field, the ability to generate realistic examples across a range of problem domains, most notably in image-to-image translation tasks such as translating photos of summer to winter or day to night, and in generating photorealistic photos of objects, scenes, and people that even humans cannot tell are fake.

Applications of Generative Adversarial Networks (GANs) are discussed [2] as below. In the original paper by Ian Goodfellow, et al. (2014), GANs were used to generate new plausible examples for the MNIST handwritten digit dataset, the CIFAR-10 small object photograph dataset, and the Toronto Face Database. Follow by 2015 paper titled “[Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks](https://arxiv.org/abs/1511.06434)” by Alec Radford, et al., they demonstrated models for generating new examples of bedrooms with Deep Convolutional Gan (DCGAN). In 2017, Tero Karras, et al. in their paper titled “[Progressive Growing of GANs for Improved Quality, Stability, and Variation](https://arxiv.org/abs/1710.10196)” demonstrate the generation of plausible realistic photographs of celebrity faces. They are so real looking, in fact, that it is fair to call the result remarkable. As such, the results received a lot of media attention. The face generations were trained on celebrity examples, meaning that there are elements of existing celebrities in the generated faces. Andrew Brock, et al. in their 2018 paper titled “[Large Scale GAN Training for High Fidelity Natural Image Synthesis](https://arxiv.org/abs/1809.11096)” demonstrate the generation of synthetic photographs with their technique BigGAN that are practically indistinguishable from real photographs. Yanghua Jin, et al. in their 2017 paper titled “[Towards the Automatic Anime Characters Creation with Generative Adversarial Networks](https://arxiv.org/abs/1708.05509)” demonstrate the training and use of a GAN for generating faces of anime characters (i.e. Japanese comic book characters).

**2.2 Generative Adversarial Networks (GANs)**

The fundamental aspect of GAN is the min-max two-person zero-sum game. In this game, one player takes the advantages at the equivalent loss of the other player. Here, the players correspond to different networks of GAN called discriminator and generator. The main objective of the discriminator consists of determining whether a sample belongs to a fake distribution or real distribution. Where as, generator aims to deceive the discriminator by generating fake sample distribution. Discriminator produces the chances or probability of a given sample to be a real sample. A higher value of probability shows that the sample is likely to be a real sample. We have to stop training when it attains the Nash Equilibrium or D(x) = 0.5 for all x. In simple words, when the generated images look almost like real images. Both the generator and the discriminator are neural networks. The basic architecture is shown in Figure 1. The generator output is connected directly to the discriminator input. Through [backpropagation](https://developers.google.com/machine-learning/glossary#backpropagation), the discriminator's classification provides a signal that the generator uses to update its weights. The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying.

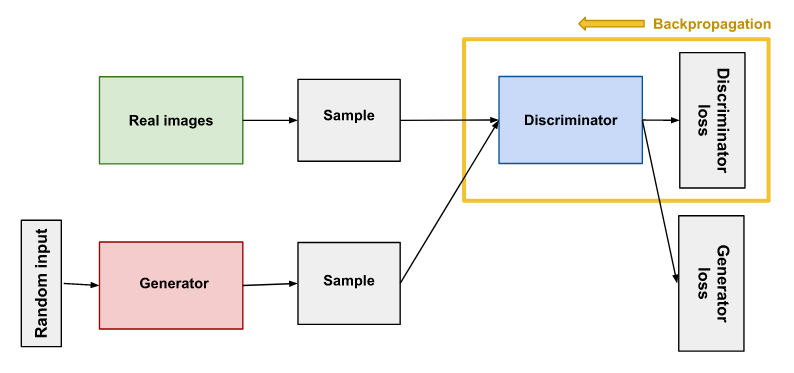
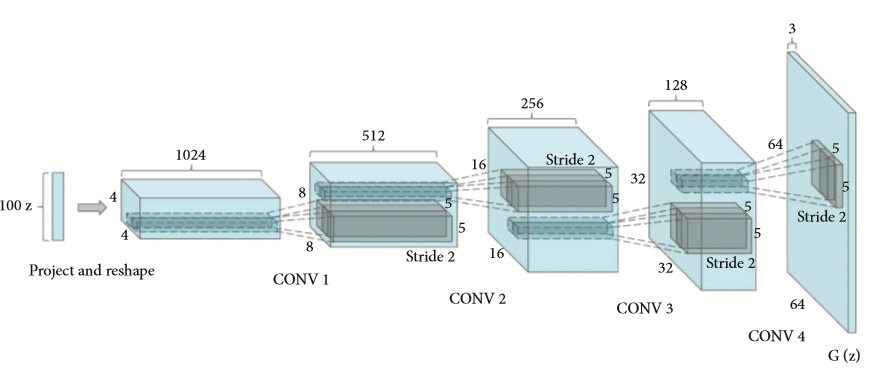


Figure 1**. Backpropagation in discriminator training.**

## Researchers continue to find improved GAN techniques and new uses for GANs. They are GAN variations [3] discussed. First, Progressive Gan, the generator's first layers produce very low resolution images, and subsequent layers add details. This technique allows the GAN to train more quickly than comparable non-progressive GANs, and produces higher resolution images. Second, Conditional GANs train on a labelled data set and let you specify the label for each generated instance. For example, an unconditional MNIST GAN would produce random digits, while a conditional MNIST GAN would let you specify which digit the GAN should generate. Third, Image-to-Image translation GANs take an image as input and map it to a generated output image with different properties. Fourth, CycleGANs learn to transform images from one set into images that could plausibly belong to another set. Fifth, Text-to-image GANs take text as input and produce images that are plausible and described by the text. Sixth, Super-resolution GANs increase the resolution of images, adding detail where necessary to fill in blurry areas. Seventh, chunks of an image are blacked out, and the system tries to fill in the missing chunks in the face inpainting task. Not all GANs produce images. For example, researchers have also used GANs to produce synthesized speech from text input.

**2.3 Deep Convolutional Generative Adversarial Networks (DCGAN)**

Radford et al. [4] proposed a new class of CNNs called Deep Convolutional Generative Adversarial Networks (DCGANs) having certain architectural constraints. These constraints involved adopting and modifying three changes to the CNN architectures (i) removing fully-connected hidden layers and replacing the pooling layers with strided convolutions on the discriminator and fractional-strided convolutions on the generator, (ii) using batch normalization on both the generative and discriminative models, (iii) using ReLU activations in every layer of the generative model except the last layer and LeakyReLU activations in all layers of the discriminative model. Compared with traditional GANs,the salient feature of the DCGAN is that a CNN is used to replace the multilayer perceptron. The pooling layer and sampling layer are removed in the CNN model. The convolution layer is used to discriminate the image in the discriminator, and the deconvolution layer is used to generate the image in the generator. Figure 2 illustrates the network structure of DCGAN generator [5]. The speciﬁc structure of the DCGAN generator is as follows: the input layer is followed by a batch normalization layer (which can hasten the convergence of the model), and the reshaping layer is used to normalize the preliminary data; then, an upsampling layer, a Conv2DTranspose layer, and a batch normalization layer are used to sample, deconvolute, and normalize the data, respectively.



**Figure 2. The network structure of DCGAN generator.**

**3. Construction of the Image Generation Models**

**3.1. Data Sources**

I am using [a Kaggle dataset created by user kvpratama](https://www.kaggle.com/kvpratama/pokemon-images-dataset) [5], with images of over 819 Pokemon 819 transparent Pokemon images in png format size 256x256. Since there is a limited number of unique Pokemon (around 800), some augmentation technique will be used to generate more training dataset. First, all the image will be flip horizontally. Then all images (original and flipped) is rotated 3, 5, and 7 degrees clockwise and counter-clockwise. The training set will be the combination of original, flipped, and rotated images. All images will be reshaped to 64x64 pixels with a white background. If an image is in png format and has a transparent background (i.e. RGBA), it will be converted to jpg format with RGB channel.

**3.2 Methods**

This paper uses the Keras framework with Tensorflow as the back end. The Keras framework is an open source artificial neural network library written in Python. Deep Convolution GAN or DCGAN is utilized to generate new Pokemon images. The network structure for the discriminator is given in Table 1, the discriminator network will be trained to discriminate between the original and generated image. The generator g, which is trained to generate image to fool the discriminator, is trained to generate image from a random input. In DCGAN architecture, the generator is represented by convolution networks that upsample the input. The goal is to process the small input and make an output that is bigger than the input. It works by expanding the input to have zero in-between and then do the convolution process over this expanded area. The convolution over this area will result in larger input for the next layer. Table 2 display the network structure for the generator. The hyperparameter for DCGAN architecture is given in the Table 3.

Table 1: The network structure for the discriminator

|  |  |  |
| --- | --- | --- |
| **Layer** | **Shape** | **Activation** |
| input | batch size, 3, 64, 64 |  |
| convolution | batch size, 64, 32, 32 | LRelu |
| convolution | batch size, 128, 16, 16 | LRelu |
| convolution | batch size, 256, 8, 8 | LRelu |
| convolution | batch size, 512, 4, 4 | LRelu |
| dense | batch size, 64, 32, 32 | Sigmoid |

Table 2: The network structure for the generator

|  |  |  |
| --- | --- | --- |
| **Layer** | **Shape** | **Activation** |
| input | batch size, 100 (Noise from uniform distribution) |  |
| reshape layer | batch size, 100, 1, 1 | Relu |
| deconvolution | batch size, 512, 4, 4 | Relu |
| deconvolution | batch size, 256, 8, 8 | Relu |
| deconvolution | batch size, 128, 16, 16 | Relu |
| deconvolution | batch size, 64, 32, 32 | Relu |
| deconvolution | batch size, 3, 64, 64 | Tanh |

Table 3: Hyperparameter for DCGAN

|  |
| --- |
| **Hyperparameter** |
| Mini-batch size of 64 |
| Weight initialize from normal distribution with std = 0.02 |
| LRelu slope = 0.2 |
| Adam Optimizer with learning rate = 0.0002 and momentum = 0.5 |

# 4. Results and Performance

# To analyze the model performance, we can look at how well the generator and discriminator grappled with each other throughout the training cycle. Ideally, the discriminator is able to distinguish images over time. The lower the losses for each model over time the better it is. Refer to the Discriminator and Generator Losses in Figure 3, the losses for both models was relatively good throughout. The discriminator was prevented from hitting 0 by the noisy labels, which made it possible for the generator to not lose out over time. Figure 4 showed the scores the discriminator gave to the generated data and the real data. A lower score indicates the discriminator thinking the image is real. Over time, the discriminator is able to start to tell which Pokemon are real or fake. This is pretty good, but it could be better if the discriminator didn’t figure out the generator so fast.

# Chart, histogram Description automatically generated

# Figure 3. Discriminator and Generator Losses

# Chart, line chart, histogram Description automatically generated

# Figure 4. Scores of Discriminator and Generator

# 

# Figure 5 Generated Pokemon images

# 5. Discussion and Conclusion

# This study obviously unable to generate clear Pokemon images (Figure 5). The results don’t scream Pokemon to me, but the process to fine-tuning the images can start from here. GANs are difficult to get right, but I think this is a good first introduction and something to expand on in the future. For further exploration, I will try to prepare the dataset by sorting the Pokemon by types (fire, water, flying, rock, etc.) or by dominant colour (brown, yellow, green, and etc).  I want to create bigger kernel sizes at the 64x64 image size and see if I get better results. I also may increase the number of filters in this image size to get more detailed images.

# References

[1] Jason, B. 2019. A Gentle Introduction to Generative Adversarial Networks (GANs). Retrieved from <https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/>. Last assessed 30/01/2023

# [2] Jason, B. 2019. 18 Impressive Applications of Generative Adversarial Networks (GANs). Retrieved from<https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/>. Last assessed 02/02/2023

## [3] Google Machine Learning Education, Learn to build ML products with Google's Machine Learning Courses. Retrieved from <https://developers.google.com/machine-learning/gan/applications>. Last assessed 05/02/2023

[4] Radford A, Metz L, Chintala S (2015) Unsupervised representation learning with deep convolutional generative adversarial networks. Available at: arXiv:1511.06434v2 [cs.LG] 7 Jan 2016.

[5] J. Viola, Y. Q. Chen, and J. Wang, “Faultface: deep convolutional generative adversarial network (DCGAN) based ball-bearing failure detection method,” Information Sciences, vol. 542, no. 4, pp. 195–211, 2021.