PokéGANs

Eidan Jacob and Catherine Tsang

Sunday April 28, 2019

1 Introduction

Can machines really think?

Artificial Intelligences (AIs) have "out-smarted" humans in Go, chess, Texas Hold'em poker, and more.[1] But despite their abilities to analyze and strategize, a question arises on whether or not they truly understand their actions. This brings to mind Richard P. Feynman, 20th century physicist, who wrote: "What I cannot create, I do not understand."[1]

Following this reasoning, we can answer the question of whether or not machines can really "understand" based on their ability to synthesize new material. In this project, we will explore machine understanding through image generation—in particular, by using Generative Adversarial Networks (GANs) to create new Pokémon.

Currently, the most high-profile open-source GAN for generating Pokémon is a Wasserstein GAN (WGAN) by the GitHub user moxiegushi.[2] We evaluate the effectiveness of this GAN and propose & test the efficacy of certain changes in order to generate Pokémon of greater realism and diversity.

2 Related Work

2.1 Generative Adversarial Networks

Generative Adversarial Networks, also known as GANs, are a popular method of machine learning in unsupervised and semi-supervised environments. In this kind of network, a generator creates synthetic data. The synthetic data, alongside real data, is fed to a discriminator. The basic infrastructure of this network can be found in Figure 1.

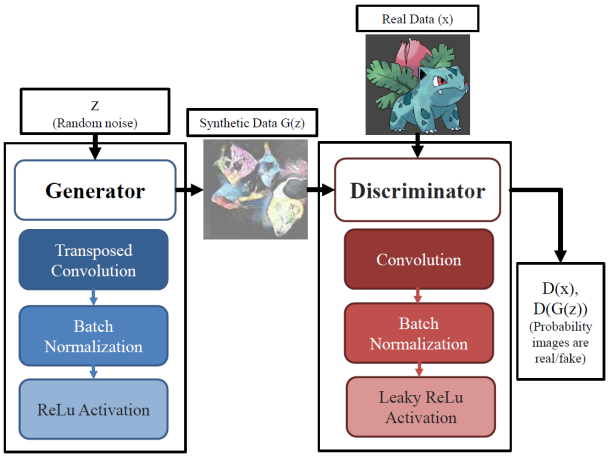

Figure 1: PokéGAN structure. The black arrows indicate the general flow of the GAN. The blue and red show the direction of the generating and discriminating process. Images all taken from moxiegushi's GitHub repository.[3]

Through this interaction, the generator and discriminator act as "adversaries"–the discriminator must learn to differentiate between real and synthetic data, and the generator must learn how to create more realistic data in order to fool the discriminator. What results is a "minimax" game, in which the optimal result occurs when Equation 1 is satisfied.[4]

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \tilde{} p_{data}(x)}[logD(x)] + \mathbb{E}_{z \tilde{} p_z(z)}[log(1 - D(G(z)))]$$

Equation 1: Minimax game for generator and discriminator of GAN.

[1] Caltech Archives. A Photograph of Richard P. Feynman's blackboard.
[2] Moxiegushi. pokeGAN. 2017a. https://github.com/moxiegushi/pokeGAN.
[3] Ibid.
[4] H. Huang, P. S. Yu, and C. Huang. An Introduction to Image Synthesis with Generative Adversarial Nets. 2018. https://arxiv.org/pdf/1803.04469.

2.2 Deep Convolutional Neural Networks

In GANs, it is common utilize Convolutional Neural Networks (CNN) as the discriminator component. The design of CNNs is based on the biology of our our brains. In this architecture, information only flows from inputs to outputs. Similarly to visual cortexes, layers of simple and complex processing allow our brains to understand the information that our visual system receives. These layers, which perform convolution and pooling in a GAN, can be grouped together in modules. More specifically, a deep CNN occurs when modules are stacked on top of each other.[5]

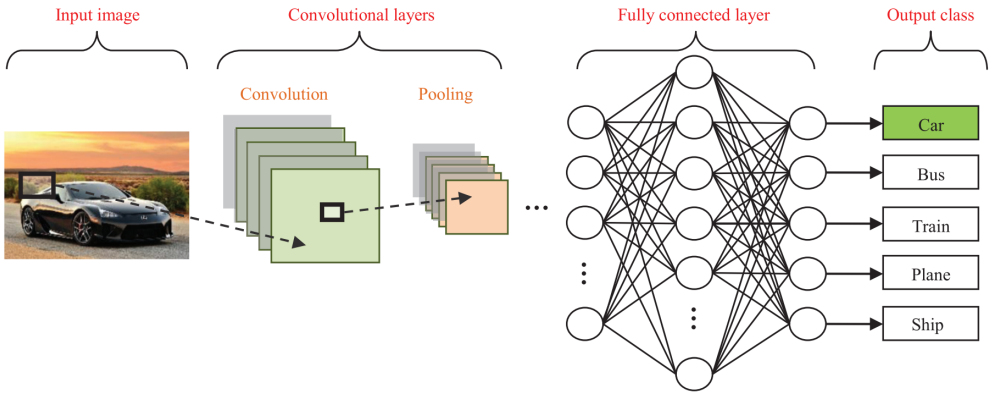

Figure 2: An example of CNN architecture. Image Source: W. Rawat and Z. Wang's comprehensive review on Deep CNNs.[6]

Figure 2 illustrates a basic CNN that may be used to classify images. The image is mapped onto convolutional layers. These layers are pooled together to determine key features in the image, which are then connected with key features typically present in different classification groups. At the end, the CNN determines what it believes is the correct classification. In Figure 2, the classification groups are based on the object present in the image. However, in the PokéGAN, there are only two classification categories: Real or Fake. The application of CNN in the PokéGAN can be seen in Figure 1, represented in the generator and discriminator's boxes.

2.3 Wasserstein Distance

Vanilla GANs typically use the Kullback-Leibler or Jensen-Shannon distances. By using these distance equations, the discriminator can measure the similarities between probability distributions and accurately determine mode clusters. However, using these two methods can lead to vanishing gradient or mode collapse.

To avoid these issues, GANs can use the Wasserstein distance, also known as the Earth Mover's (EM) distance. In a tangible example, if the probability distributions of interest were represented by piles of dirt, the EM distance is "the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution."[7] Our PokéGAN of interest uses the Wasserstein Distance. The algorithm of a basic Wasserstein GAN (WGAN) can be found below.

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: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

[5] W. Rawat and Z. Wang. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation* **29, 2352-2449**. 2017. doi:10.1162/NECO_a_00990.
[6] Ibid.
[7] L. Weng. From GAN to WGAN. 2017. https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html.

Figure 3: WGAN algorithm. Image Source: Wasserstein GAN by M. Arjovsky, S. Chintala, and L. Bottou.[8]

2.4 Data

The dataset that moxiegushi used consisted of profile images from Bulbapedia, a database of Pokémon.[9] Most, but not all, Pokémon were in moxiegushi's data set, and each Pokémon had only one image.

3 Proposed Changes and Results

3.1 Larger Dataset

We found another location on Bulbapedia with a larger sample of Pokémon, with different animation styles and angles for each Pokémon,[10] as opposed to moxiegushi's data with only one image per Pokémon. By using this kind of training data, which has a larger quantity and variety of images, we hypothesized that we could provide the generator and discriminator with a more diverse view of Pokémon and generate better images.

We used the same W/DCGAN that moxiegushi used, just with our new dataset. The results of the epochs side-by-side can be seen below in Figure 4.

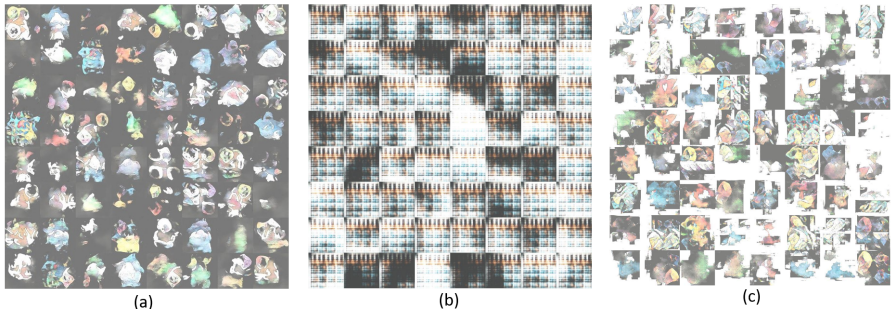

Figure 4: PokéGAN generated images. a - Image taken from moxiegushi's GitHub repository.[11] PokéGAN result with moxiegushi dataset after 5000 epochs. b - PokéGAN result with larger dataset after 50 epochs. c - PokéGAN result with larger dataset after 5000 epochs.

Despite inputting a larger dataset with greater diversity, our generated images after 5,000 epochs look less like Pokémon compared to the original data set used. After closer inspection, we found that the images in our data were not consistent. We had thought that we transformed the imaged to have completely black backgrounds, but instead, some images were unaffected, and others had strange patchy background coloring, like in Figure 5.

Figure 5: Image of Pokémon with incorrect background correction.

In the future, we would try running the W/DCGAN again with a corrected version of the larger dataset.

3.2 Capsule Network

Capsules are "locally invariant groups of neurons"[12] that encode features within an image to vectors. The magnitude of the vector represents the presence of a specific feature, and the direction represents its spatial orientation. In this way, capsule networks (CapsNets) can retain information on both the presence and spatial location of a feature, as opposed to a CNN which only relies on the former. We believed that a CapsNet would benefit our Pokémon generation. The results of moxiegushi's W/DCGAN seemed to have the coloration and patterns commonly present in Pokémon, but the shapes and features in relation to each other were muddled.

[8] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. 2017. https://arxiv.org/pdf/1701.07875.pdf.
[9] Bulbapedia. 2018. https://bulbapedia.bulbagarden.net/wiki/Main_Page.
[10] Bulbagarden Archives. Category: Ken Sugimori Pokémon artwork. 2018. https://archives.bulbagarden.net/wiki/Category:Ken_Sugimori_Pok%C3%A9mon_artwork.
[11] Moxiegushi. pokeGAN. 2017a. https://github.com/moxiegushi/pokeGAN.
[12] A. Jaiswal, W. AbdAlmageed, Y. Wu, and P. Natarajan. CapsuleGAN: Generative Adversarial Capsule Network. 2018. https://arxiv.org/abs/1802.06167.

When a discriminator uses a CapsNet, the minimax adversarial game aims to optimize Equation 2:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \tilde{} p_{data}(x)}[-L_M(D(x), T=1)] + \mathbb{E}_{z \tilde{} p_z(z)}[-L_M(D(G(z)), T=0]$$

Equation 2: The adversarial game used in a Capsule GAN (CapGAN).[13]

$$L_M = \sum_{k=1}^{K} T_k max(0, m^+ - ||v_k||)^2 + \lambda(1-T_k) * max(0, ||v_k|| - m^-)^2 \; T_k = \text{target labels } m^+ = 0.9 \; m^- = 0.1 \; \lambda = 0.5$$
$$v_k = \text{vector outputs of the final layer}$$

We used a three-layer CapsNet. In order to prevent oversaturation, we used Leaky ReLu for each convolution. Our first layer had 256 9x9 filters with a stride of 2. The next layer had 32 9x9x8 convolutions (i.e. 9x9 convolutions with 8 dimensions) with a stride of 2. Finally, we had a capsule layer with dynamic routing, which yielded 32 capsules with size 26x26 and 16 dimensions. A simplified version of this CapsNet design is in Figure 6.

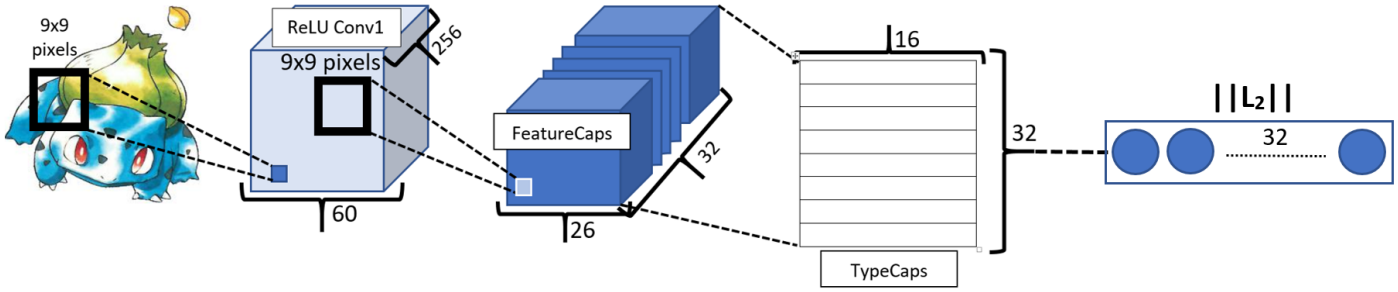

Figure 6: CapsNet architecture. Diagram adapted from paper by Sabour and Frosst.[14]

Although this should work in theory, in practice, it is unsustainable. This process was both time- and resource-intensive. Each epoch of the CapGAN took two hours to run, compared to 80 epochs every two hours for the W/DCGAN. We were also able to run the W/DCGAN on the Duke VM, but when running the CapGAN, we would get out-of-memory errors. As a result, we had to turn to Google Cloud and run the process on the largest GPU possible. It took over 700% CPU Power to run.

As a result, we unfortunately were unable to yield tangible results for out CapGAN architecture. Given more time and resources, we would experiment with the number of capsules per layer and with smaller batch numbers, among other potential factors to make it more efficient.

4 Conclusions

Although we were able to identify potential improvements for moxiegushi's PokéGAN, in the end, we were unable to find proof that we had improved the network. Given more time and resources, we would pursue further exploration and implementation of our proposed changes. The implications for better image-generating GANs reach beyond just Pokémon and animated images. These GANs could help those who are artistically-challenged bring their ideas to life–both in the animated and real world.

[13] Ibid.
[14] S. Sabour, N. Frosst, and G.E. Hinton. Dynamic Routing Between Capsules. NIPS. 2017.