

Face Pokémon-izer: Final Report

Team members: Hao-Hsiang Chuang, Jen-Feng Chang, Jheng-Hao Huang
hhchuang@stanford.edu, jenfengc@stanford.com, jhh@stanford.edu

Introduction

Nowadays, there are many messaging apps providing face filters that can transform human faces into something else, like cats, dogs, even Pokémon. However, those filters are adding predefined patterns on top of the face, which requires artists to build those patterns. As Pokémon lovers, we would like to have face filters for each Pokémon. To achieve that, we will build a **GAN**^[1] model that can Pokémon-ize human faces by feeding a certain Pokémon's images. The input to our algorithm will be a human image, the output will be a generated Pokémon-ized human face that discriminator thinks of as a Pokémon.

Related Works

There are a few related works we found during literature review. The first category is style transfer. As we are transferring real human images into Pokémon style, we started with various existing style transfer architectures, including **A Neural Algorithm of Artistic Style**^[2] and **Perceptual Losses for Real-Time Style Transfer and Super-Resolution**^[3]. We soon found out that it is not suitable for our problems because our "style" images are usually a single Pokémon with a dominant color, and it affects the generated picture a lot. We will discuss more in the experiment result section. Another category we have tried is face detection, **Face Swap**^[4] and **Anime Face Detection**^[5]. They are very good at detecting human faces or anime character faces, but not Pokémon faces. Finally, we exploited different GAN models, like **Avatar Artist Using GAN**^[6], **Baby Face Generator with CycleGAN**^[7] and **UGATIT**^[8], which we chose to use.

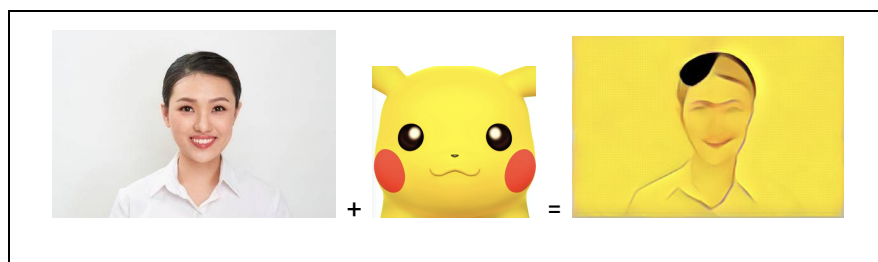
Dataset

We had 4 datasets:

1. **Labeled Pokémon Images**^[9]: This dataset contains 7000 labeled Pokémon images, including 150 different Pokémons with around 40 images for each.
2. **Pikachu's images**: In addition to the mix collection of various Pokémons, we specifically chose Pikachu and collected more of its images, applied data augmentation and ended up with 3042 Pikachu's images.
3. **Ditto's images**: We had about 275 Ditto's images after scraping online. Instead of applying data augmentation and increasing the dataset, we applied the augmentation during image preprocessing by randomly and minorly revising the input image (resizing, left/right flipping) to save instance storage.
4. **Female Images**^[10]: We used the data set provided by UGATIT official implementation^[11].

Methods

In milestone #1, we tried **A Neural Algorithm of Artistic Style**^[2] and **Perceptual Losses for Real-Time Style Transfer and Super-Resolution**^[3]. As shown below, the style transfer model mainly changed the dominant color only, as shown below.



Style transfer from human to Pokémon

As Style Transfer didn't give us a good outcome in milestone #1, we were seeking an approach which only modifies the face but not the whole image. **FaceSwap**^[4] was a reasonable approach to try. However, the model did not work well for our usage. Although the model can locate a human face precisely, it cannot locate the position of the Pokémon face. To solve the problem, we tried **Anime Face Detection**^[5], but the model cannot locate the Pokémon's

face either as it is specialized for anime human characters. Since there is no handy dataset for Pokémon’s face detection, this approach is halted.

In milestone #2, as recommended by our project TA (Advay Pal), we tried [UGATIT](#)^[7], which powers the web app [Selfie 2 Waifu](#)^[12]. It’s a GAN model that utilizes a new unsupervised image-to-image translation which incorporates a new attention module and a new learnable normalization function in an end-to-end manner. The attention module guides the model to focus on more important regions distinguishing between source and target domains based on the attention map obtained by the auxiliary classifier. The author used the model to transform between human images and anime images, here we trained this architecture by replacing anime images with Pokémon images and tuned several factors of the model based on our experiment results.

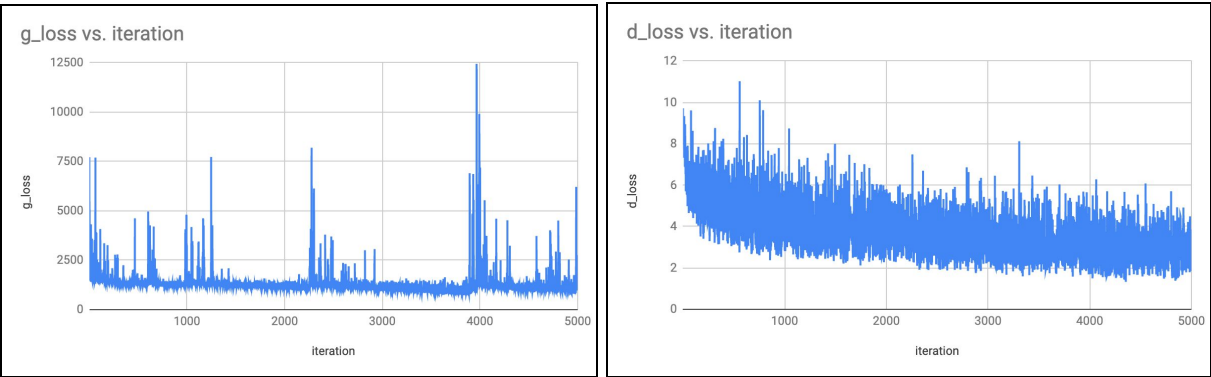
At the beginning, the training process was super slow and prone to storage outage failure. As a result, we had to reduce the number of iterations and the generated images were not very good. Also, from the loss analysis, we found out the discriminator’s loss (d_loss) converged way faster and noticeably smaller than generator’s loss (g_loss). It resulted in that the generator could improve itself because the discriminator was too good. It is a common challenge in GAN.

To address the issues mentioned above and further improve our model, we applied the following adjustments. First, to lower g_loss, we added Gaussian random noise manually every few iterations and mislabeled certain portions of input images. To speed up the training process, we tried to adjust our model with multi-GPUs implementation, following a few examples found on the internet ^{[13][14]}. As we successfully transferred our training process from a single CPU to single/multiple GPUs, we were able to train our model by a larger scale, in terms of both training time and size of data. As a result, we increased the volume of the dataset by either scraping for more images or applying image revision during image processing. Besides, we also applied transfer learning on top of the pre-trained model provided by the paper author with more level of neural networks to better suit our purposes. It also reduced the time we need to train a deeper model. We ended up with a faster-trained and better quality GAN model.


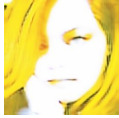
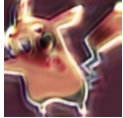
Experiments Results and Analyses

The following are the experiment results from the models we have trained, which are all based on [UGATIT](#).

For the first model, we used 196 Pikachu images with 5000(500 interactions per epoch) iterations. It took 2 days to train. Initially, the discriminator loss was around 8, which is already quite small compared to the generator loss. By the end of training, the discriminator loss was around 2. On the other hand, the generator loss started with 7k, and converged to around 1100 at the end. The generator loss cannot be lower and the model only learns yellow color.

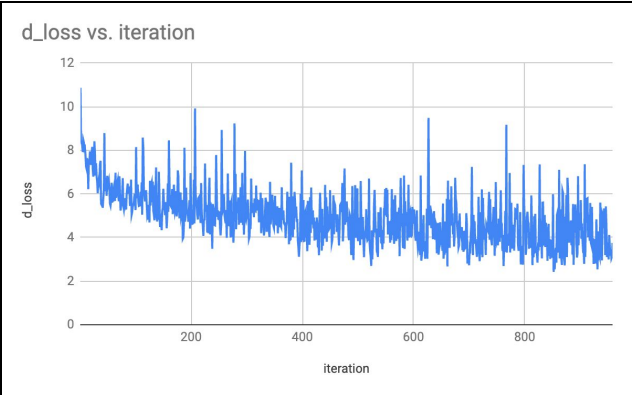
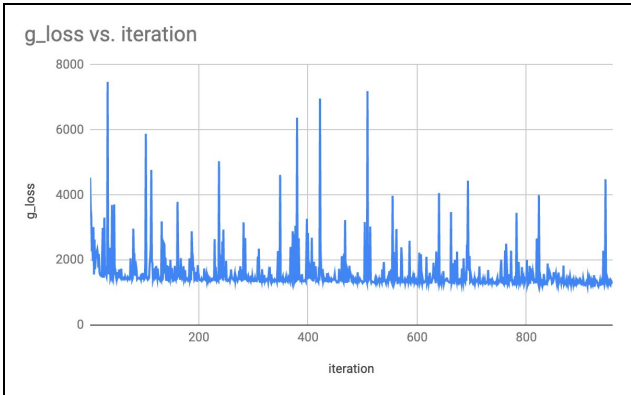



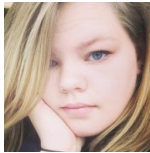




	Human to Pokémon [1]	Human to Pokémon [2]	Pokémon to Human
Input			

Output			
--------	---	--	---

In general, if the generator or the discriminator learns way faster than another, it is hard for another to improve. In our case, the discriminator is learning quicker as its loss dropped faster. As a result, in our second experiment, we decided to lower the learning rate for the discriminator by 10 times. However, it didn't seem to work.

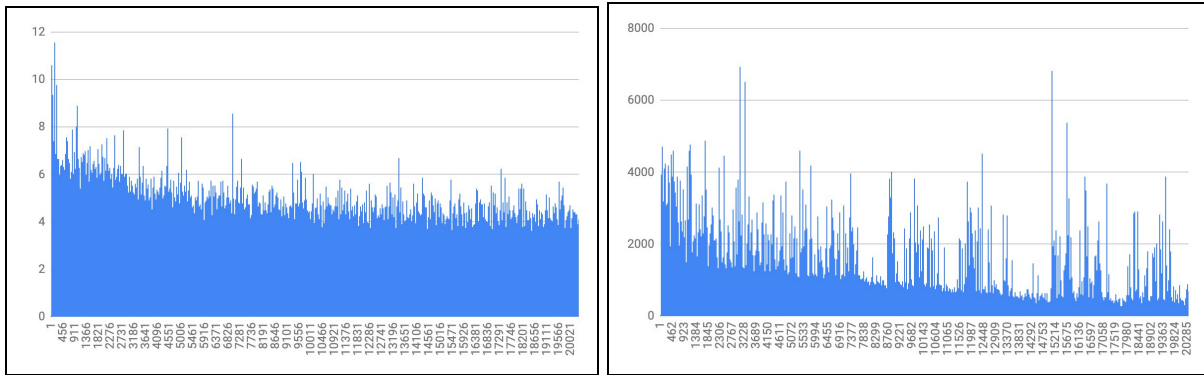
In our last experiment, we increased the dataset size from 196 to 3042 by data augmentation and ran more interactions per epoch (1000). Beside that, we also introduced noise to discriminator every 100 iterations, as suggested by **Soumith Chintala et al**^[15]. However, the program accidentally shut down at epoch 0; iteration 957. The generator loss cannot be lower since we only run 957 iterations. The generator loss is around 1300. More iterations are needed.


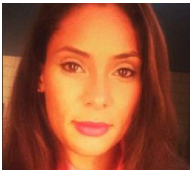
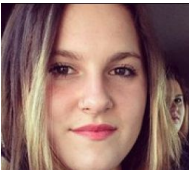
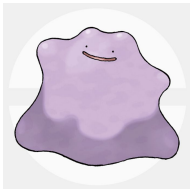






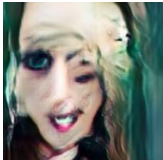



	Human to Pokémon [1]	Human to Pokémon [2]	Pokémon to Human
Input			
Output			

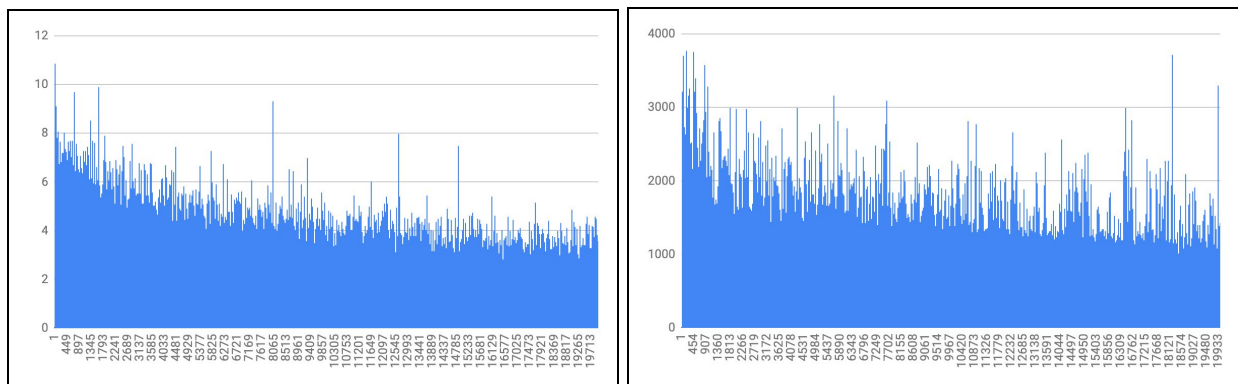
At this point, we found out that we have been using CPU to train the model, so we reset up the AWS environment and tried to use multi-GPUs. Eventually, we got the speed increased 5x. With that, we started experimenting with different datasets and modified architecture.





Because the structure of the Pikachu is complicated, we first tried using simpler Pokémons, like Ditto. We manually crawled 274 Ditto images with data augmentation. We also increased the resblock layers from 4 to 8 and introduced noise randomly every 500 iterations. After 20000 (10000 iterations per epoch) iterations, the discriminator loss is around 1 to 4. The average generator loss is around 300. With higher iterations and simpler Pokémon images, the generator loss was the best in several rounds of experiment. The Output images were also with face shapes and colors modified. The model can also transform Ditto back to human images. However, some distortion came from irrelative things in the Ditto images.



	Human to Pokémon [1]	Human to Pokémon [2]	Human to Pokémon [3]	Pokémon to Human[1]	Pokémon to Human[2]	Pokémon to Human[3]
In						
Out						

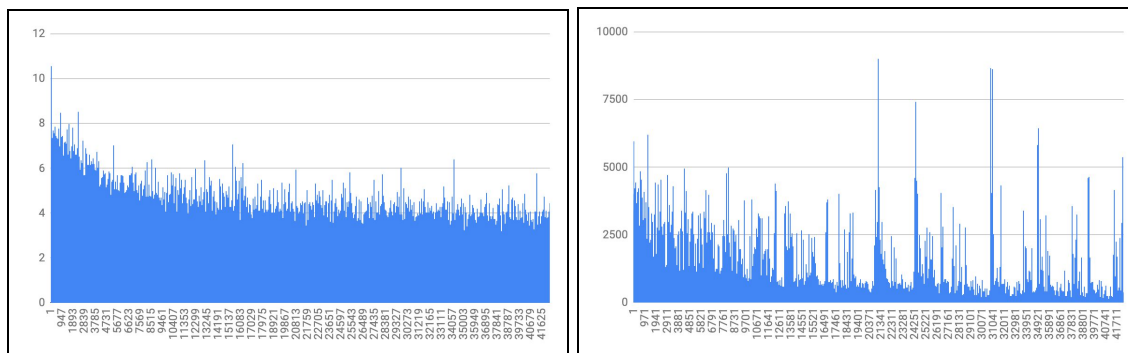
Next, we tried mixing different Pokémon. The discriminator and generator losses were similar to the Ditto dataset. However, the output images were hard to tell which Pokémon's features were used, and we cannot transform Pikachu to a human face either.








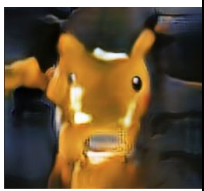

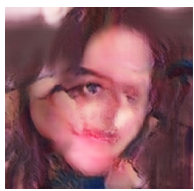
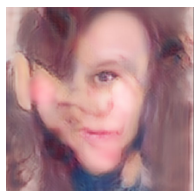
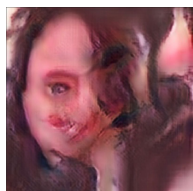


	Human to Pokémon [1]	Human to Pokémon [2]	Human to Pokémon [3]	Pokémon to Human[1]	Pokémon to Human[2]	Pokémon to Human[3]
In						



Finally, back to our original target Pokémon, Pikachu. We manually crawl 445 Pikachu images. Surprisingly, the average generator loss is around 200, which is lower than the Ditto dataset. Compared to the 2nd round of experiment, we double the size of Pikachu images and train 20 times more iterations, and the generator loss improves from 1300 to 300. The model can partially change the human face to a whole Pikachu, not the Pikachu face. We can transform a whole Pikachu to a human face.



	Human to Pokémon [1]	Human to Pokémon [2]	Human to Pokémon [3]	Pokémon to Human[1]	Pokémon to Human[2]	Pokémon to Human[3]
In						
Out						

Conclusion and Future Work

From above, we can conclude that :

- The longer the training process and the larger the input data we gave, the better the quality of the GAN
- The simpler the feature of the input images, the better quality the generated images are.
- The GAN we trained did not focus on just facial transformation, rather it's focusing on the style of the whole image, including the background.

From the conclusion, there are improvements that can be made in the future :

- If there is time constraint, always try to use single/multiple GPU computation resources.
- A better face detection model is needed, especially the one that is not limited to just human faces.
- With better face detection technique, we can then either apply face swap or limit the loss calculation to just the facial areas.
- Besides GAN, maybe it's worthwhile to try different models or a deeper neural network.

Contribution

All team members equally contributed to the project. All of us have collected data, done literature review and written code. Specifically, Hao-Hsiang Chuang did extra literature review, implemented code on top of UGATIT^[9]. Jen-Feng Chang set up AWS environment, developed analysis tools and collected/augmented extra data. Jheng-Hao Huang also implemented code with Hao-Hsiang, debugged and fixed issues, and finalized reports.

The team is also thankful to Advay Pal, our project TA who consistently provided us feedback, and Wei-Chiu Ma from MIT, who shared his insights on the project progress.

Reference

- [1] [Generative Adversarial Networks](#)
- [2] [A Neural Algorithm of Artistic Style](#)
- [3] [Perceptual Losses for Real-Time Style Transfer and Super-Resolution](#)
- [4] [Face Swap](#)
- [5] [Anime Face Detection](#)
- [6] [Avatar Artist Using GAN](#)
- [7] [Baby Face Generator with CycleGAN](#)
- [8] [Unsupervised Generative Attentional Networks With Adaptive Layer Instance Normalization For Image-To-Image Translation \(UGATIT\)](#)
- [9] [Pokémon dataset](#)
- [10] [Female Images](#)
- [11] [UGATIT official implementation](#)
- [12] [Selfie 2 Waifu](#)
- [13] [TensorFlow with multiple GPUs](#)
- [14] [Multi-GPU Training Example](#)
- [15] [How to Train a GAN? Tips and tricks to make GANs work](#)