PokéGAN: Generating Original Pokémon Sprites Using a Generative Adversarial Network

CSC 487: Deep Learning

Dr. Paul Anderson

December 12, 2025

Content created, written, and displayed by:

Lucas Summers (lsumme01@calpoly.edu)
Braeden Alonge (balonge@calpoly.edu)

1. Abstract

Generative Adversarial Networks (GANs), and its various flavors, have demonstrated remarkable capabilities in synthesizing realistic images across various domains. In this project, we attempt to use a Deep Convolutional GAN (DCGAN) trained from scratch to generate original Pokémon sprites. Our final model builds upon our basic DCGAN model from Stage 2, implementing various improvements to architecture and training including self-attention mechanisms (SAGAN-style), data augmentation, spectral normalization for discriminator stability, label smoothing, early stopping, and asymmetric learning rates. We compare our final model to our baseline model, evaluating performance first using Fréchet Inception Distance (FID), then the Inception Score (IS) and Diversity Score metrics. Our experiments demonstrate the effectiveness of GAN stabilization techniques for training on limited datasets, while also highlighting the downsides and difficulty of training GANs for image generation tasks.

Project GitHub: https://github.com/BraedenAlonge/CSC487-Project

2. Introduction

2.1 Motivation

Pokémon is an iconic franchise spanning video games, movies, TV shows, and trading cards. Since its creations, it's featured over 1,000 unique creatures with distinct artistic choices and visual attributes, such as vibrant colors, clean outlines, and imaginative body structures. Generating new, original Pokémon that could be mistaken for real ones presents an interesting challenge for generative models, which must not only capture qualities of the specific types of Pokémon, but the specific artistic style and design that makes a Pokémon recognizable.

GANs are the perfect model for this challenge. Through their use of two competing models, where a Generator learns to produce an image that can fool a Discriminator, the Generator can hopefully learn all the visual features that make Pokémon unique. However, because this requires having to train two models at the same time and in harmony, GANs are extremely hard to train, suffering from many issues like training instability, vanishing gradients, and mode collapse. These challenges are particularly pronounced when working with limited data, like with Pokémon sprites.

2.2 Problem

Our goal is to successfully train a GAN from scratch that can generate 64c64 RGB images of novel Pokémon sprites that would be visually comparable to the authentic sprites. Success is measured qualitatively by visually comparing generated images, as well as quantitatively through generative model metrics (FID, IS, Diversity Score).

2.3 Baseline Recap

In Stage 2, we implemented a baseline model, following the basic DCGAN architecture:

- **Generator:** 100-dimensional noise vector → 4×4×512 → progressive upsampling to 64×64×3 image via transposed convolutions with BatchNorm and ReLU
- **Discriminator:** 64×64×3 image → progressive downsampling to 4×4×512 via strided convolutions with BatchNorm and LeakyReLU → single probability score (real or fake)
- **Training:** BCE loss, Adam optimizer (lr = 0.0002 for both), batch size 64, 200 epochs

The best model, indicated by FID score, was only at about epoch 15, which achieved:

- **FID:** 218.59 (lower is better)
- **Inception Score:** 210 +- 0.06 (higher is better)
- **Diversity Score:** 52.84

Generated images started to get the general form and color of real Pokémon, but they still only looked like blob-like shapes with little to no fine details. FID also increased after only about 15 epochs, suggesting overfitting or training instability as a whole. The discriminator also showed signs of underfitting with accuracy only slightly above chance. Overall, while our findings were promising, we were motivated in Stage 3 to implement various improvements focused on training stability and enhancements to the architecture.

3. Related Work (optional but encouraged; at least two references)

3.1 Deep Convolutional GANs (DCGAN)

Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. International Conference on Learning Representations (ICLR). arXiv:1511.06434

3.2 Self-Attention GAN (SAGAN)

Zhang, H., Goodfellow, I., Metaxas, D., & Odena, A. (2019). Self-Attention Generative Adversarial Networks. Proceedings of the 36th International Conference on Machine Learning (ICML), 7354-7363. arXiv:1805.08318

3.3 Spectral Normalization

Miyato, T., Kataoka, T., Koyama, M., & Yoshida, Y. (2018). Spectral Normalization for Generative Adversarial Networks. International Conference on Learning Representations (ICLR). arXiv:1802.05957

4. Methodology

4.1 Finalized Architecture

Our model maintains the core DCGAN architecture from Stage 2, adding a few modular improvements that can be enabled before training.

Generator Architecture:

- **Input:** 100-dim Gaussian noise vector z
- **Linear:** 100 → 8192 (4x4x512)
    - BatchNorm1d → ReLU activation → Dropout if enabled
    - Reshape to (batch, 512, 4, 4)
- **TransposeConv2d:** 512 → 256 (4x4 → 8x8)
    - BatchNorm2d → Dropout2d if enabled → ReLU activation
- **TransposeConv2d:** 256 → 128 (8x8 → 16x16)
    - BatchNorm2d → Dropout2d if enabled → ReLU activation
    - [Self-Attention if attention_layer=16]
- **TransposeConv2d:** 128 → 64 (16×16 → 32×32)
    - BatchNorm2d → Dropout2d if enabled → ReLU activation
    - Self-Attention if attention_layer=32
- **TransposeConv2d:** 64 → 3 (32×32 → 64×64)
    - Tanh activation
- **Output:** 64×64×3 RGB image with pixels normalized to [-1,1]

Discriminator Architecture:

- **Input:** Input: 64×64×3 RGB image
- **Conv2d:** 3 → 64 (64×64 → 32×32)
    - SpectralNorm if enabled
    - LeakyReLU(0.2) → Dropout2d if enabled
    - Self-Attention if attention_layer=32
- **Conv2d:** 64 → 128 (32×32 → 16×16)
    - SpectralNorm OR BatchNorm2d
    - LeakyReLU(0.2) → Dropout2d if enabled
    - Self-Attention if attention_layer=16
- **Conv2d:** 128 → 256 (16×16 → 8×8)
    - SpectralNorm OR BatchNorm2d
    - LeakyReLU(0.2) → Dropout2d if enabled
- **Conv2d:** 256 → 512 (8×8 → 4×4)
    - SpectralNorm OR BatchNorm2d
    - LeakyReLU(0.2) → Dropout2d if enabled
- **Flatten** → **Linear** (8192 dim → 1 dim)
    - Sigmoid activation
- **Output:** probability between 0 and 1

Self-Attention Module:

Our self-attention module follows the implementation in the SAGAN design. For an input feature map x of shape (B, C, H, W), we calculate:

- **Query Vector:** $f(x) = W_q x \rightarrow (B,\ C/8,\ H \times W)$
- **Key Vector:** $g(x) = W_k x \rightarrow (B,\ C/8,\ H \times W)$
- **Value Vector:** $h(x) = W_v x \rightarrow (B,\ C,\ H \times W)$
- **Attention Matrix:** $softmax(KQ^T) \rightarrow (B,\ H \times W,\ H \times W)$
- **Output Matrix:** $\gamma \times (VA^T) + x$
    - ***Note:*** γ is a learnable parameter which starts at zero, allowing the model to initially rely on local convolutional features and gradually incorporate attention as training progresses. This stabilizes early training when attention weights are basically random

The idea is that we allow every spatial position to directly attend to every other position in a single operation. This helps the earlier layers, which still make decisions based on limited local context. It will hopefully help generate more coherent sprites, as distant features like eyes and limbs must coordinate form and color. In our case, we chose to place attention at 32x32 spatial resolution in both networks, balancing memory requirements and having enough spatial detail. The placement allows attention to coordinate features like placement and color regions before being refined, hopefully improving the look of our sprites.

Spectral Normalization:

Spectral normalization addresses the key challenge of preventing the discriminator from becoming too powerful too quickly, leading to vanishing gradients for the generator. It does this by normalizing the weights as $\bar{W} = \frac{W}{\sigma(W)}$, where $\sigma(W)$ is the spectral norm of the weight matrix, or its largest singular value. We apply spectral normalization to all discriminator layers, ensuring no layer amplifies signals excessively, keeping the discriminator's predictions and gradients in a stable range throughout training. Importantly, we remove BatchNorm when using spectral normalization, as combining both can destabilize training. Overall, we chose it because it requires no changes in the loss function, integrates well with self-attention, and adds a small computational overhead.

4.2 Training Improvements

We implemented many techniques to improve the training process. First we implemented label smoothing, where hard labels are replaced by soft labels by adding/subtracting a fixed amount, preventing the discriminator from becoming overconfident. We chose one-sided smoothing, only smoothing the real labels, as it prevents the generator from receiving incorrect gradient signals. Another label technique we employ is label flipping, where with a probability of 0.05, we randomly flip labels during discriminator training, adding noise that prevents overfitting. We also used the Two-Timescale Update Rule (TTUR), or using asymmetric learning rates for the generator and discriminator. Our 4:1 ratio (lr_g = 0.0001, lr_d = 0.0004) allows the discriminator to learn faster, providing more informative gradients to the generator and helping improve convergence to a local minimum. Another technique was gradient clipping, which we optionally add to prevent exploding gradients. Last was the addition of early stopping, based on our observations in the baseline model, where FID increased after 15 epochs. We give the model 20 epochs to improve by 1.0 FID before stopping training. The best model, in terms of FID, is always saved.

4.3 Data Augmentation

We chose to do data augmentation to address the concern that the 26,000+ images our dataset provides is not enough to train our GAN, or would cause overfitting because of a lack of diverse examples. By introducing various transformations to augment the original data, we can artificially expand the dataset size and force the discriminator to learn transformation-invariant features from the images. Overall, we implemented Horizontal flips (0.5 prob), Vertical flips (0.1 prob), and Random Rotations (5 deg). We intentionally avoided more aggressive augmentations like Color Jitter or Random Cropping in order to keep the original artistic style of Pokémon intact in our generated images.

5. Experimental Setup

5.1 Dataset

We used the "1000 Pokémon Dataset" from Kaggle, which contains 26,000+ images of 1,000 different Pokémon species. We split the data into training (90%), validation (5%), and test (5%) sets, ensuring we left as much data as possible for our GAN to train on, while also having enough data for our evaluation metrics. Furthermore, we also performed basic preprocessing, including resizing all the images from 128x128 to 64x64 and converting from RGBA (4 channels) to RGB (3 channels) for quicker training, as well as normalizing all pixel values to the [-1, 1] range to match the Generator's Tanh output.

5.2 Experimental Configurations

We define two primary configurations for comparison. Our goal is to see if advanced GAN techniques actually offer improvement to our output over a vanilla DCGAN setup in terms of our evaluation metrics and the quality of the images generated.

Baseline Configuration:

Parameter	**Value**
Batch Size	64
Epochs	200
Learning Rates	0.0002/0.0002

Beta1, Beta2	0.5, 0.999
Label Smoothing	0.0
Spectral Norm	Disabled
Self-Attention	Disabled
Data Augmentation	Disabled
Early Stopping	Disabled

Final Configuration:

Parameter	**Value**
Batch Size	128
Epochs	200
Learning Rates	0.0001/0.0004
Beta1, Beta2	0.5, 0.999
Label Smoothing	0.2 (two-sided)
Spectral Norm	Enabled
Self-Attention	Enabled (32x32)
Data Augmentation	Enabled
Early Stopping	Enabled (patience=20 epochs)

5.3 Evaluation Metrics

Because GANs are fundamentally an unstructured architecture and thus have no ground truth to compare against, we used multiple different metrics to get a good sense of model performance. The main three metrics we used were:

1. **Fréchet Inception Distance (FID):** Measures the statistical distance between real and generated image distributions in Inception V3 feature space. Lower is better.
2. **Inception Score (IS):** Measures image quality and diversity based on Inception V3

class predictors. Higher is better.

3. **Diversity Score:** Average pairwise L2 distance between generated samples in pixel space, indicating how varied the outputs are (less mode collapse). Higher is better.
4. **Discriminator Accuracy:** Classification accuracy of the discriminator on real and generated images. Useful for detecting overfitting in the discriminator.

We decided to use FID as our primary metric when evaluating and testing our model, as it's the accepted metric for image generation tasks. While the other two metrics only compare the pixel statistics of generated images to each other, FID compares generated to real images based on high-level features, making it better at indicating if images are actually improving.

5.4 Hardware and Training

We used Google Colab Pro to train our model, opting for the NVIDIA A100 over the T4 due to its 40GB of VRAM and superior speed. Training our final model with a batch size of 128 and 119 epochs (due to early stopping) took around 2-3 hours. For reproducibility, we set our random seeds (for Python, NumPy, PyTorch, and CUDA) all to 42 and also allowed for optionally setting all PyTorch operations to be deterministic (which we disabled for better speed). In case the runtime disconnected, we also implemented check pointing, saving the model whenever FID score improved as well as at the end of training. Sample generated images were also saved whenever FID score improved.

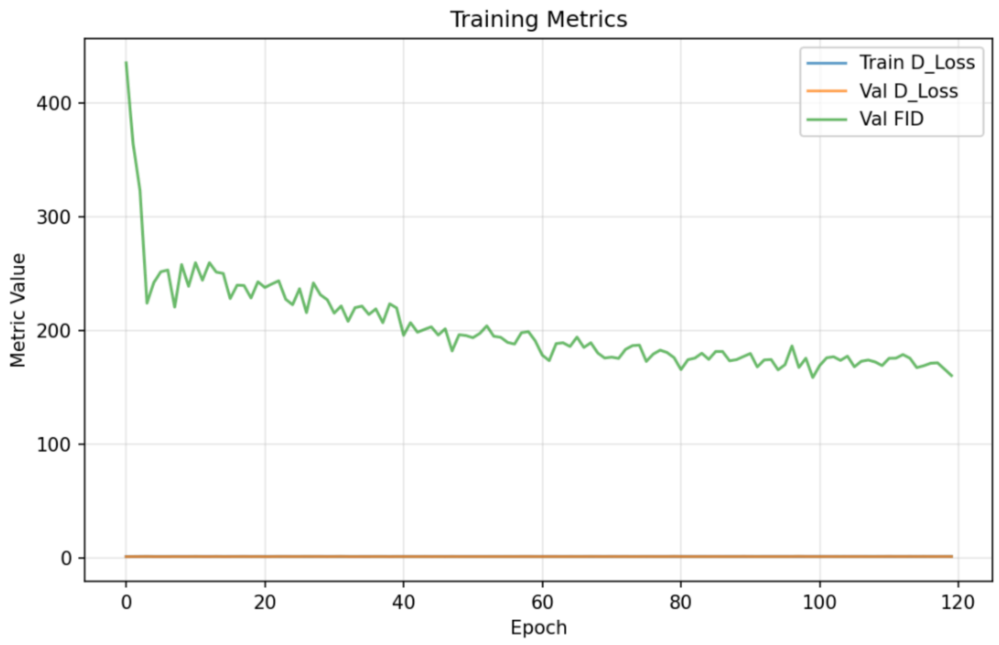
6. Results and Analysis

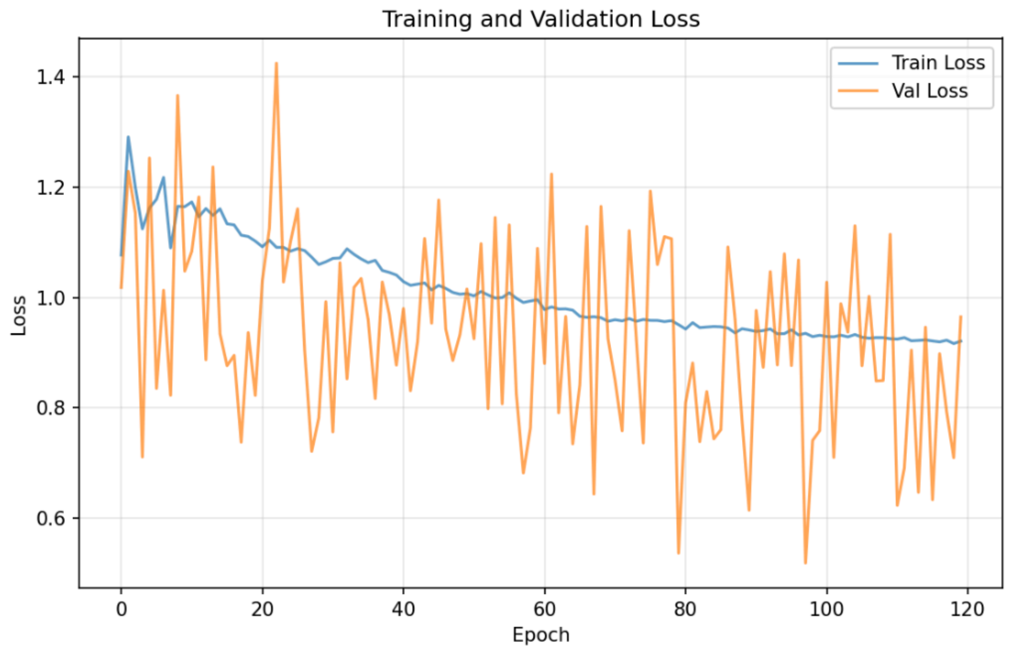
6.1 Quantitative Results

We evaluated both the baseline and final model on the held-out test using 1,000 real and 1,000 generated samples.

Metric	**Baseline Model**	**Final Model**	**Change**
FID	218.59	79.92	-138.67
Inception Score	2.10±0.06	2.95 ± 0.12	+0.85
Diversity Score	52.84	59.81	+6.97
Discriminator Acc.	22.7%	62.95%	+40.25%
Epoch	200	99 (early stopping)	-101

The final model, which trained in only half the epochs as the baseline model, clearly showed large gains across all metrics, particularly with a substantial increase in FID score. Notably, the discriminator accuracy jumped from 40.25%, indicating a much healthier adversarial balance between the networks, the main key to successful GAN training.

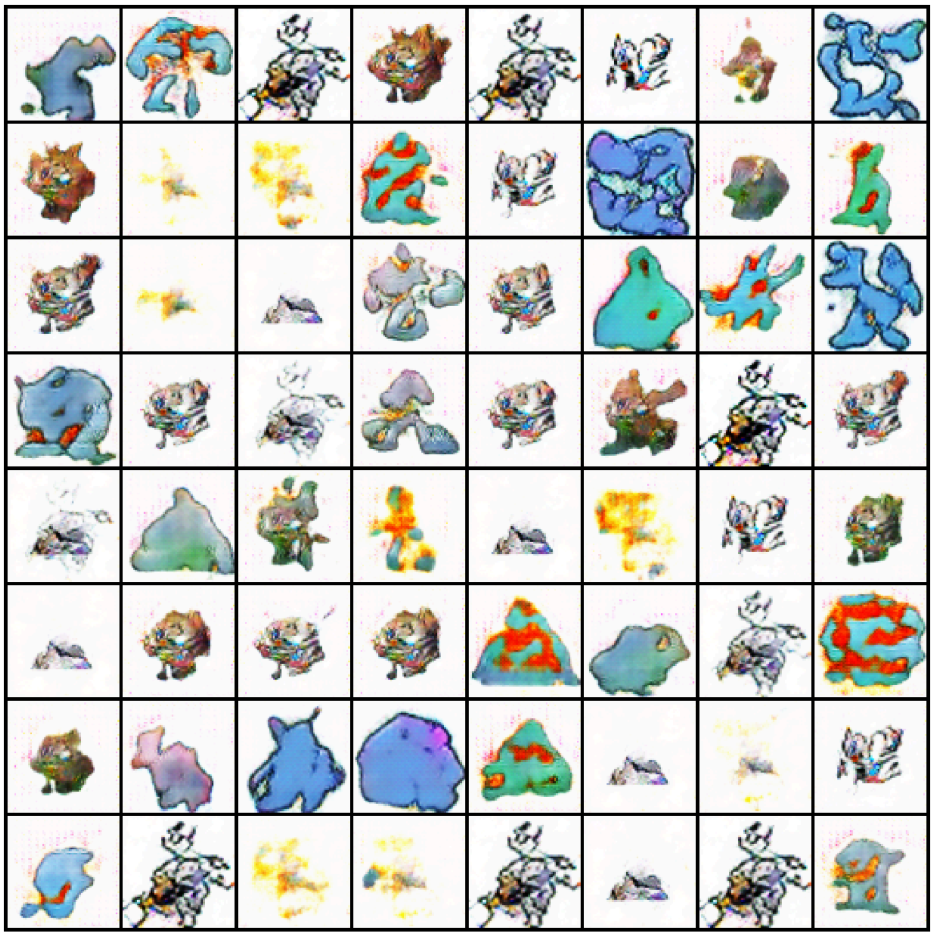




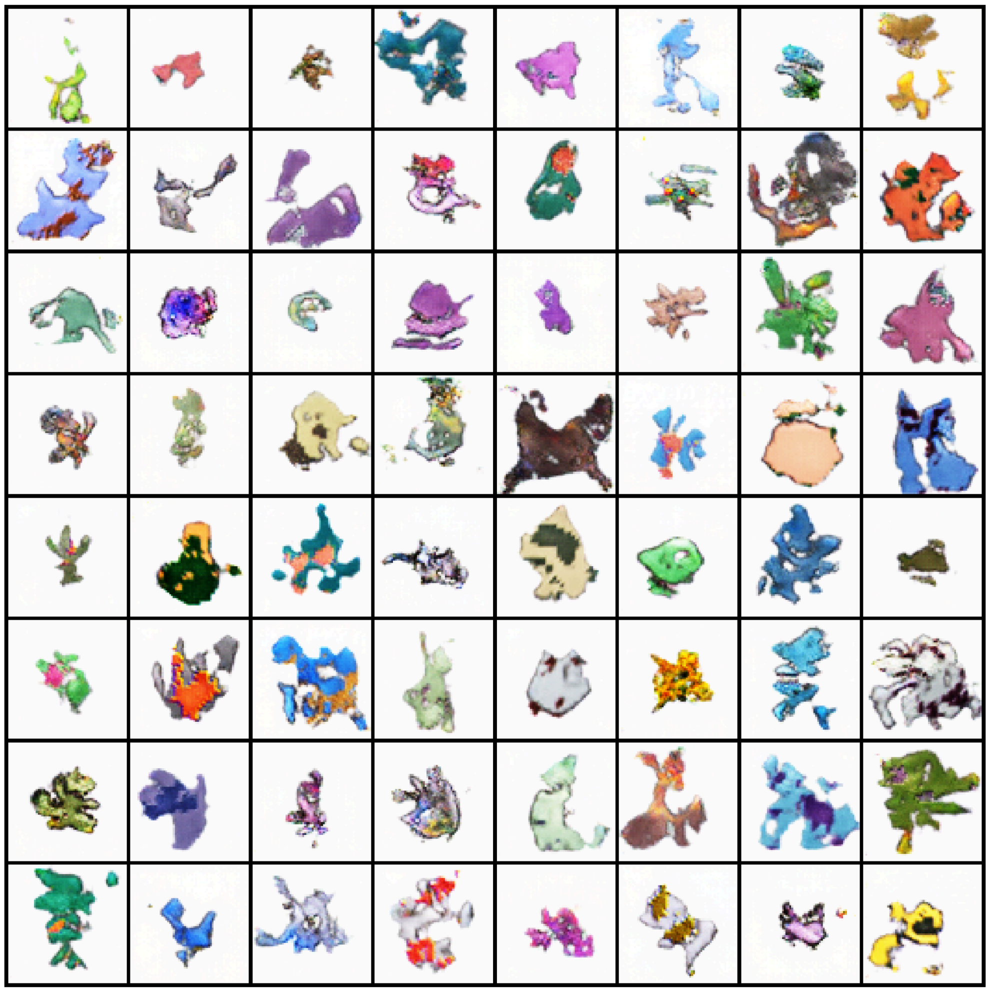
Noisy validation loss is common for GAN models, and so it is not a concern. However, the FID score curve shows a more meaningful story of the training process. In the first few epochs, we see rapid improvement, with FID score dropping to half of what it started as, as the model quickly learns basic structure and color distributions. After that, we see a steady improvement until around epoch 60, as the generator readiness shapes and features. After that, we see FID score start to settle until early stopping is triggered at epoch 119. This training curve is much more what we would expect from a successful mode, in contrast to the baseline model where FID started to increase after epoch 15. This demonstrates that the various stabilization techniques implemented in the final model actually worked, enabling sustained learning.

6.2 Qualitative Results

Here's's a recap of the 64 sample images outputted from our baseline model:



Now, here are 64 sample images outputted by our final model:



Clearly, there are some improvements in image quality, as already indicated by the improvement to our metrics. While they still are blob-like shapes, our final model is much more coherent, with sprites showing more defined body structures with recognizable silhouettes. There's also much better color consistency for the different types (earth, fire, water, etc.), with less color bleeding in between regions and the background, cleaner edges, and better shading. Some sprites also show different emergent features, like eyes, limbs, and tails. Despite this, our final model's sprites still lack the crisp lines and flat-shaded style of Pokémon sprites. Despite the introduction of self-attention, bilateral symmetry is not consistently achieved in the sprites and "body parts" don't always connect or coordinate logically, leading to floating and merged sections.

7. Discussion

Our results demonstrate that applying modern GAN stabilization techniques (such as spectral norm, self-attention, label smoothing, and TTUR) have greatly improved the quality and consistency of our generated Pokémon sprites. Compared to our baseline DCGAN, the final model was able to achieve a 138 point improvement in FID, along with visible gains in both the Inception and Diversity Score. These metrics are strong indicators that the new model has a deeper understanding of underlying Pokémon representation through structure, color, and shape. Despite these improvements, however, the sprites still fall short of the crisp outlines, symmetry, and stylized shading that is apparent in real Pokémon artwork.

A key challenge was the difficulty of training GANs on small, highly diverse datasets. With over 1,000 Pokémon species and around 25 images for each, the generator faced a broader distribution than typical GAN datasets of comparable size. This was likely a contributing factor to persistent issues like blurred shapes, incomplete body structures, and inconsistent anatomy, especially since many of the images were not front-facing and showed different parts of the Pokémon. Self-attention somewhat helped by allowing long-range dependencies for influence of spatial coherence, but the model still struggled to capture geometry and discrete edges.

The discriminator behavior was also insightful, in which the baseline discriminator underfit severely, while the final model's discriminator achieved ~63% accuracy, which is high enough to provide useful gradients for the generator but low enough to maintain adversarial balance. This balance appeared to be central to the final model's success. We also observed that improvements in stability of the discriminator (through spectral norm and TTUR) tended to correlate directly with improvements in generator quality, reinforcing findings from prior GAN research.

Overall, our findings show that stabilizing GAN training is both necessary and impactful for this task, but that the architectural improvements alone cannot fully compensate for the challenges of pixel-art generation and limited, highly diverse training data. More specialized architectures may be required to achieve results that are comparable to official Pokémon sprites.

8. Ethics and Responsible AI

Although our model generates fictional Pokémon-style creatures, a few ethical considerations do arise. First of all, the dataset consists of copyrighted artwork. While our use *does* fall under academic, non-commercial research and all training is done on publicly available sprite compilations, the ownership of the underlying content does belong to Nintendo. Therefore, these generated sprites must not be represented or used as official Pokémon. We emphasize that this project aims to study GAN behavior, not to reproduce/distribute artwork. Second, we value transparency in that users viewing the generated sprites should be aware that they are synthetic outputs of a machine learning model, not assets produced by human artists. Clear labeling prevents confusion, false attribution, and unintentional misuse. Finally, we recognize the broader implications of generative AI in creative industries. Although this project is purely academic, similar systems applied to human-created artwork require careful ethical consideration and responsibility with its deployment.

9. Conclusion and Future Work

This project demonstrates that advanced GAN training techniques can dramatically improve generative performance on small, stylistically rich datasets such as Pokémon sprites. Our final model significantly outperformed the baseline DCGAN across all metrics, more specifically in FID, where we were able to attain a reduction from 218.59 to 79.92. Qualitative results confirm that there is clearer structure, more coherent shapes, improved color consistency, and a stronger recognizability for features like limbs and eyes.

Despite these gains, these generated sprites do still exhibit many limitations: fuzzy edges, inconsistent anatomy, asymmetric shapes, and a lack of the distinctive crisp pixel art style characteristic of official Pokémon designs. For future work, these problems point towards several promising directions. First, modern diffusion architectures often outperform GANs on small datasets, so we would consider using such a model. Second, we may consider training the generator on Pokémon type, color palette, or cluster ID to help reduce variance and increase control on the Pokémon that we generate. Third, expanding the dataset (probably manually curated sprite sets) or clustering similar Pokémon before training may reduce complexity. Finally, our current dataset has many different angles of the Pokémon, including images of how they look from behind. We believe that these images are polluting the dataset such that the generator is missing eyes in the Pokémon most of the time. Separating or removing these images may prove to be beneficial for eye structure in our generated Pokémon. All in all, while our stabilized GAN approach made meaningful steps towards generating Pokémon-style sprites passable as "real," there are significant opportunities that remain to improve sharpness, symmetry, and structure of our images. Our strong foundation that we have built through the trials we have faced gives way for exploration of better generative models tailored to improved pixel-art generation.