

Lucas Summers (lsomme01@calpoly.edu)

Braeden Alonge (balonge@calpoly.edu)

CSC 487 Final Project Stage 2

i. Project Summary

PokéGan is a generative AI model aimed at creating new Pokémon sprite images using a Generative Adversarial Network (GAN) trained from scratch in PyTorch. Our model will learn the various visual features of the preexisting Pokémon sprites it trains on, aiming to produce synthetic artwork indistinguishable to the authentic ones.

ii. Model Description

Our Baseline model implements a DCGAN-style architecture, with separately trained discriminator and generator networks, each with about 2.5-3 million parameters.

Generator Architecture: The generator transforms a 100-dimensional Gaussian noise vector into a 64x64x3 RGB image through convolutional layers. It begins with a fully connected layer that projects the noise vector to a 1D tensor of size $4 \times 4 \times 512$, followed by Batch Normalization and ReLU activation. This 1D tensor is then reshaped into a $4 \times 4 \times 512$ feature map (with shape [batch, 512, 4, 4]), and progressively upsampled through our convolutional blocks:

- Layer 1: $4 \times 4 \times 512 \rightarrow 8 \times 8 \times 256$ (kernel=4, stride=2, padding=1)
- Layer 2: $8 \times 8 \times 256 \rightarrow 16 \times 16 \times 128$ (kernel=4, stride=2, padding=1)
- Layer 3: $16 \times 16 \times 128 \rightarrow 32 \times 32 \times 64$ (kernel=4, stride=2, padding=1)
- Layer 4: $32 \times 32 \times 64 \rightarrow 64 \times 64 \times 3$ (kernel=4, stride=2, padding=1)

Each transposed layer is followed by Batch Normalization and ReLU activation to stabilize training and introduce nonlinearity. The final output layer uses Tanh to produce pixel values in the range $[-1, 1]$, matching our normalized input data.

Discriminator Architecture: The discriminator is a binary classifier that takes in a 64x64x3 RGB image as input and outputs a single probability scorer indicating if the image is produced by the generator or from the training data. It uses strided convolutions instead of pooling for downsampling:

- Layer 1: $64 \times 64 \times 3 \rightarrow 32 \times 32 \times 64$ (kernel=4, stride=2, padding=1)
- Layer 2: $32 \times 32 \times 64 \rightarrow 16 \times 16 \times 128$ (kernel=4, stride=2, padding=1)
- Layer 3: $16 \times 16 \times 128 \rightarrow 8 \times 8 \times 256$ (kernel=4, stride=2, padding=1)
- Layer 4: $8 \times 8 \times 256 \rightarrow 4 \times 4 \times 512$ (kernel=4, stride=2, padding=1)

Each convolutional block uses LeakyReLU activation with slope 0.2 to prevent dying neurons, as well as Batch Normalization for stable training. The final $4 \times 4 \times 512$ feature map is flattened

and passed through a fully connected output layer with Sigmoid activation to produce a probability between 0 and 1.

Training Objective: We used the standard GAN minimax objective with Binary Cross-Entropy loss (BCE). The discriminator is trained to maximize its ability to discriminate between real and fake images, while the generator is trained to maximize its ability to fool the discriminator. Both networks use the Adam optimizer with configurable beta values and learning rates. We implement alternating updates, training the discriminator and generator in sequence each iteration.

iii. Experimental Setup

Dataset and Preprocessing: We use the “1000 Pokémon Dataset” sourced from Kaggle, which contains 26,000+ images of 1,000 Pokémon. The dataset is already split into train (80%), validation (10%), and test (10%) sets with separate directories. We process training images as follows:

- Resize from 128x128 to 64x64
- Convert RGBA to RGB (removing transparency)
- Normalized to $[-1, 1]$ range to match generator’s Tanh output

Hyperparameters: Our baseline model uses the following configuration for training:

- Batch size: 128
- Epochs: 200
- nz (size of input noise vector): 100
- Learning rate: 0.0002 (both networks)
- Beta1/Beta2 (Adam): 0.5, 0.999 (hardcoded)
- Label smoothing: 0 (no smoothing)

Training Procedure: We use the standard GAN alternating optimization technique. For each training batch, we first update the discriminator by computing losses on both real images (labeled as 1) and generated fake images (labeled as 0, with gradients detached from the generator). We then update the generator by generating new images and training it to fool the discriminator (i.e., maximize the discriminator's output probability for fake images). After each epoch, we validate the model on the validation set, compute the FID score, and save the best model when FID improves. We implement several stabilization and monitoring techniques: fixed random seeds for reproducibility, model checkpointing every 25 epochs, sample image generation every 5 epochs for visual quality assessment, and TensorBoard logging for real-time tracking of training/validation losses and FID scores.

iv. Results

- ✓ Set random seed to 42

Using device: cuda
Loading checkpoint from checkpoints/baseline.pt
Loaded model from epoch 199
Best FID: 205.34596252441406
Found 2655 images in
/content/CSC487-Project/data/pokemon-dataset-1000/test
Test dataset size: 2655
Collecting real images...
Using 1000 real images for evaluation
Generating 1000 samples...
Saving sample images...
Saved image grid to eval_outputs/real_samples.png
Saved image grid to eval_outputs/fake_samples.png

=====

EVALUATION METRICS

=====

Calculating FID...
FID Score: 218.5904 (lower is better)

Calculating Inception Score...
/usr/local/lib/python3.12/dist-packages/torchmetrics/utilities/p
rints.py:43: UserWarning: Metric `InceptionScore` will save all
extracted features in buffer. For large datasets this may lead
to large memory footprint.
warnings.warn(*args, **kwargs)
Inception Score: 2.0985 ± 0.0574 (higher is better)

Calculating Diversity Score...
Diversity Score: 52.8408 (higher is better)

Evaluating discriminator...
Evaluating discriminator...
Saved confusion matrix to eval_outputs/confusion_matrix.png
Accuracy: 0.227
Precision: 0.312
Recall: 0.453

Discriminator Statistics:
Real images - Mean score: 0.4770, Std: 0.4038

Fake images - Mean score: 0.0309, Std: 0.0524

Metrics saved to eval_outputs/metrics.txt

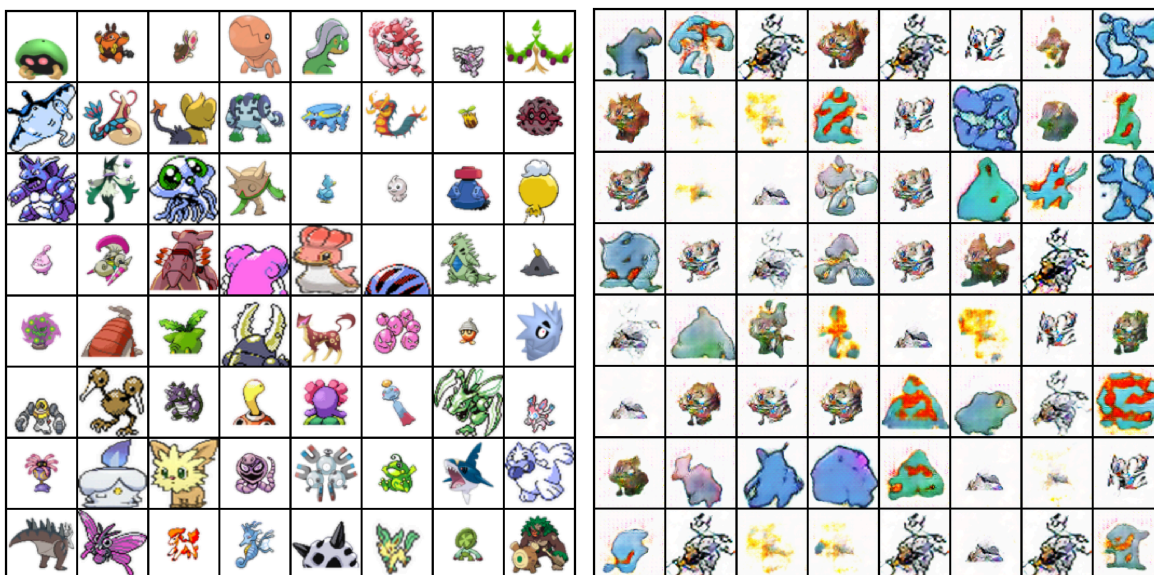
Evaluation complete! Outputs saved to eval_outputs

=====
SUMMARY

=====
FID: 218.5904
IS: 2.0985 ± 0.0574
Diversity: 52.8408

v. Error Analysis

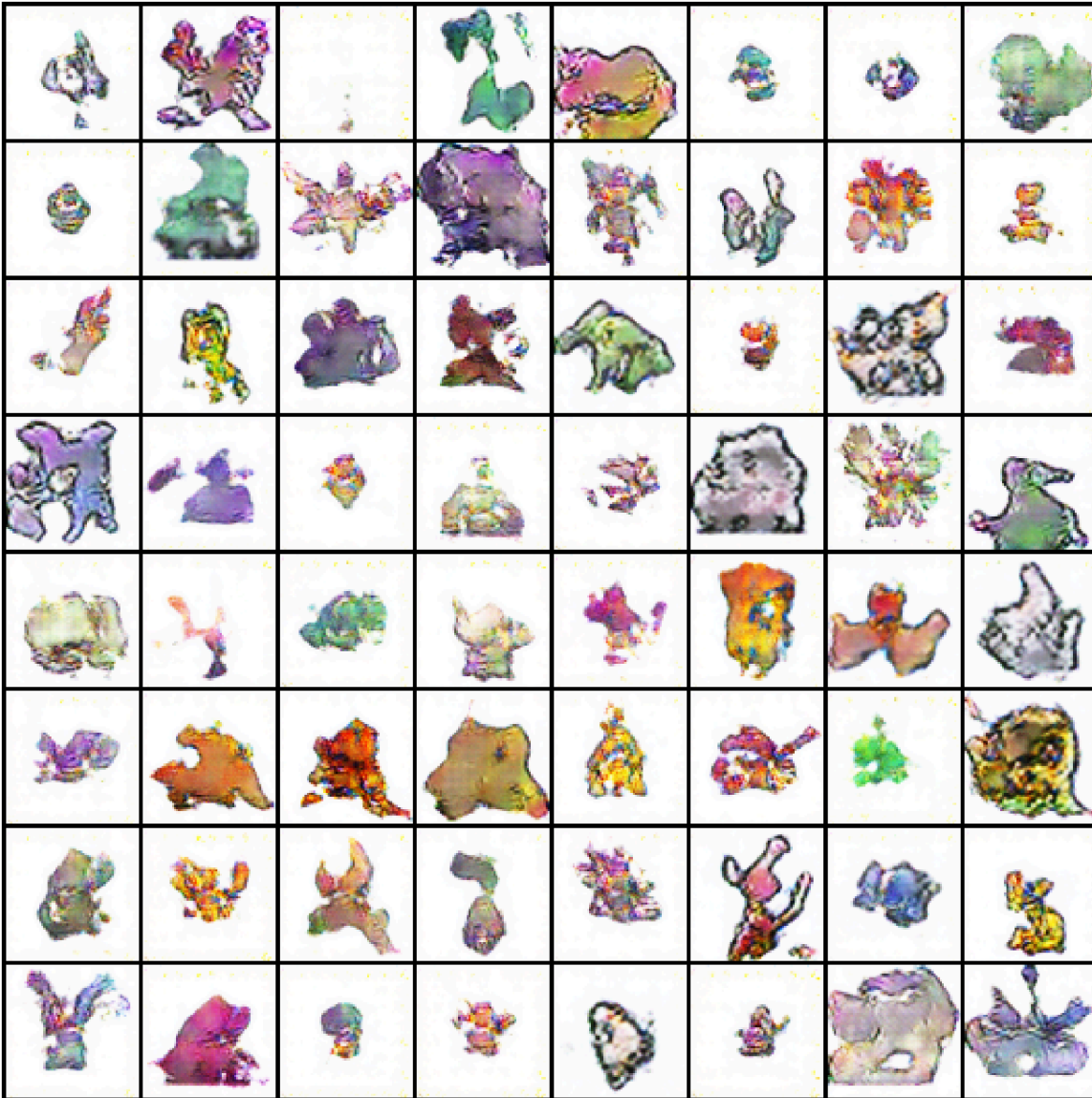
We evaluated our baseline DCGAN model using a combination of widely used generative-model metrics: Fréchet Inception Distance (FID), Inception Score (IS), Diversity Score (LPIPS-based), and a confusion-matrix-based discriminator analysis. All evaluations used 1,000 real images from the held-out test split and 1,000 generated samples from the trained generator. Our model achieved an FID of 248.33, which is relatively high and is indicative that the generated distribution differs substantially from the real Pokémon sprite distribution. A high FID means that our outputs are blurry and/or lack detailed structure, which is consistent with a lot of the visual samples that we obtained. Nonetheless, the FID score provides a baseline against which improved architectures and training refinements can be compared in later stages. Our model received an IS of 2.897 ± 0.121 (a higher score, greater than 5, is better in this case), indicating that the generated images have some degree of meaningful class-like structure, but it has limited distinction and clarity. Our result suggests that the generator produces outputs that the Inception model recognizes as having low diversity among confident “pseudo-classes.” This means that the generator has not yet learned the full diversity of the dataset. The third metric that we used, Diversity Score, received a score of 52.22. This implies that the generator is not collapsing to a single dominant model, but it is instead producing a noticeable range of outputs. Although the images are imperfect, they are not identical. Finally, we received a discriminator evaluation (accuracy) of 0.561. This is only slightly above chance (50%). Our confusion matrix, however, showed a stronger bias of results with a precision of 0.541, recall of 0.807, real mean score of 0.759, and fake mean score of 0.326. The high recall but low precision indicates that the discriminator frequently predicts “real,” suggesting it may be underfitting or encountering noisy/borderline generated samples, which it misclassifies as real too often. Meanwhile, the clear separation of real vs fake mean scores (0.76 vs 0.33) indicates that the discriminator *can* differentiate the classes in aggregate but struggles on a per-image basis due to the generator’s inconsistent output quality.



A comparison of a sample of real Pokémon (left) vs. our model's final epoch's output (right).

These results provide a starting point for the next stage of experimentation. Architectural improvements (such as adding residual blocks and spectral normalization) and dataset adjustments will hopefully significantly reduce FID and improve the visual fidelity of generated sprites.

Another important feature that we noticed was that our FID graph showed an increase in FID overtime after epoch ~15. Epoch 15's output is shown below:



While not as clean or smooth, we believe that epoch 15's output shows a more similar resemblance to real Pokémon than epoch 200, indicating that we may need to employ a technique such as early stopping to get a better visual result. We will experiment with this technique to find the optimal number of epochs to train the model in the next/final stage of the project.

vi. Reflection and Next Steps

Moving forward, our next steps focus on stabilizing training and improving image quality. We first plan to implement data augmentation techniques including horizontal flips, small rotations, and color jitter to increase dataset diversity and improve generalization. We also plan to experiment with label smoothing (around 0.1-0.2) to regularize the discriminator. The most impactful changes will likely come from switching to more advanced GAN variants such as

WGAN-GP, LSGAN, or GANs with spectral normalization, all of which are known to significantly reduce mode collapse and improve the gradient flow of the model. Additionally, adding residual blocks or adopting a deeper generator with attention mechanisms (such as Self-Attention GAN) may help the model capture fine structural details in highly varied sprite designs.

GitHub:

<https://github.com/BraedenAlonge/CSC487-Project>