Final Project

AnimeGen - Anime Face Image Generation with Generative Adversarial Networks

Kelvin K. Ahiakpor

CS463: Computer Vision

Dr. David Sasu

December 14, 2025

Introduction

Generative Adversarial Networks (GANs) [3], introduced in 2014, marked a breakthrough in image generation, enabling neural networks to produce realistic visual content from random noise. This project implements a Deep Convolutional GAN (DCGAN) to generate convincing 64x64 anime character faces, a task that requires learning visual patterns of the anime art style, including large eyes, vibrant hair colors, and simplified facial features. This project is useful for artists experiencing creative block whiles ideating new character designs. It also provides a fun exploration of Artificial Intelligence capabilities. The anime domain is suitable for training GANs because of its similar structured faces and abundance of examples offering diversity in hair colors, expressions, and overall character designs. The DCGAN implemented demonstrates the network's ability to successfully replicate structural consistency and creative variation.

Methodology

Data

The dataset used is the Anime Face Dataset from Kaggle [1], containing 63,565 anime character faces scraped from www.getchu.com [2]. The dataset has great size variation, with dimensions ranging from 25×25 to 220×220 pixels. Size distribution analysis revealed that 66.6% of images fell in the medium range (80-120px), with 27.1% in the small range (50-80px) and only 1.3% classified as tiny (<50px). 28.4% of images were smaller than the target 64×64 resolution. These were upsampled with bilinear interpolation, introducing slight blur but increasing dataset diversity.

Image preprocessing used the following pipeline: (1) resizing to 64×64 pixels via bilinear interpolation, (2) center cropping for consistent framing, (3) conversion to tensors, and (4) normalization to [-1, 1] range using mean and standard deviation of 0.5 per channel. This normalization places tensors within the output ranges of the Tanh activation in the Generator to ensure proper gradient flow during training.

Model

I implemented a DCGAN [5] consisting of two adversarially trained networks. Figure 4 in Appendix A shows an architecture similar to this project's, although it uses a 5×5 kernel, not 4×4 here, and starts with 1024, not 512 channels.

The **Discriminator** is a VGG-inspired classifier that progressively downsamples images while extracting hierarchical features with a channel expansion pattern (64→128→256→512). Spatial dimensions are reduced with strided convolutions rather than VGG's max-pooling, enabling learnable downsampling. The network processes 64×64 RGB images through four convolutional blocks:

- Layer 1: Conv2d(64 channels, stride=2, kernel=4) → LeakyReLU(0.2) → 32×32
- Layer 2: Conv2d(128, stride=2) → BatchNorm → LeakyReLU → 16×16
- Layer 3: Conv2d(256, stride=2) → BatchNorm → LeakyReLU → 8×8
- Layer 4: Conv2d(512, stride=2) → BatchNorm → LeakyReLU → 4×4
- Output: Conv2d(1, stride=1) → Sigmoid → Real/fake probability

Some key design include omitting batch normalization from the first layer to prevent input distribution instability. LeakyReLU(α=0.2) activations to allow gradient flow for negative inputs. The network is also fully convolutional with not fully connected layers. The Discriminator contains 2,765,568 parameters.

The **Generator** implements an inverse convolutional network that upsamples from a compact latent representation to full resolution. It mirrors the Discriminator's structure in reverse by progressively upsampling spatial dimensions while compressing channel depth (512→256→128→64→3).

After initializing a 100-dimensional noise vector sampled from a standard normal distribution, the network projects to 512×4×4 through a dense layer, then applies four transposed convolutional blocks:

- Initial: Dense(100 → 8192) → Reshape(512×4×4)
- Layer 1: ConvTranspose2d(256, stride=2, kernel=4) → BatchNorm → ReLU → 8×8
- Layer 2: ConvTranspose2d(128, stride=2) → BatchNorm → ReLU → 16×16
- Layer 3: ConvTranspose2d(64, stride=2) → BatchNorm → ReLU → 32×32
- Output: ConvTranspose2d(3, stride=2) → Tanh → 64×64 RGB image

The Generator uses ReLU activations in its hidden layers for faster convergence and a Tanh output activation to map generated values to the normalized image range of [-1, 1]. Batch normalization is applied on all layers except the output to stabilize training. The Generator contains 3,600,256 parameters.

Both networks use weight initialization from a normal distribution N(0, 0.02) as specified in the DCGAN paper. This initialization prevents vanishing or exploding gradients during early training.

Hyperparameters

Training followed the GAN objective, where the Discriminator maximizes $\log D(x)+\log(1-D(G(z)))$ and the Generator minimizes $\log(1-D(G(z)))$, or equivalently maximizes $\log D(G(z))$ for stronger gradients.

Hyperparameter configuration includes Adam **optimizer** with learning rate 0.0002, β_1=0.5, β_2=0.999, the Binary Cross-Entropy (BCE) **loss**, a batch size of 128 images, a latent input noise vector dimension of 100, and **Label smoothing** for real labels = 0.9 (instead of 1.0) to prevent Discriminator overconfidence which will provide unwanted vanishing gradients to the Generator.

The choice of β_1=0.5 (rather than the standard 0.9) reduces momentum in the Adam optimizer, which has been shown to stabilize GAN training by preventing excessive oscillations.

Training

Training each epoch took approximately 2 minutes for the full dataset, effectively totaling 3.5 hours. I initially trained a random subset of 10,000 images for 20 epochs to test the DCGAN and later set epochs to 100 as originally planned. With only 15 seconds per epoch, it took about 30 minutes to fully train.

The dataset was loaded with a batch size of 128, resulting in 78 batches per epoch. No data augmentation was applied, as GANs learn from the raw distribution of real images rather than augmented variations.

Both networks were trained alternately within each iteration: first updating the Discriminator on real and fake images separately, then updating the Generator to fool the Discriminator. This 1:1 training ratio maintained adversarial balance throughout training.

Results

The DCGAN successfully learned to generate convincing anime faces over 100 training epochs.

Figure 1: Training Progression: Generated Samples Over Epochs

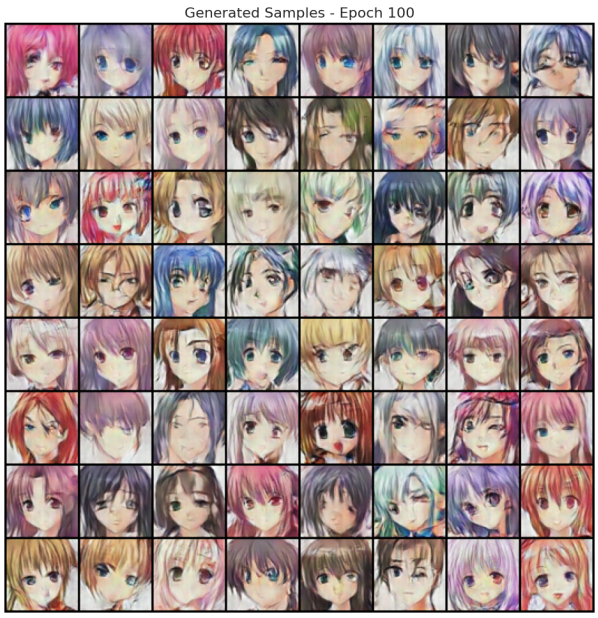

Figure 2: Final Epoch Results

Qualitative Assessment

Figure 1, shows the network's image generation progress at 20-epoch intervals. Epoch 1 discernibly starts with random noise. Epoch 20 shows the network can learn early on in training as it shows the formation of anime faces with pronounced eyes, hair, and face shapes; however, with a red tint, likely resulting from overbalanced red channels in noise. From epoch 40, the red tint disappears, and we observe effortless generation of varied hair colors. Epoch 60 displays a mild improvement from epoch 40, in generating more varied hairstyles. Some issues with epoch 60 are faces with different eye colors and unclear lip and

nose generation, resembling a smudge effect. Only slight differences exist between epochs 80 and 100, suggesting that the network may have stopped learning. Epoch 100 shows are slightly more consistent eye colors than epoch 80. What could be improved about the network after its training is learning to generate bolder hair colors, varying hairstyles and more distinct lips and noses. There were also a few faces with just one eye, smudged jaws, no lips and noses that were not penalized by the discriminator and somehow made it to epoch 100. Overall, the network effectively captures key anime face features, including expressive eyes, hair colors spanning the spectrum (pink, blue, purple, green, blonde, brown), yet fails to penalize generations that introduce inconsistent facial features and blur artifacts.

The final images generated are shown in Figure 2 above as visual support of earlier comments.

Quantitative Analysis

The 100 training epochs resulted in a Generator (G) loss of 3.04 and Discriminator (G) loss of 0.50. The G/D loss ratio of 5.44 indicates that D became moderately stronger than the G, explaining why learning plateaued after epoch 80. The high G loss (mean: 3.66, std: 1.60) compared to stable D loss (mean: 0.67, std: 0.36) suggests the G struggled to consistently fool the D in later epochs. Despite this imbalance, no mode collapse occurred - all faces show feature variation. Additionally, observe the G/D loss curves in Figure 3, which show oscillations but never a loss extremely close to 0.

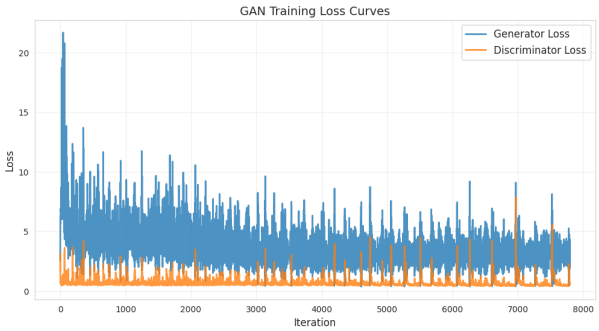

Figure 3: G/D Loss Curves Displaying Oscillation

Ethics & Impact

Image generation technology raises questions about ethics that technology adopters must address.

Copyright and Consent

The first ethical issue with deploying this project outside a controlled academic environment is the curation of the dataset. The images were scraped from Getchu, likely without explicit consent from the original artists. Web scraping, on its own, is perceived as illegal and unethical in many contexts. Here, we add to that by taking anime face designs, which are the artists' intellectual property (IP). Deploying this project will likely lead to serious copyright issues. A clear example comes from earlier this year: we were all amazed by OpenAI models that could generate Studio Ghibli-style renditions of input images until they faced IP issues and had to stop. Additional ethical concerns include deepfakes, impact on

creative labor, and bias and representation, since there are few black anime characters on Getchu.

This project serves an educational purpose of demonstrating GANs capabilities within an academic setting. Nevertheless, a wider application of this technology demands continuous moral consideration, laws, and cooperation among stakeholders

Reflection

I have come to recognize Computer Vision as a highly rewarding domain of computer science. It first involves challenging our notions of how we see, how our brains process light signals, and testing these 'seeing' theories with various deep learning architectures and concepts, such as convolutions. The progress from single-image classification to real-time object detection in footage and image generation represents significant advances in human problem-solving and self-understanding.

The most challenging aspect of this project was quickly learning about GANs and attempting to implement them during a two-week period. Although it leverages convolutions from image classification, it also brings new ideas such as latent vectors, reverse convolutions, and an appreciation of statistics - specifically sampling input noise in a calculated manner using normal distributions. The unconventional idea of training not to completely minimize loss was quite novel, although understandable in a bid to avoid mode collapse. Reading the GAN and DCGAN papers was a bittersweet experience because of the technicality involved, but after fairly understanding the papers and engaging with various resources, I could employ the best practices in training my network. These included unconventionally setting momentum to $\beta_1 = 0.5$ to stabilize training and using LeakyReLU to allow negative inputs and prevent dying neurons. I improved my understanding of GANs overall and also gained skills and patience in perusing technical papers.

For future work, I wish I could make a GAN that allows control over specific kinds of features it generates. In that case, after training on a dataset, I could input specific features I want for it to generate in an anime face. I initially named this 'explainable GANs' (XGANs), but apparently, what I describe is Conditional GANs or StyleGANs [4], since 'explainable' AI (XAI) deals with interpreting a model's decisions rather than controlling its outputs.

Appendix

A DCGAN Architecture Diagram

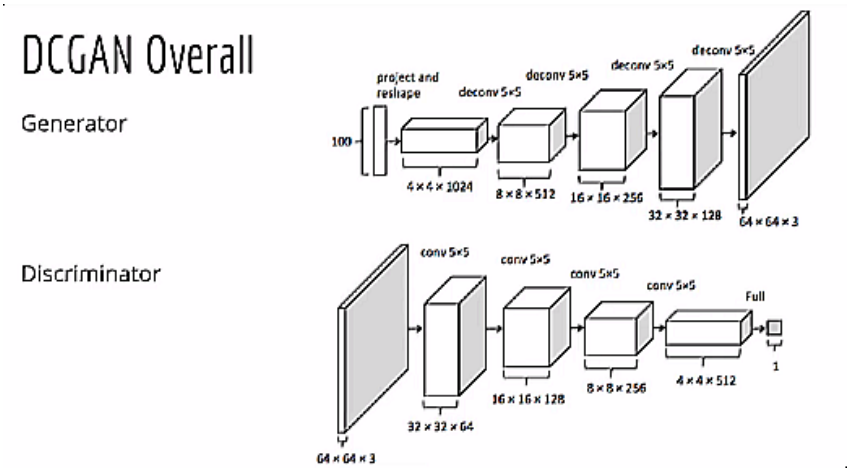

Figure 4: DCGAN architecture w/ Generator and Discriminator with layer dimensions and operations.

References

[1] Spencer Churchill. Anime face dataset, 2019.

[2] Getchu. Getchu.com, 2025.

[3] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 06 2014.

[4] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.