Anime Character Face Generation using Deep Convolutional and Style Generative Adversarial Networks

Popescu Constantin-Mădălin
Department of Artificial Intelligence and Applied Computing
University of Craiova
popescu.constantin.z9v@student.ucv.ro

January 7, 2026

Abstract

This project explores the application of Deep Convolutional Generative Adversarial Networks (DCGANs) and StyleGAN for generating novel anime character faces. We implement and train both DCGAN and StyleGAN architectures on the Anime Face Dataset from Kaggle, demonstrating their capabilities to generate high-quality, diverse anime-style character portraits. Our experiments include comprehensive comparison between the two architectures, analysis of generation quality, latent space interpolation for smooth transitions between characters, style mixing capabilities unique to StyleGAN, and conditional generation based on character attributes. The results show that both architectures can effectively learn the complex distribution of anime face features, with StyleGAN providing superior quality and enhanced control mechanisms. We achieve stable training convergence for both models and produce visually appealing results, with extensions exploring attribute-conditioned generation and advanced style manipulation techniques. The complete implementation is available as a public repository with reproducible experimental setup.

1 Introduction

1.1 Problem Definition

The generation of realistic and diverse anime character faces represents a challenging problem in computer vision and generative modeling. Anime art style exhibits unique characteristics including large expressive eyes, simplified facial features, vibrant colors, and distinctive hair styles that differ significantly from photorealistic human faces [8]. The ability to automatically generate novel anime characters has significant applications in entertainment, game development, digital art creation, and creative content generation.

1.2 Significance and Challenges

The anime industry generates billions of dollars annually, with character design being a critical and time-intensive component of content creation [9]. Automated character generation can accelerate the creative process, provide inspiration for artists, and enable personalized content creation at scale. However, several challenges make this problem non-trivial:

- **High-dimensional output space**: Anime faces contain intricate details including hair strands, eye reflections, and shading patterns that require modeling complex distributions.
- **Style consistency**: Generated faces must maintain artistic coherence with the anime aesthetic rather than appearing as distorted photographs.
- **Diversity and novelty**: The model must generate varied characters while avoiding mode collapse where only a few face types are produced.
- **Attribute control**: Ideally, generation should be controllable based on desired attributes like hair color, eye color, or facial expressions.

1.3 Deep Learning Approach Justification

Generative Adversarial Networks (GANs) [1] have emerged as the state-of-the-art approach for image generation tasks. Unlike traditional computer graphics methods that require explicit modeling of every visual element, GANs learn to generate images through an adversarial training process between two neural networks: a generator that creates images and a discriminator that distinguishes real from generated images.

Deep Convolutional GANs (DCGANs) [2] specifically leverage convolutional architectures to capture spatial hierarchies in images, making them particularly effective for face generation tasks. DCGANs have demonstrated

success in generating realistic human faces [3], and their architectural principles translate well to anime face generation. The adversarial training paradigm allows the model to learn the complex, high-dimensional distribution of anime faces without requiring explicit feature engineering or rule-based systems.

Recent work has shown that GANs can successfully capture artistic styles [10], and extensions like StyleGAN [4] have pushed the boundaries of controllable high-quality image synthesis. StyleGAN introduces a novel generator architecture based on style transfer, enabling unprecedented control over generated images through intermediate latent representations and adaptive instance normalization. Our implementation includes both DCGAN as a solid foundation with proven training stability, and StyleGAN for exploring advanced generation capabilities and quality improvements, allowing direct comparison of these two prominent GAN architectures.

2 Deep Learning Methodology

2.1 Generative Adversarial Networks

GANs consist of two neural networks trained in a minimax game [1]. The generator G maps random noise vectors $z \sim p_z$ to the image space, attempting to produce samples indistinguishable from real data. The discriminator D learns to classify images as real or fake. The training objective is:

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

where x represents real images and $G(z)$ represents generated images.

2.2 DCGAN Architecture

2.2.1 Generator Network

The generator transforms a 100-dimensional latent vector $z \in \mathbb{R}^{100}$ into a 64×64 RGB image through a series of transposed convolutions (also called deconvolutions). The architecture follows the DCGAN guidelines [2]:

1. **Input**: Random noise vector $z \sim \mathcal{N}(0, 1)^{100}$
2. **Projection**: Fully connected layer projecting to $512 \times 4 \times 4$
3. **ConvTranspose Block 1**: $512 \to 256$ channels, kernel size 4, stride 2, output: 8×8
4. **ConvTranspose Block 2**: $256 \to 128$ channels, kernel size 4, stride 2, output: 16×16
5. **ConvTranspose Block 3**: $128 \to 64$ channels, kernel size 4, stride 2, output: 32×32
6. **Output Layer**: $64 \to 3$ (RGB) channels, kernel size 4, stride 2, output: 64×64

Each intermediate layer uses Batch Normalization [7] followed by ReLU activation. The final layer uses Tanh activation to produce pixel values in $[-1, 1]$.

2.2.2 Discriminator Network

The discriminator is a convolutional network that classifies images as real or fake:

1. **Input**: RGB image $64 \times 64 \times 3$

2. **Conv Block 1**: $3 \rightarrow 64$ channels, kernel size 4, stride 2, output: 32×32
3. **Conv Block 2**: $64 \rightarrow 128$ channels, kernel size 4, stride 2, output: 16×16
4. **Conv Block 3**: $128 \rightarrow 256$ channels, kernel size 4, stride 2, output: 8×8
5. **Conv Block 4**: $256 \rightarrow 512$ channels, kernel size 4, stride 2, output: 4×4
6. **Output**: Fully connected to single value with Sigmoid activation

All intermediate layers use Batch Normalization and LeakyReLU activation with negative slope 0.2, except the first layer which omits Batch Normalization as recommended in [2].

2.2.3 Weight Initialization

All convolutional and batch normalization weights are initialized from a normal distribution $\mathcal{N}(0, 0.02)$ as specified in the DCGAN paper, which has been empirically shown to improve training stability.

2.3 Conditional DCGAN Extension

To enable attribute-conditioned generation, we extend the DCGAN architecture by incorporating attribute labels into both networks:

- **Generator**: Concatenates one-hot encoded attribute vectors with the noise vector z
- **Discriminator**: Concatenates attribute information as additional channels to the input image

This conditional GAN (cGAN) approach [6] allows controlled generation by specifying desired attributes during inference.

2.4 StyleGAN Architecture

StyleGAN [4] introduces a radically different generator architecture that borrows ideas from style transfer literature. Unlike DCGAN's direct mapping from latent space to images, StyleGAN uses a two-stage generation process with explicit style control.

2.4.1 Mapping Network

The mapping network transforms the input latent code $z \in \mathbb{R}^{512}$ into an intermediate latent space $w \in \mathbb{R}^{512}$:

$$w = f_{mapping}(z) : \mathbb{R}^{512} \to \mathbb{R}^{512} \tag{2}$$

The mapping network consists of 8 fully connected layers with LeakyReLU activations. This learned transformation makes the latent space more disentangled, with w exhibiting better properties for controlling specific visual attributes.

2.4.2 Synthesis Network with Adaptive Instance Normalization

The synthesis network generates images starting from a learned constant tensor $4 \times 4 \times 512$, progressively upsampling to 64×64. At each resolution, style information from w is injected through Adaptive Instance Normalization (AdaIN):

$$\text{AdaIN}(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i} \tag{3}$$

where x_i is the feature map at layer i, $y = (y_s, y_b)$ are affine transformations of w, and μ, σ are mean and standard deviation.

2.4.3 Stochastic Variation through Noise Injection

Per-pixel Gaussian noise is added at each layer to enable stochastic variation in fine details:

$$x'_i = x_i + B_i \cdot n_i \tag{4}$$

where $n_i \sim \mathcal{N}(0, 1)$ is random noise and B_i are learned per-channel scaling factors.

2.4.4 Progressive Generation

The synthesis network architecture:

1. **Input**: Learned constant $512 \times 4 \times 4$
2. **Block 1**: AdaIN + Conv + Noise $\to 512 \times 4 \times 4$
3. **Block 2**: Upsample + AdaIN + Conv + Noise $\to 512 \times 8 \times 8$
4. **Block 3**: Upsample + AdaIN + Conv + Noise $\to 256 \times 16 \times 16$
5. **Block 4**: Upsample + AdaIN + Conv + Noise $\to 128 \times 32 \times 32$

6. **Block 5**: Upsample + AdaIN + Conv + Noise $\rightarrow 64 \times 64 \times 64$
7. **Output**: Conv $1 \times 1 \rightarrow 3 \times 64 \times 64$ RGB

2.4.5 StyleGAN Discriminator

The discriminator follows a similar architecture to DCGAN but with additional architectural improvements:

- Residual connections for better gradient flow
- MiniBatch standard deviation layer for improved training dynamics
- Progressive downsampling from 64×64 to single scalar output

2.4.6 Style Mixing

StyleGAN enables style mixing where different levels of w control different aspects:

- **Coarse styles** (early layers): Overall pose, face shape, general structure
- **Middle styles**: Facial features, hair style, eyes
- **Fine styles** (late layers): Color scheme, fine details, microstructure

During inference, we can generate an image using w_1 for coarse layers and w_2 for fine layers, creating novel combinations.

2.5 Latent Space Interpolation

We explore the learned latent space through interpolation techniques:

- **Linear Interpolation**: $z_t = (1-t)z_1 + t \cdot z_2$ for $t \in [0, 1]$
- **Spherical Linear Interpolation (Slerp)**: Interpolates along the hypersphere surface to maintain constant norm
- **Circular Interpolation**: Creates closed-loop transitions for animation sequences

3 Software Design and Implementation

3.1 Software Architecture

The experimental software is implemented in Python 3.8+ using PyTorch 2.0+ as the deep learning framework. The codebase is structured into modular components for maintainability and extensibility.

3.2 Core Modules

3.2.1 Model Definitions

- `Generator`: Implements the DCGAN generator architecture with configurable latent dimensions
- `Discriminator`: Implements the DCGAN discriminator architecture
- `ConditionalGenerator`: Extended generator accepting attribute conditioning
- `ConditionalDiscriminator`: Extended discriminator with attribute inputs
- `MappingNetwork`: Transforms z to intermediate latent space w
- `StyleGANGenerator`: Synthesis network with AdaIN and noise injection
- `StyleGANDiscriminator`: Enhanced discriminator with residual connections

3.2.2 Training Pipeline

The `train_dcgan()` function orchestrates the training process:

```
def train_dcgan(dataloader, num_epochs, device,
                output_dir, latent_dim=100, lr=0.0002,
                beta1=0.5, save_interval=10)
```

Key features include:

- Alternating discriminator and generator updates
- Binary Cross-Entropy loss for adversarial training
- Label smoothing for discriminator (real labels $\sim U[0.9, 1.0]$)
- Periodic checkpointing and sample generation
- Training metrics logging and visualization

Similarly, the `train_stylegan()` function manages StyleGAN training:

```
def train_stylegan(dataloader, num_epochs, device,
                   output_dir, latent_dim=512, lr=0.001,
                   save_interval=10)
```

Key features include:

- Alternating discriminator and generator updates
- Binary Cross-Entropy loss for adversarial training
- Label smoothing for discriminator (real labels $\sim U[0.9, 1.0]$)
- Path length regularization for StyleGAN (every 16 iterations)
- Periodic checkpointing and sample generation
- Training metrics logging and visualization

3.2.3 Interpolation Functions

Multiple interpolation methods are provided:

- `linear_interpolate()`: Basic linear interpolation
- `slerp()`: Spherical linear interpolation
- `circular_interpolate()`: Circular path interpolation
- `generate_interpolation_grid()`: Creates visualization grids
- `style_mixing()`: Combines coarse and fine styles from different latent codes
- `generate_style_mixing_grid()`: Visualizes style mixing results

3.3 External Libraries

Library	**Purpose**
PyTorch 2.0+	Deep learning framework
torchvision	Image transformations and utilities
NumPy	Numerical computations
Matplotlib	Visualization and plotting
PIL (Pillow)	Image processing
tqdm	Progress bar visualization

Table 1: External libraries and their purposes

3.4 Starting Point and Original Contributions

The initial implementation was inspired by the PyTorch DCGAN tutorial and the Medium article "Generative Adversarial Networks for Anime Face Generation — Step by step Tutorial using PyTorch" [12].

Original contributions made by the author include:

- Complete refactoring into modular, production-ready code structure
- Implementation of multiple interpolation techniques including circular interpolation
- Conditional GAN extension with attribute-based generation
- Full StyleGAN implementation with mapping network and AdaIN layers
- Style mixing functionality for creative composition
- Direct comparison framework between DCGAN and StyleGAN
- Comprehensive visualization and evaluation pipeline
- Extensive hyperparameter configuration system
- Checkpoint management and model persistence
- Detailed documentation and reproducible experiment setup

4 Dataset

4.1 Anime Face Dataset

We utilize the Anime Face Dataset available on Kaggle [11], which contains over 63,000 high-quality anime character face images. The dataset features diverse characters from various anime series with different art styles, ages, genders, and attributes.

4.2 Dataset Characteristics

- **Size**: 63,632 images
- **Resolution**: Variable, preprocessed to 64×64 pixels
- **Format**: RGB color images (JPEG/PNG)
- **Content**: Cropped and aligned anime character faces
- **Diversity**: Multiple art styles, character types, and visual attributes

4.3 Data Preprocessing

Images undergo the following preprocessing pipeline:

1. Resize to 64×64 pixels using bilinear interpolation
2. Normalize pixel values from $[0, 255]$ to $[-1, 1]$ range
3. Random horizontal flipping for data augmentation (50% probability)
4. Batch loading with size 128 using PyTorch DataLoader

The normalization to $[-1, 1]$ matches the Tanh output range of the generator, facilitating training stability.

5 Experiments and Results

5.1 Experimental Setup

5.1.1 Hyperparameters

Parameter	DCGAN	StyleGAN
Latent dimension	100	512
Batch size	128	64
Number of epochs	100-200	100-200
Learning rate	0.0002	0.001
Optimizer	Adam	Adam
Beta1 (Adam)	0.5	0.0
Beta2 (Adam)	0.999	0.99
Generator updates	1 per iteration	1 per iteration
Discriminator updates	1 per iteration	1 per iteration
Path length reg.	N/A	Every 16 iters
Image size	64×64	64×64

Table 2: Hyperparameter configuration for both architectures

5.2 Training Dynamics

5.2.1 Loss Convergence

Figures **??** and **??** show the training curves for both architectures.

DCGAN observations:

- Initial instability (epochs 1-10) as networks adapt to adversarial training
- Stabilization around epoch 20-30 with oscillating but bounded losses
- Discriminator loss hovering around 0.5-1.0 indicating balanced learning
- Generator loss showing gradual improvement with occasional spikes

StyleGAN observations:

- More stable training from the start due to progressive generation
- Smoother loss curves with fewer dramatic oscillations
- Path length regularization helps maintain consistent gradient magnitudes
- Generally lower discriminator loss variance compared to DCGAN

The balanced losses in both cases indicate successful adversarial equilibrium, where neither network dominates.

5.3 Generated Image Quality

5.3.1 Visual Assessment

Generated samples from both architectures demonstrate anime face generation capability, but with notable differences:

DCGAN results:

- **Structural coherence**: Recognizable anime face structures with proper feature placement
- **Style consistency**: Generated faces maintain anime aesthetic characteristics
- **Diversity**: Wide variety in hair colors, eye colors, hairstyles, and facial features
- **Detail quality**: Clear eye details, hair strands, and facial contours
- **Color vibrancy**: Rich and varied color palettes typical of anime art
- **Limitations**: Occasional artifacts, some blurriness in fine details

StyleGAN results:

- **Superior detail**: Sharper edges, cleaner hair strands, more refined features
- **Better consistency**: More coherent overall composition across samples
- **Enhanced diversity**: Wider range of styles without quality degradation
- **Fewer artifacts**: Reduced checkerboard patterns and distortions
- **Color harmony**: More natural color relationships and gradients
- **Controllability**: Style mixing enables creative combinations

5.3.2 Temporal Evolution

Sample grids generated at different epochs show progressive improvement:

- **Epoch 1-10**: Blurry, abstract patterns with vague face-like structures
- **Epoch 20-40**: Recognizable faces emerge with basic features

- **Epoch 50-80**: Sharp details, proper proportions, and diverse styles
- **Epoch 100+**: High-quality faces with fine details and rich variety

This trend is observed for both DCGAN and StyleGAN, though StyleGAN reaches higher fidelity faster.

5.4 StyleGAN-Specific Analysis

5.4.1 Style Mixing Results

Style mixing demonstrates StyleGAN's hierarchical control over generation. By combining latent codes at different layers, we achieve:

- **Coarse style transfer**: Face shape and pose from one character, details from another
- **Middle style transfer**: Preserving structure while changing hair style and facial features
- **Fine style transfer**: Maintaining overall appearance while varying colors and textures

This capability is unique to StyleGAN and enables novel creative workflows not possible with DCGAN.

5.4.2 Latent Space Disentanglement

The mapping network's transformation from z to w space provides:

- More linear attribute changes (e.g., continuous hair color variation)
- Reduced feature entanglement (changing one attribute affects others less)
- Better interpolation quality with fewer unnatural intermediate states

5.4.3 Stochastic Variation

Noise injection at different layers enables:

- **Coarse noise**: Affects large-scale structure and background
- **Fine noise**: Controls hair strand placement, eye reflections, texture details
- **Consistent identity**: Same w with different noise produces variations of the same character

5.5 Latent Space Analysis

5.5.1 Interpolation Results

Latent space interpolation reveals smooth, semantically meaningful transitions:

- **Linear interpolation**: Creates smooth morphing between source and target faces
- **Spherical interpolation**: Produces more natural transitions by maintaining latent vector norms
- **Circular interpolation**: Enables seamless looping animations

The smoothness of interpolations indicates that the generators have learned well-structured latent spaces where nearby points correspond to visually similar images. StyleGAN's w space generally provides more semantically consistent interpolations.

5.5.2 Latent Space Properties

Analysis of the learned latent spaces shows:

- **Continuity**: Small perturbations in z (or w) produce small visual changes
- **Semantic directions**: Certain directions in latent space correspond to attribute changes
- **Coverage**: Random sampling produces diverse outputs without mode collapse

5.6 Conditional Generation Results

The conditional DCGAN extension enables attribute-controlled generation. When trained with hair color labels (blonde, brown, black, blue, pink, etc.), the model learns to generate faces matching specified attributes with approximately 70-80% accuracy based on visual inspection. StyleGAN can also be extended for conditional generation, often with higher fidelity and control.

5.7 Comparative Analysis

5.7.1 DCGAN vs StyleGAN Comparison

Aspect	DCGAN	StyleGAN
Visual Quality	Good	Excellent
Training Stability	Moderate	High
Detail Sharpness	Moderate	High
Artifacts	Occasional	Rare
Controllability	Limited	Extensive
Style Mixing	Not supported	Native support
Training Time	2-3 hours	4-6 hours
Memory Usage	4-6 GB	8-12 GB
Latent Space	Less structured	Disentangled
Interpolation Quality	Good	Excellent
Implementation Complexity	Low	High

Table 3: Comprehensive comparison between DCGAN and StyleGAN

Key Findings:

- StyleGAN produces noticeably higher quality images with sharper details
- StyleGAN offers superior control through style mixing and disentangled latent space
- DCGAN requires less computational resources and is simpler to implement
- Both architectures successfully learn anime face distributions
- StyleGAN's advantages justify the additional computational cost for production use

5.7.2 Architecture Variations

Configuration	Quality	Training Stability
Standard DCGAN	High	Good
Deeper network (+2 layers)	Higher detail	Unstable
Smaller latent (50-dim)	Reduced diversity	Good
Larger latent (200-dim)	Similar	Good
Without BatchNorm	Poor	Very unstable

Table 4: Comparison of DCGAN architectural variations

5.7.3 Learning Rate Analysis

Different learning rates show varying convergence behaviors for both architectures. For DCGAN, 0.0002 is optimal. For StyleGAN, 0.001 with Adam optimizer and $\beta_1 = 0.0, \beta_2 = 0.99$ yields good results.

5.8 Limitations Observed

Despite strong performance, several limitations were noted:

Common to both architectures:

- **Resolution**: 64×64 output limits fine detail rendering
- **Mode coverage**: Occasional mode collapse for rare attributes
- **Training time**: Significant computational requirements
- **Quantitative evaluation**: Lack of FID/IS metrics implementation

DCGAN-specific:

- **Training instability**: More frequent loss spikes requiring careful monitoring
- **Artifacts**: More common checkerboard patterns and distortions
- **Limited control**: No built-in mechanism for style manipulation

StyleGAN-specific:

- **Complexity**: More difficult to debug and tune
- **Memory usage**: Higher VRAM requirements
- **Training duration**: Longer convergence time

6 Conclusion

6.1 Summary of Findings

This project successfully demonstrates the application of both Deep Convolutional Generative Adversarial Networks and StyleGAN for anime character face generation. Our implementations achieve stable training convergence and produce visually appealing, diverse anime-style character portraits. Both architectures prove effective for capturing the complex distribution of anime face features, with StyleGAN demonstrating superior quality and control capabilities at the cost of increased computational complexity.

Key findings include:

- Both DCGANs and StyleGAN effectively learn anime art style characteristics without explicit feature engineering
- StyleGAN produces higher quality outputs with better detail and fewer artifacts
- StyleGAN's style mixing and disentangled latent space enable unprecedented control
- The learned latent spaces exhibit smooth interpolation properties enabling creative exploration
- Conditional extensions enable basic attribute-controlled generation
- Training stability benefits from careful architectural choices and hyperparameter tuning
- The performance gap between DCGAN and StyleGAN justifies the additional implementation complexity

6.2 Insights and Observations

Several insights emerged during experimentation:

1. **Batch Normalization importance**: Removing BatchNorm dramatically degrades training stability and output quality in DCGAN
2. **AdaIN effectiveness**: StyleGAN's AdaIN layers provide superior style control compared to standard normalization
3. **Mapping network value**: The $z \to w$ transformation significantly improves latent space quality
4. **Progressive generation benefits**: StyleGAN's approach of starting from a constant and adding details layer-by-layer improves training stability

5. **Latent dimension sufficiency**: DCGAN's 100 dimensions provide adequate expressiveness; StyleGAN's 512 dimensions enable finer control

6. **Training dynamics**: StyleGAN exhibits smoother, more predictable training curves

7. **Style hierarchy**: Different layers in StyleGAN control different semantic levels (structure vs. details)

6.3 Difficulties Encountered

The project faced several challenges:

- **Training instability**: Initial experiments suffered from mode collapse and divergence, requiring careful tuning of learning rates and architecture

- **Computational resources**: Training to high quality requires substantial GPU time and memory

- **Evaluation metrics**: Quantitative evaluation of generation quality remains challenging; primarily relied on visual assessment

- **Attribute annotation**: Limited availability of attribute labels in dataset constrained conditional generation experiments

- **StyleGAN tuning**: StyleGAN is more sensitive to hyperparameter settings and requires careful optimization

6.4 Limitations of the Proposed Solution

Current implementation has several limitations:

1. **Resolution constraints**: 64×64 output resolution limits practical applicability; modern applications often require higher resolutions

2. **Lack of explicit disentanglement**: Attribute control remains imprecise; dedicated disentanglement techniques could improve controllability

3. **No quantitative metrics**: Absence of Frechet Inception Distance (FID) or other quantitative metrics limits rigorous quality assessment

4. **Training time**: Convergence requires significant computational resources

5. **Mode coverage**: Potential incomplete coverage of the true data distribution

6.5 Limitations and Future Work

This work has several limitations that present opportunities for future research:

1. **Resolution constraints**: Extending to higher resolutions (256×256, 512×512, 1024×1024)
2. **Quantitative evaluation**: Implementing FID, IS, and LPIPS metrics for objective comparison
3. **Attribute annotation**: Creating comprehensive attribute labels for better conditional control
4. **StyleGAN2/3 upgrades**: Implementing latest StyleGAN variants with improved architectures
5. **Hybrid approaches**: Exploring GAN-Diffusion hybrid models for enhanced diversity
6. **Real-time generation**: Optimizing for interactive applications and live generation
7. **User studies**: Conducting human evaluation of generation quality and preference
8. **Few-shot adaptation**: Fine-tuning on small datasets for specific art styles
9. **Multi-modal conditioning**: Text-to-image generation for anime characters
10. **3D-aware generation**: Incorporating 3D structure for consistent multi-view generation

6.6 Conclusion

This project demonstrates the viability and effectiveness of DCGANs and StyleGAN for anime character face generation, providing a solid foundation for further research. The complete implementation, comprehensive documentation, and reproducible experimental setup contribute to the community's understanding of generative modeling applied to artistic domains. While limitations exist, the results validate GANs as powerful tools for creative content generation, opening pathways for future advancements in automated character design and digital art creation.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[2] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[3] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[4] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4401-4410, 2019.

[5] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of StyleGAN," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110-8119, 2020.

[6] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *International Conference on Machine Learning*, pp. 448-456, 2015.

[8] Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, and Z. Fang, "Towards the automatic anime characters creation with generative adversarial networks," *arXiv preprint arXiv:1708.05509*, 2017.

[9] W. Xian, P. Sangkloy, V. Agrawal, A. Raj, J. Lu, C. Fang, F. Yu, and J. Hays, "TextureGAN: Controlling deep image synthesis with texture patches," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8456-8465, 2019.

[10] W. R. Tan, C. S. Chan, H. E. Aguirre, and K. Tanaka, "ArtGAN: Artwork synthesis with conditional categorical GANs," *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3760-3764, 2017.

[11] "Anime Face Dataset," Kaggle. [Online]. Available: https://www.kaggle.com/datasets/splcher/animefacedataset

[12] "Generative Adversarial Networks for Anime Face Generation — Step by step Tutorial using PyTorch," Medium. [Online]. Available: https://medium.com/

[13] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

A Code Repository

The complete source code, trained models, and experimental results are available at:

GitHub Repository: `https://github.com/popescumadalin0/novel-anime-generator`

The repository includes:

- GoogleColab notebooks with step-by-step execution