Deep Image Synthesis with GANs, VAEs, and Diffusion Models

Project Overview:

A comprehensive exploration of generative image synthesis comparing three leading architectures on the challenging CUB-200-2011 bird dataset. This project implements multiple GAN variants (Vanilla, LSGAN, WGAN-GP) from scratch with custom ResBlock architectures, explores VAEs with β-annealing optimization, and investigates diffusion models through DDPM and DDIM sampling strategies. The implementation includes custom loss functions, stability improvements like gradient penalty and β-annealing, and rigorous evaluation through FID scoring. WGAN-GP emerged as the strongest performer with a 33.07 FID score, outperforming diffusion models (34.73) and demonstrating the effectiveness of adversarial training with proper regularization.

GitHub Repository: GenVision

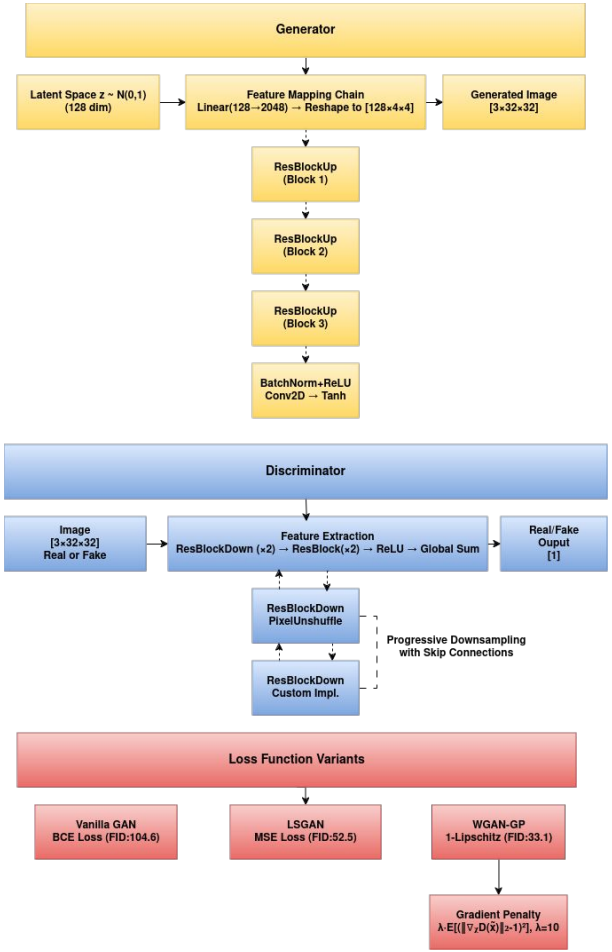
Key Technologies and Skills Used:

Languages & Frameworks: Python, PyTorch, TensorFlow, NumPy, OpenCV, scikit-learn, clean-fids, MLflow, Weights & Biases
Deep Learning: Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), Diffusion Models (DDPM, DDIM), Model Training, Hyperparameter Tuning, Loss Functions (Adversarial Loss, Reconstruction Loss, KL Divergence, Gradient Penalty)
Computer Vision: Image Synthesis, Image Generation, Feature Visualization

Generative Adversarial Networks (GANs) Architecture

Generator:

- Input: 128-dimensional noise vector (z ~ N(0,1))
- Architecture: Linear projection → Reshape to [128×4×4] → Series of ResBlockUp modules with BatchNorm and ReLU → Final Conv2D with Tanh
- ResBlockUp: Improves gradient flow and increases spatial resolution progressively
- Output: Generated RGB image [3×32×32]

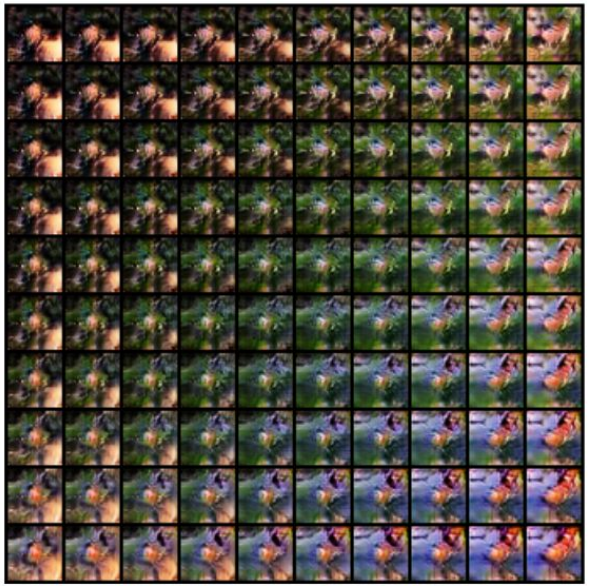


Discriminator:

- Input: RGB image [3×32×32]
- Architecture:ResBlockDown modules→ Standard ResBlocks→ReLU→Global Sum Pooling
- ResBlockDown: Feature extraction with downsampling (using DownSampleConv2D)
- Output: Scalar value indicating image authenticity (probability or Wasserstein score)

Types of GANs - Vanilla GAN

Vanilla GAN Samples

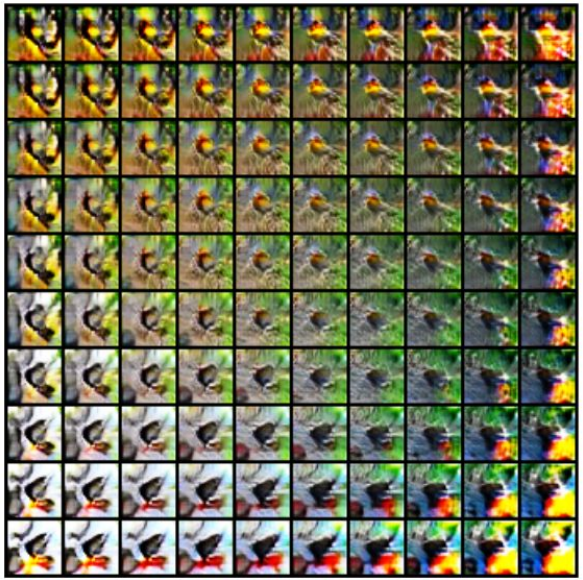


Vanilla GAN Latent Space Interpolations

GAN Variant	Loss Function	Key Features	FID Score
Vanilla GAN	Binary Cross-Entropy (BCE)	Original GAN formulation. Prone to training instability (mode collapse).	104.62

Types of GANs - LS-GAN

LS-GAN Samples

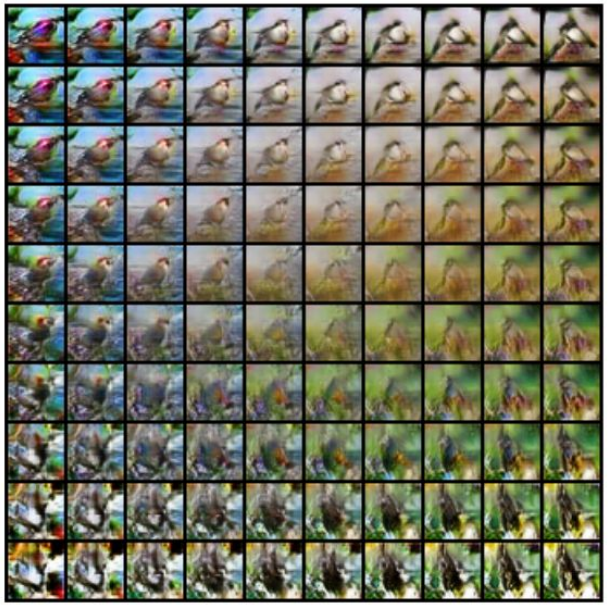


LS-GAN Latent Space Interpolations

GAN Variant	Loss Function	Key Features	FID Score
LSGAN	Least Squares Loss (MSE)	Uses Mean Squared Error instead of BCE. More stable training than Vanilla GAN.	52.48

Types of GANs - WGAN-GP

WGAN-GP Samples



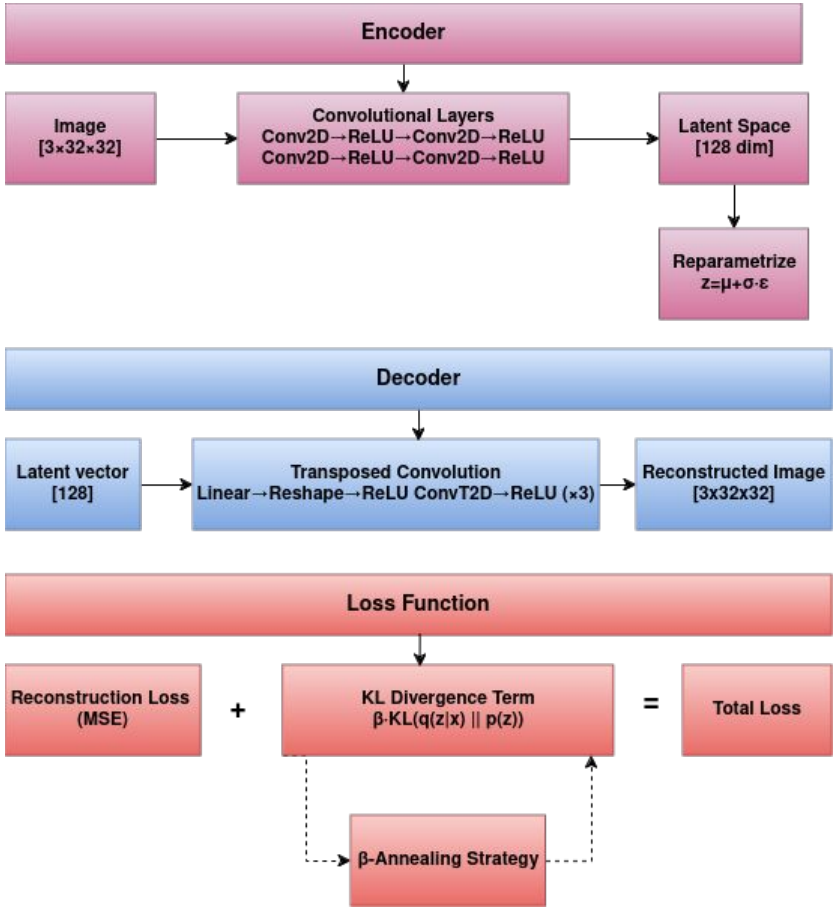
WGAN-GP Latent Space Interpolations

GAN Variant	Loss Function	Key Features	FID Score
WGAN-GP	Wasserstein Distance + Gradient Penalty (GP)	Uses Wasserstein distance for a more stable measure of the difference between distributions. Gradient Penalty enforces the 1-Lipschitz constraint.	33.07

Variational Autoencoder (VAEs) Architecture

Encoder:

- Input: RGB image [3×32×32]
- Architecture: Series of Conv2D→ReLU layers with progressive spatial dimension reduction
- Output: Mean vector (μ) and log standard deviation vector (log σ) of dimension 128
- Reparameterization trick: z = μ + σ * ε where ε ~ N(0,1)



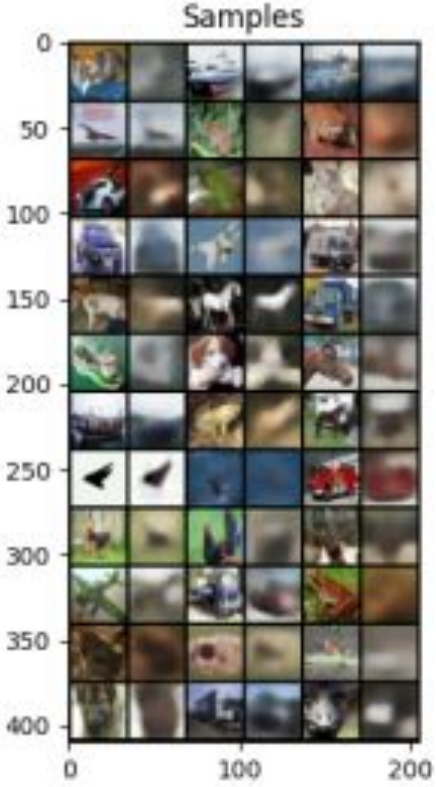
Decoder:

- Input: 128-dimensional latent vector (z)
- Architecture: Linear projection → Reshape → Series of transposed convolutions with ReLU
- Output: Reconstructed RGB image [3×32×32]

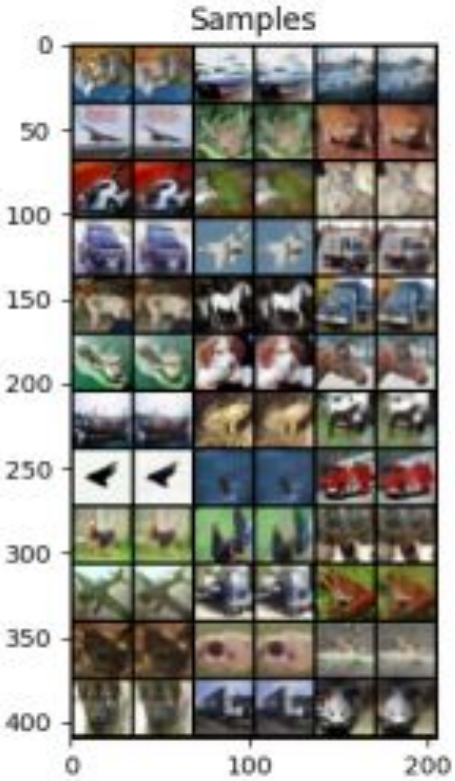
Loss Function:

- Reconstruction Loss: MSE between input and reconstructed image
- KL Divergence: Regularizes latent space toward standard normal distribution
- β-Annealing: Gradually increases β during training for better latent organization

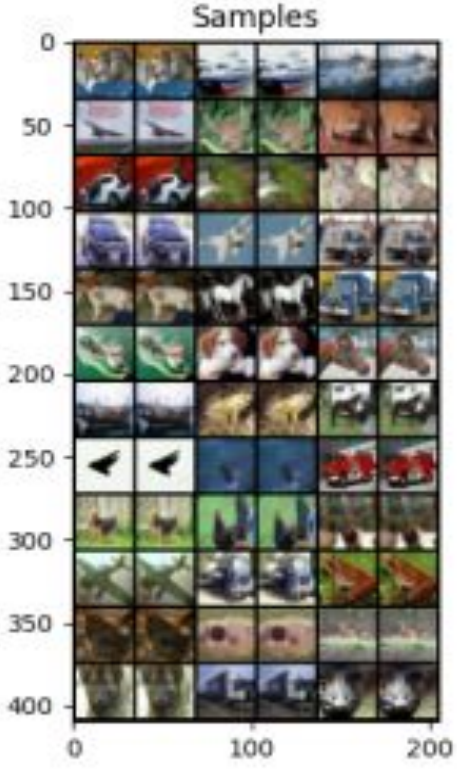
VAE Latent Space Exploration



Reconstruction sample: (size 16)



Reconstruction sample: (size 128)

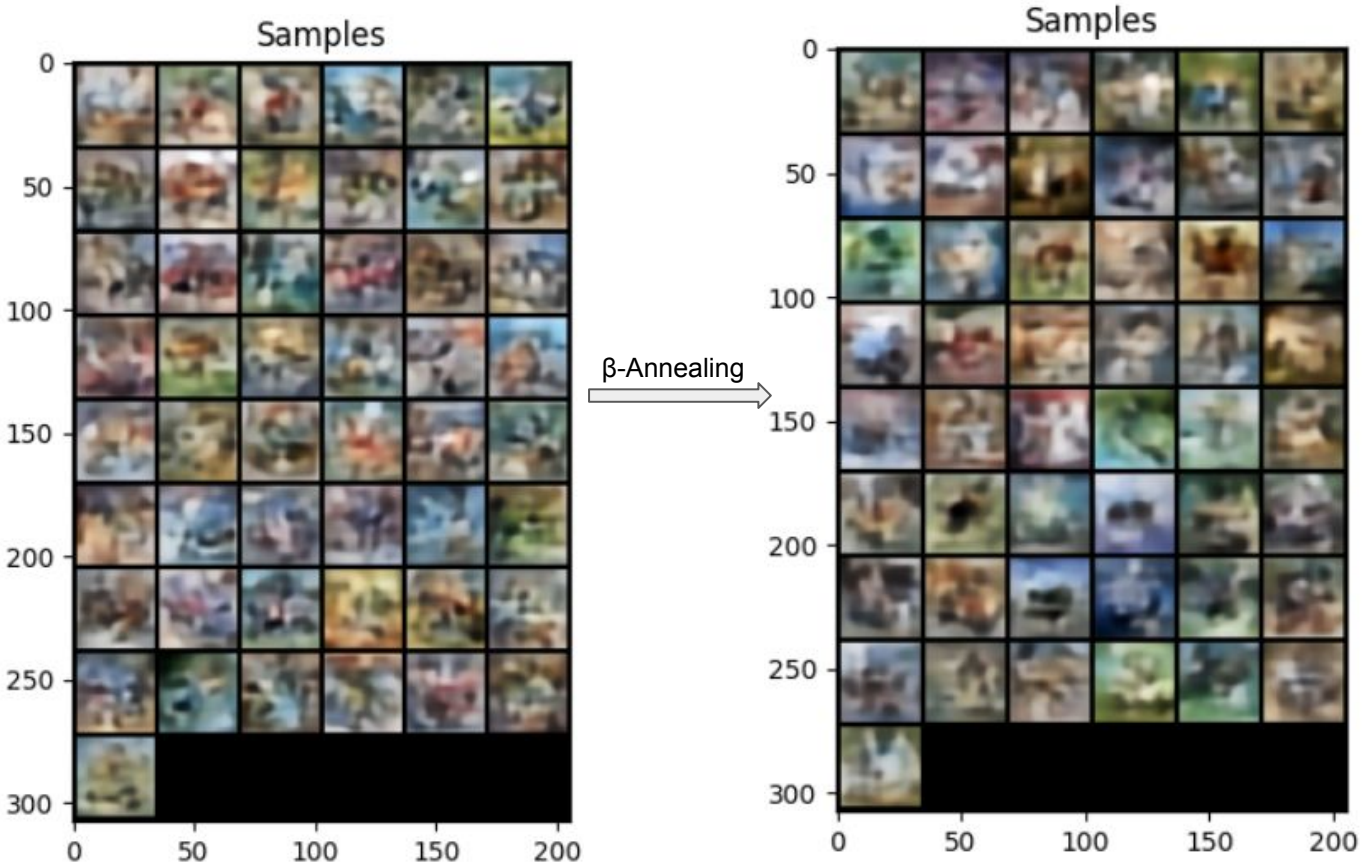


Reconstruction sample: (size 1024)

Reconstruction loss: ~200	Reconstruction loss: ~200 → ~75	Reconstruction loss: ~200 → ~75 → ~40

VAE Sample Generation and β-Annealing

To further improve the VAE's performance, we investigated the use of β-annealing. β controls the weight of the KL divergence term in the loss function. By gradually increasing β during training, we encourage the model to learn a more disentangled and well-structured latent space, which often leads to better sample quality.

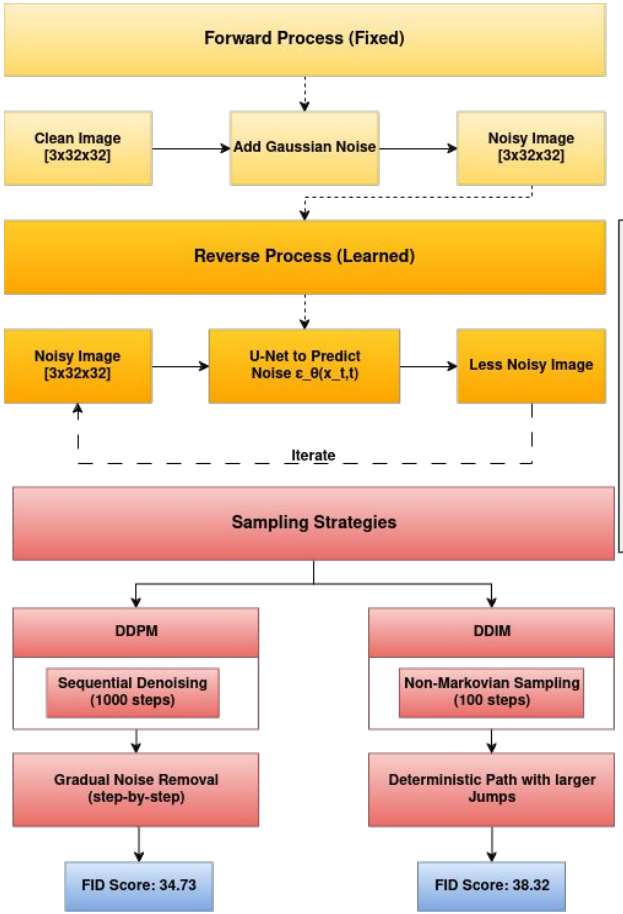


The left image shows samples generated with a fixed β value of 0.8. The right image shows samples generated after training with β-annealing (linearly increasing β from 0 to 0.8 over the first 20 epochs)

Diffusion Model Architecture with DDPM & DDIM Sampling

Forward Process (Fixed):

- Input: Clean RGB image [3×32×32]
- Process: Sequential addition of Gaussian noise over T timesteps (not learned)
- Output: Pure Gaussian noise after T steps



Reverse Process (Learned):

- Input: Noisy image at timestep t
- Architecture: U-Net predicts noise component ε_θ (x_t, t)
- Process: Iterative denoising by subtracting predicted noise
- Output: Progressively cleaner image, approaching the original distribution

Sampling Strategies:

- DDPM: Markovian process requiring ~1000 sequential denoising steps
- DDIM: Non-Markovian approach allowing larger jumps with only ~100 steps
- Trade-off: DDIM sacrifices some quality for 10× faster sampling

Diffusion Model Results

DDPM Samples

DDIM Samples

DDPM (left) achieves slightly better FID score (34.73) with 1000 sampling steps, while DDIM (right) maintains comparable quality (FID 38.32) with only 100 steps, demonstrating a significant efficiency improvement with minimal quality loss.