Deep Learning Project - Generative Models

Natalia Safiejko, Wojciech Grabias

May 2025

Contents

1 Introduction

2 Generative Models Overview

Generative models represent a fundamental class of machine learning algorithms that learn the underlying data distribution $p(\mathbf{x})$ to synthesize novel samples. These models have become indispensable across computer vision, natural language processing, and scientific computing. The three dominant paradigms—Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and Diffusion Models—each offer distinct approaches to capturing data distributions while addressing different trade-offs between sample quality, training stability, and computational efficiency.

2.1 Variational Autoencoders

Variational Autoencoders (VAEs) are a class of deep generative models that provide a principled framework for learning complex probability distributions over high-dimensional data, such as images or text. Unlike discriminative models which learn to predict labels given inputs, generative models like VAEs aim to model the joint distribution of data and latent variables, enabling tasks such as data generation, interpolation, and representation learning. VAEs are particularly useful in scenarios where labeled data is scarce, as they can leverage unsupervised learning to extract meaningful latent representations. [3]

2.1.1 Architecture of VAEs

VAEs consist of two main components:

- **Generative Model (Decoder)**: This defines a joint distribution

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z}) \tag{2.1}$$

  where $\mathbf{z}$ is a latent variable and $\mathbf{x}$ is the observed data. The decoder maps latent variables $\mathbf{z}$ to data space using a neural network.

- **Inference Model (Encoder)**: This approximates the intractable posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ with a simpler distribution $q_\phi(\mathbf{z}|\mathbf{x})$, parameterized by a neural network. The encoder maps observed data $\mathbf{x}$ to latent space.

2.1.2 Evidence Lower Bound (ELBO)

The learning objective for VAEs is the Evidence Lower Bound (ELBO), derived from variational inference:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \tag{2.2}$$

- The first term is the *reconstruction loss*, encouraging the model to generate data similar to the input.
- The second term is the Kullback-Leibler (KL) divergence, which regularizes the latent space by aligning the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ with the prior $p_\theta(\mathbf{z})$, typically a standard Gaussian $\mathcal{N}(0, \mathbf{I})$.

2.1.3 Reparameterization Trick

To enable gradient-based optimization, VAEs use the reparameterization trick:

$$\mathbf{z} = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \tag{2.3}$$

This trick allows backpropagation through stochastic nodes by expressing the latent variable $\mathbf{z}$ as a deterministic function of parameters μ_ϕ, σ_ϕ, and noise ϵ.

2.1.4 Training and Optimization

VAEs are trained using stochastic gradient descent (SGD) to maximize the ELBO. The training involves:

1. Sampling a minibatch of data $\mathbf{x}$
2. Encoding $\mathbf{x}$ into latent variables $\mathbf{z}$ using the encoder
3. Decoding $\mathbf{z}$ to reconstruct $\mathbf{x}$
4. Computing gradients of the ELBO with respect to θ and ϕ, and updating the parameters

2.1.5 Implementation

The implemented Variational Autoencoder (VAE) architecture maps images into a latent space of configurable dimensionality (default: 100). The encoder is composed of three convolutional layers with kernel size 4×4, stride 2, and padding 1, progressively increasing the number of feature maps from 64 to 256 while reducing the spatial dimensions to 8×8. Batch normalization and LeakyReLU activation ($\alpha = 0.2$) are applied after each convolution. The flattened feature map is linearly projected into two separate vectors representing the mean (μ) and log-variance ($\log \sigma^2$) of the latent distribution. Sampling from this distribution is performed via the reparameterization trick: $z = \mu + \sigma \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$. The decoder maps the latent variable z back through a fully connected layer, reshaping it into a $256 \times 8 \times 8$ tensor, and then applies three transposed convolutional layers with ReLU activations (except the final layer which uses Tanh), symmetrically reconstructing the input dimensions.

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) represent a breakthrough framework for generative modeling that employs an adversarial training process. Introduced by [1], GANs have revolutionized unsupervised learning by enabling the generation of highly realistic samples across various domains including images, audio, and text. GANs learn implicit data distributions through a game-theoretic approach.

2.2.1 Core Architecture

The GAN framework consists of two competing neural networks:

- **Generator (G)**: Maps random noise $\mathbf{z} \sim p_z(\mathbf{z})$ to data space $G(\mathbf{z})$, attempting to produce realistic samples
- **Discriminator (D)**: Acts as a classifier distinguishing between real data $\mathbf{x} \sim p_{data}(\mathbf{x})$ and generated samples $G(\mathbf{z})$

2.2.2 Adversarial Objective

The training process formulates a minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{2.4}$$

2.2.3 Training Dynamics

The optimization alternates between:

1: Initialize generator G and discriminator D
2: **for** number of training iterations **do**
3: Sample minibatch of noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$
4: Sample minibatch of real examples $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$
5: Update discriminator by ascending: $\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} [\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)}))]$
6: Update generator by descending: $\nabla_\phi \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\mathbf{z}^{(i)}))$
7: **end for**

2.2.4 Theoretical Foundations

At optimality, the generator recovers the data distribution when:

$$p_g = p_{data} \quad \text{and} \quad D^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} = \frac{1}{2} \tag{2.5}$$

2.2.5 Implementation

The generator consists of five transposed convolutional layers. It takes an input tensor of shape $(\text{latent\_dim}, 1, 1)$ and upsamples it through successive layers with kernel size 4, stride 2 (except the first layer with stride 1), and padding 1 (or 0 in the first layer). Feature map sizes evolve as follows: $(\text{ngf} \times 8) \to (\text{ngf} \times 4) \to (\text{ngf} \times 2) \to \text{ngf} \to 3$, corresponding to spatial resolutions $4 \times 4 \to 8 \times 8 \to 16 \times 16 \to 32 \times 32 \to 64 \times 64$. Each hidden layer is followed by BatchNorm and ReLU activation, with the output layer using `Tanh`.

The discriminator mirrors this structure in reverse using standard convolutions. It takes a $3 \times 64 \times 64$ image and reduces it through five convolutional layers with kernel size 4, stride 2 (except the last layer with stride 1), and padding 1 (or 0 in the last layer). The feature map progression is $3 \to \text{ndf} \to (\text{ndf} \times 2) \to (\text{ndf} \times 4) \to (\text{ndf} \times 8) \to 1$. Each intermediate layer uses LeakyReLU with slope 0.2 and, except the first, BatchNorm. The output is a scalar after flattening.

2.3 Diffusion Models

Diffusion models are a class of latent variable models inspired by non-equilibrium thermodynamics, designed to generate high-quality samples through an iterative denoising process. These models operate through two key phases:

- **Forward process:** A fixed Markov chain that gradually adds Gaussian noise to the data over T timesteps:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \quad q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{2.6}$$

  where β_t is a predefined noise schedule.

- **Reverse process:** A learned Markov chain that progressively denoises the data:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{2.7}$$

  where θ are learned parameters.

A key innovation is the connection to denoising score matching [4], where the model learns to predict the noise $\boldsymbol{\epsilon}$ at each timestep through a simplified training objective:

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\|^2 \right] \tag{2.8}$$

Diffusion models offer several advantages:

- **Stable training** compared to adversarial approaches
- **High sample quality** with FID scores competitive with state-of-the-art methods
- **Progressive generation** enabling coarse-to-fine synthesis
- **Theoretical connections** to score matching and variational inference

While computationally intensive due to iterative sampling, diffusion models provide a powerful framework for generative tasks, particularly in image synthesis where they demonstrate exceptional performance on benchmarks like CIFAR-10 and LSUN [2].

CatDiffusion Architecture. The `CatDiffusion` model, in the report referred to as Diffusion model or UNet. integrates a UNet-based denoiser and a DDPM scheduler to perform diffusion-based image generation. The default timesteps count is 500. The UNet is constructed with four resolution levels, each consisting of 4 layers per block, and channel dimensions that progressively increase from 32 to 256. It uses four `DownBlock2D` layers for downsampling and four corresponding `UpBlock2D` layers for upsampling. The DDPM scheduler employs the `squaredcos_cap_v2` beta schedule and is configured to predict the added noise (`epsilon`) across a user-defined number of timesteps.

3 Dataset and Preprocessing

For the purpose of training generative models, a publicly available dataset from Kaggle titled *Cat Dataset*[1] was used. This dataset contains high-quality RGB images of cats in various poses, environments, and lighting conditions. The dataset is not labeled for classification tasks, which makes it particularly suitable for unsupervised learning problems such as image generation.

The dataset consists of a collection of PNG images located in a single directory, without subfolder-based class separation. Each image is loaded and converted to the RGB color space using the Python Imaging Library (PIL). As we are training a generative model rather than a classifier, all images are assigned a dummy label of 0 to comply with the expected format of PyTorch's `Dataset` interface.

Prior to feeding the images into the model, we applied a series of preprocessing transformations using the `torchvision.transforms` module. These transformations include:

- **Resizing:** Each image is resized to a square shape of dimensions $H \times W$ specified by the configuration file (i.e. 64×64).

[1] `https://www.kaggle.com/datasets/borhanitrash/cat-dataset`

- **Center cropping:** After resizing, a center crop is applied to ensure uniformity and to remove potential borders or padding.
- **Normalization:** The pixel values are normalized to the range $[-1, 1]$ by applying the transformation $(x - 0.5)/0.5$ to each RGB channel. This normalization is consistent with common practices in training GANs and mandatory in case of diffusion models using the architecture of UNet.
- **Conversion to tensor:** Images are converted to PyTorch tensors, which are compatible with GPU acceleration.

The dataset is loaded using a custom PyTorch `Dataset` class and wrapped in a `DataLoader` to enable efficient mini-batch processing with shuffling and parallel data loading. This setup facilitates robust and scalable training of deep generative models.

4 Image Generation

4.1 Technical aspects of image generation

The image generation process differs across GANs, VAEs, and diffusion models, each utilizing distinct sampling strategies rooted in their underlying architectures:

- **GANs:** After training, the generator network G maps a noise vector $\mathbf{z} \sim p_z(\mathbf{z})$, typically drawn from a standard normal distribution, to the data space: $\hat{\mathbf{x}} = G(\mathbf{z})$. Since GANs are non-probabilistic, image generation is performed in a single forward pass through the generator, resulting in high-speed sampling.
- **VAEs:** To generate new samples, a latent vector $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ is drawn from the prior distribution. This vector is passed through the decoder network to produce an image: $\hat{\mathbf{x}} = p_\theta(\mathbf{x}|\mathbf{z})$. The decoder outputs either pixel values directly or parameters of a probabilistic distribution.
- **Diffusion Models (i.e., UNet-based):** Image generation involves reversing the forward diffusion process. Starting from pure Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$, the model applies a sequence of denoising steps using the learned reverse transitions $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ parameterized by a neural network such as a UNet. The final output $\hat{\mathbf{x}}_0$ is a clean sample drawn from the learned data distribution.

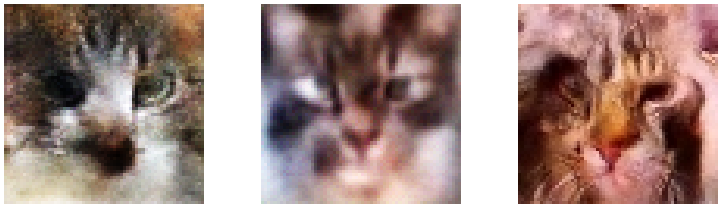
4.2 Initial generations

Sample images from post-training represent cat–like features and are a confirmation that the utilized architectures are capable of generating new images. Differences between the images' characteristics are further discussed in Section 6.

5 Training efficiency considerations

As the task of training generative models is both time and resource consuming, few adjustments were used in order to allow results analysis in a reasonable time and cost frame:

- **Model architectures**: In case of GANs, the DCGAN was used as it is on the lower–expenditure side of this model family. In case of the diffusion models, UNet–based model was chosen for the same exact reasons,



DCGAN VAE Diffusion model

Figure 1: Three sample images generated by three different architectures

- **Mixed precision**: Because of the particularly expensive training of the diffusion model, mixed precision training was utilised using `torch.amp`,
- **Hardware**: Cloud-based solutions were utilised, including Google Collab notebooks, as well as GCP workbenches.

6 Hyperparameter Tuning and Evaluation

6.1 FID Score

To quantitatively assess the performance of the generative models,the Fréchet Inception Distance (FID) was employed as the primary evaluation metric. The FID score is widely used in generative modeling tasks to measure the similarity between the distribution of generated images and that of real images. A lower FID score indicates that the generated images are more similar to real ones in terms of high-level features extracted by a pretrained classifier.

In the evaluation pipeline, the Inception v3 network pretrained on ImageNet was utilized to extract features from both real and generated images. The final fully connected layer of the Inception model was replaced with an identity mapping to output feature vectors instead of classification logits. For real images, these features were computed once and cached to avoid redundant computation.

The generative models (GAN, VAE, and diffusion models) were used to synthesize a fixed number of fake images (typically 1000) by sampling from their respective latent spaces. These images were post-processed to match the real image distribution and fed into the same Inception model to extract feature vectors. The mean and covariance of these features were computed for both real and fake images, denoted as (μ_r, Σ_r) and (μ_f, Σ_f), respectively.

The FID score is then calculated as:

$$\text{FID} = \|\mu_r - \mu_f\|^2 + \text{Tr}(\Sigma_r + \Sigma_f - 2(\Sigma_r\Sigma_f)^{1/2})$$

where $\|\cdot\|$ denotes the Euclidean norm and Tr denotes the trace of a matrix. This formulation effectively measures the distance between two multivariate Gaussian distributions fitted to the Inception features of real and generated images.

By using this unified evaluation strategy, a fair and consistent comparison across different generative models was ensured.

6.2 Training hyperparameters

Training-related hyperparameters are crucial for controlling the learning process and ensuring that the model converges to an optimal solution. These include the learning rate, batch size, and the choice of loss function. Each of

these parameters affects the speed, stability, and final performance of the model. Training-related hyperparameters are mainly evaluated quantitively, mainly due to number of models and moderately low impact of the parameters on what (in terms of structure and features, not quality) the images look like.

6.2.1 Default values

If not stated otherwise in the experiment, the default parameters of the training process for a given architecture are listed in Table 1. Different batch size for the diffusion model was mainly caused by the computational limitations.

Table 1: Default hyperparameters values for different architectures

	Epochs	Learning Rate	Batch Size	β_1
DCGAN	10	0.0001	64	0.9
VAE	10	0.0001	64	0.9
Diffusion model	10	0.0001	128	0.9

Doubling its size with comparison with DCGAN and VAE allowed nearly 1.5x faster model training and in fact denoted the best results FID-wise.

6.2.2 Batch size

Smaller batches introduce higher gradient noise, which can help escape sharp minima and improve generalization, especially in generative models where the training signal is inherently noisy. Conversely, larger batches reduce gradient noise but may lead to poorer generalization and convergence to suboptimal local minima.

Table 2: Performance comparison across different batch sizes

Architecture	**batch=64**	**batch=128**	**batch=256**
DCGAN	346.018	357.736	370.288
VAE	120.807	133.822	152.050
Diffusion model	196.418	164.738	182.891

The comparative analysis in Table 2 reveals several important patterns about batch size selection for generative models.

- **Consistent Advantage of Smaller Batches**: VAE and DCGAN architectures demonstrate superior performance with smaller batch sizes (64). The performance degradation with larger batches could stem from reduced gradient diversity during updates. Diffusion model denotes a different pattern: it performs best at the batch size of 128.

- **Architecture-Specific Sensitivity**: While VAE and DCGAN architectures benefit from smaller batches, all of the architectures demonstrate similar sensitivity to batch size changes. Since the generalization–gradient noise tradeoff affects the training dynamics universally, even structurally distinct models like VAEs, DCGANs, and UNet-based diffusion models tend to exhibit similar sensitivity patterns with respect to batch size.

- **Computational Trade-offs**: The observed patterns in VAE and DCGAN present a practical trade-off between computational efficiency (favored by larger batches) and model performance (favored by smaller batches). This tension should inform resource allocation decisions during model development. For diffusion

model, the trade-off is further exploited - optimal training process comes at more computationally-efficient setup.

Figure 2, 4 and 3 show the loss trajectories across 10 training epochs.

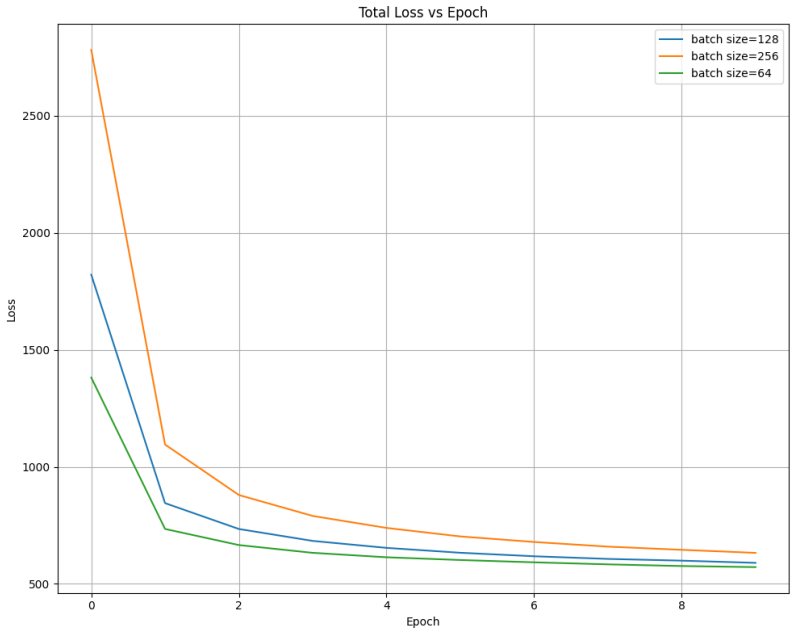


Figure 2: Total loss curve for different batch sizes during VAE training.

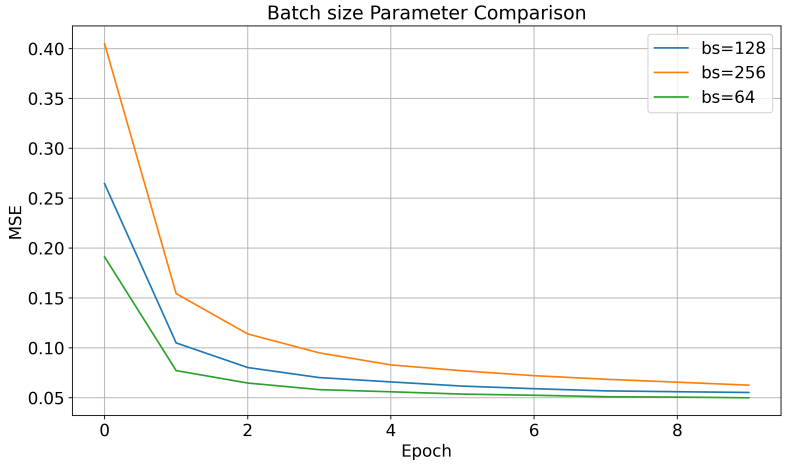


Figure 3: Total loss curve for different batch sizes during diffusion model training.

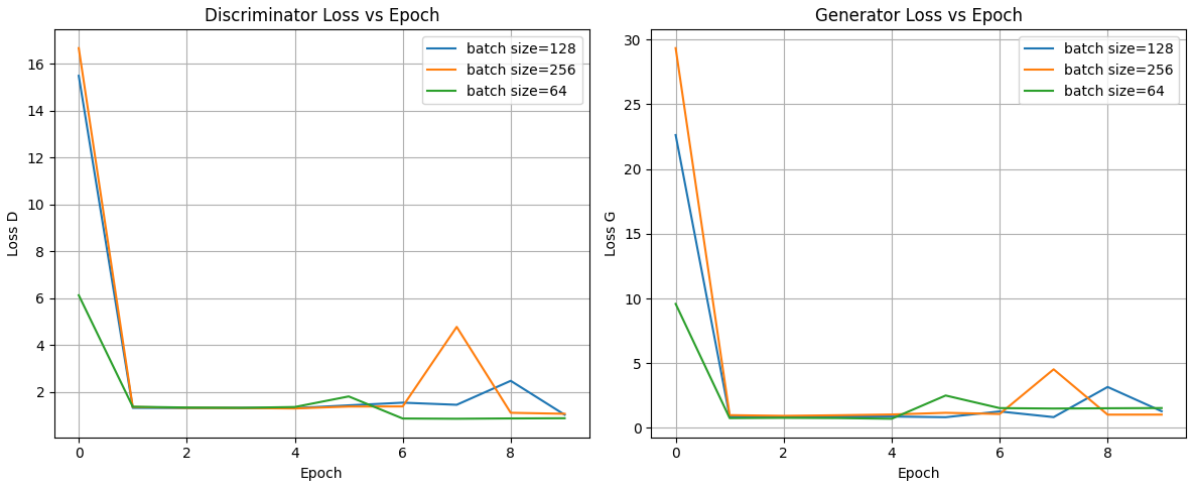


Figure 4: Discriminator and generator loss curves for different batch sizes during DCGAN training.

It can be observed that the batch size affects the learning dynamics:

- The smaller batch size (64) demonstrates more stable convergence behavior, with smoother loss curves across epochs
- Larger batch sizes (256) may lead to noisier updates due to the reduced number of parameter updates per epoch
- For the diffusion model, the final MSE reached by the model does not fully correlate to the FID score obtained magnifing the importance of the metric

6.2.3 Learning Rate

Table 3: Performance comparison across different learning rates

Architecture	**0.0001**	**0.0002**	**0.0005**	**0.001**	**0.005**	**0.01**
DCGAN	38.589	75.355	83.213	79.890	50.029	412.715
VAE	109.137	120.807	306.088	274.463	347.216	347.190
Diffusion model	147.226	261.701	230.554	298.246	340.721	NaN

The learning rate analysis in Table 3 reveals distinct patterns in how different architectures respond to learning rate variations:

- **Optimal Learning Rate Ranges**:
  - DCGAN performs best with very low learning rates (0.0001), showing degraded performance at higher values
  - VAE demonstrates good stability at moderate rates (0.0001-0.0002) but deteriorates sharply beyond 0.0005
  - Diffusion model demonstrates a monotonic decrease with higher learning rates, which is why a test with an even further-decreased parameter was conducted (lr=0.00005), but the FID score obtained was 172.739, opposing the trend.

- **Failure at High Rates**: Architectures show severe performance degradation at the highest learning rate (0.01), suggesting this range is generally unsuitable for generative model training. That was evidently apparent in case of the diffusion model, where the training process resulted in an overflow and complete divergence of the training.
- **Architecture-Specific Sensitivities**:
  - DCGAN maintains relatively stable performance across a wider range (0.0001-0.005) before collapsing
  - VAE shows a sharper performance cliff after its optimal range
  - Diffusion model presents stable learning curve for essentially all learning rate values. It also demonstrates closest final results, with the biggest difference being the MSE value after 1 to 2 epochs.
- **Non-Monotonic Behavior**: The DCGAN's unexpected improvement at lr=0.005 compared to neighboring values suggests complex optimization dynamics that warrant further investigation.

Figures 5 and 6 illustrate the evolution of discriminator and generator losses over the course of 10 epochs.

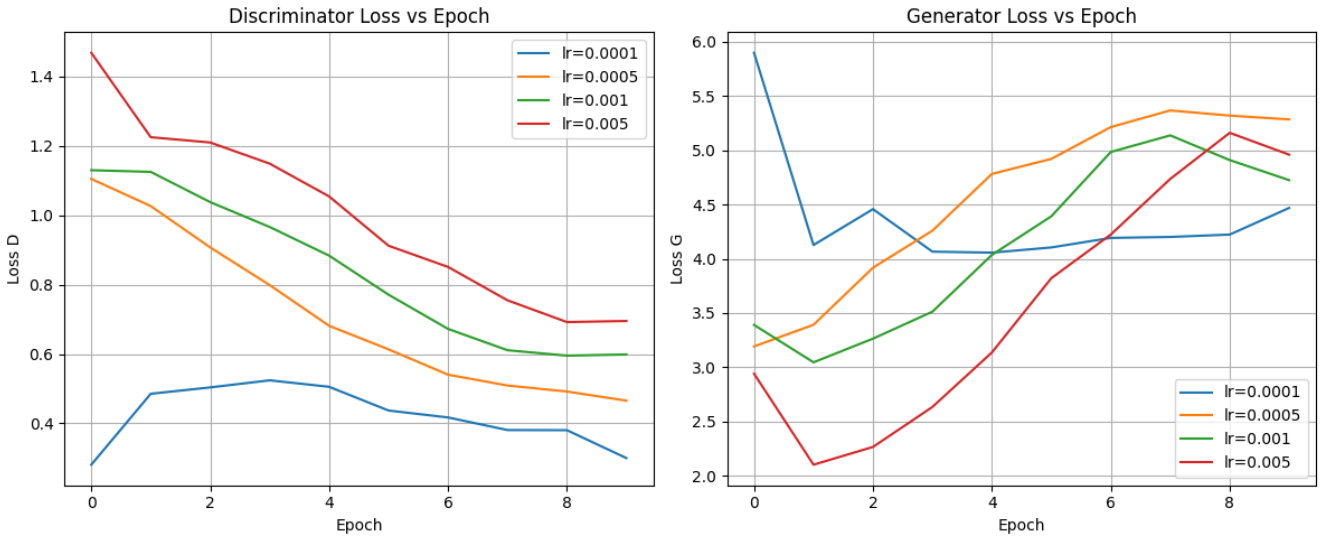


Figure 5: Discriminator and generator loss curves during DCGAN training for different learning rates.

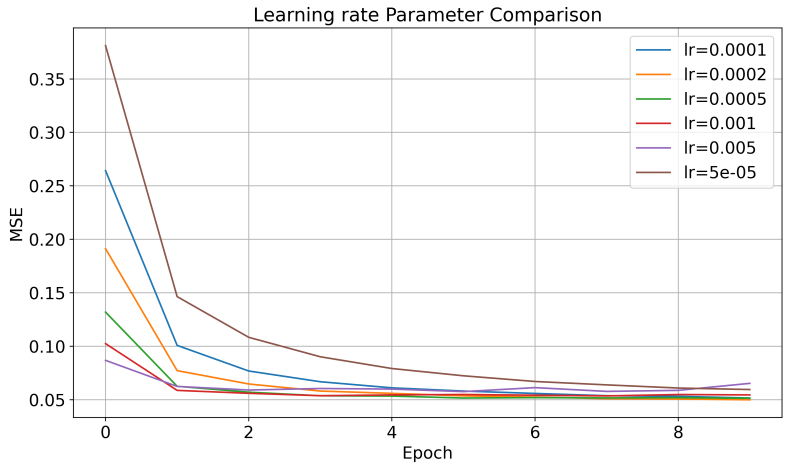


Figure 6: Total loss curve for different learning rates during diffusion model training.

The plots demonstrate that the learning rate plays a significant role in the stability and convergence behavior of both the generator and discriminator. In general:

- Extremely low learning rates result in slow convergence and minimal progress in both networks.
- Very high learning rates can lead to unstable training and oscillating loss curves, especially in the early stages.
- Moderate learning rates tend to produce more stable and gradual learning, with balanced generator and discriminator behavior.

6.2.4 Beta value

Table 4: Performance comparison across different β_1 values

Architecture	$\beta_1 = 0$	$\beta_1 = 0.3$	$\beta_1 = 0.5$	$\beta_1 = 0.7$
DCGAN	38.09	48.81	38.59	55.51
VAE	111.22	123.15	109.14	127.23
UNET	278.665	187.994	237.30	258.43

The investigation of the β_1 parameter in Adam optimization reveals several key insights:

- **Optimal Values**:
  - For VAE and DCGAN, both architectures perform best at either $\beta_1 = 0$ or $\beta_1 = 0.5$
  - DCGAN shows minimal difference between $\beta_1 = 0$ (38.09) and $\beta_1 = 0.5$ (38.59)
  - VAE demonstrates clearer preference for $\beta_1 = 0.5$
  - Diffusion UNET denotes best performance for $\beta_1 = 0.3$, though its performance is still inferior to the one for the default parameter value ($\beta_1 = 0.9$)
- **Architecture-Specific Sensitivities**:
  - DCGAN maintains relatively stable performance across more β_1 values
  - VAE shows more pronounced variation across the tested range, yet even drastic changes from default of $\beta_1 = 0.9$ to $\beta_1 = 0.0$ does not seem to affect the training drastically (for both VAE and DCGAN)
  - Diffusion model is the most sensitive to the parameter change, denoting a performance decrease of over 100 throughout the experiments. That is most likely due to the model's architectural complexity that requires parameters to be well-tuned for optimal training,

The findings support using β_1 values in the $[0, 0.5]$ range, with architecture-specific fine-tuning potentially yielding additional benefits.

6.2.5 Loss function

It was analyzed how different loss functions affect training dynamics and output quality by comparing three approaches: the DCGAN (using Standard GAN loss with architectural enhancements), Least Squares GAN (LSGAN), and Spectral Normalization GAN (SNGAN).

DCGAN with Standard GAN Loss The DCGAN framework employs the original GAN's binary cross-entropy loss but incorporates critical architectural improvements:

$$\mathcal{L}_D^{\text{DCGAN}} = -\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] - \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))] \tag{6.1}$$

$$\mathcal{L}_G^{\text{DCGAN}} = -\mathbb{E}_{z\sim p_z}[\log D(G(z))] \tag{6.2}$$

Key architectural modifications include:

- Strided convolutions instead of pooling layers
- Batch normalization in both generator and discriminator
- LeakyReLU activations in the discriminator

While these changes improve stability over vanilla SGAN, the fundamental limitations of binary cross-entropy loss remain.

Least Squares GAN (LSGAN) LSGAN replaces the binary cross-entropy with a least-squares objective:

$$\mathcal{L}_D^{\text{LSGAN}} = \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}[(D(x) - 1)^2] + \frac{1}{2}\mathbb{E}_{z\sim p_z}[(D(G(z)))^2] \tag{6.3}$$

$$\mathcal{L}_G^{\text{LSGAN}} = \frac{1}{2}\mathbb{E}_{z\sim p_z}[(D(G(z)) - 1)^2] \tag{6.4}$$

This formulation provides more stable gradients by:

- Avoiding saturation in early training
- Penalizing samples far from decision boundaries
- Maintaining consistent gradient magnitudes

Spectral Normalization GAN (SNGAN) SNGAN introduces two key innovations:

$$\mathcal{L}_D^{\text{SNGAN}} = -\mathbb{E}_{x\sim p_{\text{data}}}[\min(0, -1 + D(x))] - \mathbb{E}_{z\sim p_z}[\min(0, -1 - D(G(z)))] \tag{6.5}$$

$$\mathcal{L}_G^{\text{SNGAN}} = -\mathbb{E}_{z\sim p_z}[D(G(z))] \tag{6.6}$$

The combination of hinge loss and spectral normalization provides:

- Enforced Lipschitz continuity via spectral normalization
- More robust gradient flow through hinge loss
- Better prevention of mode collapse
- Architecture-agnostic stabilization

Table **??** demonstrates that while DCGAN's architectural improvements help (FID = 44.180), both LSGAN (35.731) and SNGAN (30.976) achieve better performance through their specialized loss formulations. Notably, SNGAN's combination of hinge loss and spectral normalization yields the best results, showing that loss function design and stabilization techniques can outperform architectural enhancements alone.

Table 5: Performance comparison across different loss functions

Method	**FID Score**
DCGAN (SGAN loss + architecture)	44.180
LSGAN	35.731
SNGAN	30.976

6.2.6 Qualitative results and their correlation with FID score

First and probably most important result in terms of image generation is that models that achieved the FID score of over or around 300, meant in all cases a disability of the model to generate new images. Generated images were either one-colored or completely noisy. Thankfully for each model architecture, a reasonable FID score was achieved, magnifying the importance of parameter tuning for image generation.
The correlation between FID score and the quality of the images was consistently negative (lower FID $\Rightarrow$ higher quality), but the relationship was not free of caveats. Figure 7 demonstrates the relationships between generated images and their FID scores. In all cases lower FID means more cat-like images with higher level of details, as well as bigger depth in terms of colors. What is more, distinct differences in the way the images are generated are very clear. The diffusion model is characterized by very vibrant colors and smooth patterns (not always cat-like) but always attempting high-resolution generation. VAE on the other hand, generates very fuzzy dream-like cats The resolution is blurry but all of the images present at least cat-like features, unlike the diffusion model. DCGAN is able to generate images of the highest quality, maintaining both cat-like features and a variety of their positions, colors and facial structures. Another interesting phenomenon can be shown using Figure 8. Despite quite decent generations, the FID score is quite significantly lower than other runs achieved by the architecture. What can be seen, however, is the model's repetition of cats' features throughout generations. Most of them have the same face and almost all have the exact same eyes in the same position. That example unravels the variability of generations factor in the score examined.

6.3 Architecture parameters

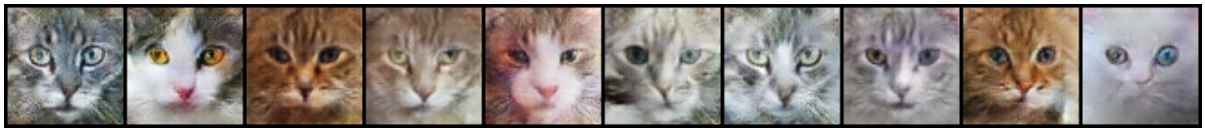
6.3.1 Latent dimension

Table 6: Performance comparison across different latent dimension sizes

Architecture	**50-dim**	**100-dim**	**200-dim**
DCGAN	39.454	38.589	41.928
VAE	113.689	109.137	104.949

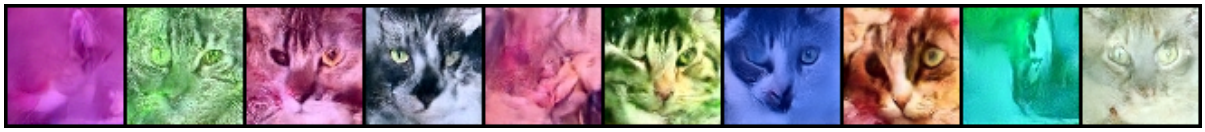
Analysis of latent dimension size variation in Table 6 reveals several key insights:

- **Architecture-Specific Responses**:
  - **DCGAN** shows minimal sensitivity to dimension size, with optimal performance at 100 dimensions
  - **VAE** demonstrates consistent improvement as dimension increases, suggesting benefits from more expressive latent spaces
- **Performance Patterns**:

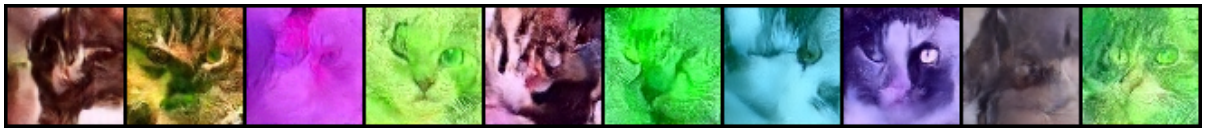


(a) DCGAN, FID=38.589

(b) DCGAN, FID=83.213



(c) Diffusion, FID=147.589



(d) Diffusion, FID=187.994

(e) VAE, FID=109.137

(f) VAE, FID=127.589

Figure 7: Comparison of images generated using DCGAN, Diffusion, and VAE models across different FID scores.

  - All architectures maintain stable performance across dimension changes
  - Relative performance differences between architectures remain consistent regardless of dimension size

- **Practical Implications**:
  - DCGAN users may prefer smaller latent spaces (100-dim) for efficiency
  - VAE practitioners might benefit from larger dimensions (200-dim) when computationally feasible
  - The modest variation suggests latent dimension is less critical than other hyperparameters

Figure 9 perfectly demonstrates the impact of latent dimension sizes on the images generated. In case of VAE, the blur patches are much more divided and split throughout the image for higher dimensions, whereas for the lower dimensions, the blur seems to be consistent and more uniform. The difference is less noticeable in case of DCGAN, yet the 200 dimension architecture appears to be slightly more pixelated.

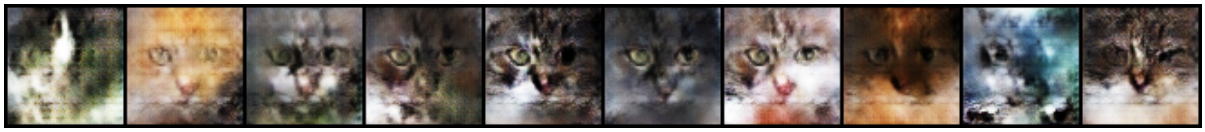
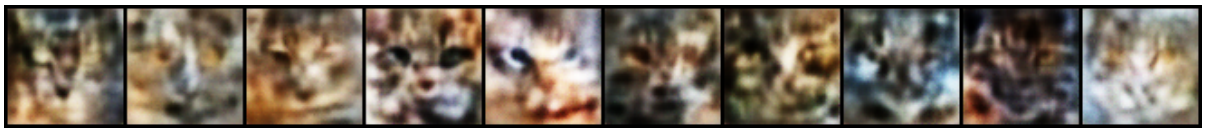


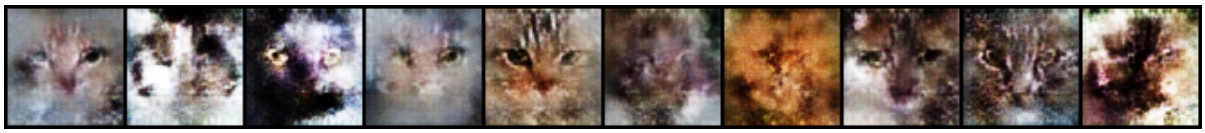
Figure 8: DCGAN, FID=102.415 in spite of decent image generation, model produces very similar images feature-wise (eyes, head position etc.)

(a) VAE, 50-dim



(b) VAE, 200-dim

(c) DCGAN, 50-dim



(d) DCGAN, 200-dim

Figure 9: Comparison of images generated using DCGAN, and VAE models across different latent dimension sizes.

6.3.2 Effects of Filter Counts in GAN Architectures

Table 7: Performance metrics for different filter counts in DCGAN architecture

Parameter	**32 filters**	**64 filters**	**128 filters**
ndf (Discriminator)	48.83	38.59	39.19
ngf (Generator)	53.02	38.59	53.66

Filter counts of both the discriminator and generator of the DCGAN architecture was modified. what is important, filters were **only changed within one of the two with the default value of 64**. For example, the value 48.83 in Table 7 refers to the architecture with ndf=32 and ngf=64. The investigation of network capacity through filter counts (ndf/ngf) reveals several important design considerations:

- **Optimal Middle Ground**:
  - Both ndf and ngf achieve best performance at 64 filters
  - The shared optimum suggests balanced capacity between generator and discriminator is crucial
- **Asymmetric Responses**:

  - Discriminator (ndf) shows more stable performance across sizes
  - Generator (ngf) exhibits sharper degradation when moving away from 64 filters
- **Performance Trade-offs**:
  - 32 filters: Under-capacity leads to +26% worse performance than optimal
  - 128 filters: Over-parameterization shows diminishing returns

These findings suggest that:

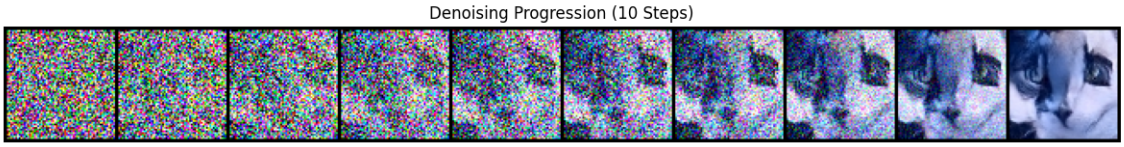
- The 64-filter convention common in DCGAN implementations is well-justified
- Generator components may require more careful tuning than discriminators
- Extreme values (very low or very high filter counts) should generally be avoided

6.3.3 Timesteps in UNet diffusion model

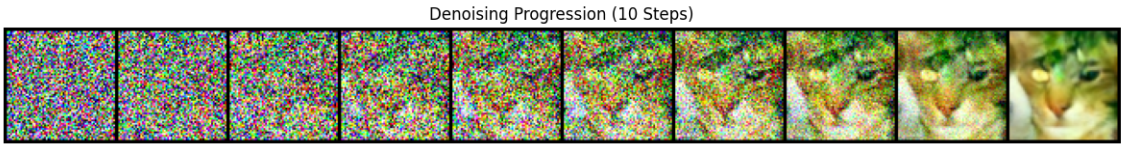
n the context of diffusion models, timesteps play a crucial role in guiding the denoising process across multiple stages. At each discrete timestep t, the U-Net takes as input a noisy image x_t and a timestep embedding, and predicts either the noise component or the denoised image. These timestep embeddings are typically encoded using sinusoidal positional encodings or learned embeddings to provide the model with a sense of temporal order. Table 8 shows little to no correlation between the timesteps and the FID score, with a slight edge for the lower timesteps. That is most likely due to limited training time and architectural resources, therefore it is easier for the model to train using little timesteps. Figure 10 demonstrates the denoising process among different timesteps counts. Each count is split into 10 equal parts (with the most left image being the initial noise, the most right being the output of the model). What can be seen is that lower timesteps count models are able to show core features of the image (such as contours, eyes etc.) relatively faster than the higher timesteps count ones.

Table 8: Performance scores at different timesteps count

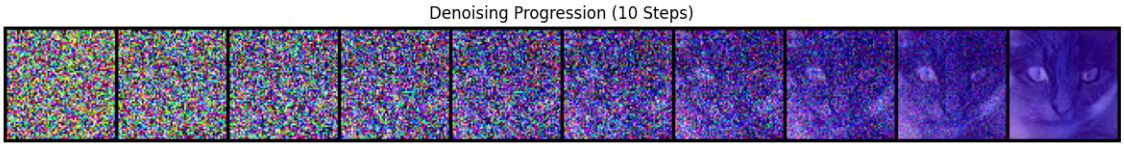
Timesteps	**100**	**250**	**500**	**750**	**1000**	**1500**
Score	83.648	171 280	140.617	231.182	200.919	177.716



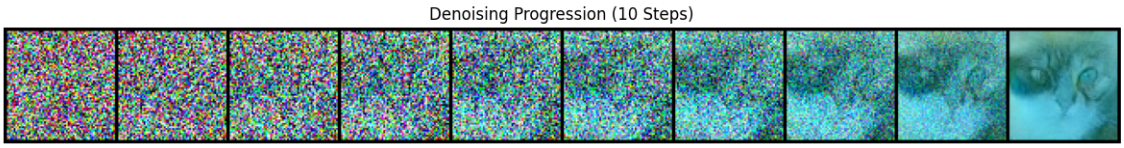
(a) 100 timesteps



(b) 500 timesteps



(c) 1000 timesteps



(d) 1500 timesteps

Figure 10: Comparison of images' denoising process for different timestep counts in the diffusion model

7 Latent Space Exploration and Interpolation

To further evaluate the learned latent space of the SNGAN model, a latent space interpolation experiment was performed. Two generated samples were randomly selected from the model and their corresponding latent vectors $\mathbf{z}_1$ and $\mathbf{z}_2$ were extracted.

Then 8 intermediate latent vectors were computed by linear interpolation:

$$\mathbf{z}_i = (1 - \alpha_i) \cdot \mathbf{z}_1 + \alpha_i \cdot \mathbf{z}_2, \quad \alpha_i \in \left\{ \frac{1}{9}, \frac{2}{9}, \ldots, \frac{8}{9} \right\}$$

This resulted in a total of 10 latent vectors. The corresponding images were generated and are visualized in Figure 11.

Figure 11: Latent space interpolation between two SNGAN-generated cat images.

Figure 12: LSGAN image generations for Dogs vs. Cats dataset

The interpolation reveals smooth and coherent transitions in both pose and texture, suggesting that the generator has learned a reasonably continuous and structured latent space. The shape and position of facial features evolve gradually across the sequence, with no abrupt artifacts or discontinuities.

Quantitatively, the model achieved an FID score of **30.976**, indicating moderate fidelity and diversity in the generated samples. While not on par with state-of-the-art GAN architectures, the results demonstrate that the SNGAN model captures meaningful latent semantics suitable for controllable generation and interpolation.

8 Cats and Dogs

To further evaluate generative model performance, we trained a **Least-Squares GAN (LSGAN)** on the popular Kaggle Dogs vs. Cats dataset. Best-performing DCGAN hyperparameters were used (BS=64, β_1=0.5, LR=0.0001, latent dim=100, 64 filters) alongside 30 epochs to further enhance the training. The key purpose of this analysis was to verify what features (cats, dogs or maybe both) would appear in generated images. The results are displayed in Figure 12. The quality of the images is significantly lower than in the case of the cats only. No evident facial features are distinguishable. What can be seen are certain fur and head-shape features, with seldom appearance of cat ears. Training a model to generate both cats and dogs simultaneously is considerably more challenging than focusing on a single class like cats. This is due to the increased diversity in appearance, structure, and texture between the two animal types, which the generator must capture within the same latent space. The model may struggle to converge on consistent features, resulting in more ambiguous and less realistic outputs.

9 Conclusions

This project presented a comprehensive comparative study of three prominent families of generative models—Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and Diffusion Models—with a particular focus on their effectiveness in generating realistic images of cats. The evaluation was conducted on a consistent dataset and controlled experimental setup, enabling a fair comparison of performance, architectural choices, and training dynamics.

Model Performance

All three models demonstrated the ability to generate cat-like images, with varying degrees of quality, diversity, and realism:

- **DCGAN** achieved the best overall FID scores and image quality, producing sharp, varied samples with distinct cat features. Its performance was also the most consistent across hyperparameter variations, making it a strong baseline for generative image tasks.
- **VAE** generated blurrier but semantically coherent images. While its output lacked fine detail, the model showed robustness in capturing core image structure, benefiting from the probabilistic latent space and regularized training.
- **Diffusion Models** offered visually compelling and high-resolution outputs, particularly in terms of color richness and structural complexity. However, their computational cost and sensitivity to hyperparameter tuning (e.g., timestep count) make them less practical for resource-constrained applications.

Hyperparameter Sensitivity

The investigation of training hyperparameters revealed:

- Smaller batch sizes tend to benefit most models by improving generalization, although the optimal batch size may vary by architecture.
- Learning rate and β_1 parameter tuning significantly impact stability and convergence, with lower values generally promoting smoother training.
- DCGAN and VAE architectures benefit from increased latent dimensionality to a moderate extent, whereas diffusion models are less sensitive to such changes.
- The choice of loss function and architectural components (e.g., filter count) has a nontrivial effect on performance, particularly for adversarial models.

Evaluation Metrics

The Fréchet Inception Distance (FID) was effective in quantifying generation quality, though qualitative inspections revealed nuances—such as sample diversity and fidelity—that FID alone cannot fully capture. Repeated features across generations, as observed in DCGAN runs, indicated potential overfitting or lack of diversity despite strong FID scores.

Key Takeaways

- There is no universally best generative model; performance depends on task requirements, resource constraints, and training duration.
- DCGAN offers an excellent trade-off between simplicity, speed, and quality, making it highly suitable for low-resource scenarios.
- Diffusion models, while computationally expensive, show great promise in high-fidelity generation tasks with sufficient resources and training time.
- Thorough hyperparameter tuning, particularly batch size and learning rate, is critical to unlocking a model's full potential.

Future Work

Potential future extensions of this project include:

- Exploring hybrid architectures that combine strengths of multiple paradigms, such as VAE-GANs or diffusion with adversarial fine-tuning.
- Applying the same comparative framework to other datasets (e.g., human faces, medical images) to assess model generalizability.
- Incorporating advanced evaluation metrics like Inception Score, precision-recall curves, or human perception-based ratings for deeper insight.
- Leveraging larger models or pretrained backbones (e.g., StyleGAN2, Imagen, or Stable Diffusion) for higher-quality synthesis and domain adaptation tasks.

Overall, this project demonstrated the capabilities and trade-offs of deep generative models in a controlled and well-documented setting, providing practical guidance for future research and real-world generative modeling applications.

References

[1] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

[3] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[4] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 2019.