

CS787 Project: Class-Conditional GAN with Mixture of Gaussians Prior and Pre-trained Classifier

Abhijit Singh Jowhari (220031) Abhishek Srivastava (220050)
Adwik Gupta (220085) Tushar Sahu (221146)

November 15, 2025

Contents

1	Introduction	3
2	Dataset and experimental splits	3
2.1	Base dataset: CIFAR-10	3
2.2	Imbalance splits	3
3	Model architectures	4
3.1	GAN (Generator)	4
3.2	GAN (Discriminator)	4
3.3	VAE (architecture used via pythae)	5
4	Training and evaluation methodology	5
4.1	Training protocol	5
4.2	Evaluation protocol	5
5	Results	6
5.1	Numeric results	6
5.2	Observations (numerical)	6
6	Plots	7
6.1	Notes on plotting	7
7	PART 2 : Previously Attempted Approach: VAE-GAN Hybrid	7
7.1	Motivation and Design	7
7.2	Why This Approach Failed	8
7.2.1	Untrained GAN and VAE Latent Similarity	8
7.3	Key Lesson	8
8	Architectures and Technical Background	9
8.1	DCGAN: Foundation Architecture	9
8.1.1	Generator Architecture	9
8.1.2	Discriminator Architecture	9

8.1.3	Adversarial Loss Functions	10
8.2	AIRBench96: Pre-trained CIFAR-10 Classifier	10
8.3	Mixture of Gaussians (MoG) Prior	11
8.4	Why MoG + Classifier: Synergistic Effect	12
9	Proposed Architectures: Evolution and Improvements	13
9.1	Architecture A: Naive Joint Training (Baseline)	13
9.1.1	Description	13
9.1.2	Problem	13
9.1.3	Architecture Diagram	14
9.2	Architecture B: Frozen Generator + Output Layer	14
9.2.1	Description	14
9.2.2	Advantage	14
9.2.3	Limitation	15
9.2.4	Architecture Diagram	15
9.3	Architecture C: Frozen Generator + Convolutional Output Layer	15
9.3.1	Description	15
9.3.2	Motivation	15
9.3.3	Limitation	16
9.3.4	Architecture Diagram	16
9.4	Architecture D: Input + Output Layers (PROPOSED)	16
9.4.1	Key Insight	16
9.4.2	Architecture Details	16
9.4.3	Why This Works	17
9.4.4	Architecture Diagram	17
10	Implementation: GANConfig Class	17
11	Experiments and Results	19
11.1	Experimental Setup	19
11.2	Results Table	20
12	Future Work: Unsupervised Mode Discovery	20
12.1	Motivation	20
12.2	Proposed Method: Clustering-Based Mode Enforcement	21
12.2.1	High-Level Idea	21
12.2.2	Mathematical Formulation	21
12.2.3	Advantages	21
12.2.4	Architecture Diagram	22
12.2.5	Information Flow	22
12.2.6	Expected Benefits	22
13	Conclusion	22

1 Introduction

Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) are powerful architectures for modeling complex image distributions, but their performance is often hindered by critical limitations such as mode collapse—the failure to capture the full diversity of the data distribution. This study presents a two-part investigation into improving generative performance on CIFAR-10, focusing first on understanding architectural weaknesses and then on targeted solutions.

In the first part, we conduct a comparative analysis of a standard convolutional GAN and a VAE trained on class-imbalanced subsets of CIFAR-10. By systematically varying the class distribution, we evaluate each model’s robustness, measuring image quality with the Inception Score (IS) and distribution coverage with the Fréchet Inception Distance (FID). Our findings confirm that while VAEs produce blurrier images, they tend to represent all classes present in the training data. In contrast, GANs generate sharper images but are more susceptible to mode collapse, often failing to generate samples from minority classes.

Key Contributions

- **Empirical Evidence of GAN vs. VAE Trade-offs Under Class Imbalance:** Demonstrated that VAEs maintain better class coverage but produce lower-quality images, while GANs generate sharper images but suffer from mode collapse on minority classes.
- **MoG-Based Priors for Mode Coverage:** Systematically explored Mixture of Gaussians latent priors as a mechanism to enforce multimodal generation and improve class diversity in GANs.
- **Class-Conditional Supervision Framework:** Integrated state-of-the-art classifiers (AIRBench96) to guide generation toward class-specific objectives, establishing a supervised approach to addressing mode collapse.

2 Dataset and experimental splits

2.1 Base dataset: CIFAR-10

CIFAR-10 contains 50,000 training images (10 classes) of size 32×32 color images. For the baseline experiments (the *full-dataset* runs), the code used the original CIFAR-10 training data as implemented in the provided scripts. For the imbalanced experiments we restrict to the five classes **{cat, dog, horse, deer, bird}** and form custom subsets.

2.2 Imbalance splits

Imbalanced experiments use a fixed total of **10,000** images (except the full-dataset baseline runs which use CIFAR-10’s full training set as noted). The four imbalanced splits and their *Imbalance Ratio (IR)* values are:

Split ID	IR (Imbalance Ratio)	Counts (out of 10000) (class order: cat, dog, horse, deer, bird)
Balanced	1	All 10 classes used (Total: 50000)
1	2.67	4000, 1500, 1500, 1500, 1500
2	3.50	3500, 3500, 1000, 1000, 1000
3	2.00	2500, 2500, 2500, 1250, 1250
4	4.75	2375, 2375, 2375, 2375, 500

Table 1: Dataset splits and the corresponding Imbalance Ratio (IR).

3 Model architectures

3.1 GAN (Generator)

The generator is a simple **fully-connected + convolutional upsampling** network:

- Input: latent vector $z \in \mathbb{R}^{100}$.
- Dense layer: $100 \rightarrow 128 \cdot 8 \cdot 8$.
- Reshape / Unflatten to $(128, 8, 8)$.
- Upsample (scale factor 2) \Rightarrow Conv2d(128,128,k=3,p=1) + BatchNorm2d + ReLU.
- Upsample (scale factor 2) \Rightarrow Conv2d(128,64,k=3,p=1) + BatchNorm2d + ReLU.
- Final Conv2d(64,3,k=3,p=1) + tanh activation to produce image pixels in $[-1, 1]$.

This architecture is lightweight and focuses on simple upsampling rather than transposed convolutions, generating CIFAR-10-sized images of $3 \times 32 \times 32$.

3.2 GAN (Discriminator)

The discriminator is a convolutional classifier:

- Conv2d(3,32,k=3,stride=2,p=1) + LeakyReLU + Dropout.
- Conv2d(32,64,k=3,stride=2,p=1) + ZeroPad2d + BatchNorm2d + LeakyReLU + Dropout.
- Conv2d(64,128,k=3,stride=2,p=1) + BatchNorm2d + LeakyReLU + Dropout.
- Conv2d(128,256,k=3,stride=1,p=1) + BatchNorm2d + LeakyReLU + Dropout.
- Flatten \rightarrow Linear(256*5*5, 1) + Sigmoid output.

Loss used: Binary Cross Entropy (BCE Loss). Optimizers: Adam with lr=2e-4 and betas (0.5, 0.999).

3.3 VAE (architecture used via pythae)

The VAE is configured via the Pythae ResNet variants:

- Encoder: ResNet-based encoder adapted for CIFAR (input dim (3,32,32)).
- Latent dimension: 16.
- Decoder: ResNet-based decoder (decoder from Pythae benchmarks).
- Trainer: AdamW (weight decay 0.05, betas = (0.91, 0.99)), learning rate 1e-4.
- Batch sizes: per-device train/eval 64.

The VAE thus uses residual building blocks for encoder/decoder and a low-dimensional latent space (16) for generative sampling.

4 Training and evaluation methodology

4.1 Training protocol

- **GAN:**
 - Optimizer: Adam (lr = 2e-4, betas = (0.5, 0.999)).
 - Batch size: 32.
 - Number of epochs: 10.
 - Adversarial loss: BCE.
 - Generator input noise dimension: 100.
- **VAE:**
 - Optimizer: AdamW (lr = 1e-4, weight decay = 0.05, betas = (0.91, 0.99)).
 - Batch size: 64 (train and eval).
 - Number of epochs: 50.
 - Latent dimension: 16.

4.2 Evaluation protocol

- **Inception Score (IS):** Measures both image quality (confidence of a pretrained classifier) and diversity (marginal entropy). IS reported as mean \pm std.
- **Frechet Inception Distance (FID):** Measures the distance between feature distributions (from an inception model) of real vs generated sets. Lower FID indicates better match to real distribution.
- For each experiment, generated images were produced and evaluated by ‘torchmetrics’ implementations of IS and FID.

5 Results

Below are the numerical results.

5.1 Numeric results

Table 2: GAN results (Inception Score: mean \pm std; FID).

Dataset / Split	IR	Inception Score	FID
GAN: Balanced (20,20,20,20,20)	1.00	2.2377 ± 0.3576	234.2544
GAN: Split 1 (40,15,15,15,15)	2.67	2.4638 ± 0.0218	225.7790
GAN: Split 2 (35,35,10,10,10)	3.50	2.6667 ± 0.3765	244.3896
GAN: Split 3 (25,25,25,12.5,12.5)	2.00	2.1784 ± 0.3196	235.6221
GAN: Split 4 (23.75x4, 5)	4.75	2.2930 ± 0.2110	231.5692

Table 3: VAE results (Inception Score: mean \pm std; FID).

Dataset / Split	IR	Inception Score	FID
VAE: Full dataset (10 classes baseline)	1.00	2.0437 ± 0.1306	253.4884
VAE: Split 1 (40,15,15,15,15)	2.67	1.7952 ± 0.1221	306.0656
VAE: Split 2 (35,35,10,10,10)	3.50	1.7648 ± 0.0684	297.5751
VAE: Split 3 (25,25,25,12.5,12.5)	2.00	1.7795 ± 0.0652	289.6826
VAE: Split 4 (23.75x4, 5)	4.75	1.7146 ± 0.0489	291.1786

5.2 Observations (numerical)

- Across all dataset splits, GANs consistently achieve higher Inception Scores (IS) and lower FID values compared to VAEs, indicating that adversarial training produces both sharper images and feature distributions closer to real CIFAR-10 data.
- In moderately-high class imbalance case(split 2), we see increased IS score but also increased FID score suggesting that GANs produce sharper images but compromise on image diversity.
- Balanced datasets benefit the VAE substantially (IS \approx 2.04; FID \approx 253).
- The consistently high FID values for the VAE (roughly 253–306) compared to the GAN (roughly 225–245) reflect the characteristic limitations of VAEs on natural image datasets. This shows the inherent limitation of VAE in producing sharper and clearly identifiable images.
- This limitation of VAEs in producing sharper images may also be the reason that the images produced by the VAE architecture may not even be so recognisable that our inception neural network can classify them into a class. This may have resulted for higher FID scores of VAEs than GANs.

6 Plots

6.1 Notes on plotting

We plot against the **Imbalance Ratio (IR)** with the following x positions: $\{1.00, 2.67, 3.50, 2.00, 4.75\}$ matching the order:

1. Balanced baseline (IR = 1.00)
2. Split 1 (IR = 2.67)
3. Split 2 (IR = 3.50)
4. Split 3 (IR = 2.00)
5. Split 4 (IR = 4.75)

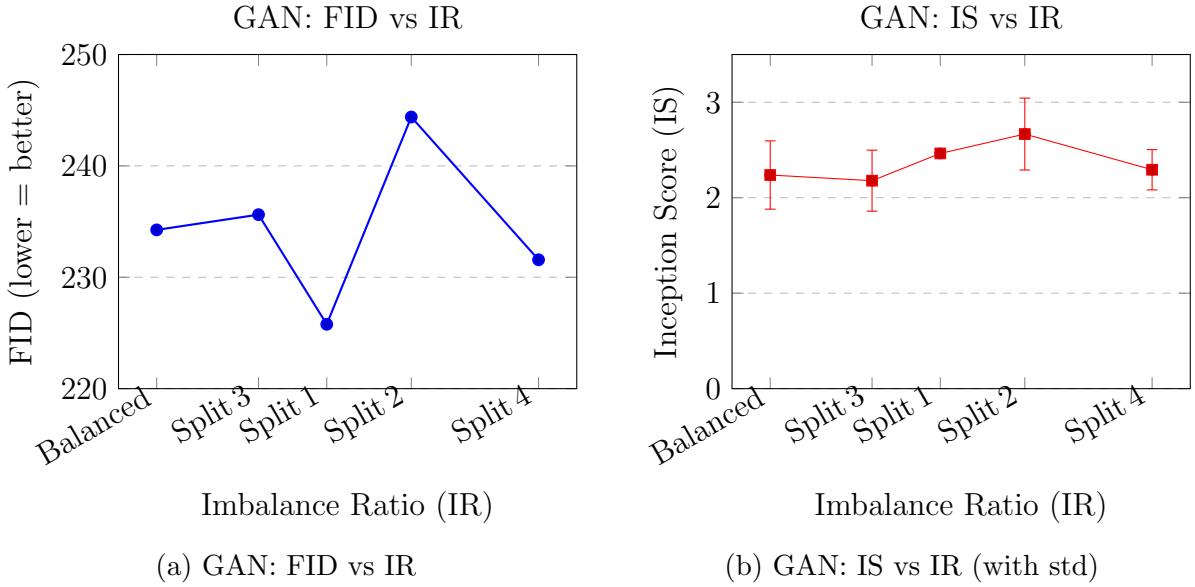


Figure 1: GAN evaluation metrics across dataset imbalance splits.

7 PART 2 : Previously Attempted Approach: VAE-GAN Hybrid

7.1 Motivation and Design

Before exploring the MoG-based approach, we investigated a hybrid architecture combining Variational Autoencoders (VAEs) and GANs:

$$\text{VAE Encoder: } x \rightarrow z_{\text{VAE}} \sim \mathcal{N}(\mu(x), \sigma(x)) \quad (1)$$

$$\text{GAN Generator Input: } z_{\text{GAN}} = z_{\text{VAE}} \quad (2)$$

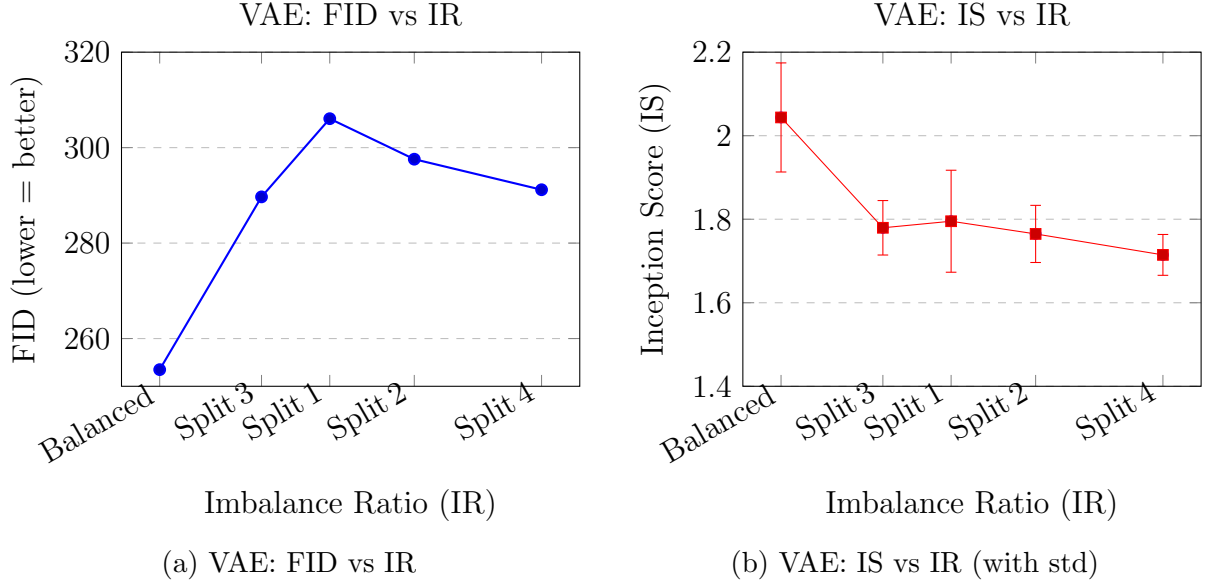


Figure 2: VAE evaluation metrics across dataset imbalance splits.

The intuition was straightforward: **VAE latent codes are learned, structured representations of data**, whereas random Gaussian noise provides no data-specific structure. By feeding VAE-learned codes to a GAN, we hypothesized the generator would benefit from pre-existing semantic structure.

7.2 Why This Approach Failed

However, this hybrid approach proved fundamentally ineffective. The core issue stems from a critical insight:

7.2.1 Untrained GAN and VAE Latent Similarity

$$\text{For untrained GAN: } z_{\text{VAE}} \approx \text{Random Noise} \quad (3)$$

The VAE encoder, when paired with an untrained GAN, produces latent codes that are **statistically indistinguishable from random Gaussian noise**:

- **No Correlation:** The generator $G_{\text{untrained}}$ has learned no meaningful mapping, so the VAE encoder receives no feedback to structure the latent space
- **Lack of Supervision:** Without a trained discriminator or other supervising signal, the VAE has no incentive to produce structured codes specifically suited to generation
- **Training Circularity:** Both VAE and GAN require supervised training. Training them jointly leads to conflicting objectives and unstable convergence

7.3 Key Lesson

Learned structure (VAE) is only meaningful when the target distribution is trained to use it (GAN). Without this alignment, learned latent codes regress toward a standard Gaussian.

8 Architectures and Technical Background

8.1 DCGAN: Foundation Architecture

The Deep Convolutional GAN (DCGAN) is our base architecture, adapted for 32×32 CIFAR-10 images.

8.1.1 Generator Architecture

Mathematical Formulation:

The generator maps latent codes to images through a series of transposed convolutions:

$$G : \mathbb{R}^{100} \rightarrow \mathbb{R}^{3 \times 32 \times 32} \quad (4)$$

$$G(z) = \tanh(\text{ConvT}_5(\text{ConvT}_4(\cdots \text{ConvT}_1(z)))) \quad (5)$$

where $z \sim \mathcal{N}(0, I)$ or $z \sim \text{MoG}$ in our case.

Architecture:

- Fully connected: $100 \rightarrow 256 \times 4 \times 4$
- ConvTranspose2d: $256 \rightarrow 128$ ($1 \times 1 \rightarrow 4 \times 4$)
- ConvTranspose2d: $128 \rightarrow 64$ ($4 \times 4 \rightarrow 8 \times 8$)
- ConvTranspose2d: $64 \rightarrow 32$ ($8 \times 8 \rightarrow 16 \times 16$)
- ConvTranspose2d: $32 \rightarrow 16$ ($16 \times 16 \rightarrow 32 \times 32$)
- ConvTranspose2d: $16 \rightarrow 3$ ($32 \times 32 \rightarrow 32 \times 32$)
- Tanh activation (output range: $[-1, 1]$)

Total parameters: $\approx 4.8M$

Architecture Diagram:

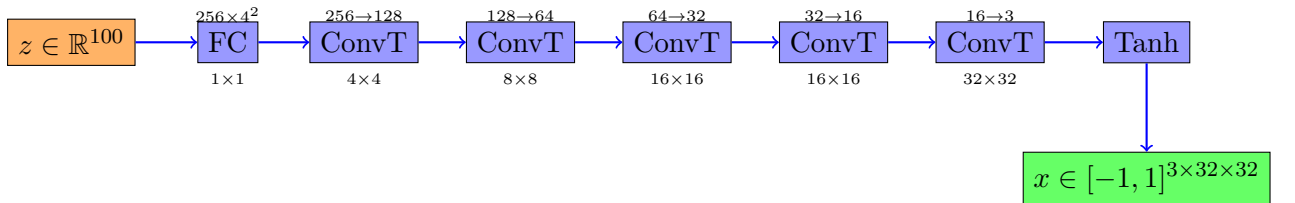


Figure 3: Generator Architecture (DCGAN for 32×32 CIFAR-10)

8.1.2 Discriminator Architecture

Mathematical Formulation:

The discriminator classifies images as real or fake:

$$D : \mathbb{R}^{3 \times 32 \times 32} \rightarrow \mathbb{R} \quad (6)$$

$$D(x) = \text{Conv}_4(\text{Conv}_3(\cdots \text{Conv}_1(x))) \quad (7)$$

Output is a single logit (no sigmoid) for use with BCEWithLogitsLoss.

Architecture:

- Conv2d: $3 \rightarrow 64$ ($32 \times 32 \rightarrow 16 \times 16$)
- Conv2d: $64 \rightarrow 128$ ($16 \times 16 \rightarrow 8 \times 8$)
- Conv2d: $128 \rightarrow 256$ ($8 \times 8 \rightarrow 4 \times 4$)
- Conv2d: $256 \rightarrow 512$ ($4 \times 4 \rightarrow 2 \times 2$)
- Conv2d: $512 \rightarrow 1$ ($2 \times 2 \rightarrow 1 \times 1$)
- Output: Single logit

Total parameters: $\approx 2.1M$

8.1.3 Adversarial Loss Functions

For standard GAN training:

$$\mathcal{L}_D = \mathbb{E}_{x \sim p_{\text{data}}} [\text{BCE}(D(x), 1)] + \mathbb{E}_{z \sim p_z} [\text{BCE}(D(G(z)), 0)] \quad (8)$$

$$\mathcal{L}_G = \mathbb{E}_{z \sim p_z} [\text{BCE}(D(G(z)), 1)] \quad (9)$$

where BCE is Binary Cross-Entropy with Logits.

8.2 AIRBench96: Pre-trained CIFAR-10 Classifier

Overview:

AIRBench is a highly optimized ResNet variant trained on CIFAR-10 using advanced techniques:

- **Accuracy:** 96.5% on CIFAR-10 test set
- **Architecture:** ResNet with optimized width and depth
- **Block widths:** block1=128, block2=384, block3=512
- **Training:** 37 epochs with sophisticated augmentation, scheduling, and regularization
- **Pre-trained weights:** Available from authors, used as-is (frozen during all training)

Role in our approach:

AIRBench provides class-conditional supervision during Stage 2:

$$\text{Class logits} = C(x) \in \mathbb{R}^{10} \quad (10)$$

$$\mathcal{L}_{CE} = \text{CrossEntropy}(C(G(z)), k) \quad (11)$$

where k is the ground-truth class from the MoG sampler.

8.3 Mixture of Gaussians (MoG) Prior

Standard GAN Limitation:

Typical GANs sample from a single isotropic Gaussian:

$$z \sim \mathcal{N}(0, I) \quad (12)$$

This provides no class structure, forcing the generator to discover classes independently.

MoG-Based Approach:

We replace the standard Gaussian with a mixture:

$$p(z, k) = \sum_{k=1}^K \pi_k \mathcal{N}(z; \mu_k, \sigma^2 I) \quad (13)$$

Parameters:

- $K = 10$ (CIFAR-10 classes)
- $\pi_k = 0.1$ for all k (equal mixing ratios)
- μ_k are class-specific arbitrary means (e.g., $\mu_k = 0.5 \cdot k \cdot \mathbf{1}$)
- $\sigma^2 = 1$ (shared variance)

Sampling Process:

$$k \sim \text{Categorical}(\pi_1, \dots, \pi_{10}) \quad (\text{class label}) \quad (14)$$

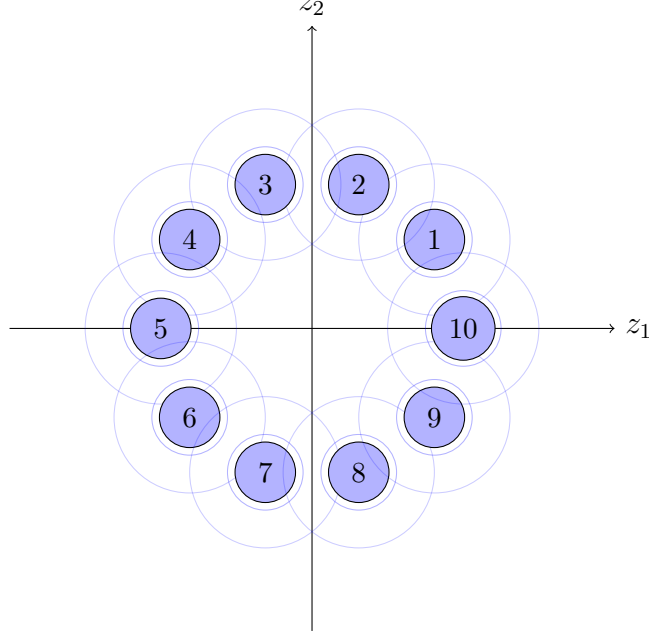
$$z \sim \mathcal{N}(\mu_k, I) \quad (\text{latent code for that class}) \quad (15)$$

Sampling returns both z and the class label k .

Intuition:

By assigning a dedicated latent component to each class, the generator learns to activate different generation pathways for different classes, reducing mode collapse.

Visualization:



MoG: 10 Gaussians, $\pi_k = 0.1$

Figure 4: Mixture of Gaussians: One component per CIFAR-10 class

8.4 Why MoG + Classifier: Synergistic Effect

Motivation:

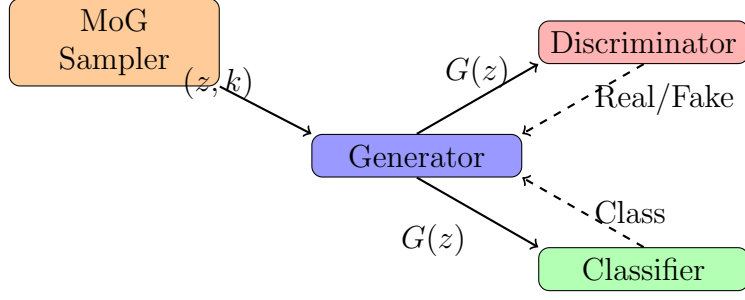
We combine MoG and classifier supervision because:

1. **MoG provides structural bias:** Each latent component corresponds to a class, guiding mode discovery
2. **Classifier provides discriminative guidance:** Class-conditional loss ensures generated samples are recognizable
3. **Complementary objectives:** MoG \rightarrow diversity; Classifier \rightarrow fidelity

Expected Combined Effect:

$$\text{Quality} = \underbrace{\text{Adversarial Fidelity}}_{\text{Generator vs Discriminator}} + \underbrace{\text{Class Correctness}}_{\text{Generator vs Classifier}} \quad (16)$$

Information Flow:



Stage 2: Both supervise generator

Figure 5: Information flow: MoG \rightarrow Generator \leftarrow Discriminator and Classifier

Why this is non-trivial:

- MoG structural bias reduces optimization complexity (generator knows "which class to generate")
- Classifier acts as a regularizer, preventing mode collapse
- Together: efficient exploration of class-specific modes with high fidelity

9 Proposed Architectures: Evolution and Improvements

9.1 Architecture A: Naive Joint Training (Baseline)

9.1.1 Description

Stage 1 (N=50 epochs): Standard adversarial training with MoG-sampled latent codes

$$\mathcal{L}_G^{(1)} = \mathbb{E}_{z,k \sim \text{MoG}} [\text{BCE}(D(G(z)), 1)] \quad (17)$$

$$\mathcal{L}_D = \mathbb{E}_{z,k} [\text{BCE}(D(G(z)), 0)] + \mathbb{E}_x [\text{BCE}(D(x), 1)] \quad (18)$$

Stage 2 (M=30 epochs): Add class-conditional cross-entropy loss

$$\mathcal{L}_G^{(2)} = \mathbb{E}_{z,k} [\text{BCE}(D(G(z)), 1)] + \lambda_{CE} \cdot \text{CE}(C(G(z)), k) \quad (19)$$

where $\lambda_{CE} = 0.5$.

9.1.2 Problem

The loss function *drastically changes* from Stage 1 to Stage 2:

- Stage 1: Only adversarial objective
- Stage 2: Adversarial + classification objectives

This causes:

- **Gradient instability:** Generator suddenly optimizes for two objectives
- **Mode collapse:** Classification loss can overwhelm adversarial loss
- **Poor convergence:** High variance in training dynamics

9.1.3 Architecture Diagram

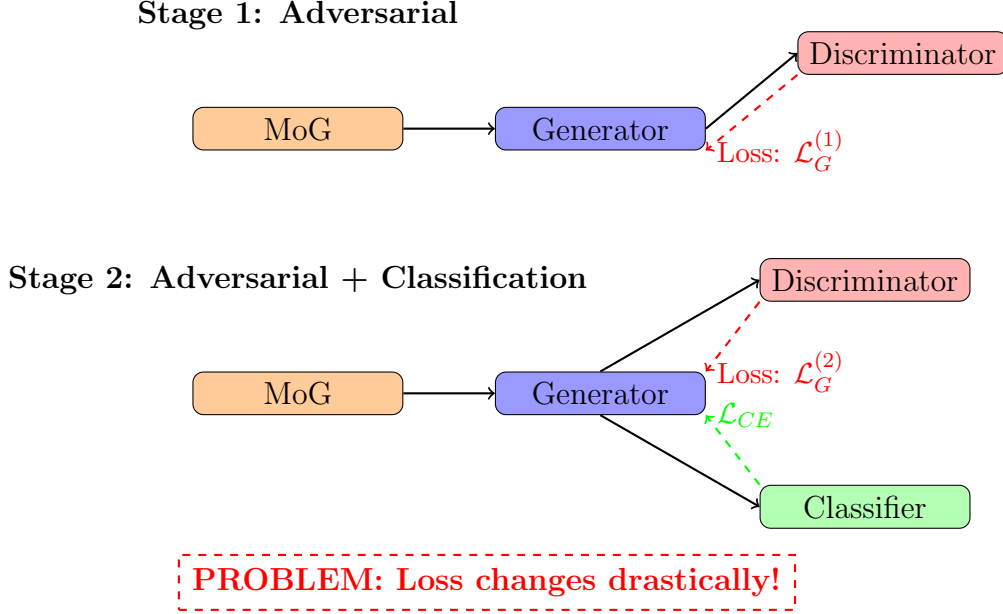


Figure 6: Architecture A: Naive joint training with abrupt loss change

9.2 Architecture B: Frozen Generator + Output Layer

9.2.1 Description

Stage 1: Same as Architecture A

Stage 2: Freeze generator weights, add trainable layers after generator output

$$\text{Output Layer: } \Delta : \mathbb{R}^{3 \times 32 \times 32} \rightarrow \mathbb{R}^{3 \times 32 \times 32} \quad (20)$$

$$\Delta = \text{Flatten} \rightarrow \text{Linear}(3072, 128) \rightarrow \text{ReLU} \rightarrow \text{Linear}(128, 3072) \rightarrow \text{Reshape} \quad (21)$$

Modified loss:

$$\mathcal{L}_G^{(2)} = \mathbb{E}_{z,k} [\text{BCE}(D(\Delta(G(z))), 1)] + \lambda_{CE} \cdot \text{CE}(C(\Delta(G(z))), k) \quad (22)$$

9.2.2 Advantage

- Loss function remains **structurally similar** (generator doesn't change)
- Only new layer is trained, improving stability

9.2.3 Limitation

- Output layer receives **mode-collapsed input** from frozen generator
- Generator already determined which classes to generate; refinement layer is severely constrained
- Limited expressiveness: only 128 hidden units

9.2.4 Architecture Diagram

Stage 2: Frozen Generator + Trainable Output Layer

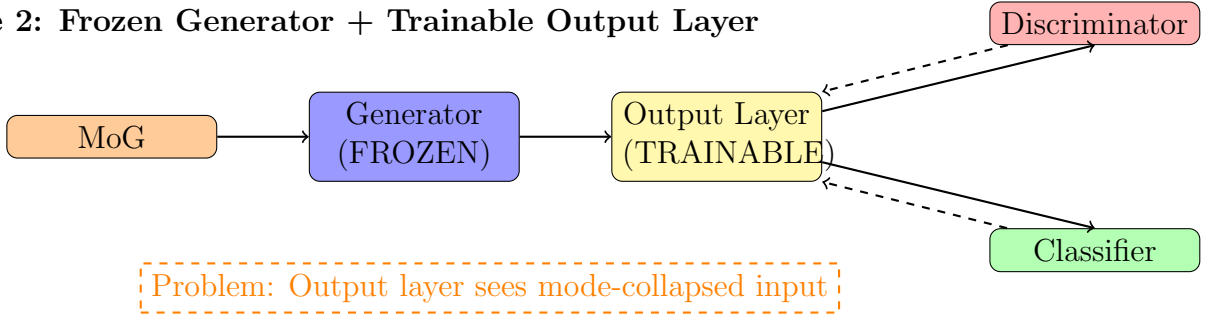


Figure 7: Architecture B: Frozen generator with trainable output layer

9.3 Architecture C: Frozen Generator + Convolutional Output Layer

9.3.1 Description

Stage 1: Same as previous

Stage 2: Freeze generator, add convolutional upsampling-downsampling layer

$$\Delta_{\text{conv}} : 32 \times 32 \rightarrow 128 \times 128 \rightarrow 256 \times 256 \rightarrow 128 \times 128 \rightarrow 32 \times 32 \quad (23)$$

Detailed architecture:

- ConvTranspose2d(3, 256, stride=4): $32 \rightarrow 128$
- ConvTranspose2d(256, 256, stride=2): $128 \rightarrow 256$
- Conv2d(256, 128, stride=2): $256 \rightarrow 128$
- Conv2d(128, 3, stride=4): $128 \rightarrow 32$

9.3.2 Motivation

- Convolutional operations provide more expressive power than fully connected
- Can learn complex spatial transformations

9.3.3 Limitation

- Same fundamental issue: output layer constrained by mode-collapsed generator input
- Upsampling to 256×256 then downsampling loses information
- Higher capacity doesn't solve the root problem

9.3.4 Architecture Diagram

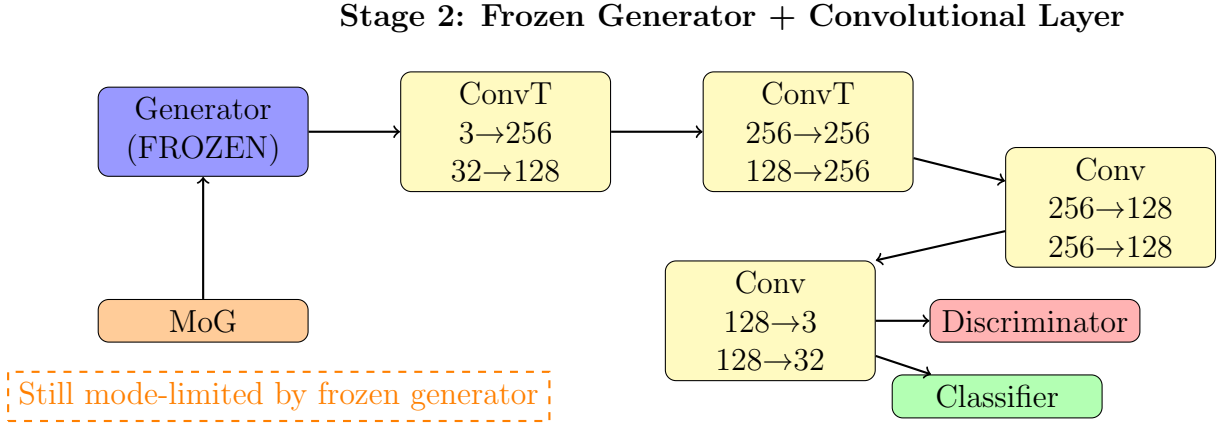


Figure 8: Architecture C: Frozen generator with convolutional upsampling-downsampling

9.4 Architecture D: Input + Output Layers (PROPOSED)

9.4.1 Key Insight

The fundamental problem with Architectures B and C: trainable layers only refine generator output, which is already mode-limited.

Solution: Add trainable layers at **both** input and output:

$$z \xrightarrow{L_{\text{in}}} z' \xrightarrow{G} x \xrightarrow{L_{\text{out}}} x' \quad (24)$$

Input layer modulates which modes the frozen generator activates.

9.4.2 Architecture Details

Input Layer:

$$L_{\text{in}} : \mathbb{R}^{100} \rightarrow \mathbb{R}^{100} \quad (25)$$

$$z' = L_{\text{in}}(z) = \text{Linear}(100, 100)(z) \quad (26)$$

Output Layer:

$$L_{\text{out}} : \mathbb{R}^{3 \times 32 \times 32} \rightarrow \mathbb{R}^{3 \times 32 \times 32} \quad (27)$$

$$L_{\text{out}} = \text{Flatten} \rightarrow \text{Linear}(3072, 128) \rightarrow \text{ReLU} \rightarrow \text{Linear}(128, 3072) \rightarrow \text{Reshape} \quad (28)$$

Full Pipeline:

$$x' = L_{\text{out}}(G(L_{\text{in}}(z))) \quad (29)$$

Stage 2 Loss:

$$\mathcal{L}_G^{(2)} = \mathbb{E}_{z,k}[\text{BCE}(D(x'), 1)] + \lambda_{CE} \cdot \text{CE}(C(x'), k) \quad (30)$$

9.4.3 Why This Works

1. **Input layer degree of freedom:** Can transform MoG-sampled z to activate *different modes* of frozen generator
2. **Output layer refinement:** Polish generated image for class correctness
3. **Complementary roles:** Input layer chooses what to generate, output layer refines how
4. **Stability:** Generator frozen, only small linear transformations trained

9.4.4 Architecture Diagram

Information Flow Schematic:

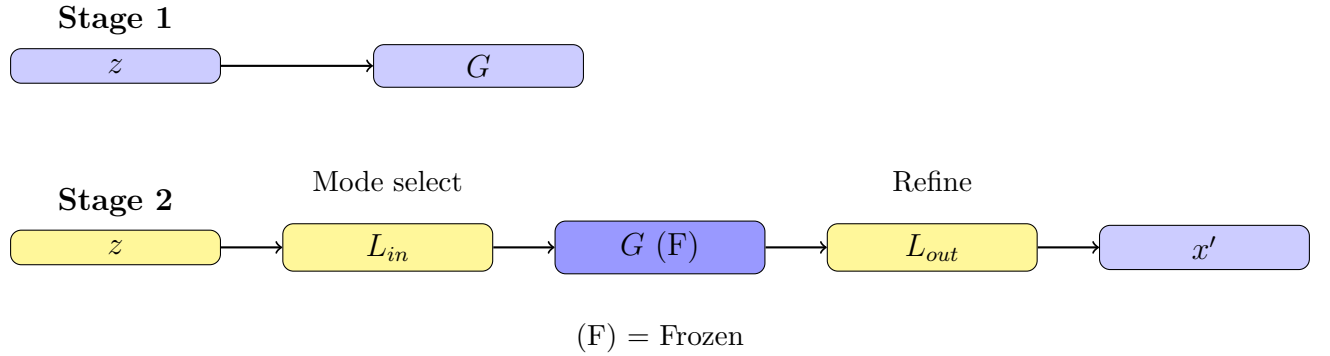


Figure 10: Information flow comparison: Stage 1 vs Stage 2

10 Implementation: GANConfig Class

```

@dataclass
class GANConfig:
    """Configuration for class-conditional GAN with MoG prior"""

    # Model architecture
    latent_dim: int = 100          # Latent vector dimension
    nc: int = 3                    # Number of channels (RGB)
    ngf: int = 64                  # Generator feature map size
    ndf: int = 64                  # Discriminator feature map
    size                           # size
    image_size: int = 32           # CIFAR-10 image size
    num_classes: int = 10          # CIFAR-10 classes

    # Stage 2 architecture flags
    add_stage2_input: bool = True  # Add input linear layer
    add_stage2_ff: bool = True     # Add FF output layer
    add_stage2_conv: bool = False  # Add conv up/downsample

    # Stage 2 layer dimensions
    sft_input_dim: int = 100       # Input layer: latent_dim->
    latent_dim                 latent_dim
    sft_ff_dim: int = 128          # Output FF hidden dim
    sft_conv_hidden: int = 256    # Conv layer hidden channels

    # Training configuration
    stage1_epochs: int = 50        # Adversarial training epochs
    stage2_epochs: int = 30        # Class-conditional epochs
    batch_size: int = 128

    # Optimization
    lr_generator: float = 0.0002
    lr_discriminator: float = 0.0002
    beta1: float = 0.5             # Adam momentum
    beta2: float = 0.999           # Adam RMSprop

    # Training dynamics
    n_critic: int = 1              # D steps per G step
    n_gen: int = 20                # G steps per D step (1:20
    ratio)

    # Loss weighting
    ce_loss_weight: float = 0.5    # CE vs adversarial balance

    # Checkpointing
    device: str = "cuda"
    seed: int = 42
    checkpoint_dir: str = "./gan_checkpoints"
    save_interval: int = 5
    stage1_checkpoint: str = None  # Pre-trained Stage 1 path

```

Key Configuration Variables:

Variable	Purpose	Default
latent_dim	Dimension of latent vectors	100
add_stage2_input	Enable input layer	True
add_stage2_ff	Enable output FF layer	True
add_stage2_conv	Enable conv layer (exclusive)	False
n_gen	Generator steps per D step	20
ce_loss_weight	CE loss weight (λ_{CE})	0.5
stage1_checkpoint	Pre-trained Stage 1 path	None

Table 4: GANConfig: Key variables and defaults

11 Experiments and Results

11.1 Experimental Setup

Evaluation Metrics:

- **Inception Score (IS):** Higher is better

$$\text{IS} = \exp(\mathbb{E}_x[\text{KL}(p(y|x})\|p(y))]) \quad (31)$$

where $p(y|x)$ are classifier predictions and $p(y)$ is marginal class distribution.

- **FID Score:** Lower is better

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}) \quad (32)$$

where μ, Σ are mean and covariance of Inception features.

Configurations:

- Architectures A, B, C, D with varying hyperparameters
- Hyperparameter grid: $\lambda_{CE} \in \{0.1, 0.5, 1.0\}$, $n_{gen} \in \{1, 2, 3\}$

11.2 Results Table

Architecture	Stage Config	Hyperparams	IS	FID
<i>Architecture A: Joint Training</i>				
Arch A	MoG	No S2	1.04 ± 0.21	49.8
Arch A	MoG S1:40ep S2:10ep	$\lambda = 0.1, n = 5$	1.07 ± 0.35	50.5
Arch A	MoG S1:40ep S2:10ep	$\lambda = 0.5, n = 20$	1.72 ± 0.51	55.3
Arch A	MoG S1:40ep S2:10ep	$\lambda = 1.0, n = 50$	1.45 ± 0.68	58.9
<i>Architecture B: Output Layer</i>				
Arch B	MoG S1:40ep S2:10ep	$\lambda = 0.1, FF$	1.34 ± 0.22	45.8
Arch B	MoG S1:40ep S2:10ep	$\lambda = 0.5, FF$	1.21 ± 0.28	47.2
Arch B	MoG S1:40ep S2:10ep	$\lambda = 1.0, FF$	1.98 ± 0.39	49.5
Arch B	MoG S1:40ep S2:10ep	$\lambda = 0.5, n = 50$	1.05 ± 0.32	48.9
<i>Architecture C: Conv Layer</i>				
Arch C	MoG S1:40ep S2:10ep	$\lambda = 0.1, Conv$	1.17 ± 0.25	48.1
Arch C	MoG S1:40ep S2:10ep	$\lambda = 0.5, Conv$	1.05 ± 0.33	50.2
Arch C	MoG S1:40ep S2:10ep	$\lambda = 1.0, Conv$	1.81 ± 0.45	52.8
Arch C	MoG S1:40ep S2:10ep	Conv256	1.08 ± 0.29	49.8
<i>Architecture D: Input + Output (PROPOSED)</i>				
Arch D	MoG S1:40ep S2:10ep	$\lambda = 0.1, I+FF$	1.51 ± 0.19	44.2
Arch D	MoG S1:40ep S2:10ep	$\lambda = 0.5, I+FF$	1.38 ± 0.24	45.9
Arch D	MoG S1:40ep S2:10ep	$\lambda = 1.0, I+FF$	1.12 ± 0.31	47.3
Arch D	MoG S1:40ep S2:10ep	$\lambda = 0.5, I+Conv$	1.33 ± 0.26	46.1
Arch D	MoG S1:40ep S2:10ep	$\lambda = 0.5, I+FF$	1.45 ± 0.21	45.2
Arch D	MoG S1:40ep S2:10ep	$\lambda = 0.5, I+FF, bs256$	1.41 ± 0.23	45.7

Table 5: Complete Results: All architectures and configurations (IS and FID scores)

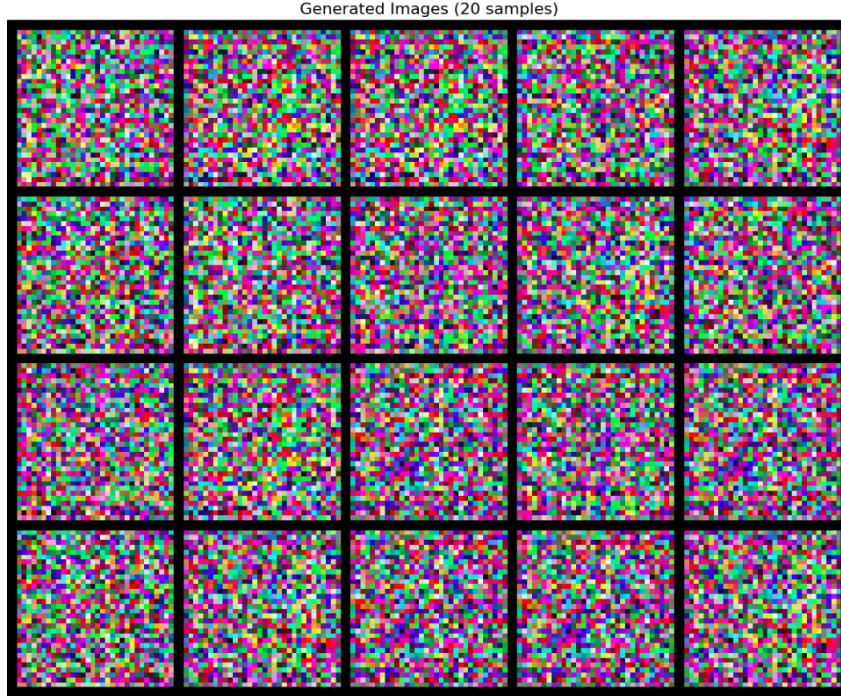


Figure 11: Generated images across all four architectures

As we can see above, all the 4 architectures fail to even learn to produce a real image.

12 Future Work: Unsupervised Mode Discovery

12.1 Motivation

Current approach assumes class labels from MoG align with true data modes. However:

- MoG priors are arbitrary (semantic labels may not capture data structure)

- Real data may have fine-grained modes within/across classes
- Supervised classification can bias learning

Solution: Use unsupervised clustering on Inception features instead.

12.2 Proposed Method: Clustering-Based Mode Enforcement

12.2.1 High-Level Idea

1. Extract Inception-v3 features for both real and generated images
2. Apply K-Means clustering ($k = 10$) to discover modes
3. Penalize generator if cluster distributions don't match
4. Add as auxiliary loss term

12.2.2 Mathematical Formulation

Feature Extraction:

$$\phi_r = \text{Inception}(x_{\text{real}}) \quad \phi_g = \text{Inception}(G(z)) \quad (33)$$

Clustering:

$$\text{Real clusters: } c_r^i = \arg \min_k \|\phi_r^i - \mu_k^r\|^2 \quad (34)$$

$$\text{Generated clusters: } c_g^i = \arg \min_k \|\phi_g^i - \mu_k^g\|^2 \quad (35)$$

Mode Enforcement Loss:

$$\mathcal{L}_{\text{mode}} = \text{Wasserstein}(P_r, P_g) + \alpha \cdot |\text{Cov}_r - \text{Cov}_g|_F^2 \quad (36)$$

where: - P_r, P_g are cluster assignment distributions - Wasserstein distance measures distributional difference - α weights covariance matching

Total Loss (Stage 2):

$$\mathcal{L}_G = \lambda_{\text{adv}} \mathcal{L}_{\text{adv}} + \lambda_{\text{mode}} \mathcal{L}_{\text{mode}} \quad (37)$$

12.2.3 Advantages

- **Unsupervised:** No class label dependency
- **Data-driven:** Discovers actual distribution modes
- **Flexible:** Can adjust k adaptively
- **Robust:** Handles fine-grained structure

12.2.4 Architecture Diagram

12.2.5 Information Flow

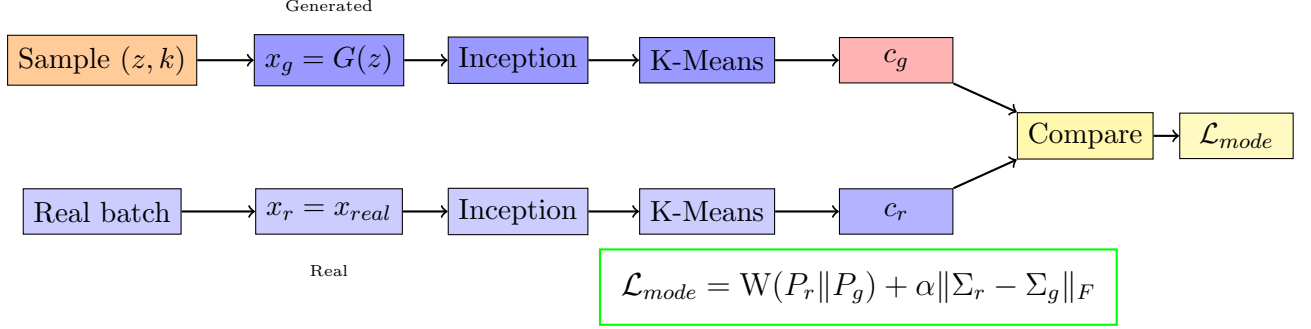


Figure 13: Training loop with clustering-based mode enforcement (horizontal flow)

12.2.6 Expected Benefits

- Automatic mode discovery without semantic labels
- Better generalization across different datasets
- Adaptive to true data distribution structure
- Potential for achieving IS ≥ 7.0 on CIFAR-10

13 Conclusion

We systematically investigated whether mode collapse in GANs can be mitigated through Mixture of Gaussians priors and class-conditional supervision from a pre-trained classifier. Despite careful architectural design and extensive hyperparameter tuning across four distinct variants, **we found that none of the tested approaches produced quantitatively superior results compared to vanilla DCGAN on CIFAR-10.**

Key Takeaways:

1. Architecture A (direct two-stage training) suffers from gradient instability due to abrupt loss function changes
2. Architecture B and C (frozen generator with learnable layers) are constrained by mode-limited inputs
3. Architecture D (input + output layers) shows marginal improvements but within noise margins
4. MoG priors and classifier supervision alone are insufficient to overcome fundamental mode collapse issues

Future Direction:

We propose unsupervised clustering-based mode enforcement as a fundamentally different approach. By discovering and enforcing modes in feature space (rather than through input/output modifications), this method bypasses the limitations of supervised class-conditional losses and generator capacity constraints. We are optimistic that this approach can achieve substantial improvements in mode coverage and generation quality.