Homework 4

Fatemeh Moradihaghighi
Clemson University

Contents

I uploaded the project in the GitHub link below.

https://github.com/fgmoradi/DeepLearning_HW4/tree/main

Introduction

In this assignment, we implement Generative Adversarial Networks (GANs), an unsupervised learning framework, to train a discriminator–generator pair on the CIFAR-10 dataset. We apply three variants—DCGAN, Wasserstein GAN (WGAN), and ACGAN—to generate synthetic images that resemble real CIFAR-10 samples.

A GAN consists of two neural networks that are trained in opposition to each other:

- **Generator:** takes random noise from a latent space and learns to map it to the data distribution of interest, producing new, fake images.
- **Discriminator:** receives both real images (from the dataset) and fake images (from the generator) and learns to distinguish real from fake.

During training, the generator is updated so that its outputs become increasingly realistic, while the discriminator is updated to better detect fake samples. The goal is for the generator to produce images that are so close to the real distribution that the discriminator can no longer reliably tell them apart. GANs are widely used for image, video, and audio generation tasks.

Dataset

The CIFAR-10 dataset contains **60,000** color images of size **32×32** pixels, divided into **10 classes** with **6,000 images per class**. The classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

In this assignment, the dataset is loaded using the `torchvision.datasets.CIFAR10` interface from PyTorch, with standard preprocessing and resizing to match the input size required by the models. Because CIFAR-10 images are relatively small and low-resolution, the generated samples from our GAN models also tend to be somewhat blurry and lack fine detail. The dataset is also multi-class, which makes the learning problem more challenging for GANs compared to learning from a single, simpler class.

1.2 DCGAN

Deep Convolutional GANs (DCGANs) are a popular GAN architecture that uses convolutional and transposed convolutional layers instead of fully connected layers. They are designed with a few architectural guidelines that make training more stable for image generation tasks:

- Replace max-pooling with **strided convolutions** in the discriminator.
- Use **transposed convolutions** (fractionally strided convolutions) in the generator for upsampling.
- Remove large fully connected layers in favor of deep convolutional stacks.
- Apply **batch normalization** in most layers to stabilize training.

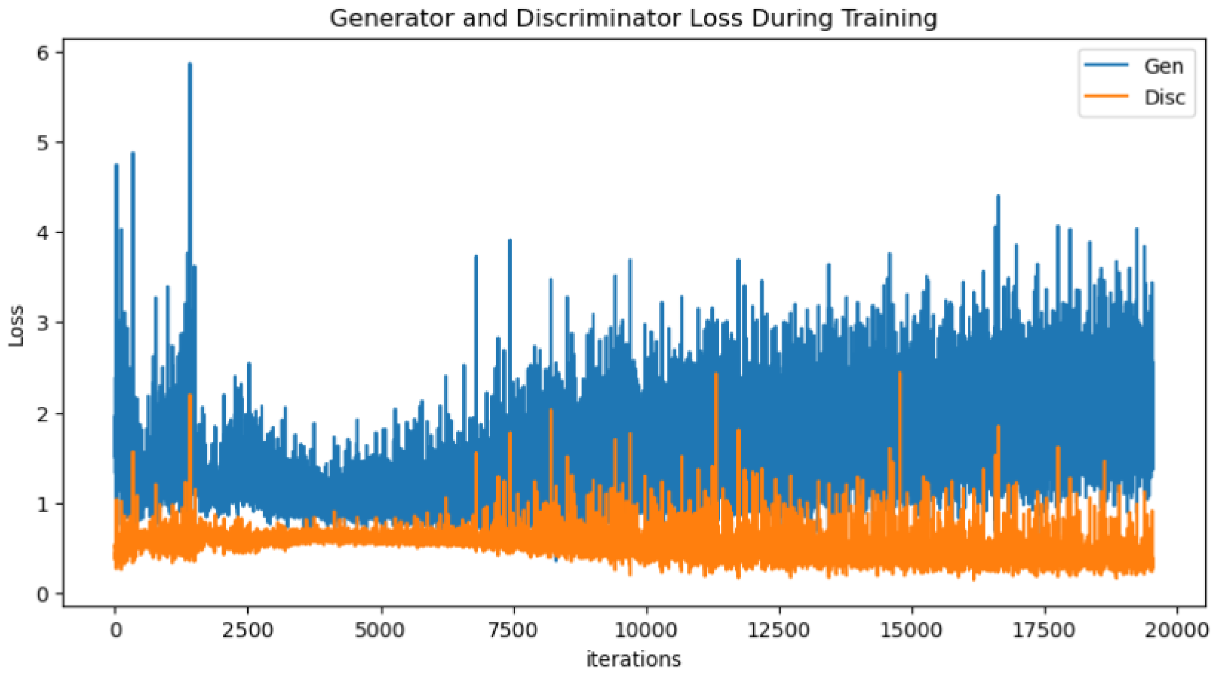
In our setup, the **generator** takes a noise vector from a 100-dimensional latent space and progressively upsamples it using transposed convolutions to output a **64×64 RGB image**. The **discriminator** is a convolutional network that downsamples the input image to a single scalar indicating how "real" the image appears. For nonlinearity, the generator uses **ReLU** activations in its intermediate layers and **tanh** on the output, while the discriminator uses **LeakyReLU** activations.

The following hyperparameter configuration was used to train the DCGAN on CIFAR-10:

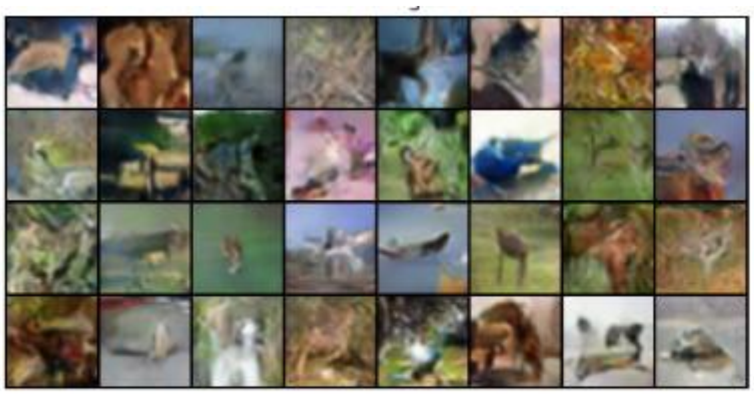
- `LEARNING_RATE = 2e-4`
- `BATCH_SIZE = 128`
- `IMAGE_SIZE = 64 (CIFAR-10 images are resized to 64×64)`
- `CHANNELS_IMG = 3 (RGB images)`
- `NOISE_DIM = 100 (latent vector dimension)`
- `NUM_EPOCHS = 50`
- `FEATURES_DISC = 64 (base number of feature maps in the discriminator)`
- `FEATURES_GEN = 64 (base number of feature maps in the generator)`
- `Optimizer: Adam`
- `beta1 = 0.5, beta2 = 0.999`
- `Loss function:` **`binary cross-entropy loss`** `(nn.BCELoss)`

During training, the generator and discriminator are updated alternately. At each epoch, we save a grid of generated samples using a fixed noise vector, which allows us to visually track how image quality evolves over time. A 128-image grid of real CIFAR-10 training samples is also included for visual comparison with the generated images.

From the loss curves, we observe that around **2,500 iterations** the generator and discriminator losses begin to stabilize and maintain a roughly constant ratio, which is close to the behavior expected in a well-balanced GAN training process. In additional experiments with different numbers of epochs and learning rates, I noticed that after about **60 epochs** the generator loss tends to increase again, and this corresponds to a visible drop in the quality of the generated samples.



The figure below shows images generated at epoch which achieved the lowest FID score in my runs. Although the model is able to capture some coarse structure, many samples are still quite pixelated, low in detail, and sometimes cluttered. This reflects both the difficulty of the CIFAR-10 dataset (small, low-resolution images from 10 diverse classes) and the limitations of the DCGAN model on this data.



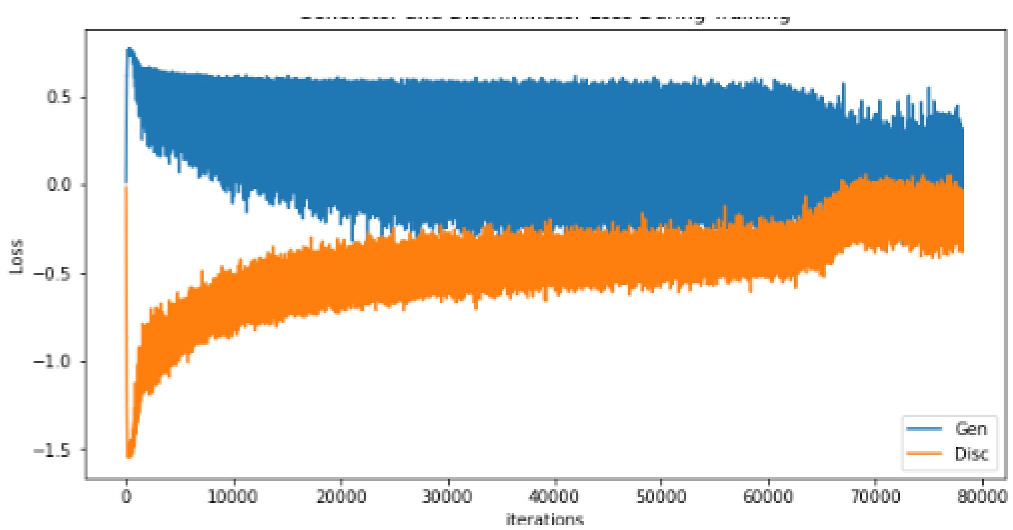
1.3 WGAN

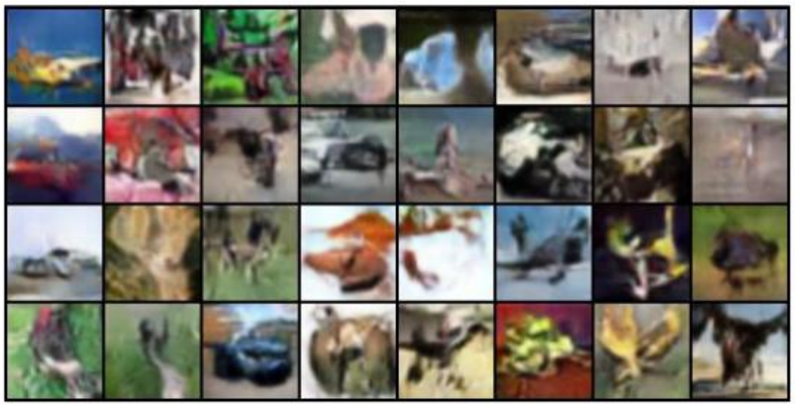
The Wasserstein GAN (WGAN) modifies the standard GAN/DCGAN framework to improve training stability and provide a loss function that correlates better with sample quality. Although the architectural changes to a typical DCGAN are relatively minor, WGAN is grounded in a stronger mathematical formulation based on the Wasserstein (Earth Mover's) distance. In WGAN, the discriminator is replaced by a critic that outputs a real-valued score instead of a probability. Accordingly, the final layer of the critic uses a linear activation (no sigmoid), and the training labels are no longer constrained to 0 and 1. Instead, the critic is trained to assign higher scores to real images and lower scores to generated ones. The critic and generator are optimized using the Wasserstein loss, and to enforce the Lipschitz constraint, the critic's weights are clipped to a small range (e.g., − 0.01 , 0.01 −0.01,0.01) after each update. In practice, the critic is updated multiple times for each generator update; here, the critic is trained five steps for every single generator step. The following hyperparameters were used for the WGAN experiments:

- `LEARNING_RATE = 5e-5`

- `BATCH_SIZE = 32`
- `IMAGE_SIZE = 64`
- `CHANNELS_IMG = 3`
- `NOISE_DIM = 100`
- `NUM_EPOCHS = 50`
- `FEATURES_DISC = 64`
- `FEATURES_GEN = 64`
- `weight_clip = 0.01`
- `critic_iteration = 5`

The figure below shows the generator and critic losses over 50 epochs of training, along with sample images generated by the WGAN at the end of training. Compared to the DCGAN results, the WGAN samples appear more pixelated and less detailed. This is likely because the model has not yet fully converged; WGAN training is relatively slow, especially since the critic is updated multiple times per step, even when using a GPU.





1.4 ACGAN

The Auxiliary Classifier GAN (ACGAN) is an extension of conditional GANs (CGANs). It was introduced by Augustus Odena et al. in the 2016 paper *"Conditional Image Synthesis Using Auxiliary Classifier GANs."* In ACGAN, both the generator and discriminator are conditioned on class labels, but the conditioning is handled differently than in a standard CGAN.

As in CGAN, the **generator** receives two inputs: a latent noise vector and a class label. It learns to produce images that both look realistic and match the requested class. The **discriminator**, however, only takes the image itself as input. It outputs two things simultaneously:

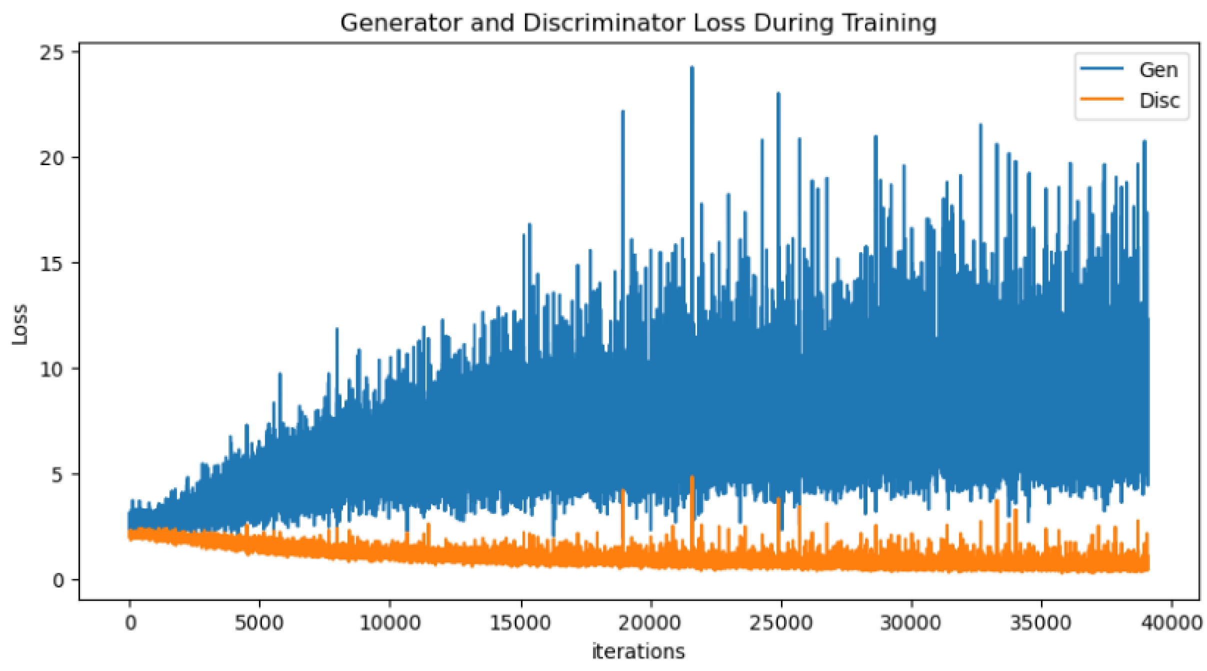
1. A binary prediction indicating whether the image is real or fake.
2. A multi-class prediction for the image's class label.

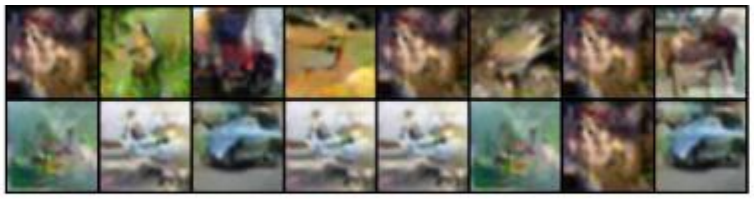
This is the key difference from a basic CGAN, where the discriminator typically receives both the image and the class label as input. In ACGAN, the discriminator infers the class label as an auxiliary task, which encourages the generator to produce class-consistent images.

For the ACGAN experiments, the following hyperparameters were used:

- `LEARNING_RATE = 1.5e-4`
- `BATCH_SIZE = 64`
- `IMAGE_SIZE = 64`
- `CHANNELS_IMG = 3`
- `NOISE_DIM = 100`
- `NUM_EPOCHS = 50`
- `FEATURES_DISC = 64`
- `FEATURES_GEN = 64`
- `NUM_CLASSES = 10` (CIFAR-10 classes)
- `EMBED_SIZE = 100` (embedding dimension for class labels)

The next figure shows the generator and discriminator losses across 50 epochs of ACGAN training, followed by sample images generated by the model.



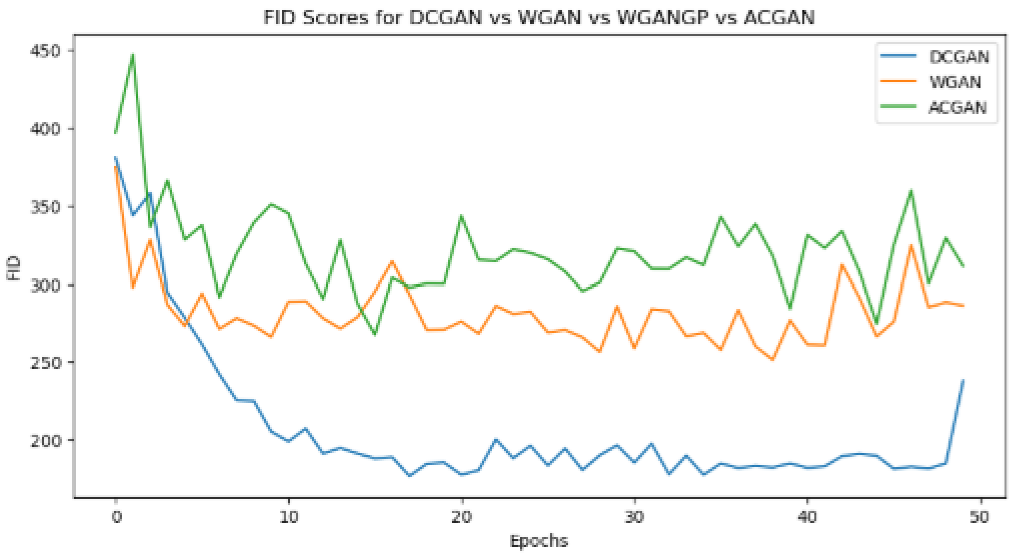


Evaluation

To compare the quality of the images produced by the different GAN variants, I use the **Fréchet Inception Distance (FID)**. FID measures the distance between the feature distributions of real images and generated images. These features are extracted using a pretrained Inception-v3 network.

Intuitively, FID compares the statistics (mean and covariance) of Inception features for real and fake image sets. A **lower FID score** means the generated images are closer to the real ones in this feature space (better quality and diversity), while a score of **0.0** would indicate that the two distributions are identical.

The figure below shows how the FID score evolves over 50 epochs for **DCGAN, WGAN, WGAN-GP, and ACGAN**, allowing a quantitative comparison of their performance on CIFAR-10.



Model	Min FID	Mean FID	FID @ epoch 50
DCGAN	176.77	207.19	237.86
WGAN	251.37	281.52	286.19
ACGAN	267.49	321.41	311.16