Homework 3: Training Generative Adversarial Networks

Rishabh Thapliyal
Electrical & Computer Engineering Department
UC San Diego
rthapliyal@ucsd.edu

1 Methodology

1.1 Model Architecture

I have utilized a Deep Convolutional Generative Adversarial Network (DCGAN) architecture, consisting of two main components: a generator and a discriminator. Both networks are implemented using convolutional layers, batch normalization, and non-linear activation functions.

1.1.1 Generator

The architecture begins with a fully connected layer that projects the latent vector into a low-resolution feature map. This is followed by a series of transposed convolutional layers that progressively upsample the feature map to the desired image size. Batch normalization and LeakyReLU activations are used throughout to promote stable training and improve gradient flow. The final layer uses a Tanh activation to produce output images with pixel values in the range $[-1, 1]$.

1.1.2 Discriminator

The discriminator acts as a binary classifier, distinguishing between real images from the dataset and fake images produced by the generator. It consists of a series of convolutional layers that downsample the input image, with **spectral normalization** and batch normalization applied to stabilize training. LeakyReLU activations are used throughout. The final output is a single value between 0 and 1, representing the probability that the input image is real.

Weight Initialization: All convolutional and batch normalization layers are initialized using a normal distribution, following DCGAN guidelines.

2 Data Preprocessing

I used the WikiArt dataset for model training using the following steps:

1. **Dataset Loading:** I loaded the WikiArt dataset using DeepLake's API, which allowed me to efficiently handle the large image collection.
2. **Image Processing:** I transformed each image to ensure consistency:
   - Each image is resized to 256×256 pixels to ensure uniform input size.
   - Converted PIL image to a PyTorch tensor.
   - The tensor values were normalized to have a mean of 0.5 and a standard deviation of 0.5, which helped stabilize and accelerate neural network training.
3. **Batch Preparation:** I grouped images into batches of 128 and shuffled them to introduce randomness. I applied the transformations to each image using a custom wrapper before batching.

4. **Storage:** I saved the processed batches as tensor files in a dedicated directory for later use.

This preprocessing pipeline standardized the dataset for efficient model training. By saving the processed tensors locally, I avoided repeated API calls during training and significantly reduced loading time.

3 Training Setup & Hyperparameters

I trained Deep Convolutional Generative Adversarial Network (DCGAN) model using PyTorch on preprocessed batches of the WikiArt dataset. The training process was designed to be robust, with support for checkpointing and resuming, as well as periodic saving of generated images and loss plots for monitoring progress.

The main hyperparameters used in training are summarized in Table 1. These values were chosen based on common practices for GAN training and adjusted for our dataset and computational resources.

Hyperparameter	Value
Epochs	500
Batch size	128
Generator learning rate	0.0002
Discriminator learning rate	0.0003
Adam β_1	0.5
Adam β_2	0.999
Latent dimension	256
Image size	256×256
Channels	3 (RGB)
Sample interval	500 batches
Parallel-processing (workers)	4

Table 1: Training Hyperparameters

4 Experimental Results

During the training of the DCGAN model, we monitored two primary loss metrics: the generator loss and the discriminator loss.

4.1 Loss Metrics and Loss Curves

- **Generator Loss:** This loss measures how well the generator is able to fool the discriminator. A lower generator loss indicates that the generator is producing images that are more likely to be classified as real by the discriminator.
- **Discriminator Loss:** This loss quantifies the discriminator's ability to distinguish between real images from the dataset and fake images produced by the generator. A lower discriminator loss means the discriminator is successfully identifying real and fake images.

Both losses were recorded at every training iteration and plotted to visualize the adversarial training process. During experimentation, both `torch.nn.BCELoss()` and `torch.nn.BCEWithLogitsLoss()` were considered for the adversarial loss function. Ultimately, `torch.nn.BCELoss()` was selected for the final training runs, as it provided stable and interpretable loss values for this setup.

Figure 1 shows the evolution of generator and discriminator losses over the course of training. Initially, both losses are relatively high as the networks begin to learn their respective tasks. As training progresses, the losses fluctuate, reflecting the adversarial nature of GAN training where the generator and discriminator are constantly trying to outsmart each other.

Figure 1: Generator and Discriminator Loss During Training

The generator loss (blue curve) and discriminator loss (orange curve) exhibit alternating peaks and valleys, which is typical in GAN training. These fluctuations indicate periods where one network temporarily outperforms the other before the balance shifts again. Despite the noise, both losses generally remain within a reasonable range, suggesting that neither network collapsed or dominated the training process.

5 Evaluation Metrics

To quantitatively assess the quality of images generated by my DCGAN, I employed two widely used evaluation metrics: the Fréchet Inception Distance (FID) and the Inception Score (IS). These metrics provide objective measures of both the realism and diversity of the generated images in comparison to real artwork from the WikiArt dataset.

5.1 Evaluation Procedure

The evaluation was performed as follows:

- The generator model was loaded from saved checkpoints at 100, 200, and 500 epochs.
- For each checkpoint, 1000 images were generated using the trained generator.
- A random sample of 1000 real images was selected from the WikiArt dataset for comparison.
- The FID and IS scores were computed between the generated images and the real images.

Epochs	Latent=128		Latent=256	
	FID	IS	FID	IS
100	332.61	3.28 ± 0.04	286.30	2.13 ± 0.11
200	206.53	3.78 ± 0.23	245.76	2.34 ± 0.05
500	194.78	3.52 ± 0.24	208.35	2.50 ± 0.08

Table 2: Model Evaluation Metrics Across Training Epochs (Image size= 256)

As shown in Table 2, both FID and IS scores improve as training progresses, indicating that the generator produces images that are increasingly similar to real artworks and more diverse over time.

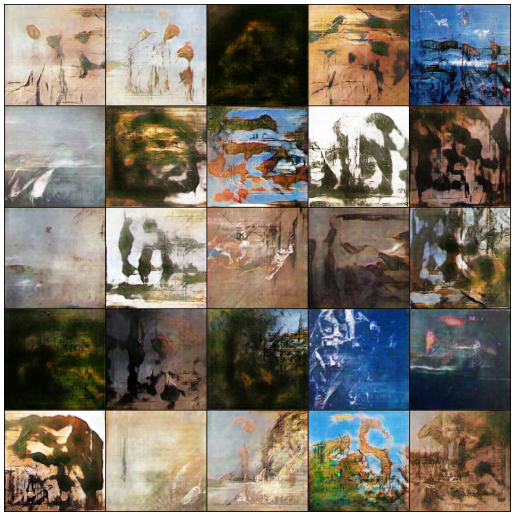
5.2 Generated Samples

Figure 2 displays 4 grids of 25 images generated by the DCGAN after training. The images exhibit a range of abstract patterns, color palettes, and brushstroke-like textures that are characteristic of the

(a) Epoch 100

(b) Epoch 200

(c) Epoch 400



(d) Epoch 500

Figure 2: Generated image samples at different training epochs using a DCGAN (image size = 256)

WikiArt dataset. As training progresses, the generated images become more visually coherent, with improved structure and diversity.

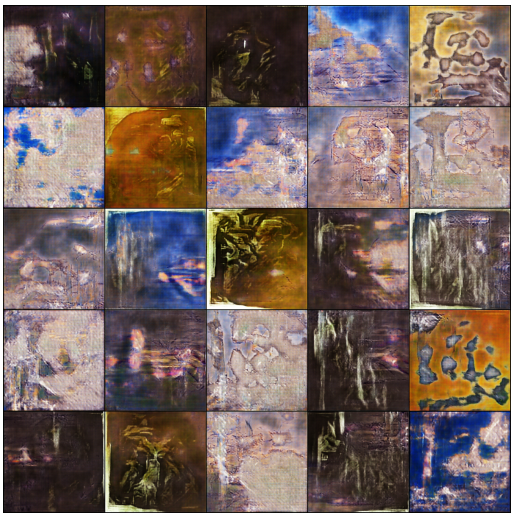
While some images still display artifacts or lack clear subject matter, many demonstrate convincing artistic qualities and stylistic variety. This suggests that the DCGAN has successfully learned to capture important aspects of the artistic domain, producing outputs that are both diverse and visually appealing.

6 Additional Experimental Results

Figure 3 displays some more generated images I obtained after training different GANs over varying batch sizes and image resolutions.

(a) Model: **GAN**, Batch size: 64, Latent dim: 128, Image size: 128

(b) Model: DCGAN, Batch size: 64, Latent dim: 128, Image size: 128



(c) Model: DCGAN, Batch size: 128, Latent dim: 128, Image size: 256

Figure 3: Comparison of generated images across different model architectures and training configurations.