GAN Reflection Report

Introduction

A Generative Adversarial Network (GAN) is a machine learning model where two networks, a generator and a discriminator, compete against each other. The generator learns to produce realistic images from random noise, while the discriminator tries to detect fake ones. Over time, the generator improves and produces outputs that resemble real-world data. In this project, I used a pretrained BigGAN model to generate synthetic images from latent vectors.

Experiment Summary

I explored three experiment types: random noise, manual edits, and slight variations.I fed the model a 128-dimensional latent vector (random noise) and a class vector corresponding to ImageNet class 207 (Golden Retriever). I explored three experiment types: random noise, manual edits, and slight variations.

The key techniques implemented were:

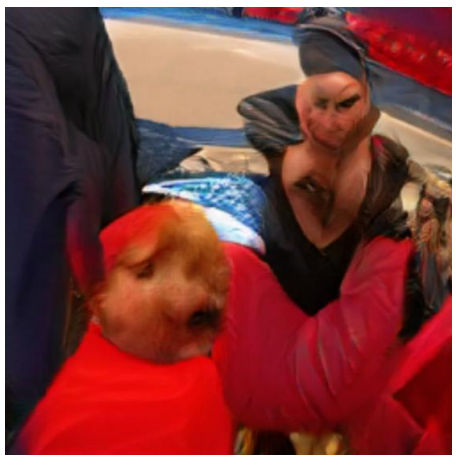
```
# Load pretrained BigGAN
model = BigGAN.from_pretrained('biggan-deep-256')

# Method 1: Truncated noise for stable generation
latent_vector = torch.from_numpy(truncated_noise_sample(truncation=0.4, batch_size=1))

class_vector = torch.from_numpy(one_hot_from_int([207], batch_size=1))  # Golden Retriever

# Method 2: Random latent vector for comparison
latent_vector = torch.randn(1, 128)

# Method 3: Manual latent vector editing
latent_vector = torch.randn(1, 128)
latent_vector[0][0] = 3.0      # Set extreme values
latent_vector[0][10] = -2.0
latent_vector[0][50] = 1.5

# Generate image
with torch.no_grad():
    generated_image = model(latent_vector, class_vector, truncation=0.4)
```

Observations

Image 1 – Truncated Latent Vector

This image was generated using a truncated latent vector. The output is realistic — the golden retriever's fur and body shape are recognizable. Although there are still minor artifacts (e.g., unclear toy object), the result strongly resembles a real-world image. Truncated vectors help stabilize generation and improve realism.

Image 2 – Random Latent Vector

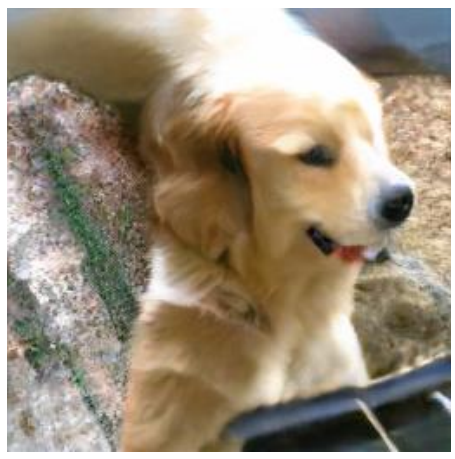


This image was generated using a random latent vector. While the class vector was set to "golden retriever," the result is heavily distorted. It appears to blend human-like figures with dog features. The background and textures are surreal and inconsistent. This demonstrates how unfiltered noise can lead to unpredictable and abstract outputs.

Image 3 – Random Latent Vector

This image was generated using a completely random latent vector. The result is highly realistic, capturing the facial structure, lighting, and fur texture of a golden retriever. Despite using an unfiltered random vector, the output happened to fall within a region of the latent space that the GAN understands well.

Image 4 – Random Latent Vector



This image displays a golden retriever in an outdoor setting, but some areas appear distorted — especially the tail and the background, which blend unnaturally. While the face and expression are mostly coherent, the image lacks full realism.

Image 5 – Manually Edited Latent Vector

This image was generated using a latent vector where I manually changed specific dimensions — setting the 0th, 10th, and 50th values to extreme values (3.0, -2.0, and 1.5). The resulting image shows a vaguely realistic golden retriever but includes strange textures, warped faces, and distorted text-like artifacts.

Reflection

Through these experiments, I observed that the GAN is highly sensitive to input noise. Using truncated noise helped produce more stable and realistic images, while raw noise often led to distorted or surreal outputs. Manually editing specific latent vector values also altered the structure or style of the generated images. I learned that small changes to the latent vector can lead to dramatic differences in image quality and composition. Challenges included understanding the effects of the latent space and interpreting distorted outputs. To improve this project, I could experiment with different truncation values and try other GAN models like StyleGAN for comparison. A more systematic approach to exploring which noise vectors work best would also help generate more consistent results.

See attached **gan_notebook.ipynb** for full code and outputs.