Lab 6 Report

Deep Learning

Name: Kai-Jie Lin
Student ID 110652019
June 14, 2024

1 Introduction

In this lab, I implemented Wasserstein GAN with Gradient Penalty (WGAN-GP) and Latent Diffusion Model (LDM) to generate images. The WGAN-GP is a variant of GAN that uses Wasserstein distance as the loss function and gradient penalty to enforce the Lipschitz constraint. The LDM is a type of generative model that combines the principles of diffusion models with latent variable models. I trained these two model on given datasets and got 0.8 score from evaluation model.

2 Implementation Details

2.1 WGAN-GP

WGAN-GP objective:
The objective of WGAN-GP is to minimize the Wasserstein distance between the real data distribution p_r and the generated data distribution p_g. To ensure that the critic function satisfies the Lipschitz constraint (a requirement for the Wasserstein distance), WGAN-GP introduces a gradient penalty. This replaces the weight clipping method used in the original WGAN. The objective function of WGAN-GP is given by:

$$\min_G \max_D \mathbb{E}_{x\sim p_r}[D(x)] - E_{z\sim p_z}[D(G(z))] + \lambda E_{\hat{x}\sim p_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{1}$$

where G is the generator, D is the critic, p_r is the real data distribution, p_z is the noise distribution, $\hat{x}$ is a sample from the distribution between real and generated data, and λ is the gradient penalty coefficient.
Generator Details:
The generator is composed of four residual blocks. Each residual block consists of two convolutional layer, batch normalization, and ReLU activation. We input a 64 dim noise and concated with one-hot encoding label condition. The output of the generator is a 3x64x64 tensor representing an RGB image.
Discriminator Details:
I implemented a projection discriminator to handle the relation between images and condition labels. The discriminator is also composed of four residual blocks. While the residual block of generator do upsampling, the residual block of discriminator do downsampling. We first use residual blocks to downsample the input image into low dimensional hidden features. Then, we encode the label with linear layer and multipy it with hidden features. Finally, we sum the value in the vector and output the critic value of given image and label.

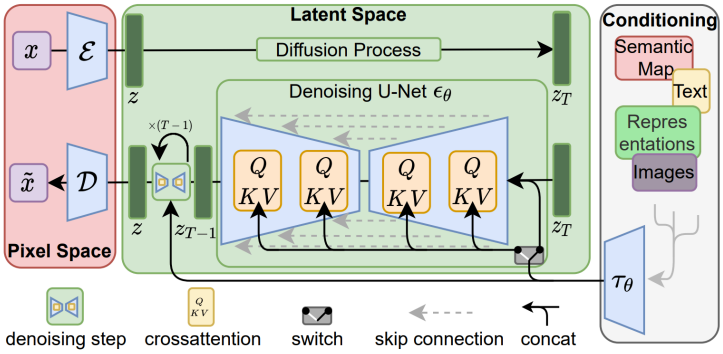
The architecture of the generator and discriminator is shown in the figure below.

```python
class Generator(nn.Module):
    def __init__(self, dim=64):
        super(Generator, self).__init__()
        self.dim = dim
        self.ln1 = nn.Linear(self.dim + 24, self.dim * 4 * 4 * 8)
        self.rb1 = ResidualBlock(8*self.dim, 8*self.dim, 3, resample = 'up')
        self.rb2 = ResidualBlock(8*self.dim, 4*self.dim, 3, resample = 'up')
        self.rb3 = ResidualBlock(4*self.dim, 2*self.dim, 3, resample = 'up')
        self.rb4 = ResidualBlock(2*self.dim, 1*self.dim, 3, resample = 'up')
        self.bn  = nn.BatchNorm2d(self.dim)

        self.conv1 = MyConvo2d(1*self.dim, 3, 3)
        self.relu = nn.ReLU()
        self.tanh = nn.Tanh()

    def forward(self, noise, label):
        input = torch.cat((noise, label), 1)
        output = self.ln1(input)
        output = output.view(-1, 8*self.dim, 4, 4)
        output = self.rb1(output)
        output = self.rb2(output)
        output = self.rb3(output)
        output = self.rb4(output)
        output = self.bn(output)
        output = self.relu(output)
        output = self.conv1(output)
        output = self.tanh(output)
        return output
```

```python
class Discriminator(nn.Module):
    def __init__(self, dim=64):
        super(Discriminator, self).__init__()
        self.dim = dim
        self.conv1 = MyConvo2d(3, self.dim, 3)
        self.rb1 = ResidualBlock(self.dim, 2*self.dim, 3, resample = 'down', hw=self.dim)
        self.rb2 = ResidualBlock(2*self.dim, 4*self.dim, 3, resample = 'down', hw=int(self.dim/2))
        self.rb3 = ResidualBlock(4*self.dim, 8*self.dim, 3, resample = 'down', hw=int(self.dim/4))
        self.rb4 = ResidualBlock(8*self.dim, 8*self.dim, 3, resample = 'down', hw=int(self.dim/8))
        self.ln1 = nn.Linear(8*self.dim * 4 * 4, 1)
        self.lab_enc = nn.Linear(24, 8*self.dim * 4 * 4)
        self.relu = nn.ReLU()

    def forward(self, input, label):
        output = self.conv1(input.contiguous())
        output = self.rb1(output)
        output = self.rb2(output)
        output = self.rb3(output)
        output = self.rb4(output)
        output = self.relu(output)
        h = output.view(-1, 8*self.dim * 4 * 4)
        output = self.ln1(h)
        label = self.lab_enc(label)
        output += (h * label).sum(1, keepdim=True)
        return output.view(-1)
```

2.2 Latent Diffusion Model

The architecture of the LDM is shown in the figure below.



It mainly consists of two parts: variational autoencoder and UNet. In the forward pass of diffusion process, we first encode the image into latent representation with variational autoencoder. Then, we gradually add noise to the latent representation. In the denoising process, we use UNet and VAE decoder to recover the original image from the noisy latent representation. The objective function of LDM is given by:

$$\mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,1),t}[\|\epsilon - \epsilon_\theta(z_t, c, t)\|_2^2] \tag{2}$$

where $\mathcal{E}(x)$ is the encoder, ϵ is the noise, $\epsilon_\theta(z_t, c, t)$ is the unet model, z_t is the latent representation, c is the condition label, and t is the time step.

VAE Details:

To get a good generation result, I implemented vector quantized variational autoencoder (VQ-VAE). The encoder is composed of three downsampling residual blocks. The decoder is composed

of three upsampling residual blocks. Each residual block consists of two convolutional layer, batch normalization, and ReLU activation. The output of the encoder is a 6x8x8 tensor representing the latent representation. I used a codebook of size 8192 to quantize the latent representation. The loss function is consists of reconstruction loss, codebook loss, and VQGAN loss. This ensures that the reconstructions are confined to the image manifold by enforcing local realism and avoids bluriness introduced by relying solely on pixel-space losses such as L2 or L1 objectives.
The loss function is given by:

$$\mathcal{L}_{\text{recon}} = \|x - \text{Decoder}(\text{Encoder}(x))\|_2^2 \tag{3}$$

$$\mathcal{L}_{\text{G}} = -\text{Discriminator}(\text{Decoder}(\text{Encoder}(x))) \tag{4}$$

$$\mathcal{L}_{\text{codebook}} = \|z_e - \text{Embedding}(\text{Quantize}(z_e))\|_2^2 \tag{5}$$

$$\mathcal{L}_{\text{D}} = \text{Discriminator}(\text{Decoder}(\text{Encoder}(x))) - \text{Discriminator}(x) \tag{6}$$

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{G}} + \mathcal{L}_{\text{D}} + \mathcal{L}_{\text{codebook}} \tag{7}$$

UNet Details:
I implemented a time-conditioned UNet as denoising model, augmented with the cross-attention mechanism to handle flexible conditioning information for image generation. The Unet is composed of encoder, decoder and middle block. The encoder is composed of six residual blocks and two spatial transformers. The decoder is also composed of six residual blocks and two spatial transformers. The middle block is composed of two residual blocks and one spatial transformer. The spatial transformer is used to align the latent representation with the condition label. The cross-attention mechanism is used to handle the relation between latent representation and condition label.
DDIM Sampler Details:
In sampling process, I used Denoising Diffusion Implicit Model (DDIM) to generate images. DDIM has the same marginal noise distribution but deterministically maps noise back to the original data samples. During generation, we don't have to follow the whole chain $t = 1, ..., T$, but rather a subset of steps. Let's denote $s < T$ as two steps in this accelerated trajectory. The DDIM update step is:

$$q_{\sigma,s<t}(z_s|z_t, z_0) = \mathcal{N}(z_s; \sqrt{\bar{\alpha}_s}(\frac{z_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(z_t)}{\sqrt{\bar{\alpha}_t}} + \sqrt{1-\bar{\alpha}_t - \sigma_t^2}\epsilon_\theta(z_t), \sigma_s^2\text{I}) \tag{8}$$

We can get final sample x_0 with VAE decoder: $x_0 = \text{Decoder}(z_0)$. DDIM can produce the good quality samples when s is small. With DDIM, it is possible to train the diffusion model up to any arbitrary number of forward steps but only sample from a subset of steps in the generative process.

3 Results and Discussions

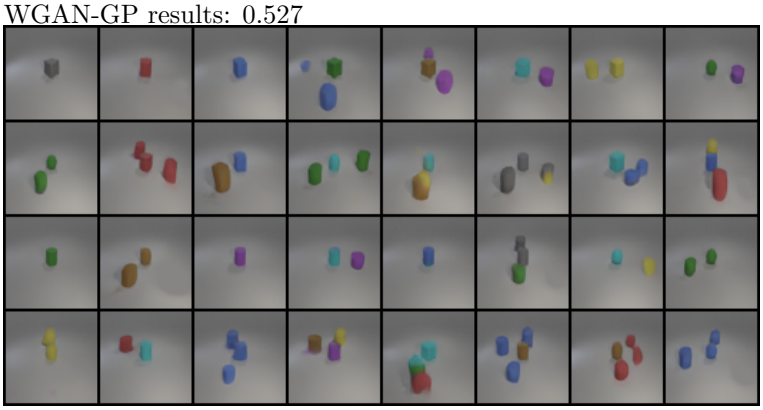
3.1 Synthetic Image Grids and Denoising Process

Hyperparameters settings:
WGAN-GP: Learning rate = 0.0003, Batch size = 64, Adam optimizer, critic iteration = 5, gp λ = 10, train epoch = 500.
VAE for LDM: Learning rate = 0.0001, Batch size = 64, Adam optimizer, codebook size = 8192, embedding dim = 6, train epoch = 300.

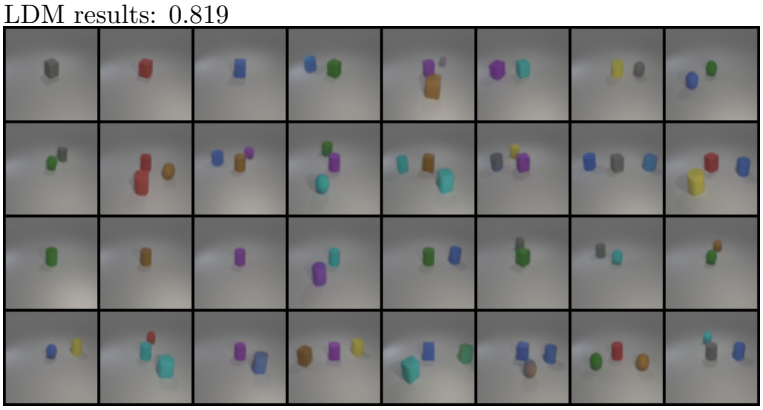
LDM: Learning rate = 0.000003, Batch size = 32, Adam optimizer, transformer layers = 1, attention heads = 8, train epoch = 1000.
DDIM: β scheduler = cosine, number of ddim steps = 50.

Test1

WGAN-GP results: 0.527



LDM results: 0.819

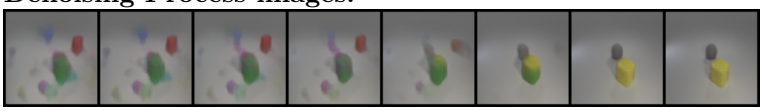


Test2

WGAN-GP results: 0.678

LDM results: 0.821

Denoising Process images:



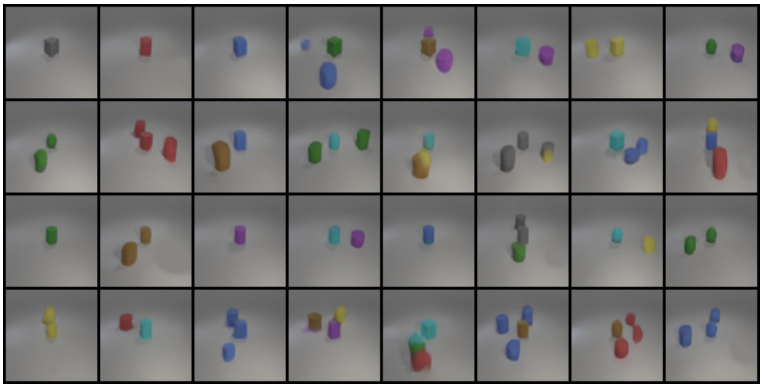
We can observe that the biginning images are not like noise. This is because we sample the noisy data in latent space and use VAE decoder to generate the image.

3.2 Compare GAN and Diffusion Models

GAN:
Training: Faster but unstable and sensitive to hyperparameters.
Inferencing: Fast with potentially inconsistent quality.
Computing Resources: Generally efficient but resource-intensive during unstable training phases.
Generation Quality: High quality but prone to mode collapse and artifacts.
DDPM:
Training: Stable but slow and complex.
Inferencing: Slow with consistent high quality.
Computing Resources: Resource-intensive due to iterative nature.
Generation Quality: High fidelity and diversity with fewer artifacts.

3.3 Extra experiments

Classifier Guidance:
I tried to include classifier guidance into my DDIM, but the results are worse than original. I think the reason is that the classifier need to classify the image with noise, which is hard to get a good result.
Test1: 0.75



Test2: 0.78