Lab7 Let's play DDPM

311551040

邱以中

1. Introduction

這次作業我們的目標是使用 conditional 的 diffusion model 來生成 iclevr dataset 的圖片，給定多個物體的描述(包括形狀、顏色)，生成對應描述的圖片。

2. Implementation details

- Describe how you implement your model, including your choice of DDPM, UNet architectures, noise schedule, and loss functions

  (1) 這次作業我使用的 DDPM 是 pixel space 的

  (2) 使用的 Unet 為 diffusers 套件中的 UNet2DModel

  具體架構如下

```
from diffusers import DDPMScheduler, UNet2DModel
```

```
self.model = UNet2DModel(
    sample_size = 64,
    in_channels = 3,
    out_channels = 3,
    layers_per_block = 2,
    class_embed_type = None,
    #num_class_embeds = 2325, #C (24, 3) + 1
    block_out_channels = (128, 128, 256, 256, 512, 512),
    down_block_types=(
    "DownBlock2D",  # a regular ResNet downsampling block
    "DownBlock2D",
    "DownBlock2D",
    "DownBlock2D",
    "AttnDownBlock2D",  # a ResNet downsampling block with spatial self-attenti
    "DownBlock2D",
    ),
    up_block_types=(
        "UpBlock2D",  # a regular ResNet upsampling block
        "AttnUpBlock2D",  # a ResNet upsampling block with spatial self-attenti
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
    ),
)
'''embedding'''
self.model.class_embedding = nn.Linear(24 ,class_emb_size)
```

  因為輸入是一個 one hot encoding，所以我改寫了 UNet2DModel 的 class_embedding，將它改成一個 linear 的網路

  (3) Noise schedule 則是使用 diffusers 套件中的 DDPMScheduler

Noise 步數設為 1000

```
12    from diffusers import DDPMScheduler, UNet2DModel
```

```
noise_scheduler = DDPMScheduler(num_train_timesteps=1000, beta_schedule='squaredcos_cap_v2')
```

(4) Loss function

loss function 則是使用 MSE loss

```
mse = nn.MSELoss()
```

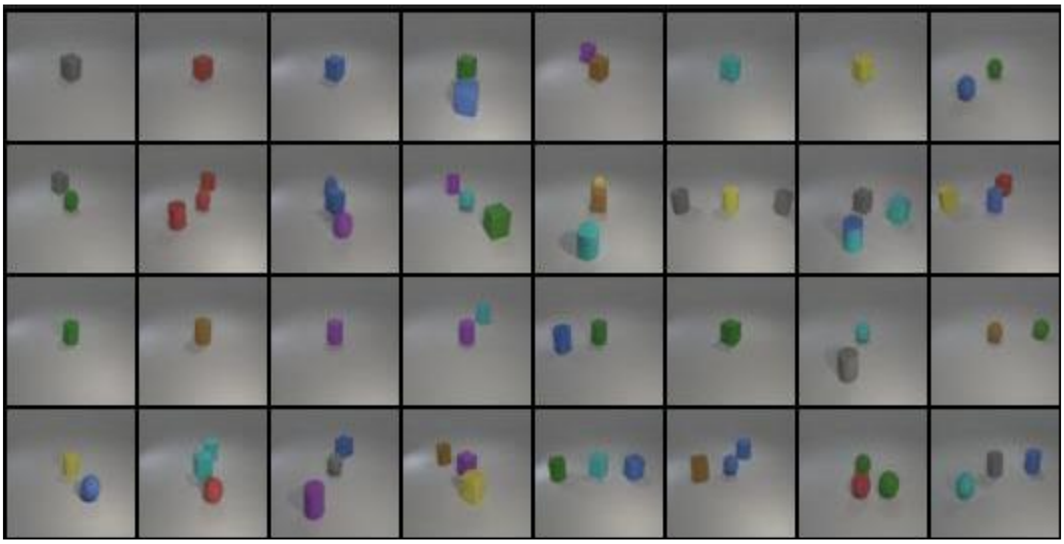
```
loss = mse(noise, predicted_noise)
```

- Specify the hyperparameters (learning rate, epochs, etc.)
  - Batch size: 32
  - Lr: 0.0001
  - Epoch: 40

3. Results and discussion

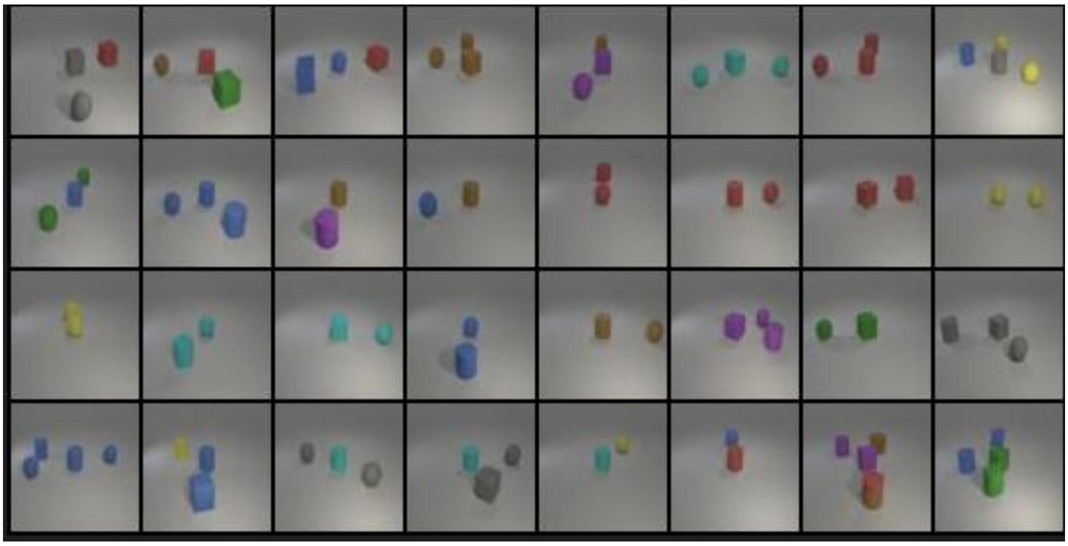
- **Show your results based on the testing data**
  - **Test**

```
(DL) icchiu@VLLab_24_146:~/NYCU_2023_ML/Lab7$ python ddpm.py --mode test
  0%|                                                                  | 0/1 [
00:00<?, ?it/s]05:13:57 - INFO: Sampling 32 new images....
1000it [01:35, 10.49it/s][[C
100%|██████████████████████████████████████████████████████████████████| 1/1 [01:36<00
:00, 96.07s/it]
accuracy:0.8333333333333334
```



  - **New_test**

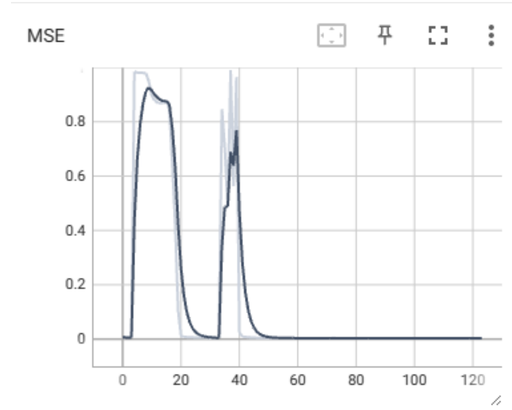
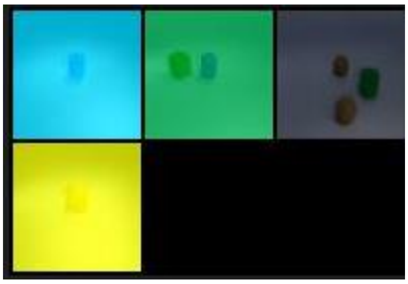
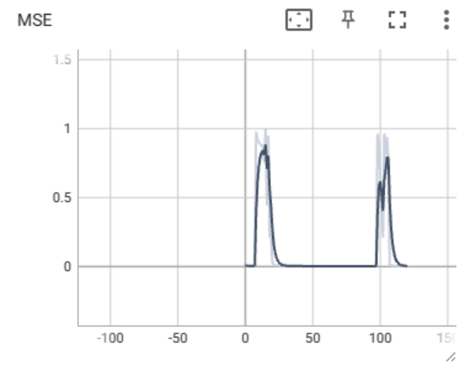
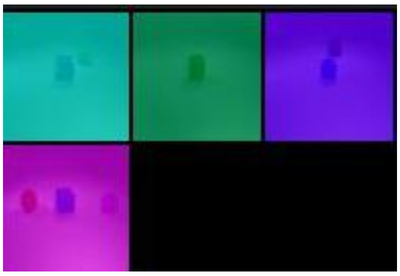
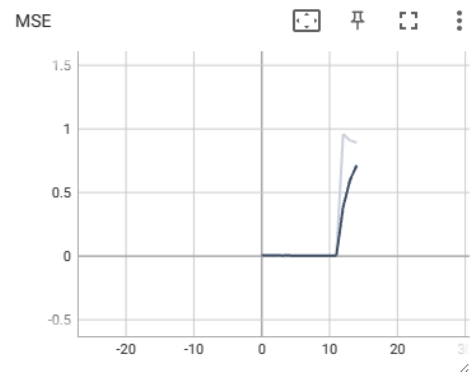
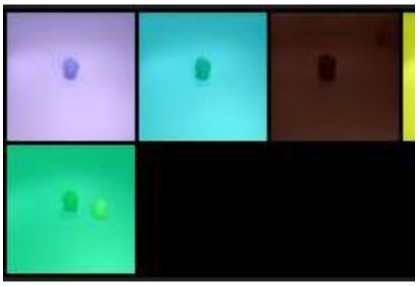
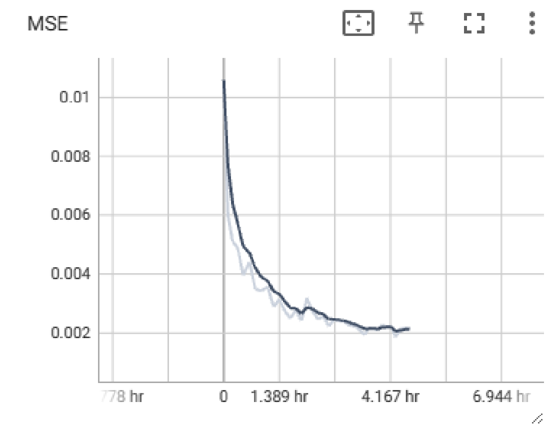
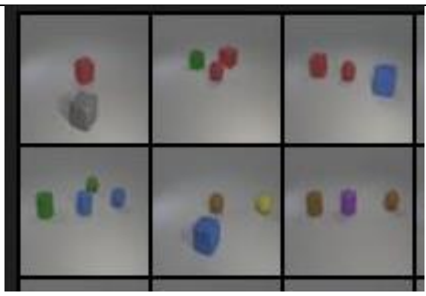
```
[redacted]6:~/NYCU_2023_ML/Lab7$ python ddpm.py --mode new_test
  0%|                                                                    | 0/1 [00:00<?, ?it/s]
03:58:04 - INFO: Sampling 32 new images....
1000it [02:54,  5.73it/s]
100%|██████████████████████████████████████████████████| 1/1 [02:55<00:00, 175.28s/it]
accuracy:0.8095238095238095
```



- **Discuss the results of different model architectures.**

  這次作業我一開始是選用一個比較小的 Unet 架構，架構圖如下，並嘗試使用不同加入 condition 的方法，但是結果都不太理想，最後發現可能是模型太小，無法很好的學習，最後加大 model 才讓效果提升，以下是我做的一些實驗。

```
self.model = UNet2DModel(
    sample_size=64,           # the target image resolution
    # in_channels=3 + num_classes, # direct
    # in_channels=3 + class_emb_size, # linear
    in_channels=3, # embedding
    out_channels=3,           # the number of output channels
    class_embed_type = None,
    layers_per_block=2,       # how many ResNet layers to use per UNet
    #small
    block_out_channels=(32, 64, 64),
    down_block_types=(
        "DownBlock2D",
        "AttnDownBlock2D",    # a ResNet downsampling block with spati
        "AttnDownBlock2D",
    ),
    up_block_types=(
        "AttnUpBlock2D",
        "AttnUpBlock2D",      # a ResNet upsampling block with spatial
        "UpBlock2D",         # a regular ResNet upsampling block
    ),
)
```

	MSE Loss	Result
小 model + 將 condition 經過 linear layer concate 在 xt 上		
小 model + 將 condition 直接 concate 在 xt 上		
小 model + 改寫 UNet2DModel 的 class_embedding		
大 model + 改寫 UNet2DModel 的 class_embedding		

在上述表格中可以發現使用小 model 進行生成，都會導致 loss 的不穩定，以及背景會五顏六色，而使用較大的 model 就能很好的解決這些問題，因此最後使用較大的 model 來進行訓練以及生成。