NYCU DLP Lab6 – Let's Play DDPM

312551086 張紀睿

i. Introduction

本次作業中，實作了 conditional Denoising Diffusion Probabilistic Model，根據給定的 label 生成指定的圖片。首先，根據資料集的 json 格式設計 Dataloader 以載入訓練影像與標籤並進行 one-hot encoding。之後，再選定 conditional DDPM 的設定，並設計 noise schedule 以及 UNet 之模型架構，進行模型訓練。訓練完畢後，使用 Evaluator 評估生成圖像之好壞，目標達到 80%的 Accuracy。

ii. Implementation details

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

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

我選擇在 label 層面進行 diffusion 與 denoising，使用 DDPMScheduler[1] 幫助進行 noise schedule，並選用了 cosine scheduling 的方式。

```
class ClassConditionedUnet(nn.Module):
    def __init__(self, num_classes=24, class_emb_size=128,
    blocks = [0, 1, 1], channels = [1, 2, 2]): #Default setting as tutorial
        super().__init__()
        first_channel = class_emb_size//4
        down_blocks = ["DownBlock2D" if x == 0 else "AttnDownBlock2D" for x in blocks]
        up_blocks = ["UpBlock2D" if x == 0 else "AttnUpBlock2D" for x in reversed(blocks)]
        channels = [first_channel * x for x in channels]
        self.model = UNet2DModel(
            sample_size = 64,
            in_channels = 3,
            out_channels = 3,
            layers_per_block = 2,
            block_out_channels = (channels),
            down_block_types=(down_blocks),
            up_block_types=(up_blocks),
        )
        self.model.class_embedding = nn.Linear(num_classes, class_emb_size)

    def forward(self, x, t, label):
        return self.model(x, t, label).sample
```

UNet 模型方面，我使用了 UNet2DModel[2]作為模型，並加入了 class embedding 使其能輸入 label 一同學習。在 UNet2DModel 的 class 中，會將 label 經過 class_embedding 以後，與 time_embedding 之結果相加，以取

得在 label 與 timestep 上的資訊。我在外層寫的 class 可以在 initialize 時根據想要的模型架構輸入對應的 list 來建構模型，blocks 用以設定一般的 block 或 attention block，channel 則用來設定每層的 channel 數量，會根據第一層的數量乘上倍數，第一層的 channel 數必須是 class_embed_size 的 1/4，而 Default 的設定值是在 diffuser 提供的 tutorial unit2 [3]中使用的參數，非我實際訓練時所使用的，詳細設定會在 hyperparameters 段落介紹。

```
print('Start_Training')
for epoch in range(cur_epoch+1, n_epochs):
    train_loss = []
    for x, label in tqdm(train_loader):

        x, label = x.to(device), label.to(device)
        label = label.squeeze(1)
        noise = torch.randn_like(x)
        timesteps = torch.randint(0, 999, (x.shape[0],)).long().to(device)
        noisy_x = noise_scheduler.add_noise(x, noise, timesteps)
        output = model(noisy_x, timesteps, label)

        loss = loss_function(output, noise)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss.append(loss.item())
```

模型訓練時，會先隨機產生 noise 並抽取 timestep，透過 noise_scheduler 將原圖根據 timestep 添加一定程度的 noise，再將 noisy_x, timestep 與 label 一起輸入模型預測 noise，並透過 MSELoss 計算 Loss。

```
for y in test_loader:
    y = y.to(device)
    x = torch.randn(1, 3, 64, 64).to(device)
    for i, t in tqdm(enumerate(noise_scheduler.timesteps)):

        with torch.no_grad():
            y = y.squeeze(1)
            residual = model(x, t, y)

        x = noise_scheduler.step(residual, t, x).prev_sample
        if count == 0 and i % 120 == 0:
            generating.append(x.detach().cpu().squeeze(0))

    accuracy = eval_model.eval(x, y)
    accuracys.append(accuracy)
    print('image', count, ':', accuracy)
```

Testing 階段，則是先產生 noise，並由最後的 timestep 一步步向前還原，以獲得產生的圖片，再將圖片與其 label 輸入 evaluation model 以取得 Accuracy。

2. Specify the hyperparameters (learning rate, epochs, etc.)

在訓練中，我使用以下設定

Batch_size: 8

Loss Function: nn.MSELoss()

Optimizer: Adam

Learning rate: 1e-5

DDPMScheduler timesteps: 1000

Beta_schedule: squaredcos_cap_v2

Epoch: 69 (用此 checkpoint 取得 Result and Discussion 章節展示之結果)

UNet Config:

- Class_emb_size: 512
- Blocks: [0, 0, 0, 0, 0, 0] (6 DownBlocks and 6 UpBlocks without attention)
- Channels: [1, 1, 2, 2, 4, 4] (128, 128, 256, 256, 512, 512)

iii. Results and discussion

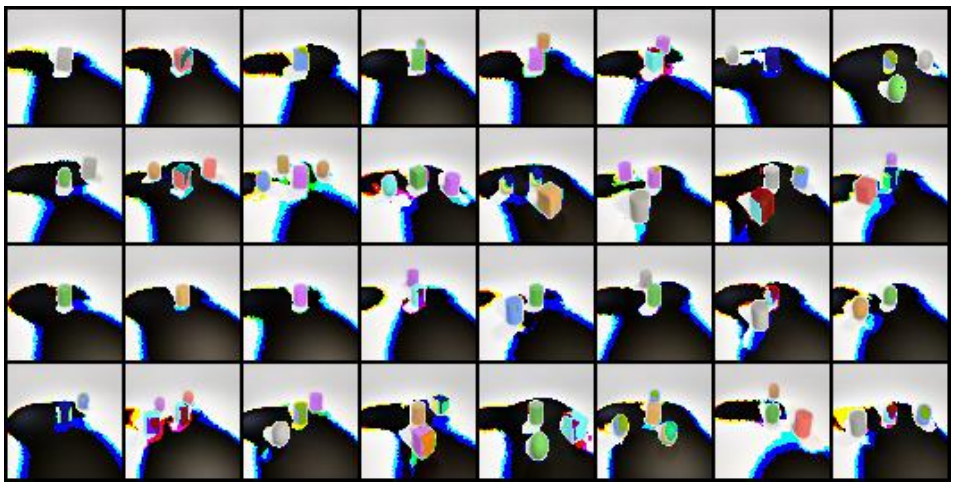
1. Show your accuracy screenshot based on the testing data.

```
Test Accuracy:
0.8177083333333334
```

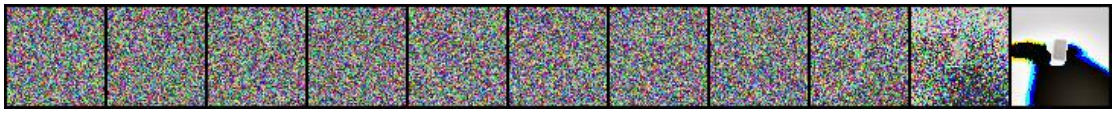
```
New Test Accuracy:
0.8229166666666669
```

2. Show your synthetic image grids and a progressive generation image.

Test Result



Test image 0 generating



New Test Result

New Test image 0 generating

雖然生成圖片的底下會有奇怪的黑底，但由於生成的圖形與 label 符合，因此仍然達到了 81%以上的 Accuracy。

3. Discuss the results of different model architectures or methods.

a. 在 class embedding 上，除了使用上述將其與 time embedding 相加的方式外，我也曾嘗試 diffuser 提供的 tutorial unit2 [3] 中，直接將 one_hot label 轉換 class_embed_size x 64 x 64 並直接 concat 到 x 輸入 model 的方式來訓練，雖然某些生成之圖片質量看起來比較好，但有許多圖片背景顏色會有干擾，且大部分的圖形 label 都不正確，因此 accuracy 反而更差，以下為範例之輸出:

b. 在 prediction type 上，我嘗試過預測 noisy sample 而非 noise，然而即使 train 了超過 20 epoch，模型在 test 時仍然只能 generate 出一堆雜訊，因此從結果來看，predict noise 更適合此次的任務。

iv. Reference

[1] DDPMScheduler:
https://huggingface.co/docs/diffusers/api/schedulers/ddpm
[2] UNet2DModel:
https://huggingface.co/docs/diffusers/api/models/unet2d
[3] HuggingFace Diffuser Tutorial Unit2:
https://github.com/huggingface/diffusion-models-class/tree/main/unit2