

NYCU DL Lab6
Generative Models
313552049 鄭博涵

Report:

1. Introduction

在這次 lab 中，我們需要實作 Diffusion model，在評估階段時，我們使用一個預訓練的 ResNet-18 模型來計算生成圖像的準確率，本次 lab 的目的是盡可能提升 accuracy，而經過實驗後，我測出來的 accuracy 是 0.8073 跟 0.8594 (test.json and new_test.json)

2. Implementation Details

A. Describe how you implement your model, including your choice of DDPM, noise schedule

DDPM:

```
< class ConditionalDDPM(nn.Module):
>     def __init__(self, num_classes=24, dim=512):
>         super(ConditionalDDPM, self).__init__()
>         channel = dim // 4
>
>         self.ddpm = UNet2DModel(
>             sample_size=64,    # 輸入影像大小為64 x 64
>             in_channels=3,   # RGB圖像
>             out_channels=3,
>             layers_per_block=2, # 每個block中包含2層
>
>             # 設定每個 block 中的通道數量，依次為 channel, channel, channel*2, channel*2, channel*4, channel*4
>             block_out_channels=(channel, channel, channel * 2, channel * 2, channel * 4),
>             # 定義下採樣 block 的類型，包含多個 DownBlock2D 和一個 AttnDownBlock2D
>             down_block_types=("DownBlock2D", "DownBlock2D", "DownBlock2D", "DownBlock2D", "AttnDownBlock2D"),
>
>             up_block_types=("AttnUpBlock2D", "UpBlock2D", "UpBlock2D", "UpBlock2D", "UpBlock2D"),
>             # class_embed_type="projection",
>             class_embed_type="identity",      # 設定嵌入類型，表示直接使用輸入的class embedding
>         )
>
>         self.class_embedding = nn.Sequential(
>             nn.Linear(num_classes, dim),
>             nn.ReLU(),
>             nn.BatchNorm1d(dim)       # 對輸出進行正規化
>         )
>
```

在上述程式中，設計了 Conditional Diffusion Model，而內部的主要

架構是 UNet2DModel (從 diffuser 引入進來)，其配置為：

Sample_size: 64, 代表說生成圖像的尺寸為 64 x 64

in_channels: 3, 表示輸入的圖像為 RGB 圖像

out_channels: 3, 表示輸出的通道數為 3

layers_per_block: 2, 表示每個 block 包含兩個卷積層

block_out_channels: 設置一個隨著層數增加的 channels 列表，確保 model 在更深的層數中有更高的特徵表達能力

down_block_types 以及 up_block_types: 分別定義了下採樣和上採樣的類型，採用了包括注意力機制的 block 類型來提升 model 的性能，並且將 class_embed_type 設為 identity，表示使用直接輸入的 class_embedding 進行處理

Forwarding:

```
def forward(self, x, t, label):
    # 計算class embedding
    class_embed = self.class_embedding(label)    # 將類別標籤轉換為dim的向量
    return self.ddpm(x, t, class_embed).sample    # 使用ddpm model進行forward pass，並返回生成的樣本
```

首先根據輸入的類別標籤計算對應的 class_embedding，接著將計算得到的嵌入向量作為條件，與 model 的其他輸入(image and 時間步長)一起傳入 UNet model，進行 forward pass，最終生成重建的樣本

Noise schedule:

```
self.noise_scheduler = DDPMscheduler(num_train_timesteps=self.total_timesteps, beta_schedule=args.beta_schedule)
```

我使用了 DDPMscheduler 來設定 noise schedule，而我們的參數設定是 total_timestep=1000，這表示 model 將從純隨機噪音開始，經過 1000 步的逐漸去噪，最終生成清晰的影像

而 `beta_schedule` 參數則是選用了 `squaredcos_cap_v2` 這種日程表，這個日程表使用了修正後的 `cos` 函數來控制噪音的增加速度使得在 `train` 的初期和後期，噪音變化更為平滑

```
timesteps = self.get_random_timesteps(batch_size)
timesteps = timesteps.squeeze() # 確保 timesteps 是 1 維數組

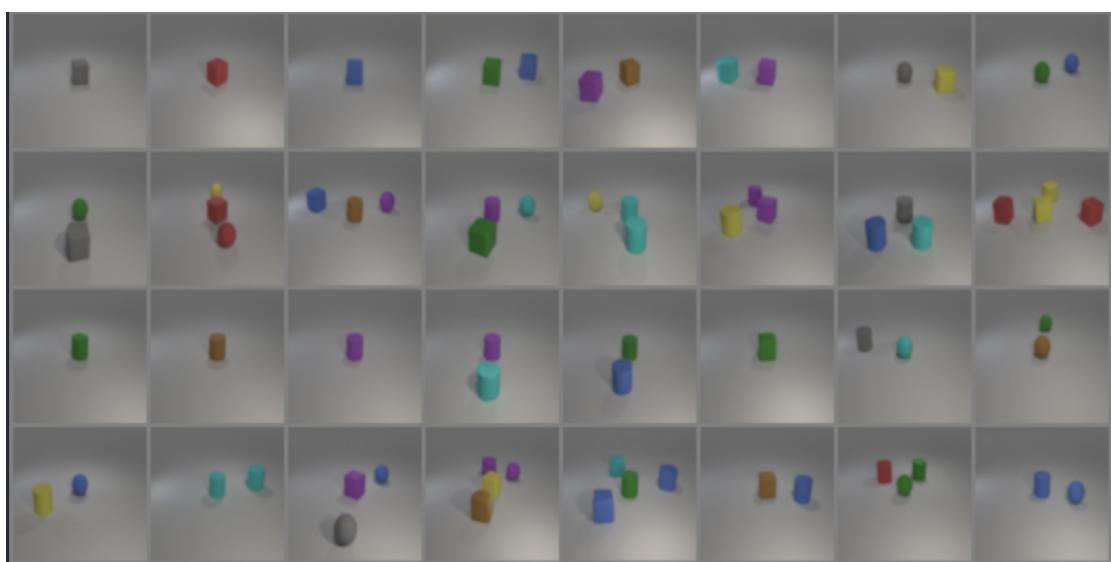
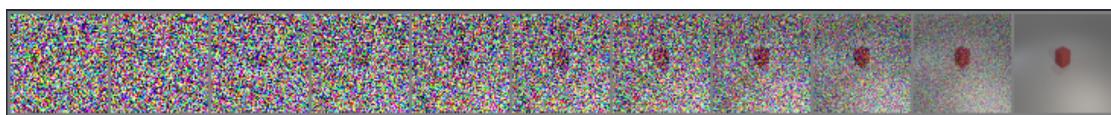
noisy_x = self.noise_scheduler.add_noise(x, noise, timesteps) # 經過noise後的圖像
```

在 `model` 訓練的每一步中，隨機選擇一個時間步長 t ，並根據噪音日程將噪音添加到輸入圖像數據上，最後生成了帶有噪音的圖像，以作為 `model` 的輸入，這個設計可以使 `model` 學習在不同的時間點處理噪音

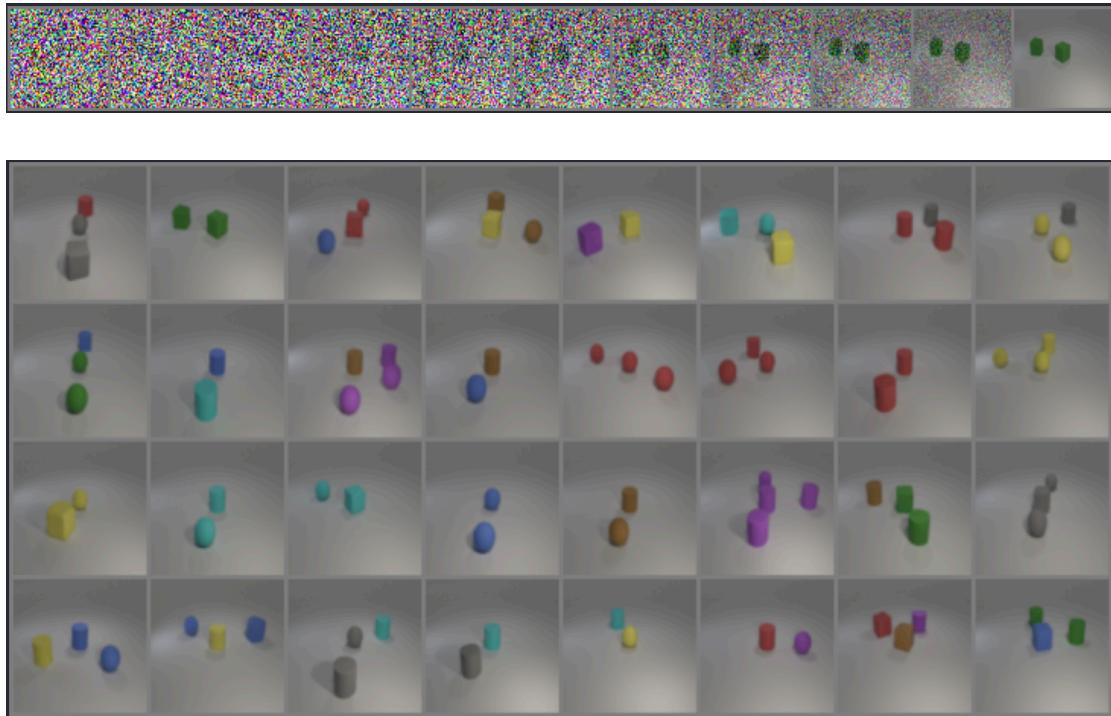
3. Results and Discussion

- A. Show your synthetic image grids and a denoising process image

Test:



New_test:



B. Discussion of your extra implementations or experiments

```
""" 保存model跟optim的檢查點 """
def save_checkpoint(self, epoch):
    save_dir = os.path.join(self.args.save_dir, f"checkpoint_{epoch}.pth")
    torch.save({
        'model': self.model.state_dict(),
        'optimizer': self.optimizer.state_dict(),
        'scheduler': self.scheduler.state_dict(),
        'epoch': epoch
    }, save_dir)

def load_checkpoint(self, path):
    checkpoint = torch.load(path, weights_only=True)
    self.model.load_state_dict(checkpoint['model'])
    self.optimizer.load_state_dict(checkpoint['optimizer'])
    if 'scheduler' in checkpoint:
        self.scheduler.load_state_dict(checkpoint['scheduler'])
    return checkpoint.get('epoch', 20)
```

我額外設計了 model 的檢查點保存與加載功能 (save_checkpoint 和 load_checkpoint), 這樣才可以在之前 train 過的 model 上繼續 train 就不用重頭 train

```
self.fast_train_loader = DataLoader(iclevrDataset(args.dataset, mode='train', partial=0.3), batch_size=args.batch_size, shuffle=True, num_workers=args.num_workers)
```

```
""" 設置fast_train"""
def fast_train(self):
    for epoch in range(1, self.args.num_fast_epochs + 1):
        train_loss = self.train_one_epoch(epoch, self.fast_train_loader)
        current_lr = self.scheduler.get_last_lr()[0] # 獲取調度器最近設置的學習率
        print(f"Epoch {epoch}/{self.total_epoch} | Train Loss: {train_loss:.4f} | lr: {current_lr:.7f}")
```

此外，我有用到 lab4 當中的 fast_train 技巧，除了可以加快 train 的時間之外，我一開始 learning rate 會設比較大，是 0.0007，好讓他一開始能快速收斂，並搭配著 lr scheduler 把 lr 降低一些些，在第一個 epoch 時，便能將 loss 降到 0.0425，到第 20epoch 時，loss 在 0.002 左右，這時我有用 checkpoint 把他先記下來，然後從 checkpoint 模型開始 train 時，因為這時 loss 已經很小了，所以直接調整我的 lr 到 0.000001，讓他 loss 降到 0.0016 左右，就可以在 test 和 new_test 達到 accuracy 0.8 以上的效果。

Optimizer:

```
def choose_optim(self):
    # optimizer = torch.optim.SGD(self.model.parameters(), lr=self.learning_rate, momentum=0.8, weight_decay=1e-4)      # bad
    # optimizer = torch.optim.Adam(self.model.parameters(), lr=self.learning_rate) # bad
    # optimizer = torch.optim.RMSprop(self.model.parameters(), lr=self.learning_rate, alpha=0.99)   # 對於噪聲梯度的問題 處理佳 bad
    # optimizer = torch.optim.Adamax(self.model.parameters(), lr=self.learning_rate) # bad
    optimizer = torch.optim.AdamW(self.model.parameters(), lr=self.learning_rate, betas=(0.9, 0.95))    # good
    # optimizer = torch.optim.ASGD(self.model.parameters(), lr=self.learning_rate, lambd=0.0001, alpha=0.75, t0=1000000.0) # bad
    # optimizer = torch.optim.Adagrad(self.model.parameters(), lr=self.learning_rate, weight_decay=1e-4)   # good
    # optimizer = torch.optim.Adadelta(self.model.parameters(), rho=0.9, eps=1e-6, weight_decay=1e-4)   # good
    return optimizer
```

另外我在選 optimizer 方面試了非常多種，其中最適合這次 lab 的 optimizer 是 AdamW，我想是因為 AdamW 較適合用來 model 收斂到較低的 loss 時使用

第二名是 Adadelta，我覺得選 optimizer 的在 train 上面的效果差蠻多的，有些 optimizer 會一直停在 loss=1. 多，這也說明了不是每個 optimizer 都適合每個實作，透過選用 optimizer，我 train 一次的時間只需要 1 小時多(跑 30 個 epoch)，省下了不少時間

Experimental results:

test.json: 0.8073

```
● (jack) (base) sqoojack@sqoojack-computer:~/文件/博涵公司/交大/暑修-深度學習/Lab06$ /home/sqoojack/anaconda3/envs/jack/bin/python /home/sqoojack/文件/博涵公司/交大/暑修-深度學習/Lab06/code/test.py
100% |██████████| 32/32 [07:43<00:00, 14.49s/it, image: 31, accuracy: 0.3333]
test accuracy: 0.8073
```

new test.json: 0.8594

```
● (jack) (base) sqoojack@sqoojack-computer:~/文件/博涵公司/交大/暑修-深度學習/Lab06$ /home/sqoojack/anaconda3/envs/jack/bin/python /home/sqoojack/文件/博涵公司/交大/暑修-深度學習/Lab06/code/test.py
100% |██████████| 32/32 [07:44<00:00, 14.51s/it, image: 31, accuracy: 1.0000]
new test accuracy: 0.8594
```

The command for inference process:

要下載 diffusers 模組，並將我的 model 權重下載到 checkpoints 資料夾(名字為 DL_lab6_313552049_鄭博涵.pth) ,

將 checkpoint.pth, new_test.json, test.json, train.json, object.json 存到 code 資料夾裡，接著要用 test.json 來測的話就 command 輸入

“python3 code/test.py” 即可，如果是 new_test.json 的話就把原本 datasets 那行註解掉，並將下一行(new_test) 的註解拿掉即可(如附圖)，最後結果會存在 result 資料夾

(注意要用 cuda 執行，不是 CPU)

```
if __name__ == "__main__":
    set_seed(100)
    # ckpt = 'checkpoints/checkpoint_23(best).pth'
    ckpt = "checkpoints/DL_lab6_313552049_鄭博涵.pth"    # 要將上傳的權重model存在checkpoint裡面
    timesteps = 1000
    model = load_model(ckpt)
    noise_scheduler = get_scheduler(timesteps)
    eval_model = evaluation_model()

    """ 在讀test, new_test的時候要切換這兩個 """
    datasets = { "test": DataLoader(iclevrDataset("iclevr", "test"), batch_size=1, worker_init_fn=seed_worker, generator=g)}
    # datasets = { "new_test": DataLoader(iclevrDataset("iclevr", "new_test"), batch_size=1, worker_init_fn=seed_worker, generator=g)}

    for name, loader in datasets.items():
        acc = generate_images(loader, noise_scheduler, model, eval_model, name)
        print(f'{name} accuracy: {np.mean(acc):.4f}')

    noise_scheduler = get_scheduler(timesteps)
    noise_scheduler.timesteps = noise_scheduler.timesteps.to(device)
```

Reference: [hank891008/Deep-Learning: NYCU Deep Learning](https://github.com/hank891008/Deep-Learning)

[Spring 2024 \(github.com\)](https://github.com/hank891008/Deep-Learning)