Lab6 report

1. Introduction

In this lab, I need to implement a conditional Denoising Diffusion Probabilistic Model (DDPM)to generate synthetic images according to multi-label conditions.

Requirements:

- Implement my conditional DDPM setting.
- Design my noise schedule and UNet architecture
- Choose my loss functions
- Implement the training function, testing function, and data loader
- Evaluate the accuracy of test.json and new test.json
- Show the synthetic images in grids for two testing files
- Plot the progressive generation process for generating an image

2. Implementation details

(1) DDPM

```python
def training_stage(self):
    for epoch in range(self.num_epochs):
        epoch = self.current_epoch
        progress_bar = tqdm(total=len(self.train_loader), disable=not self.accelerator.is_local_main_process)
        progress_bar.set_description(f"Epoch {epoch+1}/{self.num_epochs}")
        total_loss = 0
        for i, (x, class_label) in enumerate(self.train_loader):
            x, class_label = x.to(self.args.device), class_label.to(self.args.device)
            noise = torch.randn_like(x)
            timesteps = torch.randint(0, 1000, (x.shape[0],)).long().to(self.args.device)
            noisy_image = self.noise_scheduler.add_noise(x, noise, timesteps)
            with self.accelerator.accumulate(self.model):
                noise_pred = self.model(noisy_image, timesteps, class_label).sample
                loss = self.loss_fn(noise_pred, noise)
                total_loss += loss.item()
                self.accelerator.backward(loss)
                self.accelerator.clip_grad_norm_(self.model.parameters(), 1.0)
                self.optimizer.step()
                self.lr_scheduler.step()
                self.optimizer.zero_grad()
            logs = {"loss": total_loss / (i+1), "lr": self.lr_scheduler.get_last_lr()[0], "step": self.global_step}
            progress_bar.update(1)
            progress_bar.set_postfix(**logs)
            self.accelerator.log(logs, step=self.global_step)
            self.global_step += 1
        self.losses.append(total_loss / len(self.train_loader))
        if epoch % 5 == 0 or epoch == self.num_epochs - 1:
            self.classification_stage(epoch)
        self.current_epoch += 1
```

這部分是 DDPM 的模型訓練部分。在訓練開始時，我們將總共要訓練的次數設定為 num_epochs，並使用 tqdm 工具來建立一個進度條以追踪訓練進度。在每次的訓練迭代中，我們首先產生一個隨機 noise，這是用於與輸入 x 具有相同形狀的隨機數值。然後隨機生成一組 timesteps，範圍在 0 到 1000 之

間。接著利用 noise_scheduler 的 add_noise 方法，將這些 noise 加到輸入 x 上，從而得到一張噪音圖像 noisy_image。接下來的步驟是將這張 noisy_image 連同 timesteps 和 class_label 一起輸入到模型中。模型的目的是嘗試預測這張圖像的 noise，也就是 noise_pred。為了評估模型的效果，我們將模型預測的噪音和原始噪音進行比較，利用 MSELoss 計算它們之間的差異。理想情況下，模型預測的 noise 應該非常接近原始噪音。在計算出 loss 之後，我們使用一個 adamw 來更新模型的權重，目的是減少 loss。這個過程在每次迭代中都會重複。而在每 5 個 epoch 結束或在最後一個訓練周期結束時，程式會執行一個名為 classification_stage 的方法，進行分類任務，用來評估模型成效。這就是整個 DDPM 的訓練過程，目的是使模型能夠更精確地預測圖像中的噪音。

(2) UNet architectures

```
self.model = newUNet2D(
    sample_size=args.sample_size,
    in_channels=3,
    out_channels=3,
    layers_per_block=args.layers_per_block,
    block_out_channels=(args.block_dim, args.block_dim, args.block_dim*2, args.block_dim*2, args.block_dim*4, args.block_dim*4),
    down_block_types=(
        "DownBlock2D",
        "DownBlock2D",
        "DownBlock2D",
        "DownBlock2D",
        "AttnDownBlock2D",
        "DownBlock2D",
    ),
    up_block_types=(
        "UpBlock2D",
        "AttnUpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
    ),
)
```

```
self.class_embedding = nn.Linear(24, time_embed_dim)

# if class_embed_type is None and num_class_embeds is not None:
#     self.class_embedding = nn.Embedding(num_class_embeds, time_embed_dim)
# elif class_embed_type == "timestep":
#     self.class_embedding = TimestepEmbedding(timestep_input_dim, time_embed_dim)
# elif class_embed_type == "identity":
#     self.class_embedding = nn.Identity(time_embed_dim, time_embed_dim)
# else:
#     self.class_embedding = None
```

我的 newUNet2D 架構改自 hugging face 的 UNet2DModel 架構，跟原本的架構差異在，上圖註解的地方原本沒有註解以及我多增加了 self.class_embedding = nn.Linear(24, time_embed_dim)，這是因為我希望可以將 multi-label conditions 送進 model，之後我會將 labels conditions 透過 linear 層變成跟 time Embedding 同尺寸後，就可以將他跟 time Embedding 相加，一起送進模型。至於 newUNet2D 的 down_block_types 跟 up_block_types 則是完全按照 hugging face 的設定。

(3) noise schedule

```
self.beta_schedule = "squaredcos_cap_v2" if args.beta_schedule == "cosine" else "linear"
self.noise_scheduler = DDPMScheduler(num_train_timesteps=1000, prediction_type=args.predict_type, beta_schedule=self.beta_schedule)
```

在 diffusers 的 DDPMScheduler 中，beta_schedule 可以選擇 linear、squaredcos_cap_v2，上面程式碼的 cosine 代表 squaredcos_cap_v2，也是我本次實驗用的 beta_schedule，我 timestep 一樣標準的設 1000，prediction_type 我一樣用 DDPMScheduler 中的 epsilon (predicting the noise of the diffusion process)。

(4) loss functions

```
self.loss_fn = nn.MSELoss()
```

我利用 mse 當作我的 loss function，去計算模型預測的 noise 跟原本的 noise 算 loss。

(5) Specify the hyperparameters

learning rate: 0.0001
batch size: 64
num_epochs: 120
predict_type: epsilon
optimizer: AdamW
loss function: MSE loss
beta_schedule: squaredcos_cap_v2

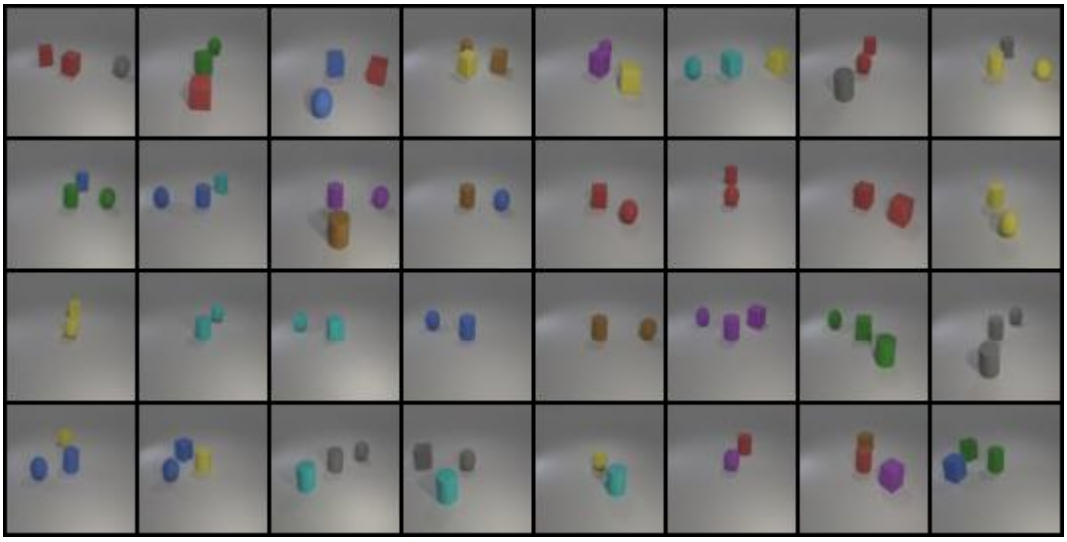
3. Results and discussion

(1) Show your accuracy screenshot based on the testing data

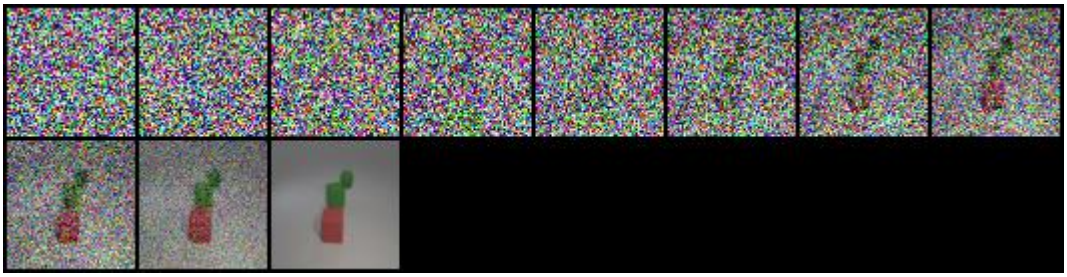
```
> Accuracy: [Test]: 0.9583, [New Test]: 0.9643
```

上圖的 0.9583 是 test.json 的 accuracy，0.9643 是 new test.json 的 accuracy

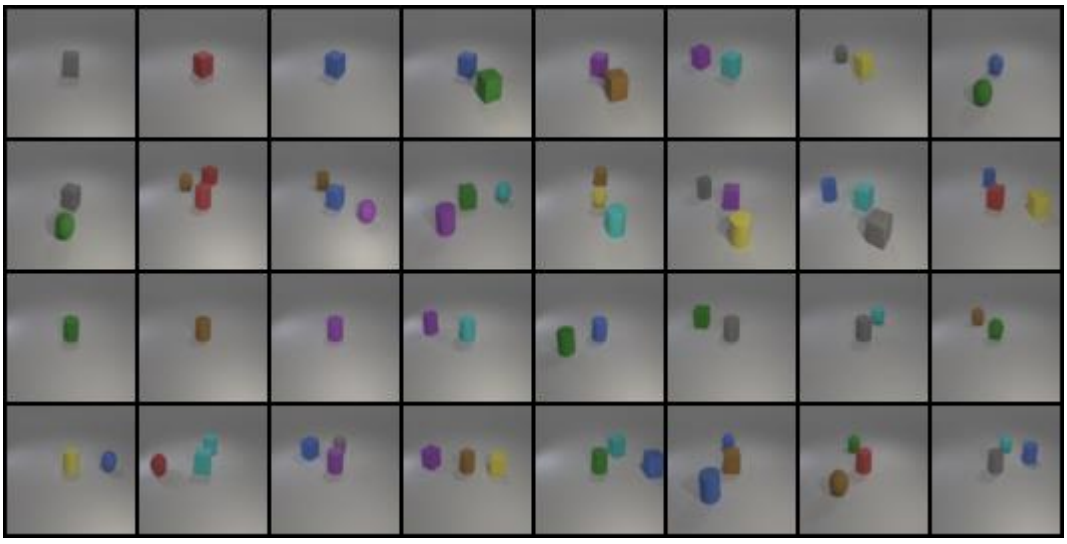
(2) Show your synthetic image grids and a progressive generation image



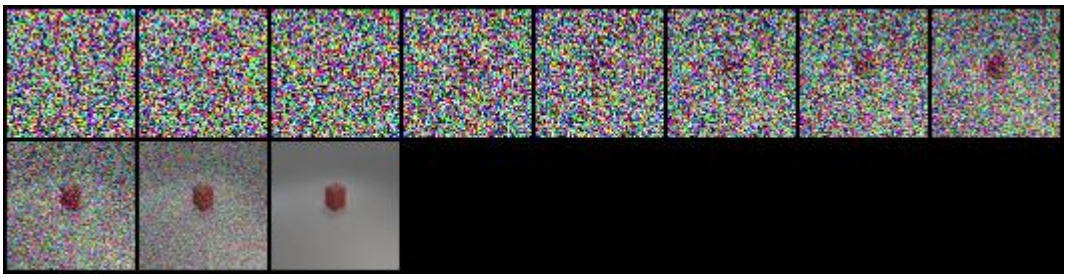
上圖是 new test 的 synthetic image grids 圖



上圖是 new test 的 progressive generation 圖，我每隔 111 的 denoise 取 1 張圖出來



上圖是 test 的 synthetic image grids 圖



上圖是 test 的 progressive generation 圖，我一樣每隔 111 的 denoise 取 1 張圖出來

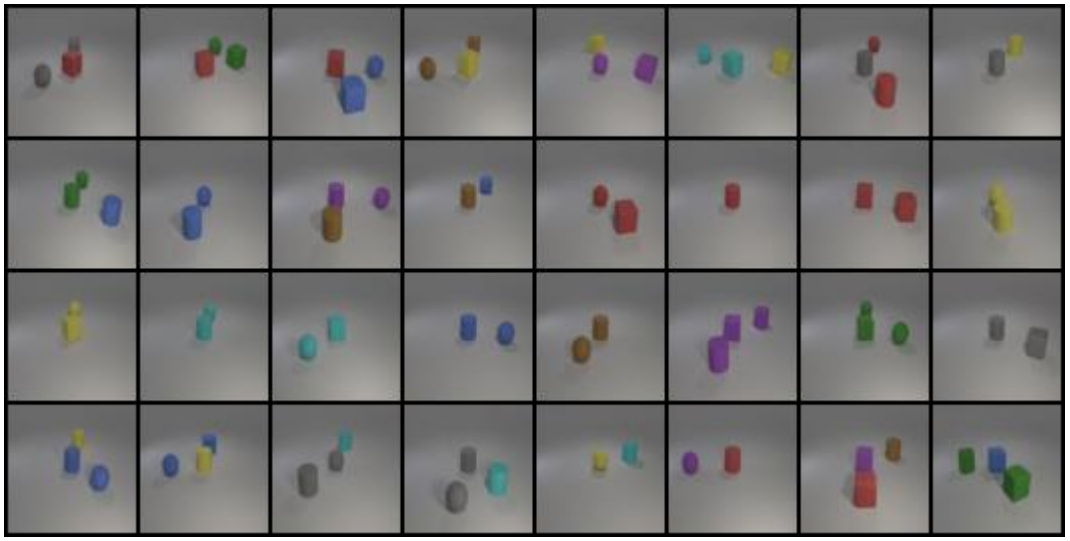
(3) Discuss the results of different model architectures or methods

(a) 不同的 beta_schedule

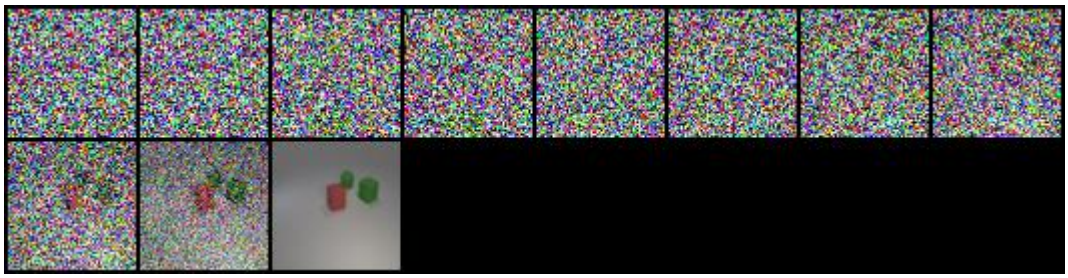
在 diffusers 的 DDPMScheduler 中，beta_schedule 可以選擇 linear、squaredcos_cap_v2，上面 result 的 accuracy 是用 squaredcos_cap_v2，所以我實作了其他條件都不變的情況下改用 linear 看看結果會變得如何，以下是 linear 的 accuracy 的 screenshot

```
> Accuracy: [Test]: 0.8750, [New Test]: 0.8929
```

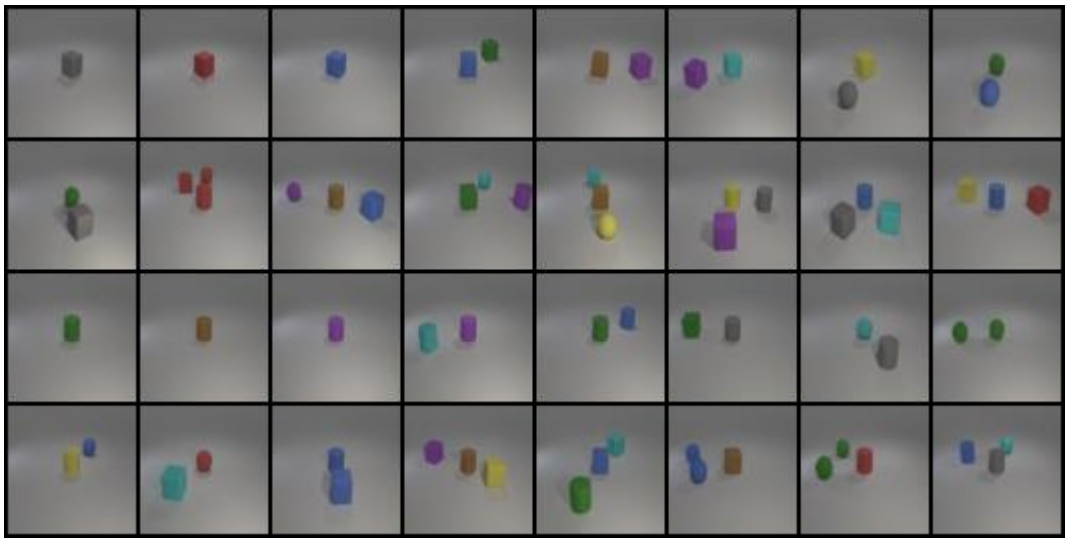
從上圖可以看到跟原本的 squaredcos_cap_v2 比起來，accuracy 明顯大幅下降。



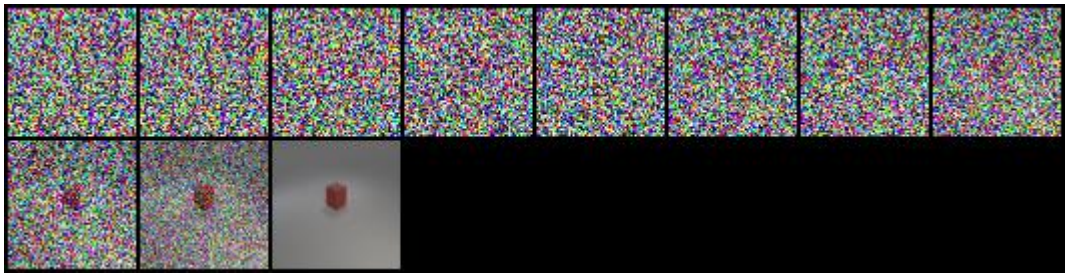
上圖是改用 linear 的 new test 的 synthetic image grids 圖



上圖是改用 linear 的 new test 的 progressive generation 圖



上圖是改用 linear 的 test 的 synthetic image grids 圖



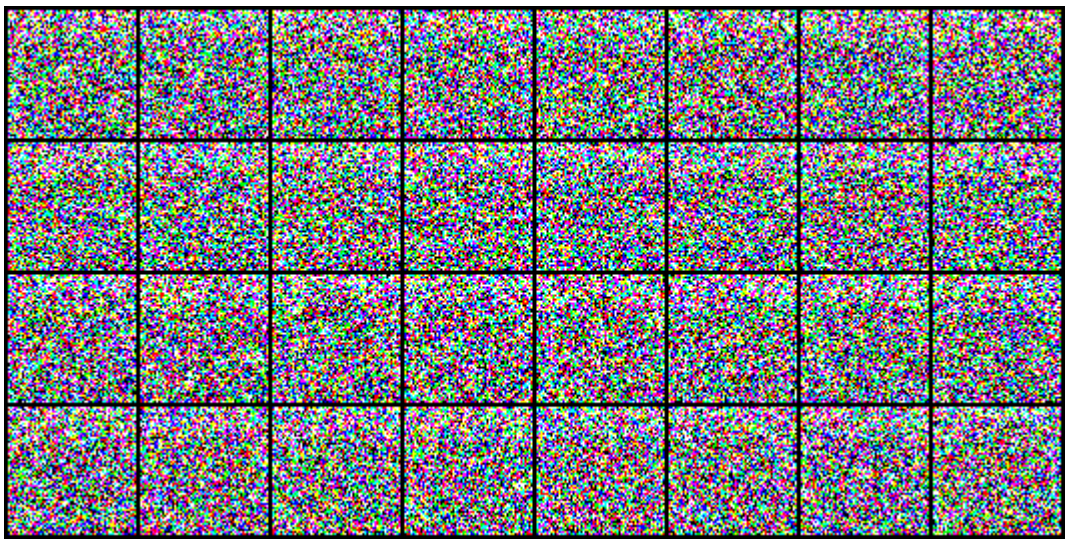
上圖是改用 linear 的 test 的 progressive generation 圖

(b) 不同的 prediction_type

在 diffusers 的 DDPMScheduler 中，prediction_type 可以選擇 sample (directly predicting the noisy sample)、epsilon (predicting the noise of the diffusion process)，上面 result 的 accuracy 是用 epsilon，所以我實作了其他條件都不變的情況下改直接用 sample 看看結果會變得如何，以下是 sample 的 accuracy 的 screenshot

```
> Accuracy: [Test]: 0.1111, [New Test]: 0.1071
```

從上圖可以看到，如果沒有用 diffusion process 來 predict noise 的話，accuracy 會變得很不好，不管是 test 或 new test 最後產生出來的圖都是雜訊，如下圖



4. Reference

https://huggingface.co/docs/diffusers/tutorials/basic_training