DLP LAB7

Let’s play DDPM

311551170 林琨堯

1. Introduction

這次的 lab 需要實作 DDPM 對影像中的積木進行分類。

DDPM 是近年提出的 generation network，通過迭代將 diffusion process 應用於高斯分佈等 base distribution。過程中逐漸將基礎分佈轉變為 target data distribution。在每個擴散步驟中，將 noise 從當前分佈注入樣本，並使用稱為 diffusion model 對樣本進行 denoise，使其更接近 target data distribution。

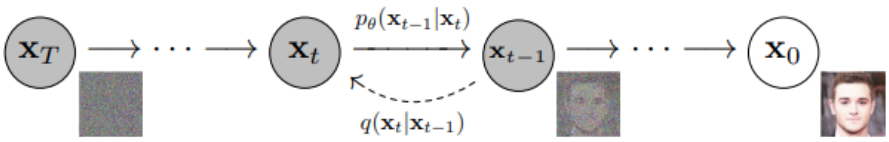


Figure 2: The directed graphical model considered in this work.

2. Implementation details

2.1 DDPM

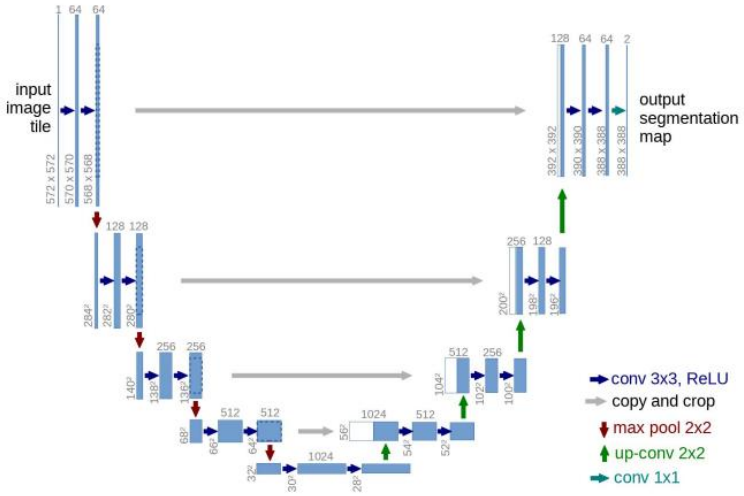
用 diffusers.DDPMScheduler 進行 noising 及 denoising。

DDPMScheduler 可以在經過一段時間之後，在影像上加入 gaussian noise，設定如下:

noise_scheduler = DDPMScheduler (num_train_timesteps=1000, beta_schedule='squaredcos_cap_v2')，每張影像最多傳遞 1000 次，也就是 T={0~999}，beta_schedule 控制 denoising 的方法。

2.2 U-Net

架構如圖，用 diffusers.UNet2DModel 可以快速建立 U-Net



```
net = UNet2DModel(
    sample_size = 64,
    in_channels = 3,
    out_channels = 3,
    layers_per_block = 2,
    class_embed_type = None,
    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-attention
    "DownBlock2D",
    ),
    up_block_types=(
        "UpBlock2D",  # a regular ResNet upsampling block
        "AttnUpBlock2D",  # a ResNet upsampling block with spatial self-attention
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
    ),
)
```

2.3 noise schedule

如圖，noise 和迭代次數都是 random，然後再用 2.1 定義的 noise_scheduler 加上 noise

```
noise = torch.randn_like(img)
timesteps = torch.randint(0, 999, (img.shape[0],)).long().to(device)
noisy_img = noise_scheduler.add_noise(img, noise, timesteps)
```

2.4 loss function

因為影像都是積木，所以只需要簡單判斷 predict 和 ground truth 的差異即可，所以選用 MSELoss 作為 loss function。

2.5 hyper-parameters

epochs = 200，但是當 test acc 和 new_test acc 大於 0.85 時會跳出迴圈

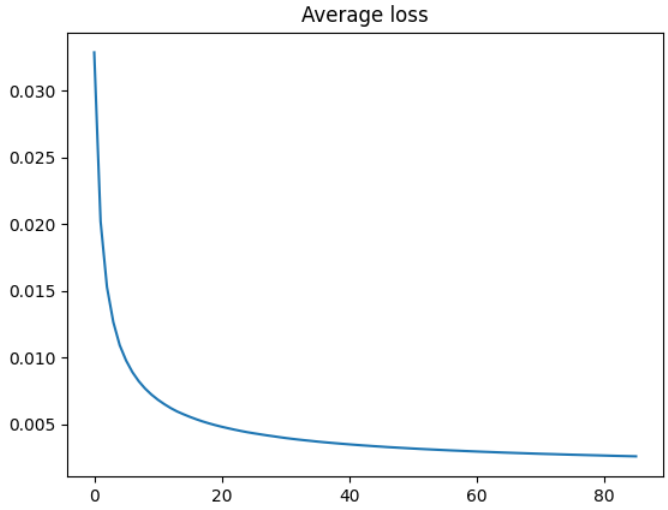
batch size = 32

3. Dataset Descriptions

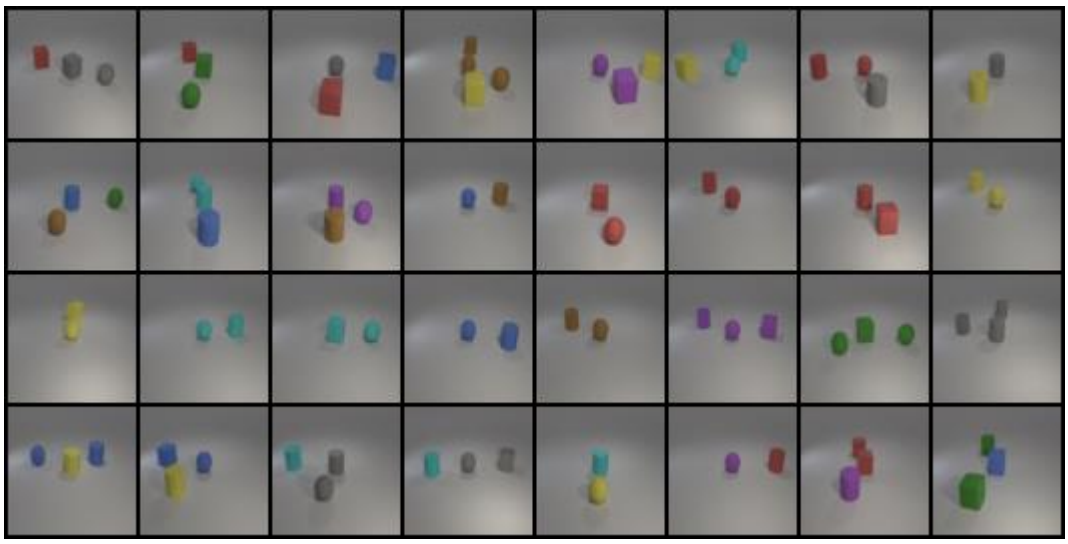
讀取影像後，會重新設定成(64, 64, 3)的影像，且 label 會轉成 one_hot_encoding

4. Result

```
epoch 85: Average loss = 0.002609
==test.json==
1000it [01:04, 15.52it/s]
Test Result: 0.875
==test_new.json==
1000it [01:04, 15.52it/s]
Test Result: 0.8690476190476191
```

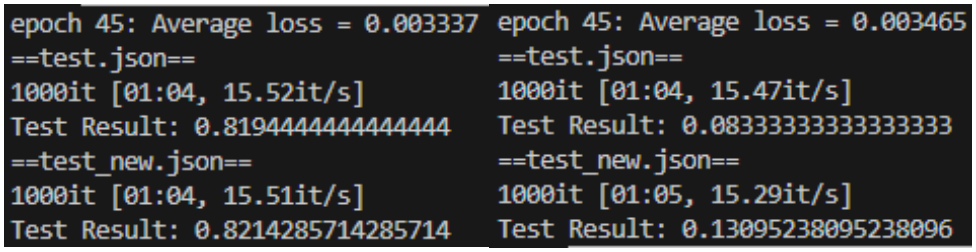


Test F1-score = 0.875



New_Test F1-score = 0.869

利用 embedding 對資料進行降維，減少運算量，加快收斂。在沒有 embedding 的情況下，經過相同的 epochs 得到的 accuracy 遠低於有 embedding 得到的 accuracy。



有 embedding　　無 embedding