Lab6 - Generative Models

312551133
鄧長軒

Introduction

在這次lab中，我實作了GAN and DDPM兩種模型，資料集為具有多種物體的圖片，與相對應的multi label，我成功將multi-label condition embedding 應用到模型中，讓模型可以透過multi-label，在圖片中生成出具有多個不同形狀與顏色的物體。最後在test evaluator中，GAN得到0.8與0.82的準確率，DDPM得到0.95與0.95的準確率。

Implementation details

Implement a conditional GAN

- 程式碼大部分是參考 Pytorch DCGAN tutorial 完成的。
- 訓練方式為：先用 batch_size = 128 訓練，得到準確率最高的模型參數後，再用 batch_size = 64 繼續訓練，再依序用 batch_size = 32, 16, 8, 4, 2得到最後模型參數。
- 想法是：一開始訓練時 batch_size 調大，讓模型學到不管 label 是什麼，背景都是白的。之後 batch_size 漸漸調小，讓模型學到 label 會影響物體顏色跟形狀。

GAN - generator

```python
class Generator(nn.Module):
    def __init__(self, nz, ngf, nc, num_classes):
        super(Generator, self).__init__()

        self.main = nn.Sequential(
            nn.ConvTranspose2d(nz + num_classes, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, noise, labels):
        labels = labels.unsqueeze(2).unsqueeze(3)
        noise_label = torch.cat((noise, labels), dim=1)
        return self.main(noise_label)
```

noise的形狀為[100,1,1]，

one hot label為24，

先把label unsqueeze為[24,1,1]，再直接跟noise concat在一起，變成[124,1,1]，然後輸入到generator裡。

GAN - discriminator

```
class Discriminator(nn.Module):
    def __init__(self, nc, ndf):
        super(Discriminator, self).__init__()

        self.main = nn.Sequential(
            nn.Conv2d(nc + nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, img, labels):
        labels=labels.repeat_interleave(2,dim=1)
        labels = torch.nn.functional.pad(labels, (0, 16))
        labels = labels.unsqueeze(1).unsqueeze(1)
        labels = labels.expand(img.shape)
        img_label = torch.cat((img, labels), 1)
        return self.main(img_label)
```

1. 對label用repeat_interleave，形狀變[48]
ex: (0,1,0,1) -> (0,0,1,1,0,0,1,1)

2. 在後面補0，形狀變成[64]
ex: (0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0)

3. 對它複製64次，變成[64,64]

4. concat到img上，藉此完成
condition embedding。

GAN - loss function

- criterion 使用 Pytorch DCGAN tutorial 建議的 nn.BCELoss()
- 沒有使用pretrained evaluator

GAN - training function

```
for real_images, one_hot_labels in tqdm(dataloader):
    netD.zero_grad()
    one_hot_labels = one_hot_labels.to(device)
    real_images = real_images.to(device)
    label_real = torch.full((one_hot_labels.size(0),), 1, dtype=torch.float, device=device)
    output = netD(real_images, one_hot_labels).view(-1)
    lossD_real = criterion(output, label_real)
    lossD_real.backward()

    noise = torch.randn(one_hot_labels.size(0), nz, 1, 1, device=device)
    fake_images = netG(noise, one_hot_labels)
    label_fake = torch.full((one_hot_labels.size(0),), 0, dtype=torch.float, device=device)
    output = netD(fake_images.detach(), one_hot_labels).view(-1)
    lossD_fake = criterion(output, label_fake)
    lossD_fake.backward()
    optimizerD.step()

    acc=evaluation_model.eval(fake_images, one_hot_labels)
    total_acc+=acc

    netG.zero_grad()
    output = netD(fake_images, one_hot_labels).view(-1)
    lossG = criterion(output, label_real)
    loss+=lossG.item()
```

1. 用真圖片訓練discriminator
2. 用假圖片訓練discriminator
3. 更新discriminator
4. 計算generator生成出的圖片的loss，更新generator
5. pretrained evaluator只有輸出準確率，讓我決定哪個模型效果好，並沒有參與訓練。

GAN - testing function

```
noise = torch.randn(num_test_samples, nz, 1, 1, device=device)
with torch.no_grad():
    fake_images = netG(noise, test_labels)
evaluation_model = evaluator.evaluation_model()
acc=evaluation_model.eval(fake_images, test_labels)
print(acc)
```

- 建立noise，將noise跟one hot label輸入到generator。
- 用pretrained evaluator計算準確率。

Implement a conditional DDPM

- 程式碼大部分是參考Hugging Face Diffusion Models Course完成的。
- 有使用library: from diffusers import DDPMScheduler, UNet2DModel

DDPM - model

```python
class ClassConditionedUnet(nn.Module):
  def __init__(self):
    super().__init__()
    self.model = UNet2DModel(
        sample_size=64,
        in_channels=6,
        out_channels=3,
        layers_per_block=2,
        block_out_channels=(64, 128, 256, 512),
        down_block_types=(
            "DownBlock2D",
            "DownBlock2D",
            "DownBlock2D",
            "AttnDownBlock2D",
        ),
        up_block_types=(
            "AttnUpBlock2D",
            "UpBlock2D",
            "UpBlock2D",
            "UpBlock2D",
          ),
    )
  def forward(self, x, t, one_hot_labels):
    labels=one_hot_labels.repeat_interleave(2,dim=1)
    labels = torch.nn.functional.pad(labels, (0, 16))
    labels = labels.unsqueeze(1).unsqueeze(1)
    labels = labels.expand(x.shape)
    net_input = torch.cat((x, labels), 1)
    return self.model(net_input, t).sample
```

- 使用UNet架構，是diffusers提供的UNet2DModel。
- 在接近bottleneck的block，使用了attention block，實驗發現可以提高準確率。
- condition embedding的方法跟GAN discriminator中的方法一樣，label的形狀[24] -> [48] -> [64] -> [64,64]，再與圖片concat在一起。

DDPM - noise schedule, time embeddings

- 用 diffusers 提供的 DDPMScheduler 來完成。
- noise schedule 使用 squaredcos_cap_v2。
- noisy_x = noise_scheduler.add_noise(x, noise, timesteps)，輸入noise跟time， noise_scheduler會算出加了noise的圖片。

DDPM - loss functions

- 使用Hugging Face Diffusion Models Course 建議的nn.MSELoss()
- 沒有使用pretrained evaluator

DDPM - training function

```python
for epoch in range(num_epochs):
    accum=0
    losses=0
    for x, y in tqdm(dataloader):
        x = x.to(device)
        y = y.to(device)
        noise = torch.randn_like(x)
        timesteps = torch.randint(0, num_train_timesteps-1, (x.shape[0],)).long().to(device)
        noisy_x = noise_scheduler.add_noise(x, noise, timesteps)
        pred = net(noisy_x, timesteps, y)
        loss = loss_fn(pred, noise)
        losses+=loss.item()
        loss=loss/accum_grad
        loss.backward()
        accum+=1
        if accum % accum_grad == 0:
            optimizer.step()
            optimizer.zero_grad()
```

- 隨機產生time
- 用noise跟time算出加了noise的圖片。
- 將noisy_x, time, label輸入模型，輸出預測noise。
- 預測noise跟原本的noise算loss。
- 根據gradient accumulate更新參數。

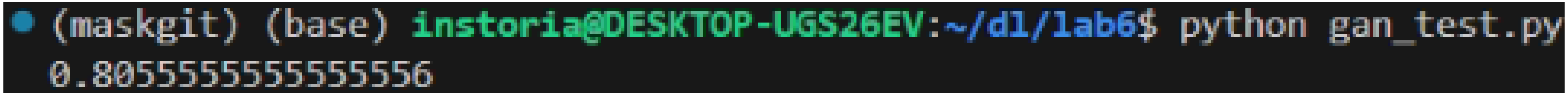
DDPM - testing function

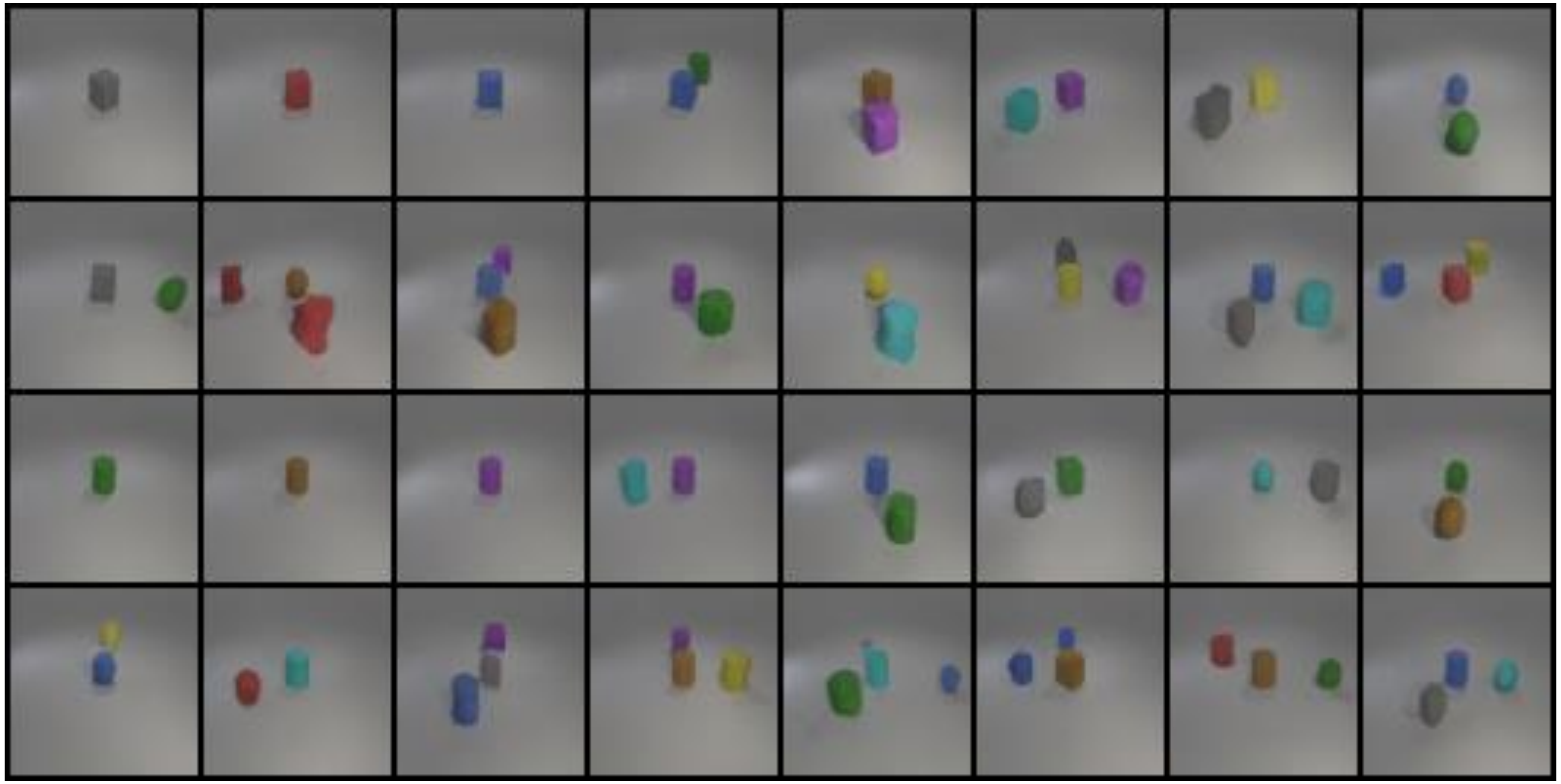
```
for i, t in tqdm(enumerate(noise_scheduler.timesteps)):

    with torch.no_grad():
        residual = net(x, t, test_labels)

    x = noise_scheduler.step(residual, t, x).prev_sample
```

- 將模糊圖片, time, label 輸入模型，輸出預測noise。
- 將noise, time, 模糊圖片輸入noise_scheduler，消除一部份noise，輸出較清楚的圖片。

Results and discussion

GAN - test.json





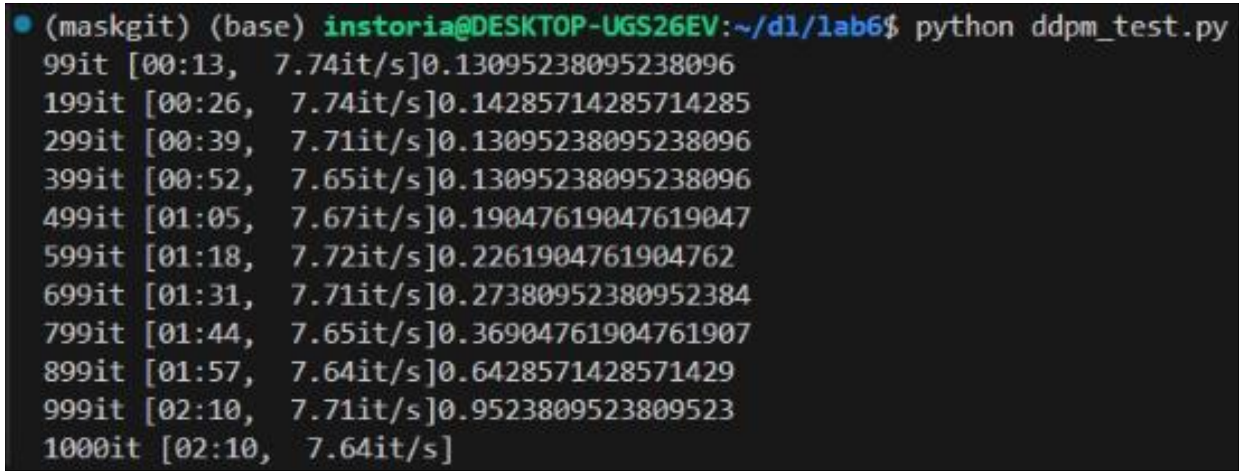
accuracy=0.80

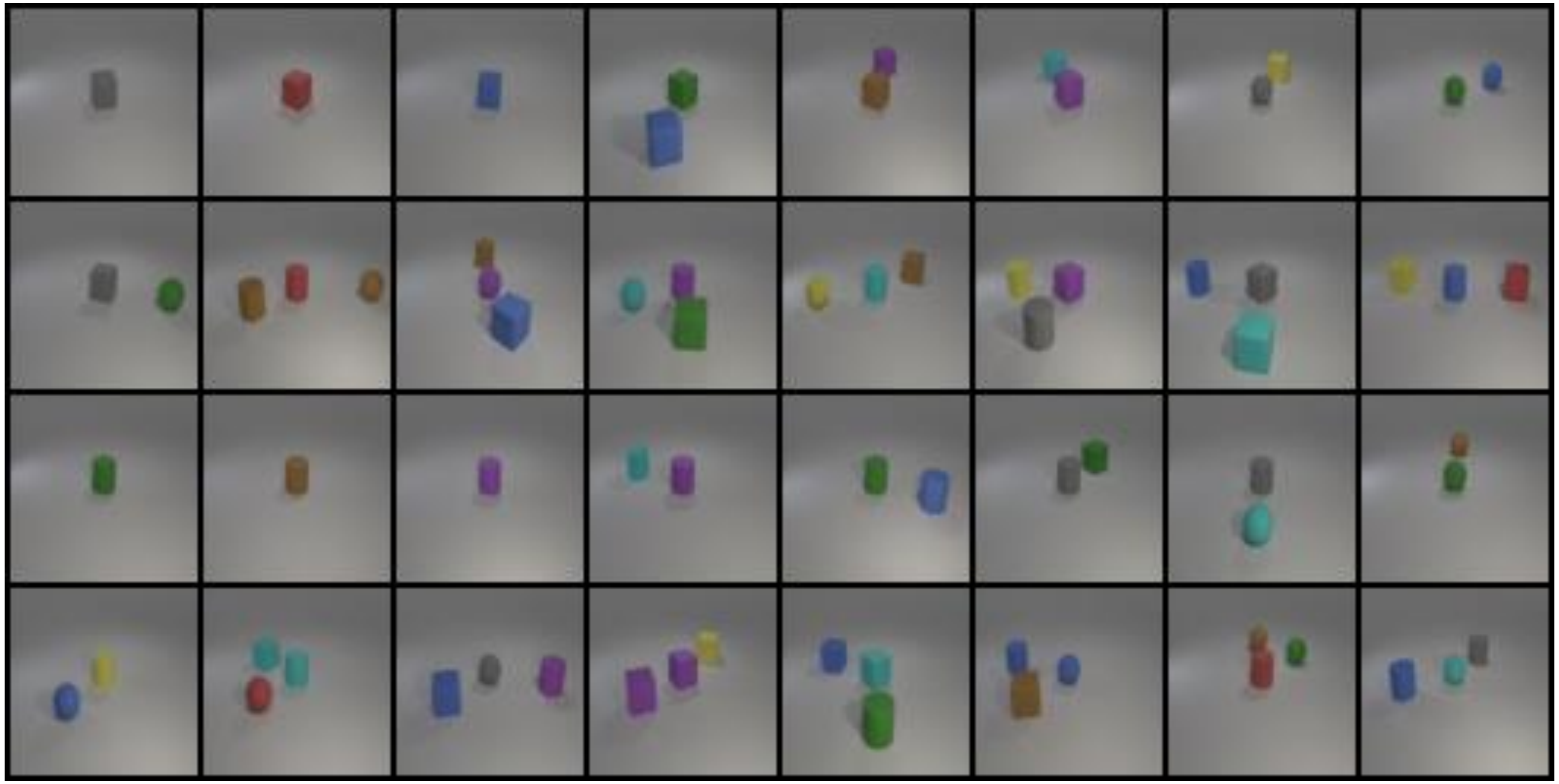
GAN - new_test.json

```
(maskgit) (base) instoria@DESKTOP-UGS26EV:~/dl/lab6$ python gan_test.py
0.8214285714285714
```

accuracy=0.82

DDPM - test.json

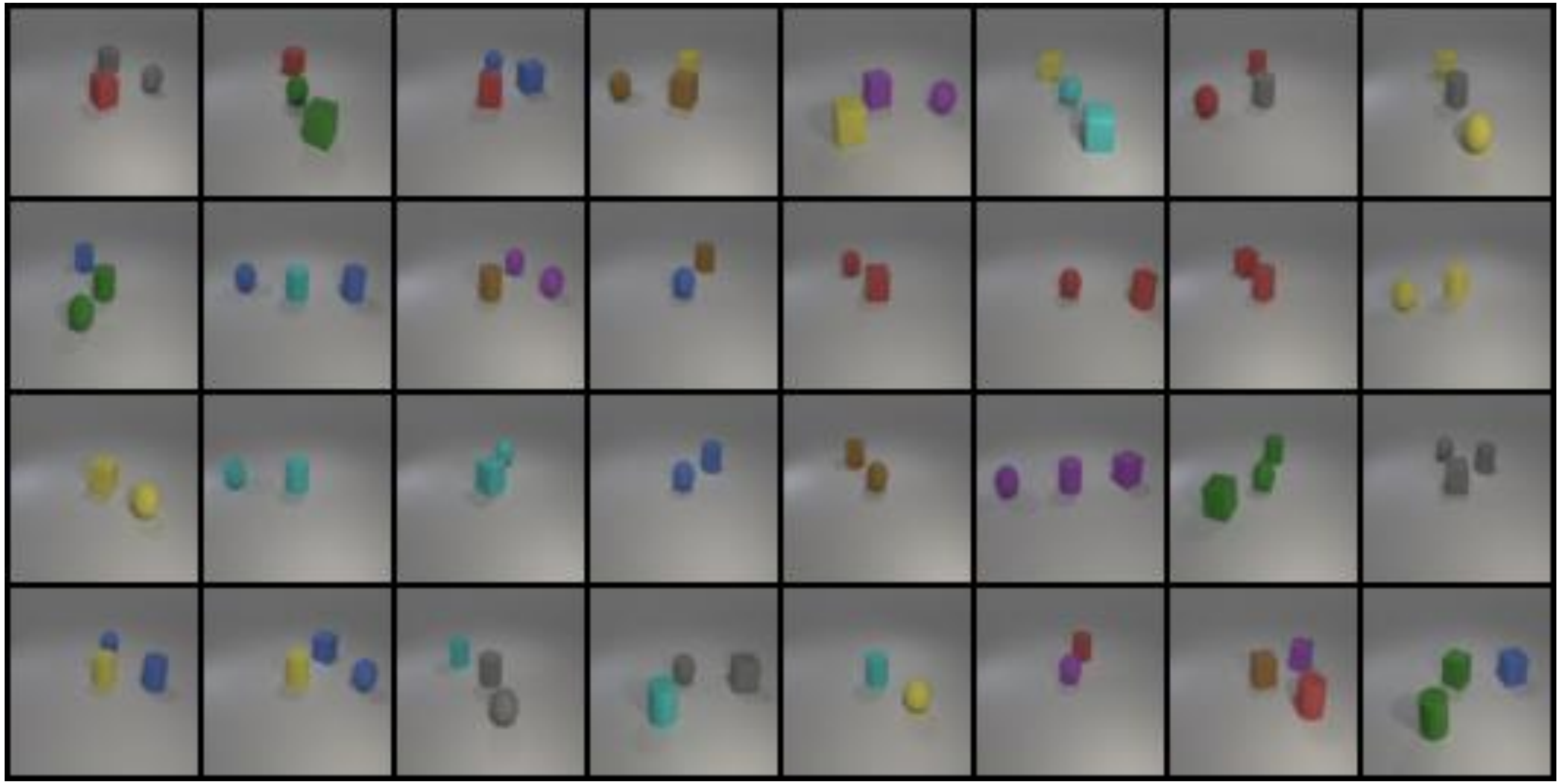




accuracy=0.95

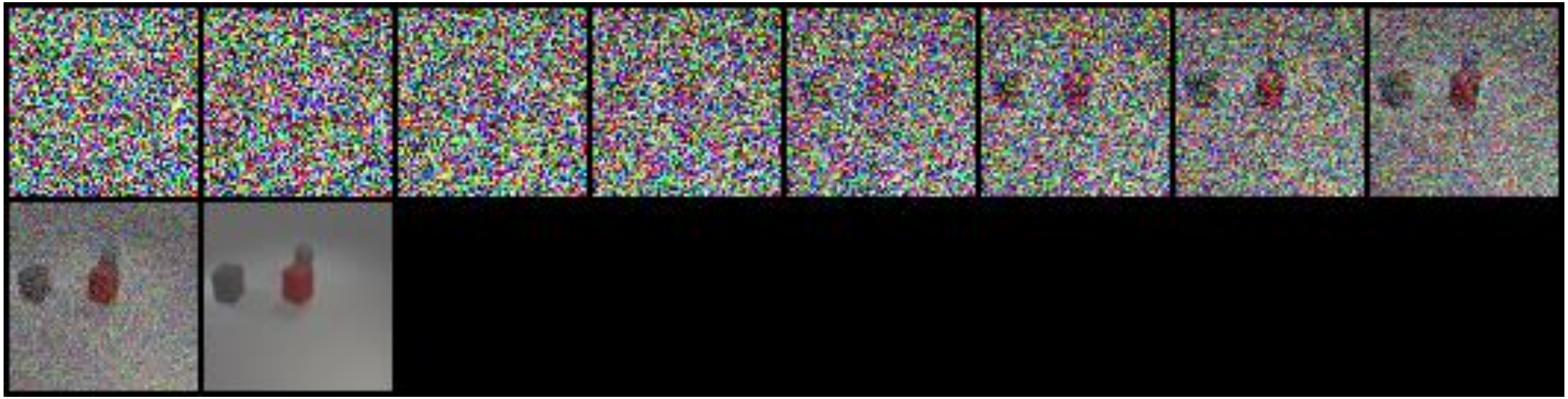
DDPM - new_test.json

```
(maskgit) (base) instoria@DESKTOP-UGS26EV:~/dl/lab6$ python ddpm_test.py
99it [00:13,  7.61it/s]0.1527777777777778
199it [00:26,  7.57it/s]0.1111111111111111
299it [00:39,  7.63it/s]0.09722222222222222
399it [00:53,  7.62it/s]0.08333333333333333
499it [01:06,  7.61it/s]0.125
599it [01:19,  7.61it/s]0.16666666666666666
699it [01:32,  7.36it/s]0.1527777777777778
799it [01:45,  7.64it/s]0.3194444444444444
899it [01:58,  7.58it/s]0.5694444444444444
999it [02:12,  7.59it/s]0.9583333333333334
1000it [02:12,  7.56it/s]
```



accuracy=0.95

DDPM denoising process image



label set ["gray cube", "red cube", "gray sphere"]

Compare the advantages and disadvantages of the GAN and DDPM models

- DDPM:
  - 優點：訓練穩定，sampling時如果出錯，後續有機會補救回來。
  - 缺點：生成速度慢，模型較複雜。

- GAN:
  - 優點：生成速度快。
  - 缺點：訓練不穩定，有可能mode collapse。

Discussion of extra implementations

- 在訓練GAN時，我原本有使用pretrained evaluator，pretrained evaluator 算出的 accuracy，將loss乘上1- accuracy後，再做參數更新。
- 想法是將accuracy納入考慮，如果accuracy=1，代表對evaluator已經很好了，不需要再訓練，所以loss=0。
- 但後來發現效果不好，推測是因為GAN本身就很不穩定了，如果再多一個因素來影響loss，只會更加不穩定。