# Lab6: Generative Models

謝鎧駿 B103040021

## 1. Introduction

本次 Lab 實作了 Conditional DDPM，針對 iclevr 資料集的 multi-label conditions 去生成圖像，我主要使用 Hugging face 的 Unet2Dmodel 搭配 attention 與 class condition embedding，並使用 cosine denoise scheduler，最後，我使用 pre-trained evaluator 計算 accuracy 並視覺化特定 label 的 denoising 過程。

## 2. Implementation details

### I. Model

我使用 Hugging face 的 UNet2Dmodel 去實作 conditional DDPM，模型由 5 個 down block and up block 組成，並在其中使用 attention block。而 condition 會先通過一層 linear 使其有相同 dimension，再去通過 SiLU 讓 condition 的 embedding 擁有非線性。

最後，channel 等於 dimension 四分之一是因為這樣才能對到 embedding 的 tensor dimension。

```python
class ConditionalDDPM(nn.Module):
    def __init__(self, num_classes=24, dim=512):
        super().__init__()
        channel = dim // 4
        self.ddpm = UNet2DModel(
            sample_size=64,
            in_channels=3,
            out_channels=3,
            layers_per_block=2,
            block_out_channels=[channel, channel, channel*2, channel*2, channel*4],
            down_block_types=["DownBlock2D", "DownBlock2D", "DownBlock2D", "AttnDownBlock2D", "DownBlock2D"],
            up_block_types=["UpBlock2D", "AttnUpBlock2D", "UpBlock2D", "UpBlock2D", "UpBlock2D"],
            class_embed_type="identity",
        )
        self.class_embedding = nn.Sequential(
            nn.Linear(num_classes, dim),
            nn.SiLU()
        )

    def forward(self, x, t, label):
        class_embed = self.class_embedding(label)
        return self.ddpm(sample=x, timestep=t, class_labels=class_embed).sample
```

### II. Train

在 train 中，total timestamps 設定為 1000，且 Noise schedule 使用 squaredcos_cap_v2，Loss function 為 MSE，Optimizer 為 Adam。

在 training 的過程，每次從 dataset load 一張圖與其 Label，並隨機一個 timestamp，接著用 Noise schedule 將高斯雜訊加到圖上，再將雜訊圖、timestamp、Label 丟給 model 做 predict，最後用 MSE 計算 Loss 並用 Adam optimize。

並且我使用 wandb 去觀測 Loss 的變化。

```python
# Hyperparameters
EPOCHS = 500
BATCH_SIZE = 32
LR = 1e-5
T = 1000 # time stamp

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = ConditionalDDPM().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=LR)

scheduler = DDPMScheduler(
    num_train_timesteps = T,
    beta_schedule = "squaredcos_cap_v2",
    prediction_type = "epsilon"
)

train_dataset = ICLEVRDataset('train.json', img_dir='../iclevr/iclevr')
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, drop_last=True)
wandb.init(project="DL_Lab6_DDPM", name="conditional_ddpm_run")

# Training
for epoch in range(EPOCHS):
    model.train()
    running_loss = 0.0
    for imgs, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}"):
        imgs = imgs.to(device)
        labels = labels.to(device)
        batch_size = imgs.size(0) # shape[0]

        timesteps = torch.randint(0, scheduler.config.num_train_timesteps, (batch_size,), device=device).long()

        noise = torch.randn_like(imgs)
        noisy_imgs = scheduler.add_noise(imgs, noise, timesteps)

        pred_noise = model(noisy_imgs, timesteps, labels)
        loss = nn.MSELoss()(pred_noise, noise)

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

        running_loss += loss.item()

        wandb.log({"batch_loss": loss.item()})

    avg_loss = running_loss / len(train_loader)
    print(f"Epoch {epoch+1}, Loss: {avg_loss:.6f}")

    wandb.log({"epoch_loss": avg_loss})

    # Save
    if epoch%2 == 0:
        save_checkpoint(model, optimizer, f"./results/ddpm_epoch{epoch+1}.pth")
```

## III. Test

Testing 的步驟與 training 大同小異，首先載入存好的 model 並定義 Noise schedule 後，載入 testing dataset。

```python
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_ckpt_path = "./results/ddpm_epoch99.pth"
save_dir = "./results_img"
os.makedirs(save_dir, exist_ok=True)
batch_size = 32
timesteps = 1000
guidance_weight = 1.0

model = ConditionalDDPM().to(device)
model.load_state_dict(torch.load(model_ckpt_path, map_location=device, weights_only=True)['model_state_dict'])
model.eval()

scheduler = DDPMScheduler(
    num_train_timesteps=timesteps,
    beta_schedule="squaredcos_cap_v2",
    prediction_type="epsilon"
)
scheduler.set_timesteps(timesteps)

evaluator = evaluation_model()

test_dataset = ICLEVRDataset(json_path = "test.json", img_dir=None)
new_test_dataset = ICLEVRDataset(json_path = "new_test.json", img_dir=None)

test_labels = torch.stack([test_dataset[i] for i in range(len(test_dataset))])
new_test_labels = torch.stack([new_test_dataset[i] for i in range(len(new_test_dataset))])
```

接著針對 testing dataset 的 condition 去一步一步地去預測 noise 並用 noise schedule 減掉預測出來的 noise，最後減完後，將圖片 denormalize 後存起來即可。

而計算 accuracy 的方式就是直接 call evaluator 去計算。

```python
def denormalize(x):
    return (x + 1) / 2

def sample_images(label_tensor, desc="Sampling"):
    model.eval()
    batch_size = label_tensor.size(0)
    x = torch.randn(batch_size, 3, 64, 64, device=device)
    labels = label_tensor.to(device)
    for t in tqdm(scheduler.timesteps, desc=desc):
        t_tensor = torch.full((batch_size,), t, device=device, dtype=torch.long)
        with torch.no_grad():
            noise_pred = model(x, t_tensor, labels)
        x = scheduler.step(noise_pred, t, x).prev_sample
    return x

def save_images(imgs, folder, prefix):
    imgs = denormalize(imgs)
    grid = make_grid(imgs, nrow=8)
    save_image(grid, os.path.join(folder, f"{prefix}_grid.png"))
    for idx, img in enumerate(imgs):
        save_image(img, os.path.join(folder, f"{prefix}_{idx:03d}.png"))

def evaluate_accuracy(imgs, labels):
    acc = evaluator.eval(imgs, labels)
    return acc

gen_test_imgs = sample_images(test_labels, desc="Sampling test set")
save_images(gen_test_imgs, save_dir, "test")

gen_new_imgs = sample_images(new_test_labels, desc="Sampling new_test set")
save_images(gen_new_imgs, save_dir, "new_test")

print("Evaluating...")
acc_test = evaluate_accuracy(gen_test_imgs, test_labels)
acc_new_test = evaluate_accuracy(gen_new_imgs, new_test_labels)
print(f"Accuracy on test.json: {acc_test * 100:.2f}%")
print(f"Accuracy on new_test.json: {acc_new_test * 100:.2f}%")
```

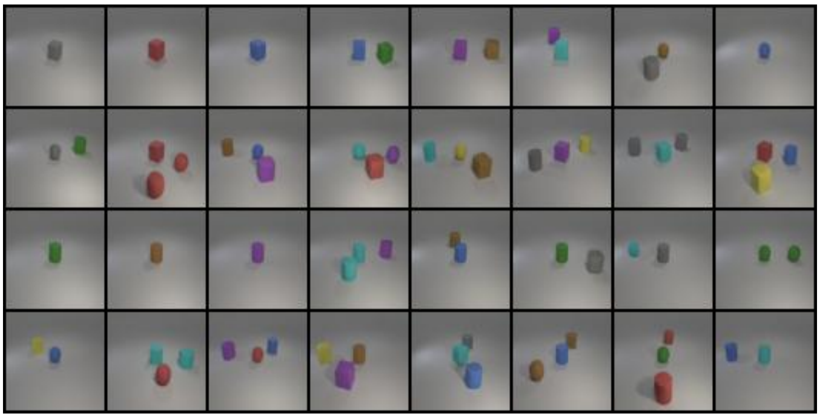
最後是完成 TA 指定的條件，我直接對照 object.json 將那三個物體的 index 設為 1，接著丟給 model 跟上面做一樣的事情後，將每 100 個 timestamps 的圖存起來就完成了。

```python
# label: ["red sphere", "cyan cube", "cyan cylinder"]
labels = [6, 9, 22]
label_vec = torch.zeros(24)
label_vec[[idx for idx in labels]] = 1
label_vec = label_vec.unsqueeze(0).to(device)

frames = []
x = torch.randn(1, 3, 64, 64, device=device)

with torch.no_grad():
    for t in tqdm(reversed(range(scheduler.num_train_timesteps)), desc="Denoising process"):
        t_tensor = torch.full((1,), t, device=device, dtype=torch.long)
        noise_pred = model(x, t_tensor, label_vec)
        x = scheduler.step(noise_pred, t, x).prev_sample
        if t % 100 == 0:
            frames.append(denormalize(x.squeeze(0).cpu()))

grid = make_grid(frames, nrow=len(frames))
save_image(grid, os.path.join(save_dir, "denoise_process.png"))
```

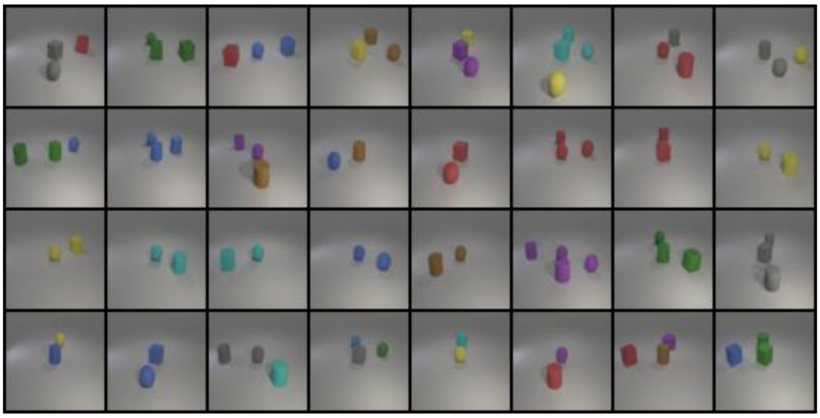
## 3. Results and Discussion:

### I. Show your synthetic image grids:

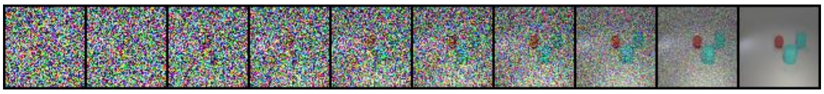
- Test.json:



- New_test.json:



- Specific label denoising process:



### II. Discussion of your extra implementations or experiments:

- Extra experiments:

  我打算進行在訓練 10 個 epoch 下，不同 Noise schdule 的比較。

#### i. squaredcos_cap_v2

```
Evaluating...
Accuracy on test.json: 0.29
Accuracy on new_test.json: 0.42
```

#### ii. linear

```
Evaluating...
Accuracy on test.json: 0.19
Accuracy on new_test.json: 0.32
```

iii. **sigmoid**

```
Evaluating...
Accuracy on test.json: 0.22
Accuracy on new_test.json: 0.39
```

觀察以上三種 Noise schedule，可以發現 squaredcos_cap_v 擁有最好的表現，我推測是因為他是平滑的 cosine Noise schedule，所以訓練過程中能學到更多有效還原階段的特徵。

而 Linear 是三者之中最差的，因為 noise 隨時間均勻增加會導致訓練初期圖像太快被破壞，使 model 難以學習到早期的 denoise 還原。

最後是 Sigmoid，表現居中，沒有 Linear 的問題，但能力不如 squaredcos_cap_v。

4. **Experimental results:**

```
Evaluating...
Accuracy on test.json: 0.88
Accuracy on new_test.json: 0.90
```