311551148 HW7 Report

1. Introduction (5%)

   這次 lab 要實作 DDPM，並且訓練在 iclevr dataset 上，希望訓練完成後，可以輸入想要的 label，得到生成的圖片會符合輸入的 label，比較困難的部分是這個 dataset 一張照片裡會有多個物體，所以使用 onehot 的方式定義每張圖片所對應的 label

2. Implementation details (20%)

   甲、Describe how you implement your model, including your choice of DDPM, UNet architectures, noise schedule, and loss functions. (15%)

   dataset, dataloader 回傳 img label pair，label 已經轉換成 onehot label

```python
class iclevr_dataset(dataset):
    def __init__(self, args, mode="train", transform=default_transform):
        super(iclevr_dataset, self).__init__(args, mode, transform)

        self.args = args
        self.mode = mode
        self.transforms = transform

        ## Get all data paths
        self.train_img_dirs = []
        self.train_labels = []
        obj_data = json.load(open(args.objects_json))

        for key, value in json.load(open(args.train_data_json)).items():
            self.train_img_dirs.append(args.dataset_root + key)
            onehot = torch.zeros((24))
            for cls in value:
                onehot[obj_data[cls]] = 1
            self.train_labels.append(onehot)


    def __len__(self):
        return len(self.train_labels)

    def get_img(self, index):
        img_path = self.train_img_dirs[index]
        img_arr = Image.open(img_path).convert('RGB')
        img_tensor = self.transforms(img_arr)

        return img_tensor

    def get_label(self, index):
        return self.train_labels[index]


    def __getitem__(self, index):
        img  = self.get_img(index)
        label = self.get_label(index)
        return img, label
```

使用的是最原始的 DDPM，預設的 sample time step=1000

```python
class Diffusion:
    def __init__(self, noise_steps=1000, beta_start=1e-4, beta_end=0.02, img_size=(240, 320), device="cuda:1"):
        self.noise_steps = noise_steps
        self.beta_start = beta_start
        self.beta_end = beta_end

        self.beta = self.prepare_noise_schedule().to(device)
        self.alpha = 1. - self.beta
        self.alpha_hat = torch.cumprod(self.alpha, dim=0)

        self.img_size = img_size
        self.device = device

    def prepare_noise_schedule(self):
        return torch.linspace(self.beta_start, self.beta_end, self.noise_steps)

    def noise_images(self, x, t):
        sqrt_alpha_hat = torch.sqrt(self.alpha_hat[t])[:, None, None, None]
        sqrt_one_minus_alpha_hat = torch.sqrt(1 - self.alpha_hat[t])[:, None, None, None]
        Ɛ = torch.randn_like(x)
        return sqrt_alpha_hat * x + sqrt_one_minus_alpha_hat * Ɛ, Ɛ

    def sample_timesteps(self, n):
        return torch.randint(low=1, high=self.noise_steps, size=(n,))
```

Unet 使用的是 conditional unet，在最原始的 Unet 使用 positional embedding 的方式搭配 attention layer 把 label 以及 time step 餵進去

```python
class UNet_conditional(nn.Module):
    def __init__(self, c_in=3, c_out=3, time_dim=256, num_classes=None, device="cuda:1"):
        super().__init__()
        self.device = device
        self.time_dim = time_dim
        self.inc = DoubleConv(c_in, 64)
        self.down1 = Down(64, 128)
        self.sa1 = SelfAttention(128, 32)
        self.down2 = Down(128, 256)
        self.sa2 = SelfAttention(256, 16)
        self.down3 = Down(256, 256)
        self.sa3 = SelfAttention(256, 8)

        self.bot1 = DoubleConv(256, 512)
        self.bot2 = DoubleConv(512, 512)
        self.bot3 = DoubleConv(512, 256)

        self.up1 = Up(512, 128)
        self.sa4 = SelfAttention(128, 16)
        self.up2 = Up(256, 64)
        self.sa5 = SelfAttention(64, 32)
        self.up3 = Up(128, 64)
        self.sa6 = SelfAttention(64, 64)
        self.outc = nn.Conv2d(64, c_out, kernel_size=1)

        if num_classes is not None:
            self.label_emb = nn.Sequential(
                                nn.Linear(num_classes, 128),
                                nn.ReLU(),
                                nn.Linear(128, 128),
                                nn.ReLU(),
                                nn.Linear(128, time_dim),
                                nn.ReLU()
                                )
            # self.label_emb = nn.Embedding(num_classes, time_dim)
```

```python
def forward(self, x, t, y):
    t = t.unsqueeze(-1).type(torch.float)
    t = self.pos_encoding(t, self.time_dim)

    if y is not None:
        t += self.label_emb(y) # t:[4, 256]


    x1 = self.inc(x) # x:[4, 3, 240, 320], x1:[4, 64, 240, 320]
    # print(f"{x1.shape=}")
    x2 = self.down1(x1, t) # x2:[4, 128, 120, 160]
    # print(f"{x2.shape=}")
    x2 = self.sa1(x2)
    # print(f"{x2.shape=}")
    x3 = self.down2(x2, t)
    x3 = self.sa2(x3)
    x4 = self.down3(x3, t)
    x4 = self.sa3(x4)

    x4 = self.bot1(x4)
    x4 = self.bot2(x4)
    x4 = self.bot3(x4)

    x = self.up1(x4, x3, t)
    x = self.sa4(x)
    x = self.up2(x, x2, t)
    x = self.sa5(x)
    x = self.up3(x, x1, t)
    x = self.sa6(x)
    output = self.outc(x)
    return output
```

noise scheduler 是使用 linear 的 scheduler

```python
def prepare_noise_schedule(self):
    return torch.linspace(self.beta_start, self.beta_end, self.noise_steps)
```

loss function 是使用 MSE，模型預測的東西是 sample 出來的 noise

```python
x_t, noise = diffusion.noise_images(images, t)
if np.random.random() < 0.1:
    labels = None
predicted_noise = model(x_t, t, labels)
loss = mse(noise, predicted_noise)

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

sampling 的部分使用 classifier free guildance 生成圖片

```python
def sample(self, model, n, labels, cfg_scale=3):
    logging.info(f"Sampling {n} new images....")
    model.eval()
    with torch.no_grad():
        x = torch.randn((n, 3, self.img_size[0], self.img_size[1])).to(self.device)
        for i in tqdm(reversed(range(1, self.noise_steps)), position=0):
            t = (torch.ones(n) * i).long().to(self.device)
            predicted_noise = model(x, t, labels)
            if cfg_scale > 0:
                uncond_predicted_noise = model(x, t, None)
                predicted_noise = torch.lerp(uncond_predicted_noise, predicted_noise, cfg_scale)
            alpha = self.alpha[t][:, None, None, None]
            alpha_hat = self.alpha_hat[t][:, None, None, None]
            beta = self.beta[t][:, None, None, None]
            if i > 1:
                noise = torch.randn_like(x)
            else:
                noise = torch.zeros_like(x)
            x = 1 / torch.sqrt(alpha) * (x - ((1 - alpha) / (torch.sqrt(1 - alpha_hat))) * predicted_noise) + torch.sqrt(beta) * noise
    model.train()
    x = (x.clamp(-1, 1) + 1) / 2
    x = (x * 255).type(torch.uint8)
    return x
```

基本上就是帶入 DDPM paper 的公式給定 xt 以及 predict noise，可以回推到 xt-1，最後*255 轉換成 int 從成圖片 。

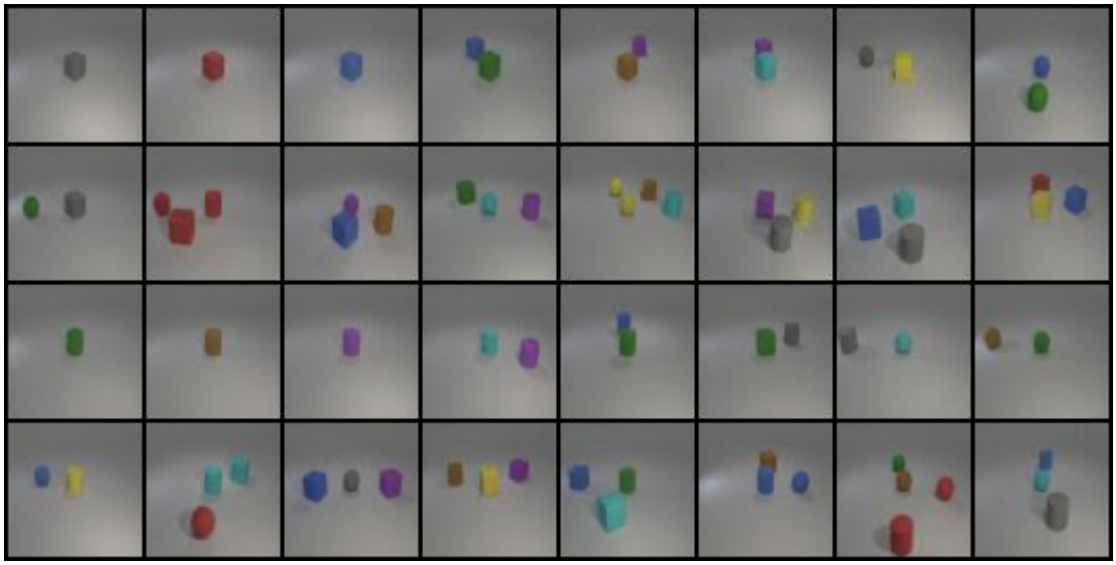
乙、Specify the hyperparameters (learning rate, epochs, etc.) (5%)

lr = 0.0003, img_size = 64 * 64, epoch = 500, batch_size = 6

3. Results and discussion (25%)

甲、Show your results based on the testing data. (5%) (including images)

test data



new_test

test data 以及 new_test data 的 evaulation score

test : 0.861111

new_test : 0.952380

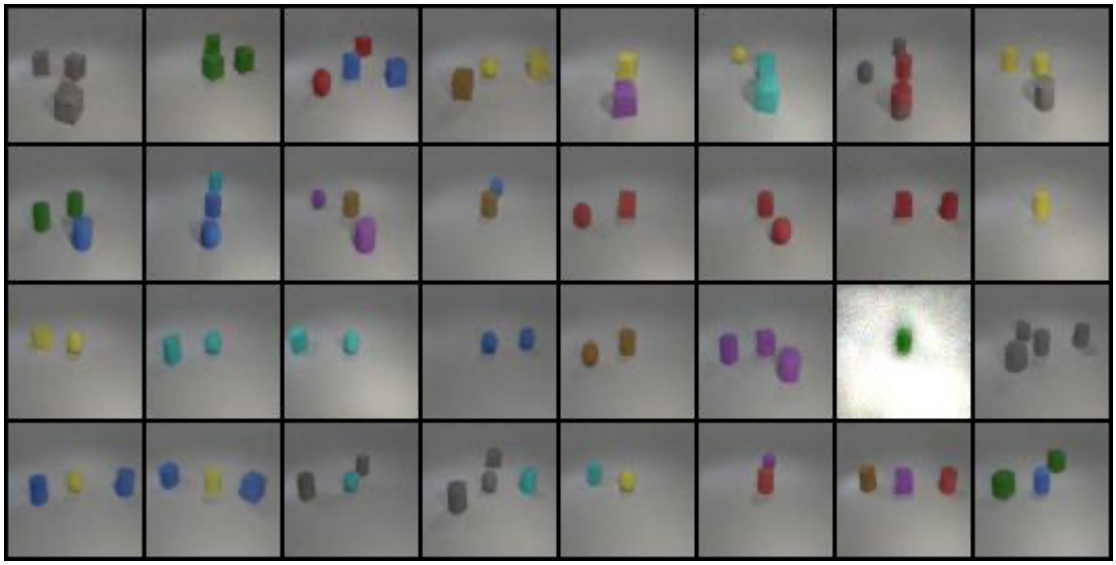
```
(dlp) shlu2240@eva-All-Series:/eva_data2/shlu2240/NCTU_DLP/lab7/TA_dataset$ python3 evaluator.py
/home/shlu2240/anaconda3/envs/dlp/lib/python3.9/site-packages/torchvision/models/_utils.py:208: Us
instead.
  warnings.warn(
/home/shlu2240/anaconda3/envs/dlp/lib/python3.9/site-packages/torchvision/models/_utils.py:223: Us
the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
/eva_data2/shlu2240/NCTU_DLP/lab7/TA_dataset/evaluator.py:108: UserWarning: Creating a tensor from
array() before converting to a tensor. (Triggered internally at /opt/conda/conda-bld/pytorch_16784
  test_labels = torch.tensor(test_labels).to(device)
test_acc: 0.8611111111111112
new test_acc: 0.9523809523809523
```

乙、Discuss the results of different model architectures. (20%) For example, what is the effect with or without some specific embedding methods, or what kind of prediction type is more effective in this case.

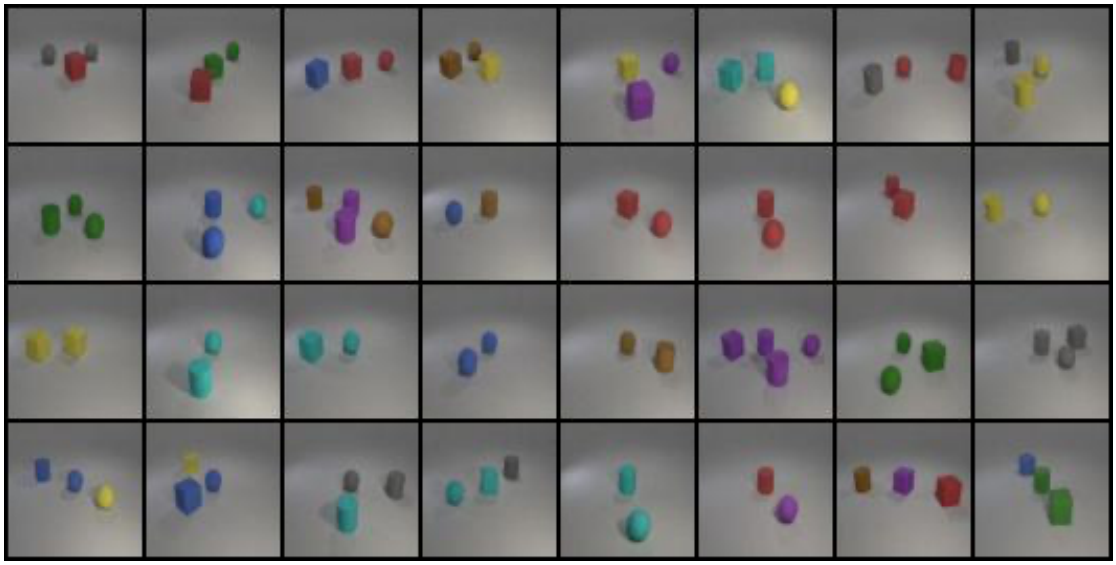
有嘗試使用更深的 embedding layer 來更好的 encode onehot vector，比較兩層 hidden layer 以及一層 hidden layer，在同樣的訓練 epoch 下，更深的 embedding layer 看起來生成的更好

```
self.label_emb = nn.Sequential(
                 nn.Linear(num_classes, 128),
                 nn.ReLU(),
                 nn.Linear(128, 128),
                 nn.ReLU(),
                 nn.Linear(128, time_dim),
                 nn.ReLU()
                 )
```

```
self.label_emb = nn.Sequential(
                 nn.Linear(num_classes, 128),
                 nn.ReLU(),
                 # nn.Linear(128, 128),
                 # nn.ReLU(),
                 nn.Linear(128, time_dim),
                 nn.ReLU()
                 )
```



(deeper embedding layer)



也有使用 classifier free guidance 來生成指定 label 的圖片，在訓練時 0.1 的比例不使用 label，0.9 的比例使用 label，研究顯示可以讓模型更 general，進而提升生成品質。

```
predicted_noise = model(x, t, labels)
if cfg_scale > 0:
    uncond_predicted_noise = model(x, t, None)
    predicted_noise = torch.lerp(uncond_predicted_noise, predicted_noise, cfg_scale)
```

也有嘗試使用 EMA 來讓訓練更加穩定，基本上就是複製一份模型，使用學習中的模型以特定比例來更新複製的模型，所以要是模型有哪個 step 突然受到離群值影響大幅更新，因為 EMP 只會以比例更新 reference model，所以 reference model 就不會受到太大的影響，進而讓訓練變得更加 smooth

```
class EMA:
    def __init__(self, beta):
        super().__init__()
        self.beta = beta
        self.step = 0

    def update_model_average(self, ma_model, current_model):
        for current_params, ma_params in zip(current_model.parameters(), ma_model.parameters()):
            old_weight, up_weight = ma_params.data, current_params.data
            ma_params.data = self.update_average(old_weight, up_weight)

    def update_average(self, old, new):
        if old is None:
            return new
        return old * self.beta + (1 - self.beta) * new

    def step_ema(self, ema_model, model, step_start_ema=2000):
        if self.step < step_start_ema:
            self.reset_parameters(ema_model, model)
            self.step += 1
            return
        self.update_model_average(ema_model, model)
        self.step += 1

    def reset_parameters(self, ema_model, model):
        ema_model.load_state_dict(model.state_dict())
```