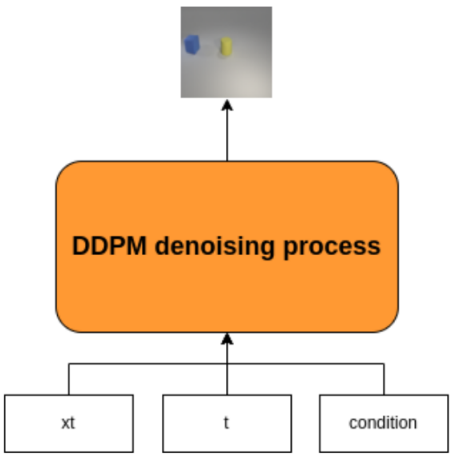
Lab 6 Diffusion model

411551007 林穎志

1. Introduction

We implement the conditional Denoising Diffusion Probabilistic Models (DDPM) in this lab. While giving the input image x_t, time step t and condition, the DDPM should generate the corresponding synthetic images as the figure shown below. There are totally 24 labels in this task.

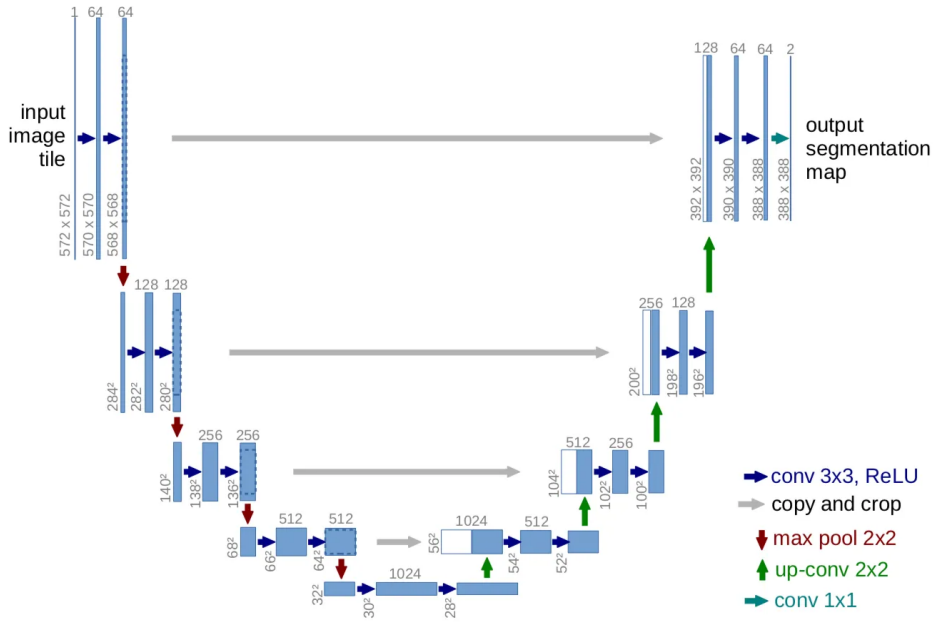


2. Implementation details

- Model implementation, Choice of DDPM, Unet

Batch size	Learning rate	Epochs	Prediction type	Model
128	0.0001	150	Epsilon	Unet

    - We build the model with the class object in diffusers.
    - Architecture
        - As the DDPM needs to implement noise prediction in the phase of denoising process, we maintain the architecture of Unet in DDPM. Unet is an Ecoder / Decoder architecture. Including downsample blocks, upsample blocks and skip connection.




```
model = MyConditionedUNet(
            sample_size=sample_size,
            in_channels=3,
            out_channels=3,
            layers_per_block=layers,
            block_out_channels=(block_dim, block_dim, block_dim*2, block_dim*2, block_dim*4, block_dim*4),
            down_block_types=(
                "ResnetDownsampleBlock2D",
                "ResnetDownsampleBlock2D",
                "ResnetDownsampleBlock2D",
                "ResnetDownsampleBlock2D",
                "AttnDownBlock2D",
                "ResnetDownsampleBlock2D",
            ),
            up_block_types=(
                "ResnetUpsampleBlock2D",
                "AttnUpBlock2D",
                "ResnetUpsampleBlock2D",
                "ResnetUpsampleBlock2D",
                "ResnetUpsampleBlock2D",
                "ResnetUpsampleBlock2D",
            ),
            class_embed_type=embed_type,
        )
```

- We add an embedding layer to convert the dimension of condition vector to the same dimension as time embedding. The forward function is also designed with respect to the class embedding. We do addition between the timestep embedding and the class embedding. Both information will be learned in the model.

```
# class embedding
self.class_embedding = nn.Linear(24, time_embed_dim)  # time_embed_dim = 512

def forward(self, x, t, class_labels):
    class_emb = self.class_embedding(class_labels).to(dtype=self.dtype)
    emb = emb + class_emb
```

- Main idea of training DDPM:
  - Sample some random noises and time steps to the noise scheduler. It will output the noisy image.
  - Input the noise image, timesteps and hte condition into the conditionUNet and output the prediction of noise.
  - Calculate the MSE loss between the given sample noises and the prediction noise then do propagation.

- Noise schedule
  - We implement the noise schedule by the API DDPMScheduler and the parameter settings are shown as below:

```
noise_scheduler = DDPMScheduler(num_train_timesteps=1000, beta_schedule=beta_schedule)
```

- Loss functions
  - MSE loss is chosen to calculate the difference between the sample noise and the noise prediction.

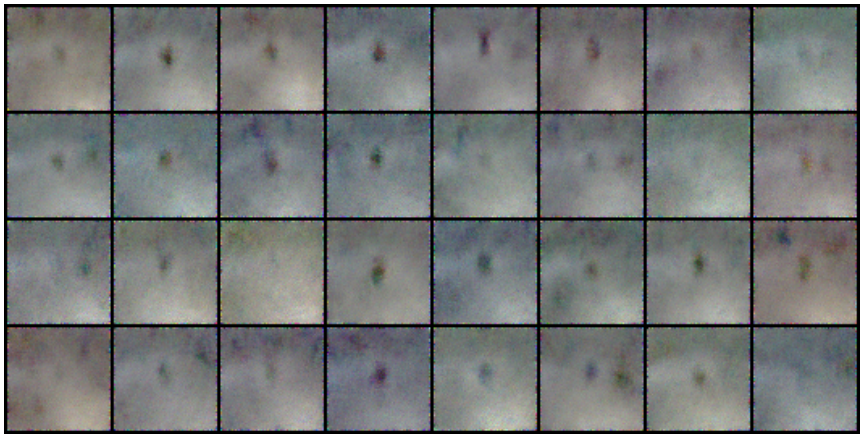
```
loss_fn = nn.MSELoss()
loss = loss_fn(noise_pred, noise)
```

3. Results and Discussion

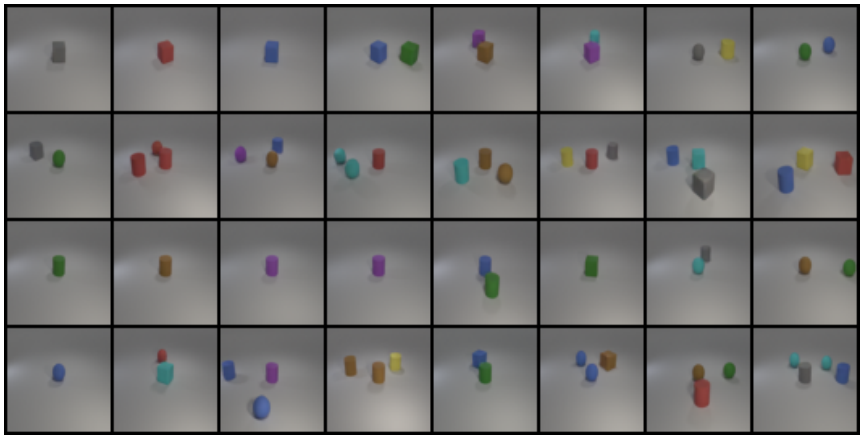
A. Screenshot of accuracy

B. Synthetic image grids and a progressive generation image

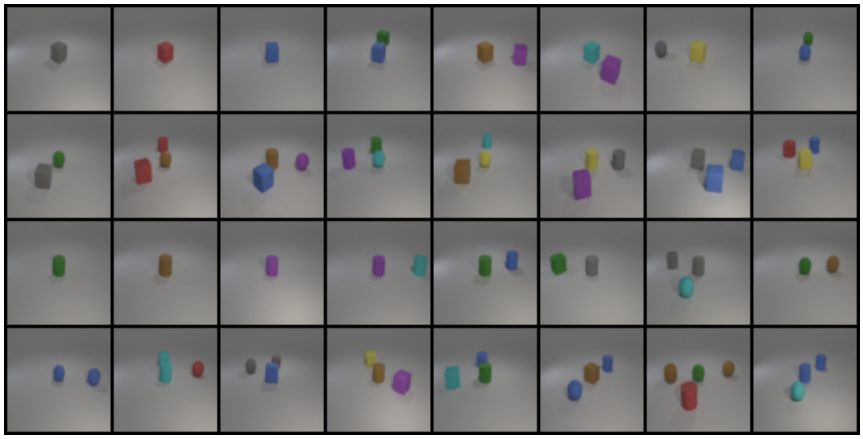
- New test:
  - **Accuracy: 0.9286**
  - Epoch: 0



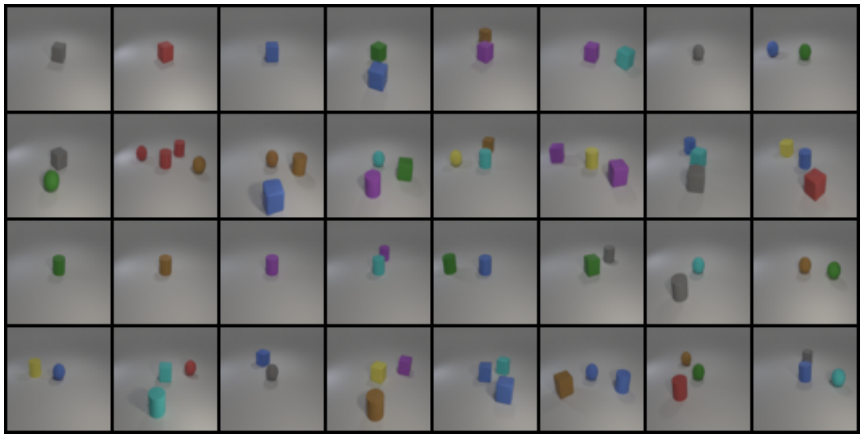
- Epoch: 50



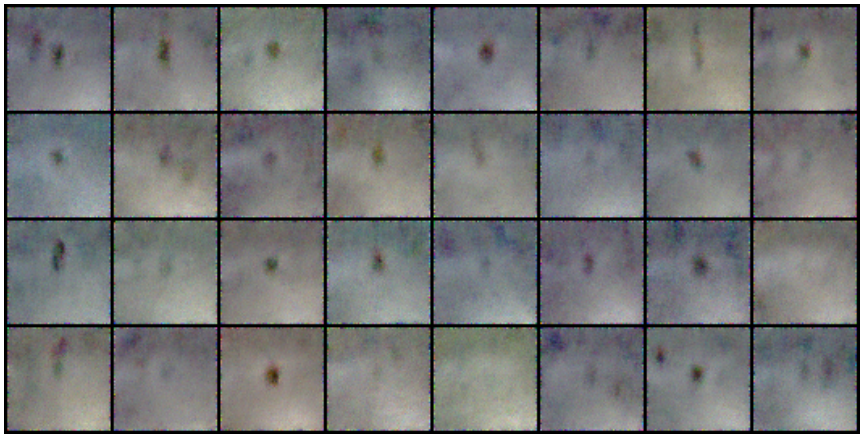
- Epoch: 100



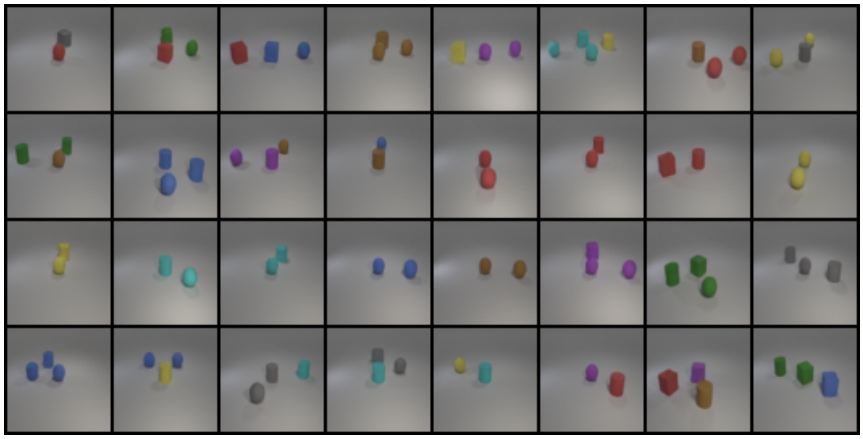
- Epoch: 150



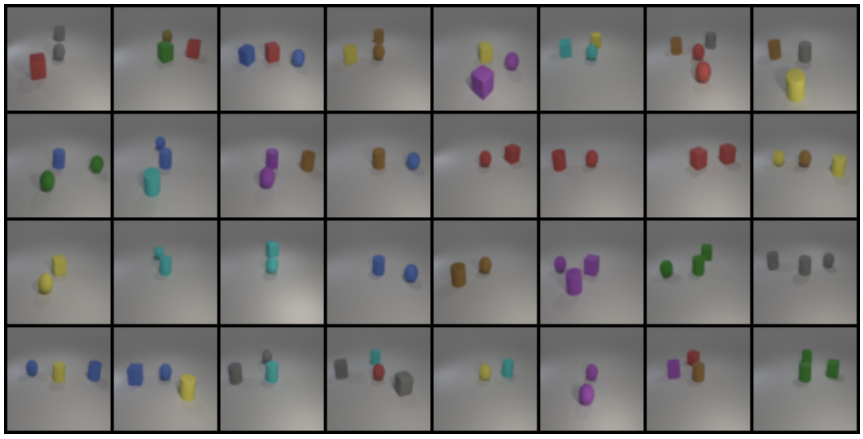
- Test:
  - **Accuracy: 0.8750**
  - Epoch: 0



  - Epoch: 50



  - Epoch: 100



- Epoch: 150

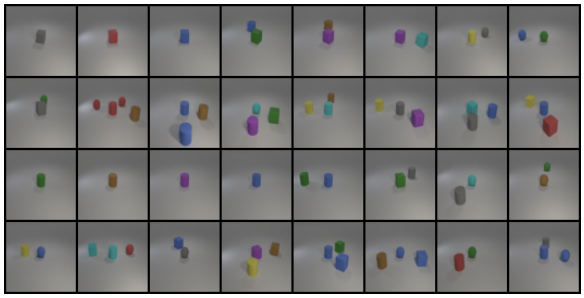
C. Discussion

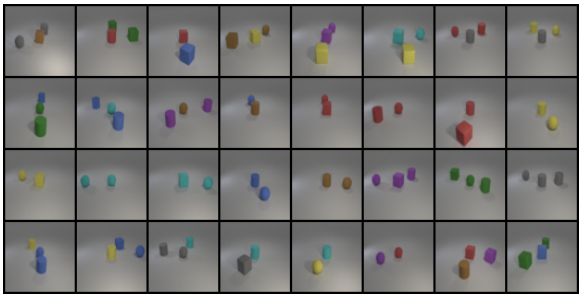
A. Different embedding method

As you can see the code below, we use a special label embeddings to embed our one hot vector condition. The result shows that adding such a design cannot improve the accuracy.

Accuracy	Test	New test
Original	0.8750	0.9286
Special label embedding	0.7361	0.8333

```
if args.label_emb:
    self.class_embedding = nn.Sequential(nn.Linear(24, int(time_embed_dim/4)),
                                         nn.ReLU(),
                                         nn.Linear(int(time_embed_dim/4), int(time_embed_dim/2)),
                                         nn.ReLU(),
                                         nn.Linear(int(time_embed_dim/2), time_embed_dim)
                                         )
```

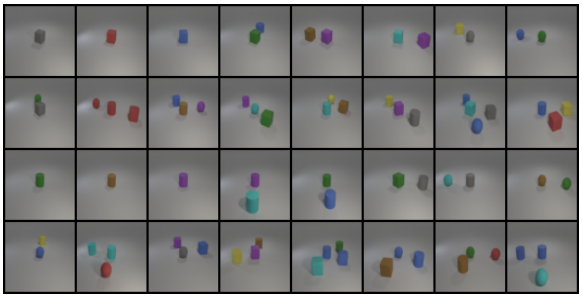




B. Different beta schedule

We change the cosine noise schedule to linear noise schedule. The result show that the original cosine schedule we use is better than linear scheduel.

Accuracy	Test	New test
Original (Cosine)	0.8750	0.9286
Linear	0.8571	0.8889



C. Different UNet architecture

We change the original downsample/ upsample block to Resblock. Which same as the conception of ResNet, adding a skip connection between 2 convolution blocks. The result shows that using Resblock does improve the performance.

Accuracy	Test	New test
Original	0.8750	0.9286
ResBlock	0.9028	0.9286