Deep Learning – Lab 6

Lab6 - Let's Play DDPM

I. Introduction

- Deep Diffusion Probabilistic Model (DDPM) is a powerful generative model that excels in capturing complex data distributions. It leverages a diffusion process to iteratively refine a noise-corrupted input, gradually revealing the underlying data. By integrating attention mechanisms and conditioning on relevant information, such as object labels, DDPM enhances its ability to learn intricate relationships within the data. This unique architecture, guided by a carefully designed loss function like MSE, enables DDPM to effectively denoise and generate high-quality samples while offering a versatile framework for various tasks.

II. Implementation details

i. Unet

```
eps_model = MyConditionedUNet(
    sample_size=64,
    in_channels = 3,
    out_channels = 3,
    layers_per_block = 2,
    block_out_channels = (64, 128, 256, 256, 512, 512),
    down_block_types = (
        "DownBlock2D",
        "AttnDownBlock2D",
        "DownBlock2D",
        "AttnDownBlock2D",
        "DownBlock2D",
        "AttnDownBlock2D",
    ),
    up_block_types=(
        "AttnUpBlock2D",
        "UpBlock2D",
        "AttnUpBlock2D",
        "UpBlock2D",
        "AttnUpBlock2D",
        "UpBlock2D",
    ),
    class_embed_type='timestep',)
```

- I choose to use a total of 6 layers for the overall architecture, which can be divided into three blocks. Each block consists of a network layer and an attention layer. I aim to compute attention after each network output to generate an attention map. This approach allows the network to focus more on the correlations between different pixels (features) and enhance their relevancy.

ii. DDPM

- I opt for the standard Deep Diffusion Probabilistic Model (DDPM) architecture, while incorporating object labels as conditions.

iii. noise schedule

- I find that employing the cosine noise schedule yields significantly better results compared to using a linear schedule.

iv. loss functions

- Utilizing the Mean Squared Error (MSE) between the denoised output and the ground truth as

the loss function.

```
parser = argparse.ArgumentParser(description = __doc__)
parser.add_argument('--test_only', action = 'store_true')
parser.add_argument('--model_path', default = './Best_UNet_model.pkl')
parser.add_argument('--data_path', default = './data')
parser.add_argument('--save_image_path', default = './image')
parser.add_argument('--seed', default = 'Lalalalex')
parser.add_argument('--epoch', default = 100, type = int)
parser.add_argument('--n_samples', default = 64, type = int)
parser.add_argument('--train_time_steps', default = 1000, type = int)
parser.add_argument('--image_channel', default = 3, type = int)
parser.add_argument('--image_size', default = 64, type = int)
parser.add_argument('--batch_size', default = 32, type = int)
parser.add_argument('--learning_rate', default = 1e-4, type = float)
args = parser.parse_args()
```

v. hyperparameters

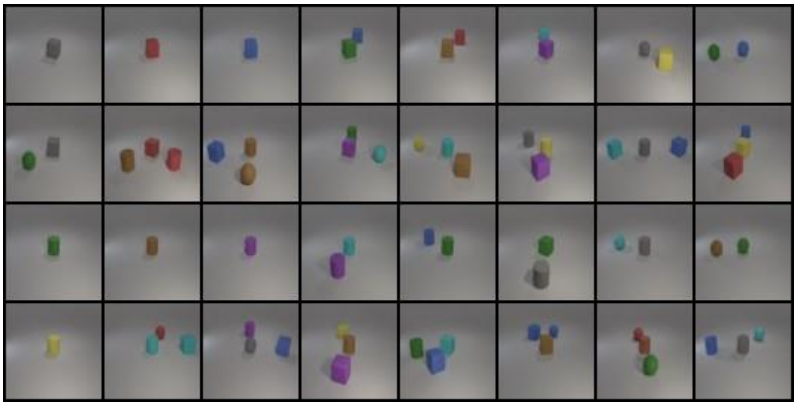
III. Results and discussion

i. Show your accuracy screenshot based on the testing data

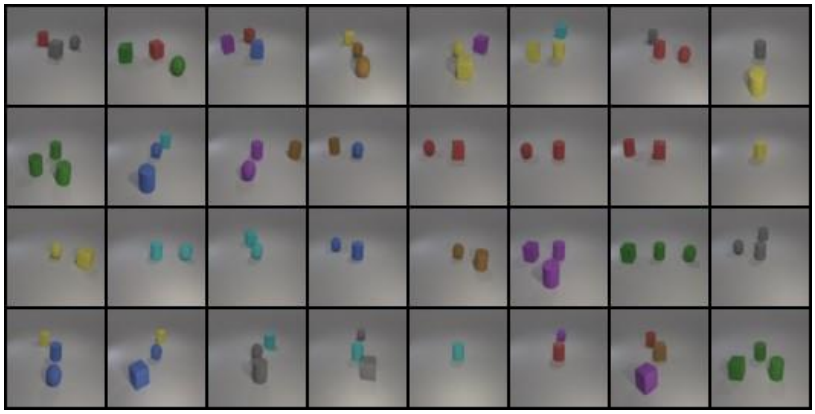
```
Test: 0.8194444444444444
New Test: 0.8690476190476191
```

ii. Show your synthetic image grids and a progressive generation image.

- Test



- New test



iii. Discuss the results of different model architectures or methods.

- I've experimented with modifying the architecture by concatenating embeddings along the channel dimension and passing them through a CNN for output, but this approach didn't yield satisfactory results.
- I've also attempted to use Attention only in the final layer while employing regular blocks for the rest. While this led to a faster increase in accuracy, however, the final performance wasn't as strong.

IV. Reference

- DDPM and UNet Library: https://github.com/huggingface/diffusers
- The Unet.py is reference this: https://github.com/MichaelLee-ceo/NYCU-DLP/blob/main/Lab7/code/generate.py