Lab6 - Let's Play DDPM

Ting-Han, Lin

- Deadline: August 29, 2023, 23:55
- Format: Experimental report (.pdf) and source code (.py). Zip all files in one file and name it like DLP_LAB6_yourstudentID_name.zip.

1 Lab Description

In this lab, you need to implement a conditional Denoising Diffusion Probabilistic Model (DDPM) to generate synthetic images according to multi-label conditions. Recall that we have seen the generation capability of conditional VAE in Lab 4. To achieve higher generation capacity, especially in computer vision, DDPM is proposed and has been widely applied to style transfer and image synthesis. In this lab, given a specific condition, your model should generate the corresponding synthetic images (see Fig. 1). For example, given "blue cube" and "yellow cylinder", your model should generate the synthetic images with a blue cube and a yellow cylinder and meanwhile, input your generated images to a pre-trained classifier for evaluation.

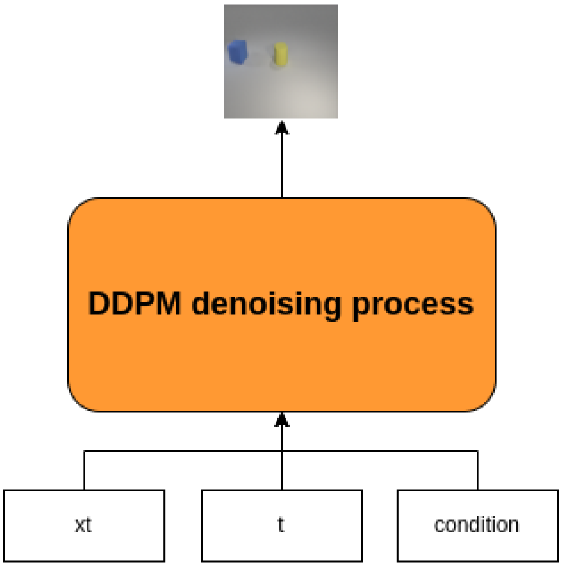

Figure 1: The illustration of conditional DDPM

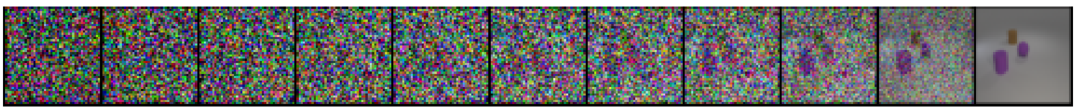

Figure 2: The progressive generation example

2 Requirements

- **Implement a conditional DDPM**
  - Choose your conditional DDPM setting.
  - Design your noise schedule and UNet architecture
  - Choose your loss functions.
  - Implement the training function, testing function, and data loader.
- **Generate the synthetic images**
  - Evaluate the accuracy of test.json and new_test.json.
  - Show the synthetic images in grids for two testing files.
  - Plot the progressive generation process for generating an image.

3 Implementation Details

3.1 Type of DDPM

The diffusion and denoising process of DDPM can be performed on pixel or latent domain. Pixel-based DDPM models the data distribution by applying a sequence of diffusion steps to a simple base distribution, such as a standard normal distribution, to obtain a more complex distribution that approximates the true data distribution. The diffusion process involves repeatedly adding noise to the data and then gradually reducing the noise level over the diffusion steps.

On the other hand, the Latent Diffusion Model is a variant of the denoising diffusion probabilistic model that operates on the latent space of a pre-trained neural network. Instead of directly modeling the data distribution, it models the distribution of the latent representations of the data. It also uses a sequence of diffusion steps, but these steps are applied to the latent representations.

In this lab, you can choose DDPM to perform on pixel or latent space. Also, you need to select the noise schedule and condition embedding for time steps and labels.

3.2 Design of UNet

To model the conditional distribution of the image given its noisy version, many of DDPM's architectures are based on UNet. The UNet consists of a downsampling path, where the input image is gradually downscaled through a series of convolutional layers, and an upsampling path, where the output image is gradually upscaled back to its original size through a series of transpose convolutional layers. Also, there are some skip connections between features of same resolution.

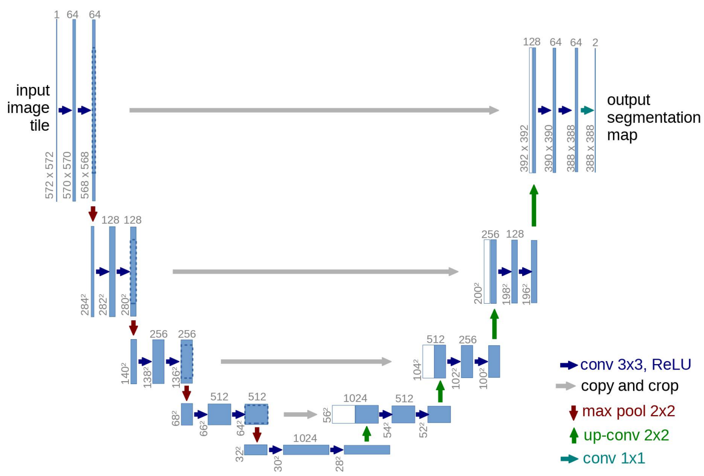

Figure 3: Unet architecture

3.3 Other implementation details

- You can choose any DDPM architecture you like. Also, feel free to use any library that helps your implementation (e.g. Hugging face diffuser), but you need to include it in your report's reference.
- Except for the provided files, you cannot use other training data. (e.g. background image)
- Use the function of a pre-trained classifier, *eval*(images, labels) to compute accuracy of your synthetic images.
- Use *make_grid*(images) and *save_image*(images, path) from torchvision_utils import save_image, make_grid to save your image (8 images a row, 4 rows).
- The same object will not appear twice in an image.
- You should normalize the RGB images with transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) before inputting them into the pre-trained classifier, and apply de-normalization while generating RGB images.

4 Dataset Descriptions

You can download file.zip from new e3, except iclevr.zip in open-source google drive (link also provided on e3). There are 7 files in the file.zip: readme.txt, train.json, test.json, new_test.json, object.json, evaluator.py, and checkpoint.pth. All the details of the dataset are in the readme.txt.

5 Scoring Criteria

1. Report (50%)
   - Introduction (5%)
   - Implementation details (20%)
     - Describe how you implement your model, including your choice of DDPM, UNet architectures, noise schedule, and loss functions. (15%)
     - Specify the hyperparameters (learning rate, epochs, etc.) (5%)
   - Results and discussion (25%)
     - Show your accuracy screenshot based on the testing data. (5%)
     - Show your synthetic image grids and a progressive generation image. (10%)
     - Discuss the results of different model architectures or methods. (10%) 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.
2. Experimental results (50%) (based on results shown in your report)
   - Classification accuracy on test.json and new_test.json. (25%+25%)

Accuracy	Grade
score $\geq$ 0.8	100%
0.8 > score $\geq$ 0.7	90%
0.7 > score $\geq$ 0.6	80%
0.6 > score $\geq$ 0.5	70%
0.5 > score $\geq$ 0.4	60%
score < 0.4	0%

6 Output Examples

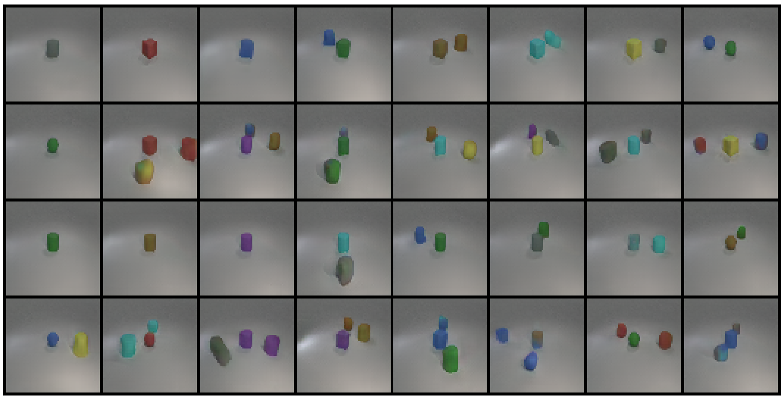

Figure 4: The synthetic images of F1-score of 0.638

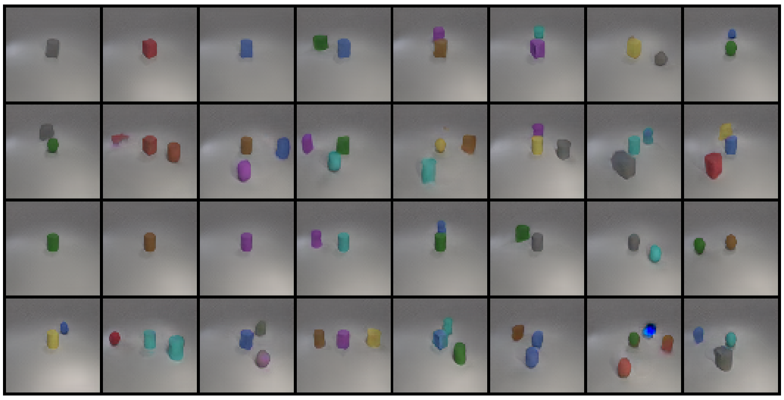

Figure 5: The synthetic images of F1-score of 0.806