Lab6 - Generative Models

Ting-Han Lin

- **Deadline: June 6, 2024, 11:59 a.m.**
- **Format: Experimental report (.pdf) and source code (.py). Zip all files in one file and name it DL_LAB6_YourStudentID_YourName.zip.**
- **No demo for this lab.**

1. Lab Description

Recall that in Lab 4, we explored the generative capabilities of conditional VAEs, and in Lab 5, MaskGIT we used is based on VQGAN (though we did not implement the GAN part). This time, we will learn about two more powerful generative models: **GAN and DDPM**.

In this lab, you need to implement two kinds of mentioned generative models to generate synthetic images according to multi-label conditions: Given a specific set of label conditions, your GAN and DDPM should generate the corresponding synthetic images (see Fig. 1). For example, “red sphere,” “yellow cube” and “gray cylinder” are given, your model should generate the synthetic images containing those objects. Meanwhile, you also need to input your generated images into a pre-trained classifier for evaluation.

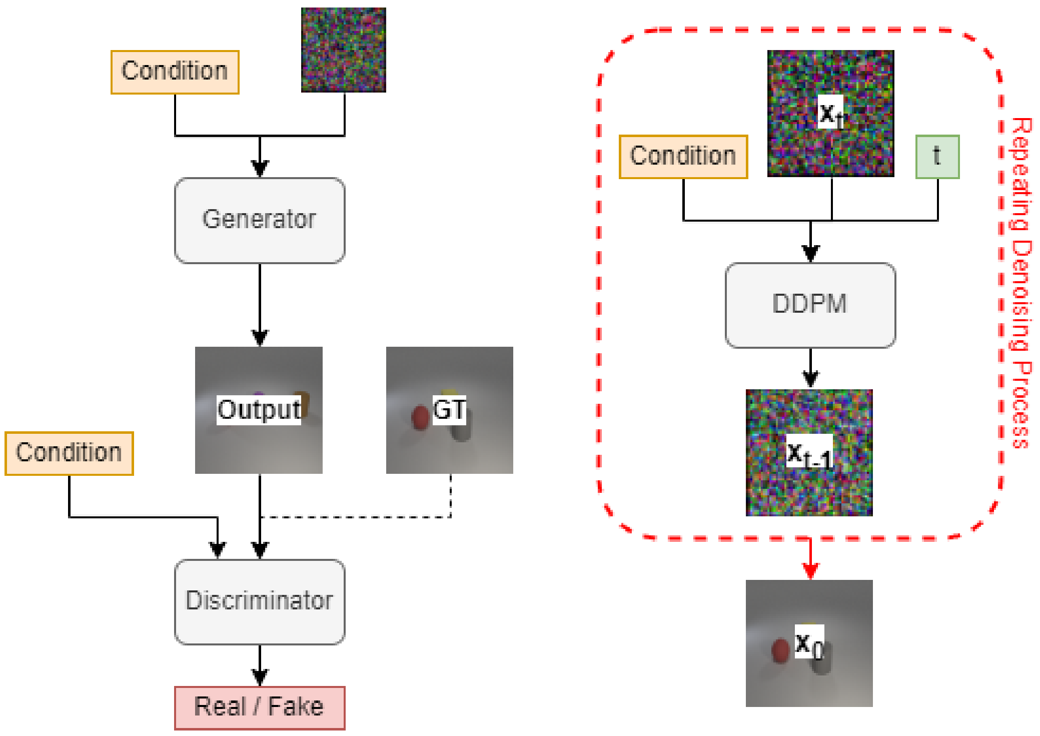

Fig. 1: The overview of conditional GAN (left) and conditional DDPM (right).

2. Requirements

- Prepare dataset

- Implement data loader
- Define your multi-label condition embedding method

- Implement a conditional GAN

- Choose your conditional GAN architecture
- Design your generator and discriminator
- Choose your loss function, maybe including the pretrained evaluator in it
- Implement the training function and testing function, also apply some training tricks if necessary

- Implement a conditional DDPM

- Choose your conditional DDPM setting, design your model architecture
- Design your noise schedule, time embeddings, sampling method, etc. You can also include the pretrained evaluator as a "classifier guidance" in sampling
- Choose your loss functions (e.g. reparameterization type)
- Implement the training function, testing function

- Evaluate the results

- Evaluate the accuracy of test.json and new test.json, input the output images of each model to the pretrained evaluator to get the accuracies
- Show the synthetic images in grids for two models * two testing files
- Show the denoising process in a grid for sampling an image from your DDPM, **with the label set ["red sphere," "yellow cube," "cyan cylinder"]**
- Compare the advantages and disadvantages of the GAN and DDPM models (e.g. in the view of training, generation capacity, potential problems, etc)

3. Implementation Details

3.1 Basic Rules

- For each model, **you can choose any architecture you like**. Also, feel free to use any library that helps your work, but you need to mention the implementation details and include the reference in your report.
- Except for the provided files, you cannot use other training data. (e.g. background image)
- Use the function of a pre-trained evaluator to compute the accuracy of your synthetic images.
- The evaluator is based on ResNet18. By inputting synthesized images and one-hot GT labels, the evaluator will return accuracy. Its class script and pretrained weight are given. **DO NOT modify any of them, BUT involving the classification result in your discriminator or classifier guidance is allowed. Please inherit the class if you need extra functionalities.**

- The normalization of input images is ***transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))***. **You must input the normalized image into the pretrained classifier**. You should apply denormalization to generate RGB images.
- The resolution of input for the pretrained evaluator is 64x64. You can design your own output resolution and resize it for evaluation
- Use ***make_grid(images)*** from torchvision.utils to generate grids for your testing image (8 images a row, 4 rows for each JSON file) and denoising process (at least 8 images in 1 row from noisy to clear).
- If you have no idea how to design your GAN and DDPM, follow 3.2 and 3.3 for simple suggestions.

3.2 DCGAN

- Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks
- Pytorch DCGAN tutorial

You can follow the above paper and tutorial to learn how to implement an unconditional DCGAN. Further modifying the design into conditional architecture and applying some advanced techniques will help.

3.3 DDPM

- Denoising Diffusion Probabilistic Models
- Hugging Face Diffusion Models Course

The Hugging Face Diffuser library offers comprehensive functionality to help you implement various diffusion model architectures, scheduling methods, sampling types, reparameterizations, etc. You can refer to the tutorial linked above to design your diffusion model for this lab.

4. File Descriptions

You can download file.zip from e3, except iclevr.zip, from the open-source Google Drive (link). 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

- Report (60%)

- Introduction (5%)
- Implementation details (20%)
  - Describe the implementation details of the two models, which are listed in part 2. (10% for each model)

- Results and discussion (35%)
    - Show your synthetic image grids (total 12%: 3% * 2 models * 2 testing data) and a denoising process image (3%)
    - Compare the advantages and disadvantages of the GAN and DDPM models (10%)
    - Discussion of your extra implementations or experiments (10%)

- Experimental results (40%) (based on results shown in your report)

- Classification accuracy on test.json and new test.json (40%)
    - Show your accuracy screenshots
    - 10% * 2 models * 2 testing data

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

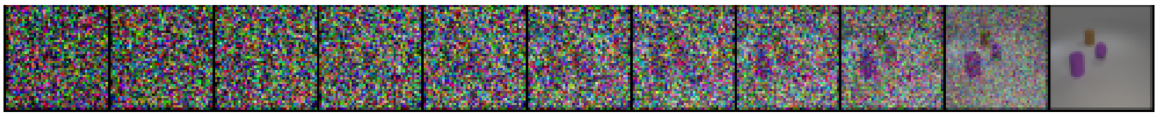

Fig. 2: Example of the denoising process image.

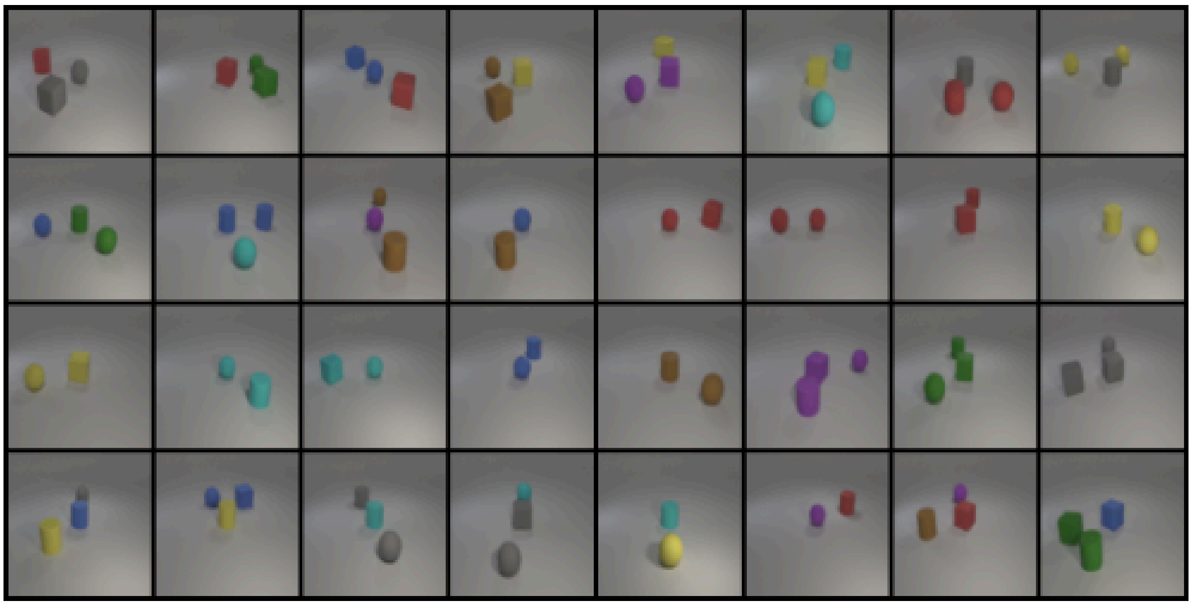

Fig. 3: The synthetic image grid on new_test.json. (F1-score 0.821)