Lab5 - Let's Play GANs

Mei-Hui, Li

- Deadline: August 24, 2021 11:55 a.m.
- Demo Date: August 24, 2021
- Format: Experimental report (.pdf) and source code (.py). Zip all files in one file and name it like DLP_LAB5_yourstudentID_name.zip.

1 Lab Description

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

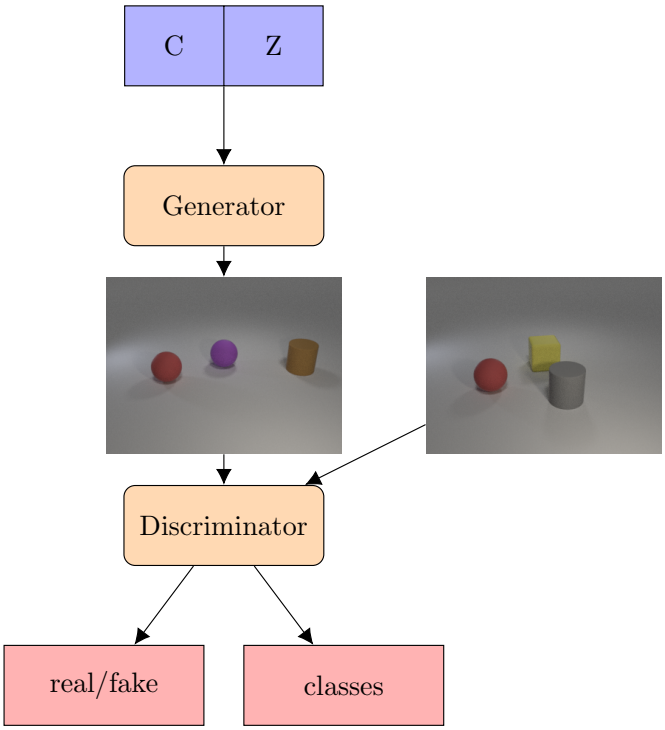

Figure 1: The illustration of cGAN.

2 Requirements

- **Implement a conditional GAN**
  - Choose your cGAN architecture.
  - Design your generator and discriminator.
  - Choose your loss functions.
  - Implement the training function, testing function, and data loader.
- **Generate the synthetic images**
  - Evaluate the accuracy of test.json
  - Evaluate the accuracy of new_test.json which will be released before demo.

3 Implementation Details

3.1 Architecture of conditional GAN

There are Variants of conditional GAN architectures. You can choose one of the models in the following list or a hybrid version as your conditional GAN structure

- conditional GAN [7]
- Auxiliary GAN [9]
- Projection discriminator [8]

3.2 Design of generator and discriminator.

Many GAN structures are developed to generator high quality images. Basically, all GAN architecture are based on DCGAN which is mainly composed of convolution neural network. You can adopt one of the structures or a hybrid

- DCGAN
- Super resolution GAN [5]
- Self-attention GAN [10]
- Progressive growing of GAN [4]

3.3 Training loss functions

Besides the significant progress of GAN architectures, the training function is also play an important role. Your loss function should combine with your choice of cGAN as it will take a part in your training function, e.g., auxiliary loss. Here are the reference training loss functions.

- GAN [2]
- LSGAN [6]
- WGAN [1]
- WGAN-GP [3]

3.4 Other implementation details

- Your can ignore previous suggestion and choose any GAN architecture you like.
- 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.
- The normalization of input images is transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)). You should not apply normalization on your generated images but apply de-normalization while generating RGB images.

4 Dataset Descriptions

You can download the lab5_gans.zip file in open source google drive. There are 7 files and one folder in the lab5_gans.zip: readme.txt, train.json, and test.json, object.json, classifier_weight.pth, dataset.py, evaluator.py, and images folder. All the details of the dataset are in the readme.txt.

5 Scoring Criteria

1. Report (40%)
   - Introduction (5%)
   - Implementation details (15%)
     - Describe how you implement your model, including your choice of cGAN, model architectures, and loss functions. (10%)
     - Specify the hyperparameters (learning rate, epochs, etc.) (5%)
   - Results and discussion (20%)
     - Show your results based on the testing data. (5%) (including images)
     - Discuss the results of different models architectures. (15%) For example, what is the effect with or without some specific loss terms, or what kinds of condition design is more effective to help cGAN.
2. Demo (60%)
   - Classification accuracy on test.json and new_test.json. (20% + 20%)

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

   - Questions (20%)

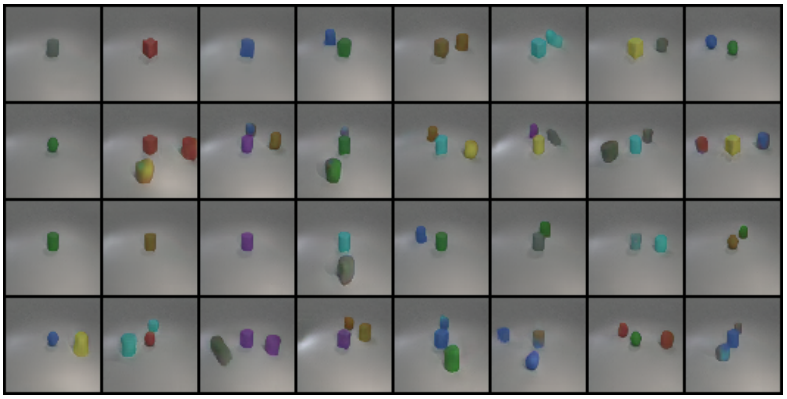

Figure 2: The synthetic images of F1-score of 0.638

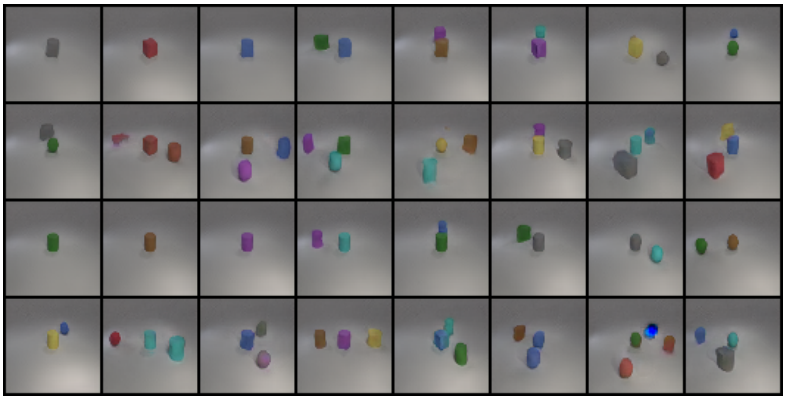

Figure 3: The synthetic images of F1-score of 0.806

6 Output Examples

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[3] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[4] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2017.

[5] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2016.

[6] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2016.

[7] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[8] Takeru Miyato and Masanori Koyama. cgans with projection discriminator, 2018.

[9] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans, 2016.

[10] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2018.