AI604: Deep Learning for CV
Fall 2020
Homework 3

Due on Nov 21 (in KLMS or email: course.davian@gmail.com)

Implementation: DCGAN and CycleGAN

The objective of this assignment is to generate the images from a normal distribution Z (DCGAN) and to translate an image from a source domain to a target domain in the absence of paired examples (CycleGAN). Although there exist many variants of the model architecture, you should strictly follow this guidelines to pass all test examples.

Setting up the environment

First, you need to download a Summer2Winter dataset by typing the following command line:

```
bash ./data/download_dataset.sh
```

Second, you need to create a conda environment, as follows:

```
conda env create -f env.yml
conda activate assn3
```

You can test your implementation by running the provided test codes located in each `.py` file.

1 Unconditional Image Generation using DCGAN

DCGAN (Deep Convolutional Generative Adversarial network) presented in 2016 have suggested the optimal model architecture for generating the 64×64 images from a normal distribution Z. You need to implement DCGAN to understand the basic structures and training process of GANs and compare various loss functions to check out the performance difference.

1.1 (10 points) Model

The basic structure of the model is given by skeleton codes, with empty parts in the model's constructor and `forward` function. You need to complete the class `DCGAN_Generator` and `DCGAN_Discriminator` in `dcgan.py`. Your model should follow the model architecture details in Table 1.1. Otherwise, you may not be able to pass the test cases. Check the comments in the code for implementation details.

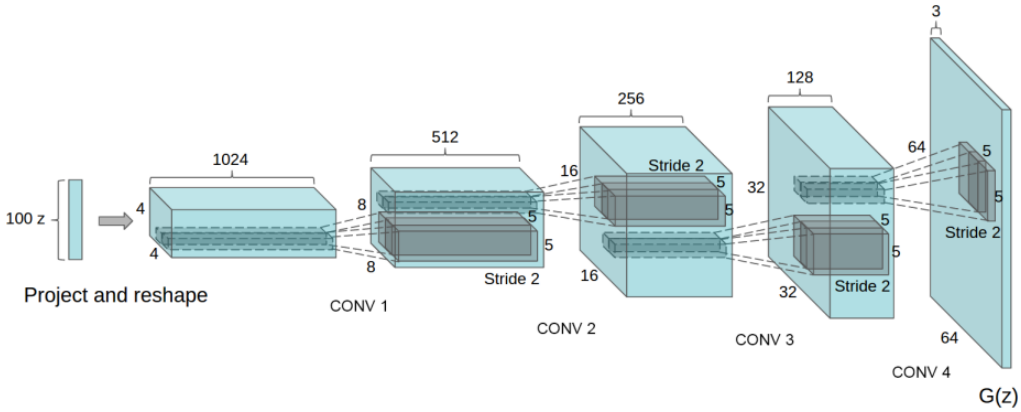

Figure 1: Overview of the DCGAN generator from the official paper. This figure shows a model for generating 64×64 images. Therefore, you should check Table 1.1 to implement DCGAN for CIFAR-10 (32×32) dataset.

Part	Layer	Input	Output	Layer Information
Generator	Layer1	(1, 1, 256)	(4, 4, 256)	Deconv-(N256, K4, S1, P0), BN, ReLU
	Layer2	(4, 4, 256)	(8, 8, 128)	Deconv-(N128, K4, S2, P1), BN, ReLU
	Layer3	(8, 8, 128)	(16, 16, 64)	Deconv-(N64, K4, S2, P1), BN, ReLU
	Layer4	(16, 16, 64)	(32, 32, 3)	Deconv-(N3, K4, S2, P1), Tanh
Discriminator	Layer1	(32, 32, 3)	(16, 16, 64)	Conv-(N64, K4, S2, P1), BN, LeakyReLU
	Layer2	(16, 16, 64)	(8, 8, 128)	Conv-(N128, K4, S2, P1), BN, LeakyReLU
	Layer3	(8, 8, 128)	(4, 4, 256)	Conv-(N256, K4, S2, P1), BN, LeakyReLU
	Layer4	(4, 4, 256)	(1, 1, 1)	Conv-(N1, K4, S1, P0)

Table 1: DCGAN architecture details. DCGAN consists of a generator and a discriminator. We use several notations; Conv: the convolution, Deconv: the transposed convolution, N: the number of the output channels, K: the kernel size, S: the stride size, P: the padding size, BN: batch normalization. The slope of LeakyReLU in a negative regime is 0.2.

1.2 (20 points) Training

With the model implemented in the previous section, you need to implement the class `DCGAN_Solver`. The discriminator network tries to distinguish between real and fake images, while the generator network tries to fool the discriminator by generating real-looking images. You need to implement a total of four loss functions including GAN, LSGAN, WGAN and WGAN-GP. For WGAN-GP loss function, complete and use `GPLoss` in `dcgan.py`.

2 Unpaired Image-to-Image Translation using CycleGAN

CycleGAN is an image-to-image translation model which translates an image from a source domain to a target domain without input-output image pairs. In this section, you need to implement the model that translates a summer picture into a winter picture in similar scene structures.

2.1 (10 points) Model

You need to implement the class `CycleGAN_Generator` and `CycleGAN_Discriminator` in `cyclegan.py`. Your model should follow the model architecture details in Table 2. Otherwise, you may not be able to pass the test cases. Check the comments in the code for implementation details.

Generator			
Layer	**Input**	**Output**	**Layer Information**
Downsample	(256, 256, 3)	(256, 256, 64)	Conv-(N64, K7, S1, RP3, B), IN, ReLU
Downsample	(256, 256, 64)	(128, 128, 128)	Conv-(N128, K3, S2, P1, B), IN, ReLU
Downsample	(128, 128, 128)	(64, 64, 256)	Conv-(N256, K3, S2, P1, B), IN, ReLU
Residual Block×6	(64, 64, 256)	(64, 64, 256)	Conv-(N256, K3, S1, RP1, B), IN, ReLU Conv-(N256, K3, S1, RP1, B), IN
Upsample	(64, 64, 256)	(128, 128, 128)	Deconv(N128, K3, S2, P1, OP1, B), IN, ReLU
Upsample	(128, 128, 128)	(256, 256, 64)	Deconv(N64, K3, S2, P1, OP1, B), IN, ReLU
Upsample	(256, 256, 64)	(256, 256, 3)	Conv(N3, K7, S1, RP3), Tanh
Discriminator			
Layer	**Input**	**Output**	**Layer Information**
Layer1	(256, 256, 3)	(128, 128, 64)	Conv-(N64, K4, S2, P1), LeakyReLU
Layer2	(128, 128, 64)	(64, 64, 128)	Conv-(N128, K4, S2, P1, B), IN, LeakyReLU
Layer3	(64, 64, 128)	(32, 32, 256)	Conv-(N256, K4, S2, P1, B), IN, LeakyReLU
Layer4	(32, 32, 256)	(31, 31, 512)	Conv-(N512, K4, S1, P1, B), IN, LeakyReLU
Layer5	(31, 31, 512)	(30, 30, 1)	Conv-(N1, K4, S1, P1)

Table 2: CycleGAN architecture details. CycleGAN is composed of a generator and a discriminator. CycleGAN generator consists of three downsampling layers, six residual blocks, and three upsampling layers. Unlike DCGAN, you should note that CycleGAN uses the reflection padding and the instance normalization. We use several notations; Conv: the convolution, Deconv: the transposed convolution, N: the number of the output channels, K: the kernel size, S: the stride size, P: the padding size, RP: the reflection padding size, OP: the output padding size for deconvolution, IN: instance normalization. The slope of LeakyReLU in a negative regime is 0.2.

2.2 (20 points) Training

You need to implement all the training processes of CycleGAN. First, you need to complete the class `Summer2WinterDataset` in `dataloader.py`. This dataset class loads the images from domain A (summer) and B (winter). Note that CycleGAN has to train with unpaired images. If you load the paired images, you will get only a half of the score.
Second, using the model implemented in the previous section, you need to complete the class `CycleGan_Solver` in `cyclegan.py`. To train CycleGAN, you should implement three loss functions, such as an adversarial loss, a cycle consistency loss, and an identity mapping loss. The adversarial loss and the cycle consistency losses can be implemented as shown in Figure 2. In addition, the identity mapping loss is to regularize the generator to be close to an identity mapping when real samples of the target domain are provided as the input to the generator: $\mathbf{L}_{identity}(G_{AB}, G_{BA}) = \mathbb{E}_{x \sim p_{data}(B)}[||G_{AB}(x) - x||_1] + \mathbb{E}_{y \sim p_{data}(A)}[||G_{BA}(y) - y||_1]$. x is an image sampled in domain A, and y is an image sampled in domain B.

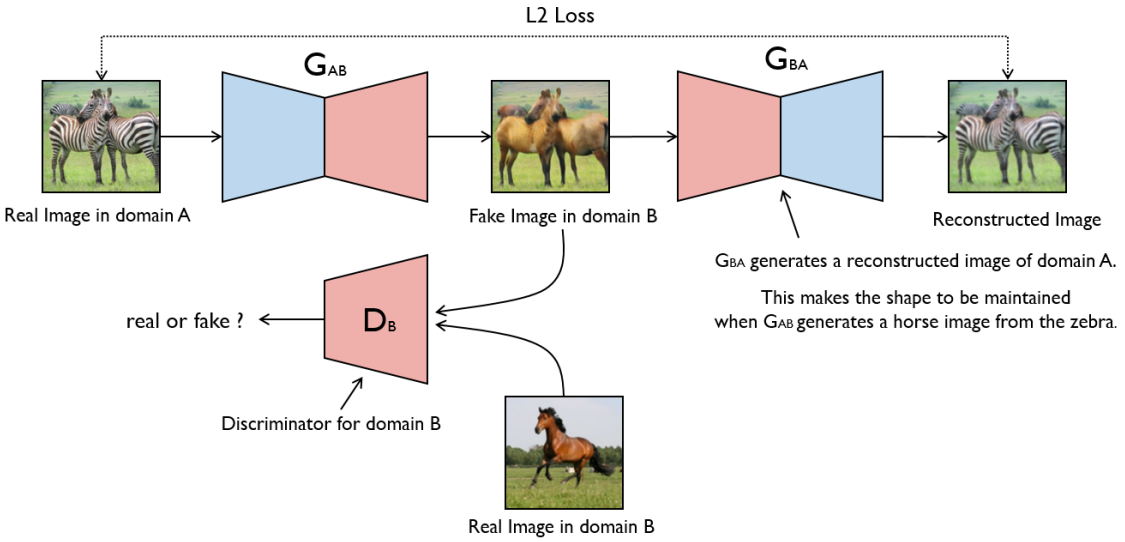

Figure 2: Overview of CycleGAN.

3 Evaluation

GAN models are not easy to evaluate because it generates non-existing photos as its output. Thus, the method of evaluating the distribution of generated images are often used to evaluate GANs, one of which is the Inception Score.

3.1 (10 points) Inception Score

The Inception score evaluates the quality of generated images, specifically synthetic image outputs generated by GAN models. The inception score uses a pre-trained Inception v3 model for image classification. In order to implement the inception score, you use the Inception v3 model to predict the label distribution for each generated image and create the marginal distribution which shows how much variety the generator outputs have. Afterwards, by comparing each image's label distribution with the marginal label distribution for the entire set of images, one can compute the score of how much these two distributions differ. The more they differ, the higher a score. You need to complete the class `Inception_Score` in `inception_score.py`.

3.2 (10 points) Testing

With the Inception score implemented in the previous section, you need to evaluate your DCGAN and CycleGAN models. The test code is already provided, so if you implemented and trained DCGAN and CycleGAN correctly, you can run the evaluation by changing the train option as false in `dcgan.py` and `cyclegan.py`. You should submit your inception scores for DCGAN (GAN, LSGAN, WGAN, WGAN-GP) and CycleGAN implemented in the previous sections.

4 How to submit your assignment

1. Download the attached file. It contains the skeleton codes.
2. Fill in the skeleton codes.
3. Modify the name of the file and zip it into hw3-[your student id].zip, which should contain the following files:
   - **dataloader.py, dcgan.py, cyclegan.py, inception_score.py, util.py**: this file should contain your source code.
   - **results/dcgan/images folder, results/cyclegan/images folder**: this folder should contain your generated images during training DCGAN and CycleGAN.
   - **inception_score.txt**: this file should contain your inception scores. That includes five inception scores for DCGAN (GAN, LSGAN, WGAN, WGAN-GP) and CycleGAN.
4. There is no penalty on the poor results. However, since it checks whether or not you have trained your models, we recommend you to implement your models considering the training time. (Expected training time for DCGAN: less than 4 hours / Expected time for CycleGAN: less than 5 hours)
5. Make sure that no other files are included in the tar.gz file.
6. For KAIST-registered students, submit the tar.gz file at KLMS (http://klms.kaist.ac.kr). For the others, send an e-mail to course.davian@gmail.com with attached tar.gz file.