IMVFX HW2: CONDITIONAL GAN PRACTICE

Link

- Sample Code :
  - Conditional GAN (cGAN)
    https://colab.research.google.com/drive/1w3xU131lJVuNkPraI-mN6cfPZk1bIzQL?usp=sharing
  - Pix2pix
    https://colab.research.google.com/drive/1ncf43rmQ4pXmTsTCa3tiL_T-FMTDkDbh?usp=sharing
- Colab tutorial :
  https://colab.research.google.com/drive/1ecb4-tImhe2CD6DSzxJtt63CAxt2c_Ap
- DEEP LEARNING WITH PYTORCH: A 60 MINUTE BLITZ
  https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

Outline

1. Homework2 Introduction
2. GAN
3. GAN vs. cGAN
3. cGAN vs. pix2pix
4. Steps for training cGAN
5. Homework requirement
6. Homework hint
7. Homework submission

Homework2 Introduction

In this homework, you are going to use cGAN (Conditional Generative Adversarial Network) for image generation and pix2pix for image-to-image translation.

Datasets:

Two datasets:

1. MNIST dataset (train: 60000 imgs)

2. Facades dataset (train: 400 imgs / val: 100 imgs / test: 106 imgs)

* You need to use the first dataset to train cGAN and the second dataset to train pix2pix.

GAN

1. Invented by Ian Goodfellow in 2014. paper
2. Made of two models:

   Generator: Generate the fake data(images) that look like training data(images).

   Discriminator: Look at an image and output the data(images) is true or fake data(images)

3. Try to achieve an equilibrium of the game by training the model.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)} \left[log D(x) \right] + \mathbb{E}_{z \sim p_z(z)} \left[log(1 - D(G(z))) \right]$$

GAN vs. cGAN paper

Condition label
(i.e., 0, 1, …, 9)

True image and label
from training data

Generator

Discriminator

True

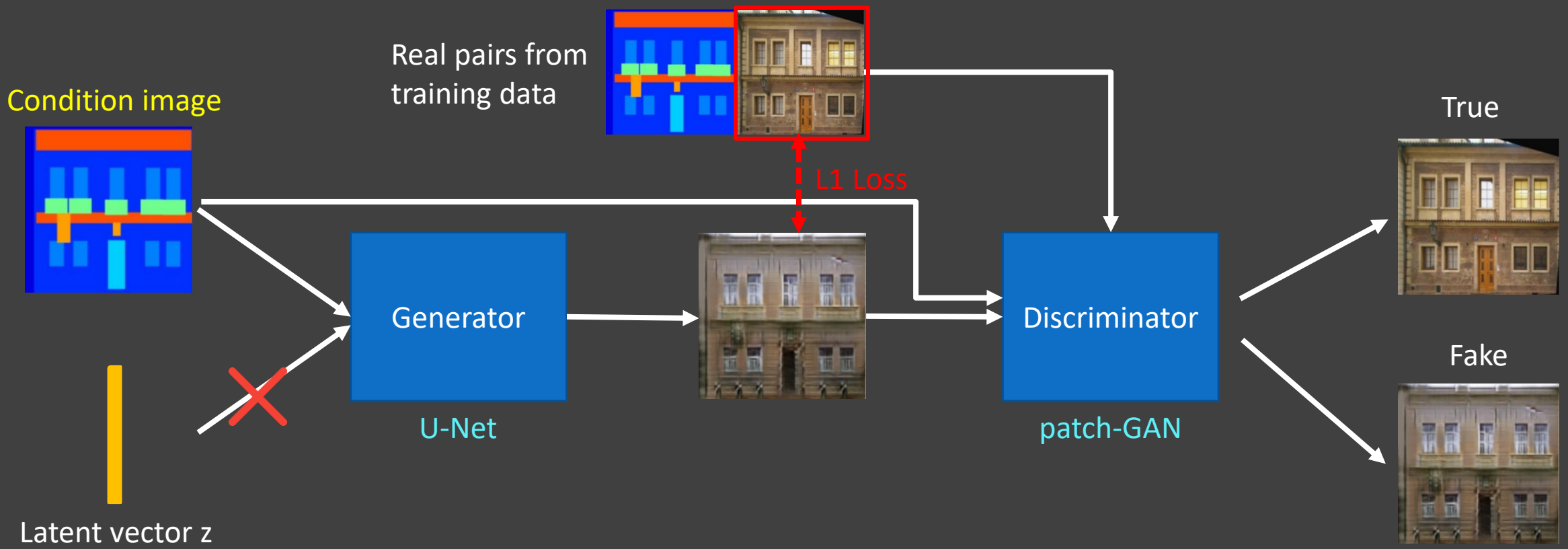
Fake

Compose of:
Linear layers Batch norm layers
LeakyReLU activations

Compose of:
Linear layers
Batch norm layers
LeakyReLU activations

Latent vector z
Sample from Pz(z)

cGAN vs. Pix2pix paper



For implementation 1-1

Steps for training cGAN

1. Set up the parameters
2. Load the data
3. Initialize model weight
4. Build conditional GAN model (Generator, Discriminator)
5. Set up loss functions and optimizers
6. Training

For implementation 1-1

Step 1. Set up the parameters

Set up the parameters for training,

you can change the parameters to improve your training result. (ex. Set the lr = 0.0005)

And also remember to set up cuda device.

```
# Number of workers for dataloader
workers = 2
# Batch size during training
batch_size = 64
# The size of images
image_size = 28
# Number of channels in the training images. For gray images this is 1
nc = 1
# Size of z latent vector
nz = 100
# Number of image classes (i.e. 0-9)
nclass = 10
# Number of training epochs
num_epochs = 5
# Learning rate for optimizers
lr = 0.0002
# Beta1 hyperparam for Adam optimizers
beta1 = 0.5
# The weight for L1 loaa
lamb = 100
# Number of GPUs available. Use 0 for CPU mode.
ngpu = 1
# Save checkpoints every few epochs
save_steps = 5
# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0)
                      else "cpu")
```

For implementation 1-1

Step 2. Load the data

Then we have to load the data from torchvision.datasets. You can use transforms to resize image or normalize the image.

```
import torchvision.datasets as dset
import torchvision.transforms as transforms
# Define training and testing data augmentation
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Create the train dataset
train_dataset = dset.MNIST(dataroot, train=True, download=True, transform=transform)
# Create the train dataloader
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=workers)
```

For implementation 1-1

Step 3. Initialize model weight

We can initialize all models by Normal distribution with mean=0, stdev=0.02.
This function is applied to the models immediately after initialization.

```
# custom weights initialization called on netG and netD
def weights_init(net):
    def init_func(m):
        classname = m.__class__.__name__
        if classname.find('Conv') != -1:
            nn.init.normal_(m.weight.data, 0.0, 0.02)
        elif classname.find('BatchNorm') != -1:
            nn.init.normal_(m.weight.data, 1.0, 0.02)
            nn.init.constant_(m.bias.data, 0)
        elif classname.find('Linear') != -1:
            nn.init.normal_(m.weight.data, 0.0, 0.02)
            nn.init.constant_(m.bias.data, 0)

    net.apply(init_func)
```

For implementation 1-1

Step 4. Build cGAN model - Generator

Define the Generator class:

Two functions:

1. __init__:

Initialize the layers.

(Linear, LeakyReLU, Tanh)

2. forward:

Forward propagation your input through layers.

```python
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        # The size of generator input.
        input_dim = nz + nclass
        # The size of generator output.
        output_dim = image_size*image_size

        # It will have a 10-dimensional encoding for all the 10 digits.
        self.label_embedding = nn.Embedding(nclass, nclass)

        self.main = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),

            nn.Linear(256, 512),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),

            nn.Linear(512, 1024),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),

            nn.Linear(1024, output_dim),
            nn.Tanh()
        )

    def forward(self, x, labels):
        c = self.label_embedding(labels)
        x = torch.cat([x,c], dim=1)
        output = self.main(x)
        return output
```

For implementation 1-1

Step 4. Build cGAN model - Generator

After define the model class, create a generator by the class.

Remember to put your generator to the device and apply the weight initialize function.

```
# Create the generator
netG = Generator().to(device)

# Apply the weights_init function to randomly initialize all weights
#  to mean=0, stdev=0.2.
weights_init(netG)
```

For implementation 1-1

Step 4. Build cGAN model - Discriminator

Define the Discriminator class:

Two functions:

1. __init__:

Initialize the layers.

(Linear, LeakyReLU, Dropout)

2. forward:

Forward propagation your input through layers.

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        # The size of discriminator input.
        input_dim = image_size*image_size + nclass
        # The size of discriminator output.
        output_dim = 1
        # It will have a 10-dimensional encoding for all the 10 digits.
        self.label_embedding = nn.Embedding(nclass, nclass)

        self.main = nn.Sequential(
            nn.Linear(input_dim, 1024),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.Dropout(0.3),

            nn.Linear(1024, 512),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.Dropout(0.3),

            nn.Linear(512, 256),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.Dropout(0.3),

            nn.Linear(256, output_dim),
            nn.Sigmoid()
        )


    def forward(self, x, labels):
        c = self.label_embedding(labels)
        x = torch.cat([x, c], dim=1)
        output = self.main(x)

        return output
```

For implementation 1-1

Step 4. Build cGAN model - Discriminator

Similar with generator, create a discriminator here. Put the network to device and apply weight_init function.

```
# Create the Discriminator
netD = Discriminator().to(device)

# Apply the weights_init function to randomly initialize all weights
#  to mean=0, stdev=0.2.
weights_init(netD)
```

For implementation 1-1

Step 5. Set up loss functions and optimizers

Set up the loss function and optimizer here. Here we use Binary Cross Entropy loss.
Why?
Please write down the reason in reports, discuss them with the BCE loss function, GAN objective and the code in Step6.

```
# Initialize Loss functions
criterionGAN = nn.BCELoss().to(device)
```

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

BCE loss function

```
# Setup Adam optimizers for both G and D
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))
```

You can try different optimizer here, like SGD, Momentum …

For implementation 1-1

Step 6. Training

Then we can start training,

GAN is hard to train, in order to get a decent result, we need to apply some ganhack tricks in our training loop, include:

1. Construct different mini-batches for real and fake images and adjust G's objective function to maximize logD(G(z)).
2. First, train discriminator: maximize log(D(x)) + log(1 - D(G(z)))

   Then train the generator: minimizing log(1−D(G(z))) -> maximize log(D(G(z)))

Create the noise vector(latent vector z), the condition label and the label for true and fake data.

```
# Create the latent vectors and condition labels that we will use to visualize
# the progression of the generator
fixed_noise = torch.randn(100, nz, device=device)
fixed_labels = torch.randint(0, 10, (100,)).to(device)

# Establish convention for real and fake labels during training
real_label = 1.0
fake_label = 0.0
```

For implementation 1-1

Step 6. Training - Train the discriminator

Due to the separate mini-batch suggestion from ganhacks, we will calculate this in two steps.

Step 1. Construct a batch of real samples and real labels from the training set, forward pass through D, calculate the loss (log(D(x))).

```
## Train with all-real batch
netD.zero_grad()
# Format batch
true_img = data[0].view(-1, image_size*image_size).to(device)
b_size = true_img.size(0)
digit_labels = data[1].to(device)
# Forward pass real batch through D
pred_real = netD(true_img, digit_labels).view(-1)
# Calculate loss on all-real batch
label = torch.full((b_size,), real_label, dtype=torch.float, device=device)
errD_real = criterionGAN(pred_real, label)
```

For implementation 1-1

Step 6. Training - Train the discriminator

Step 2. Construct a batch of fake samples with the current generator, forward pass this batch through D, calculate the loss (log(1−D(G(z)))).

```
## Train with all-fake batch
# Generate fake image batch with G
noise = torch.randn(b_size, nz, device=device)
fake_labels = torch.randint(0, 10, (b_size,), device=device)
fake = netG(noise, fake_labels)
# Classify all fake batch with D
pred_fake = netD(fake.detach(), fake_labels).view(-1)
# Calculate D's loss on the all-fake
label = torch.full((b_size,), fake_label, dtype=torch.float, device=device)
errD_fake = criterionGAN(pred_fake, label)
```

For implementation 1-1

Step 6. Training - Train the discriminator

Sum the loss and backward pass to accumulate the gradients. (So, the gradient is computed by forward a batch of real and a batch of fake images)

Remember to call the optimizer step after backward to update the model weight.

```
# Compute error of D as sum over the fake and the real batches
errD = (errD_real + errD_fake)*0.5
# Calculate the gradients for this batch
errD.backward()
# Update D
optimizerD.step()
```

For implementation 1-1

Step 6. Training - Train the Generator

Forward the fake data but calculate loss with real label.

Fake labels are real for generator cost.

```
netG.zero_grad()
fake = netG(noise, fake_labels)
# Since we just updated D, perform another forward pass of all-fake batch through D
pred_fake = netD(fake, fake_labels).view(-1)
label.fill_(real_label)   # fake labels are real for generator cost
# Calculate G's loss based on this output
errG = criterionGAN(pred_fake, label)
# Calculate gradients for G
errG.backward()
# Update G
optimizerG.step()
```

For implementation 1-1

Step 6. Training

Remember to save the model weights when you are training:

You can use torch.save(model, PATH), torch.save(model.state_dict(), PATH) for saving the model weight.

When you want to evaluate data or trained on pretrained weight:

You can use torch.load(PATH), model.load_state_dict(torch.load(PATH)).

Example on pytorch.org

For implementation 1-2, 1-3

Plot the loss and save images

For plot loss:

Save the loss values when you are training, you can save them in a list.

Then use the function in matplotlib.pyplot to plot the loss. Please check pyplot.

For save images:

Save the images by function torchvision.utils.save_image().

```
torchvision.utils.save_image(img_samples, filename, nrow=10)
```

The fake images from your generator.

Save directory

The images number per row.

For implementation 1-4

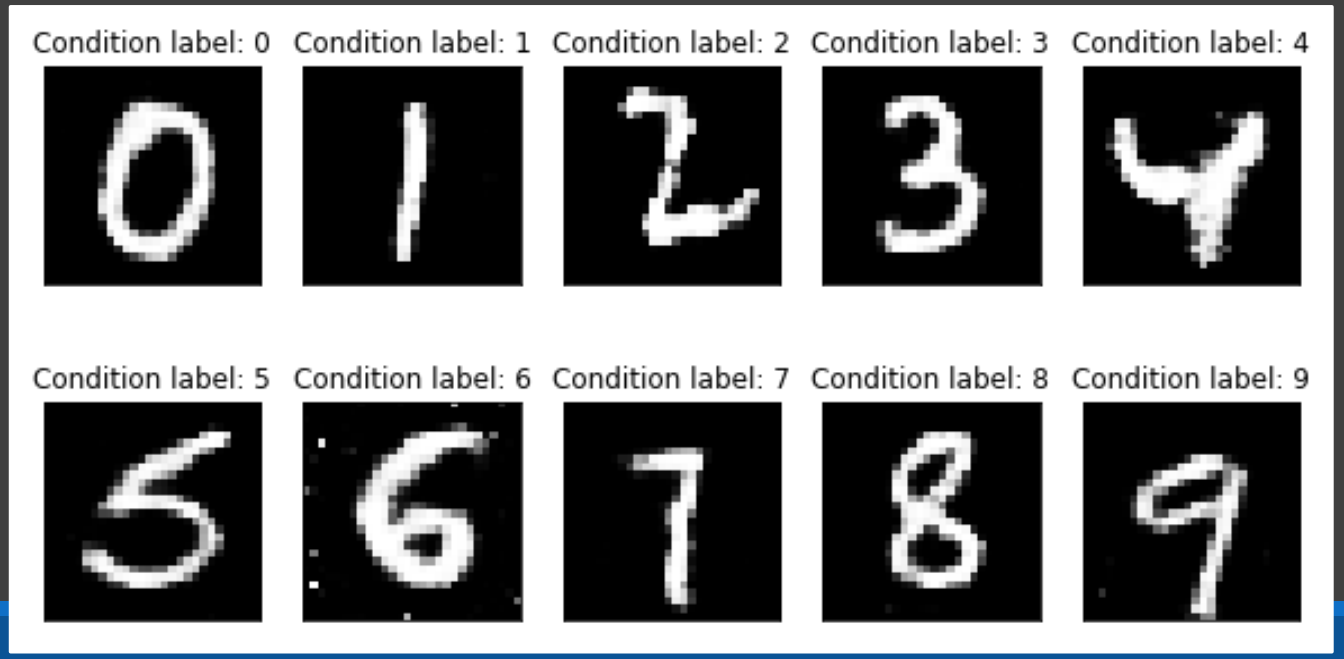
Generate images with specified condition labels

After finish training your model, generate images with specified condition labels instead of using random condition labels.

You can generate the fixed labels as:

```
fixed_labels = torch.tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).to(device)
```

Sample output →



For implementation 1-5

Train a pix2pix model and validate it

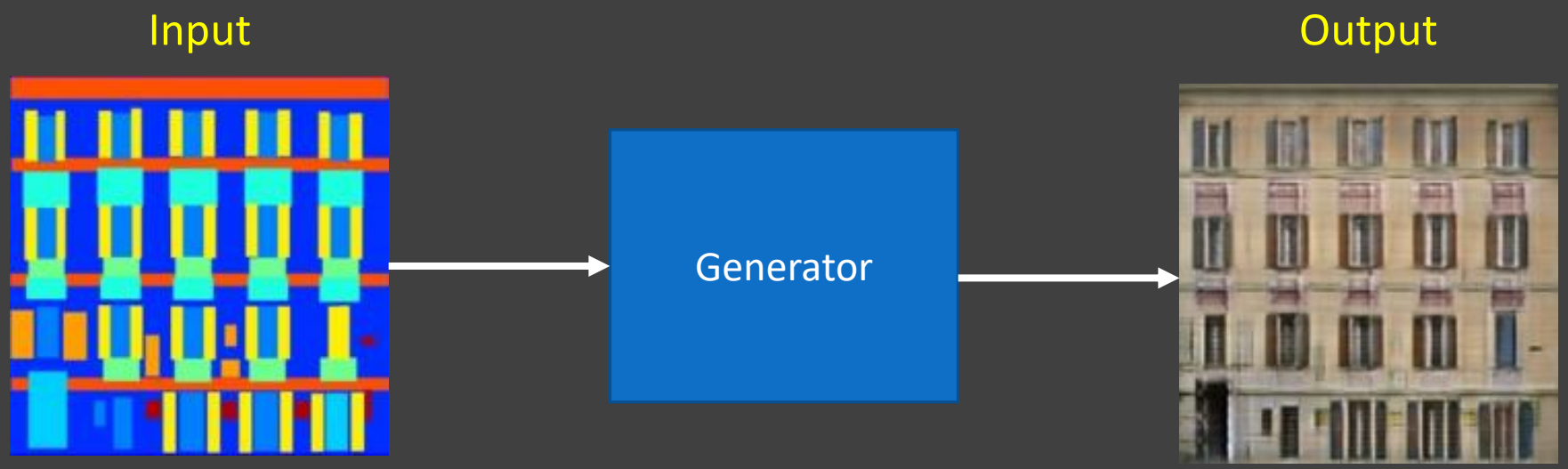
In this part, you need to do:

1. Initialize the Pix2Pix model (build models, set up loss functions and optimizers)
2. Finish the training loop (update Generator network and Discriminator network)
3. Set appropriate hyperparameters (i.e., epochs) to train the pix2pix model
4. Check the evaluation results during training

For implementation 1-7

Use trained model to do image-to-image translation

After finish training your model, use it to do image-to-image translation on the test dataset.



Homework requirement – essential part

1. Implementation(65%):
   - CGAN

     1-1 Train a cGAN model and generate the images. (15%)

     1-2 Plot the loss value of discriminator and generator versus training iteration. (5%)

     1-3 Store the generated 10*10 Grid images in report. (5%)

     1-4 Generate images with specified condition labels. Paste the image in report. (10%)
   - Pix2pix

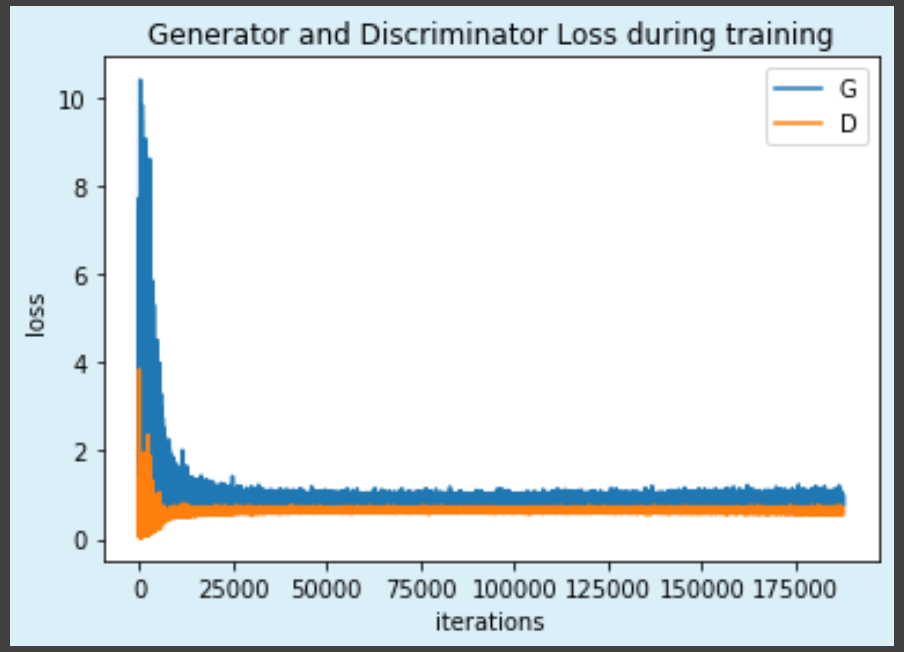
     1-5 Train a pix2pix model and validate it on the validation dataset. (20%)

     1-6 Plot the loss value of discriminator and generator versus training iteration. (5%)

     1-7 Use trained model to do image-to-image translation on the test dataset. (5%)

Homework requirement - Implementation

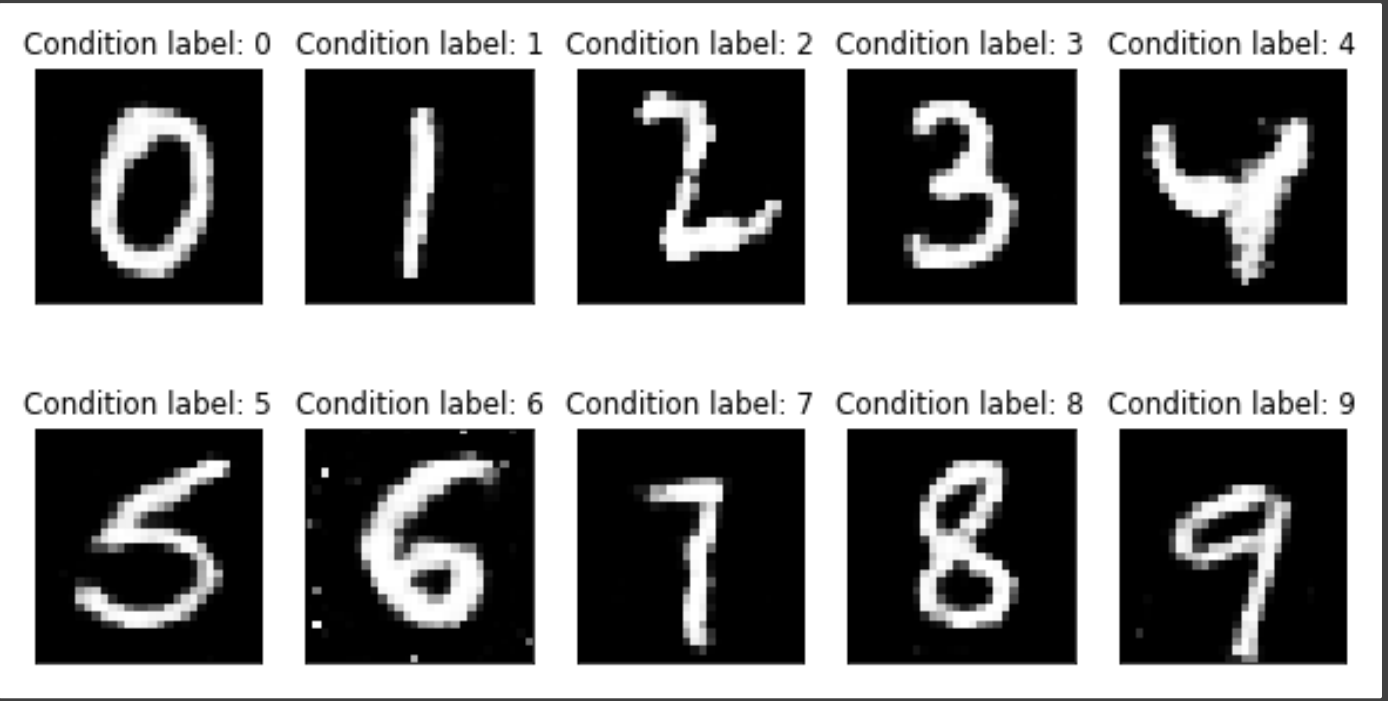
Example for 1-2 、1-6



Example for 1-3

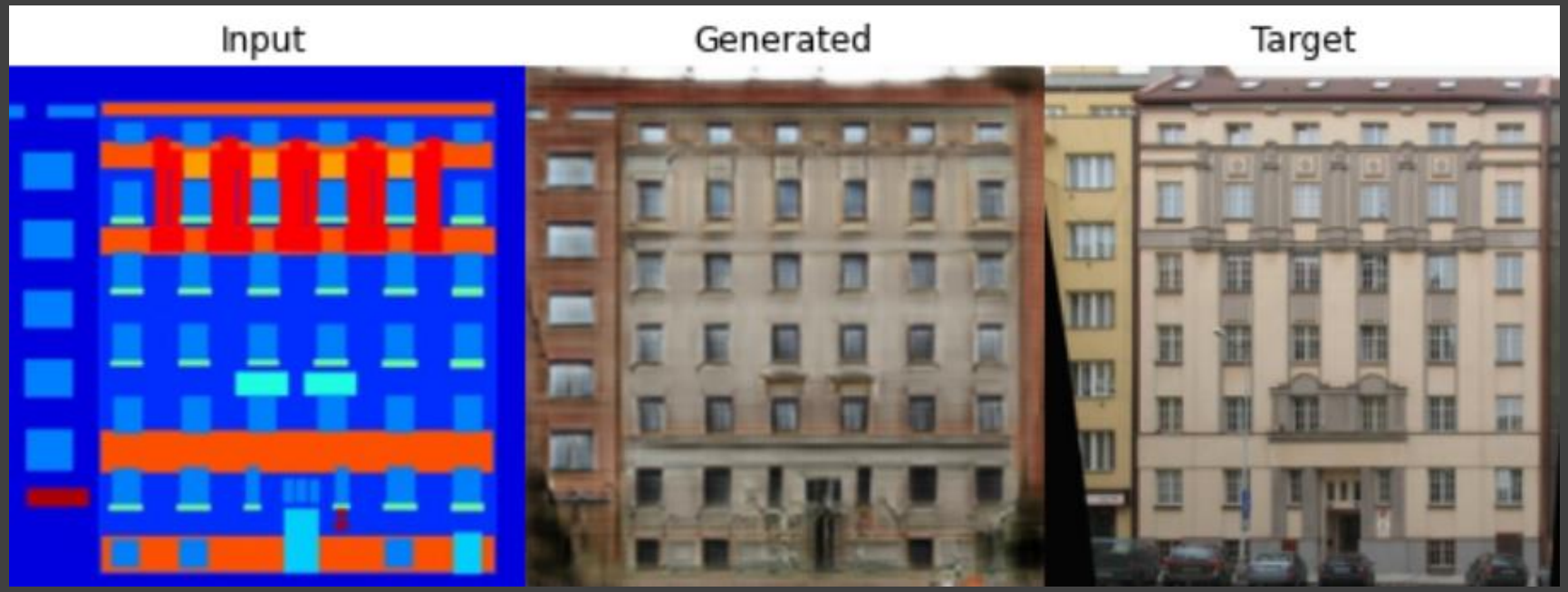
Homework requirement - Implementation

Example for 1-4



Homework requirement - Implementation

Example for 1-7



Homework requirement – essential part

2. Report(25%): You can write in Chinese or English.

   2-1 Introduction (Brief introduction about HW2.) (2%)

   2-2 Network structure: cGAN & Pix2pix (3%)

   2-3 Experiment result (10%)

      2-3.1 Plot of Loss curve: cGAN & Pix2pix (Put the loss picture here.)

      2-3.2 Show the generated 10*10 Grid images in different epochs (Example for 1-3.)

      2-3.3 Show the results of implementation 1-4

      2-3.4 Show five results of implementation 1-7

   2-4 Experiment discussion (10%)

      2-4.1 Discussion about the difference between cGAN and Pix2pix

      2-4.2 Discussion about the reason why we use BCE Loss for training (Please check p.16)

      2-4.3 Discussion about the result of different settings (i.e., epochs, learning rate, some tricks of training GAN, …)

      2-4.4 Others about the experiment you want to discuss

Advanced part – Bonus(10%)

- Try with different type of cGANs (state-of-the-art) and do some further experiment to get a better result.
- Train pix2pix on different dataset (i.e., edges2shoes)

- Try other paper for image-to-image translation (state-of-the-art)

- Need to show your results and write them in report.
- The total score of the essential part is 90 points. For the advanced part (bonus), your score will be judged based on how much you outperform your classmates.

Homework hint

1. Sample codes will be provided in colab, include the structure of cGAN/Pix2pix and training function.

2. GAN Hack, some tricks of training GAN will be provided here.

3. If you use any code from GitHub or other open-source projects, you have to properly cite the source in the report. Otherwise, you will get a penalty of 5 points .

Homework submission

The files you need to submit:

1. Your python(.py) source code (if you use Colab, go to File(left up corner)-> download.ipynb) (studentID.py / studentID.ipynb)

   * You can also hand in the python source code in two files. (studentID_cGAN.py / studentID_pix2pix.py or .ipynb)

2. Your report in .pdf (studentID.pdf)

Zip all the files above to studentID_hw2.zip and upload the zip file to new E3 before the deadline.

* Due on 12/16/2021 23:55:00 PM

* Please follow the submission format carefully or you may lose 5 points for penalty.

Reference

GAN Implementation
https://github.com/soumith/ganhacks
https://pytorch.org/tutorials/beginner/saving_loading_models.html

Data source
http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/

GAN papers
https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (GAN)
https://arxiv.org/pdf/1411.1784.pdf (cGAN)
https://arxiv.org/pdf/1611.07004.pdf (Pix2pix)

State-of-the-art
https://paperswithcode.com/task/conditional-image-generation
https://paperswithcode.com/task/image-to-image-translation