Weekly Work Report

Author Name

VISION@OUC

August 5, 2018

1 Research problem

I am working on the paper named discriminative region proposal adversarial networks for high-quality image-to-image translation and written by Chao Wang *et al.* [6]. Running codes written by myself as the auxiliary experiment are main tasks needing me to devote myself to do.

2 Research approach

I need to understand the main structure of the network and codes written by Chao Wang, becasue the main task is realising some function coded by myself. Thus, the learning and coding of Python [4] and Pytorch [1] is necessary.

3 Research progress

I have learned the first, second and fourth part at deep learning course [3] to get the basic and principle understanding of neural network. Then, reading and understanding the article [6] becomes the first task. Last week, I have tried to write easy codes to implement networks like autoencoder. However, when it comes to GAN and DCGAN, the most thing I did is watching codes cloned at github and imitating to write only the part of defining networks.

4 Progress in this week

In this week, the main task is running and understanding codes about pix2pix. Before watching the pix2pix codes, I read the article which introduces pix2pix [5], and run codes cloned from github [2] at the server.

4.1 Read the article

Because this pix2pix is close to the network designed by Chao Wang, and the major task is evaluating the network compared to pix2pix. Thus, I read this article about pix2pix carefully.

The innovation point of pix2pix is that authors used a novel autoencoder network with skip connecitons named "U-Net" (see Figure 1) at generator. Let Ck denote a Convolution-BatchNorm-ReLU layer with k filters. CDk denotes a Convolution-BatchNorm-Dropout-ReLU layer with a dropout rate of 50%. All ocnvolutions are 4×4 spatial filters applied with stride 2. Convolutions in the encoder, and in the discriminator, downsample by a factor of 2, whereas in the decoder they upsample by factor of 2.

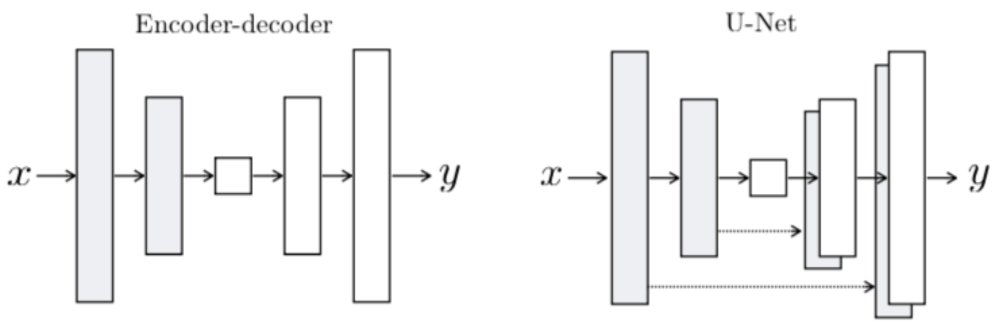


Figure 1: Two choices for the architecture of the generator. The "U-Net" is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks.

Generator architectures: The encoder-decoder architecture consists of:
encoder: $C64 - C128 - C256 - C512 - C512 - C512 - C512 - C512$

```
---------------- Options ---------------
               batchSize: 1
                   beta1: 0.5
         checkpoints_dir: ./checkpoints
          continue_train: False
                dataroot: ./datasets/facades                  [default: None]
            dataset_mode: aligned
             display_env: main
            display_freq: 400
              display_id: 1
           display_ncols: 4
            display_port: 8097
          display_server: http://localhost
         display_winsize: 256
             epoch_count: 1
                fineSize: 256
                 gpu_ids: 0
               init_gain: 0.02
               init_type: normal
                input_nc: 3
                 isTrain: True                                [default: None]
               lambda_L1: 100.0
                loadSize: 286
                      lr: 0.0002
          lr_decay_iters: 50
               lr_policy: lambda
        max_dataset_size: inf
                   model: pix2pix                             [default: cycle_gan]
                nThreads: 4
              n_layers_D: 3
                    name: facades_pix2pix                     [default: experiment_name]
                     ndf: 64
                     ngf: 64
                   niter: 100
             niter_decay: 100
              no_dropout: False
                 no_flip: False
                 no_html: False
                no_lsgan: True
                    norm: batch
               output_nc: 3
                   phase: train
               pool_size: 0
              print_freq: 100
          resize_or_crop: resize_and_crop
         save_epoch_freq: 5
```

Figure 2: The start of running pix2pix.

U-Net decoder: $CD512-CD1024-CD1024-CD1024-CD1024-C512-C256-C128$
The reason why channels of U-Net decoder is the double of channels of encoder is that the channels not only comes from the last layer, but also comes from the mirror layers because of skip connections. After the last layer in the decoder, a convolution is applied to map to the number of output channels, followed by a Tanh function. As an exceptin to the above notation, Batch-Norm is not applied to the first $C64$ layer in the encoder. All ReLU in the encoder are leaky, with slope 0.2, while ReLUs in the decoder are not leaky.

Discriminaotr architectures: Authors applied 70×70 PatchGAN to alleviate articfacts and achieve slightly better similar scores.
70×70 **discriminator architectue:** $C64-C128-C256-C512$.
After the last layer, a convolution is appplied to map to a 1 dimensional output, followed by a Sigmoid function. BatchNorm is not applied to the first $C64$ layer. All ReLUs are leaky, with slope 0.2.

The other innovation point is adding the L1 loss to the cAGAN. The discriminator's job remains unchanged, but the generaotr is tasked to not only fool the discriminator but also to be near the ground truth output in an L2 sense. Authors also explore this option, using L1 distance rather than L2 as L1 encourages less blurring. Taking all factors into consideration, the final objective is [5]:

$$G^* = arg\min_G\max_D \mathcal{L}_{cGAN}(G,D)+\lambda\mathcal{L}_{L1}(G). \tag{1}$$

4.2 Run codes

Before running codes downloaded from github [2], I learned some grammar about Python [4] to make me understand codes clearly. I imitated codes written at the website to make me have a profound impression as reading the course. Then, installing some libraries at the terminal to match codes by using [2]:

$$pip\ \ install\ \ visdom\ \ dominate \tag{2}$$

After cloning codes to the server which ip is 222.195.147.33 and downloading the dataset, I run the train.py with GPU. I need to run the train.py code at one termimal just as shown at Figure 2, and open

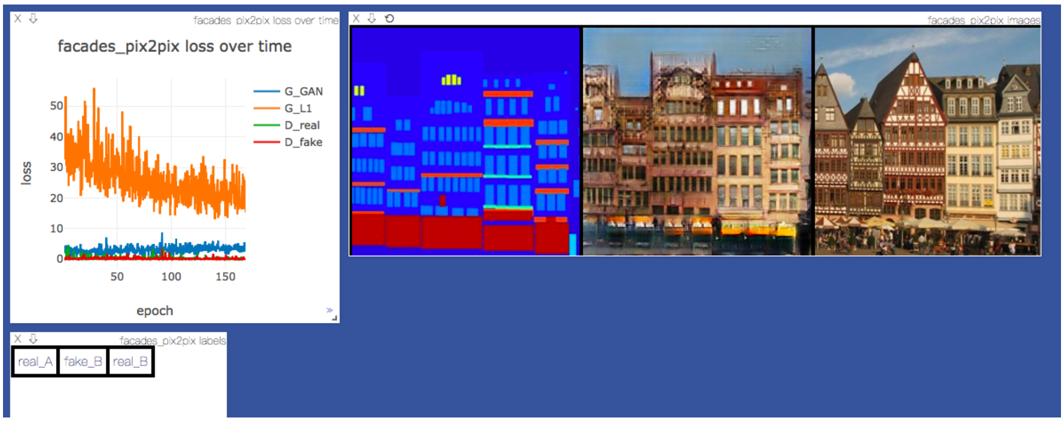


Figure 3: The result of pix2pix.

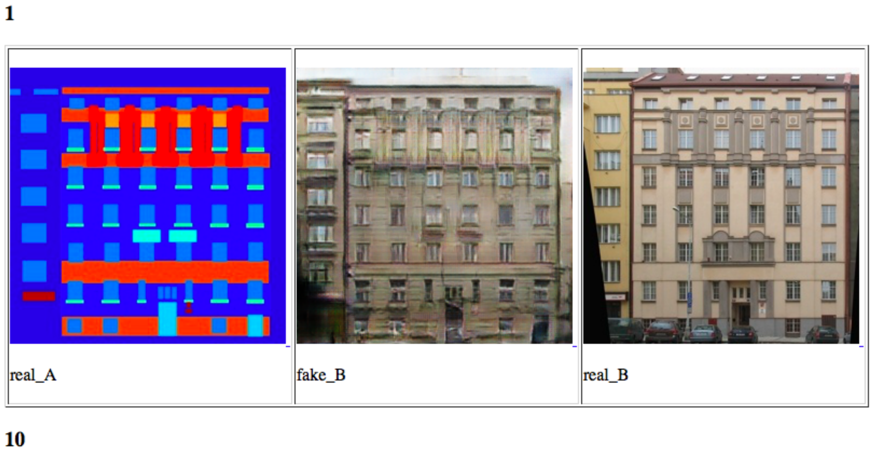


Figure 4: The test result of pix2pix.

another terminal to view training results and loss plots by running $python \quad -m \quad visdom.server$, and click the URL http://localhost:8097 (see Figure 3).

When the training has done, I run the test.py at the server and all results are saved to a html file. Downloading the folder named results from the server to the local, I open this html file to see images generated by pix2pix (see Figure 4)

4.3 Read and understand codes

The whole codes contain many folders like options, models and data. The main codes I read are train.py (Figure 5), network.py (Figure 6) and pix2pix_model.py (Figure 7) with some interpretions searched from baidu or Python files.

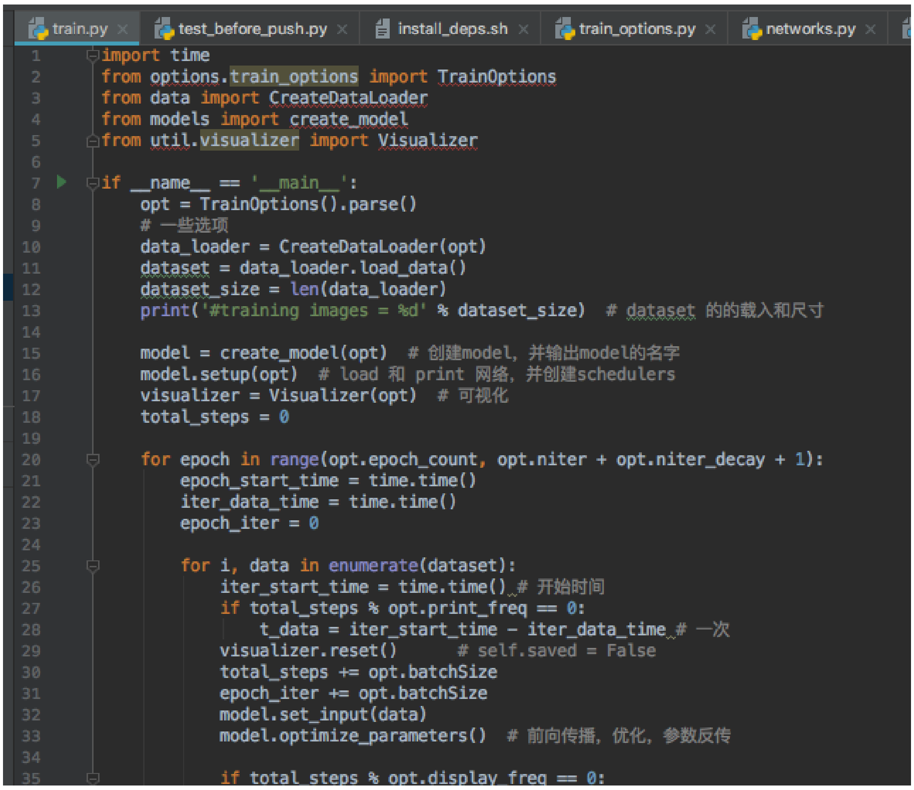
References

[1] bilibili. https://www.bilibili.com/video/av15997678/?p=1. 1

[2] GitHub. https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix. 1, 2

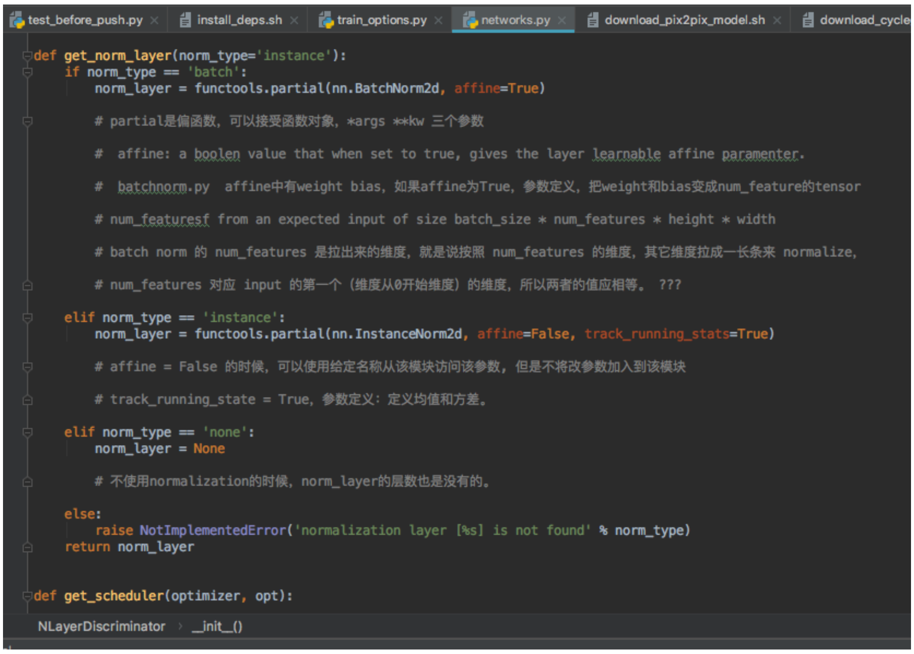
[3] Neural Networks & Deep Learning. https://mooc.study.163.com/course/2001281002#/info. 1

[4] Xuefeng Liao. https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000. 1, 2



```python
import time
from options.train_options import TrainOptions
from data import CreateDataLoader
from models import create_model
from util.visualizer import Visualizer

if __name__ == '__main__':
    opt = TrainOptions().parse()
    # 一些选项
    data_loader = CreateDataLoader(opt)
    dataset = data_loader.load_data()
    dataset_size = len(data_loader)
    print('#training images = %d' % dataset_size)  # dataset 的的载入和尺寸

    model = create_model(opt)  # 创建model，并输出model的名字
    model.setup(opt)  # load 和 print 网络，并创建schedulers
    visualizer = Visualizer(opt)  # 可视化
    total_steps = 0

    for epoch in range(opt.epoch_count, opt.niter + opt.niter_decay + 1):
        epoch_start_time = time.time()
        iter_data_time = time.time()
        epoch_iter = 0

        for i, data in enumerate(dataset):
            iter_start_time = time.time() # 开始时间
            if total_steps % opt.print_freq == 0:
                t_data = iter_start_time - iter_data_time # 一次
            visualizer.reset()      # self.saved = False
            total_steps += opt.batchSize
            epoch_iter += opt.batchSize
            model.set_input(data)
            model.optimize_parameters()  # 前向传播，优化，参数反传

            if total_steps % opt.display_freq == 0:
```

Figure 5: Train.



```python
def get_norm_layer(norm_type='instance'):
    if norm_type == 'batch':
        norm_layer = functools.partial(nn.BatchNorm2d, affine=True)

        # partial是偏函数，可以接受函数对象，*args **kw 三个参数

        #  affine: a boolen value that when set to true, gives the layer learnable affine paramenter.

        #  batchnorm.py  affine中有weight bias，如果affine为True，参数定义，把weight和bias变成num_feature的tensor

        # num_featuresf from an expected input of size batch_size * num_features * height * width

        # batch norm 的 num_features 是拉出来的维度，就是说按照 num_features 的维度，其它维度拉成一长条来 normalize,

        # num_features 对应 input 的第一个（维度从0开始维度）的维度，所以两者的值应相等。 ???

    elif norm_type == 'instance':
        norm_layer = functools.partial(nn.InstanceNorm2d, affine=False, track_running_stats=True)

        # affine = False 的时候，可以使用给定名称从该模块访问该参数，但是不将改参数加入到该模块

        # track_running_state = True，参数定义：定义均值和方差。

    elif norm_type == 'none':
        norm_layer = None

        # 不使用normalization的时候，norm_layer的层数也是没有的。

    else:
        raise NotImplementedError('normalization layer [%s] is not found' % norm_type)
    return norm_layer


def get_scheduler(optimizer, opt):
```

Figure 6: Network.

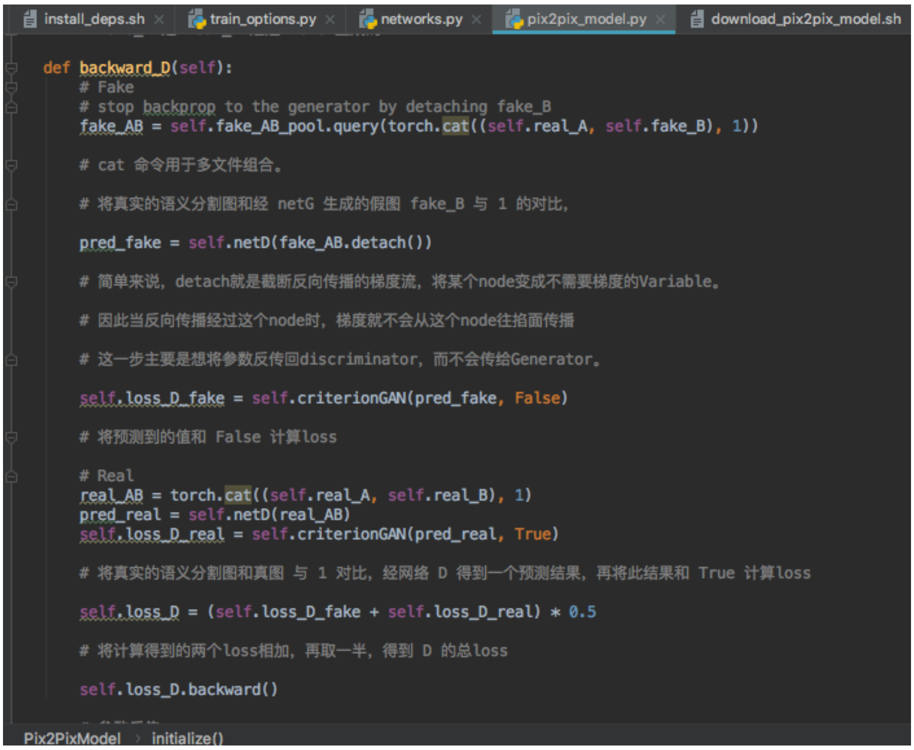


Figure 7: Pix2pix_model.

[5] P. Isola, J. Y. Zhu, T. Zhou, and A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 1, 2

[6] C. Wang, H. Zheng, Z. Yu, Z. Zheng, Z. Gu, and B. Zheng. Discriminative region proposal adversarial networks for high-quality image-to-image translation. In *ECCV*, 2018. 1