Weekly Work Report

Author Name

VISION@OUC

August 12, 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.* [5]. 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 read papers and codes about GAN, DCGAN and Stack *et al.* to add experiments for codes written by Chao Wang to get more functions and explaination.

3 Research progress

Before this week, I have written codes about autoencoder with both all linear layers and convolution layers. When it comes to GAN and DCGAN, I just can read codes written by others at github and write the defination code about networks. Then, I have read the pix2pix codes cloned from github [3] and DRPAN codes [1] at last week, and have a basic understanding to it.

4 Progress in this week

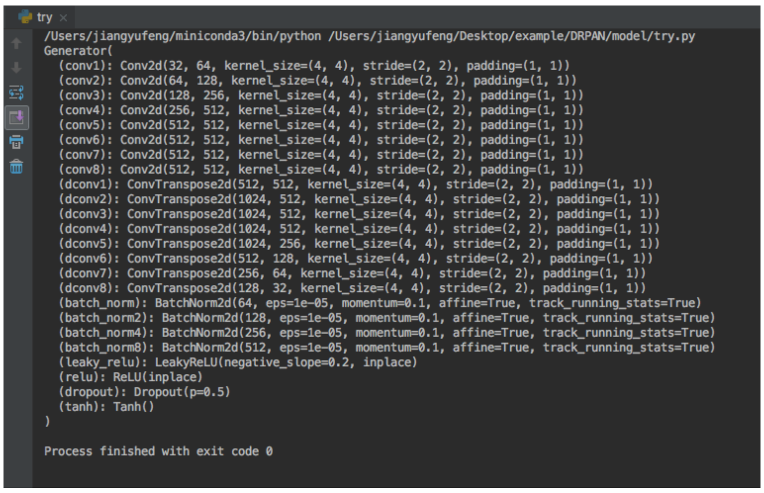
In this week, the main task is running and reading DRPAN codes [1] and trying to write code for implementing a generator using dilation convolution.

4.1 DRPAN

The first thing I have done in this week is running DRPAN codes on the server via a sequence:

$$python\ main.py\ --config\ configs/facades.yaml\ --cuda\ --gpu\_ids\ 0 \tag{1}$$

where the gpu_id needs me to modify after checking the use information of gpu using a code: $nvidia-smi$. When training StackGan-like model, there are some things need to modify. One is verifying the class name called DatasetFromFolder into Aligned_Dataset, because there is not a DatasetFromFolder class in data/dataset.py. Two is modifying the outf and outf_test path to the folder creaeted by me at the server. The last one is adding the parser.add_opption about "–cuda". The final results on DRPAN training and StackGan-like training is stored at folders named checkpoints and checkpointss separately.

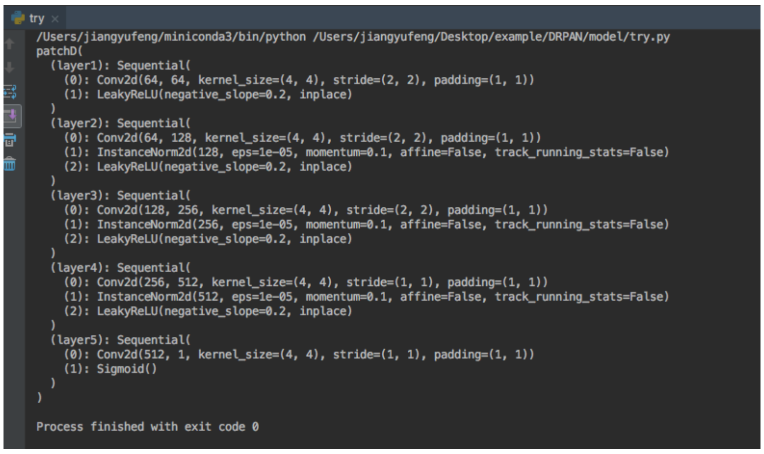


```
/Users/jiangyufeng/miniconda3/bin/python /Users/jiangyufeng/Desktop/example/DRPAN/model/try.py
Generator(
  (conv1): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv4): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv5): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv6): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv7): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv8): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (dconv1): ConvTranspose2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (dconv2): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (dconv3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (dconv4): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (dconv5): ConvTranspose2d(1024, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (dconv6): ConvTranspose2d(512, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (dconv7): ConvTranspose2d(256, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (dconv8): ConvTranspose2d(128, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (batch_norm): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batch_norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batch_norm4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batch_norm8): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (leaky_relu): LeakyReLU(negative_slope=0.2, inplace)
  (relu): ReLU(inplace)
  (dropout): Dropout(p=0.5)
  (tanh): Tanh()
)

Process finished with exit code 0
```

Figure 1: The generator structure of U-Net.

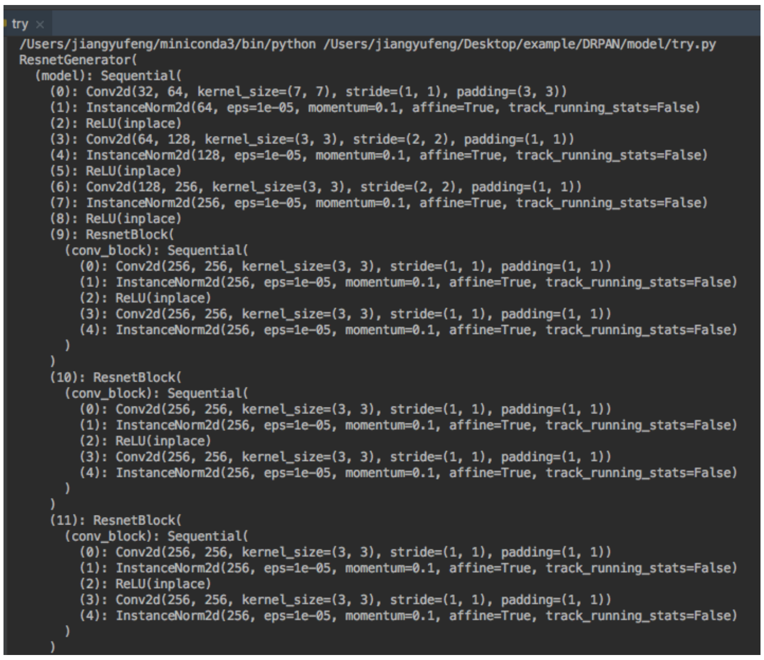
Reading codes is started from main.py, and searching some functions meaning and useage to understand the code better after knowing the effect of every module. The second step is reading the trainer.py to get a deeper understanding about the procedure on training. The third main part is network.



```
/Users/jiangyufeng/miniconda3/bin/python /Users/jiangyufeng/Desktop/example/DRPAN/model/try.py
patchD(
  (layer1): Sequential(
    (0): Conv2d(64, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1))
    (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (2): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer5): Sequential(
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1))
    (1): Sigmoid()
  )
)

Process finished with exit code 0
```

Figure 2: The patch discriminator structure.



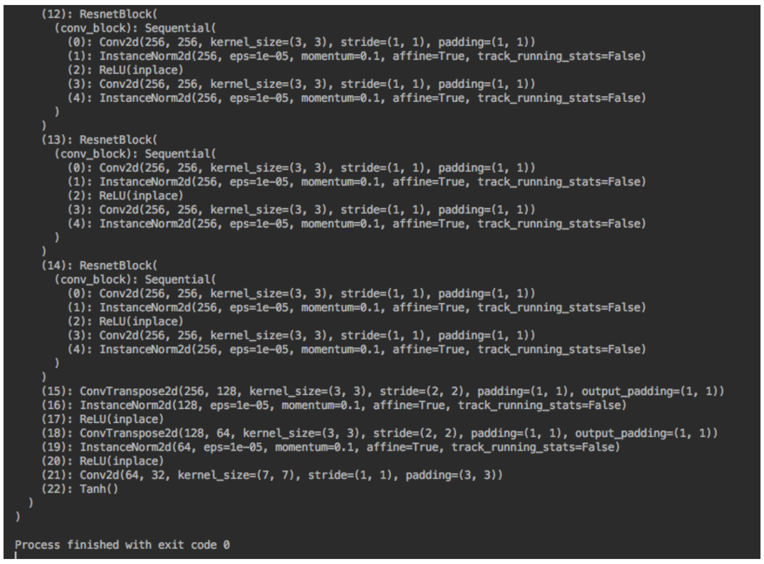
```
/Users/jiangyufeng/miniconda3/bin/python /Users/jiangyufeng/Desktop/example/DRPAN/model/try.py
ResnetGenerator(
  (model): Sequential(
    (0): Conv2d(32, 64, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (1): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (2): ReLU(inplace)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (5): ReLU(inplace)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (8): ReLU(inplace)
    (9): ResnetBlock(
      (conv_block): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (2): ReLU(inplace)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
    (10): ResnetBlock(
      (conv_block): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (2): ReLU(inplace)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
    (11): ResnetBlock(
      (conv_block): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (2): ReLU(inplace)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
```

Figure 3: The first part of Resnet generator structure.

In this python file, there are some networks for generator, discriminator and reviser. The first option about generator is Resnet, and the whole sturcture is shown at Figure 3 and Figure 4.

The front layers use convolution and Instance batch norm, and the activation function is ReLU. Correspondingly, the last few layers use deconvolution and Instance batch norm, the activation is also using ReLU except the last layer which does not use batch norm and uses tanh function instead of ReLU. The midden layers use 6 resnet blocks. The second option for generator is U-Net similar to pix2pix [4] where there are an autoencoder network with skip connections (see Figre 1).

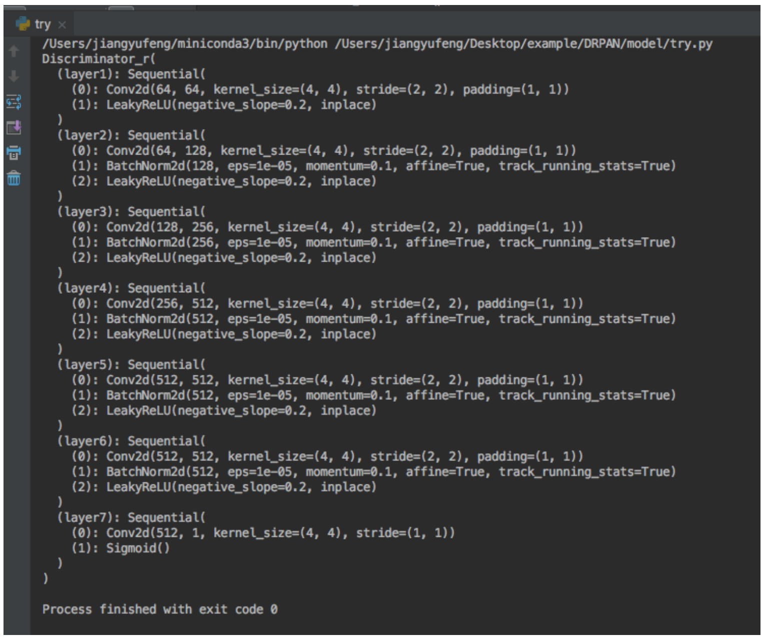
The patch discriminator with 5 convolution layers is shown at Figure 2. In this network, the first layer does not use instance batch norm and the last layer only use convolution and sigmoid function. The activation function changes to LeakyReLU function.



```
    (12): ResnetBlock(
      (conv_block): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (2): ReLU(inplace)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
    (13): ResnetBlock(
      (conv_block): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (2): ReLU(inplace)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
    (14): ResnetBlock(
      (conv_block): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (2): ReLU(inplace)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
    (15): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (16): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (17): ReLU(inplace)
    (18): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (19): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (20): ReLU(inplace)
    (21): Conv2d(64, 32, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (22): Tanh()
  )
)

Process finished with exit code 0
```

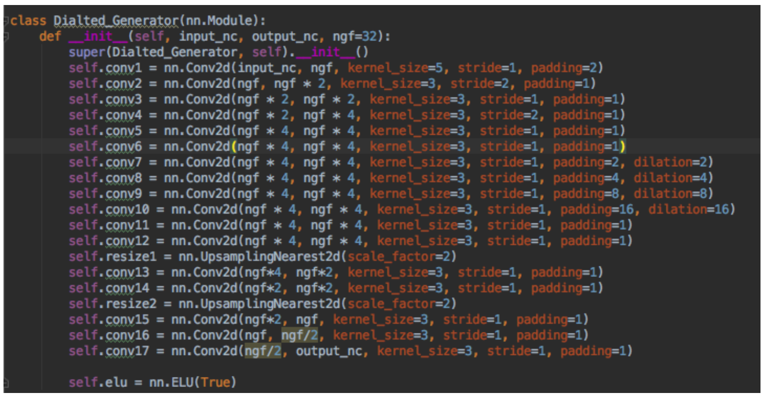
Figure 4: The second part of Resnet generator structure.



```
/Users/jiangyufeng/miniconda3/bin/python /Users/jiangyufeng/Desktop/example/DRPAN/model/try.py
Discriminator_r(
  (layer1): Sequential(
    (0): Conv2d(64, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer5): Sequential(
    (0): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer6): Sequential(
    (0): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace)
  )
  (layer7): Sequential(
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
    (1): Sigmoid()
  )
)

Process finished with exit code 0
```

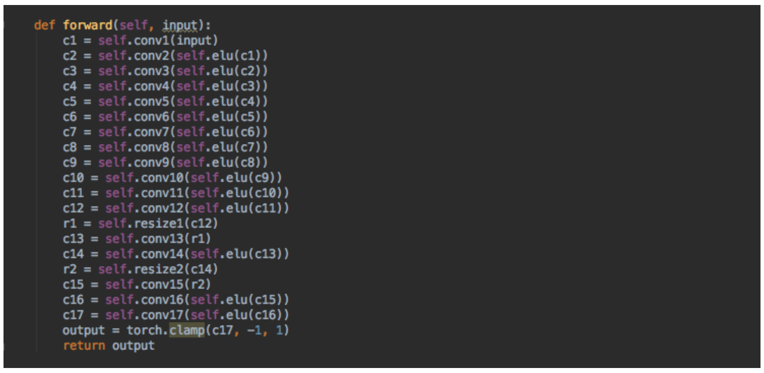
Figure 5: The reviser structure.



```
class Dialted_Generator(nn.Module):
    def __init__(self, input_nc, output_nc, ngf=32):
        super(Dialted_Generator, self).__init__()
        self.conv1 = nn.Conv2d(input_nc, ngf, kernel_size=5, stride=1, padding=2)
        self.conv2 = nn.Conv2d(ngf, ngf * 2, kernel_size=3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(ngf * 2, ngf * 2, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(ngf * 2, ngf * 4, kernel_size=3, stride=2, padding=1)
        self.conv5 = nn.Conv2d(ngf * 4, ngf * 4, kernel_size=3, stride=1, padding=1)
        self.conv6 = nn.Conv2d(ngf * 4, ngf * 4, kernel_size=3, stride=1, padding=1)
        self.conv7 = nn.Conv2d(ngf * 4, ngf * 4, kernel_size=3, stride=1, padding=2, dilation=2)
        self.conv8 = nn.Conv2d(ngf * 4, ngf * 4, kernel_size=3, stride=1, padding=4, dilation=4)
        self.conv9 = nn.Conv2d(ngf * 4, ngf * 4, kernel_size=3, stride=1, padding=8, dilation=8)
        self.conv10 = nn.Conv2d(ngf * 4, ngf * 4, kernel_size=3, stride=1, padding=16, dilation=16)
        self.conv11 = nn.Conv2d(ngf * 4, ngf * 4, kernel_size=3, stride=1, padding=1)
        self.conv12 = nn.Conv2d(ngf * 4, ngf * 4, kernel_size=3, stride=1, padding=1)
        self.resize1 = nn.UpsamplingNearest2d(scale_factor=2)
        self.conv13 = nn.Conv2d(ngf*4, ngf*2, kernel_size=3, stride=1, padding=1)
        self.conv14 = nn.Conv2d(ngf*2, ngf*2, kernel_size=3, stride=1, padding=1)
        self.resize2 = nn.UpsamplingNearest2d(scale_factor=2)
        self.conv15 = nn.Conv2d(ngf*2, ngf, kernel_size=3, stride=1, padding=1)
        self.conv16 = nn.Conv2d(ngf, ngf/2, kernel_size=3, stride=1, padding=1)
        self.conv17 = nn.Conv2d(ngf/2, output_nc, kernel_size=3, stride=1, padding=1)

        self.elu = nn.ELU(True)
```

Figure 6: The layers of dilation convolution network.



```
    def forward(self, input):
        c1 = self.conv1(input)
        c2 = self.conv2(self.elu(c1))
        c3 = self.conv3(self.elu(c2))
        c4 = self.conv4(self.elu(c3))
        c5 = self.conv5(self.elu(c4))
        c6 = self.conv6(self.elu(c5))
        c7 = self.conv7(self.elu(c6))
        c8 = self.conv8(self.elu(c7))
        c9 = self.conv9(self.elu(c8))
        c10 = self.conv10(self.elu(c9))
        c11 = self.conv11(self.elu(c10))
        c12 = self.conv12(self.elu(c11))
        r1 = self.resize1(c12)
        c13 = self.conv13(r1)
        c14 = self.conv14(self.elu(c13))
        r2 = self.resize2(c14)
        c15 = self.conv15(r2)
        c16 = self.conv16(self.elu(c15))
        c17 = self.conv17(self.elu(c16))
        output = torch.clamp(c17, -1, 1)
        return output
```

Figure 7: The forward propagation of dilation convolution network.

Similarly, the reviser (see Figure 5) also has a discriminator purpose with 7 convolution layers where the activation function is LeakyReLU with slope 0.2. The first layer does not use batch norm and the last layer only use sigmoid function.

4.2 Dilation convolution

After reading DRPAN codes, the next thing needed to do is adding a generator network about dilation convolution. The network of paper [6] uses dilation convolutions at the midden layers. Their code is published at github [2]. However, they use tensorflow instead of pytorch. Thus, the main thing I have done is reading their code and understanding the stage1 network. Then, writting my own code with pytorch using their network.

In terms of layer inplementations, they use mirror padding for all convolution layers and remove batch normalization layers which they found deteriorates color coherence. Also, they use ELUs as activation functions instead of ReLU, and clip the output filter values instead of using *tanh* or *sigmoid* functions. Based on their network thought, I write this network (see Figure 6 and Figure 7).

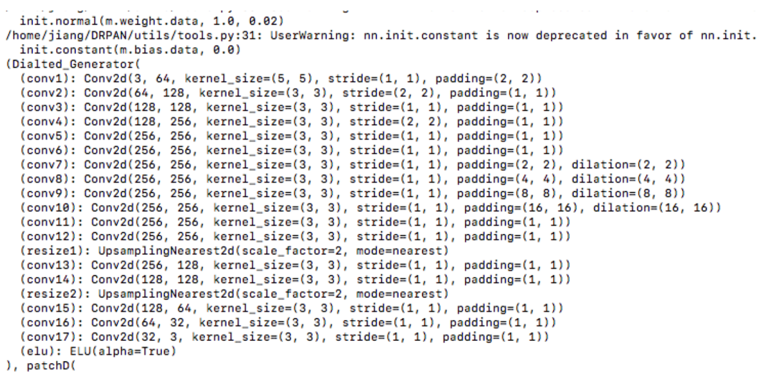
After modifying configs and facades.yaml, the network can run at the server. The whole generator network printed at the server is shown at Figure 8. The code is running, and the results can be seen at tomorrow morning.

References

[1] GitHub_godisboy. https://github.com/godisboy/DRPAN. 1

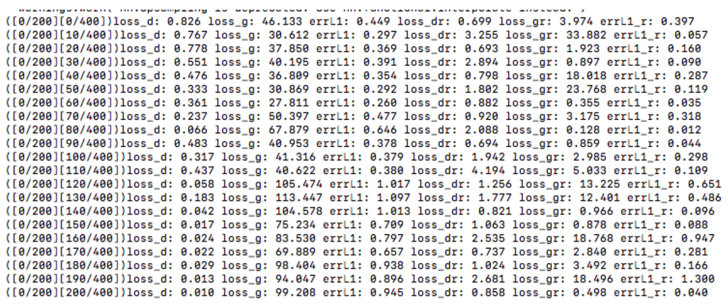
[2] GitHub_jiahuiyu. 4

[3] GitHub_junyanz. https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix. 1



```
  init.normal(m.weight.data, 1.0, 0.02)
/home/jiang/DRPAN/utils/tools.py:31: UserWarning: nn.init.constant is now deprecated in favor of nn.init.
  init.constant(m.bias.data, 0.0)
(Dialted_Generator(
  (conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv6): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv7): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2))
  (conv8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4), dilation=(4, 4))
  (conv9): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(8, 8), dilation=(8, 8))
  (conv10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(16, 16), dilation=(16, 16))
  (conv11): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (resize1): UpsamplingNearest2d(scale_factor=2, mode=nearest)
  (conv13): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv14): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (resize2): UpsamplingNearest2d(scale_factor=2, mode=nearest)
  (conv15): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv16): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv17): Conv2d(32, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (elu): ELU(alpha=True)
), patchD(
```

Figure 8: The layers of the whole dilation convolution network.



```
([0/200][0/400])loss_d: 0.826 loss_g: 46.133 errL1: 0.449 loss_dr: 0.699 loss_gr: 3.974 errL1_r: 0.397
([0/200][10/400])loss_d: 0.767 loss_g: 30.612 errL1: 0.297 loss_dr: 3.255 loss_gr: 33.882 errL1_r: 0.057
([0/200][20/400])loss_d: 0.778 loss_g: 37.850 errL1: 0.369 loss_dr: 0.693 loss_gr: 1.923 errL1_r: 0.160
([0/200][30/400])loss_d: 0.551 loss_g: 40.195 errL1: 0.391 loss_dr: 2.894 loss_gr: 0.897 errL1_r: 0.090
([0/200][40/400])loss_d: 0.476 loss_g: 36.809 errL1: 0.354 loss_dr: 0.798 loss_gr: 18.018 errL1_r: 0.287
([0/200][50/400])loss_d: 0.333 loss_g: 30.869 errL1: 0.292 loss_dr: 1.802 loss_gr: 23.768 errL1_r: 0.119
([0/200][60/400])loss_d: 0.361 loss_g: 27.811 errL1: 0.260 loss_dr: 0.882 loss_gr: 0.355 errL1_r: 0.035
([0/200][70/400])loss_d: 0.237 loss_g: 50.397 errL1: 0.477 loss_dr: 0.920 loss_gr: 3.175 errL1_r: 0.318
([0/200][80/400])loss_d: 0.066 loss_g: 67.879 errL1: 0.646 loss_dr: 2.088 loss_gr: 0.128 errL1_r: 0.012
([0/200][90/400])loss_d: 0.483 loss_g: 40.953 errL1: 0.378 loss_dr: 0.694 loss_gr: 0.859 errL1_r: 0.044
([0/200][100/400])loss_d: 0.317 loss_g: 41.316 errL1: 0.379 loss_dr: 1.942 loss_gr: 2.985 errL1_r: 0.298
([0/200][110/400])loss_d: 0.437 loss_g: 40.622 errL1: 0.380 loss_dr: 4.194 loss_gr: 5.033 errL1_r: 0.109
([0/200][120/400])loss_d: 0.058 loss_g: 105.474 errL1: 1.017 loss_dr: 1.256 loss_gr: 13.225 errL1_r: 0.651
([0/200][130/400])loss_d: 0.183 loss_g: 113.447 errL1: 1.097 loss_dr: 1.777 loss_gr: 12.401 errL1_r: 0.486
([0/200][140/400])loss_d: 0.042 loss_g: 104.578 errL1: 1.013 loss_dr: 0.821 loss_gr: 0.966 errL1_r: 0.096
([0/200][150/400])loss_d: 0.017 loss_g: 75.234 errL1: 0.709 loss_dr: 1.063 loss_gr: 0.878 errL1_r: 0.088
([0/200][160/400])loss_d: 0.024 loss_g: 83.530 errL1: 0.797 loss_dr: 2.535 loss_gr: 18.768 errL1_r: 0.947
([0/200][170/400])loss_d: 0.022 loss_g: 69.889 errL1: 0.657 loss_dr: 0.737 loss_gr: 2.840 errL1_r: 0.281
([0/200][180/400])loss_d: 0.029 loss_g: 98.404 errL1: 0.938 loss_dr: 1.024 loss_gr: 3.492 errL1_r: 0.166
([0/200][190/400])loss_d: 0.013 loss_g: 94.047 errL1: 0.896 loss_dr: 2.681 loss_gr: 18.496 errL1_r: 1.300
([0/200][200/400])loss_d: 0.010 loss_g: 99.208 errL1: 0.945 loss_dr: 0.858 loss_gr: 0.498 errL1_r: 0.040
```

Figure 9: The training procedure of DRPAN adding dilation convolutions.

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

[5] 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

[6] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. Huang. Generative image inpainting with contextual attention. *arXiv preprint arXiv:1801.07892*, 2018. 4