```python
class Discriminator(nn.Module):
  ##TODO1 implement the discriminator (critic)
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=128, kernel_size=5, stride=2, padding=2)
        self.conv2 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=5, stride=2, padding=2)
        self.conv3 = nn.Conv2d(in_channels=256, out_channels=512, kernel_size=5, stride=2, padding=2)

        self.fc = nn.Linear(in_features=512*4*4, out_features=1)

    def forward(self, x):
        x = F.leaky_relu(self.conv1(x), 0.1)
        x = F.leaky_relu(self.conv2(x), 0.1)
        x = F.leaky_relu(self.conv3(x), 0.1)

        x = x.view(-1, 512*4*4)
        x = self.fc(x)

        return x
```

```python
class Generator(nn.Module):
    ##TODO2 implement the generator (actor)
    def __init__(self):
        super().__init__()

        self.fc = nn.Linear(128, 512*4*4)

        self.conv_block1 = nn.Sequential(
            nn.ConvTranspose2d(512, 256, kernel_size=5, stride=2, padding=2, output_padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
        )
        self.conv_block2 = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=5, stride=2, padding=2, output_padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
        )
        self.conv_block3 = nn.Sequential(
            nn.ConvTranspose2d(128, 1, kernel_size=5, stride=2, padding=2, output_padding=1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        x = self.fc(x)
        x = x.view(-1, 512, 4, 4)

        x = self.conv_block1(x)
        x = self.conv_block2(x)
        x = self.conv_block3(x)

        x = F.interpolate(x, size=(28, 28), mode='bilinear', align_corners=False)

        return x
```

TODO 3: What is the input and output shape of the generator and discriminator network? Verify that the implemented networks are the same as the answer you have provided.

```python
from torchinfo import summary
# Test the discriminator
print(summary(discriminator, input_size=((1, 1, 28, 28))))

# Test the generator
print(summary(generator, input_size=((1, 128))))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Discriminator                            [1, 1]                    --
├─Conv2d: 1-1                            [1, 128, 14, 14]          3,328
├─Conv2d: 1-2                            [1, 256, 7, 7]            819,456
├─Conv2d: 1-3                            [1, 512, 4, 4]            3,277,312
├─Linear: 1-4                            [1, 1]                    8,193
==========================================================================================
Total params: 4,108,289
Trainable params: 4,108,289
Non-trainable params: 0
Total mult-adds (M): 93.25
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.37
Params size (MB): 16.43
Estimated Total Size (MB): 16.80
==========================================================================================
```

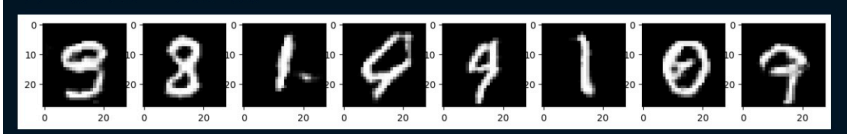
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Generator                                [1, 1, 28, 28]            --
├─Linear: 1-1                            [1, 8192]                 1,056,768
├─Sequential: 1-2                        [1, 256, 8, 8]            --
│    └─ConvTranspose2d: 2-1              [1, 256, 8, 8]            3,277,056
│    └─BatchNorm2d: 2-2                  [1, 256, 8, 8]            512
│    └─ReLU: 2-3                         [1, 256, 8, 8]            --
├─Sequential: 1-3                        [1, 128, 16, 16]          --
│    └─ConvTranspose2d: 2-4              [1, 128, 16, 16]          819,328
│    └─BatchNorm2d: 2-5                  [1, 128, 16, 16]          256
│    └─ReLU: 2-6                         [1, 128, 16, 16]          --
├─Sequential: 1-4                        [1, 1, 32, 32]            --
│    └─ConvTranspose2d: 2-7              [1, 1, 32, 32]            3,201
│    └─Sigmoid: 2-8                      [1, 1, 32, 32]            --
==========================================================================================
Total params: 5,157,121
Trainable params: 5,157,121
Non-trainable params: 0
Total mult-adds (M): 423.81
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.86
Params size (MB): 20.63
Estimated Total Size (MB): 21.49
==========================================================================================
```

```
NUM_ITERATION = 3000
BATCH_SIZE = 32
fixed_z = torch.randn((8, 128)).cuda()
def schedule(i):
  lr = 1e-4
  if(i > 2500): lr *= 0.1
  return lr
losses = {'D' : [None], 'G' : [None]}

## TODO4 initialize missing hyperparameter and optimzer
G_optimizer = torch.optim.Adam(generator.parameters(), lr=1e-4, betas=(0.0, 0.9))
D_optimizer = torch.optim.Adam(discriminator.parameters(), lr=1e-4, betas=(0.0, 0.9))
GP_lambda = 10
n_critic = 5
```

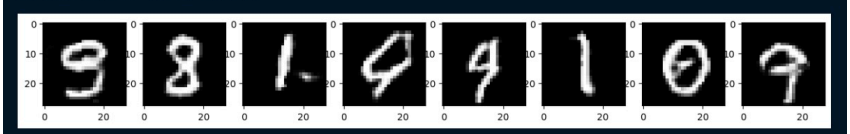
```
# TODO5 implement dataloader
import torch
from torch.utils.data import DataLoader, Dataset

class CustomDataset(Dataset):
    def __init__(self, data, z_dim):
        self.data = data
        self.z_dim = z_dim

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        x = torch.FloatTensor(self.data[idx])
        z = torch.randn(self.z_dim)
        epsilon = torch.rand(1)

        return x, z, epsilon

Z_DIM = 128
BATCH_SIZE = 32

custom_dataset = CustomDataset(trainX, Z_DIM)
dataloader = DataLoader(custom_dataset, batch_size=BATCH_SIZE, shuffle=True)
```

```python
from tqdm import tqdm
for i in tqdm(range(NUM_ITERATION)):
  ## TODO6 update learning rate
  for params in D_optimizer.param_groups:
    params['lr'] = schedule(i)
  for params in G_optimizer.param_groups:
    params['lr'] = schedule(i)

  discriminator.train()
  for t in range(n_critic):
    ## TODO7 line 4: sample data from dataloader
    real_images, z, epsilon = next(iter(dataloader))

    real_images = real_images.cuda()
    z = z.cuda()
    epsilon = epsilon.cuda().view(-1, 1, 1, 1)

    ## TODO8 line5-7 : calculate discriminator loss
    D_optimizer.zero_grad()

    fake_images = generator(z)

    alpha = epsilon.view(-1, 1, 1, 1)
    interpolated_images = alpha * real_images + (1 - alpha) * fake_images.detach()
    interpolated_images.requires_grad_(True)

    real_predict = discriminator(real_images)
    fake_predict = discriminator(fake_images.detach())
    interpolated_predict = discriminator(interpolated_images)

    gradients = torch.autograd.grad(outputs=interpolated_predict, inputs=interpolated_images,
                                    grad_outputs=torch.ones_like(interpolated_predict),
                                    create_graph=True, retain_graph=True)[0]
    gradients_penalty = ((gradients.norm(2, dim=1) - 1) ** 2).mean() * GP_lambda

    d_loss = fake_predict.mean() - real_predict.mean() + gradients_penalty
    losses['D'].append(d_loss.item())

    ## TODO9 : line 9 update discriminator loss
    d_loss.backward()
    D_optimizer.step()
  ## TODO10 : line 11-12 calculate and update the generator loss
  generator.train()
  G_optimizer.zero_grad()

  fake_images = generator(z)
  fake_predict = discriminator(fake_images)
  g_loss = -fake_predict.mean()
  losses['G'].append(g_loss.item())

  g_loss.backward()
  G_optimizer.step()

  # Output visualization : If your reimplementation is correct, the generated images should start resembling a digit character after 500 iterations.
  discriminator.eval()
  generator.eval()
  if(i % 100 == 0):
    plt.figure(figsize = (15,75))
    print(losses['D'][-1], losses['G'][-1])
    with torch.no_grad():
      res = generator(fixed_z).cpu().detach().numpy()
    for k in range(8):
      plt.subplot( int('18{}'.format(k+1)) )
      plt.imshow( res[k].transpose(1, 2, 0)[..., 0], cmap = 'gray' )
    plt.show()
```

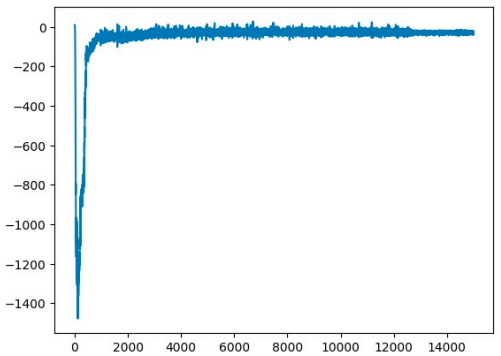
```
 93%|▉| 2799/3000 [02:42<00:09, 21.08it/s]
-28.047470092773438 -421.8146057128906
```

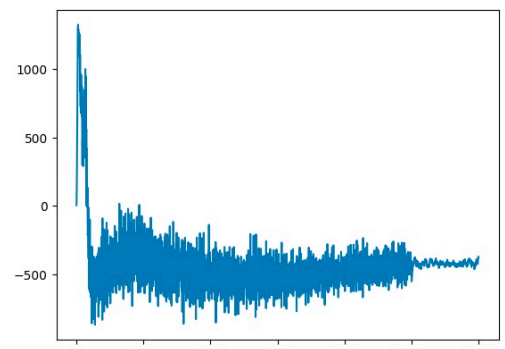


```
 97%|▉| 2900/3000 [02:47<00:04, 20.50it/s]
-28.122135162353516 -412.7256774902344
```



```
100%|▉| 3000/3000 [02:53<00:00, 17.27it/s]
```





```python
class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=4, stride=2, padding=1, with_norm=True):
        super().__init__()
        self.conv = nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=kernel_size, stride=stride, padding=padding)
        self.instancenorm = nn.InstanceNorm2d(out_channels)
        self.activation = nn.LeakyReLU(0.2)
        self.with_norm = with_norm

    def forward(self, x):
        x = self.conv(x)
        if self.with_norm:
            x = self.instancenorm(x)
        x = self.activation(x)

        return x

class Discriminator(nn.Module):
  #TODO11 implement the discriminator network
    def __init__(self):
        super().__init__()
        self.block64 = ConvBlock(in_channels=6, out_channels=64, with_norm=False)
        self.block128 = ConvBlock(in_channels=64, out_channels=128)
        self.block256 = ConvBlock(in_channels=128, out_channels=256)
        self.block512 = ConvBlock(in_channels=256, out_channels=512, stride=1)

        self.conv = nn.Conv2d(in_channels=512, out_channels=1, kernel_size=4, stride=1, padding=1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.block64(x)
        x = self.block128(x)
        x = self.block256(x)
        x = self.block512(x)
        x = self.conv(x)
        x = self.sigmoid(x)
        return x

discriminator = Discriminator().cuda()
```

```python
#TODO12 verify the discriminator
from torchinfo import summary
summary(discriminator, input_size=((1, 6, 256, 256)))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Discriminator                            [1, 1, 30, 30]            --
├─ConvBlock: 1-1                         [1, 64, 128, 128]         --
│    └─Conv2d: 2-1                       [1, 64, 128, 128]         6,208
│    └─LeakyReLU: 2-2                    [1, 64, 128, 128]         --
├─ConvBlock: 1-2                         [1, 128, 64, 64]          --
│    └─Conv2d: 2-3                       [1, 128, 64, 64]          131,200
│    └─InstanceNorm2d: 2-4               [1, 128, 64, 64]          --
│    └─LeakyReLU: 2-5                    [1, 128, 64, 64]          --
├─ConvBlock: 1-3                         [1, 256, 32, 32]          --
│    └─Conv2d: 2-6                       [1, 256, 32, 32]          524,544
│    └─InstanceNorm2d: 2-7               [1, 256, 32, 32]          --
│    └─LeakyReLU: 2-8                    [1, 256, 32, 32]          --
├─ConvBlock: 1-4                         [1, 512, 31, 31]          --
│    └─Conv2d: 2-9                       [1, 512, 31, 31]          2,097,664
│    └─InstanceNorm2d: 2-10              [1, 512, 31, 31]          --
│    └─LeakyReLU: 2-11                   [1, 512, 31, 31]          --
├─Conv2d: 1-5                            [1, 1, 30, 30]            8,193
├─Sigmoid: 1-6                           [1, 1, 30, 30]            --
==========================================================================================
Total params: 2,767,809
Trainable params: 2,767,809
Non-trainable params: 0
Total mult-adds (G): 3.20
==========================================================================================
Input size (MB): 1.57
Forward/backward pass size (MB): 18.62
Params size (MB): 11.07
Estimated Total Size (MB): 31.27
==========================================================================================
```

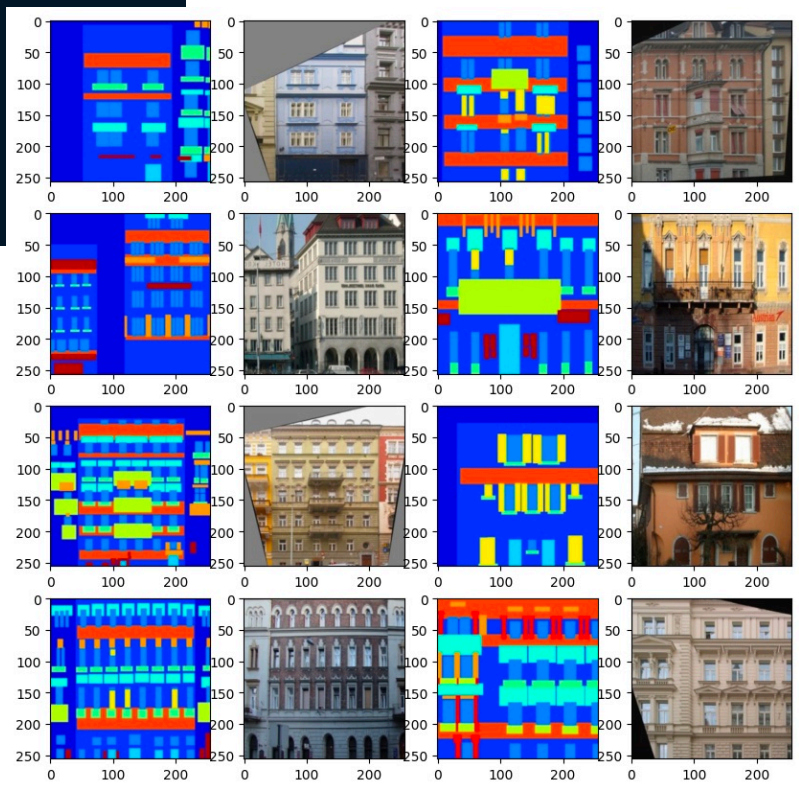
```python
class ConvTBlock(nn.Module):
  def __init__(self, in_channels, out_channels, kernel_size=4, stride=2, padding=1, p=0.5, with_norm=True, with_relu=True):
    super(ConvTBlock, self).__init__()
    self.conv = nn.ConvTranspose2d(in_channels=in_channels, out_channels=out_channels, kernel_size=kernel_size, stride=stride, padding=padding)
    self.instancenorm = nn.InstanceNorm2d(out_channels)
    self.dropout = nn.Dropout(p=p)
    self.relu = nn.ReLU()
    self.with_relu = with_relu
    self.with_norm = with_norm

  def forward(self, x, x_enc = None):
    if x_enc is not None:
      x = torch.cat((x, x_enc), 1)
    x = self.conv(x)
    if self.with_norm:
      x = self.instancenorm(x)
    x = self.dropout(x)
    if self.with_relu:
      x = self.relu(x)
    return x

class Generator(nn.Module):
    #TODO13 implement the generator network
    def __init__(self):
      super().__init__()

      # encoder
      self.encoder1 = ConvBlock(in_channels=3, out_channels=64, with_norm=False)
      self.encoder2 = ConvBlock(in_channels=64, out_channels=128)
      self.encoder3 = ConvBlock(in_channels=128, out_channels=256)
      self.encoder4 = ConvBlock(in_channels=256, out_channels=512)
      self.encoder5 = ConvBlock(in_channels=512, out_channels=512)
      self.encoder6 = ConvBlock(in_channels=512, out_channels=512)
      self.encoder7 = ConvBlock(in_channels=512, out_channels=512)
      self.encoder8 = ConvBlock(in_channels=512, out_channels=512, with_norm=False)


      # decoder
      self.decoder1 = ConvTBlock(in_channels=512, out_channels=512)
      self.decoder2 = ConvTBlock(in_channels=1024, out_channels=512)
      self.decoder3 = ConvTBlock(in_channels=1024, out_channels=512)
      self.decoder4 = ConvTBlock(in_channels=1024, out_channels=512)
      self.decoder5 = ConvTBlock(in_channels=1024, out_channels=256)
      self.decoder6 = ConvTBlock(in_channels=512, out_channels=128)
      self.decoder7 = ConvTBlock(in_channels=256, out_channels=64)
      self.decoder8 = ConvTBlock(in_channels=128, out_channels=3, with_norm=False, with_relu=False)

      self.tanh = nn.Tanh()

    def forward(self, x):
      e1 = self.encoder1(x)
      e2 = self.encoder2(e1)
      e3 = self.encoder3(e2)
      e4 = self.encoder4(e3)
      e5 = self.encoder5(e4)
      e6 = self.encoder6(e5)
      e7 = self.encoder7(e6)
      e8 = self.encoder8(e7)

      d1 = self.decoder1(e8)
      d2 = self.decoder2(d1, e7)
      d3 = self.decoder3(d2, e6)
      d4 = self.decoder4(d3, e5)
      d5 = self.decoder5(d4, e4)
      d6 = self.decoder6(d5, e3)
      d7 = self.decoder7(d6, e2)
      d8 = self.decoder8(d7, e1)

      o = self.tanh(d8)
      return o

generator = Generator().cuda()
```

```python
#TODO14 verify the generator
from torchinfo import summary
summary(generator, input_size=((1, 3, 256, 256)))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Generator                                [1, 3, 256, 256]          --
├─ConvBlock: 1-1                         [1, 64, 128, 128]         --
│    └─Conv2d: 2-1                       [1, 64, 128, 128]         3,136
│    └─LeakyReLU: 2-2                    [1, 64, 128, 128]         --
├─ConvBlock: 1-2                         [1, 128, 64, 64]          --
│    └─Conv2d: 2-3                       [1, 128, 64, 64]          131,200
│    └─InstanceNorm2d: 2-4               [1, 128, 64, 64]          --
│    └─LeakyReLU: 2-5                    [1, 128, 64, 64]          --
├─ConvBlock: 1-3                         [1, 256, 32, 32]          --
│    └─Conv2d: 2-6                       [1, 256, 32, 32]          524,544
│    └─InstanceNorm2d: 2-7               [1, 256, 32, 32]          --
│    └─LeakyReLU: 2-8                    [1, 256, 32, 32]          --
├─ConvBlock: 1-4                         [1, 512, 16, 16]          --
│    └─Conv2d: 2-9                       [1, 512, 16, 16]          2,097,664
│    └─InstanceNorm2d: 2-10              [1, 512, 16, 16]          --
│    └─LeakyReLU: 2-11                   [1, 512, 16, 16]          --
├─ConvBlock: 1-5                         [1, 512, 8, 8]            --
│    └─Conv2d: 2-12                      [1, 512, 8, 8]            4,194,816
│    └─InstanceNorm2d: 2-13              [1, 512, 8, 8]            --
│    └─LeakyReLU: 2-14                   [1, 512, 8, 8]            --
├─ConvBlock: 1-6                         [1, 512, 4, 4]            --
│    └─Conv2d: 2-15                      [1, 512, 4, 4]            4,194,816
│    └─InstanceNorm2d: 2-16              [1, 512, 4, 4]            --
│    └─LeakyReLU: 2-17                   [1, 512, 4, 4]            --
├─ConvBlock: 1-7                         [1, 512, 2, 2]            --
│    └─Conv2d: 2-18                      [1, 512, 2, 2]            4,194,816
│    └─InstanceNorm2d: 2-19              [1, 512, 2, 2]            --
│    └─LeakyReLU: 2-20                   [1, 512, 2, 2]            --
├─ConvBlock: 1-8                         [1, 512, 1, 1]            --
│    └─Conv2d: 2-21                      [1, 512, 1, 1]            4,194,816
│    └─LeakyReLU: 2-22                   [1, 512, 1, 1]            --
├─ConvTBlock: 1-9                        [1, 512, 2, 2]            --
│    └─ConvTranspose2d: 2-23             [1, 512, 2, 2]            4,194,816
│    └─InstanceNorm2d: 2-24              [1, 512, 2, 2]            --
│    └─Dropout: 2-25                     [1, 512, 2, 2]            --
│    └─ReLU: 2-26                        [1, 512, 2, 2]            --
├─ConvTBlock: 1-10                       [1, 512, 4, 4]            --
│    └─ConvTranspose2d: 2-27             [1, 512, 4, 4]            8,389,120
│    └─InstanceNorm2d: 2-28              [1, 512, 4, 4]            --
│    └─Dropout: 2-29                     [1, 512, 4, 4]            --
│    └─ReLU: 2-30                        [1, 512, 4, 4]            --
├─ConvTBlock: 1-11                       [1, 512, 8, 8]            --
│    └─ConvTranspose2d: 2-31             [1, 512, 8, 8]            8,389,120
│    └─InstanceNorm2d: 2-32              [1, 512, 8, 8]            --
│    └─Dropout: 2-33                     [1, 512, 8, 8]            --
│    └─ReLU: 2-34                        [1, 512, 8, 8]            --
├─ConvTBlock: 1-12                       [1, 512, 16, 16]          --
│    └─ConvTranspose2d: 2-35             [1, 512, 16, 16]          8,389,120
│    └─InstanceNorm2d: 2-36              [1, 512, 16, 16]          --
│    └─Dropout: 2-37                     [1, 512, 16, 16]          --
│    └─ReLU: 2-38                        [1, 512, 16, 16]          --
├─ConvTBlock: 1-13                       [1, 256, 32, 32]          --
│    └─ConvTranspose2d: 2-39             [1, 256, 32, 32]          4,194,560
│    └─InstanceNorm2d: 2-40              [1, 256, 32, 32]          --
│    └─Dropout: 2-41                     [1, 256, 32, 32]          --
│    └─ReLU: 2-42                        [1, 256, 32, 32]          --
├─ConvTBlock: 1-14                       [1, 128, 64, 64]          --
│    └─ConvTranspose2d: 2-43             [1, 128, 64, 64]          1,048,704
│    └─InstanceNorm2d: 2-44              [1, 128, 64, 64]          --
│    └─Dropout: 2-45                     [1, 128, 64, 64]          --
│    └─ReLU: 2-46                        [1, 128, 64, 64]          --
├─ConvTBlock: 1-15                       [1, 64, 128, 128]         --
│    └─ConvTranspose2d: 2-47             [1, 64, 128, 128]         262,208
│    └─InstanceNorm2d: 2-48              [1, 64, 128, 128]         --
│    └─Dropout: 2-49                     [1, 64, 128, 128]         --
│    └─ReLU: 2-50                        [1, 64, 128, 128]         --
├─ConvTBlock: 1-16                       [1, 3, 256, 256]          --
│    └─ConvTranspose2d: 2-51             [1, 3, 256, 256]          6,147
│    └─Dropout: 2-52                     [1, 3, 256, 256]          --
├─Tanh: 1-17                             [1, 3, 256, 256]          --
==========================================================================================
Total params: 54,409,603
Trainable params: 54,409,603
Non-trainable params: 0
Total mult-adds (G): 18.14
==========================================================================================
Input size (MB): 0.79
Forward/backward pass size (MB): 33.72
Params size (MB): 217.64
Estimated Total Size (MB): 252.15
==========================================================================================
```

```python
# TODO15 implement a dataloader
from torchvision import transforms
from torch.utils.data import DataLoader, Dataset, RandomSampler

class Pix2PixDataset(Dataset):
    def __init__(self, x, y, transformer):
        self.x = x.astype(np.float32)
        self.y = y.astype(np.float32)
        self.transformer = transformer

    def __getitem__(self, index):
        x = torch.tensor(self.x[index])
        y = torch.tensor(self.y[index])

        if self.transformer:
            xy_combined = torch.cat((x.unsqueeze(0), y.unsqueeze(0)), dim=0)
            xy_transformed = self.transformer(xy_combined)

            x_transformed, y_transformed = torch.split(xy_transformed, [1, 1], dim=0)
            x_transformed = x_transformed.squeeze(0)
            y_transformed = y_transformed.squeeze(0)
        else:
            x_transformed, y_transformed = x, y

        return x_transformed, y_transformed

    def __len__(self):
        return self.x.shape[0]

transformer = transforms.Compose([
  transforms.Resize(286),
  transforms.RandomCrop(256),
  transforms.RandomHorizontalFlip(0.5),
])

train_dataset = Pix2PixDataset(trainX, trainY, transformer)
train_loader = DataLoader(train_dataset, batch_size=1, shuffle=True)
```

```python
#TODO16 show that the dataloader is working properly
def to_image(x):
  x = x.numpy()
  return (0.5 * x.transpose((1, 2, 0)) + 0.5)[..., ::-1]

plt.figure(figsize = (10, 10))
for i in range(8):
  x, y = next(iter(train_loader))
  plt.subplot(4, 4, 2*i+1)
  plt.imshow(to_image(x[0]), cmap="gray")
  plt.subplot(4, 4, 2*i+2)
  plt.imshow(to_image(y[0]), cmap="gray")
plt.show()
```



```python
losses = {'D' : [None], 'G' : [None]}

for i in tqdm(range(20001)):
  #TODO17 sample the data from the dataloader
  x, y = next(iter(train_loader))
  x = x.cuda()
  y = y.cuda()

  #TODO18 calculate the discriminator loss and update the discriminator
  discriminator.train()
  D_optimizer.zero_grad()

  # Real pairs
  real_pairs = torch.cat((x, y), dim=1)
  D_real = discriminator(real_pairs)
  D_real_loss = torch.log(D_real + 1e-12).mean()

  # Fake pairs
  fake_y = generator(x)
  fake_pairs = torch.cat((x, fake_y), dim=1)
  D_fake = discriminator(fake_pairs.detach())
  D_fake_loss = torch.log(1 - D_fake + 1e-12).mean()

  # Combined D loss
  D_loss = -0.5 * (D_real_loss + D_fake_loss)  # Multiply by -0.5 as suggested
  D_loss.backward()
  D_optimizer.step()

  losses["D"].append(D_loss.item())

  #TODO19 calculate the generator loss and update the generator
  generator.train()
  G_optimizer.zero_grad()

  G_fake = discriminator(fake_pairs)
  G_gan_loss = -torch.log(G_fake + 1e-12).mean()

  G_L1_loss = F.l1_loss(fake_y, y) * LAMBDA

  G_loss = G_gan_loss + G_L1_loss
  G_loss.backward()
  G_optimizer.step()

  losses["G"].append(G_loss.item())

  # Output visualization : If your reimplementation is correct, the generated images should start resembling a facade after 2,500 iterations
  discriminator.eval()
  generator.eval()
  if(i % 2500 == 0):
    with torch.no_grad():
      print(losses['D'][-1], losses['G'][-1])
      plt.figure(figsize = (40, 16))
      gs1 = gridspec.GridSpec(4, 4)
      gs1.update(wspace=0.025)

      sampleX_vis = 0.5 * valX[:16][:, ::-1, :, :] + 0.5
      sampleY = 0.5 * valY[:16][:, ::-1, :, :] + 0.5
      sampleX = torch.tensor(valX[:16]).cuda()
      pred_val = 0.5 * generator(sampleX).cpu().detach().numpy()[:, ::-1, :, :] + 0.5
      vis = np.concatenate([sampleX_vis, sampleY, pred_val], axis = 3)
      for i in range(vis.shape[0]):
        ax1 = plt.subplot(gs1[i])
        plt.title('Input / GT / predicted')
        plt.axis('off')
        plt.imshow( vis[i].transpose(1, 2, 0), cmap="gray" )
      plt.show()
```