5번째 미팅발표

임에딘

코멘트

1. BeatGAN – structure 그려오기
2. BeatGAN 모델에 distill 함수 적용하여 통합 Loss 구하기

DCGAN

The structure of G_D learns the architecture of the generator from DCGAN [Radford et al., 2015]. We use 5 1D transposed convolutional layers followed by batch-norm and leaky ReLU activation, with slope of the leak set to 0.2.

DCGAN

DCGAN은 기존 GAN에 존재했던 fully-connected 구조의 대부분을 **CNN구조**로 대체한 것

- Discriminator에서는 모든 pooling layer를 strided convolution으로 바꾸고,
  Generator에서는 pooling layers를 fractional-strided convolutions(=Transposed Convolution)으로 바꾼다.

- Generator와 Discriminator에 Batch normalization을 적용한다.
  이때, Generator의 output layer와 Discriminator의 input layer에는 적용하지 않는다.

- Fully connected hidden layer 삭제

- Generator에서 활성화 함수는 ReLU 사용하되, output layer에서만 Tanh 사용

- Discriminator의 활성화 함수는 LeakyReLU를 사용

Illustration of network structure

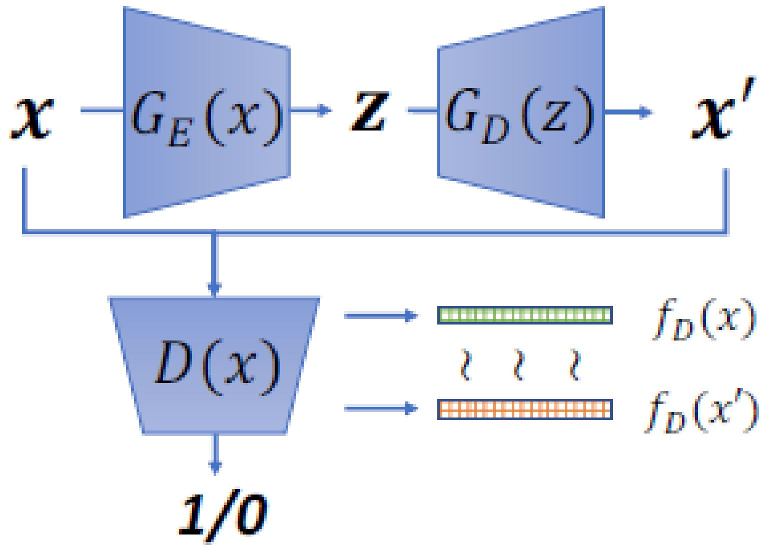
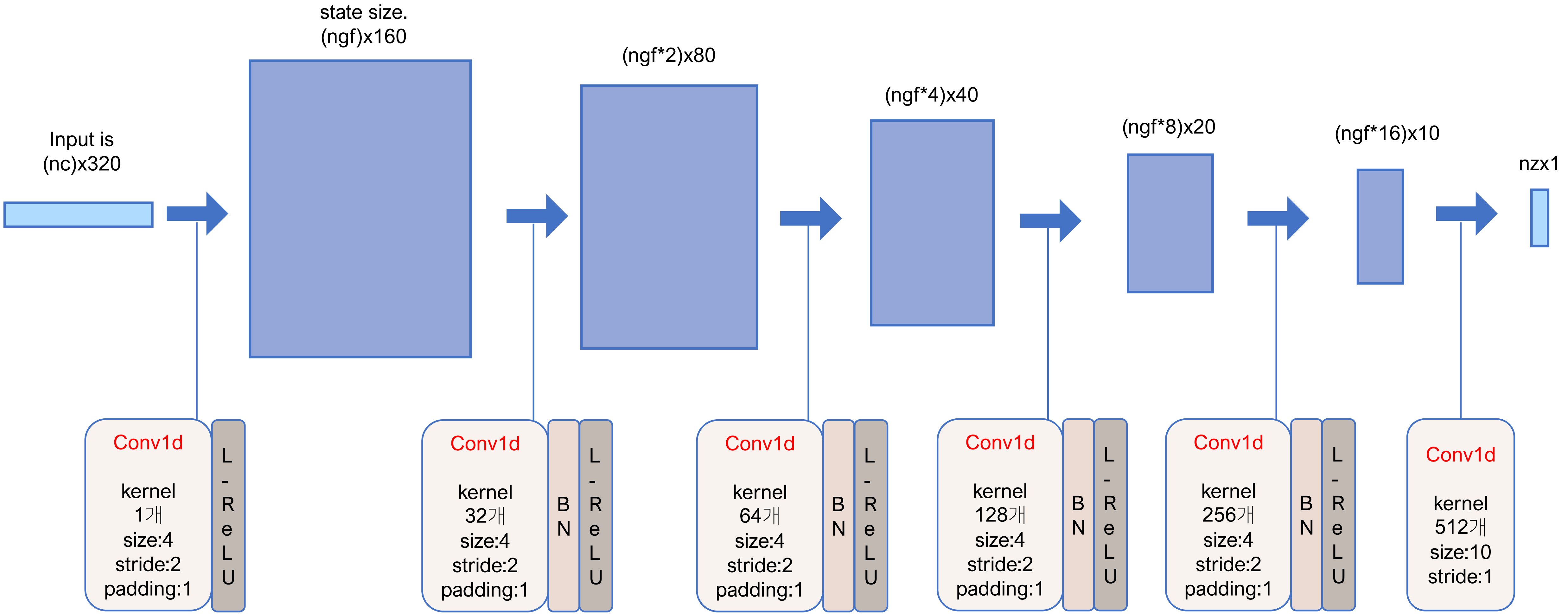
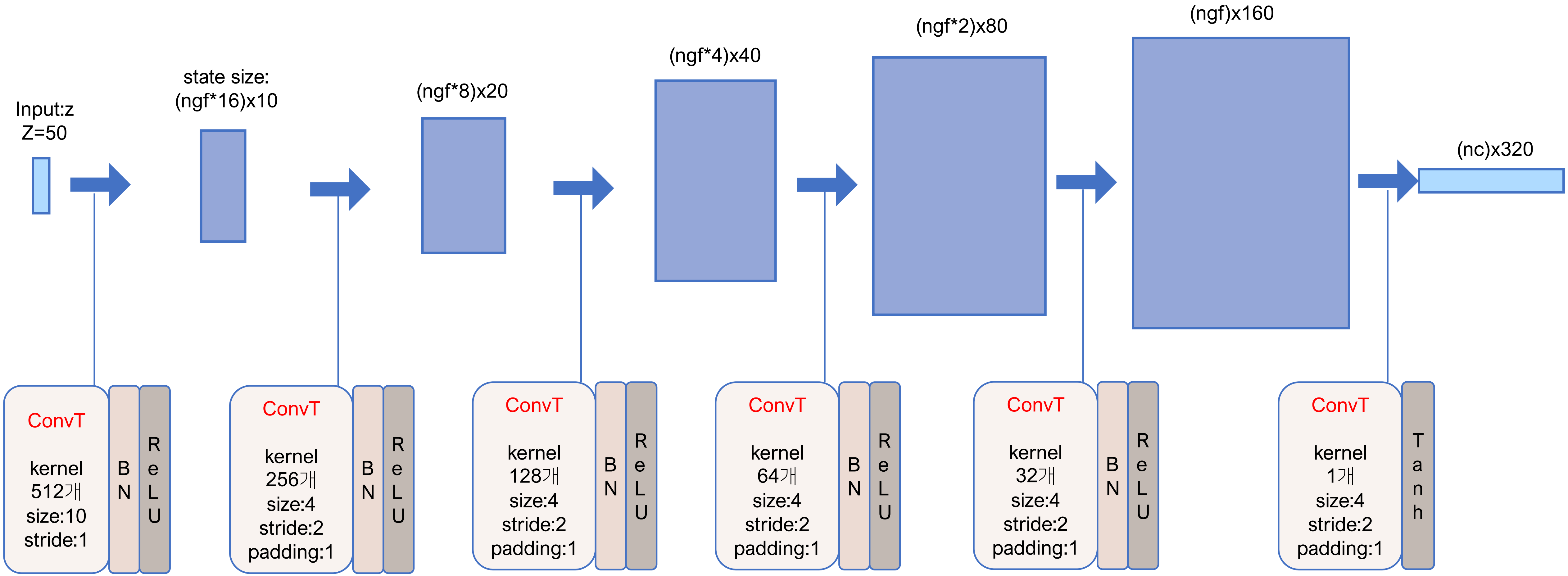


Figure 2: Illustration of our network structure

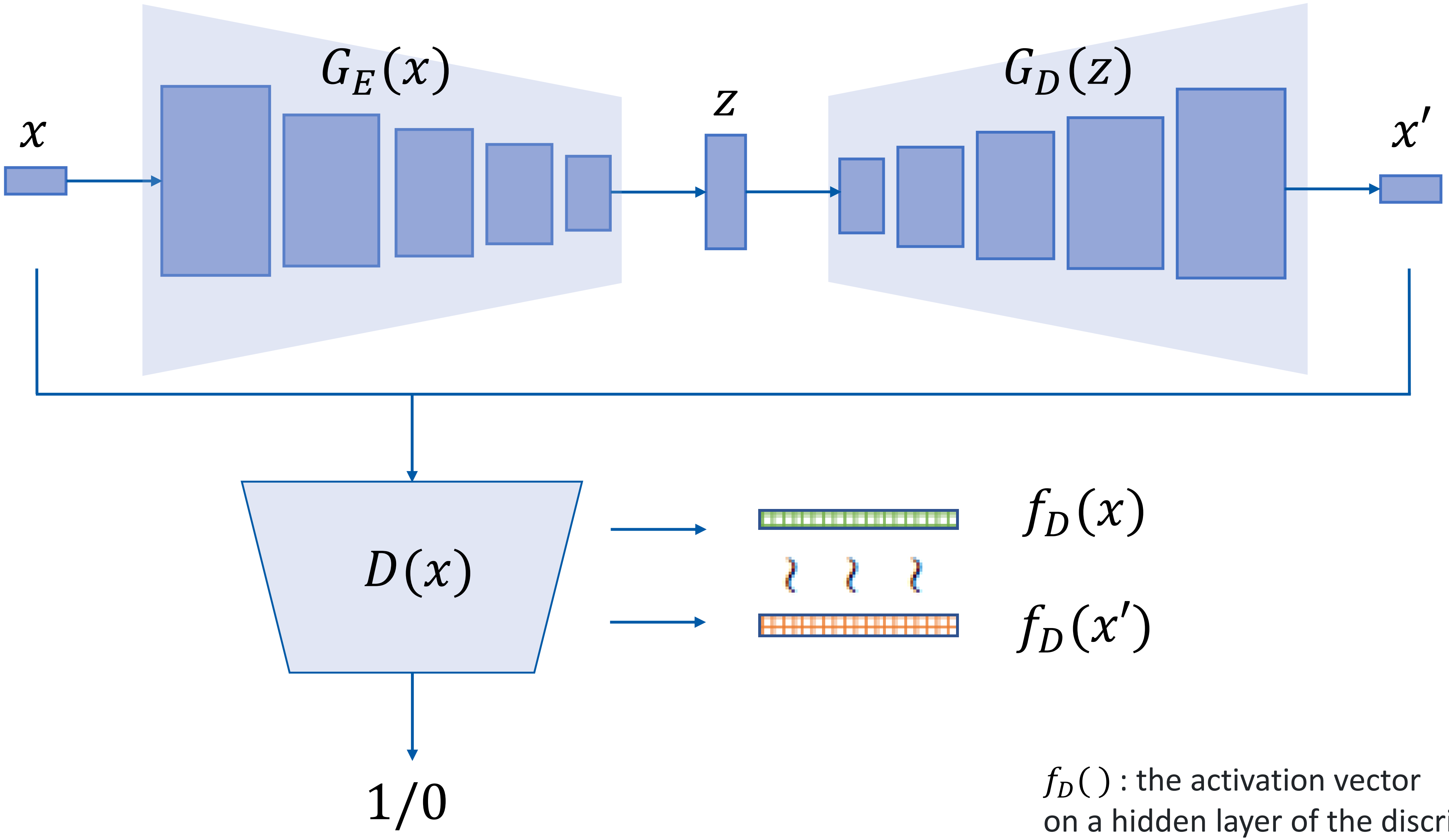
The structure of Encoder network G_E



The structure of Decoder network G_D



network structure



$f_D(\,)$: the activation vector
on a hidden layer of the discriminator

class Discriminator

```
class Discriminator(nn.Module):

    def __init__(self, opt):
        super(Discriminator, self).__init__()
        model = Encoder(opt.ngpu,opt,1)
        layers = list(model.main.children())

        self.features = nn.Sequential(*layers[:-1])
        self.classifier = nn.Sequential(layers[-1])
        self.classifier.add_module('Sigmoid', nn.Sigmoid())

    def forward(self, x):
        features = self.features(x)
        features = features
        classifier = self.classifier(features)
        classifier = classifier.view(-1, 1).squeeze(1)

        return classifier, features
```

layers :
encoder에서 nn.sequential을 사용해 구현한 신경망을 이용

classifier에
기존의 time series인지 (x)
Generator가 생성한 time series인지 (x')
판단하기 위해 Sigmoid적용

classifier와 features를 리턴

class Generator

```
class Generator(nn.Module):

    def __init__(self, opt):
        super(Generator, self).__init__()
        self.encoder1 = Encoder(opt.ngpu,opt,opt.nz)
        self.decoder = Decoder(opt.ngpu,opt)

    def forward(self, x):
        latent_i = self.encoder1(x)
        gen_x = self.decoder(latent_i)
        return gen_x, latent_i
```

network 구조 그림에 따르면,

$G_E(x) = z$

$G_D(z) = x'$

class BeatGAN – def __init__ ()

```
class BeatGAN(AD_MODEL):
```

```
def __init__(self, opt, dataloader, device):
    super(BeatGAN, self).__init__(opt, dataloader, device)
    self.dataloader = dataloader
    self.device = device
    self.opt=opt

    self.batchsize = opt.batchsize
    self.nz = opt.nz
    self.niter = opt.niter

    self.G = Generator( opt).to(device)
    self.G.apply(weights_init)
    if not self.opt.istest:
        print_network(self.G)

    self.D = Discriminator(opt).to(device)
    self.D.apply(weights_init)
    if not self.opt.istest:
        print_network(self.D)
```

```
self.out_d_real = None
self.feat_real = None

self.fake = None
self.latent_i = None
self.out_d_fake = None
self.feat_fake = None

self.err_d_real = None
self.err_d_fake = None
self.err_d = None

self.out_g = None
self.err_g_adv = None
self.err_g_rec = None
self.err_g = None
```

class BeatGAN – def __init__ ()

BCELoss (이진 교차 엔트로피 손실)
결과가 1이나 0 으로 명확히 나뉘는 이산형 변수를 예측하는 분류 문제에 적합

```
self.bce_criterion = nn.BCELoss()
self.mse_criterion=nn.MSELoss()


self.optimizerD = optim.Adam(self.D.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
self.optimizerG = optim.Adam(self.G.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))


self.total_steps = 0
self.cur_epoch=0


self.input = torch.empty(size=(self.opt.batchsize, self.opt.nc, self.opt.isize), dtype=torch.float32, device=self.device)
self.label = torch.empty(size=(self.opt.batchsize,), dtype=torch.float32, device=self.device)
self.gt    = torch.empty(size=(opt.batchsize,), dtype=torch.long, device=self.device)
self.fixed_input = torch.empty(size=(self.opt.batchsize, self.opt.nc, self.opt.isize), dtype=torch.float32, device=self.device)
self.real_label = 1
self.fake_label= 0
```

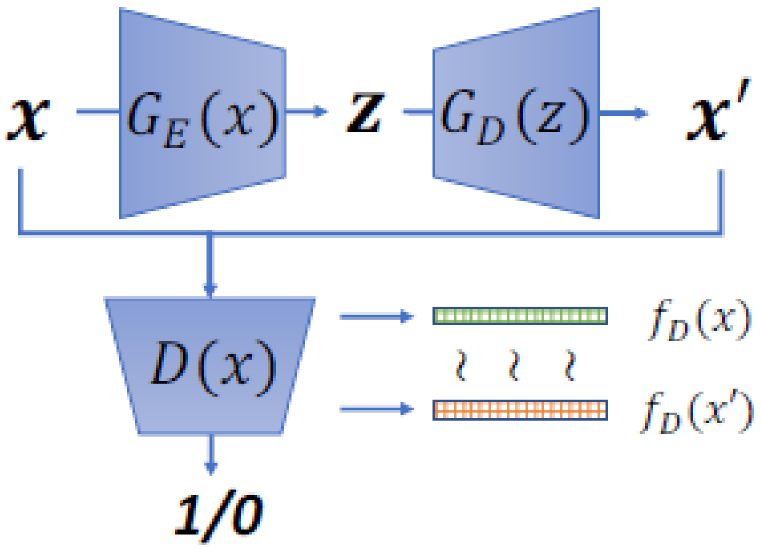
Adam Optimizer : 많은 작업에 대해 대체로 SGD Optimizer보다 나은 성능을 보여줌
(관성을 이용해 국소 최적화 문제 해결, 각 학습 파라미터마다 다른 학습률 적용 가능)

def update_netg(self):

```
def update_netg(self):
    self.G.zero_grad()
    self.label.data.resize_(self.opt.batchsize).fill_(self.real_label)
    self.fake, self.latent_i = self.G(self.input)
    self.out_g, self.feat_fake = self.D(self.fake)
    _, self.feat_real = self.D(self.input)


    # self.err_g_adv = self.bce_criterion(self.out_g, self.label)   # loss for ce
    self.err_g_adv=self.mse_criterion(self.feat_fake,self.feat_real)  # loss for feature matching
    self.err_g_rec = self.mse_criterion(self.fake, self.input)  # constrain x' to look like x


    self.err_g =  self.err_g_rec + self.err_g_adv * self.opt.w_adv
    self.err_g.backward()
    self.optimizerG.step()
```

- 기울기 초기화
- Train with real
- Train with fake
- err_g 구함
- 역전파 후 가중치 갱신

손실 (loss/error)



err_g=err_g_rec+err_g_adv

- err_g_rec (constrain x’ to look like x)
  (fake(x’)와 input(x)의 mse loss)
- err_g_adv (feature matching loss)
  (feat_fake와 feat_real의 mse loss)

err_d = err_d_real+err_d_fake

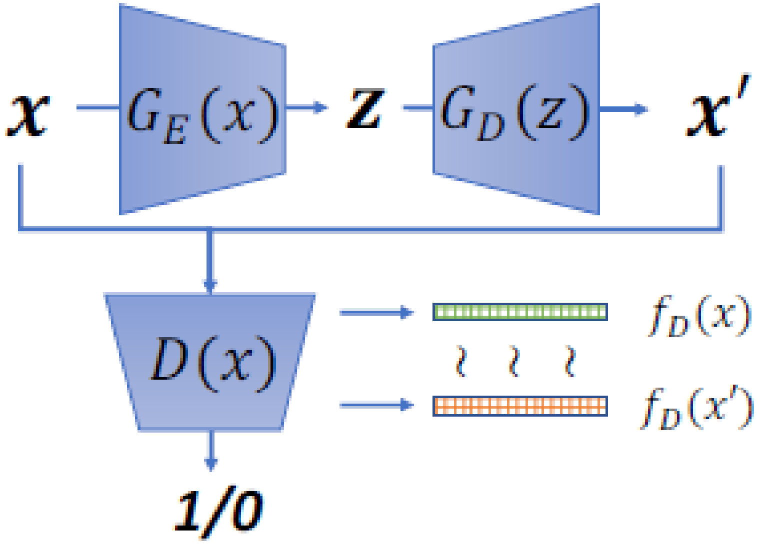
- err_d_real (out_d_real과 real_label의 bce loss)
- err_d_fake (out_d_fake와 fake_label의 bce loss)

def update_netd(self):

```
    def update_netd(self):
        ##

        self.D.zero_grad()
        # --
        # Train with real
        self.label.data.resize_(self.opt.batchsize).fill_(self.real_label)
        self.out_d_real, self.feat_real = self.D(self.input)
        # --
        # Train with fake
        self.label.data.resize_(self.opt.batchsize).fill_(self.fake_label)
        self.fake, self.latent_i = self.G(self.input)
        self.out_d_fake, self.feat_fake = self.D(self.fake)
        # --


        self.err_d_real = self.bce_criterion(self.out_d_real, torch.full((self.batchsize,), self.real_label, device=self.device))
        self.err_d_fake = self.bce_criterion(self.out_d_fake, torch.full((self.batchsize,), self.fake_label, device=self.device))


        self.err_d=self.err_d_real+self.err_d_fake
        self.err_d.backward()
        self.optimizerD.step()
```

- 기울기 초기화
- Train with real
- Train with fake
- err_d 구함
- 역전파 후 가중치 갱신

손실 (loss/error)



err_g=err_g_rec+err_g_adv

- err_g_rec (constrain x’ to look like x)
  (fake(x’)와 input(x)의 mse loss)
- err_g_adv (feature matching loss)
  (feat_fake와 feat_real의 mse loss)

err_d = err_d_real+err_d_fake

- err_d_real (out_d_real과 real_label의 bce loss)
- err_d_fake (out_d_fake와 fake_label의 bce loss)

def validate, def get_errors

```python
def validate(self):
    '''
    validate by auc value
    :return: auc
    '''
    y_,y_pred=self.predict(self.dataloader["val"])
    rocprc,rocauc,best_th,best_f1=evaluate(y_,y_pred)
    return rocauc,best_th,best_f1
```

```python
def get_errors(self):

    errors = {'err_d':self.err_d.item(),
                'err_g': self.err_g.item(),
                'err_d_real': self.err_d_real.item(),
                'err_d_fake': self.err_d_fake.item(),
                'err_g_adv': self.err_g_adv.item(),
                'err_g_rec': self.err_g_rec.item(),
              }


    return errors
```

def predict() :

```python
def predict(self,dataloader_,scale=True):
    with torch.no_grad():

        self.an_scores = torch.zeros(size=(len(dataloader_.dataset),), dtype=torch.float32, device=self.device)
        self.gt_labels = torch.zeros(size=(len(dataloader_.dataset),), dtype=torch.long,    device=self.device)
        self.latent_i  = torch.zeros(size=(len(dataloader_.dataset), self.opt.nz), dtype=torch.float32, device=self.device)
        self.dis_feat = torch.zeros(size=(len(dataloader_.dataset), self.opt.ndf*16*10), dtype=torch.float32,
                                    device=self.device)


        for i, data in enumerate(dataloader_, 0):

            self.set_input(data)
            self.fake, latent_i = self.G(self.input)


            # error = torch.mean(torch.pow((d_feat.view(self.input.shape[0],-1)-d_gen_feat.view(self.input.shape[0],-1)), 2), dim=1)
            #
            error = torch.mean(
                torch.pow((self.input.view(self.input.shape[0], -1) - self.fake.view(self.fake.shape[0], -1)), 2),
                dim=1)

            self.an_scores[i*self.opt.batchsize : i*self.opt.batchsize+error.size(0)] = error.reshape(error.size(0))
            self.gt_labels[i*self.opt.batchsize : i*self.opt.batchsize+error.size(0)] = self.gt.reshape(error.size(0))
            self.latent_i [i*self.opt.batchsize : i*self.opt.batchsize+error.size(0), :] = latent_i.reshape(error.size(0), self.opt.nz)


        # Scale error vector between [0, 1]
        if scale:
            self.an_scores = (self.an_scores - torch.min(self.an_scores)) / (torch.max(self.an_scores) - torch.min(self.an_scores))

        y_=self.gt_labels.cpu().numpy()
        y_pred=self.an_scores.cpu().numpy()


        return y_,y_pred
```

리턴되는 y_, y_pred

y_ : ground truth label
y_pred : anomalous score

def predict → an_scores

```
for i, data in enumerate(dataloader_, 0):

    self.set_input(data)
    self.fake, latent_i = self.G(self.input)


    # error = torch.mean(torch.pow((d_feat.view(self.input.shape[0],-1)-d_gen_feat.view(self.input.shape[0],-1)), 2), dim=1)
    #
    error = torch.mean(
        torch.pow((self.input.view(self.input.shape[0], -1) - self.fake.view(self.fake.shape[0], -1)), 2),
        dim=1)

    self.an_scores[i*self.opt.batchsize : i*self.opt.batchsize+error.size(0)] = error.reshape(error.size(0))
    self.gt_labels[i*self.opt.batchsize : i*self.opt.batchsize+error.size(0)] = self.gt.reshape(error.size(0))
    self.latent_i [i*self.opt.batchsize : i*self.opt.batchsize+error.size(0), :] = latent_i.reshape(error.size(0), self.opt.nz)
```

Then, the anomalousness score for x is calculated as:

$$A(x) = ||x - G(x)||_2 \qquad (2)$$

def predict → gt_labels

```
for i, data in enumerate(dataloader_, 0):

    self.set_input(data)
    self.fake, latent_i = self.G(self.input)


    # error = torch.mean(torch.pow((d_feat.view(self.input.shape[0],-1)-d_gen_feat.view(self.input.shape[0],-1)), 2), dim=1)
    #
    error = torch.mean(
        torch.pow((self.input.view(self.input.shape[0], -1) - self.fake.view(self.fake.shape[0], -1)), 2),
        dim=1)

    self.an_scores[i*self.opt.batchsize : i*self.opt.batchsize+error.size(0)] = error.reshape(error.size(0))
    self.gt_labels[i*self.opt.batchsize : i*self.opt.batchsize+error.size(0)] = self.gt.reshape(error.size(0))
    self.latent_i [i*self.opt.batchsize : i*self.opt.batchsize+error.size(0), :] = latent_i.reshape(error.size(0), self.opt.nz)
```

BeatGAN모델의 init함수에서,

```
self.gt     = torch.empty(size=(opt.batchsize,),
                        dtype=torch.long, device=self.device)
```

gt : ground truth

```
def set_input(self, input):
    #[old/error!] self.input.data.resize_(input[0].size()).copy_(input[0])
    with torch.no_grad():
        self.input.resize_(input[0].size()).copy_(input[0])
    #[old/error!]self.gt.data.resize_(input[1].size()).copy_(input[1])
    with torch.no_grad():
        self.gt.resize_(input[1].size()).copy_(input[1])

    # fixed input for view
    if self.total_steps == self.opt.batchsize:
        self.fixed_input.data.resize_(input[0].size()).copy_(input[0])
```

```python
class Encoder(nn.Module):
    def __init__(self, ngpu,opt,out_z):
        super(Encoder, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 320
            nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 160
            nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 80
            nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 40
            nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 20
            nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*16) x 10

            nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
            # state size. (nz) x 1
        )

    def forward(self, input):
        if input.is_cuda and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)

        return output
```

```python
class Decoder(nn.Module):
    def __init__(self, ngpu,opt):
        super(Decoder, self).__init__()
        self.ngpu = ngpu
        self.main=nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose1d(opt.nz,opt.ngf*16,10,1,0,bias=False),
            nn.BatchNorm1d(opt.ngf*16),
            nn.ReLU(True),
            # state size. (ngf*16) x10
            nn.ConvTranspose1d(opt.ngf * 16, opt.ngf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 20
            nn.ConvTranspose1d(opt.ngf * 8, opt.ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 40
            nn.ConvTranspose1d(opt.ngf * 4, opt.ngf*2, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf*2),
            nn.ReLU(True),
            # state size. (ngf*2) x 80
            nn.ConvTranspose1d(opt.ngf * 2, opt.ngf , 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf ),
            nn.ReLU(True),
            # state size. (ngf) x 160
            nn.ConvTranspose1d(opt.ngf , opt.nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 320


        )

    def forward(self, input):
        if input.is_cuda and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)
        return output
```

distill.py

```python
class DistillKL(nn.Module):
    def __init__(self, args):
        super(DistillKL, self).__init__()
        self.T = args.temp

    def forward(self, y_s, y_t):
        B, C, H, W = y_s.size()
        p_s = F.log_softmax(y_s/self.T, dim=1)
        p_t = F.softmax(y_t/self.T, dim=1)
        loss = F.kl_div(p_s, p_t.detach(), reduction='sum') * (self.T**2) / (B * H * W)
        return loss
```

$$
\begin{aligned}
&\mathcal{L}_{KD}(x;\theta_c,\theta_t,K) \\
&= D_{KL}(\mathrm{softmax}(\frac{f_c(x;\theta_c)}{K})||\mathrm{softmax}(\frac{f_t(x;\theta_t)}{K}))
\end{aligned}
$$

distill.py

```python
def att(args, bifpn):
    return Attention(args)

class Attention(nn.Module):
    def __init__(self, args):
        super(Attention, self).__init__()
        self.p = 2
        self.kd = DistillKL(args)
        self.alpha = args.alpha
        self.beta = args.beta

    def forward(self, o_s, o_t, g_s, g_t):
        loss = self.alpha * self.kd(o_s, o_t)
        loss += self.beta * sum([self.at_loss(f_s, f_t.detach()) for f_s, f_t in zip(g_s, g_t)])

        return loss

    def at_loss(self, f_s, f_t):
        return (self.at(f_s) - self.at(f_t)).pow(2).mean()

    def at(self, f):
        return F.normalize(f.pow(self.p).mean(1).view(f.size(0), -1))
```

$$\mathcal{L}_F(T, F; \theta_c, \theta_t) = \Sigma_{i=1}^{n} ||\phi(T_i) - \phi(F_i)||_2$$

$$\alpha \cdot \mathcal{L}_{KD}(x; \theta_c, \theta_t, K) + \beta \cdot \mathcal{L}_F(T, F; \theta_c, \theta_t)$$