7번째 미팅발표

2021-11-17

임에딘

코멘트

1. KL divergence 대신 음수 취해서 곱하고 training
2. Feature distillation loss 함수 정의
3. Feature을 다르게 매핑하여 latent space와 다른 곳에 지정되도록 생각

Knowledge distillation loss 변경

```python
class DistillKL(nn.Module):
    def __init__(self, opt):
        super(DistillKL, self).__init__()
        opt = Options().parse()
        self.T = opt.temperature
        #self.alpha= opt.alpha

    def forward(self, y_s, y_t):
        #B, C, H, W = y_s.size()
        y_s=torch.from_numpy(y_s)
        y_t=torch.from_numpy(y_t)
        y_s=y_s.reshape(1,len(y_s))
        y_t=y_t.reshape(1,len(y_t))
        #print("y_s reshape: ", y_s.shape)
        #print("y_t reshape: ",y_t.shape)
        p_s = F.log_softmax(y_s/self.T, dim=1)
        p_t = F.softmax(y_t/self.T, dim=1)
        #loss = self.alpha*F.kl_div(p_s, p_t.detach(), reduction='sum') * (self.T**2) / y_s.shape[0]
        loss= (-1)*p_s*p_t.detach()
        return loss
```

$$\mathcal{L}_{old}(\mathbf{y}_o, \hat{\mathbf{y}}_o) = -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o) = -\sum_{i=1}^{l} y_o'^{(i)} \log \hat{y}_o'^{(i)}$$

$$y_o'^{(i)} = \frac{(y_o^{(i)})^{1/T}}{\sum_j (y_o^{(j)})^{1/T}}, \quad \hat{y}_o'^{(i)} = \frac{(\hat{y}_o^{(i)})^{1/T}}{\sum_j (\hat{y}_o^{(j)})^{1/T}}.$$

Learning without Forgetting (2017 CVPR)

결과

```
################################
#########  Result  ############
ap:0.901716198219636
auc:0.9348297707462155
best th:0.009415941312909126 --> best f1:0.8019966722129783
(py37)
edin@DESKTOP-FF4AVFF MINGW64 ~/Desktop/4-2/캡디1/코드/BeatGAN-master (master)
$
```

목표인 0.95이상의 AUC보다는 성능이 낮게 나왔다
KL발산으로 loss를 냈을 때의 AUC인 0.9474보다 낮게 나왔다

output distillation, feature distillation

*** 저번 미팅시간에서 제기된 문제 :**

교사 네트워크의 output만 지식을 전달하는 것은 학생 네트워크가 groud truth를 교육하는 것과 유사하므로 성능 향상 크게 없음

*** 지식 증류에 관한 연구 :**

1) 교사 네트워크에 포함된 정보를 더 이용하기 위해 output 증류 대신 feature 증류에 대해 몇 가지 접근 방식을 제안한 연구
- FitNets (ICLR, 2015)

2) 특징을 축소된 차원을 갖는 표현으로 변환하여 학생에게 전달하는 방법 제안한 연구
- Paying more attention to attention : Improving the performance of convolutional neural networks via attention transfer (ICLR,2017)
- A gift from knowledge distillation : Fast optimization, network minimization and transfer learning (CVPR, 2017)

3) 증류에서 전달되는 정보의 양을 증가시키는 방법을 제안한 연구
- Paraphrasing complex network : Network compression via factor transfer (NIPS, 2018)
- Knowledge transfer via distillation of activation boundaries formed by hidden neurons (AAAI, 2019)

Feature Distillation 관련된 연구 논문 요약

A Comprehensive Overhaul of Feature Distillation (2019 ICCV)

- 네트워크를 압축하는 과정에서의 특징 증류 방법을 새롭게 제안
  (교사 변환, 학생 변환, 증류 특정 위치, 거리 함수 측면 고려)

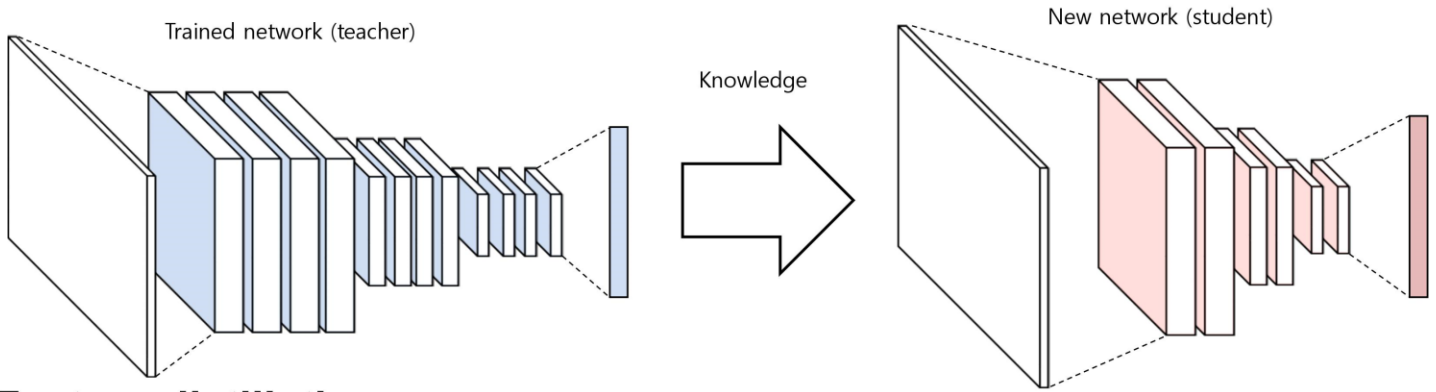
- 증류 손실 정의 :
  새로 설계한 활성화 함수 margin ReLU, 특징 증류 위치를 ReLU 앞쪽으로 변경한 pre-ReLU 방법 , 부분 L2 거리 함수를 사용하여 학생모델이 훈련하는 과정에서 부정적인 영향을 주는 불필요한 정보의 증류 생략

- 연구 결과 : 이미지 분류, 객체 감지, 의미 분할 등 다양한 작업에서 향상된 성능을 보여줌

https://sites.google.com/view/byeongho-heo/overhaul

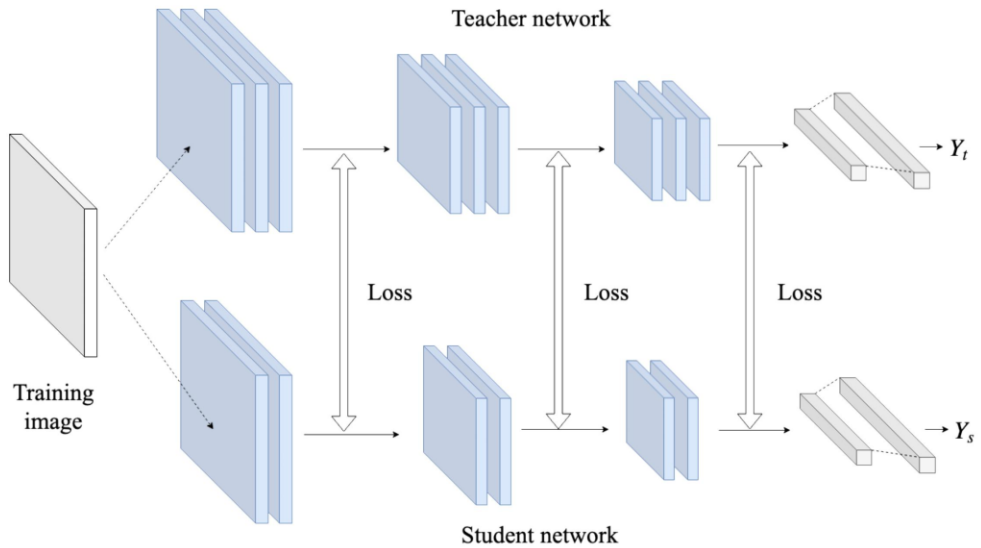
Knowledge distillation

- Transfer from knowledge from trained network (teacher)
  to a new network (student)
- Network compression (large teacher to small student)



Feature distillation

- Knowledge distillation based on hidden layer response (feature)



A Comprehensive Overhaul of Feature Distillation (2019 ICCV)

Feature position : pre-ReLU

- Previous feature position is not consistent for various architecture
- Distillation needs consistent feature position
- We selects pre-ReLU position before spatial resolution reduction as distillation position

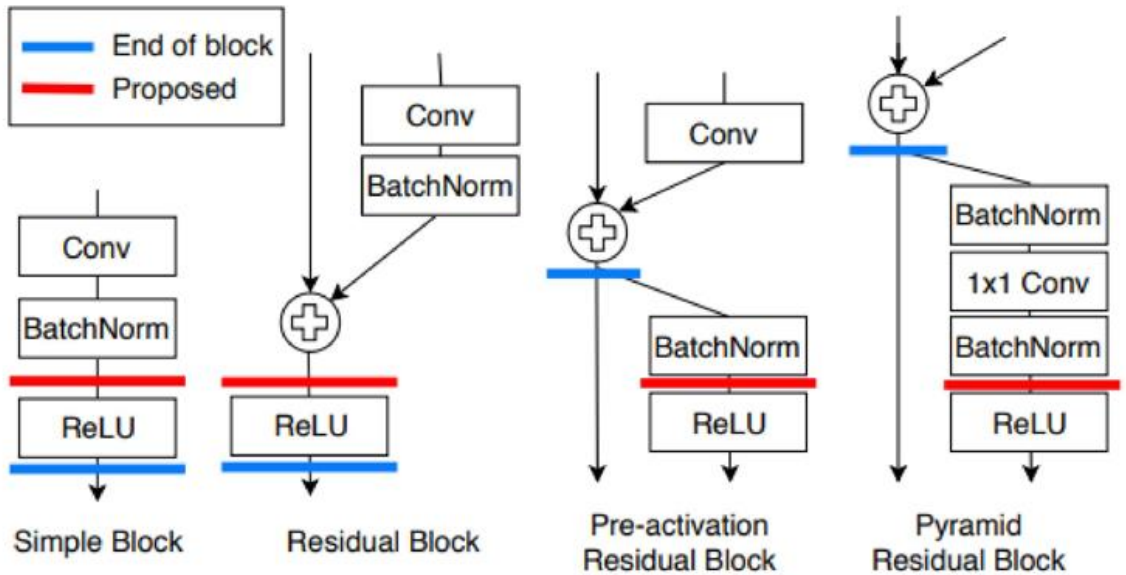
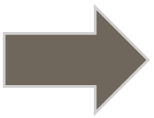


Figure 3. Position of distillation target layer. We place the distillation layer between the last block and the first ReLU. The exact location differs according to the network architecture.

Encoder 수정

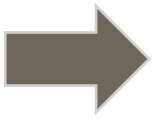
```python
class Encoder(nn.Module):
    def __init__(self, ngpu,opt,out_z):
        super(Encoder, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 320
            nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # 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
        )
```



```python
class Encoder(nn.Module):
    def __init__(self, ngpu,opt,out_z):
        super(Encoder, self).__init__()
        self.ngpu = ngpu
        self.conv1=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
        self.conv2=nn.Sequential(
            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
        self.conv3=nn.Sequential(
            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
        self.conv4=nn.Sequential(
            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
        self.conv5=nn.Sequential(
            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
        self.conv6=nn.Sequential(
            nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False)
        )
        # state size. (nz) x 1
```

Decoder 수정

```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
```



```python
class Decoder(nn.Module):
    def __init__(self, ngpu,opt):
        super(Decoder, self).__init__()
        self.ngpu = ngpu
        self.conv1=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
        self.conv2=nn.Sequential(
            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
        self.conv3=nn.Sequential(
            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
        self.conv4=nn.Sequential(
            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
        self.conv5=nn.Sequential(
            nn.ConvTranspose1d(opt.ngf * 2, opt.ngf , 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf ),
            nn.ReLU(True)
        )
        # state size. (ngf) x 160
        self.conv6=nn.Sequential(
            nn.ConvTranspose1d(opt.ngf , opt.nc, 4, 2, 1, bias=False),
            nn.Tanh()
        )
        # state size. (nc) x 320
```

Encoder, Decoder 수정 (forward함수)

```python
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
def forward(self, input):

    output=self.conv1(input)
    output=self.conv2(output)
    output=self.conv3(output)
    output=self.conv4(output)
    output=self.conv5(output)
    output=self.conv6(output)

    return output
```

기존의 코드 Discriminator, Generator

```python
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
```

```python
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
```

Discriminator 수정

기존: layer을 한번에 받아와서 features 정의
변경: layer 차례로 받아오면서 feature도 차례로 받아왔고 리스트 feat_list를 정의

```
class Discriminator(nn.Module):

    def __init__(self, opt):
        super(Discriminator, self).__init__()
        model = Encoder(opt.ngpu,opt,1)
        #layers = list(model.main.children())
        layers=list(model.conv1.children())
        self.feat1=nn.Sequential(*layers[:-1])
        layers.extend(list(model.conv2.children()))
        self.feat2=nn.Sequential(*layers[:-1])
        layers.extend(list(model.conv3.children()))
        self.feat3=nn.Sequential(*layers[:-1])
        layers.extend(list(model.conv4.children()))
        self.feat4=nn.Sequential(*layers[:-1])
        layers.extend(list(model.conv5.children()))
        self.feat5=nn.Sequential(*layers[:-1])
        layers.extend(list(model.conv6.children()))
        self.features=nn.Sequential(*layers[:-1])

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

```
def forward(self, x):
    #feature 1~6 (6: features)
    feat1=self.feat1(x)
    feat1=feat1
    feat2=self.feat2(x)
    feat2=feat2
    feat3=self.feat3(x)
    feat3=feat3
    feat4=self.feat4(x)
    feat4=feat4
    feat5=self.feat5(x)
    feat5=feat5

    features = self.features(x)
    features = features

    feat_list=[feat1,feat2,feat3,feat4,feat5,features]

    classifier = self.classifier(features)
    classifier = classifier.view(-1, 1).squeeze(1)

    return classifier, features ,feat_list
```

knowledge distillation(KL), feature distillation(FT)

```
self.bce_criterion = nn.BCELoss()
self.mse_criterion=nn.MSELoss()
self.kd_criterion=DistillKL(opt)
self.ft_criterion=DistillFT(opt)
```

```
#knowledge distillation
class DistillKL(nn.Module):
    def __init__(self, opt):
        super(DistillKL, self).__init__()
        opt = Options().parse()
        self.T = opt.temperature
        #self.alpha= opt.alpha

    def forward(self, y_s, y_t):
        #B, C, H, W = y_s.size()
        y_s=torch.from_numpy(y_s)
        y_t=torch.from_numpy(y_t)
        y_s=y_s.reshape(1,len(y_s))
        y_t=y_t.reshape(1,len(y_t))
        p_s = F.log_softmax(y_s/self.T, dim=1)
        p_t = F.softmax(y_t/self.T, dim=1)
        loss = self.alpha*F.kl_div(p_s, p_t.detach(), reduction='sum') * (self.T**2) / y_s.shape[0]
        #loss= (-1)*p_s*p_t.detach()
        return loss
```

knowledge distillation(KL), feature distillation(FT)

```
self.bce_criterion = nn.BCELoss()
self.mse_criterion=nn.MSELoss()
self.kd_criterion=DistillKL(opt)
self.ft_criterion=DistillFT(opt)
```

```python
#feature distillation
class DistillFT(nn.Module):
    def __init__(self, opt):
        super(DistillFT, self).__init__()
        opt = Options().parse()
        self.p = 2

    def forward(self, g_s, g_t):
        loss = 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))
```

손실 함수 통합

```
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.feat_list_fake = self.D(self.fake)
    _, self.feat_real, self.feat_list_real = self.D(self.input)



    # self.err_g_adv = self.bce_criterion(self.out_g, self.label)   # loss for ce
1   self.err_g_adv=self.mse_criterion(self.feat_fake,self.feat_real)  # loss for feature matching
2   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 = self.err_g_rec + self.err_g_adv * self.opt.w_adv
3   self.err_g+= self.kd_criterion(self.y_,self.teacher.y_pred_t) #knowledge distillation
4   self.err_g+= self.ft_criterion(self.feat_list_fake,self.feat_list_real)# feature distillation

    self.err_g.backward()
    self.optimizerG.step()
```

① f(x)와 f(x')의 손실
② x와 x'의 손실
③ knowledge distillation 손실
④ features 손실

train/eval teacher model & train/eval student model

```
##################################
########  Result  ############
ap:0.8944049200563835
auc:0.9320394393071163
best th:0.006316778715699911 --> best f1:0.80494722648440003
(py37)
edin@DESKTOP-FF4AVFF MINGW64 ~/Desktop/4-2/캡디1/코드/BeatGAN-master (master)
$
```

```
##################################
########  Result  ############
ap:0.9144875470715339
auc:0.9472463622432733
best th:0.0040773982182145l2 --> best f1:0.8200832904403164
(py37)
edin@DESKTOP-FF4AVFF MINGW64 ~/Desktop/4-2/캡디1/코드/BeatGAN-master (master)
$
```

teacher model의 feature 정보 전달…

```python
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.feat_list_fake = self.D(self.fake)
    _, self.feat_real, self.feat_list_real = self.D(self.input)
    _,_,self.teacher.feat_list_fake=self.teacher.D(self.fake)
    _,_,self.teacher.feat_list_real=self.teacher.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 = self.err_g_rec + self.err_g_adv * self.opt.w_adv
    self.err_g+= self.kd_criterion(self.y ,self.teacher.y_pred_t) #knowledge distillation
    self.err_g+= self.ft_criterion(self.feat_list_fake,self.teacher.feat_list_fake)# feature distillation

    self.err_g.backward()
    self.optimizerG.step()
```

```
Epoch: [39] [ 200/ 975] D_loss(R/F): 0.000015/0.000533, G_loss: 0.365170
Epoch: [39] [ 300/ 975] D_loss(R/F): 0.000012/0.000153, G_loss: 0.375199
Epoch: [39] [ 400/ 975] D_loss(R/F): 0.000018/0.000132, G_loss: 0.380990
Epoch: [39] [ 500/ 975] D_loss(R/F): 0.000017/0.000027, G_loss: 0.398751
Epoch: [39] [ 600/ 975] D_loss(R/F): 0.000009/0.000200, G_loss: 0.373725
Epoch: [39] [ 700/ 975] D_loss(R/F): 0.000003/0.000127, G_loss: 0.422658
Epoch: [39] [ 800/ 975] D_loss(R/F): 0.000034/0.000136, G_loss: 0.389787
Epoch: [39] [ 900/ 975] D_loss(R/F): 0.000012/0.000057, G_loss: 0.399347
[39] auc:0.7007 th:0.1038 f1:0.4390      best_auc:0.8966 in epoch[4]
```