CycleGANs

1 Main work

图像到图像的转换是一类视觉和图形问题，其目标是使用对齐的图像对训练集来学习输入图像和输出图像之间的映射关系。但是，对于许多任务，配对训练数据并不可用。论文提出了一种在没有配对数据的情况下将图像从源域 X 转换到目标域 Y 的学习方法，其目标是学习一个映射$G : X \rightarrow Y$，使得来自 $G(X)$ 的图像分布与使用对抗性损失的分布 Y 无法区分。由于此映射高度欠约束，论文将其与逆映射 $F : Y \rightarrow X$ 耦合，并引入循环一致性损失以强制执行 $F(G(X)) \approx X$（反之亦然）。最后，论文在不存在配对训练数据的几个任务上进行了定性实验和定量实验，证明了方法的优越性。

2 Method

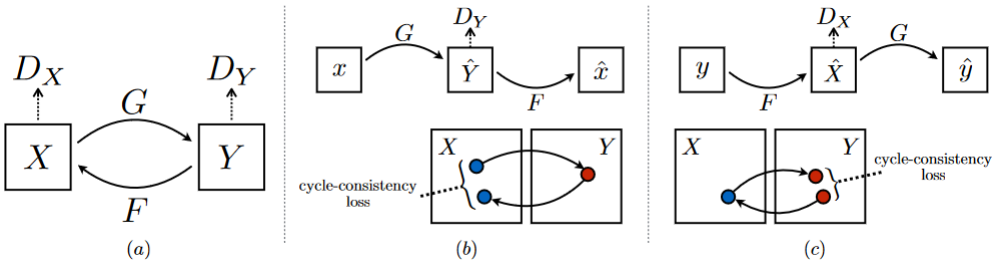

图 1: Frame of CycleGAN

如图 1，论文模型包括两个映射函数 $G : X \rightarrow Y$ 和 $F : Y \rightarrow X$ 以及相关的对抗性鉴别器 D_Y, D_X。给定训练样本 $\{x_i\}_{i=1}^N$, $x_i \in X$, $\{y_i\}_{i=1}^M$, $y_i \in Y$，目标为学习 X 和 Y 之间的映射函数，数据分布表示为 $x \sim p_{data}(x), y \sim p_{data}(y)$， D_X旨在区分图像$\{x\}$和$\{F(y)\}$, D_Y旨在区分$\{y\}$和$\{G(x)\}$。

论文针对学习任务，提出了两种损失约束：对抗性损失和循环一致性损失。其中，对抗性损失用于约束生成图像的分布与目标域中的数据分布匹配，循环一致性损失用于防止所学习到的映射关系G和F彼此矛盾。两种损失表达式分别如下：

对抗性损失：

$$\begin{aligned}\mathcal{L}_{\mathrm{GAN}}\left(G, D_Y, X, Y\right) = & \mathbb{E}_{y \sim p_{\mathrm{data}}(y)}\left[\log D_Y(y)\right] \\ & + \mathbb{E}_{x \sim p_{\mathrm{data}}(x)}\left[\log\left(1 - D_Y(G(x))\right)\right],\end{aligned} \tag{1}$$

对应代码为：

```
# calculate GAN loss for discriminator D_A
self.loss_D_A = self.backward_D_basic(self.netD_A, self.real_B, fake_B)
# Calculate GAN loss for discriminator D_B
self.loss_D_B = self.backward_D_basic(self.netD_B, self.real_A, fake_A)
```

循环一致性损失：

$$\begin{aligned}\mathcal{L}_{\mathrm{cyc}}\left(G, F\right) = & \mathbb{E}_{x \sim p_{\mathrm{data}}(x)}\left[\|F(G(x)) - x\|_1\right] \\ & + \mathbb{E}_{y \sim p_{\mathrm{data}}(y)}\left[\|G(F(y)) - y\|_1\right].\end{aligned} \tag{2}$$

对应代码为：

```
# Forward cycle loss || G_B(G_A(A)) - A||
self.loss_cycle_A = self.criterionCycle(self.rec_A, self.real_A) *
    lambda_A
# Backward cycle loss || G_A(G_B(B)) - B||
self.loss_cycle_B = self.criterionCycle(self.rec_B, self.real_B) *
    lambda_B}
```

在网络结构上，CycleGAN包含三个卷积，几个残差块，两个stride为1/2的小步幅卷积和一个将特征映射回RGB的卷积。对于分辨率为128×128的图像，采用6个block，而对256×256及更高分辨率的图像使用9个block。对于鉴别器网络，CycleGAN使用70×70的PatchGANS，以此对相应分辨率的重叠图像块进行真假分类。这种补丁级鉴别器架构比全图像鉴别器具有更少的参数，并且可以以全卷积的方式处理任意大小的图像。网络部分代码如下所示：

```
if netG == 'resnet_9blocks':
    net = ResnetGenerator(input_nc, output_nc, ngf, norm_layer=
        norm_layer, use_dropout=use_dropout, n_blocks=9)
elif netG == 'resnet_6blocks':
    net = ResnetGenerator(input_nc, output_nc, ngf, norm_layer=
        norm_layer, use_dropout=use_dropout, n_blocks=6)
...
```