神经网络报告 #Final

吴雨欣 191240060 匡亚明学院

1 GAN model

1.1 原理

GAN 包含两个模型，生成模型 (generative model) 和判别模型 (discriminate model)。生成模型的目标是生成与原数据相似的、尽可能真实的实例；而判别模型则需要判断给定的输入实例是真实的还是来自 G 网络构造出来的。

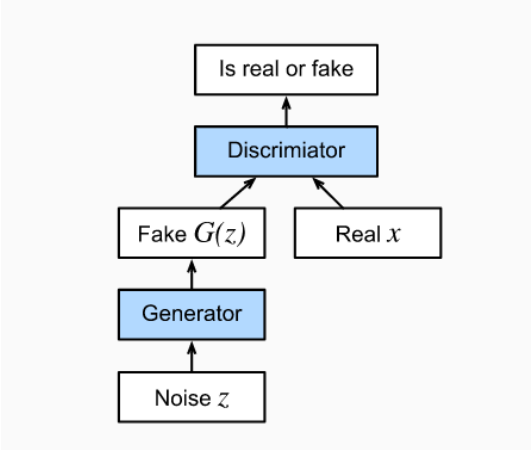


图 1: GAN model

如图，有两个网络 G(Generator) 和 D(Discriminator)。G 网络接受一个随机噪声 z，以此生成图片 G(z)，即其输出。而 D 网络接受 G(z) 及真实的图片数据，它需要判断这个输入是 True(真实图片) 还是 False(G(z))；D 网络的输出 D(x) 代表 x 为真实图片的概率，1 代表 100% 真实，0 代表是真实图片的可能性为 0%。

1.2 代码框架

GAN 的代码主要分成三个部分：数据获取及预处理，网络构建，训练及输出。

1.2.1 数据获取及预处理

在数据预处理方面，利用 transform 包中的 Resize 函数将其统一成 64×64，随机选取一张图片进行输出，结果显示合理。

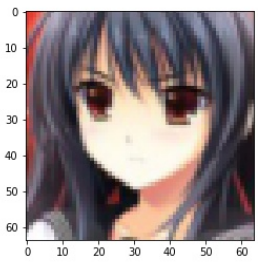


图 2: random xjj

1.2.2 网络构建

网络的构建参考了《DIVE INTO DEEP LEARNING》中的 GAN 结构，以下将从 Generator 和 Discriminator 两部分分别说明其架构。

Generator

下图为 G 网络的模型图。Generator 共有 5 个 stage，前 4 个 stage 分别由一个 ConvTranspose2d 层，一个 Batchnorm 层和一个 ReLU 层组成，只是输入输出的 channel 数不同，最后一个 stage 则是由一个 ConvTranspose2d 层和一个 Tanh 层组成。

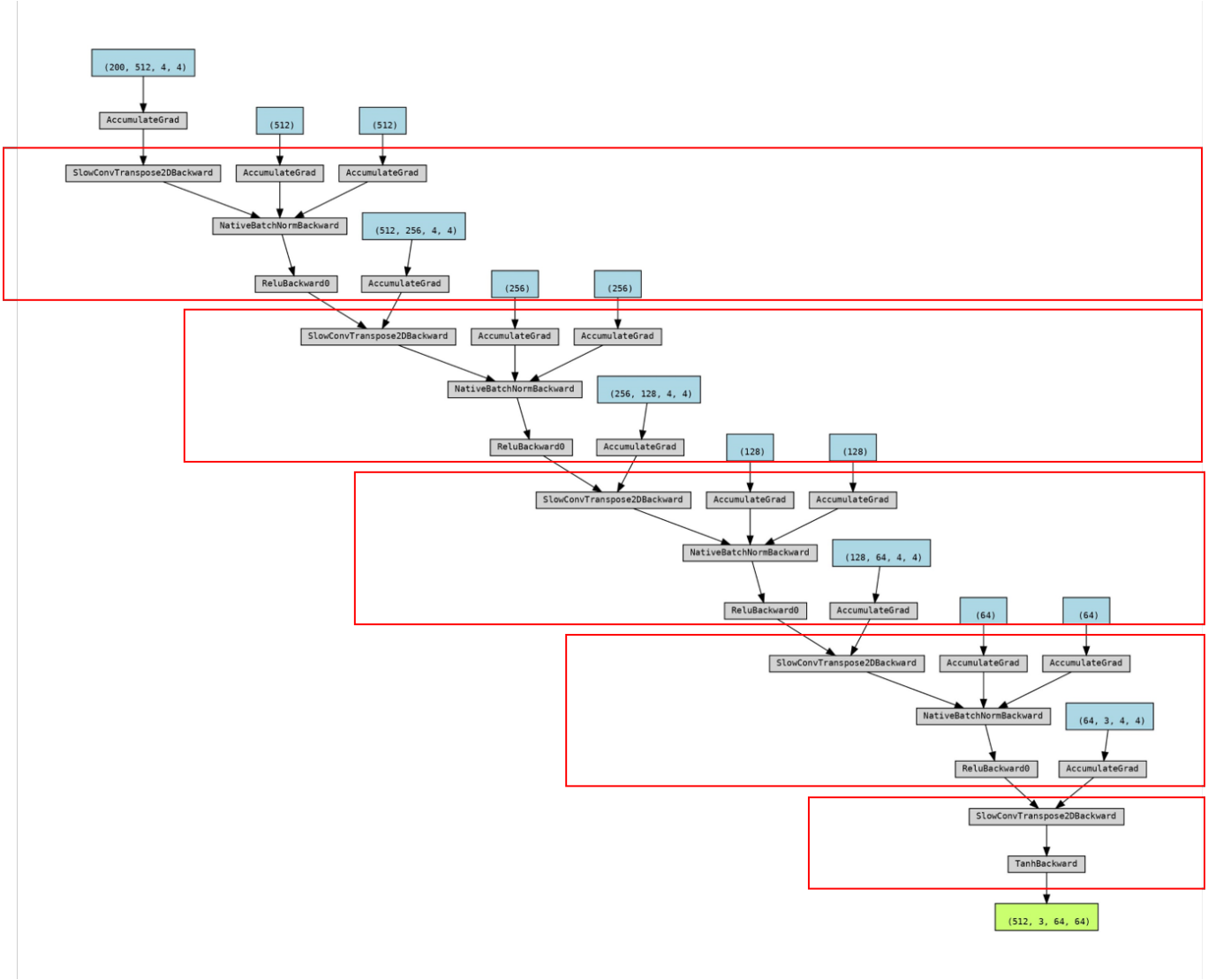


图 3: GAN_G

最终输出的是一个 $[512, 3, 64, 64]$ 的四维数据，即 batchsize 为 512 的三通道 64×64 的图片。

Generator 的 loss function 如下，其中 fake_y 是 Discriminator 对 Generator 产生的图片 x 的输出结果，ones 是一组全 1 向量。

```
loss_G = loss(fake_Y, ones)
```

optimizer 采用的是 torch.optim 包中的 SGD 函数。

```
trainer_G = torch.optim.SGD(net_G.parameters(), lr=lr)
```

Discriminator

下图为 D 网络的模型图。Discriminator 也是由 5 个 stage 组成，但不同于 Genrerator，前四个 stage 分别由一个 Conv2d 层，一个 Batchnorm 层和一个 ReLU 层组成，最后一个 stage 则是一个 Conv2d 层。

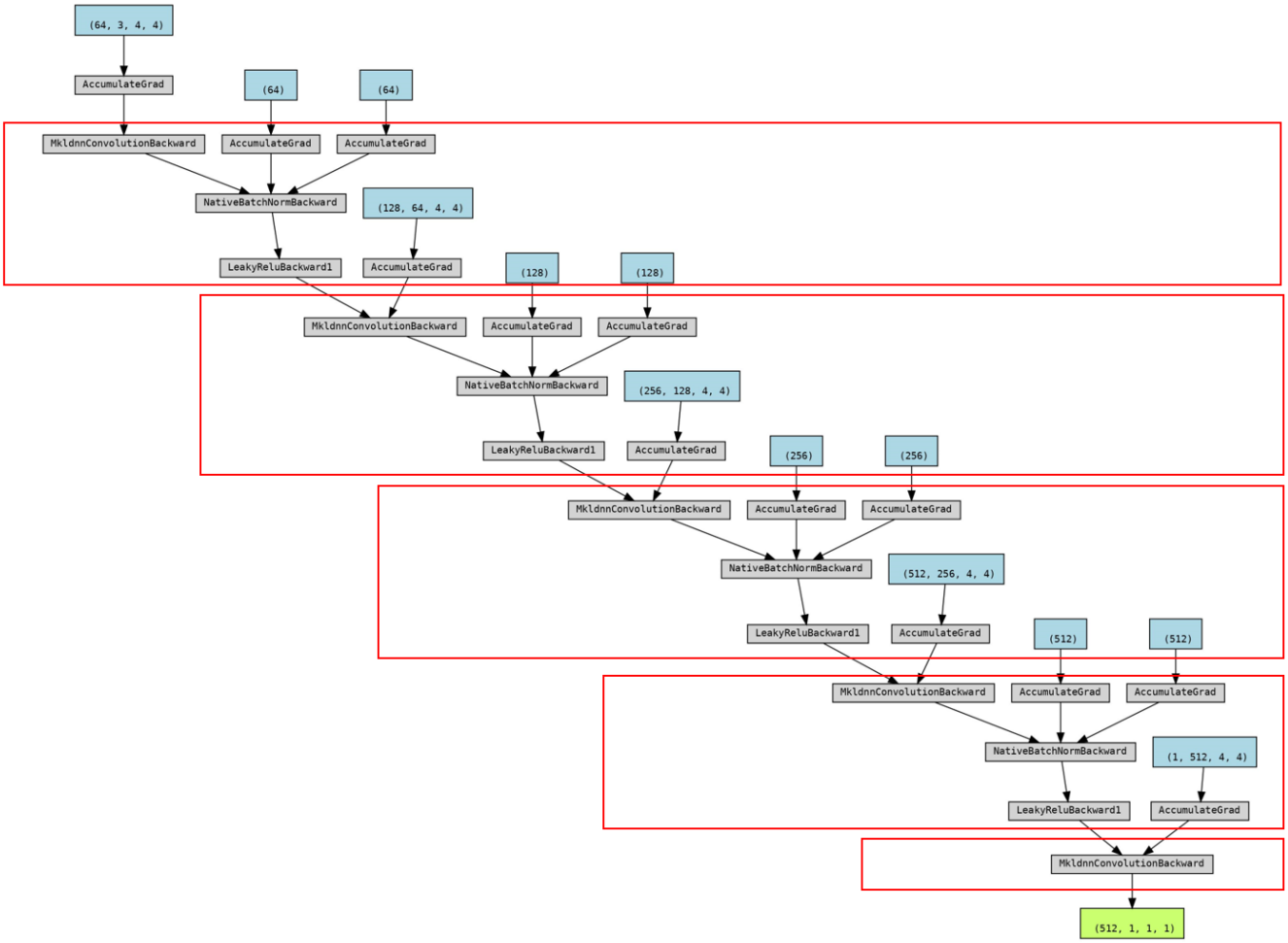


图 4: GAN_D

Generator 的 loss function 如下,ones 和 zeros 分别对应一组全 1 向量和一组全 0 向量。

```
real_Y = net_D(X)
fake_X = net_G(Z)
fake_Y = net_D(fake_X.detach())
loss_D = (loss(real_Y, ones) + loss(fake_Y, zeros)) / 2
```

optimizer 采用的是 torch.optim 包中的 SGD 函数。

```
trainer_D = torch.optim.SGD(net_D.parameters(), lr=lr)
```

1.2.3 参数说明

表 1: GAN parameter

parameter	Value	Meaning
optimizer	SGD(lr=0.005)	
loss function	BCEWithLogitsLoss(reduction='sum')	针对多分类问题, 把 Sigmoid 和 BCE-Loss 合成一步
latent dimension	200	传入 G 的随机向量的维度
activation function	G:ReLU，D:LeakyReLU	
batch size	512	
epochs	30	

1.3 训练结果

训练 30 轮，将 losses_D 和 losses_G 的变化情况绘制再同一张图中比较，易知 D 的 loss 按照期望在不断下降，而 G 的 loss 则逐渐上升。

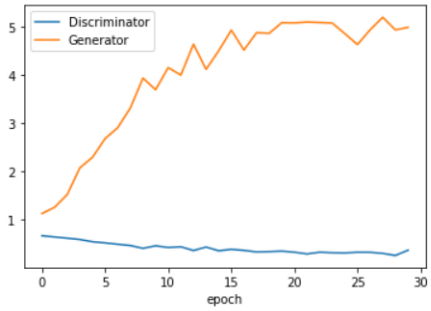
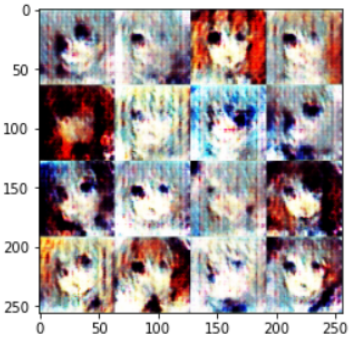
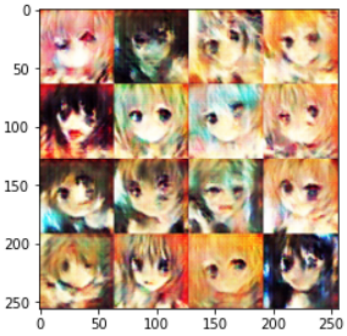


图 5: GAN Result

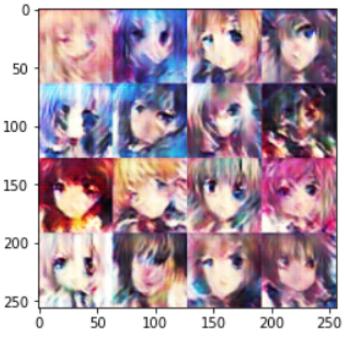
总体的训练情况较好，生成的图像也随着训练轮次的增加而越来越清晰。以下展示部分轮次的训练结果。



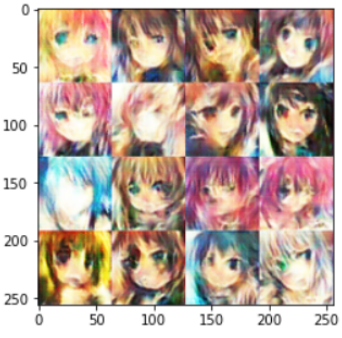
(a) epoch 10



(b) epoch 22



(c) epoch 26



(d) epoch 30

图 6: GAN Output

2 WGAN-GP model

2.1 原理

WGAN-GP 是 WGAN 的改进，相比于一般的 GAN，区别如下：

- Discriminator 最后一层去掉 Sigmoid；
- loss function 改变
- 每次更新 Discriminator 参数后，截断其绝对值以满足 lipschitz 连续
- optimizer 改变

2.2 代码框架

2.2.1 网络构建

网络的构建参考了 github 上相关文档，以下将从 Generator 和 Discriminator 两部分分别说明其架构。

Generator

与 GAN 相同，Generator 共有 5 个 stage，前 4 个 stage 分别由一个 ConvTranspose2d 层，一个 Batchnorm 层和一个 ReLU 层组成，只是输入输出的 channel 数不同，最后一个 stage 则是由一个 ConvTranspose2d 层和一个 Tanh 层组成。

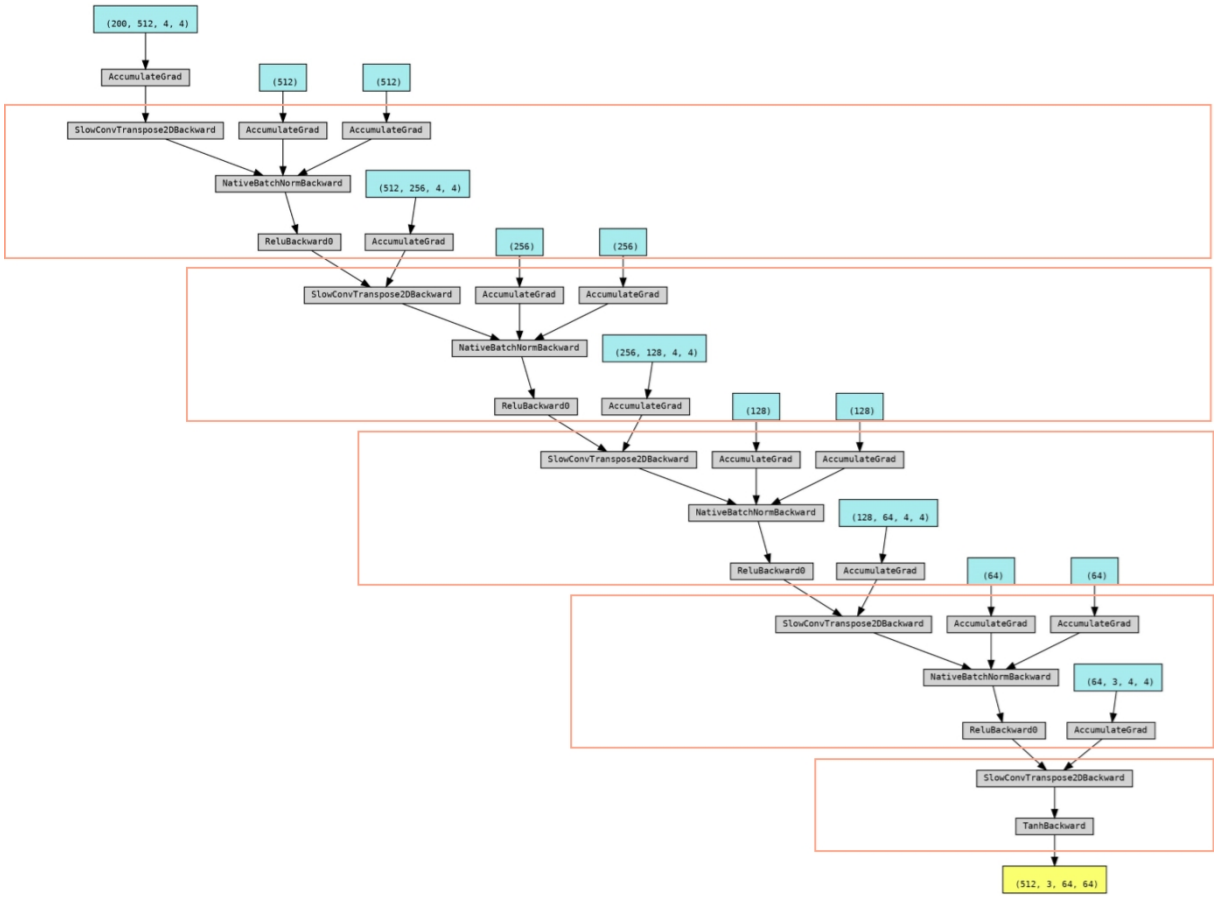


图 7: WGAN-GP_G

最终输出的依旧是一个 $[512, 3, 64, 64]$ 的四维数据。

Generator 的 loss function 如下，其中 fake_y 是 Discriminator 对 Generator 产生的图片 x 的输出结果，ones 是一组全 1 向量。

```
loss_G = loss(fake_Y, ones.reshape(fake_Y.shape))
```

optimizer 采用的是 torch.optim 包中的 Adam 函数。

```
trainer_hp = {'lr': lr, 'betas': [0.5, 0.999]}
trainer_G = torch.optim.Adam(net_G.parameters(), **trainer_hp)
```

Discriminator

下图为 D 网络的模型图。Discriminator 也是由 5 个 stage 组成，但不同于 Genrerator，前四个 stage 分别由一个 Conv2d 层，一个 Batchnorm 层和一个 ReLU 层组成，最后一个 stage 则是一个 Conv2d 层。

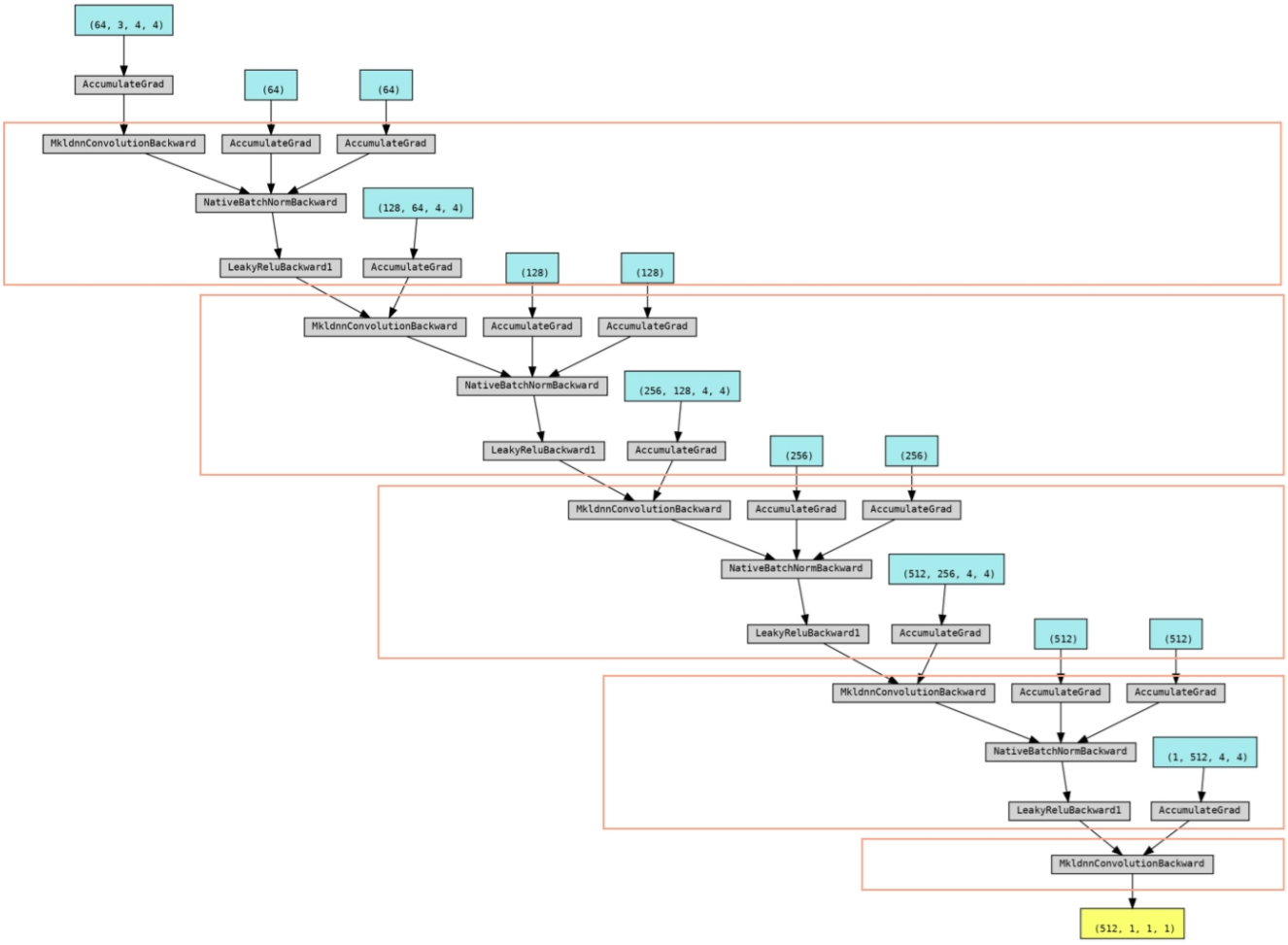


图 8: WGAN-GP_D

Generator 的 loss function 如下, 与 GAN 的区别在于增加了梯度惩罚 gradient penalty。

```
real_Y = net_D(X)
fake_X = net_G(Z)
fake_Y = net_D(fake_X.detach())
loss_D = (loss(real_Y, ones) + loss(fake_Y, zeros)) / 2
loss_D = loss_D + compute_gradient_penalty(X, fake_X.detach(), net_D)
```

optimizer 采用的是 torch.optim 包中的 Adam 函数。

```
trainer_hp = {'lr': lr, 'betas': [0.5, 0.999]}
trainer_D = torch.optim.Adam(net_D.parameters(), **trainer_hp)
```

2.2.2 参数说明

表 2: GAN parameter

parameter	Value	Meaning
optimizer	Adam(lr=0.001)	
loss function	gradient penalty	
latent dimension	200	传入 G 的随机向量的维度
activation function	G:ReLU，D:LeakyReLU	
batch size	512	
epochs	30	

2.3 训练结果

训练 30 轮，将 losses_D 和 losses_G 的变化情况绘制再同一张图中比较，易知 D 的 loss 按照期望在不断下降，而 G 的 loss 则逐渐上升。

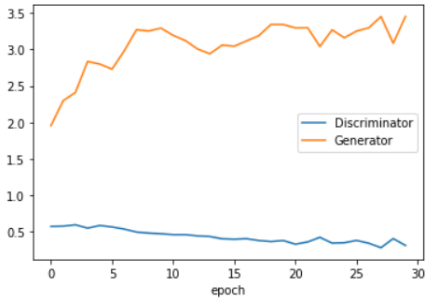
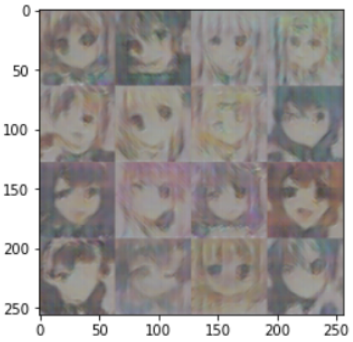
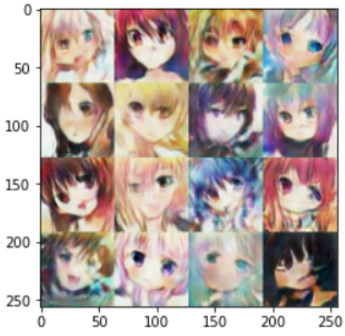


图 9: WGAN-GP Result

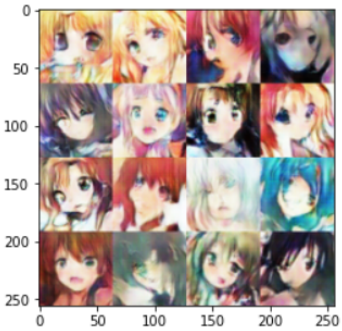
以下展示部分轮次的训练结果。



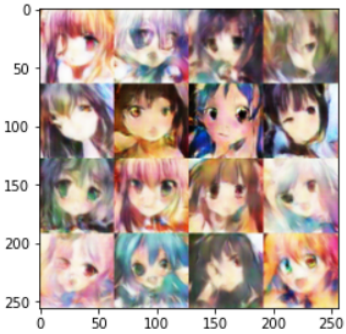
(a) epoch 10



(b) epoch 22



(c) epoch 26



(d) epoch 30

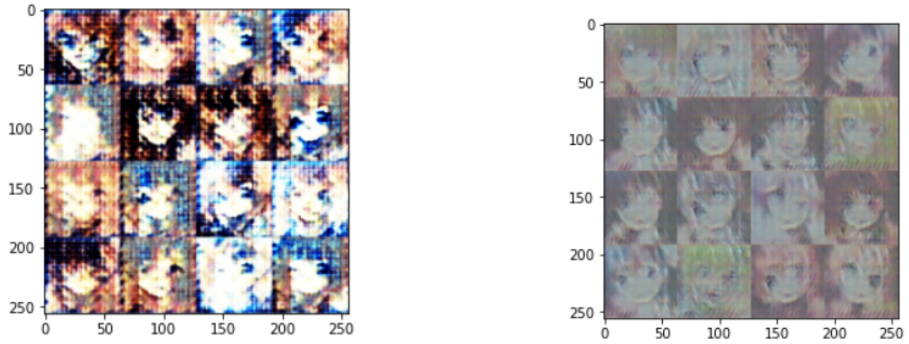
图 10: WGAN-GP Output

3 简简简要分析

将 GAN 和 WGAN-GP 的同一轮次训练结果进行比较，GAN 在前 10 轮仍在不成样的图片中挣扎，但 WGAN-GP 在第六轮就显现出人眼能分辨的人像了，唯一的不足在于灰度值较高。这一结果也符合两个模型的 loss 曲线所呈现的，WGAN 的 G-loss 在 epoch6 左右就达到较高的值且之后的上升比较缓慢；而 GAN 的 G-loss 在这一轮次周围仍然在快速的变化中。

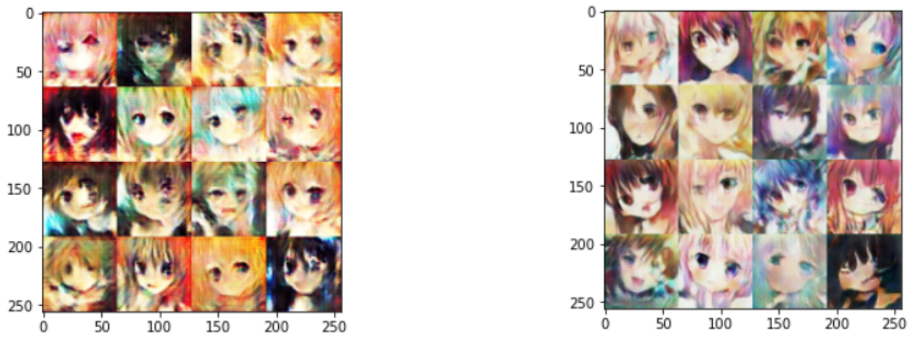
在 20 轮后，两种模型生成的图片均开始变得明亮合理，但 GAN 生成的图片仍然具有特定的色彩分布，如 epoch22，整体的色彩是橘色调，而 WGAN-GP 则更为丰富。

因此，无论是在图片噪点多少还是色彩方面，GWAN-GP 生成的图片质量明显高于 GAN。此外，在跑程序的时候，也能明显地感知到 WGAN 的速度快于 GAN。



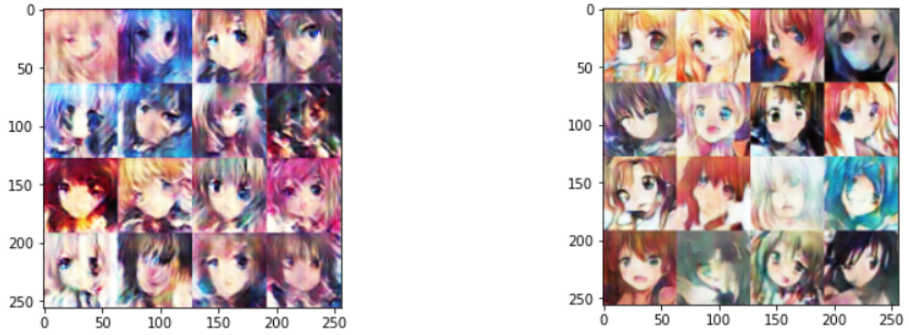
(a) GAN (b) WGAN-GP

图 11: epoch 6



(a) GAN (b) WGAN-GP

图 12: epoch 22



(a) GAN (b) WGAN-GP

图 13: epoch 26

4 参考资料

[1]www.d2l.ai

[2]https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/wgan_gp