GAN

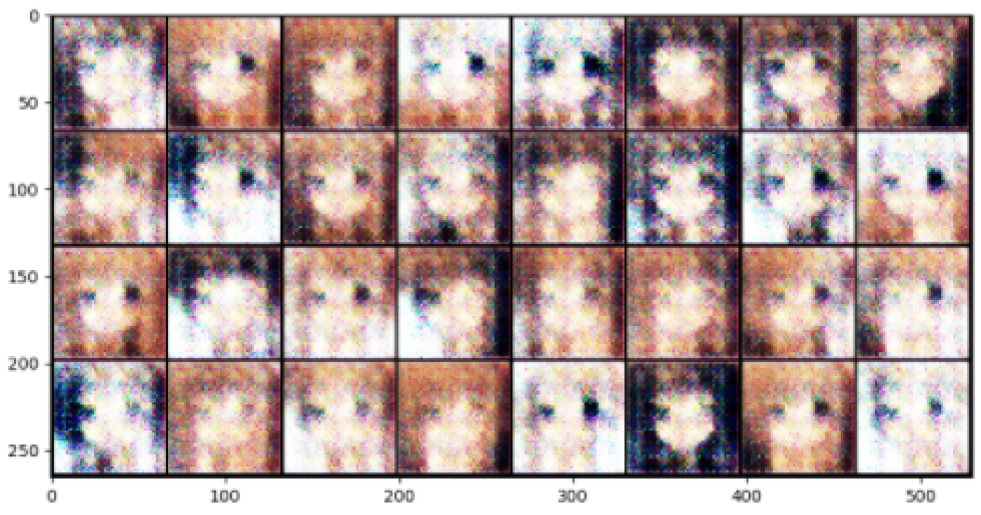
陈乐偲

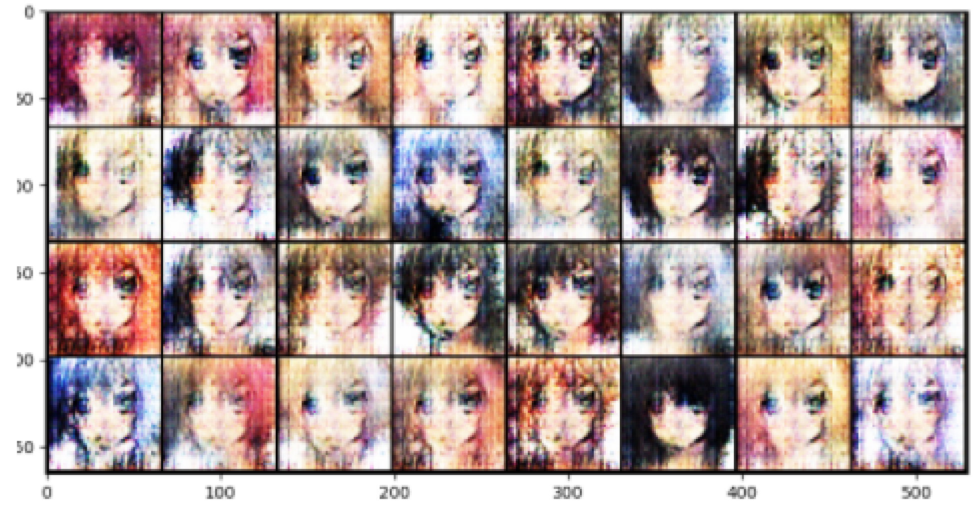
基于GAN的动漫头像生成

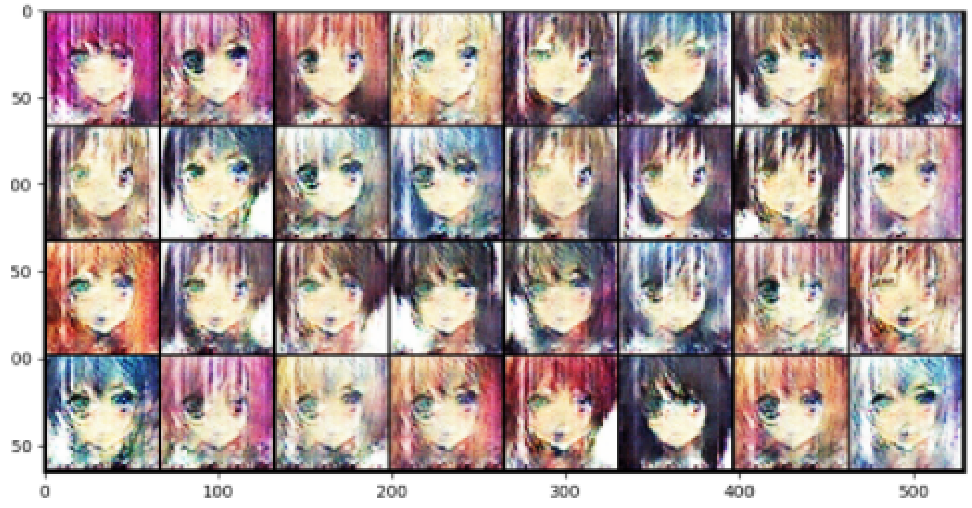
GAN 由两个网络组成，一个用于生成，一个用于判别。

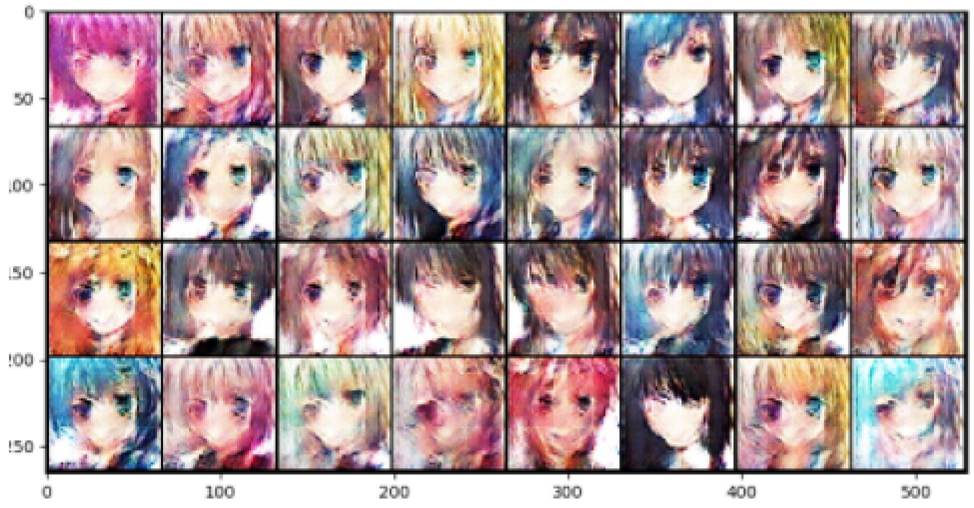
```
def Conv(n_input, n_output, k_size=4, stride=2, padding=0, bn=False):
    return nn.Sequential(
        nn.Conv2d(n_input, n_output, kernel_size=k_size, stride=stride, padding=padding,
bias=False),
        nn.BatchNorm2d(n_output),
        nn.LeakyReLU(0.2, inplace=True),
        nn.Dropout(p=0.2, inplace=False))

#使用反卷积做上采样
def Deconv(n_input, n_output, k_size=4, stride=2, padding=1):
    return nn.Sequential(
        nn.ConvTranspose2d(n_input, n_output,kernel_size=k_size,stride=stride,
padding=padding,bias=False),
        nn.BatchNorm2d(n_output),
        nn.ReLU(inplace=True))

class Generator(nn.Module):
    def __init__(self, z=100, nc=64):
        super(Generator, self).__init__()
        self.net = nn.Sequential(
            Deconv(z, nc*8, 4,1,0),
            Deconv(nc*8, nc*4, 4,2,1),
            Deconv(nc*4, nc*2, 4,2,1),
            Deconv(nc*2, nc, 4,2,1),
            nn.ConvTranspose2d(nc,3, 4,2,1,bias=False),
            nn.Tanh()
        )

    def forward(self, input):
        return self.net(input)

class Discriminator(nn.Module):
    def __init__(self, nc=64):
        super(Discriminator, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, nc,kernel_size=4,stride=2,padding=1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            Conv(nc, nc*2, 4,2,1),
            Conv(nc*2, nc*4, 4,2,1),
            Conv(nc*4, nc*8, 4,2,1),
            nn.Conv2d(nc*8, 1,4,1,0, bias=False),
            nn.Sigmoid())

    def forward(self, input):
        #print(input.shape)
        return self.net(input)

class GANModel():
    def __init__(self):
        self.dis_model = Discriminator().to(device)
        self.gen_model = Generator().to(device)
        self.real_label = 1.
        self.fake_label = 0.
        self.lrD = 2e-4
        self.lrG = 2e-4

        self.criterion = nn.BCELoss()
        self.optim_D = optim.Adam(self.dis_model.parameters(), lr=self.lrD, betas=(0.5, 0.999))
        self.optim_G = optim.Adam(self.gen_model.parameters(), lr=self.lrG, betas=(0.5, 0.999))

        self.epoches = 30
        self.fixed_noise = torch.randn(32, 100, 1,1, device=device)
        self.update = 2

        #writer.add_graph(model=self.dis_model,
input_to_model=torch.randn([128,3,64,64]).to(device))
        #writer.add_graph(model=self.gen_model, input_to_model=self.fixed_noise)

    def train(self, dataloader):

        for epoch in range(self.epoches):
            for i, data in enumerate(dataloader):

                #训练判别模型
                self.dis_model.zero_grad()

                #在正样例上训练
                real_img = data[0].to(device)
                B = real_img.shape[0]
                label = Uniform(0.95, 1.0).sample((B,)).to(device)
                #label = torch.full((B,), self.real_label, device=device, dtype=torch.float)
                output = self.dis_model(real_img).view(-1)
                loss_real = self.criterion(output, label)
                loss_real.backward()

                #在负样例上训练
                noise = torch.randn(B, 100, 1,1, device=device)
                fake_img = self.gen_model(noise)
                label = Uniform(0., 0.05).sample((B,)).to(device)
                #label = torch.full((B,), self.fake_label, device=device, dtype=torch.float)
                output = self.dis_model(fake_img.detach()).view(-1)
                loss_fake = self.criterion(output, label)
                loss_fake.backward()

                #训练判别器
                loss_dis = loss_real + loss_fake
                #loss_dis.backward()
                self.optim_D.step()

                #训练生成器
                self.gen_model.zero_grad()
                output = self.dis_model(fake_img).view(-1)
                label = Uniform(0.95, 1.0).sample((B,)).to(device)
                #label = torch.full((B,), self.real_label, device=device, dtype=torch.float)
                loss_gen = self.criterion(output, label)
                loss_gen.backward()
                self.optim_G.step()

                if i % 300 == 0:
                    print('epoch:',epoch,'loss_dis:',loss_dis.item(), 'loss_gen:',
loss_gen.item())

            with torch.no_grad():
                fake_img = self.gen_model(self.fixed_noise).detach().cpu()

                fake_img = make_grid(fake_img, padding=2, normalize=True)
                fig =plt.figure(figsize=(10,10))
                plt.imshow(fake_img.permute(1,2,0))
                plt.savefig('./result/' + str(epoch) + '.png')
                #writer.add_figure(tag='gen_results', figure=fig)


img_size = 64
data_transforms = T.Compose([
    T.Resize(img_size),
    T.CenterCrop(img_size),
    T.ToTensor(),
    T.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

dataset = torch_dataset.ImageFolder(root='./data/', transform=data_transforms)
dataloader = DataLoader(dataset=dataset, batch_size=128, shuffle=True, num_workers=4)
GAN = GANModel()
GAN.train(dataloader)
```

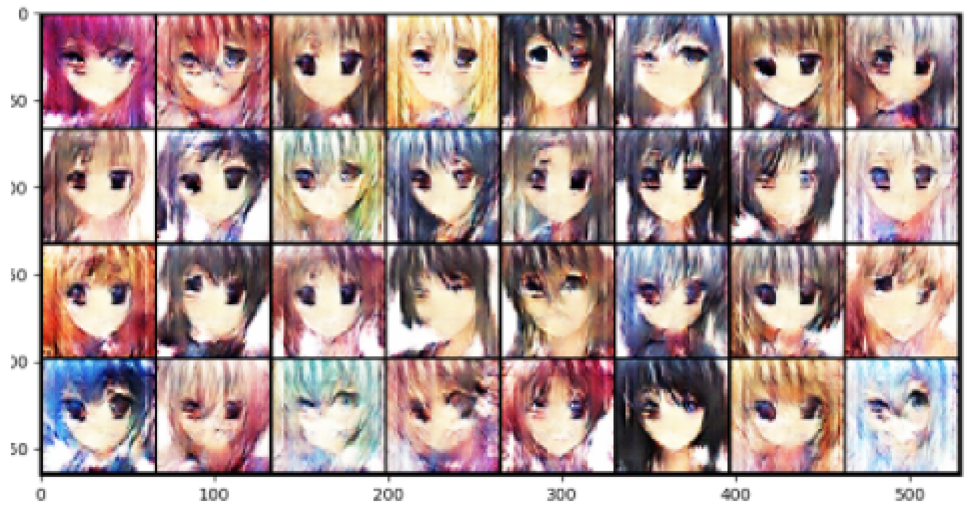
生成结果如下：

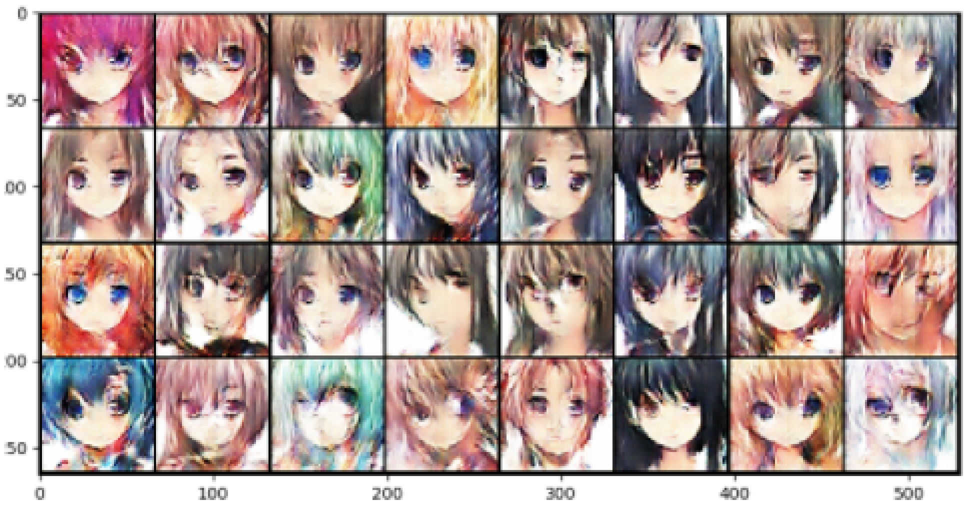


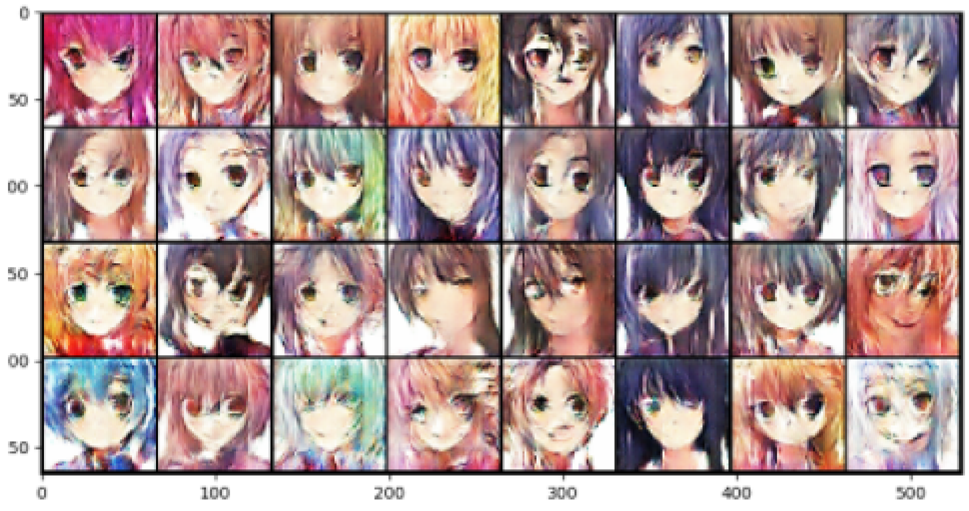

0
50
0
100
200
300
400
500


0
50
00
50
00
50
0
100
200
300
400
500


0
50
0
100
200
300
400
500


0
50
00
50
00
50
0
100
200
300
400
500


0
50
00
50
00
50
0
100
200
300
400
500


0
50
00
50
00
50
0
100
200
300
400
500