Lab6

0856108 謝宗祐

1. Introduction

   In this lab, I have to implement the conditional GAN and generate synthetic images based on multi-labels conditions. The architecture of the conditional GAN and other details are all decided by myself, and the details are clearly introduced in the below sections.

2. Implementation details

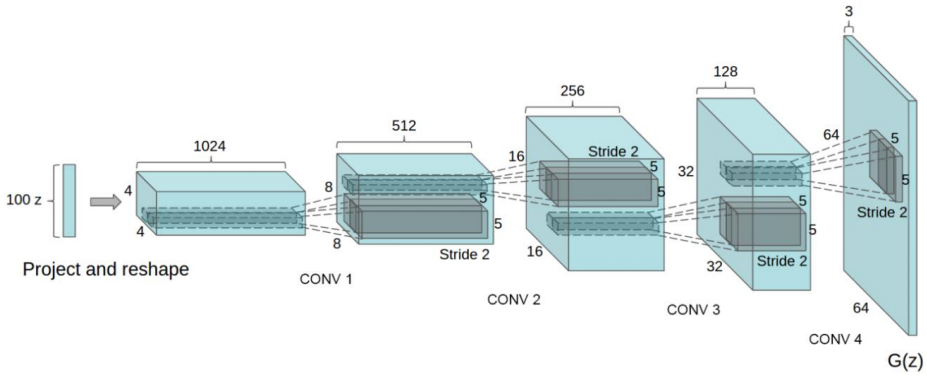
   A. Describe how you implement your model, including your choice of cGAN, model architectures, and loss functions.

      I. Choice of cGAN

         I choose conditional GAN as my cGAN. For the generator, I first use a linear layer to map the one-hot labels to the dimension of latent space, and then I use multiplication to combine the latent and labels together. For the discriminator, I also map the labels to the same dimension of images (64x64), and then simply concatenate images and labels together (bs, 4, 64, 64). The labels can be seen as a new channel for images.

      II. Model architecture

         For the GAN model architecture, I use Deep convolutional GAN.



The number in the above figure is for reference only.

```
class Generator(nn.Module):
    def __init__(self, in_dim, dim=64):
        super(Generator, self).__init__()
        self.label_embed = nn.Linear(24, in_dim)
        def dconv_bn_relu(in_dim, out_dim):
            return nn.Sequential(
                nn.ConvTranspose2d(in_dim, out_dim, 5, 2,
                                   padding=2, output_padding=1, bias=False),
                nn.BatchNorm2d(out_dim),
                nn.ReLU())

        self.l1 = nn.Sequential(
            nn.Linear(in_dim, dim * 8 * 4 * 4, bias=False),
            nn.BatchNorm1d(dim * 8 * 4 * 4),
            nn.ReLU())

        self.l2_5 = nn.Sequential(
            dconv_bn_relu(dim * 8, dim * 4),
            dconv_bn_relu(dim * 4, dim * 2),
            dconv_bn_relu(dim * 2, dim),
            nn.ConvTranspose2d(dim, 3, 5, 2, padding=2, output_padding=1),
            nn.Tanh())
```

```
class Discriminator(nn.Module):
    def __init__(self, in_dim, dim=64):
        super(Discriminator, self).__init__()
        self.label_embed = nn.Linear(24, dim*dim)
        def conv_bn_lrelu(in_dim, out_dim):
            return nn.Sequential(
                nn.Conv2d(in_dim, out_dim, 5, 2, 2),
                nn.BatchNorm2d(out_dim),
                nn.LeakyReLU(0.2),
                )

        self.ls = nn.Sequential(
            nn.Conv2d(in_dim, dim, 5, 2, 2), nn.LeakyReLU(0.2),
            conv_bn_lrelu(dim, dim * 2),
            conv_bn_lrelu(dim * 2, dim * 4),
            conv_bn_lrelu(dim * 4, dim * 8),
            nn.Conv2d(dim * 8, 1, 4),
            nn.sigmoid()
            )
```

III. Loss function

Choose the loss function used in CGAN.

CGAN	Arxiv	$L_D^{CGAN} = E\left[\log\left(D(x,c)\right)\right] + E\left[\log\left(1 - D(G(z),c)\right)\right]$ $L_G^{CGAN} = E\left[\log\left(D(G(z),c)\right)\right]$

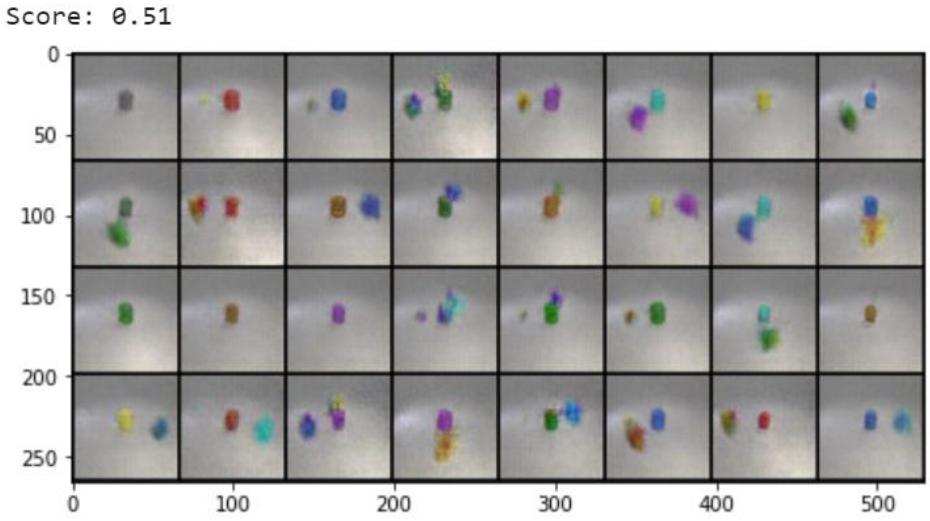
B. Specify the hyperparameters

```
# hyperparameters
batch_size = 64
z_dim = 128
lr = 1e-4
n_epoch = 100
```

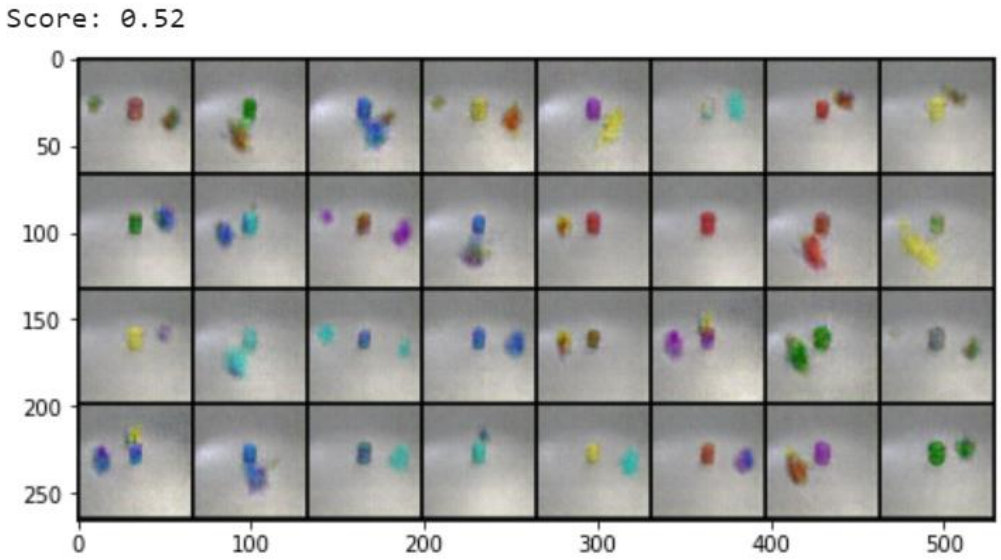
```
# loss criterion
criterion = nn.BCELoss()

# optimizer
opt_D = torch.optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))
opt_G = torch.optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
```

3. Results and discussion

A. Show your results based on the testing data



Because I have the late submission for this lab, I also show the result of new test data below.



B. Discuss the results of different models architectures

I. At first, I found that the discriminator loss is so small (close to zero), so I guess the discriminator is too strong. Therefore, I updates the discriminator for every 6 batch iteration, and updates the generator

for every batch iteration. This procedure truly leads to the better results than the original training procedures (train G and D for every batch iteration). The score improves from 0.39 to 0.51.

II. I google for the training tips for GAN, and I try so many methods to improve my cGAN, such as label smoothing, noisy labels, add batchnormalized layer, replace the ReLU to leaky ReLU. By lots of experiments, the last two approaches(BN, leaky ReLU) is helpful for my model.