Deep Learning Homework 3

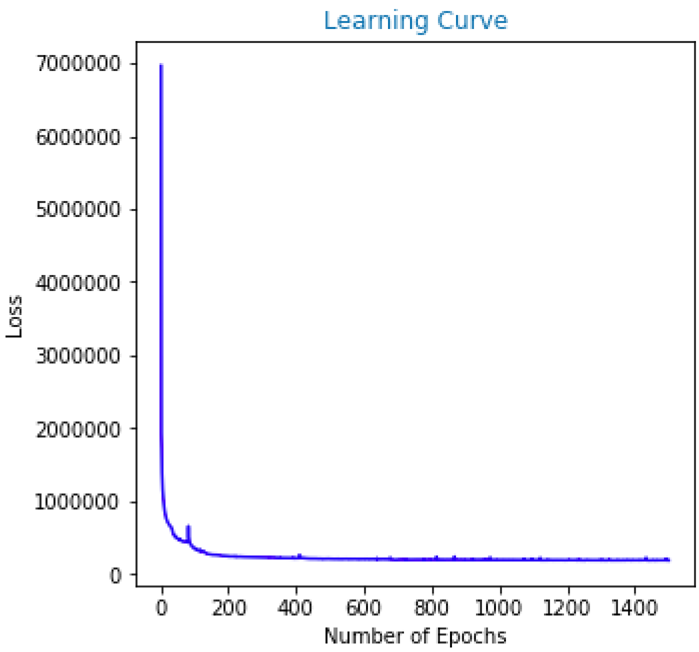
Student ID: 0416106 **Name**: 彭敬樺

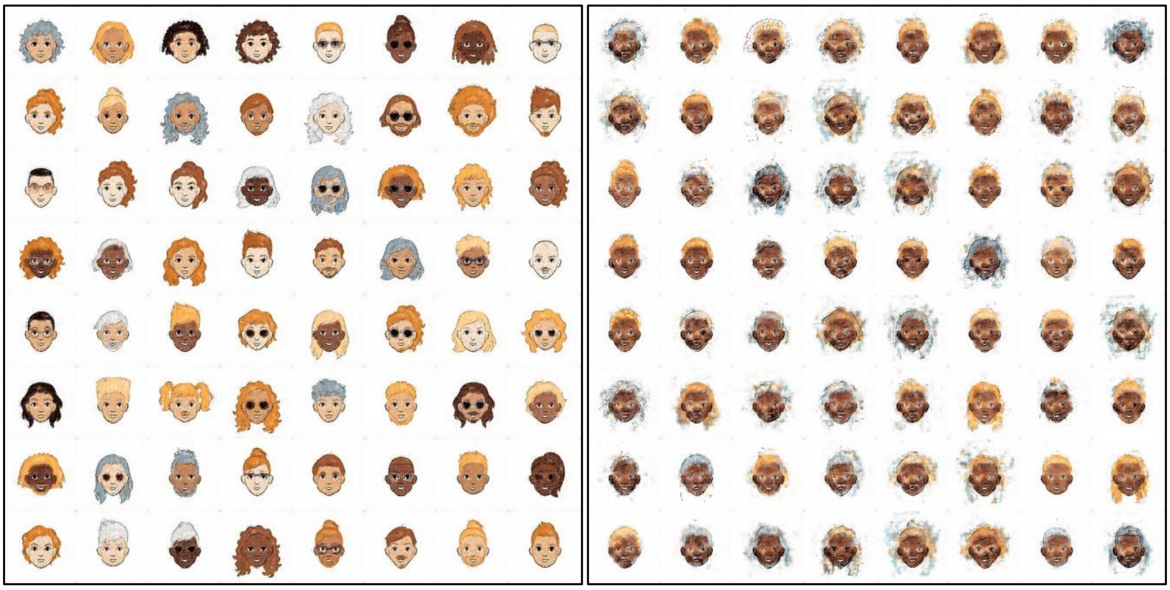
1.

```
train_transform = transforms.Compose(
    [transforms.Resize((INPUT_SIZE,INPUT_SIZE)),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
train_data = torchvision.datasets.ImageFolder(
    root = train_data_path,
    transform = train_transform
    )
trainloader = torch.utils.data.DataLoader(train_data, batch_size=BATCH_SIZE,
                                          shuffle=True, num_workers=4)
```

```
import torch.nn as nn
import torch.nn.functional as F

class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.conv1 = nn.Conv2d(3, 6, 7, padding = 3)
        self.pool1 = nn.MaxPool2d(4, 4, return_indices=True)
        self.conv2 = nn.Conv2d(6, 16, 5, padding = 2)
        self.pool2 = nn.MaxPool2d(2, 2, return_indices=True)
        self.conv3 = nn.Conv2d(16, 32, 3, padding = 1)
        self.pool3 = nn.MaxPool2d(2, 2, return_indices=True)
        self.fc1 = nn.Linear(32 * 16 * 16, 2048)
        self.fc2 = nn.Linear(2048, 512)
        self.fc3 = nn.Linear(512, 50)

        self.indices1 = 0
        self.indices2 = 0
        self.indices3 = 0

        self.rconv1 = nn.ConvTranspose2d(6, 3, 7, padding = 3)
        self.rpool1 = nn.MaxUnpool2d(4, 4)
        self.rconv2 = nn.ConvTranspose2d(16, 6, 5, padding = 2)
        self.rpool2 = nn.MaxUnpool2d(2, 2)
        self.rconv3 = nn.ConvTranspose2d(32, 16, 3, padding = 1)
        self.rpool3 = nn.MaxUnpool2d(2, 2)
        self.rfc1 = nn.Linear(2048, 32 * 16 * 16)
        self.rfc2 = nn.Linear(512, 2048)
        self.rfc3 = nn.Linear(50, 512)
```

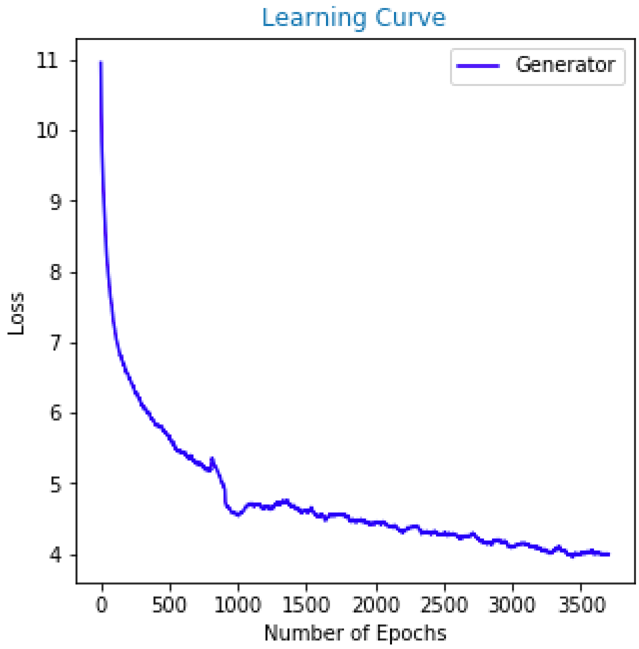
```python
    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std

    def encode(self, x): #3, 256, 256
        x = F.relu(self.conv1(x)) #6, 256, 256
        x, self.indices1 = self.pool1(x) #6, 64, 64
        x = F.relu(self.conv2(x)) #16, 64, 64
        x, self.indices2 = self.pool2(x) #16, 32, 32
        x = F.relu(self.conv3(x)) #32, 32, 32
        x, self.indices3 = self.pool3(x) #32, 16, 16
        x = x.view(-1, 32 * 16 * 16)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        mean = self.fc3(x)
        var = self.fc3(x)
        return mean, var

    def decode(self, x):
        x = F.relu(self.rfc3(x))
        x = F.relu(self.rfc2(x))
        x = F.relu(self.rfc1(x))
        x = x.view(x.size(0), 32, 16, 16)
        x = self.rpool3(x, self.indices3)
        x = F.relu(self.rconv3(x))
        x = self.rpool2(x, self.indices2)
        x = F.relu(self.rconv2(x))
        x = self.rpool1(x, self.indices1)
        x = self.rconv1(x)
        return x

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
```

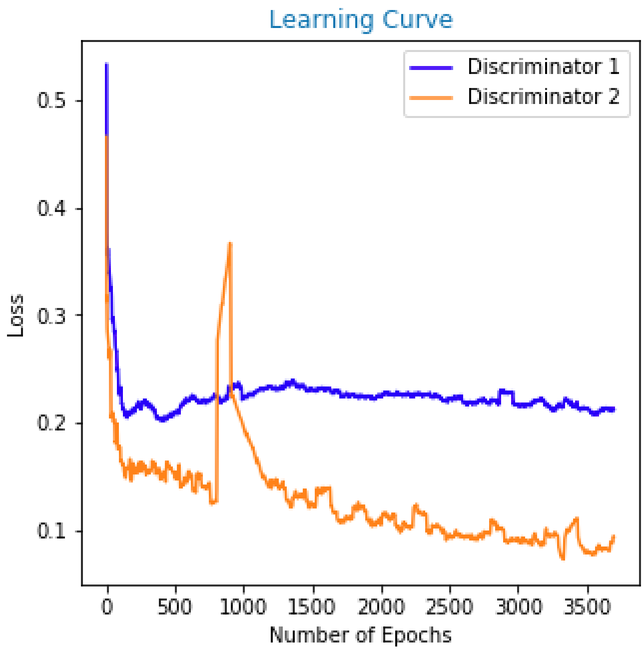




Left: reconstructed image. Right: prior sample

2.


Learning Curve
Generator
Loss
11
10
9
8
7
6
5
4
0
500
1000
1500
2000
2500
3000
3500
Number of Epochs


Learning Curve
Discriminator 1
Discriminator 2
Loss
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500
3000
3500
Number of Epochs

Mode collapse is usually referred to a problem when all the generator outputs are identical (all of them or most of the samples are equal). In the real world, distributions are complicated and multimodal, for example, the probability distribution which describes data may have multiple “peaks” where different sub-groups of samples are concentrated. In such a case a generator can learn to yield images only from one of the sub-groups, causing mode collapse.

In this model, this should not be serious issue. The reason is as following. Cycle consistency can be viewed as a form of regularization. By enforcing cycle consistency, CycleGAN framework prevents generators from excessive hallucinations and mode collapse, both of which will cause unnecessary loss of information and thus increase in cycle consistency loss.