Deep learning Assignment 2

Optimizer

```
lr = 0.0002
batch_size = 128
criterion = torch.nn.BCELoss()
g_optimizer = torch.optim.Adam(generator.parameters(), lr=lr)
d_optimizer = torch.optim.Adam(discriminator.parameters(), lr=lr)
```

learning rate는 0.0002, batch size는 128로 설정하였다. optimizer은 Adam optimizer을 선택하였고, loss function은 Binary cross entropy loss를 사용하였다.

Model 1

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

      self.dropout = 0.3
      self.image_size = 64

      self.layer1 = nn.Sequential(
          nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5),
          nn.BatchNorm2d(6),
          nn.LeakyReLU(0.2, inplace=True),
          nn.MaxPool2d(2),
          nn.Dropout(self.dropout))

      self.image_size = int(((self.image_size - 5) + 1) / 2)

      self.layer2 = nn.Sequential(
          nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5),
          nn.BatchNorm2d(16),
          nn.LeakyReLU(0.2, inplace=True),
          nn.MaxPool2d(2),
          nn.Dropout(self.dropout))

      self.image_size = int(((self.image_size - 5) + 1) / 2)

      self.layer3 = nn.Sequential(
          nn.Linear(in_features= 16 * self.image_size * self.image_size, out_features=1024, bias=True),
          nn.BatchNorm1d(1024),
          nn.LeakyReLU(0.2, inplace=True),
          nn.Dropout(self.dropout))

      self.layer4 = nn.Sequential(
          nn.Linear(in_features=1024, out_features=512, bias=True),
          nn.BatchNorm1d(512),
          nn.LeakyReLU(0.2, inplace=True),
          nn.Dropout(self.dropout))

      self.layer5 = nn.Sequential(
          nn.Linear(in_features=512, out_features=256, bias=True),
          nn.BatchNorm1d(256),
          nn.LeakyReLU(0.2, inplace=True),
          nn.Dropout(self.dropout))
```

```
      self.layer6 = nn.Sequential(
          nn.Linear(in_features= 256, out_features=1, bias=True),
          nn.Sigmoid(),)

    def forward(self, input):
      input = self.layer1(input)
      input = self.layer2(input)
      input = input.view(-1, 16 * self.image_size * self.image_size)
      input = self.layer3(input)
      input = self.layer4(input)
      input = self.layer5(input)
      input = self.layer6(input)
      return input
```

먼저 다른 실험에서 썼던 모델을 그대로 들고와봤다. 악성코드를 이미지로 표현하여 악성코드인지 아닌지를 분석하는 binary classifier이다.

Discriminator 또한 진짜인지 가짜인지 판별해주는 binary classifier이기에 여기에 활용될 수 있다고 생각했다.

Convolution layer는 2겹을 썼고 각 convolution 에는 batch normalization과 max pooling이 적용되었다.

LeakyReLU는 0.2로 조교님께서 실습시간에 잘 된다던 parameter를 주었고 dropout은 0.3로 통일하여 주었다

Linear layer는 4겹을 썼다. 각 layer은 batch normalization과 마찬가지로 LeakyReLU 0.2가 적용되었다.

마지막 레이어는 빼고 dropout 0.3을 주었다.

```
100%|██████████| 1583/1583 [12:19<00:00,  2.14it/s]
100%|██████████| 24/24 [00:11<00:00,  2.17it/s]
fid score : 240.9550539584593
```

fid score은 240으로 성능이 좋지 않았다.

fake이미지는 실제와 같지 않았다.

fid score가 너무 높았기에 이건 parameter 조절의 문제가 아니라, 모델 구조 자체의 문제가 있다고 판단하였다.

Model 2

```
class Discriminator(nn.Module):
    def __init__(self, d=64):
      super(Discriminator, self).__init__()
```

```
      self.layer1 = nn.Sequential(
          nn.Conv2d(in_channels=3, out_channels=d, kernel_size=4, stride=2, padding=1),
          nn.BatchNorm2d(d),
          nn.LeakyReLU(0.2, inplace=True),)


      self.layer2 = nn.Sequential(
          nn.Conv2d(in_channels=d, out_channels=d*2, kernel_size=4, stride=2, padding=1),
          nn.BatchNorm2d(d*2),
          nn.LeakyReLU(0.2, inplace=True),)


      self.layer3 = nn.Sequential(
          nn.Conv2d(in_channels=d*2, out_channels=d*4, kernel_size=4, stride=2, padding=1),
          nn.BatchNorm2d(d*4),
          nn.LeakyReLU(0.2, inplace=True),)


      self.layer4 = nn.Sequential(
          nn.Conv2d(in_channels=d*4, out_channels=d*8, kernel_size=4, stride=2, padding=1),
          nn.BatchNorm2d(d*8),
          nn.LeakyReLU(0.2, inplace=True),)


      self.layer5 = nn.Sequential(
          nn.Conv2d(in_channels=d*8, out_channels=1, kernel_size=4, stride=1, padding=0),
          nn.Sigmoid())

    def forward(self, input):
      input = self.layer1(input)
      input = self.layer2(input)
      input = self.layer3(input)
      input = self.layer4(input)
      input = self.layer5(input)

      return input
```

Convolution layer 5개를 썼다. Generator에서 영감을 받아 Generator 구조의 정확히 반대로 구현해보았다.

```
100%|██████████| 1583/1583 [12:17<00:00,  2.15it/s]
100%|██████████| 24/24 [00:10<00:00,  2.19it/s]
fid score : 38.13353551619531
```

fid score은 38로 준수한 성능을 보여주었다.

fake 이미지 또한 실제와 비슷하였다. 이는 Generator가 진짜 데이터의 분포를 잘 학습 했다고 볼 수 있다.