학습 내용

- 3.3 GAN으로 MNIST 데이터의 패턴을 학습하여 세상에 없는 글씨체를 만든다

3.3 GAN with MNIST

- MNIST 데이터셋의 패턴을 학습하여 스스로 글씨를 생성할 수 있을까? 

Data Loader 생성

In [1]:
```
import torch
if torch.cuda.is_available() == True:
    device = 'cuda:0'
    print('현재 가상환경 GPU 사용 가능상태')
else:
    device = 'cpu'
    print('GPU 사용 불가능 상태')
```

```
현재 가상환경 GPU 사용 가능상태
```

In [2]:
```
import torch
import torchvision.transforms as transforms
from torchvision.transforms import ToTensor, Resize, Normalize, RandomHorizontalFlip, RandomCrop
import torchvision.datasets as datasets

batch_size = 100

# MNIST Dataset
transform = transforms.Compose([ToTensor(), Normalize(mean=(0.5,), std=(0.5,))]) # -1 ~ 1 사이로 정규화

train_dataset = datasets.MNIST(root='./', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./', train=False, transform=transform, download=False)

# Data Loader (Input Pipeline)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```

Generator와 Discriminator 모델 아키텍처 설계

In [3]:
```
import torch.nn as nn
import torch.nn.functional as F

# 가짜 생성기
class Generator(nn.Module):

    # 코딩타임

    def __init__(self, g_input_dim, g_output_dim):
        super(Generator, self).__init__()
        self.fc1 = nn.Linear(g_input_dim, 256) # 100 -> 256
        self.fc2 = nn.Linear(256, 512) # 256 -> 512
        self.fc3 = nn.Linear(512, 1024) # 512 -> 1024
        self.fc4 = nn.Linear(1024, g_output_dim) # 1024 -> 784

    def forward(self, x):
        x = F.leaky_relu(self.fc1(x), 0.2)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.leaky_relu(self.fc3(x), 0.2)
        return torch.tanh(self.fc4(x)) # Tanh 사용으로 -1 ~ 1 사이로 데이터 생성

    ##########

# 가짜 판별기
class Discriminator(nn.Module):

    # 코딩타임

    def __init__(self, d_input_dim):
```

```python
        super(Discriminator, self).__init__()
        self.fc1 = nn.Linear(d_input_dim, 1024) # 784 -> 1024
        self.fc2 = nn.Linear(1024, 512) # 1024 -> 512
        self.fc3 = nn.Linear(512, 256) # 512 -> 256
        self.fc4 = nn.Linear(256, 1) # 256 -> 1

    def forward(self, x):
        x = F.leaky_relu(self.fc1(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = F.dropout(x, 0.3)
        return torch.sigmoid(self.fc4(x))

    ##########
```

두 모델 G, D 선언

In [4]:

```python
# build network
z_dim = 100
mnist_dim = train_dataset.train_data.size(1) * train_dataset.train_data.size(2) # 28 * 28 = 784

G = Generator(g_input_dim = z_dim, g_output_dim = mnist_dim).to(device)
D = Discriminator(mnist_dim).to(device)
```

```
C:₩Users₩nyan₩anaconda3₩envs₩mldl₩lib₩site-packages₩torchvision₩datasets₩mnist.py:62: UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
```

Optimizer, Loss Function 선언

In [5]:

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

# optimizer
lr = 0.0002
G_optimizer = torch.optim.Adam(G.parameters(), lr = lr)
D_optimizer = torch.optim.Adam(D.parameters(), lr = lr)
```

Generator 학습 알고리즘 제작

In [6]:

```python
# from torch.autograd import Variable

def G_train(x):
    #=======================Train the generator=======================#

    # 코딩타임

    G.zero_grad()

    z = torch.randn(batch_size, z_dim).to(device) # 100 * 100 행렬을 -1 ~ 1 사이 수로 생성
    y = torch.ones(batch_size, 1).to(device) # 1로 가득찬 y를 생성

    G_output = G(z)
    D_output = D(G_output)
    G_loss = criterion(D_output, y) # D를 속여랏!

    # Generator 파라미터만 역전파 + 경사하강 시행
    G_loss.backward()
    G_optimizer.step()

    ##########

    return G_loss.data.item()
```

Discriminator 학습 알고리즘 제작

In [7]:

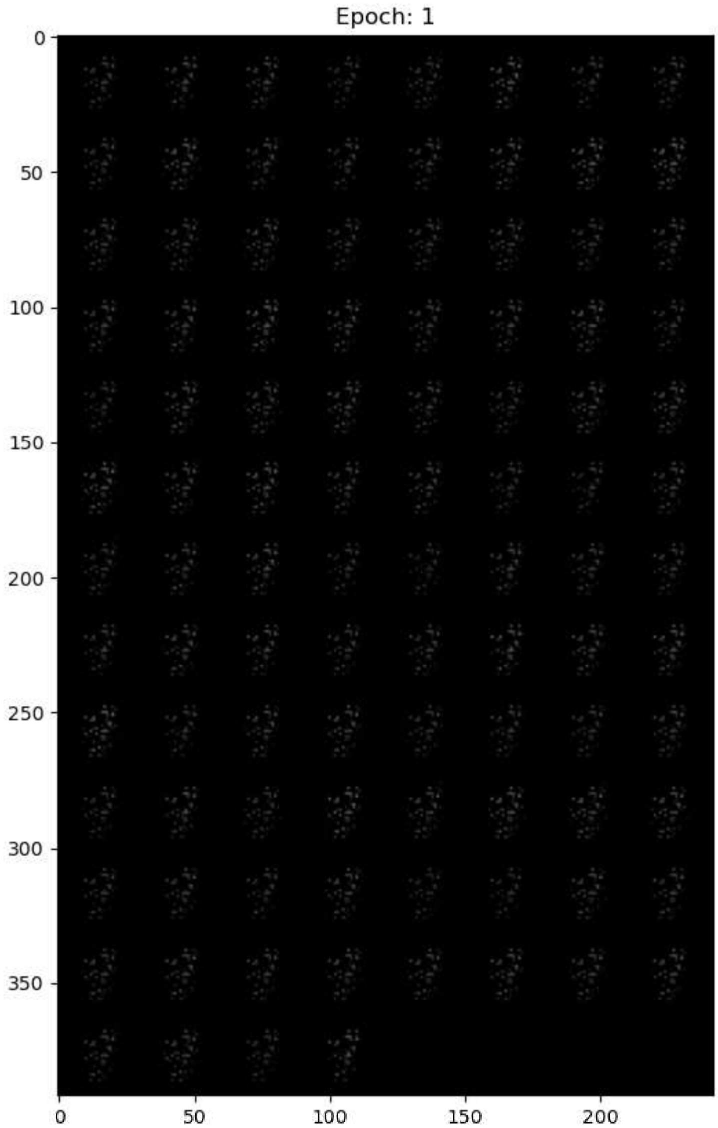
```python
def D_train(x):
    #=======================Train the discriminator=======================#

    # 코딩타임

    D.zero_grad()

    # 진짜 정보로 Discriminator 학습
    x_real = x.view(-1, mnist_dim) # 실제 MNIST 데이터 1장 불러오기
    y_real = torch.ones(batch_size, 1) # 1로 가득찬 100개 리스트 생성(레이블 1 = 진짜)

    x_real, y_real = x_real.to(device), y_real.to(device) # gpu

    D_output = D(x_real)
    D_real_loss = criterion(D_output, y_real)


    # 가짜 정보로 Discriminator 학습
    z = torch.randn(batch_size, z_dim).to(device) # 랜덤 노이즈 생성(100 * 100)
    x_fake, y_fake = G(z), torch.zeros(batch_size, 1).to(device) # 1로 가득찬 100개 리스트 생성(fake 니까~)

    D_output = D(x_fake)
    D_fake_loss = criterion(D_output, y_fake)

    # Disgriminator의 파라미터만 역전파 + 경사하강 시행
    D_loss = D_real_loss + D_fake_loss
    D_loss.backward()
```

```
    D_optimizer.step()

    ##########

    return  D_loss.data.item()
```

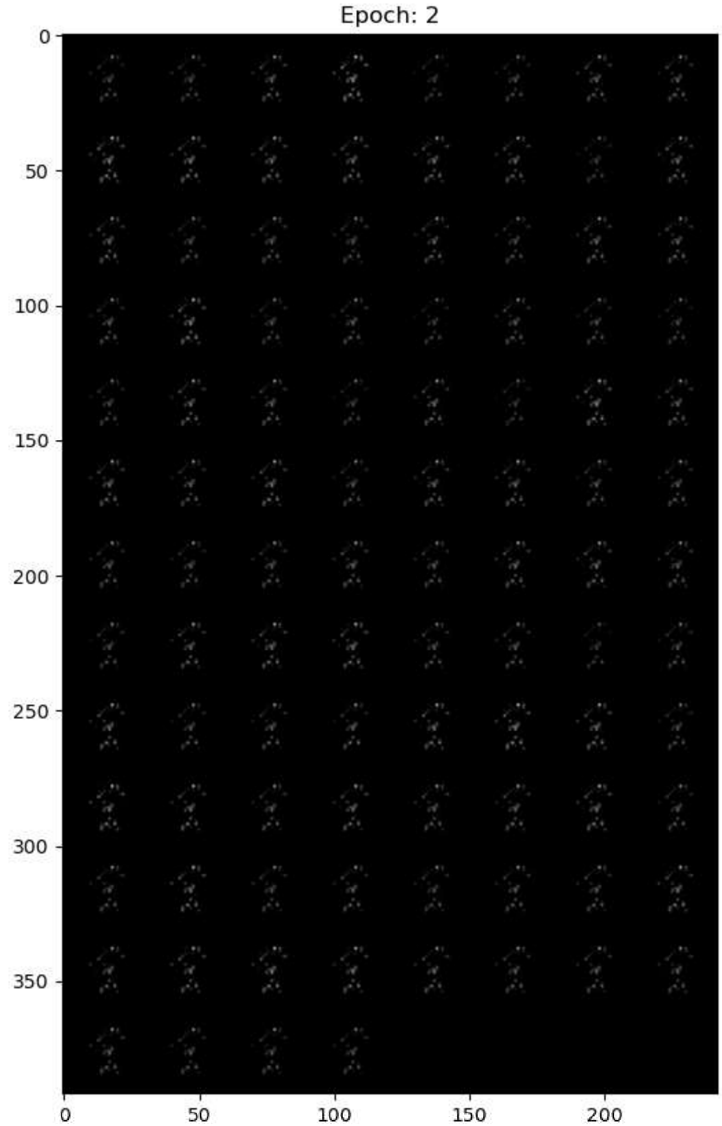
1 epoch씩 학습시키며, 결과 출력하기!

In [8]:
```
from torchvision.utils import save_image
import matplotlib.pyplot as plt
import cv2

n_epoch = 20
cnt = 0
for epoch in range(1, n_epoch+1):
    D_losses, G_losses = [], []
    for batch_idx, (x, _) in enumerate(train_loader):
        D_losses.append(D_train(x))
        G_losses.append(G_train(x))

    print_dis_loss = round(float(torch.mean(torch.FloatTensor(G_losses))), 5)
    print_gen_loss = round(float(torch.mean(torch.FloatTensor(D_losses))), 5)
    print('[{}/{}]: loss_discre.: {}, loss_gen.: {}'.format(epoch, n_epoch, print_dis_loss, print_gen_loss))

    with torch.no_grad():
        test_z = torch.randn(batch_size, z_dim).to(device)
        generated = G(test_z)
        img_path = './GAN_MNIST.png'
        save_image(generated.view(generated.size(0), 1, 28, 28), img_path)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        plt.figure(figsize=(10,10))
        plt.imshow(img)
        plt.title('Epoch: {}'.format(epoch))
        plt.show()
    cnt += 1

print('어쩌면...GAN을 과제에 사용할 수 있을지도...? >_<')
```

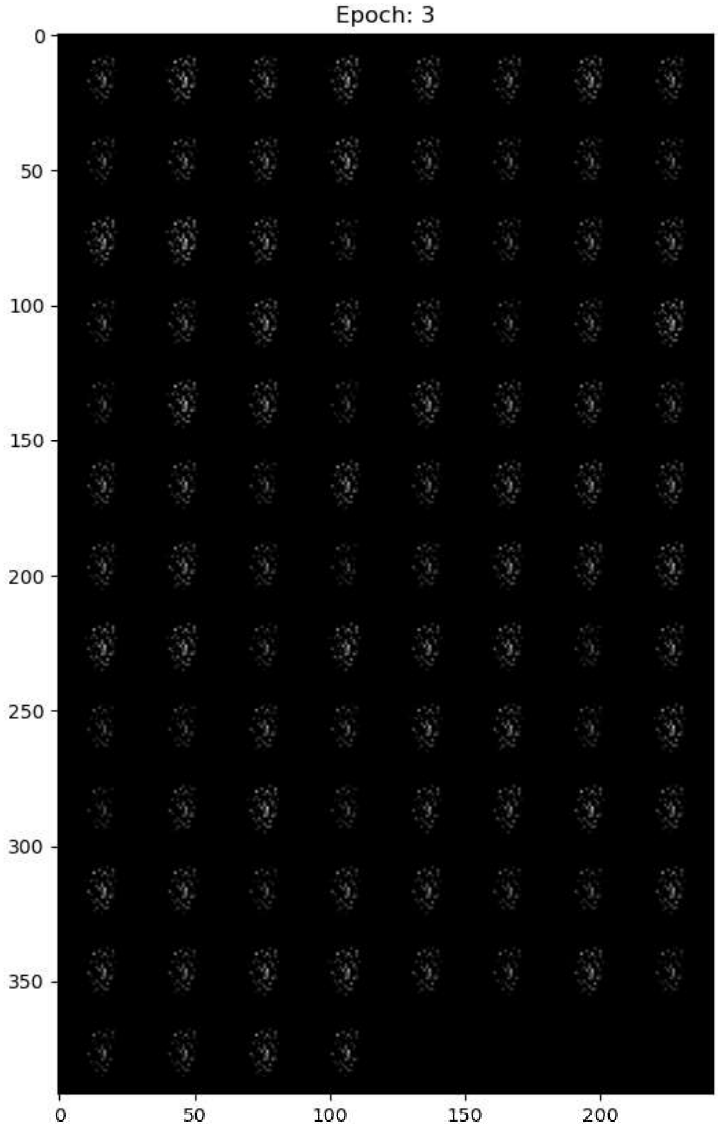
```
[1/20]: loss_discre.: 2.28317, loss_gen.: 1.1254
```



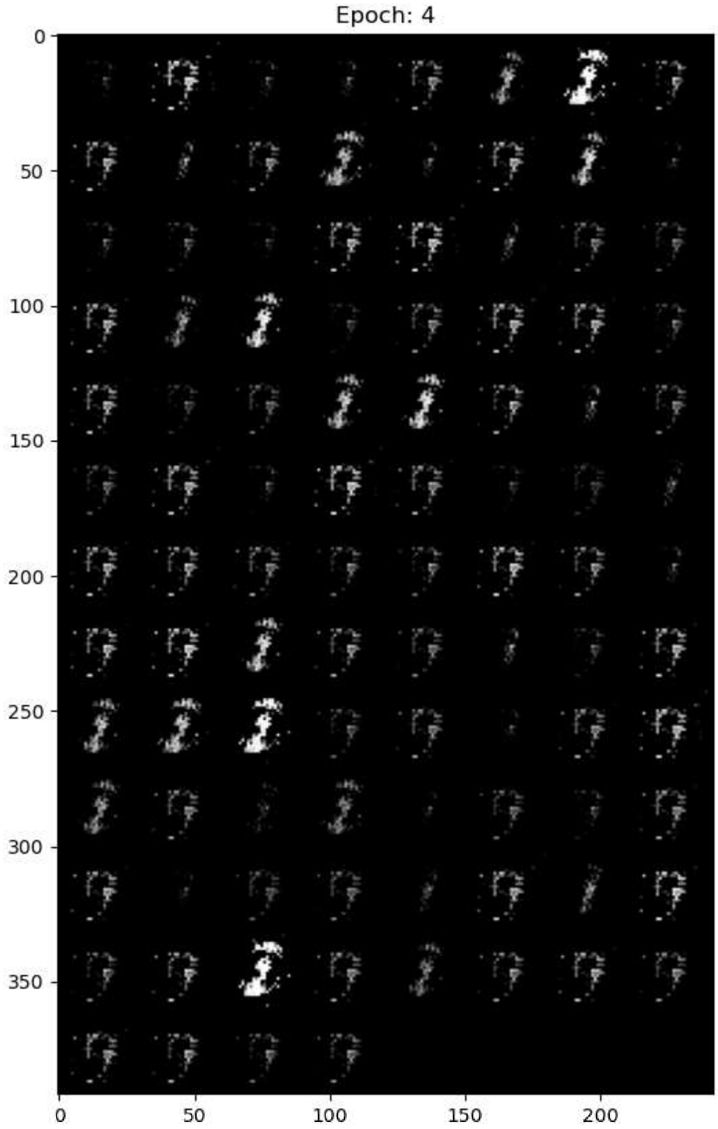

```
[2/20]: loss_discre.: 1.24144, loss_gen.: 1.15558
```



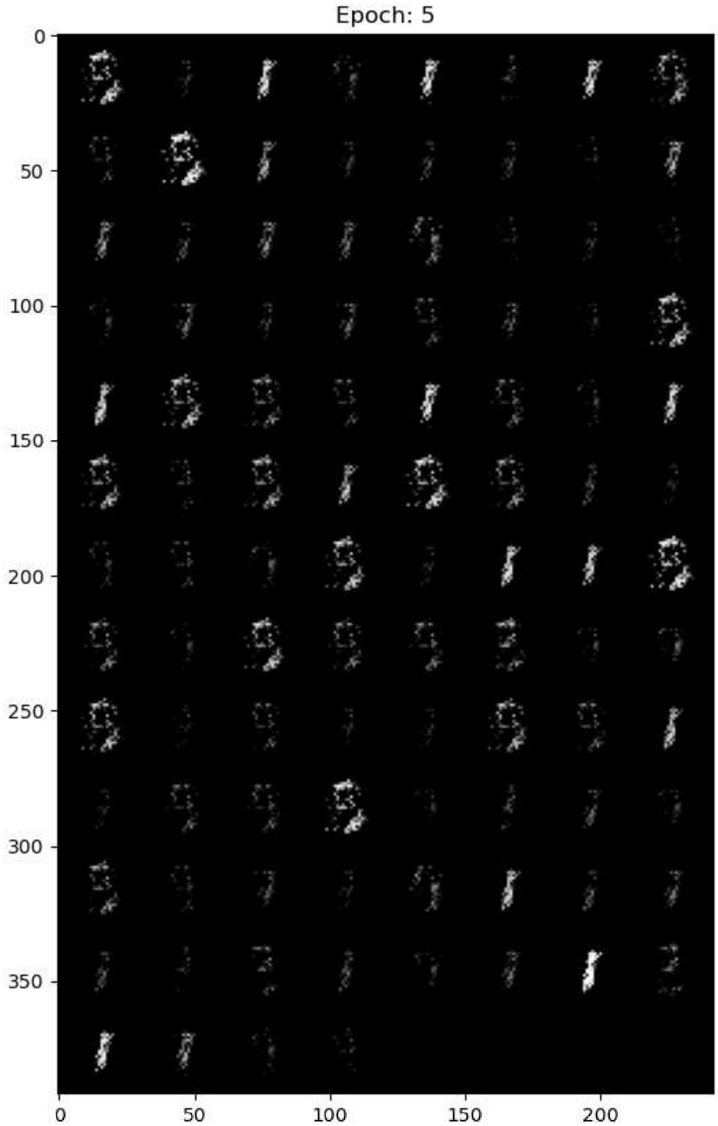
[3/20]: loss_discre.: 2.46415, loss_gen.: 0.92984



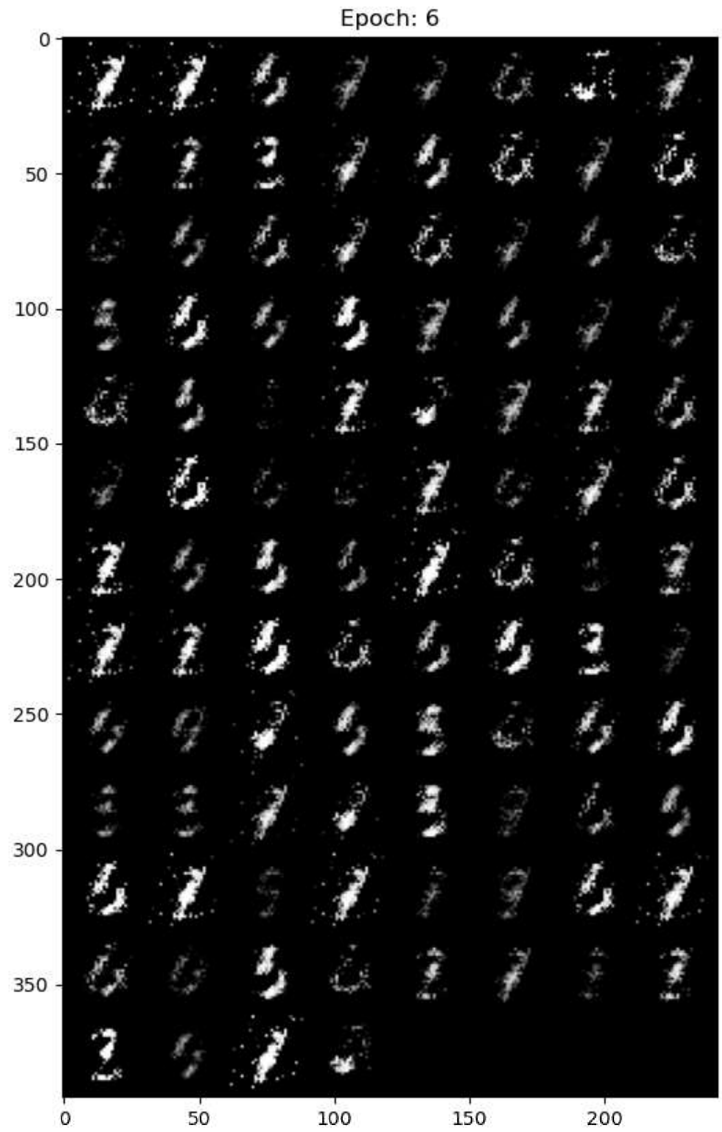
[4/20]: loss_discre.: 2.82064, loss_gen.: 0.57636



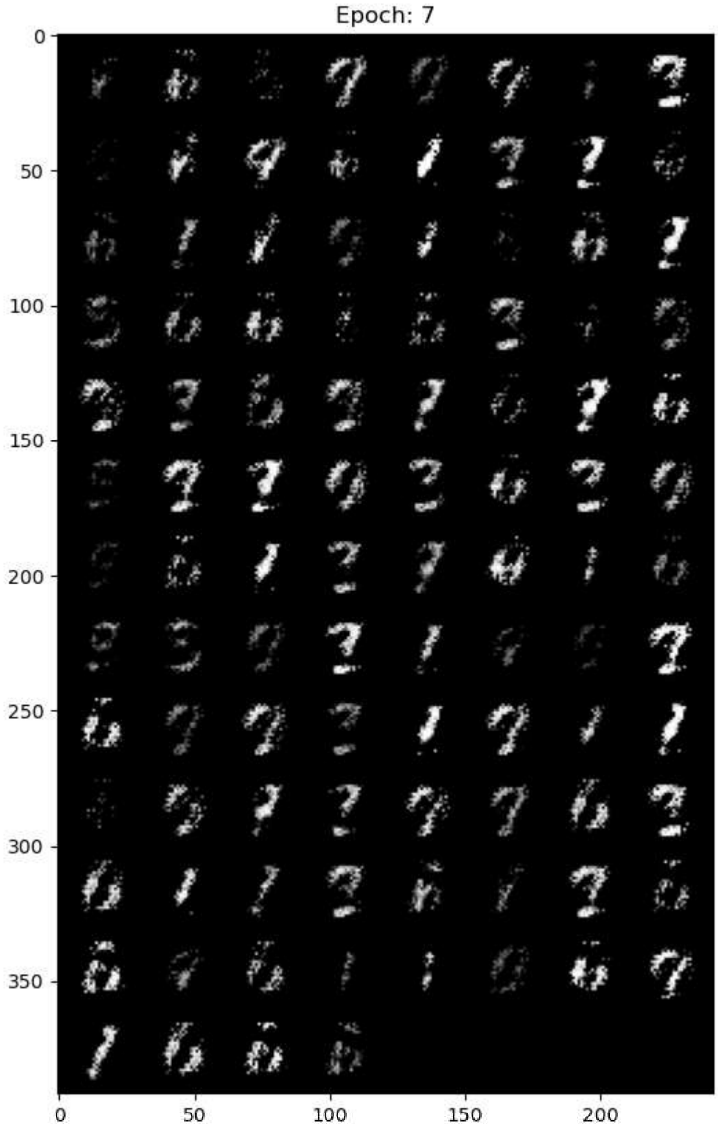
[5/20]: loss_discre.: 2.76436, loss_gen.: 0.60252



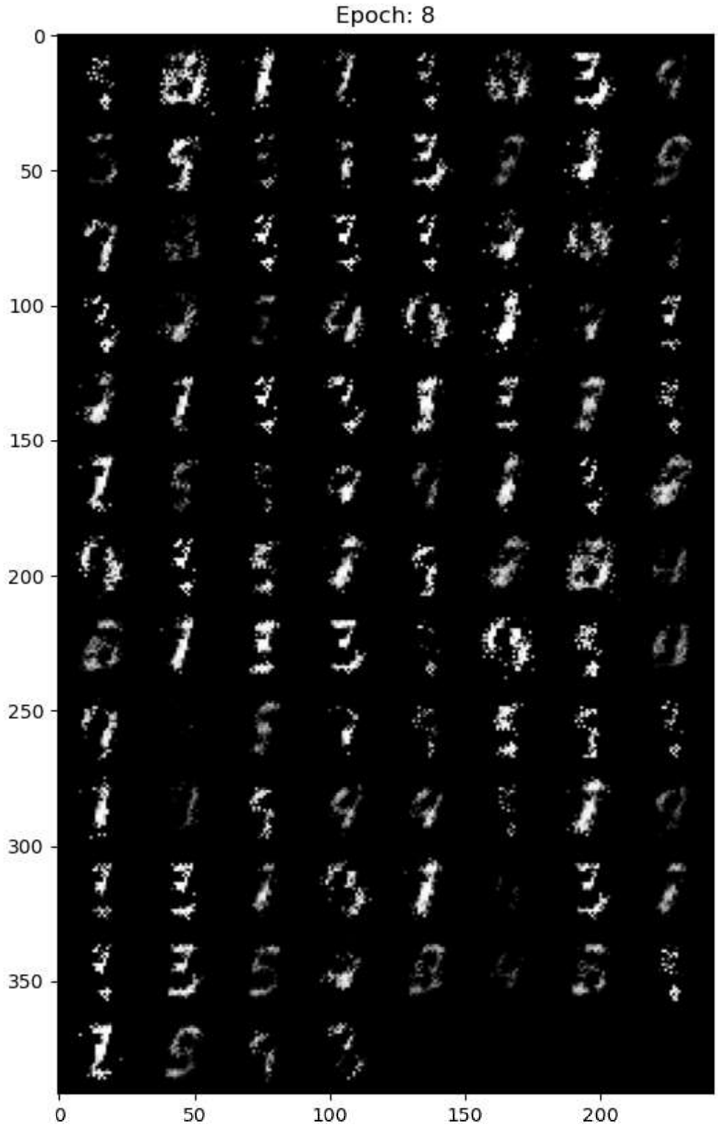
[6/20]: loss_discre.: 3.03271, loss_gen.: 0.46877



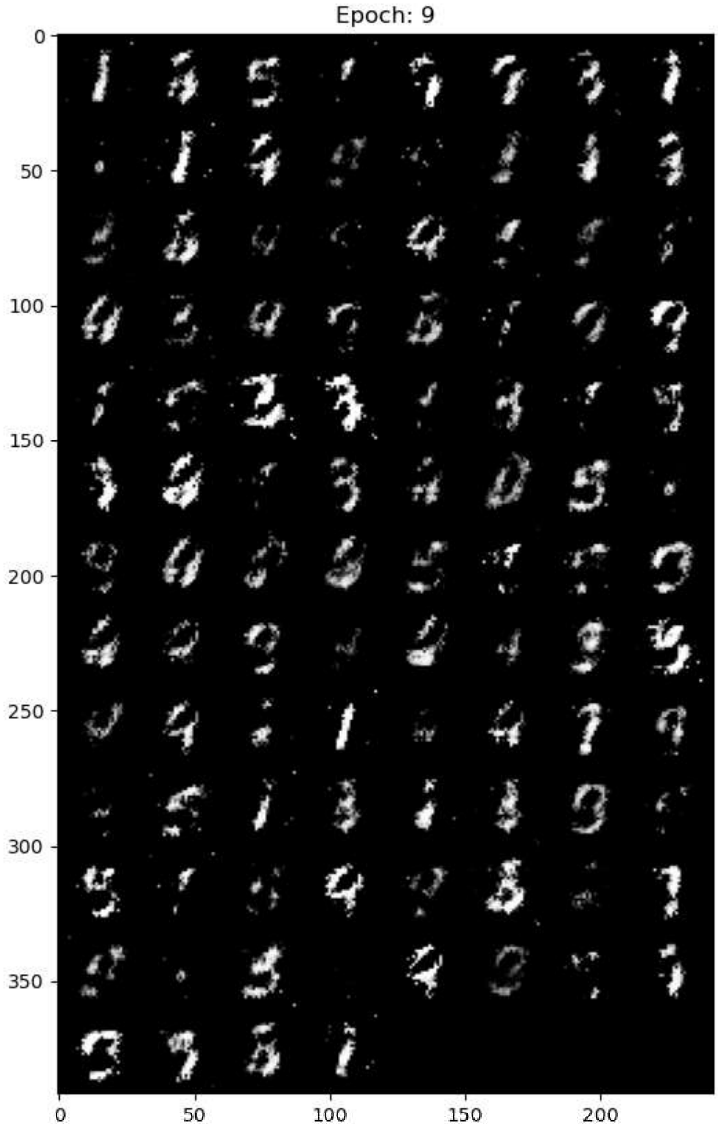
[7/20]: loss_discre.: 2.87183, loss_gen.: 0.51492



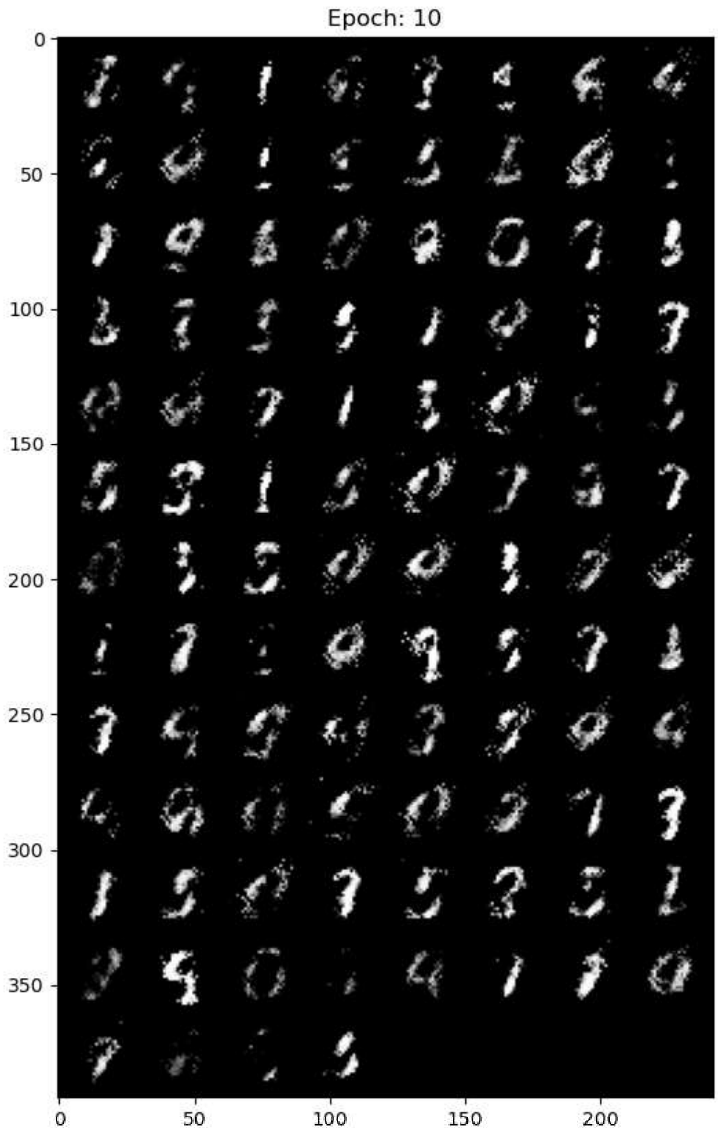
[8/20]: loss_discre.: 2.74167, loss_gen.: 0.52653



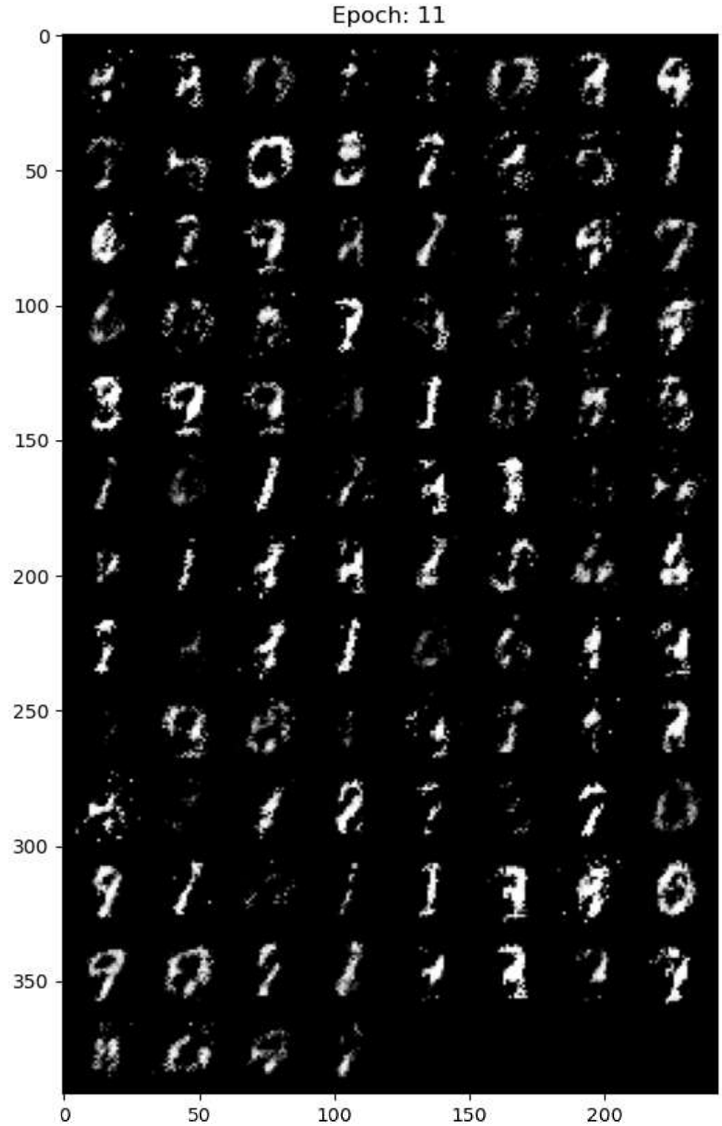
[9/20]: loss_discre.: 2.50233, loss_gen.: 0.62972



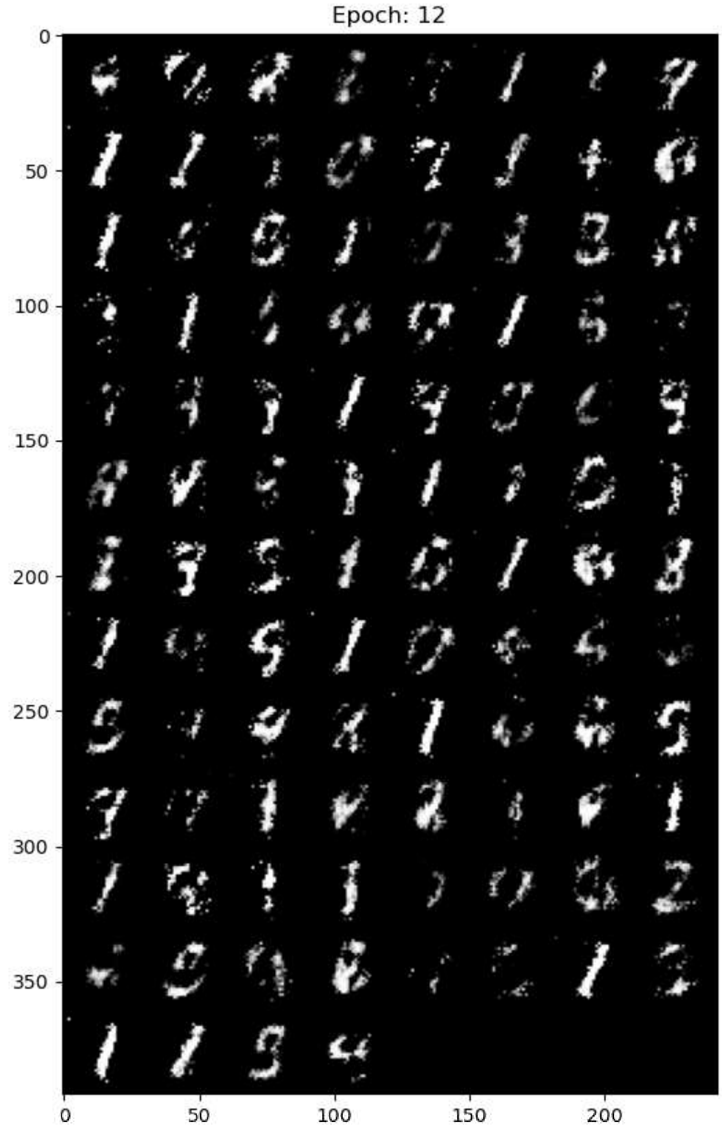
[10/20]: loss_discre.: 2.32682, loss_gen.: 0.65559



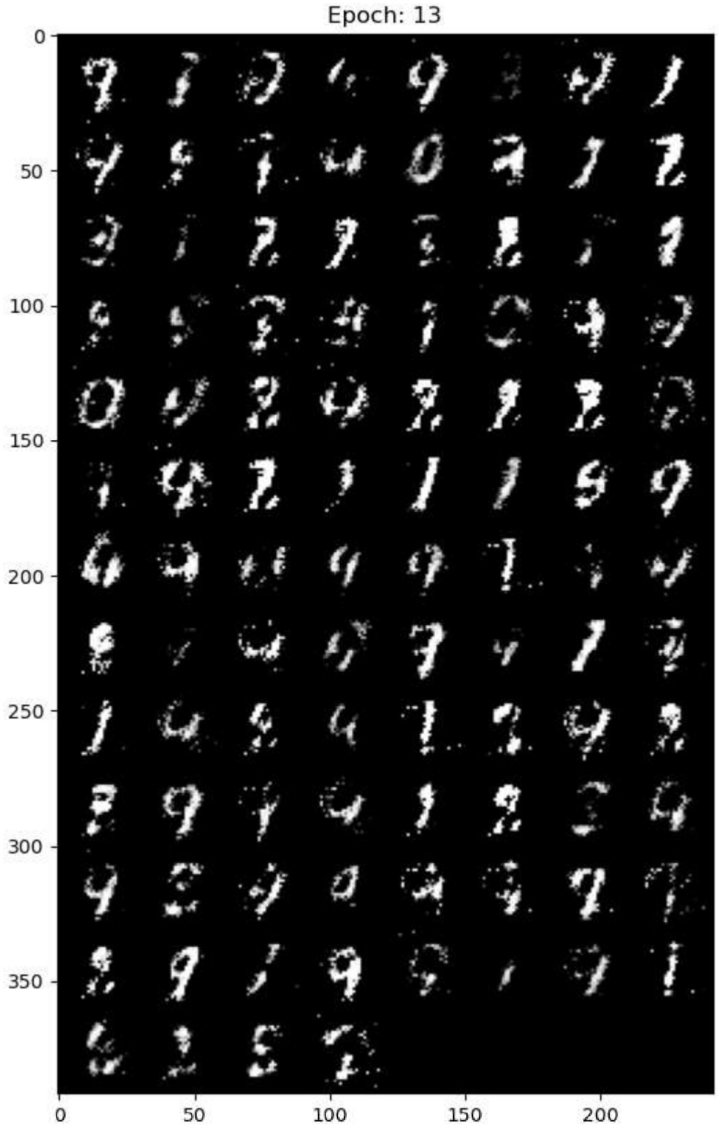
[11/20]: loss_discre.: 2.26705, loss_gen.: 0.64586



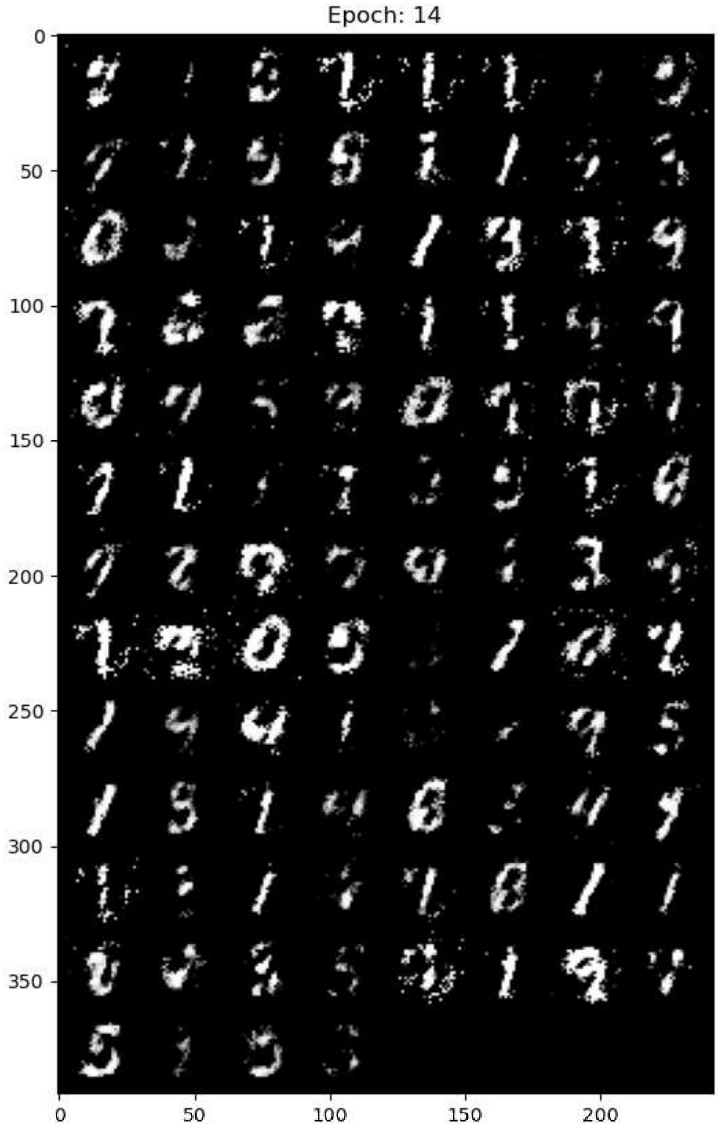
[12/20]: loss_discre.: 2.11244, loss_gen.: 0.71015



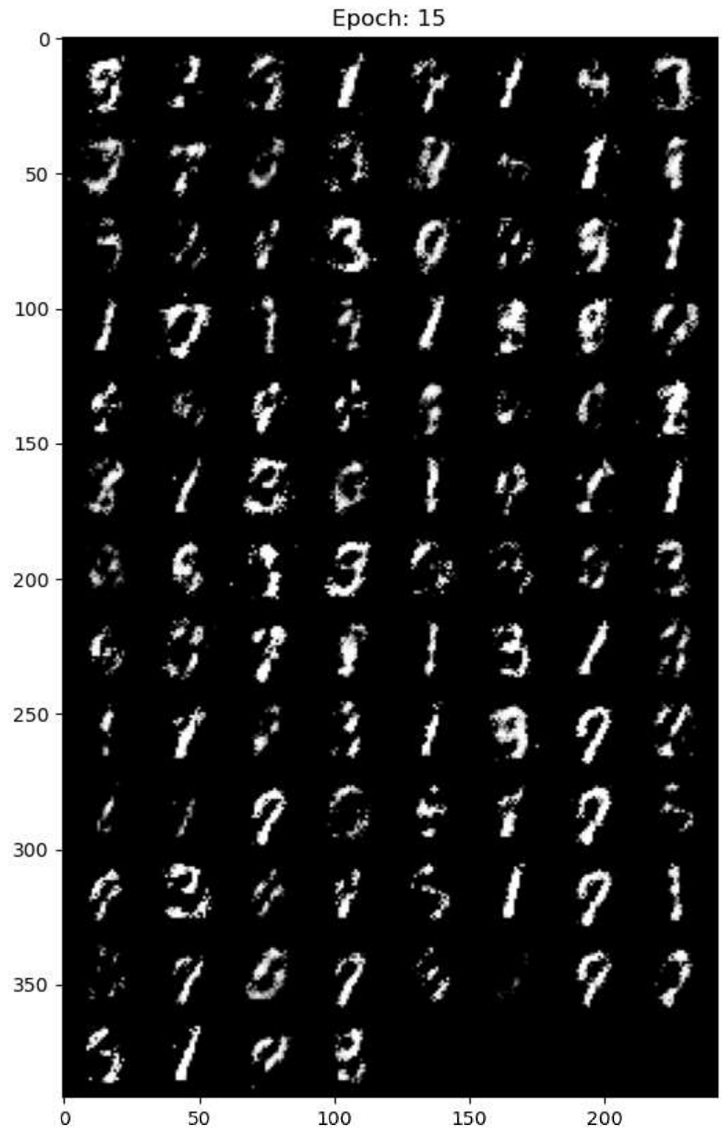
[13/20]: loss_discre.: 2.16832, loss_gen.: 0.71317



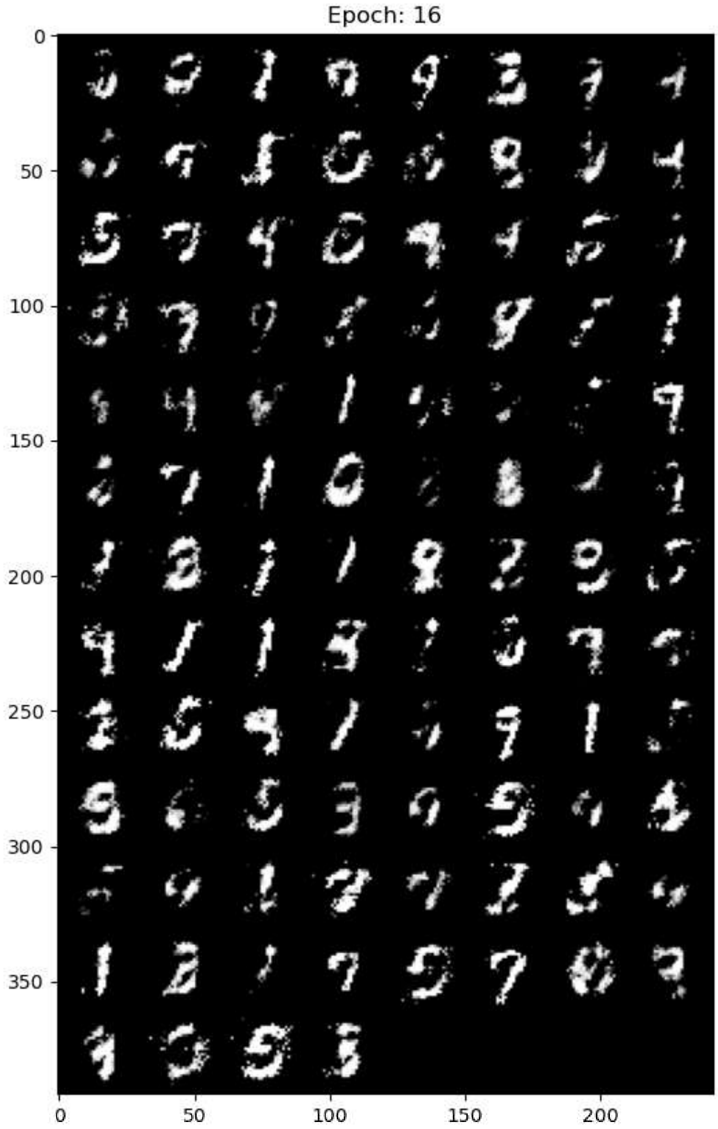
[14/20]: loss_discre.: 2.13793, loss_gen.: 0.70944



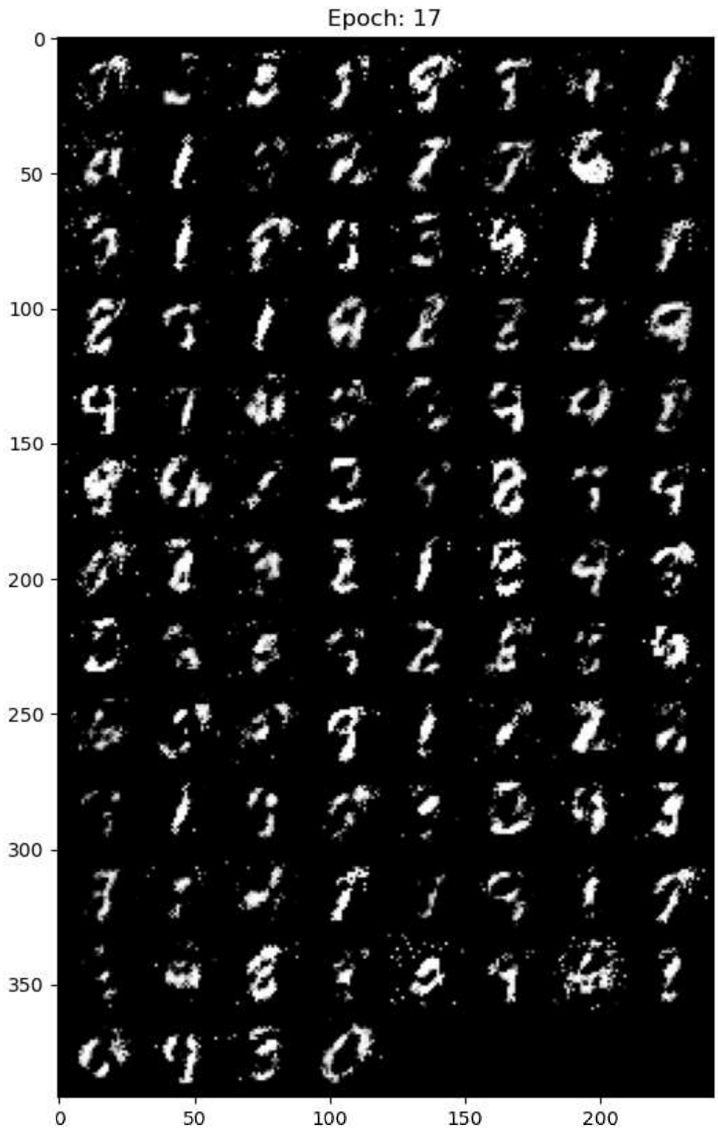
[15/20]: loss_discre.: 1.98844, loss_gen.: 0.75723



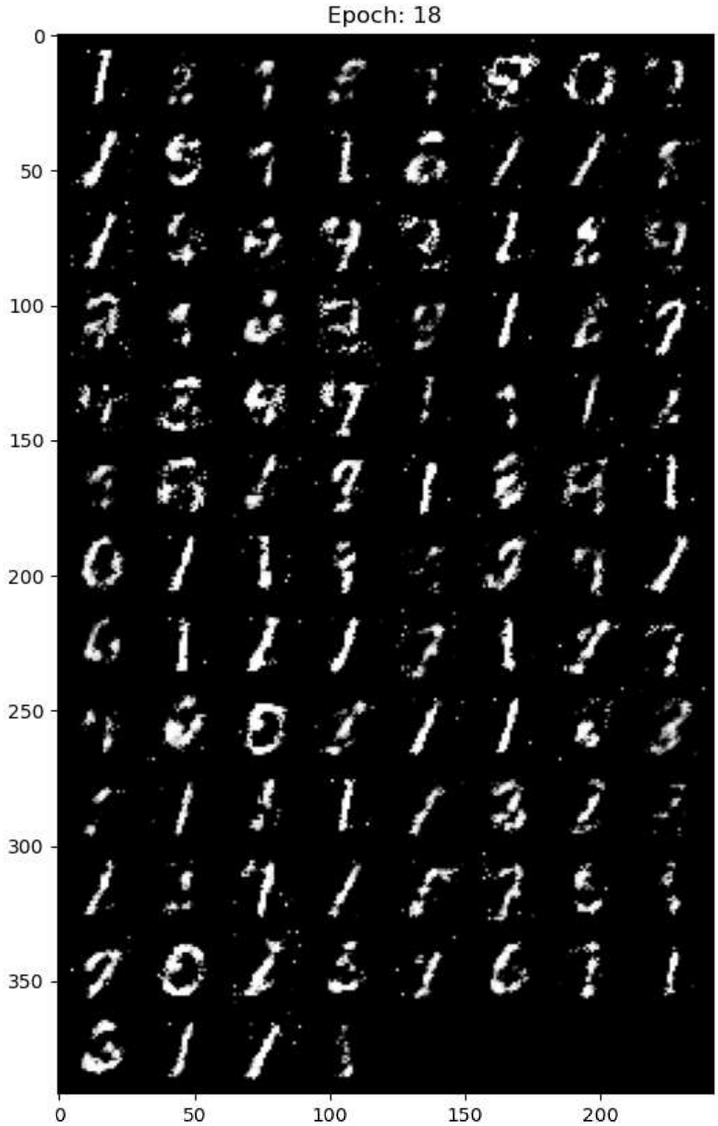
[16/20]: loss_discre.: 1.98564, loss_gen.: 0.76585



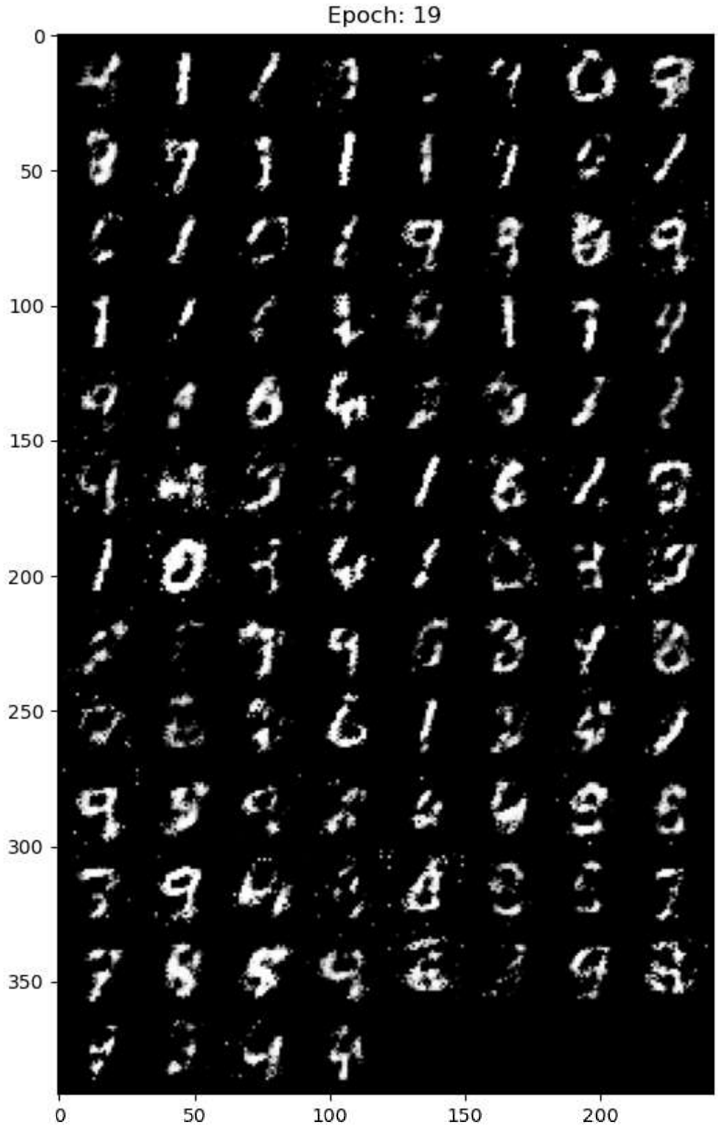
[17/20]: loss_discre.: 2.06535, loss_gen.: 0.75362



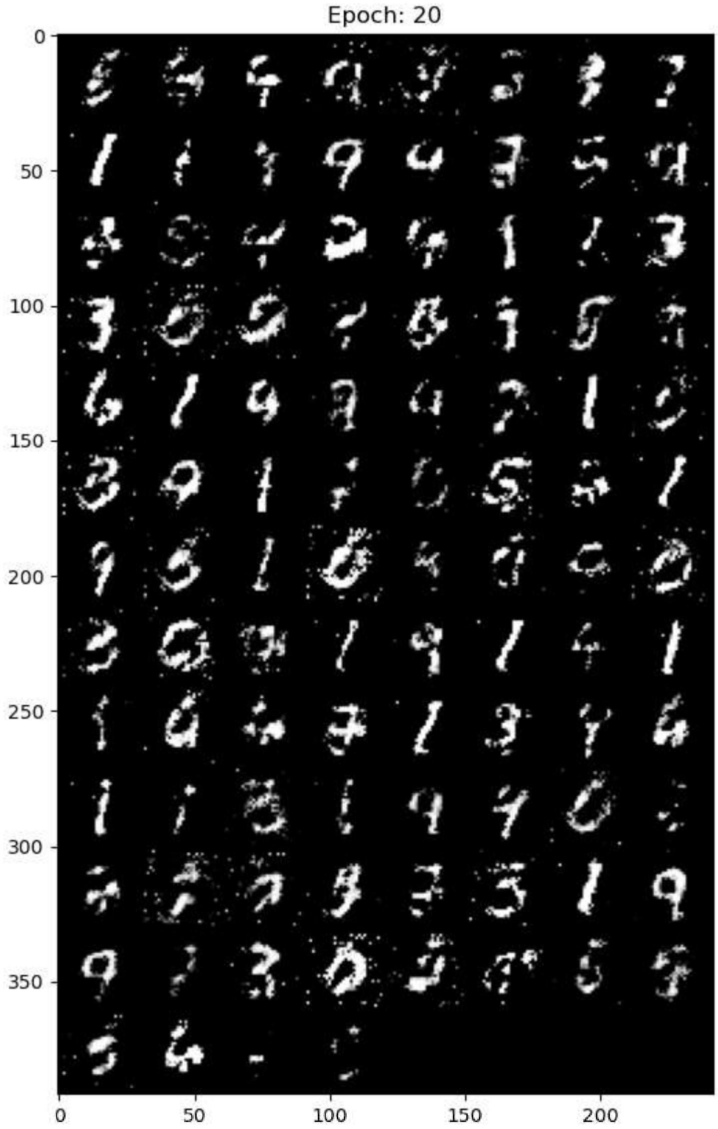
[18/20]: loss_discre.: 1.99676, loss_gen.: 0.75535



[19/20]: loss_discre.: 1.76478, loss_gen.: 0.83042



[20/20]: loss_discre.: 1.68866, loss_gen.: 0.89078



어쩌면...GAN을 과제에 사용할 수 있을지도...? >_<

고생 많으셨습니다. 좋은 결과 있기를 바랍니다.