InfoGAN Code Review

Week 5/ SAI-CV-박제건

InfoGAN 등장배경

- **논문 제목이 주는 힌트:**
  Interpretable Representation Learning by Information Maximizing

- **기존의 Latent Vector의 의미를 알기 어려웠음.**
  –> Latent Vector 이동과 Output 간의 상관관계가 적음(Entangled Representation)

- **따라서 기존 Object Function에 새로운 term 추가.**
  –> Latent Vector Code와 Output 간의 상호 정보량 최대화 하는 방향으로 학습

Adversarial Nets

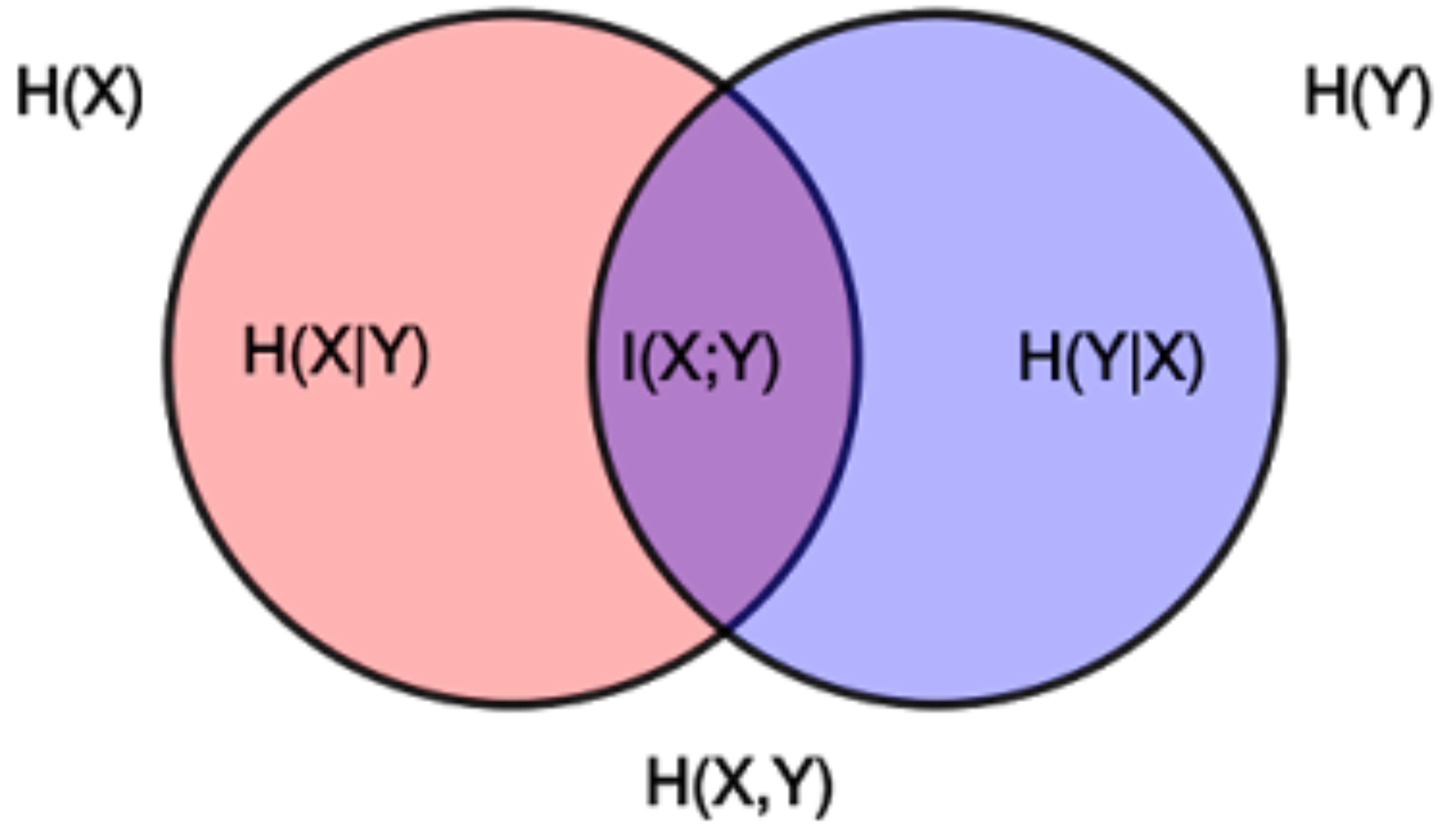
$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_{\text{generated}}(z)}[1-\log D(G(z))]$$

$$\downarrow$$

$$\min_{G,Q}\max_{D} V_{\text{InfoGAN}}(D,G,Q) = V(D,G) - \lambda L_I(G,Q)$$

💡 MinMax game with a variational regularization of mutual information and a hyperparameter λ

Mutual Information for Inducing Latent Codes



- $I(X;Y) = D_{\mathrm{KL}}(P_{(X,Y)} || P_X \otimes P_Y)$
  KLD의 정의를 사용해서 정리하면 아래와 같다.

- $I(X;Y) = \int_{\mathcal{Y}} \int_{\mathcal{X}} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right) dxdy$

- 엔트로피 : $H(X) = -\int_{\mathcal{X}} p(x) \log p(x) dx$ 로 표현 할 수 있으므로,

- 조건부 엔트로피
  $H(X|Y) = -\int_{\mathcal{X}} \int_{\mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(y)} dydx = \int_{\mathcal{X}} \int_{\mathcal{Y}} p(x,y) \log p(x|y) dydx$
  $= \mathbb{E}_{x \sim P_X}[\mathbb{E}_{y \sim P_Y}[\log P(X|Y)]]$

- 상호정보량은 다음과 같이 표현할 수 있다.
  $I(X;Y) = H(X) - H(X|Y) = \mathbb{E}_{X \sim P_X}[\mathbb{E}_{Y \sim P_{Y|X}}[\log P(X|Y)]] + H(X)$

위의 상호 정보량 식에서 $X \leftarrow c, Y \leftarrow G(z,c)$를 대입해보면,
$I(c; G(z,c)) \ = \mathbb{E}_{x \sim G(z,c)}[\mathbb{E}_{c' \sim P(c|x)}[\log P(c'|x)]] + H(c)$ 가 된다.

$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(c; G(z,c))$

→ G가 최소화 할 때 상호 정보량 최대화

Variational Mutual Information Maximazation

$$\begin{aligned}
I(c; G(z,c)) &= H(c) - H(c|G(z,c)) \\
&= \mathbb{E}_{x\sim G(z,c)}[\mathbb{E}_{c'\sim P(c|x)}[\log P(c'|x)]] + H(c) \\
&= \mathbb{E}_{x\sim G(z,c)}[\underbrace{D_{KL}(P(\cdot|x)||Q(\cdot|x))}_{\geq 0} + \mathbb{E}_{c'\sim P(c|x)}[\log Q(c'|x)]] + H(c) \\
&\geq \mathbb{E}_{x\sim G(z,c)}[\mathbb{E}_{c'\sim P(c|x)}[\log Q(c'|x)]] + H(c) \\
&= \mathbb{E}_{c\sim P(c), x\sim G(z,c)}[\log Q(c|x)] + H(c) \\
&\overset{\text{let}}{=} L_I(G,Q)
\end{aligned}$$

⬅️ Auxiliary Distribution Q 도입 & Lower Bound

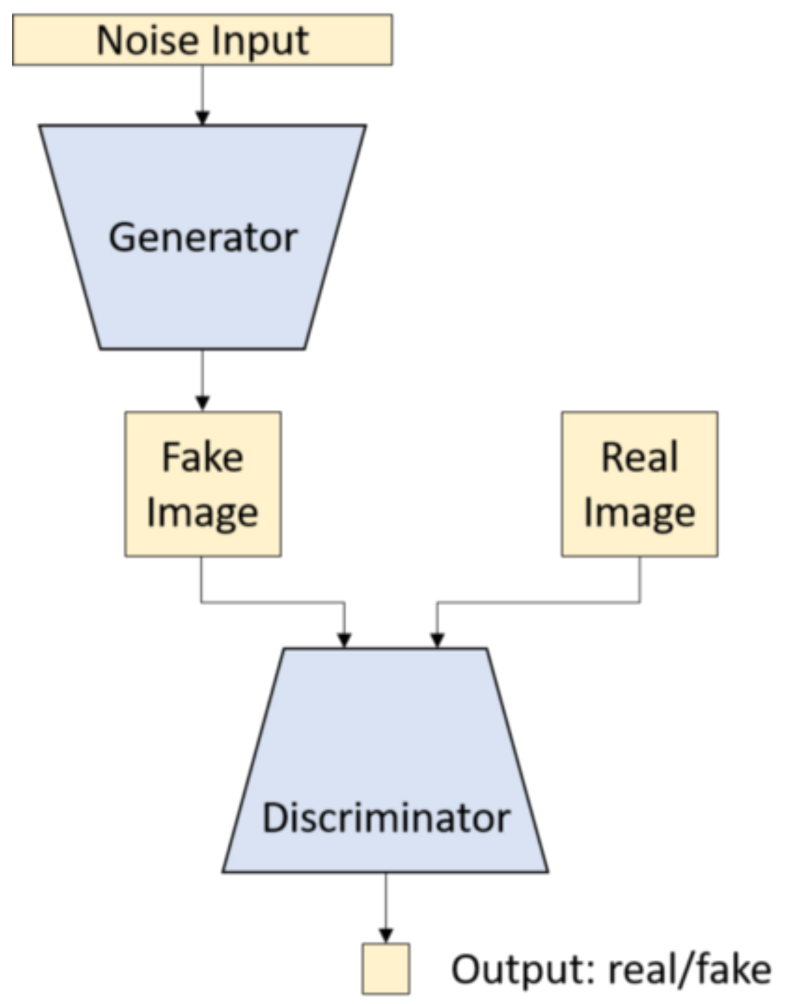
⬅️ Lemma 5.1(Law of Total Expectation) for Sampling

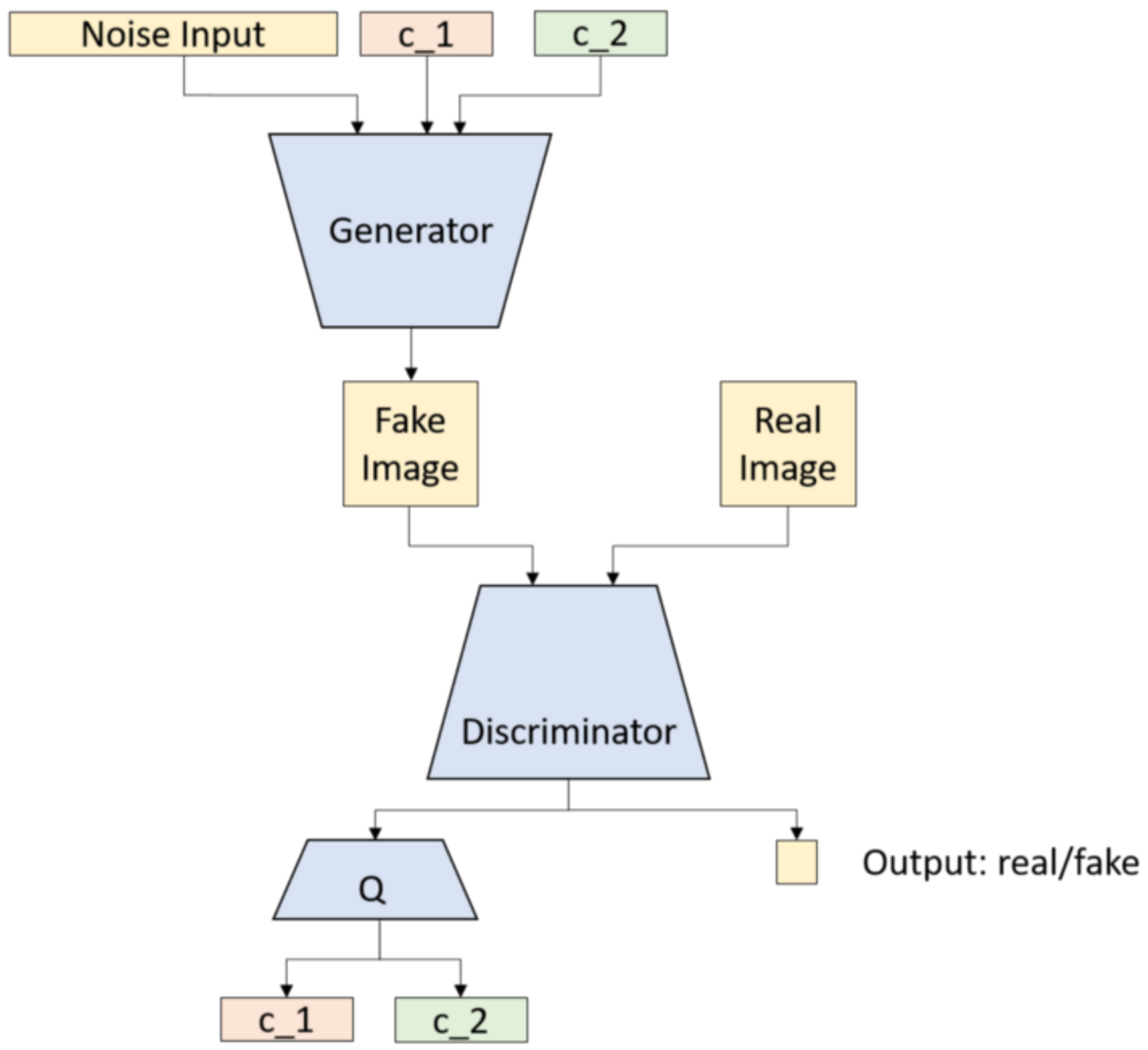
$$\min_{G,Q}\max_{D} V_{\text{InfoGAN}}(D,G,Q) = V(D,G) - \lambda L_I(G,Q)$$

- G, Q는 L_I도 최대화 해야 함
- Q는 $G(z,c)$를 다시 c'로 잘 바꿔야 하고, G는 Q가 잘 바꿀 수 있도록 $x = G(z,c)$를 잘 생성해야 함.

Share Part (Paper Structure)

```python
import torch.nn as nn


class SharePart(nn.Module):

    ''' front end part of discriminator and Q'''

    def __init__(self):
        super(SharePart, self).__init__()

        self.main = nn.Sequential(
          nn.Conv2d(1, 64, 4, 2, 1),
          nn.LeakyReLU(0.1, inplace=True),
          nn.Conv2d(64, 128, 4, 2, 1, bias=False),
          nn.BatchNorm2d(128),
          nn.LeakyReLU(0.1, inplace=True),
          nn.Conv2d(128, 1024, 7, bias=False),
          nn.BatchNorm2d(1024),
          nn.LeakyReLU(0.1, inplace=True),
        )

    def forward(self, x):
        output = self.main(x)
        return output
```





Discriminator D & Recognition Q

```python
class D(nn.Module):

    def __init__(self):
        super(D, self).__init__()

        self.main = nn.Sequential(
            nn.Conv2d(1024, 1, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        output = self.main(x).view(-1, 1)
        return output


class Q(nn.Module):

    def __init__(self):
        super(Q, self).__init__()

        self.conv = nn.Conv2d(1024, 128, 1, bias=False)
        self.bn = nn.BatchNorm2d(128)
        self.lReLU = nn.LeakyReLU(0.1, inplace=True)
        self.conv_disc = nn.Conv2d(128, 10, 1)
        self.conv_mu = nn.Conv2d(128, 2, 1)
        self.conv_var = nn.Conv2d(128, 2, 1)

    def forward(self, x):

        y = self.lReLU(self.bn(self.conv(x)))
        disc_logits = self.conv_disc(y).squeeze()

        mu = self.conv_mu(y).squeeze()
        var = self.conv_var(y).squeeze().exp()

        return disc_logits, mu, var
```

⬅️ D : FC output layer for D

- < Q network 의 구조 >
- FC
- BN(128)
- Activation(Leaky ReLU)
- Disc-Mu-Var

Generator G & Weight Init

```python
class G(nn.Module):

    def __init__(self):
        super(G, self).__init__()

        self.main = nn.Sequential(
          nn.ConvTranspose2d(74, 1024, 1, 1, bias=False),
          nn.BatchNorm2d(1024),
          nn.ReLU(True),
          nn.ConvTranspose2d(1024, 128, 7, 1, bias=False),
          nn.BatchNorm2d(128),
          nn.ReLU(True),
          nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
          nn.BatchNorm2d(64),
          nn.ReLU(True),
          nn.ConvTranspose2d(64, 1, 4, 2, 1, bias=False),
          nn.Sigmoid()
        )

    def forward(self, x):
        output = self.main(x)
        return output

def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
         m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)
```

⬅️ G 입력이 74인 이유 :

10 Categorical

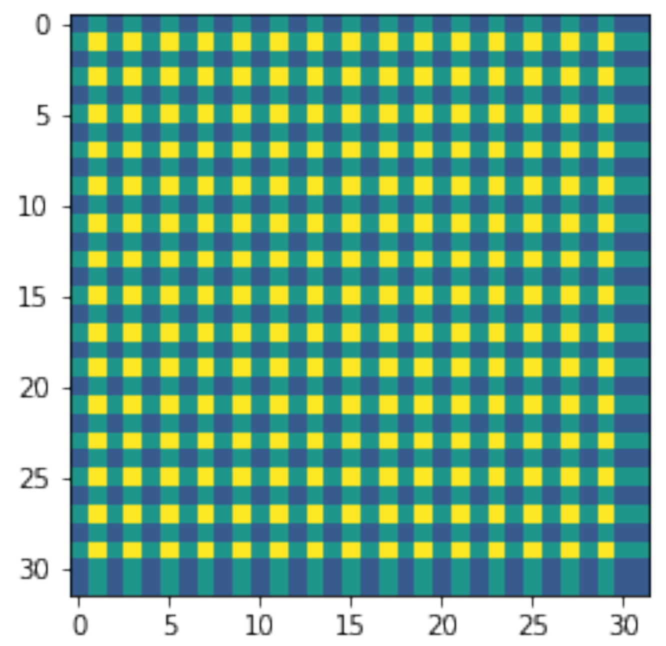
+

2 Latent

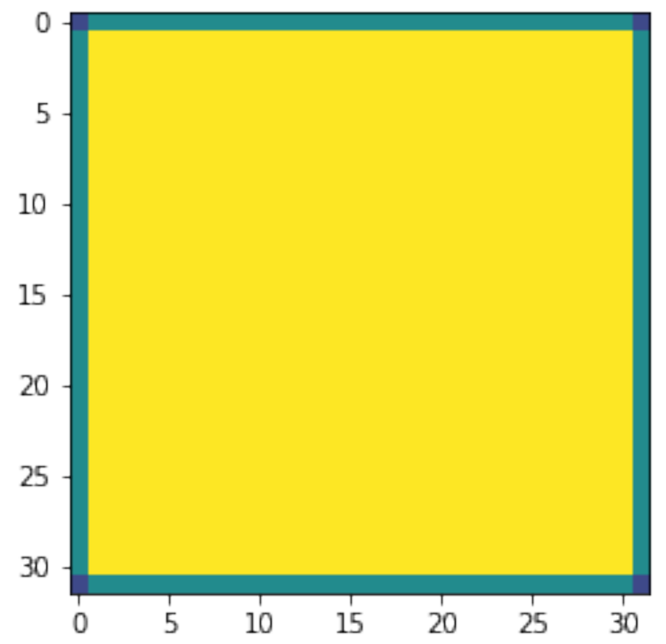
+

62 Noise Variable

Cf) ConvTranspose2D Checkerboard Artifact



kernel : 3, stride : 2, padding : 1, output_padding : 1



Kernel : 4, stride:2, padding : 1

Negative Log-Likelihood Gaussian

$$= \mathbb{E}_{c \sim P(c), x \sim G(z,c)} [\underline{\log Q(c|x)}] + H(c)$$

Maximizing Object

Minimizing NLL of $\boxed{q(C|X)}$

$$
\begin{aligned}
l(\mu, \sigma^2; x_1, \ldots, x_n) &= \ln\left(L(\mu, \sigma^2; x_1, \ldots, x_n)\right) \\
&= \ln\left(\left(2\pi\sigma^2\right)^{-n/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2 \right)\right) \\
&= \ln\left(\left(2\pi\sigma^2\right)^{-n/2}\right) + \ln\left(\exp\left(-\frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2 \right)\right) \\
&= \boxed{-\frac{n}{2} \ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2} \\
&= -\frac{n}{2} \ln(2\pi) - \frac{n}{2} \ln\left(\sigma^2\right) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2
\end{aligned}
$$

```
class log_gaussian:

    def __call__(self, x, mu, var):
        logli = -0.5*(var.mul(2*np.pi)+1e-6).log() - \
        (x-mu).pow(2).div(var.mul(2.0)+1e-6)
        return logli.sum(1).mean().mul(-1)
```

Sampling from noise tensor

```
# sampling from noise tensor
def _noise_sample(self, dis_c, con_c, noise, bs):
    idx = np.random.randint(10, size=bs)
    c = np.zeros((bs, 10))
    c[range(bs),idx] = 1.0

    dis_c.data.copy_(torch.Tensor(c))
    con_c.data.uniform_(-1.0, 1.0)
    noise.data.uniform_(-1.0, 1.0)
    z = torch.cat([noise, dis_c, con_c], 1).view(-1, 74, 1, 1)

    return z, idx
```

⬅️ c : [100,10] 크기의 0~9 one_hot encoding

Discrete : c 활용 (0~9) 숫자 인식

C & noise : Uniform Sampling(-1 ~ 1)

Z = concatenate [noise, discrete, continuous]

```
def train(self):

    real_x = torch.FloatTensor(self.batch_size, 1, 28, 28).cuda()
    label = torch.FloatTensor(self.batch_size, 1).cuda()
    dis_c = torch.FloatTensor(self.batch_size, 10).cuda()
    con_c = torch.FloatTensor(self.batch_size, 2).cuda()
    noise = torch.FloatTensor(self.batch_size, 62).cuda()


    criterionD = nn.BCELoss().cuda()
    criterionQ_dis = nn.CrossEntropyLoss().cuda()
    criterionQ_con = log_gaussian()

    optimD = optim.Adam([{'params':self.SP.parameters()}, {'params':self.D.parameters()}], lr=0.0002, betas=(0.5,
    optimG = optim.Adam([{'params':self.G.parameters()}, {'params':self.Q.parameters()}], lr=0.001, betas=(0.5, (
```

Discriminator : Binary Cross Entropy

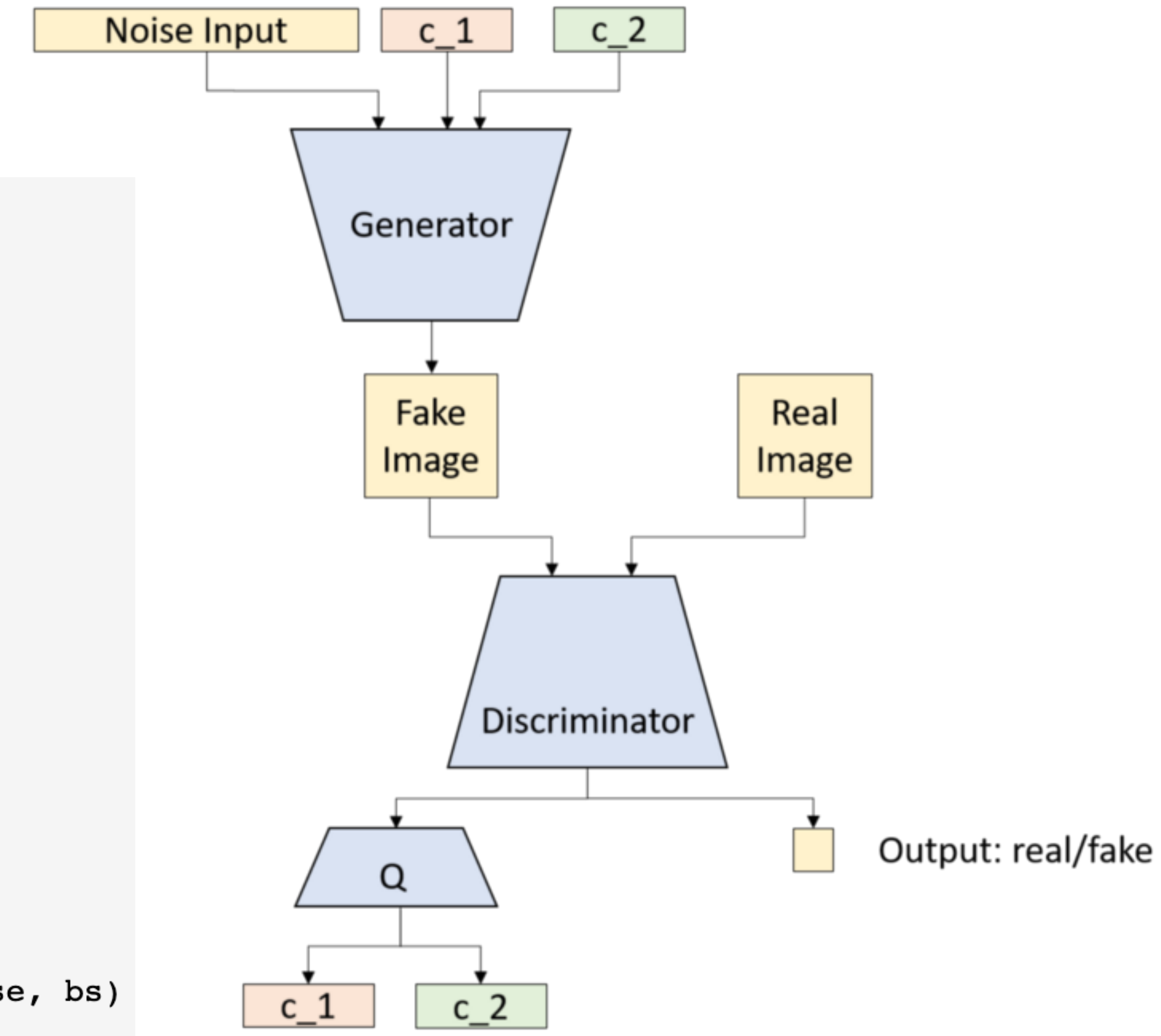
Discrete : (Cartegorical) Cross Entropy

Continuous : Gaussian Negative Log Likelihood

Training Procedure

[Discriminator]

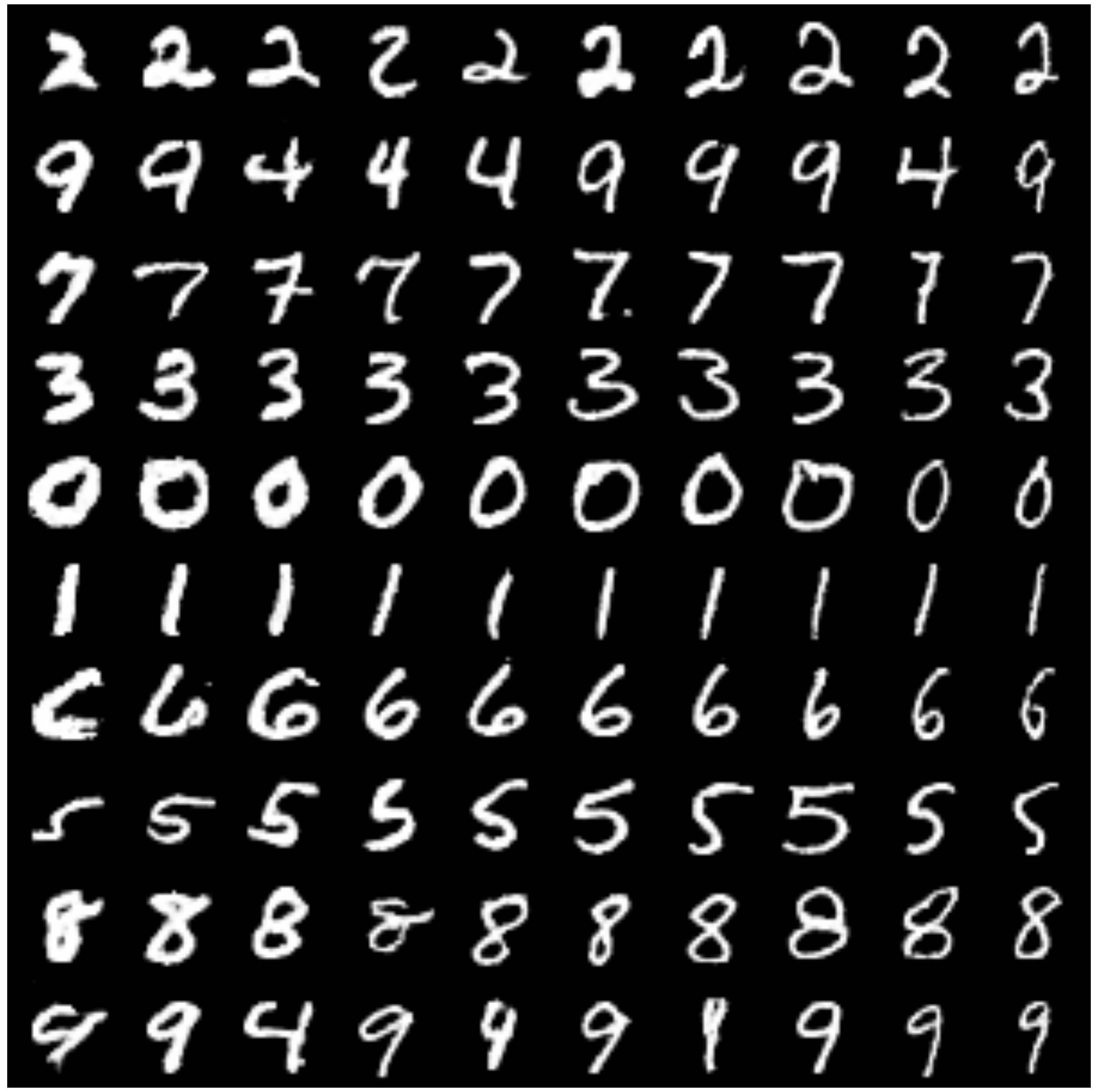
```
# real part
optimD.zero_grad()

x, _ = batch_data

bs = x.size(0)
real_x.data.resize_(x.size())
label.data.resize_(bs, 1)
dis_c.data.resize_(bs, 10)
con_c.data.resize_(bs, 2)
noise.data.resize_(bs, 62)

real_x.data.copy_(x)
sp_out1 = self.SP(real_x)
probs_real = self.D(sp_out1)
label.data.fill_(1)
loss_real = criterionD(probs_real, label)
loss_real.backward()

# fake part
z, idx = self._noise_sample(dis_c, con_c, noise, bs)
fake_x = self.G(z)
sp_out2 = self.SP(fake_x.detach())
probs_fake = self.D(sp_out2)
label.data.fill_(0)
loss_fake = criterionD(probs_fake, label)
loss_fake.backward()

D_loss = loss_real + loss_fake

optimD.step()
```



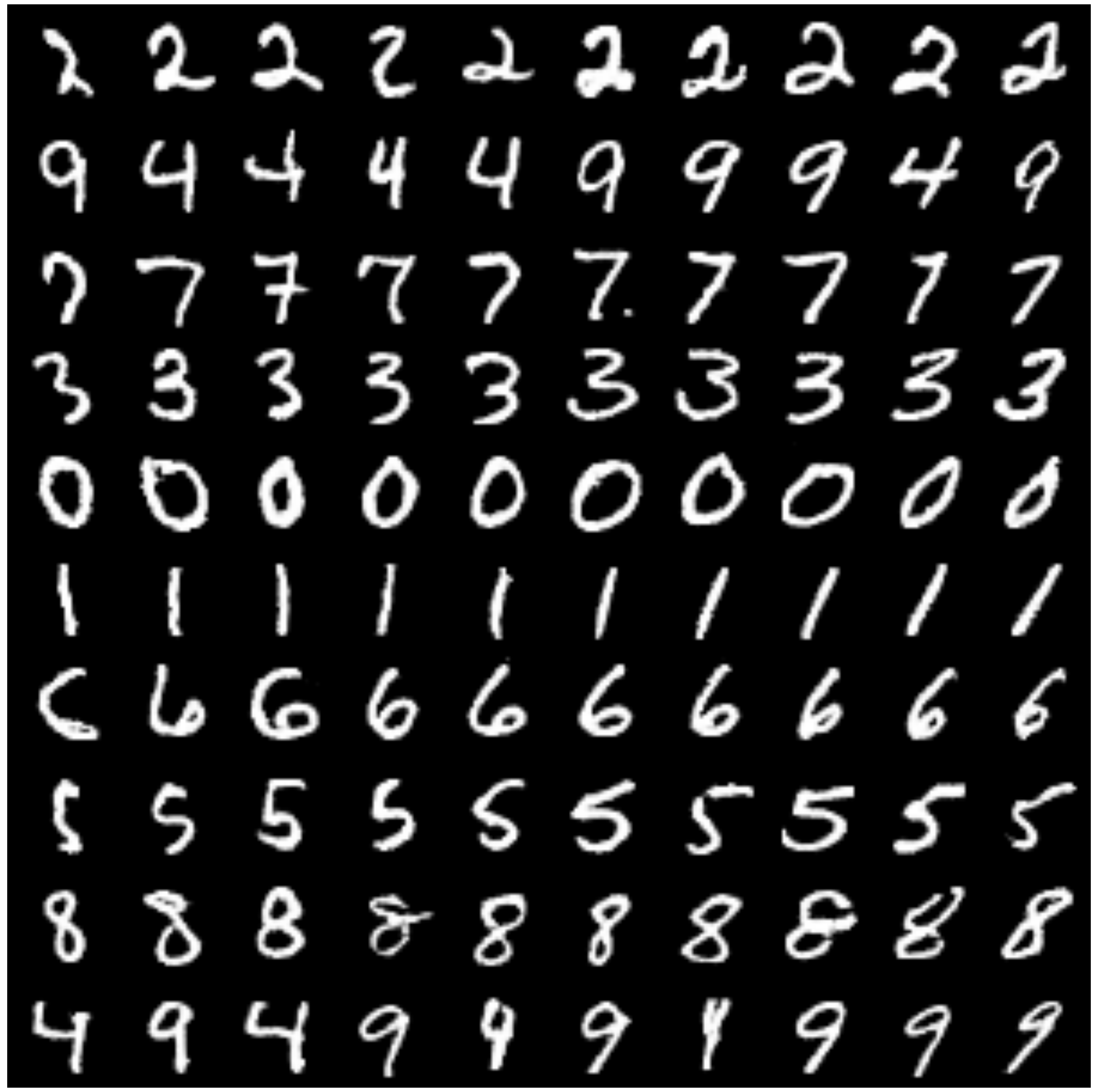
[G & Q]

```
# G and Q part
optimG.zero_grad()

sp_out = self.SP(fake_x)
probs_fake = self.D(sp_out)
label.data.fill_(1.0)

reconstruct_loss = criterionD(probs_fake, label)

q_logits, q_mu, q_var = self.Q(sp_out)
class_ = torch.LongTensor(idx).cuda()
target = Variable(class_)
dis_loss = criterionQ_dis(q_logits, target)
con_loss = criterionQ_con(con_c, q_mu, q_var)*0.1

G_loss = reconstruct_loss + dis_loss + con_loss
G_loss.backward()
optimG.step()
```

Results (100 epochs)



C1 Vector : Thickness



C2 Vector : Rotation

감사합니다
Q&A