비주얼 컴퓨팅 최신기술
기말과제 보고서
[GAN]

AI빅데이터융합경영학과 20172853 장성현

1. 기존 DCGAN 실습 코드를 참고하여 final_project_DCGAN.ipynb 완성하시오.

1-1 Generator (10pt)

```
class Generator(nn.Module):
    '''
    Generator Class
    Values:
        z_dim: Noise 벡터의 차원(scalar)
        im_chan: Output image의 채널 수(scalar)
              ( MNIST,Fashion MNIST : 1 / Cifar-10 : 3 )
        hidden_dim: hidden vector의 차원(scalar)
    '''
    def __init__(self, z_dim=10, im_chan=1, hidden_dim=64):
        super(Generator, self).__init__()
        self.z_dim = z_dim
        # Build the neural network
        self.gen = nn.Sequential(
            self.build_gen_block(z_dim, hidden_dim * 4),
            self.build_gen_block(hidden_dim * 4, hidden_dim * 2, kernel_size=4, stride=1),
            self.build_gen_block(hidden_dim * 2, hidden_dim),
            self.build_gen_block(hidden_dim, im_chan, kernel_size=4, final_layer=True),
        )

    def build_gen_block(self, input_channels, output_channels, kernel_size=3, stride=2, final_layer=False):
        '''
        Transposed convolution, Batch normalization, activation function을 이용하여,
        DCGAN의 생성자 블록과 일치하는 일련의 연산들을 반환하는 함수
        Parameters:
            input_channels: input feature의 차원(scalar)
            output_channels: output feature의 차원(scalar)
            kernel_size: convolution 필터 사이즈(scalar)
            stride: convolution 보폭(scalar)
            final_layer: boolean값으로, final layer일 경우 true, else false
        '''

        #     Steps(total : 10 points):
        #       1) 위에서 주어진 매개변수들을 사용하여 transposed convolution 수행
        #       2) final layer를 제외하고 batch norm 수행
        #       3) batch norm 이후 ReLU 활성화 함수 수행
        #       4) final layer의 경우, ReLU대신 Tanh 활성화 함수 수행
        # Build the neural block
        if not final_layer:
            return nn.Sequential(
                #### START CODE HERE(5 points) ####
                nn.ConvTranspose2d(input_channels, output_channels, kernel_size, stride),
                nn.BatchNorm2d(output_channels),
                nn.ReLU()
                #### END CODE HERE ####
            )
        else: # Final Layer
            return nn.Sequential(
                #### START CODE HERE(5 points) ####
                nn.ConvTranspose2d(input_channels, output_channels, kernel_size, stride),
                nn.Tanh()
                #### END CODE HERE ####
            )

    def unsqueeze_noise(self, noise):
        '''
        noise vector가 주어지면, width와 height가 1 그리고 channels값이 z_dim인 shape의 tensor를 반환하는 함수
        Parameters:
            noise shape : (n_samples, z_dim)
            output shape : (n_samples, z_dim, 1, 1)
        '''
        return noise.view(len(noise), self.z_dim, 1, 1)

    def forward(self, noise):
        '''
        Generator의 순전파 연산 함수
        noise 텐서를 입력으로 받아 생성된 이미지를 반환
        Parameters:
            noise: noise shape : (n_samples, z_dim)
        '''
        x = self.unsqueeze_noise(noise)
        return self.gen(x)

def get_noise(n_samples, z_dim, device='cpu'):
    '''
    noise 벡터 생성 함수
    주어진 파라미터를 입력받아 정규 분포로부터 난수들로 채워진 형태의 tensor 생성
    Parameters:
        n_samples: 생성하는 샘플들의 수(scalar)
        z_dim: noise vector의 차원(scalar)
        device: cpu / gpu
    '''
    return torch.randn(n_samples, z_dim, device=device)
```

1. 기존 DCGAN 실습 코드를 참고하여 final_project_DCGAN.ipynb 완성하시오.

1-2 Discriminator (10pt)

```python
class Discriminator(nn.Module):
    '''
    Discriminator Class
    Values:
        im_chan: Output image의 채널 수(scalar)
              ( MNIST,Fashion MNIST : 1 / Cifar-10 : 3 )
    hidden_dim: hidden vector의 차원(scalar)
    '''
    def __init__(self, im_chan=1, hidden_dim=16):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            self.build_disc_block(im_chan, hidden_dim),
            self.build_disc_block(hidden_dim, hidden_dim * 2),
            self.build_disc_block(hidden_dim * 2, 1, final_layer=True),
        )

    def build_disc_block(self, input_channels, output_channels, kernel_size=4, stride=2, final_layer=False):
        '''
        Function to return a sequence of operations corresponding to a discriminator block of DCGAN,
        corresponding to a convolution, a batchnorm (except for in the last layer), and an activation.
        Parameters:
            input_channels: input feature의 차원(scalar)
            output_channels: output feature의 차원(scalar)
            kernel_size: convolution 필터 사이즈(scalar)
            stride: convolution 보폭(scalar)
            final_layer: boolean값으로, final layer일 경우 true, else false
        '''
        #     Steps(total : 10 points):
        #       1) 위에서 주어진 매개변수들을 사용하여 convolution 수행
        #       2) final layer를 제외하고 batch norm 수행
        #       3) batch norm 이후 LeakyReLU 활성화 함수 수행. 이때 slope 0.2는 고정(Optional Hint 참고)
        #       4) 기존 DCGAN 실습과 달리 final layer에는 어떠한 activation을 적용하지 않음
```

```python
        # Build the neural block
        if not final_layer:
            return nn.Sequential(
                #### START CODE HERE(5 points) ####
                nn.Conv2d(input_channels, output_channels, kernel_size, stride),
                nn.BatchNorm2d(output_channels),
                nn.LeakyReLU(0.2, inplace=True)
                #### END CODE HERE ####
            )
        else: # Final Layer
            return nn.Sequential(
                #### START CODE HERE(5 points) ####
                nn.Conv2d(input_channels, output_channels, kernel_size, stride)
                #### END CODE HERE ####
            )
```

```python
    def forward(self, image):
        '''
        Discriminator의 순전파 연산 함수
        image 텐서를 입력으로 받아 real/fake를 나타내는 1D tensor를 반환
        Parameters:
            image: flattened 된 image tensor
        '''
        disc_pred = self.disc(image)
        return disc_pred.view(len(disc_pred), -1)
```

1. 기존 DCGAN 실습 코드를 참고하여 final_project_DCGAN.ipynb 완성하시오.

1-3 학습 과정 코드 주석 (10pt)

```python
# Training Process Cell
n_epochs = 50
cur_step = 0
mean_generator_loss_with_bce = 0
mean_discriminator_loss_with_bce = 0

mean_generator_loss_with_mse = 0
mean_discriminator_loss_with_mse = 0

for epoch in range(n_epochs): # n_epochs만큼 학습 진행
    # Dataloader returns the batches
    # dataloader의 mini batch 1개씩 학습
    for real, _ in tqdm(dataloader):
        # real의 길이를 통해 batch size 계산
        cur_batch_size = len(real)
        # real device에 할당
        real = real.to(device)

        ## Update discriminator ##
        disc_opt_with_bce.zero_grad()
        disc_opt_with_mse.zero_grad()

        # noise vector 생성
        fake_noise = get_noise(cur_batch_size, z_dim, device=device)

        # 생성한 noise vector를 입력하여 BCE Loss를 이용하는 Generator를 통해 fake 이미지를 생성
        fake_with_bce = gen_with_bce(fake_noise)
        # 위와 동일한 noise vector를 입력하여 MSE Loss를 이용하는 Generator로 fake 이미지를 생성
        fake_with_mse = gen_with_mse(fake_noise)

        # BCE Loss를 사용하는 Discriminator를 통해 BCE Loss를 사용하는 Generator로 생성된 fake 이미지의 진위 판별
        disc_fake_pred_with_bce = disc_with_bce(fake_with_bce.detach())
        # MSE Loss를 사용하는 Discriminator를 통해 MSE Loss를 사용하는 Generator로 생성된 fake 이미지의 진위 판별
        disc_fake_pred_with_mse = disc_with_mse(fake_with_mse.detach())

        # BCE Loss를 사용하는 Discriminator를 통해 fake 이미지의 Loss 계산
        disc_fake_loss_with_bce = bce_loss(disc_fake_pred_with_bce, torch.zeros_like(disc_fake_pred_with_bce))
        # MSE Loss를 사용하는 Discriminator를 통해 fake 이미지의 Loss 계산
        disc_fake_loss_with_mse = mse_loss(disc_fake_pred_with_mse, torch.zeros_like(disc_fake_pred_with_mse))

        # BCE Loss를 사용하는 Discriminator를 통해 real 이미지 진위 판별
        disc_real_pred_with_bce = disc_with_bce(real)
        # MSE Loss를 사용하는 Discriminator를 통해 real 이미지 진위 판별
        disc_real_pred_with_mse = disc_with_mse(real)
```

```python
        # BCE Loss를 사용하는 Discriminator를 통해 real 이미지의 Loss 계산
        disc_real_loss_with_bce = bce_loss(disc_real_pred_with_bce, torch.ones_like(disc_real_pred_with_bce))
        # fake 이미지, real 이미지로 구한 BCE Loss들의 평균 계산
        disc_loss_with_bce = (disc_fake_loss_with_bce + disc_real_loss_with_bce) / 2

        # MSE Loss를 사용하는 Discriminator를 통해 real 이미지의 Loss 계산
        disc_real_loss_with_mse = mse_loss(disc_real_pred_with_mse, torch.ones_like(disc_real_pred_with_mse))
        # fake 이미지, real 이미지로 구한 MSE Loss들의 평균 계산
        disc_loss_with_mse = (disc_fake_loss_with_mse + disc_real_loss_with_mse) / 2

        # Keep track of the average discriminator loss
        mean_discriminator_loss_with_bce += disc_loss_with_bce.item() / display_step
        mean_discriminator_loss_with_mse += disc_loss_with_mse.item() / display_step

        # Update gradients

        # BCE Loss를 사용하는 Discriminator 변화율을 역전파(backward)를 이용해 계산
        disc_loss_with_bce.backward(retain_graph=True)
        # MSE Loss를 사용하는 Discriminator 변화율을 역전파(backward)를 이용해 계산
        disc_loss_with_mse.backward(retain_graph=True)

        # Update optimizer
        # BCE Loss를 사용하는 Discriminator 업데이트
        disc_opt_with_bce.step()
        # MSE Loss를 사용하는 Discriminator 업데이트
        disc_opt_with_mse.step()

        ## Update generator ##
        gen_opt_with_bce.zero_grad()
        gen_opt_with_mse.zero_grad()

        # noise vector 생성
        fake_noise_2 = get_noise(cur_batch_size, z_dim, device=device)

        # 생성한 noise vector으로 BCE Loss를 사용하는 Generator로 fake 이미지 생성
        fake_2_with_bce = gen_with_bce(fake_noise_2)
        # 생성한 noise vector으로 MSE Loss를 사용하는 Generator로 fake 이미지 생성
        fake_2_with_mse = gen_with_mse(fake_noise_2)

        # Update된 BCE Loss를 사용하는 Discriminator를 통해 fake 이미지의 진위 판별
        disc_fake_pred_with_bce = disc_with_bce(fake_2_with_bce)
        # Update된 MSE Loss를 사용하는 Discriminator를 통해 fake 이미지의 진위 판별
        disc_fake_pred_with_mse = disc_with_mse(fake_2_with_mse)
```

1. 기존 DCGAN 실습 코드를 참고하여 final_project_DCGAN.ipynb 완성하시오.

1-3 학습 과정 코드 주석 (10pt)

```
        # BCE Loss를 사용하는 Generator를 통해 fake 이미지의 Loss 계산
        gen_loss_with_bce = bce_loss(disc_fake_pred_with_bce, torch.ones_like(disc_fake_pred_with_bce))
        # BCE Loss를 사용하는 Generator 변화율을 역전파(backward)를 이용해 계산
        gen_loss_with_bce.backward()
        # BCE Loss를 사용하는 Generator 업데이트
        gen_opt_with_bce.step()

        # MSE Loss를 사용하는 Generator를 통해 fake 이미지의 Loss 계산
        gen_loss_with_mse = mse_loss(disc_fake_pred_with_mse, torch.ones_like(disc_fake_pred_with_mse))
        # MSE Loss를 사용하는 Generator 변화율을 역전파(backward)를 이용해 계산
        gen_loss_with_mse.backward()
        # MSE Loss를 사용하는 Generator 업데이트
        gen_opt_with_mse.step()

        # Keep track of the average generator loss
        mean_generator_loss_with_bce += gen_loss_with_bce.item() / display_step
        mean_generator_loss_with_mse += gen_loss_with_mse.item() / display_step

        ## Visualization code ##
        if cur_step % display_step == 0 and cur_step > 0:
            # BCE Loss 출력 코드
            print(f"Step {cur_step}: Generator loss(BCE): {mean_generator_loss_with_bce}, Discriminator loss(BCE): {mean_discriminator_loss_with_bce}")
            # MSE Loss 출력 코드
            print(f"Step {cur_step}: Generator loss(MSE): {mean_generator_loss_with_mse}, Discriminator loss(MSE): {mean_discriminator_loss_with_mse}")
            # BCE Loss를 이용해 학습한 fake 이미지 시각화
            show_tensor_images(fake_with_bce, 'fake(bce)')
            # MSE Loss를 이용해 학습한 fake 이미지 시각화
            show_tensor_images(fake_with_mse, 'fake(mse)')
            # real 이미지 시각화
            show_tensor_images(real, 'real')

            # 평균 Loss값 초기화
            mean_generator_loss_with_bce = 0
            mean_generator_loss_with_mse = 0
            mean_discriminator_loss_with_bce = 0
            mean_discriminator_loss_with_mse = 0

        cur_step += 1
```

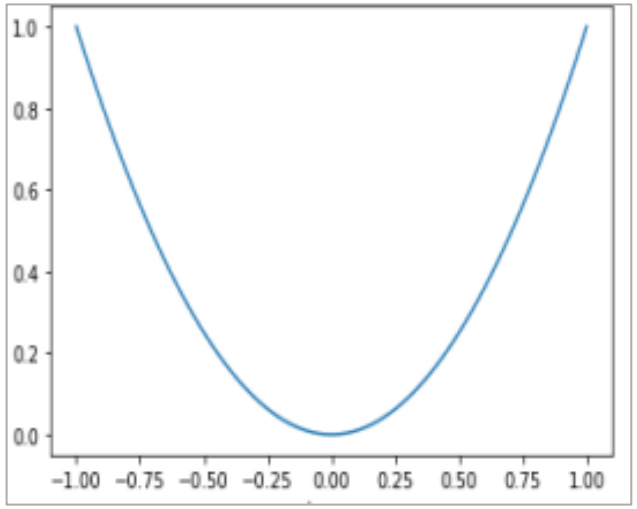
1. 기존 DCGAN 실습 코드를 참고하여 final_project_DCGAN.ipynb 완성하시오.

1-4 두 개의 Loss 함수 적용시 비교 결과 분석 (10pt)

- MSE Loss**(Mean Squared Error)**

평균제곱오차 손실의 정의는 간단합니다. 출력 노드에서 나온 값과 원하는 목푯값 사이의 차이를 계산하면 됩니다. 이 오차를 제곱한다면 값은 항상 양수입니다. 평균제곱오차는 이 제곱한 오차들의 평균입니다.

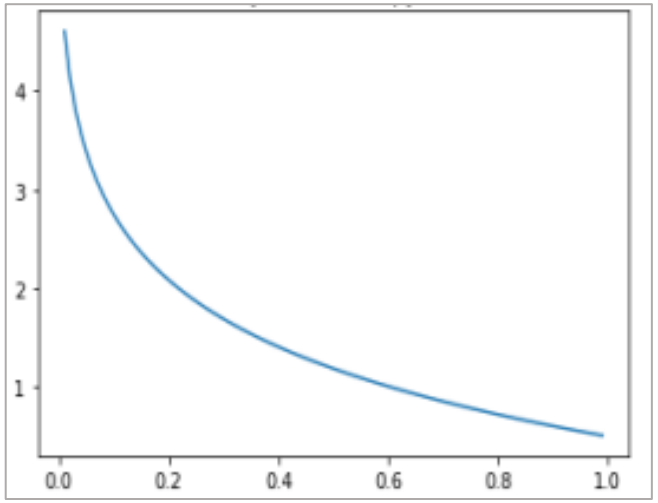
$$L = \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2 \quad \text{...(MSE)}$$



- BCE Loss**(Binary Cross Entropy)**

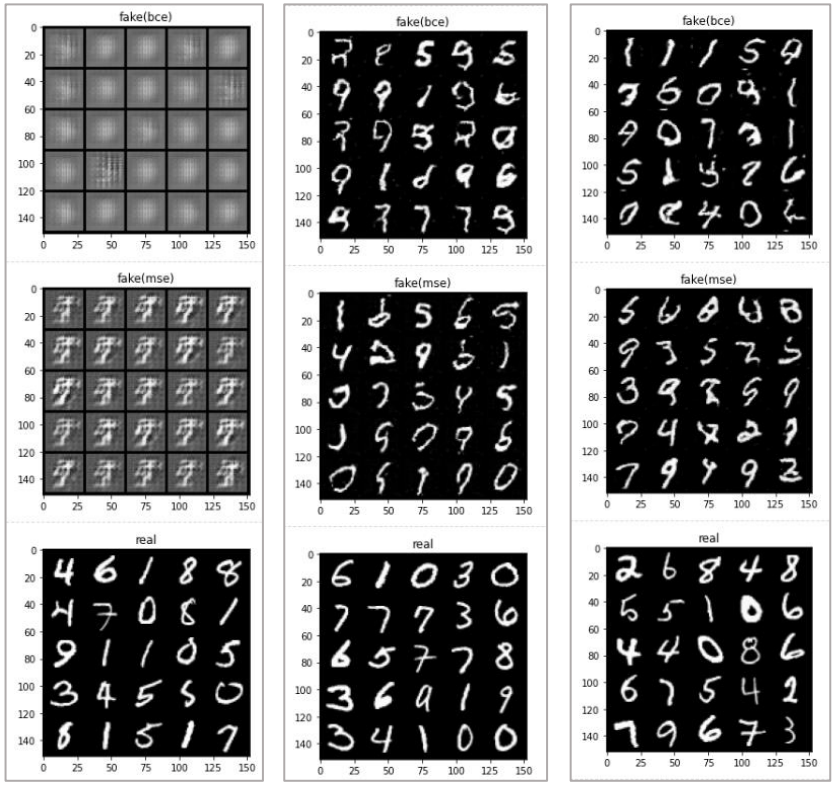
이진 교차 엔트로피 손실은 확률과 불확실성에 기반을 둡니다.
이는 '가까운'을 판단하는 척도, 즉 두 가지가 얼마나 다른지 판단하는 방법이 필요할 때 사용하는 Loss Function입니다.

$$L = -\frac{1}{N}\sum_{i=1}^{N} t_i log(y_i) + (1 - t_i) log(1 - y_i) \quad \text{...(binary crossentropy)}$$



1. 기존 DCGAN 실습 코드를 참고하여 final_project_DCGAN.ipynb 완성하시오.

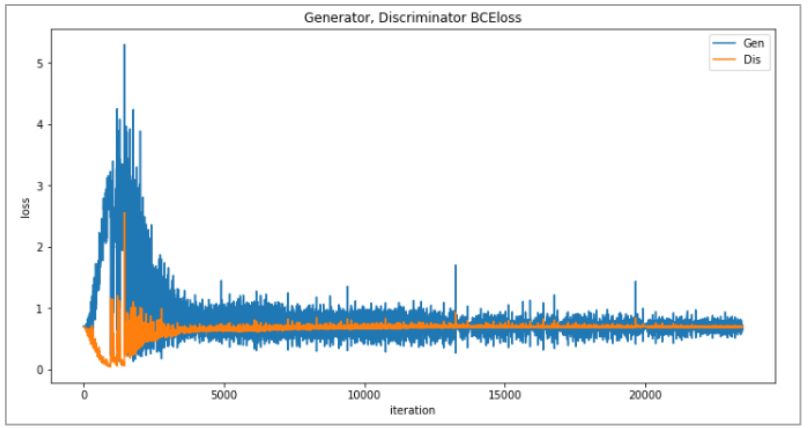
1-4 두 개의 Loss 함수 적용시 비교 결과 분석 (10pt)



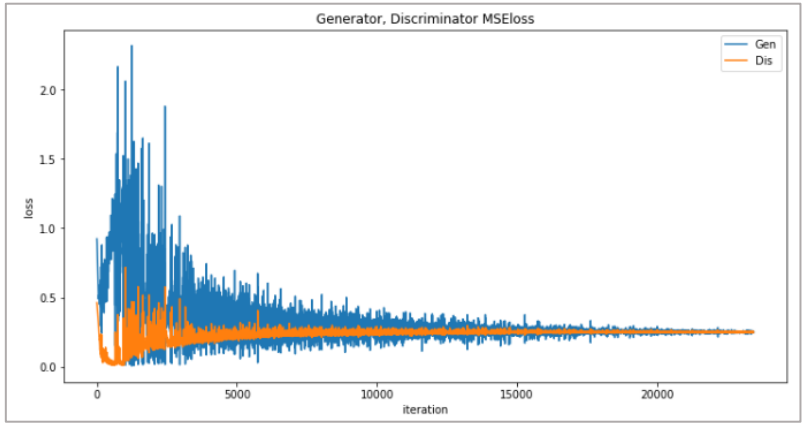
1000 Step 10000 Step 23000 Step

두 가지 Loss Function으로 실험을 진행한 결과를 캡쳐하였습니다.

모두 학습이 진행될수록 생성 이미지의 선명도 및 정교함이 실제 데이터에 가까워지는 것을 볼 수 있었습니다.



BCE Loss



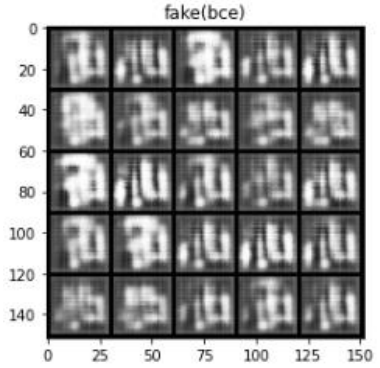
MSE Loss

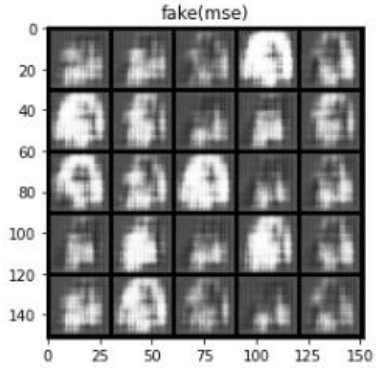
두 가지 Loss Function에 대한 시각화 결과를 캡쳐하였습니다.

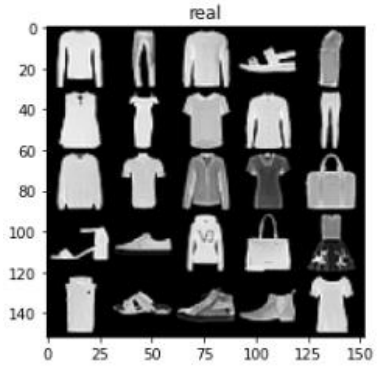
두 그래프의 차이를 보면 y축 값이 BCE Loss는 0~5인 반면, MSE Loss는 0~2.0으로 MSE가 더 적합하고 안정적인 Loss Function이라는 것을 알 수 있습니다.

2. 완성된 final_project_DCGAN.ipynb를 이용하여, Fashin MNIST 혹은 Cifar-10 데이터에 대하여 학습을 진행한 후 결과를 확인하시오.

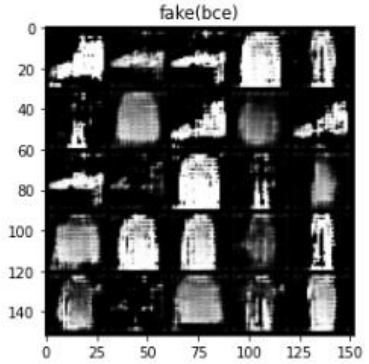
```
Step 1000: Generator loss(BCE): 1.3724676621556289, Discriminator loss(BCE): 0.4340427639186381
Step 1000: Generator loss(MSE): 0.5354167674507481, Discriminator loss(MSE): 0.1598489246144891
```

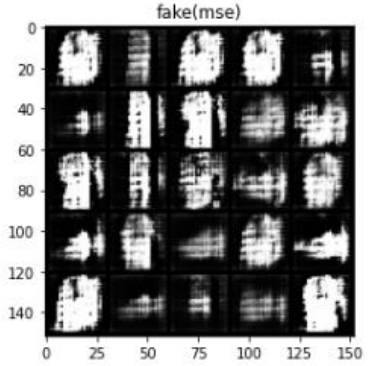


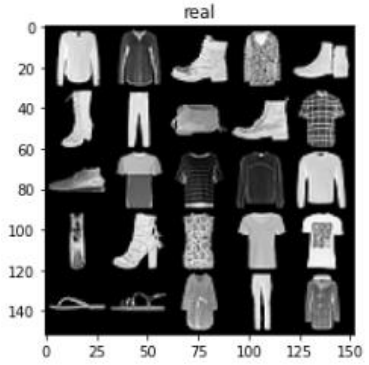





```
Step 23000: Generator loss(BCE): 0.6949691865444187, Discriminator loss(BCE): 0.6936412296295168
Step 23000: Generator loss(MSE): 0.25117187547683684, Discriminator loss(MSE): 0.2502024991810322
```







1번에서 학습을 진행하였던 코드를 이용하여 Fashion MNIST 데이터로 학습을 진행해보았습니다. 학습이 진행될수록 표현이 더욱 정교하고 선명해지는 것을 직접 확인할 수 있었습니다.

3. final_project_CycleGAN.ipynb 실습 코드를 완성하시오.

3-1 Discriminator Loss (10pt)

```
# UNQ_C1 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: get_disc_loss
def get_disc_loss(real_X, fake_X, disc_X, adv_criterion):
    '''
    주어진 입력들로부터 Discriminator의 loss값 반환

    Parameters:
        real_X: Pile X로부터 실제 이미지
        fake_X: class X의 생성된 이미지
        disc_X: class X의 Discriminator; 이미지들을 가지고 class X가 진짜인지 가짜인지 반환하는
                예측한 행렬들
        adv_criterion: adversarial loss 함수; 식별자로 예측한 값과 target label을 사용해
                       adversarial loss 반환
    '''
    #     Steps:
    #       1) fake_X를 입력으로 class X의 Discriminator를 이용하여 'disc_fake_X_hat' 생성
    #       2) adv_criterion을 사용하여 disc_fake_X_hat과 false label의 손실값 계산하여 'disc_fake_X_loss' 변수에 저장
    #       3) real_X를 입력으로 class X의 Discriminator를 이용하여 'disc_real_X_hat' 생성
    #       4) adv_criterion을 사용하여 disc_real_X_hat과 true label의 손실값 계산하여 'disc_real_X_loss' 변수에 저장

    #### START CODE HERE (10 points) ####
    disc_fake_X_hat = disc_X(fake_X)
    disc_fake_X_loss = adv_criterion(disc_fake_X_hat, torch.zeros_like(disc_fake_X_hat))
    disc_real_X_hat = disc_X(real_X)
    disc_real_X_loss = adv_criterion(disc_real_X_hat, torch.ones_like(disc_real_X_hat))

    #### END CODE HERE ####
    disc_loss = (disc_fake_X_loss + disc_real_X_loss) / 2
    return disc_loss
```

3. final_project_CycleGAN.ipynb 실습 코드를 완성하시오.

3-2 Adversarial Loss (10pt)

```
# UNQ_C2 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: get_gen_adversarial_loss
def get_gen_adversarial_loss(real_X, disc_Y, gen_XY, adv_criterion):
    '''
    주어진 입력들로부터 생성자의 adversarial loss값을 반환

    Parameters:
        real_X: pile X로부터 실제 이미지들
        disc_Y: class Y의 식별자; 이미지들을 가지고 class Y가 진짜인지 가짜인지 반환
        gen_XY: class X에서 Y로의 생성자; 이미지들을 가지고 class Y로 변형된 이미지들 반환
        adv_criterion: adversarial loss 함수; 식별자로 예측된 값과 target label들을 가지고 adversarial loss값을 반환
    '''
    #     Steps:
    #       1) real_X를 입력으로 class X에서 Y로의 생성자를 이용하여 'fake_Y' 이미지 생성
    #       2) fake_Y를 입력으로 class Y의 식별자를 이용하여 'disc_fake_Y_hat' 생성
    #       3) adv_criterion을 사용하여 disc_fake_Y_hat과 real label의 손실값 계산하여 'adversarial_loss' 변수에 저장

    #### START CODE HERE (10 points) ####
    fake_Y = gen_XY(real_X)
    disc_fake_Y_hat = disc_Y(fake_Y)
    adversarial_loss = adv_criterion(disc_fake_Y_hat, torch.ones_like(disc_fake_Y_hat))

    #### END CODE HERE ####
    return adversarial_loss, fake_Y
```

3. final_project_CycleGAN.ipynb 실습 코드를 완성하시오.

3-3 Identity Loss (10pt)

```
# UNQ_C3 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: get_identity_loss
def get_identity_loss(real_X, gen_YX, identity_criterion):
    '''
    주어진 입력들로부터 생성자의 identity loss 반환

    Parameters:
        real_X: pile X로부터 실제 이미지들
        gen_YX: class Y에서 X로의 생성기; 이미지들을 가지고 class X로 변형된 이미지들 반환
        identity_criterion: identity loss 함수; X로부터 실제 이미지를 가져오고,
                            그 이미지들을 Y->X 생성기를 통과시키고,
                            identity loss를 반환
    '''
    #     Steps:
    #       1) real_X를 입력으로 class Y에서 X로의 생성자를 이용하여 'identity_X' 이미지 생성
    #       2) identity_criterion을 사용하여 identity_X와 real_X의 손실값 계산하여 'identity_loss' 변수에 저장

    #### START CODE HERE (10 points) ####
    identity_X = gen_YX(real_X)
    identity_loss = identity_criterion(identity_X, real_X)

    #### END CODE HERE ####
    return identity_loss, identity_X
```

3. final_project_CycleGAN.ipynb 실습 코드를 완성하시오.

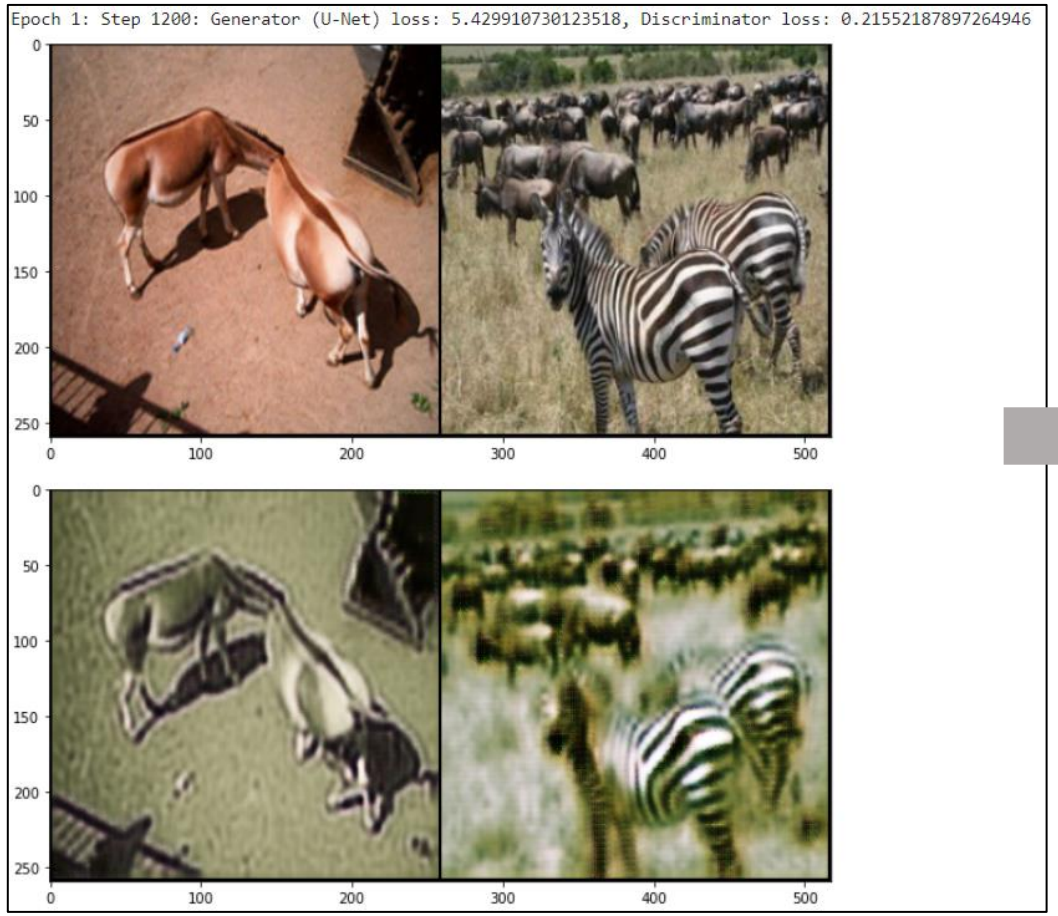
3-4 Cycle Consistency Loss (10pt)1-1 Generator (10pt)

```
# UNQ_C4 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: get_cycle_consistency_loss
def get_cycle_consistency_loss(real_X, fake_Y, gen_YX, cycle_criterion):
    '''
    주어진 입력들로부터 생성자의 cycle consistency loss값을 반환

    Parameters:
        real_X: pile X로부터 실제 이미지들
        fake_Y: class Y의 생성된 이미지들
        gen_YX: class Y에서 X로의 생성기; 이미지들을 가지고 class X로 변형된 이미지들 반환
        cycle_criterion: cycle consistency loss 함수; X로부터 실제 이미지들을 가져오고
                         그 이미지들을 X->Y 생성자에 통과시키고 Y->X 생성자에 통과한
                         cycle consistency loss값을 반환
    '''
    #     Steps:
    #       1) fake_Y를 입력으로 class Y에서 X로의 생성자를 이용하여 'cycle_X' 이미지 생성
    #       2) cycle_criterion을 사용하여 cycle_X와 real_X의 손실값 계산하여 'cycle_loss' 변수에 저장

    #### START CODE HERE (10 points) ####
    cycle_X = gen_YX(fake_Y)
    cycle_loss = cycle_criterion(cycle_X, real_X)

    #### END CODE HERE ####
    return cycle_loss, cycle_X
```

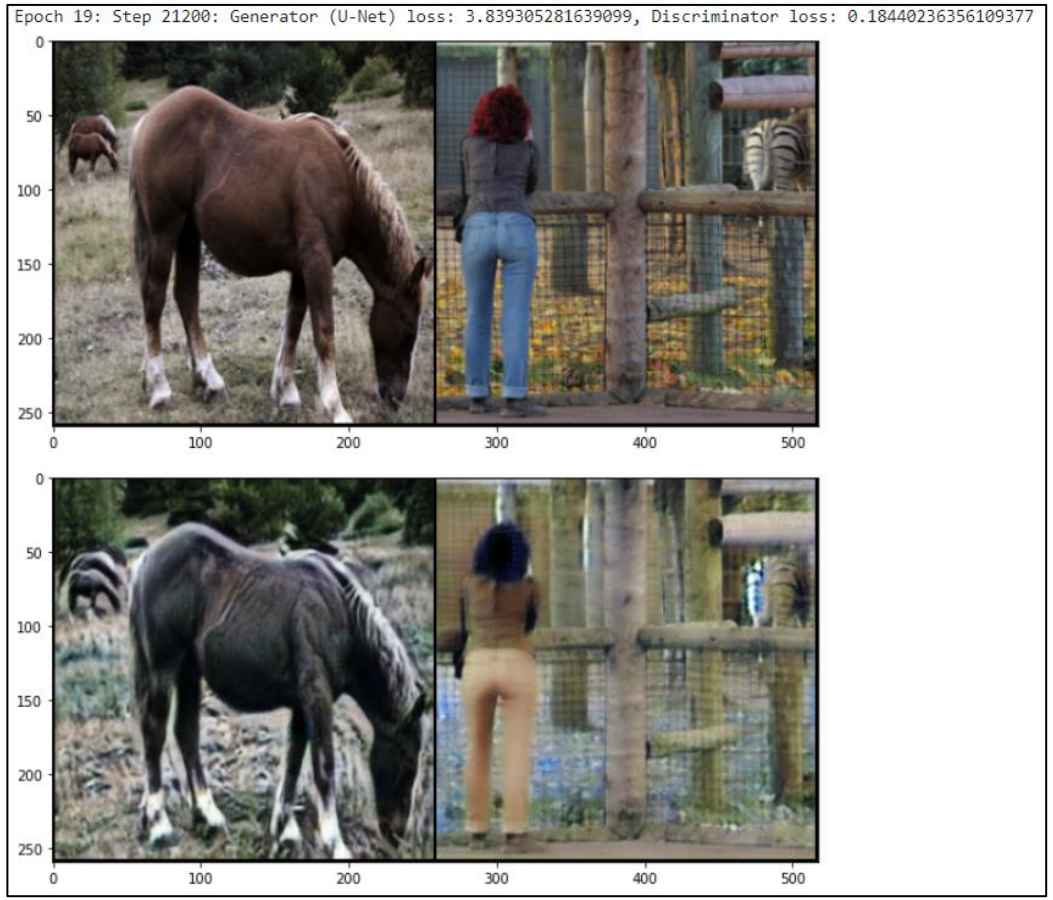
3. final_project_CycleGAN.ipynb 실습 코드를 완성하시오.

3-5 결과물

1200 Step



21200 Step



계속 학습할수록 생성되는 그림이 선명하고 정교해지는 것을 볼 수 있습니다!

감사합니다