In [ ]:

```
```

201600779 김영민

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [11]:

```
import h5py
import zipfile
import imageio
import os
import numpy as np
import matplotlib.pyplot as plt
import random
# 라이브러리 임포트

import torch
import torch.nn as nn
from torch.utils.data import Dataset

import pandas
```

In [17]:

```
%%time

# HDF5 패키지가 위치하는 경로
hdf5_file = '/content/drive/MyDrive/GAN_basic/celeba_aligned_small.h5py'

# 몇개 이미지를 HDF5로 패키징할지 설정
total_images = 20000

with h5py.File(hdf5_file, 'w') as hf:

    count = 0

    with zipfile.ZipFile('/content/drive/MyDrive/GAN_basic/img_align_celeba_20000.zip', 'r') as zf:
      for i in zf.namelist():
        if (i[-4:] == '.jpg'):
          # 이미지 추출
          ofile = zf.extract(i)
          img = imageio.imread(ofile)
          os.remove(ofile)

          # 이미지 데이터를 HDF5 파일에 새로운 이름으로 추가
          hf.create_dataset('img_align_celeba/'+str(count)+'.jpg', data=img, compression="gzip", com

          count = count + 1
          if (count%1000 == 0):
            print("images done .. ", count)
            pass

          # total_images 수만큼만 추가
          if (count == total_images):
            break
          pass

        pass
      pass
```

```
images done ..  1000
images done ..  2000
images done ..  3000
images done ..  4000
images done ..  5000
images done ..  6000
images done ..  7000
images done ..  8000
images done ..  9000
images done ..  10000
images done ..  11000
images done ..  12000
images done ..  13000
images done ..  14000
images done ..  15000
images done ..  16000
images done ..  17000
images done ..  18000
images done ..  19000
images done ..  20000
CPU times: user 3min 9s, sys: 12.6 s, total: 3min 22s
Wall time: 4min 13s
```

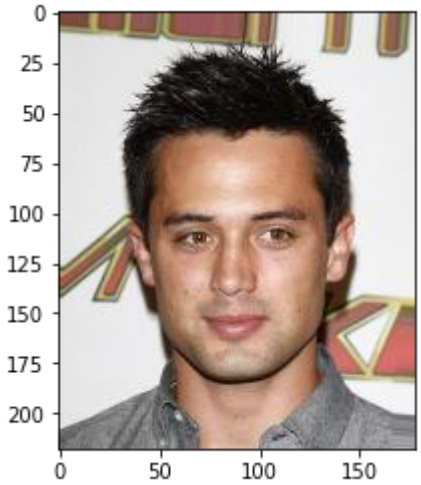
In [12]:

```
with h5py.File('/content/drive/MyDrive/GAN_basic/celeba_aligned_small.h5py', 'r') as file_object:

  for group in file_object:
    print(group)
    pass
```

```
img_align_celeba
```

In [13]:

```
with h5py.File('/content/drive/MyDrive/GAN_basic/celeba_aligned_small.h5py', 'r') as file_object:

  dataset = file_object['img_align_celeba']

  image = np.array(dataset['6.jpg'])
  plt.imshow(image, interpolation='none')
  pass
```



In [14]:

```
# CUDA가 가능한지 확인
# 가능하다면, cuda에 기본 형태를 설정

if torch.cuda.is_available():
  torch.set_default_tensor_type(torch.cuda.FloatTensor)
  print("using cuda:", torch.cuda.get_device_name(0))
  pass

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

device
```

```
using cuda: Tesla K80
```

Out[14]:

```
device(type='cuda')
```

In [19]:

```
# 데이터셋 클래스

class CelebADataset(Dataset):

    def __init__(self, file):
        self.file_object = h5py.File(file, 'r')
        self.dataset = self.file_object['img_align_celeba']
        pass

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, index):
        if (index >= len(self.dataset)):
          raise IndexError()
        img = np.array(self.dataset[str(index)+'.jpg'])
        return torch.cuda.FloatTensor(img) / 255.0

    def plot_image(self, index):
        plt.imshow(np.array(self.dataset[str(index)+'.jpg']), interpolation='nearest')
        pass

    pass
```

In [20]:

```
# Dataset 객체 생성

celeba_dataset = CelebADataset('/content/drive/MyDrive/GAN_basic/celeba_aligned_small.h5py')
```

In [21]:

```
celeba_dataset.plot_image(43)
```

In [22]:

```
def generate_random_image(size):
    random_data = torch.rand(size)
    return random_data


def generate_random_seed(size):
    random_data = torch.randn(size)
    return random_data
```

In [23]:

```
class View(nn.Module):
    def __init__(self, shape):
        super().__init__()
        self.shape = shape,

    def forward(self, x):
        return x.view(*self.shape)
```

In [24]:

```
class Discriminator(nn.Module):

    def __init__(self):
        # 파이토치 부모 클래스 초기화
        super().__init__()

       # 신경망 레이어 정의
        self.model = nn.Sequential(
            View(218*178*3),

            nn.Linear(3*218*178, 100),
            nn.LeakyReLU(),

            nn.LayerNorm(100),

            nn.Linear(100, 1),
            nn.Sigmoid()
        )

        # 손실 함수 생성
        self.loss_function = nn.BCELoss()

        # 옵티마이저 생성
        self.optimiser = torch.optim.Adam(self.parameters(), lr=0.0001)

        # 진행 측정을 위한 변수 초기화
        self.counter = 0;
        self.progress = []

        pass


    def forward(self, inputs):
        # 모델 실행
        return self.model(inputs)


    def train(self, inputs, targets):
        # 신경망 출력 계산
        outputs = self.forward(inputs)

        # 손실 계산
        loss = self.loss_function(outputs, targets)

        # 매 10회마다 에러를 누적하고 카운터를 증가
        self.counter += 1;
        if (self.counter % 10 == 0):
            self.progress.append(loss.item())
            pass
        if (self.counter % 1000 == 0):
            print("counter = ", self.counter)
            pass

        # 기울기를 초기화 하고 역전파 후 가중치 갱신
        self.optimiser.zero_grad()
        loss.backward()
        self.optimiser.step()

        pass
```

```
    def plot_progress(self):
        df = pandas.DataFrame(self.progress, columns=['loss'])
        df.plot(ylim=(0), figsize=(16,8), alpha=0.1, marker='.', grid=True, yticks=(0, 0.25, 0.5, 1.
        pass

    pass
```

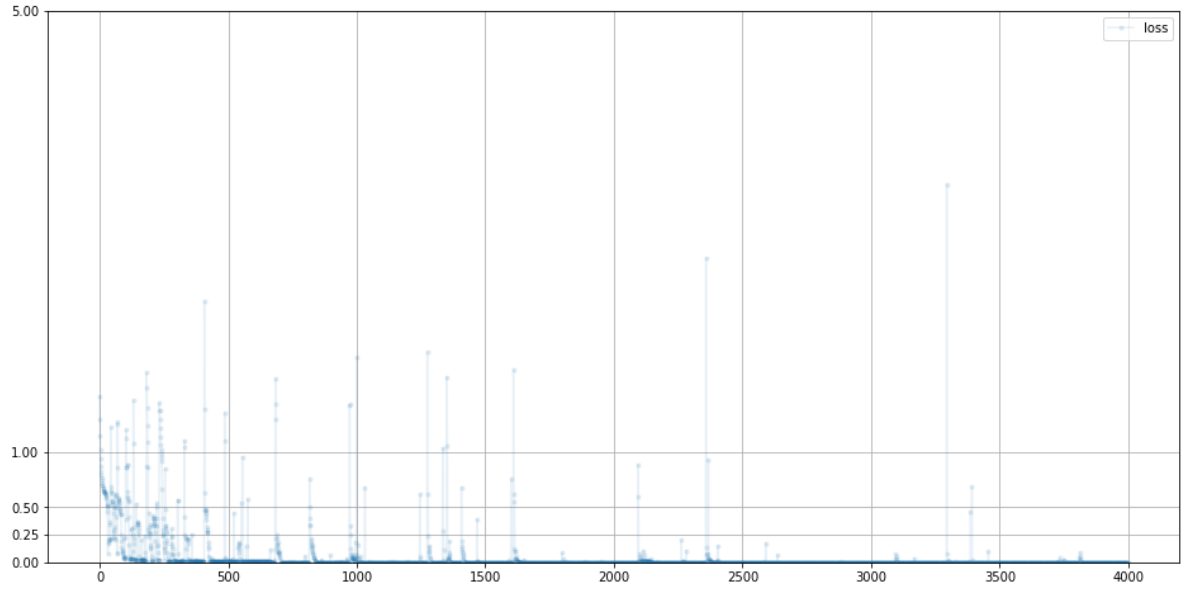
In [25]:

```
%%time
# 판별기가 임의의 노이즈와 실제 데이터를 구별할 수 있는지 테스트

D = Discriminator()
# 모델을 cuda로 배치
D.to(device)

for image_data_tensor in celeba_dataset:
    # 실제 데이터
    D.train(image_data_tensor, torch.cuda.FloatTensor([1.0]))
    # 생성된 데이터
    D.train(generate_random_image((218,178,3)), torch.cuda.FloatTensor([0.0]))
    pass
```

```
counter =  1000
counter =  2000
counter =  3000
counter =  4000
counter =  5000
counter =  6000
counter =  7000
counter =  8000
counter =  9000
counter =  10000
counter =  11000
counter =  12000
counter =  13000
counter =  14000
counter =  15000
counter =  16000
counter =  17000
counter =  18000
counter =  19000
counter =  20000
counter =  21000
counter =  22000
counter =  23000
counter =  24000
counter =  25000
counter =  26000
counter =  27000
counter =  28000
counter =  29000
counter =  30000
counter =  31000
counter =  32000
counter =  33000
counter =  34000
counter =  35000
counter =  36000
counter =  37000
counter =  38000
counter =  39000
counter =  40000
CPU times: user 5min 41s, sys: 6.73 s, total: 5min 48s
Wall time: 5min 53s
```

In [26]:

```
D.plot_progress()
```



In [27]:

```
for i in range(4):
  image_data_tensor = celeba_dataset[random.randint(0,20000)]
  print( D.forward( image_data_tensor ).item() )
  pass

for i in range(4):
  print( D.forward( generate_random_image((218,178,3))).item() )
  pass
```

```
0.9997747540473938
0.9990861415863037
0.9999809265136719
0.9999529123306274
4.191141852061264e-05
3.912367537850514e-05
3.970282705267891e-05
6.213576125446707e-05
```

In [28]:

```
class Generator(nn.Module):

    def __init__(self):
        # 파이토치 부모 클래스 초기화
        super().__init__()

        # 신경망 레이어 정의
        self.model = nn.Sequential(
            nn.Linear(100, 3*10*10),
            nn.LeakyReLU(),

            nn.LayerNorm(3*10*10),

            nn.Linear(3*10*10, 3*218*178),

            nn.Sigmoid(),
            View((218,178,3))
        )

        # 옵티마이저 생성
        self.optimiser = torch.optim.Adam(self.parameters(), lr=0.0001)

        # 진행 측정을 위한 변수 초기화
        self.counter = 0;
        self.progress = []

        pass


    def forward(self, inputs):
        # 모델 실행
        return self.model(inputs)


    def train(self, D, inputs, targets):
        # 신경망 출력 계산
        g_output = self.forward(inputs)

        # 판별기에 값 전달
        d_output = D.forward(g_output)

        # 오차 계산
        loss = D.loss_function(d_output, targets)

        # 매 10회마다 에러를 누적하고 카운터를 증가
        self.counter += 1;
        if (self.counter % 10 == 0):
            self.progress.append(loss.item())
            pass

        # 기울기를 초기화 하고 역전파 후 가중치 갱신
        self.optimiser.zero_grad()
        loss.backward()
        self.optimiser.step()

        pass


    def plot_progress(self):
```

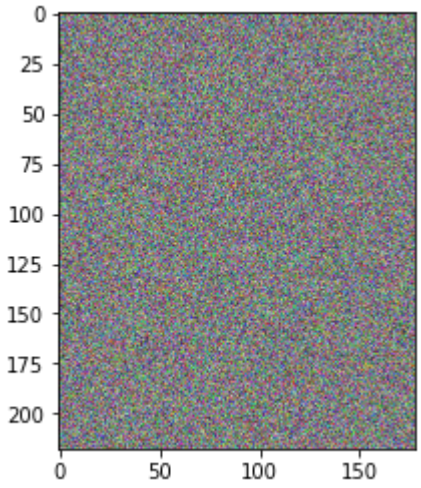
```
        df = pandas.DataFrame(self.progress, columns=['loss'])
        df.plot(ylim=(0), figsize=(16,8), alpha=0.1, marker='.', grid=True, yticks=(0, 0.25, 0.5, 1.
        pass

    pass
```

In [29]:

```
# 생성기의 출력이 올바른 타입과 형태를 지니고 있는지 확인

G = Generator()
# move model to cuda device
G.to(device)

output = G.forward(generate_random_seed(100))

img = output.detach().cpu().numpy()

plt.imshow(img, interpolation='none', cmap='Blues')
```

Out[29]:

<matplotlib.image.AxesImage at 0x7f48fef0e450>



In [30]:

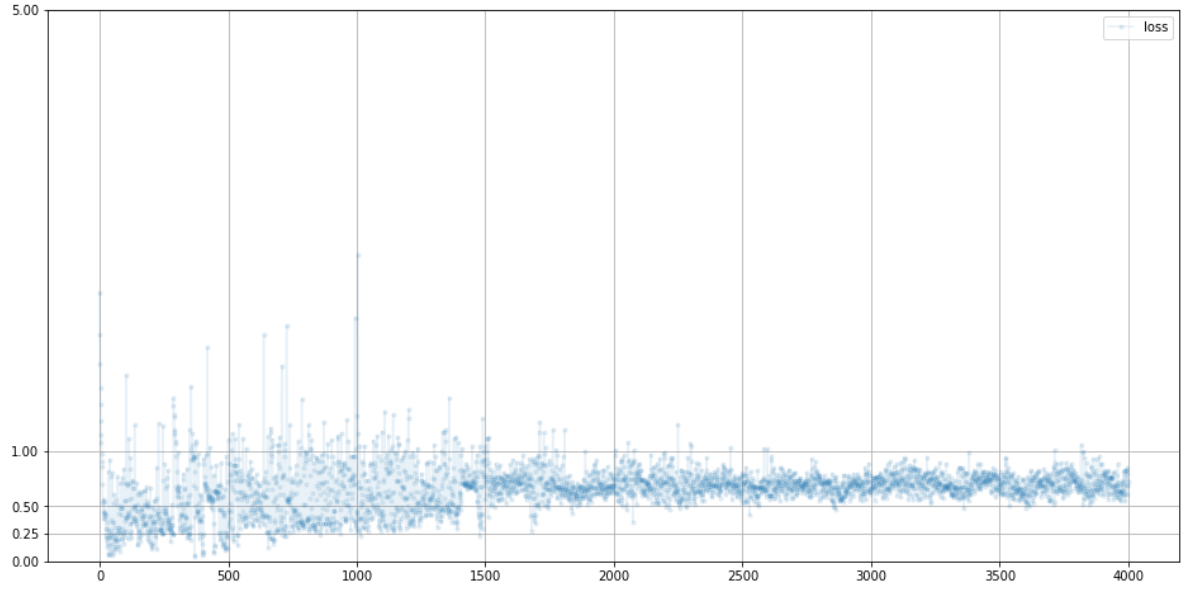
```
%%time

# 판별기 및 생성기 생성

D = Discriminator()
D.to(device)
G = Generator()
G.to(device)

epochs = 1

for epoch in range(epochs):
  print ("epoch = ", epoch + 1)

  # 판별기와 생성기 훈련

  for image_data_tensor in celeba_dataset:
    # 참일 경우 판별기 훈련
    D.train(image_data_tensor, torch.cuda.FloatTensor([1.0]))

    # 거짓일 경우 판별기 훈련
    # G의 기울기가 계산되지 않도록 detach() 함수를 이용
    D.train(G.forward(generate_random_seed(100)).detach(), torch.cuda.FloatTensor([0.0]))

    # 생성기 훈련
    G.train(D, generate_random_seed(100), torch.cuda.FloatTensor([1.0]))

    pass

  pass
```

```
epoch =  1
counter =  1000
counter =  2000
counter =  3000
counter =  4000
counter =  5000
counter =  6000
counter =  7000
counter =  8000
counter =  9000
counter =  10000
counter =  11000
counter =  12000
counter =  13000
counter =  14000
counter =  15000
counter =  16000
counter =  17000
counter =  18000
counter =  19000
counter =  20000
counter =  21000
counter =  22000
counter =  23000
counter =  24000
counter =  25000
counter =  26000
```

```
counter =  27000
counter =  28000
counter =  29000
counter =  30000
counter =  31000
counter =  32000
counter =  33000
counter =  34000
counter =  35000
counter =  36000
counter =  37000
counter =  38000
counter =  39000
counter =  40000
CPU times: user 15min 10s, sys: 7.82 s, total: 15min 18s
Wall time: 15min 16s
```

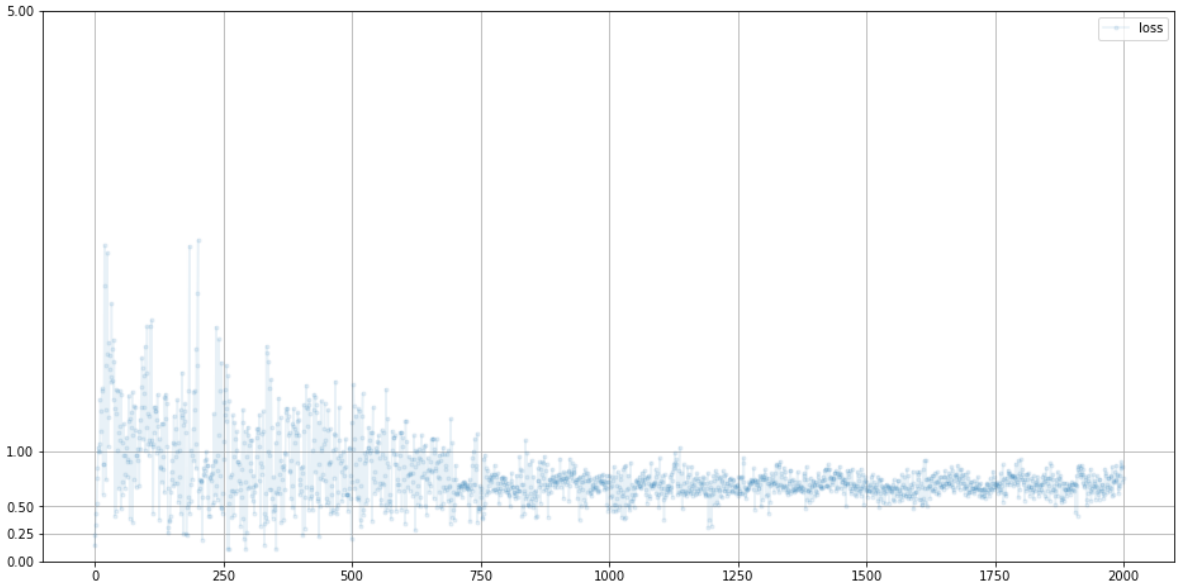
In [32]:

```
D.plot_progress()
```



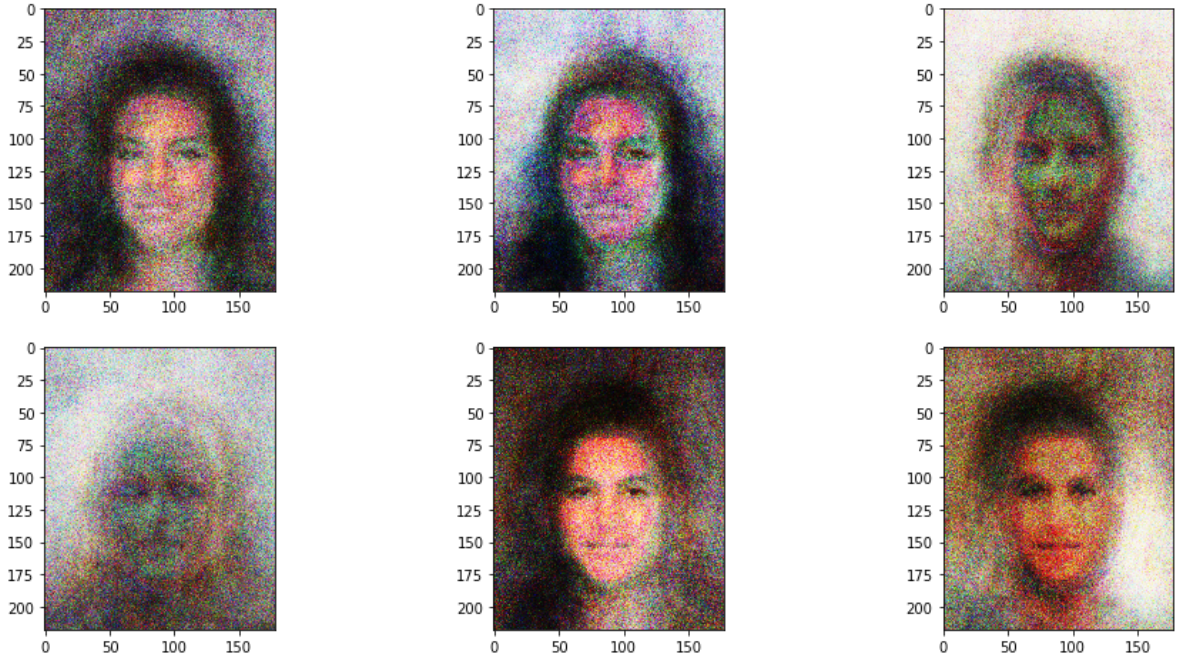
In [33]:

```
G.plot_progress()
```



In [34]:

```
# 훈련된 생성기로부터 몇개의 출력을 플롯

# 3열, 2열의 격자로 이미지 출력
f, axarr = plt.subplots(2,3, figsize=(16,8))
for i in range(2):
    for j in range(3):
        output = G.forward(generate_random_seed(100))
        img = output.detach().cpu().numpy()
        axarr[i,j].imshow(img, interpolation='none', cmap='Blues')
        pass
    pass
```



In [35]:

```
# 현재 텐서에 할당된 메모리 (기가바이트 단위)

torch.cuda.memory_allocated(device) / (1024*1024*1024)
```

Out[35]:

```
0.6999893188476562
```

In [36]:

```
# 프로그램 실행시 최대 메모리 소비량 (기가바이트 단위)

torch.cuda.max_memory_allocated(device) / (1024*1024*1024)
```

Out[36]:

```
1.0939888954162598
```

In [37]:

```
# 메모리 소비 요약

print(torch.cuda.memory_summary(device, abbreviated=True))
```

```
|===========================================================================|
|                  PyTorch CUDA memory summary, device ID 0                 |
|---------------------------------------------------------------------------|
|            CUDA OOMs: 0            |        cudaMalloc retries: 0         |
|===========================================================================|
|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Freed  |
|---------------------------------------------------------------------------|
| Allocated memory      |  733992 KB |    1120 MB |   19249 GB |   19248 GB |
|---------------------------------------------------------------------------|
| Active memory         |  733992 KB |    1120 MB |   19249 GB |   19248 GB |
|---------------------------------------------------------------------------|
| GPU reserved memory   |    1220 MB |    1220 MB |    1220 MB |       0 B  |
|---------------------------------------------------------------------------|
| Non-releasable memory |    9432 KB |   13412 KB |  538642 MB |  538633 MB |
|---------------------------------------------------------------------------|
| Allocations           |      56    |      87    |    4480 K  |    4480 K  |
|---------------------------------------------------------------------------|
| Active allocs         |      56    |      87    |    4480 K  |    4480 K  |
|---------------------------------------------------------------------------|
| GPU reserved segments |      16    |      16    |      16    |       0    |
|---------------------------------------------------------------------------|
| Non-releasable allocs |      16    |      16    |    1981 K  |    1981 K  |
|---------------------------------------------------------------------------|
| Oversize allocations  |       0    |       0    |       0    |       0    |
|---------------------------------------------------------------------------|
| Oversize GPU segments |       0    |       0    |       0    |       0    |
|===========================================================================|
```

201600779 김영민

In [ ]:

```
```