GAN - Bonus Points

For this part of the assignment you implement two different types of generative adversarial networks. We will train the networks on the Celeb A dataset which is a large set of celebrity face images.

```
import torch
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision.datasets import ImageFolder
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

%load_ext autoreload
%autoreload 2
```

```
!wget https://uofi.box.com/shared/static/q4pf89jtkvjndi4f8ip7wofuulhhphjj.zip
!mkdir celeba_data
!unzip q4pf89jtkvjndi4f8ip7wofuulhhphjj.zip -d celeba_data
!rm q4pf89jtkvjndi4f8ip7wofuulhhphjj.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: celeba_data/celeba_train_128res/175430_crop.jpg
  inflating: celeba_data/celeba_train_128res/157564_crop.jpg
  inflating: celeba_data/celeba_train_128res/039881_crop.jpg
  inflating: celeba_data/celeba_train_128res/150303_crop.jpg
  inflating: celeba_data/celeba_train_128res/145071_crop.jpg
  inflating: celeba_data/celeba_train_128res/165362_crop.jpg
  inflating: celeba_data/celeba_train_128res/197644_crop.jpg
  inflating: celeba_data/celeba_train_128res/015423_crop.jpg
  inflating: celeba_data/celeba_train_128res/188193_crop.jpg
  inflating: celeba_data/celeba_train_128res/124213_crop.jpg
  inflating: celeba_data/celeba_train_128res/160818_crop.jpg
  inflating: celeba_data/celeba_train_128res/169663_crop.jpg
  inflating: celeba_data/celeba_train_128res/066406_crop.jpg
  inflating: celeba_data/celeba_train_128res/135163_crop.jpg
  inflating: celeba_data/celeba_train_128res/116258_crop.jpg
  inflating: celeba_data/celeba_train_128res/113618_crop.jpg
  inflating: celeba_data/celeba_train_128res/000109_crop.jpg
  inflating: celeba_data/celeba_train_128res/110779_crop.jpg
  inflating: celeba_data/celeba_train_128res/006145_crop.jpg
  inflating: celeba_data/celeba_train_128res/181839_crop.jpg
  inflating: celeba_data/celeba_train_128res/028015_crop.jpg
  inflating: celeba_data/celeba_train_128res/133077_crop.jpg
  inflating: celeba_data/celeba_train_128res/107053_crop.jpg
  inflating: celeba_data/celeba_train_128res/076031_crop.jpg
  inflating: celeba_data/celeba_train_128res/110833_crop.jpg
  inflating: celeba_data/celeba_train_128res/166880_crop.jpg
  inflating: celeba_data/celeba_train_128res/143283_crop.jpg
```

10h 58m 55s completed at 10:45 AM

```
inflating: celeba_data/celeba_train_128res/014888_crop.jpg
inflating: celeba_data/celeba_train_128res/053605_crop.jpg
inflating: celeba_data/celeba_train_128res/163895_crop.jpg
inflating: celeba_data/celeba_train_128res/117500_crop.jpg
inflating: celeba_data/celeba_train_128res/192585_crop.jpg
inflating: celeba_data/celeba_train_128res/013972_crop.jpg
inflating: celeba_data/celeba_train_128res/191699_crop.jpg
inflating: celeba_data/celeba_train_128res/028127_crop.jpg
inflating: celeba_data/celeba_train_128res/080856_crop.jpg
inflating: celeba_data/celeba_train_128res/177193_crop.jpg
inflating: celeba_data/celeba_train_128res/066551_crop.jpg
inflating: celeba_data/celeba_train_128res/063110_crop.jpg
inflating: celeba_data/celeba_train_128res/070300_crop.jpg
inflating: celeba_data/celeba_train_128res/172739_crop.jpg
inflating: celeba_data/celeba_train_128res/198872_crop.jpg
inflating: celeba_data/celeba_train_128res/158186_crop.jpg
inflating: celeba_data/celeba_train_128res/147772_crop.jpg
inflating: celeba_data/celeba_train_128res/137374_crop.jpg
inflating: celeba_data/celeba_train_128res/167688_crop.jpg
inflating: celeba_data/celeba_train_128res/052678_crop.jpg
inflating: celeba_data/celeba_train_128res/126112_crop.jpg
inflating: celeba_data/celeba_train_128res/021357_crop.jpg
inflating: celeba_data/celeba_train_128res/009576_crop.jpg
inflating: celeba_data/celeba_train_128res/126043_crop.jpg
inflating: celeba_data/celeba_train_128res/031951_crop.jpg
inflating: celeba_data/celeba_train_128res/176003_crop.jpg
inflating: celeba_data/celeba_train_128res/093079_crop.jpg
```

```
from gan.train import train
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "gpu")
```

GAN loss functions

In this assignment you will implement two different types of GAN cost functions. You will first implement the loss from the original GAN paper. You will also implement the loss from LS-GAN.

GAN loss

TODO: Implement the `discriminator_loss` and `generator_loss` functions in `gan/losses.py`.

The generator loss is given by:

$$\ell_G = -\mathbb{E}_{z\sim p(z)}\left[\log D(G(z))\right]$$

and the discriminator loss is:

$$\ell_D = -\mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] - \mathbb{E}_{z\sim p(z)}\left[\log(1-D(G(z)))\right]$$

Note that these are negated from the equations presented earlier as we will be *minimizing* these losses.

HINTS: You should use the `torch.nn.functional.binary_cross_entropy_with_logits` function to output from the discriminator. Given a score $s \in \mathbb{R}$ and a label $y \in \{0, 1\}$, the binary cross entropy loss is

$$bce(s, y) = -y * \log(s) - (1 - y) * \log(1 - s)$$

Instead of computing the expectation of $\log D(G(z))$, $\log D(x)$ and $\log(1 - D(G(z)))$, we will be averaging over elements of the minibatch, so make sure to combine the loss by averaging instead of summing.

```
1 from gan.losses import discriminator_loss, generator_loss
```

Least Squares GAN loss

TODO: Implement the `ls_discriminator_loss` and `ls_generator_loss` functions in `gan/losses.py`.

We'll now look at Least Squares GAN, a newer, more stable alernative to the original GAN loss function. For this part, all we have to do is change the loss function and retrain the model. We'll implement equation (9) in the paper, with the generator loss:

$$\ell_G = \frac{1}{2}\mathbb{E}_{z \sim p(z)}\left[\left(D(G(z)) - 1\right)^2\right]$$

and the discriminator loss:

$$\ell_D = \frac{1}{2}\mathbb{E}_{x \sim p_{\text{data}}}\left[\left(D(x) - 1\right)^2\right] + \frac{1}{2}\mathbb{E}_{z \sim p(z)}\left[\left(D(G(z))\right)^2\right]$$

HINTS: Instead of computing the expectation, we will be averaging over elements of the minibatch, so make sure to combine the loss by averaging instead of summing. When plugging in for $D(x)$ and $D(G(z))$ use the direct output from the discriminator (`scores_real` and `scores_fake`).

```
1 from gan.losses import ls_discriminator_loss, ls_generator_loss
```

GAN model architecture

TODO: Implement the `Discriminator` and `Generator` networks in `gan/models.py`.

We recommend the following architectures which are inspired by DCGAN:

Discriminator:

- convolutional layer with in_channels=3, out_channels=128, kernel=4, stride=2
- convolutional layer with in_channels=128, out_channels=256, kernel=4, stride=2
- batch norm
- convolutional layer with in_channels=256, out_channels=512, kernel=4, stride=2
- batch norm
- convolutional layer with in_channels=512, out_channels=1024, kernel=4, stride=2

Instead of Relu we LeakyReLu throughout the discriminator (we use a negative slope value of 0.2). You can use simply use relu as well.

The output of your discriminator should be a single value score corresponding to each input sample. See `torch.nn.LeakyReLU`.

Generator:

Note: In the generator, you will need to use transposed convolution (sometimes known as fractionally-strided convolution or deconvolution). This function is implemented in pytorch as `torch.nn.ConvTranspose2d`.

- transpose convolution with in_channels=NOISE_DIM, out_channels=1024, kernel=4, stride=1
- batch norm
- transpose convolution with in_channels=1024, out_channels=512, kernel=4, stride=2
- batch norm
- transpose convolution with in_channels=512, out_channels=256, kernel=4, stride=2
- batch norm
- transpose convolution with in_channels=256, out_channels=128, kernel=4, stride=2
- batch norm
- transpose convolution with in_channels=128, out_channels=3, kernel=4, stride=2

The output of the final layer of the generator network should have a `tanh` nonlinearity to output values between -1 and 1. The output should be a 3x64x64 tensor for each sample (equal dimensions to the images from the dataset).

```
from gan.models_bonus_128 import Discriminator, Generator
```

Data loading: Celeb A Dataset

The CelebA images we provide have been filtered to obtain only images with clear faces and have been cropped and downsampled to 128x128 resolution.

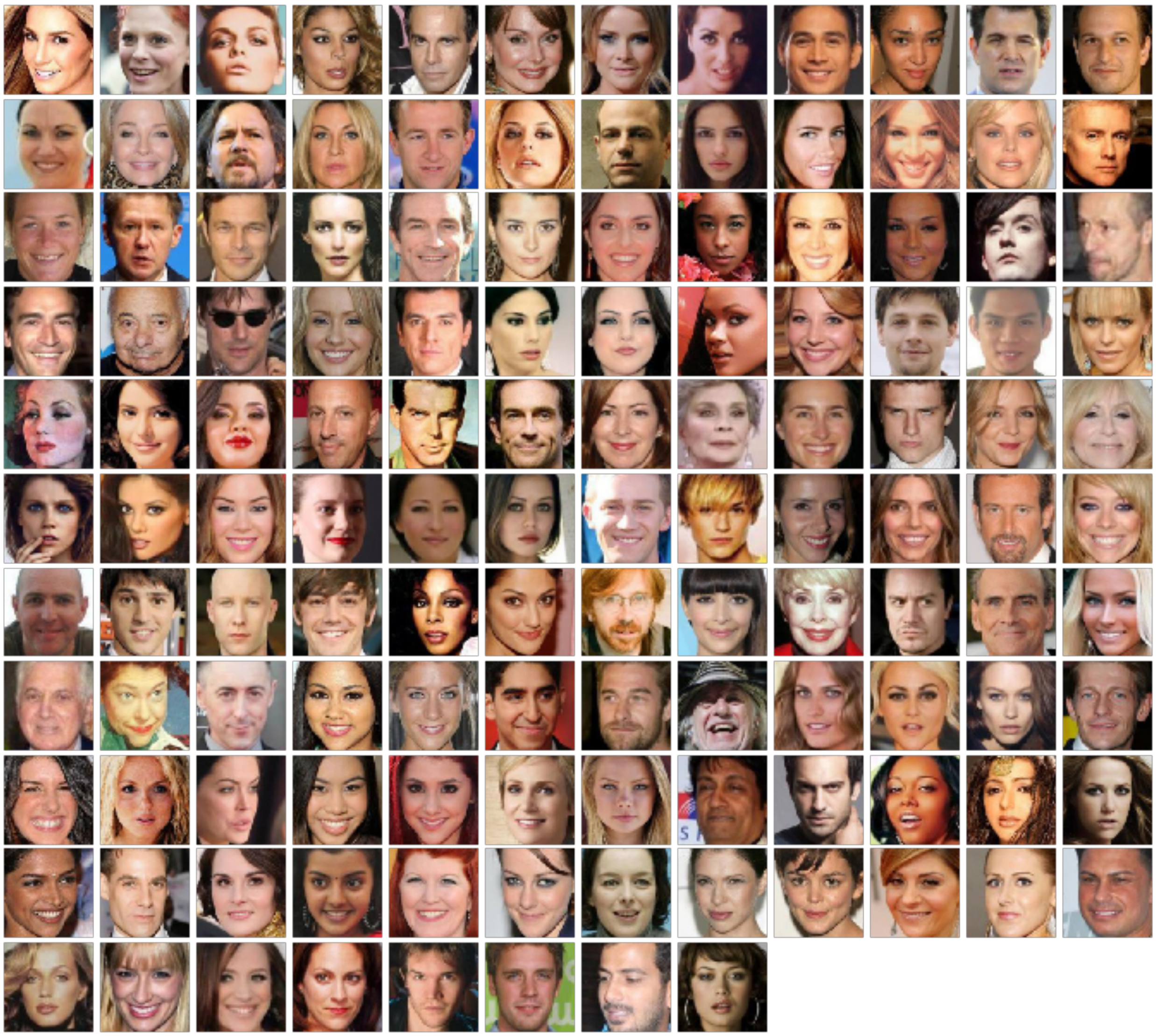
```
batch_size = 128
scale_size = 128  # use the original resize the images 128x128 for training

celeba_root = 'celeba_data'
```

```
celeba_train = ImageFolder(root=celeba_root, transform=transforms.Compose([
  transforms.Resize(scale_size),
  transforms.ToTensor(),
```

```
from gan.utils import show_images

imgs = celeba_loader_train.__iter__().next()[0].numpy().squeeze()
show_images(imgs, color=True)
```



Training

TODO: Fill in the training loop in `gan/train.py`.

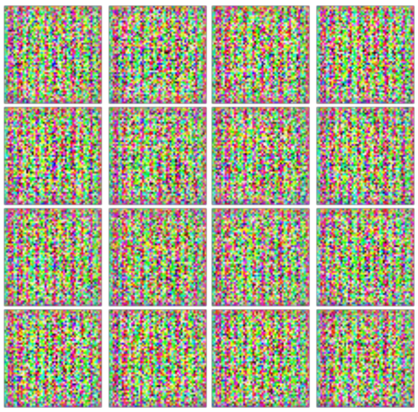
```
NOISE_DIM = 100
```

```
D_optimizer = torch.optim.Adam(D.parameters(), lr=learning_rate, betas = (0.5, 0.999))
G_optimizer = torch.optim.Adam(G.parameters(), lr=learning_rate, betas = (0.5, 0.999))
```

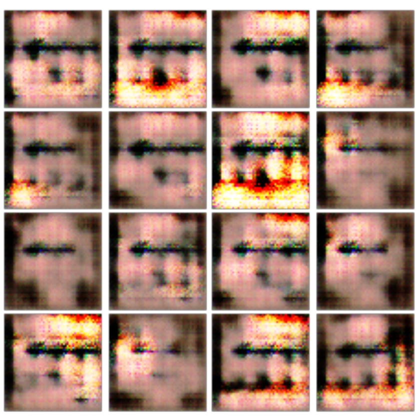
```
# original gan
train(D, G, D_optimizer, G_optimizer, discriminator_loss,
          generator_loss, num_epochs=NUM_EPOCHS, show_every=300,
          train_loader=celeba_loader_train, device=device)

```

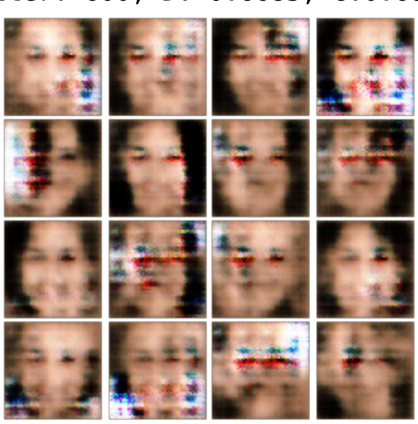
```
EPOCH:  1
Iter: 0, D: 0.694, G:0.8789
```

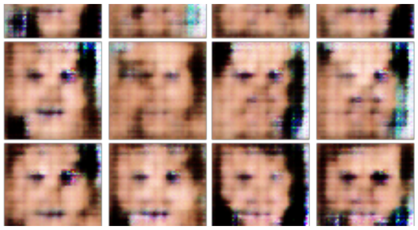


```
Iter: 300, D: 0.692, G:0.7342
```



```
Iter: 600, D: 0.6683, G:0.6823
```





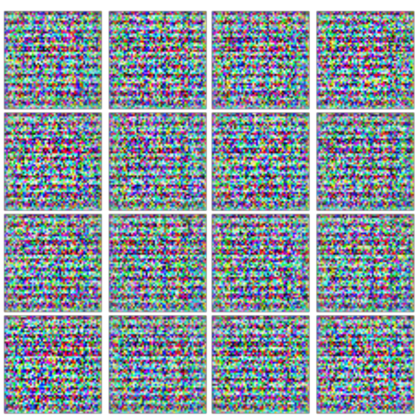
Train LS-GAN

```
D = Discriminator().to(device)
G = Generator(noise_dim=NOISE_DIM).to(device)
```

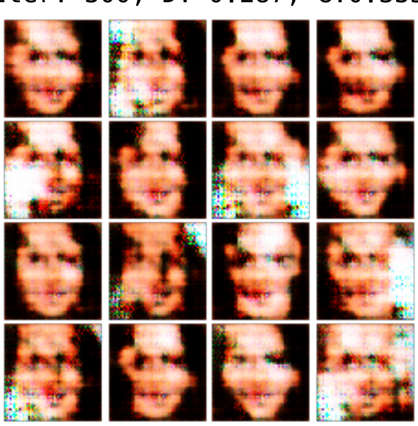
```
D_optimizer = torch.optim.Adam(D.parameters(), lr=learning_rate, betas = (0.5, 0.999))
G_optimizer = torch.optim.Adam(G.parameters(), lr=learning_rate, betas = (0.5, 0.999))
```

```
# ls-gan
train(D, G, D_optimizer, G_optimizer, ls_discriminator_loss,
          ls_generator_loss, num_epochs=NUM_EPOCHS, show_every=300,
          train_loader=celeba_loader_train, device=device)
```

```
EPOCH:  1
Iter: 0, D: 0.3609, G:0.6999
```



```
Iter: 300, D: 0.287, G:0.3357
```



```
Iter: 900, D: 0.2654, G:0.3943
```

1