MP4_P1

November 20, 2020

1 Generative Adversarial Networks

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 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
```

[ ]:
```
from gan.train import train
```

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

2 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.

[ ]:
```
from gan.losses import discriminator_loss, generator_loss
```

2.0.1 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:

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

and the discriminator loss:

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

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`).

[ ]:
```
from gan.losses import ls_discriminator_loss, ls_generator_loss
```

3 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
- batch norm
- convolutional layer with in_channels=1024, out_channels=1, kernel=4, stride=1

Use padding = 1 (not 0) for all the convolutional layers.

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 import Discriminator, Generator
```

4 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 = 64  # We resize the images to 64x64 for training

celeba_root = 'celeba_data'
```

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

celeba_loader_train = DataLoader(celeba_train, batch_size=batch_size,␣
 ↪drop_last=True)
```

4.0.1 Visualize dataset

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

5 Training

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

[ ]:
```
NOISE_DIM = 100
NUM_EPOCHS = 50
learning_rate = 0.0002
```

5.0.1 Train GAN

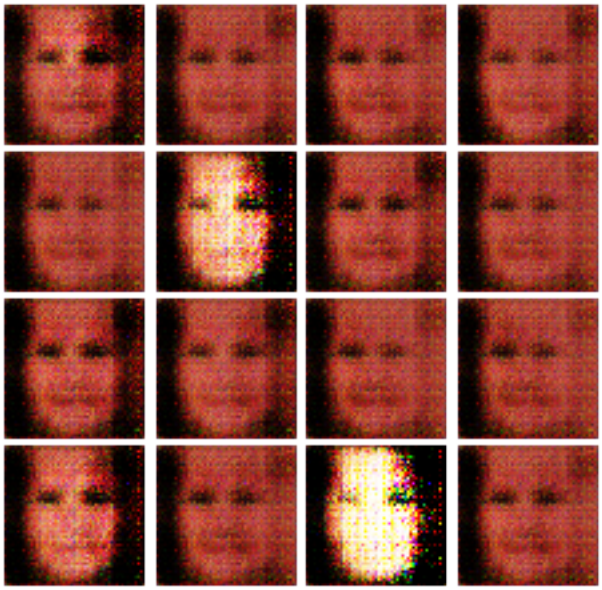
[ ]:
```
D = Discriminator().to(device)
G = Generator(noise_dim=NOISE_DIM).to(device)
```

[ ]:
```
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))
```

[ ]:
```
# original gan - spectral normalization
train(D, G, D_optimizer, G_optimizer, discriminator_loss,
          generator_loss, num_epochs=NUM_EPOCHS, show_every=150,
          train_loader=celeba_loader_train, device=device)
```

```
EPOCH:  1
Iter: 0, D: 1.464, G:5.203
```

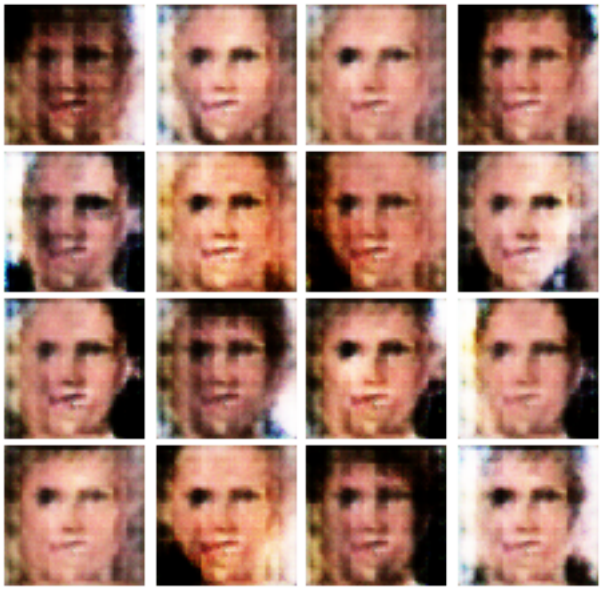
```
Iter: 150, D: 0.4822, G:4.225
```



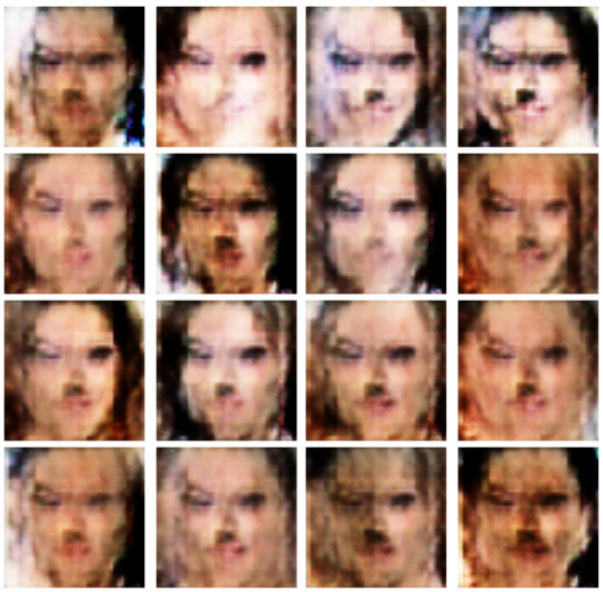
Iter: 300, D: 0.8303, G:5.985

Iter: 450, D: 0.6145, G:2.894

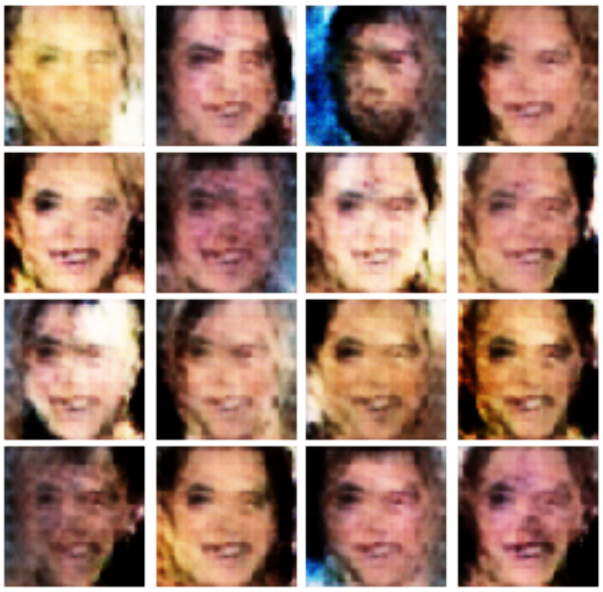
Iter: 600, D: 1.239, G:2.7



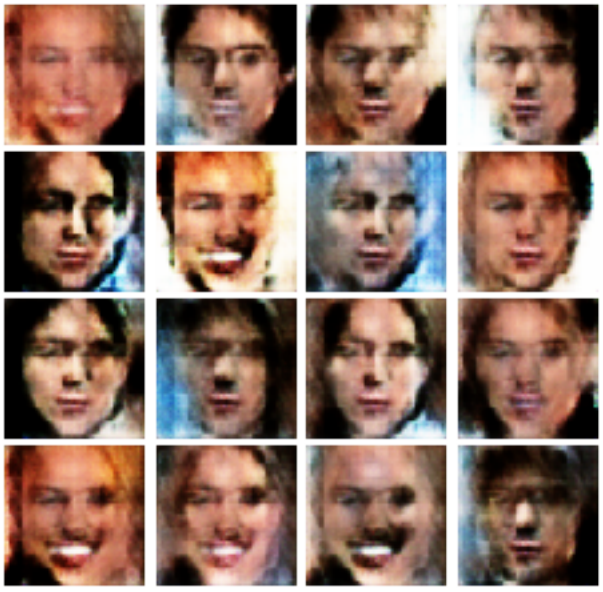
Iter: 750, D: 0.6017, G:3.364



```
Iter: 900, D: 0.7745, G:3.329
```

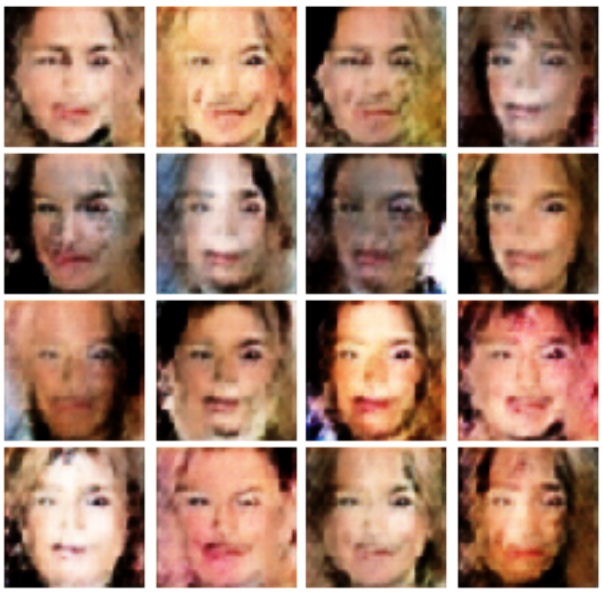


```
EPOCH:  2
Iter: 1050, D: 0.787, G:3.696
```

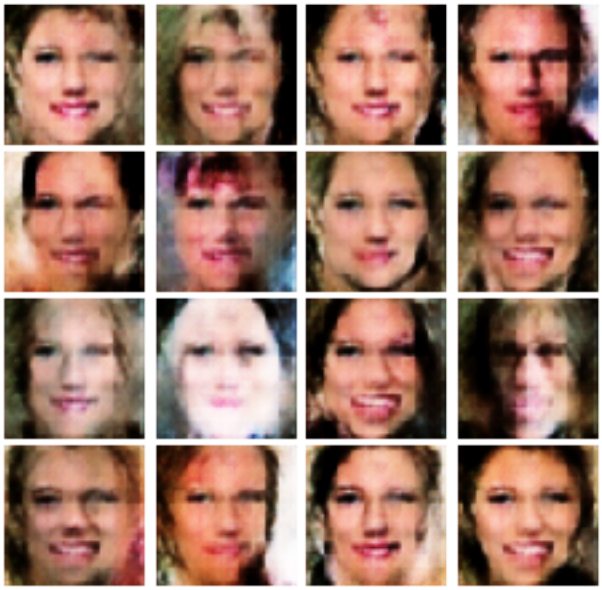


Iter: 1200, D: 0.6609, G:4.173

Iter: 1350, D: 0.4499, G:2.334



Iter: 1500, D: 1.141, G:1.866



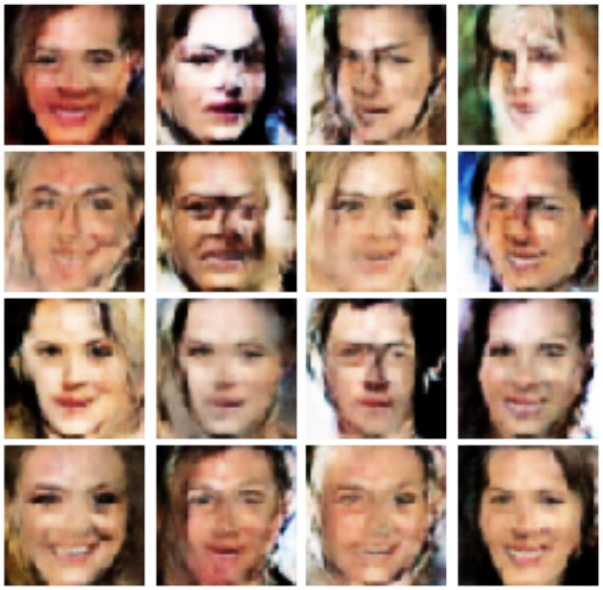
Iter: 1650, D: 0.8206, G:4.874

```
Iter: 1800, D: 0.4146, G:3.103
```

```
Iter: 1950, D: 0.2798, G:4.045
```

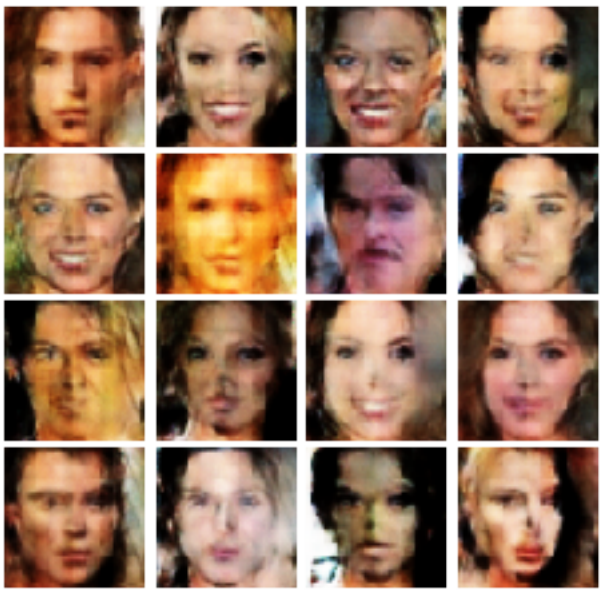
```
EPOCH:  3
Iter: 2100, D: 0.4125, G:2.982
```

```
Iter: 2250, D: 0.4839, G:2.603
```



```
Iter: 2400, D: 0.4345, G:2.079
```

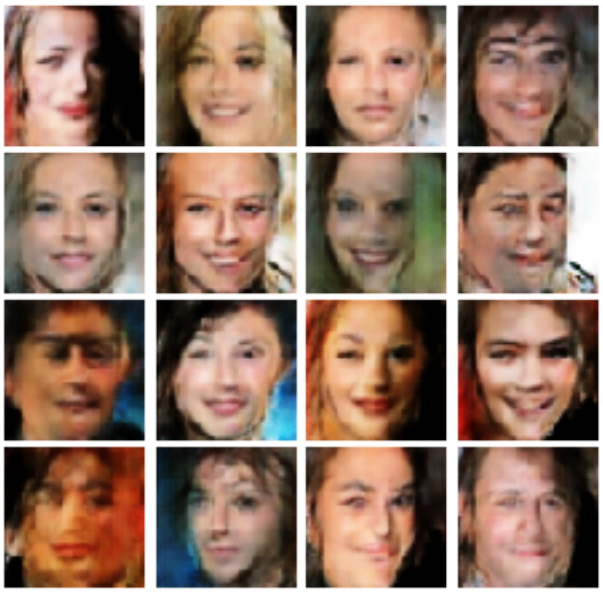
```
Iter: 2550, D: 0.2387, G:4.919
```



Iter: 2700, D: 0.2685, G:2.578

Iter: 2850, D: 0.473, G:2.788

EPOCH: 4
Iter: 3000, D: 0.6404, G:4.263



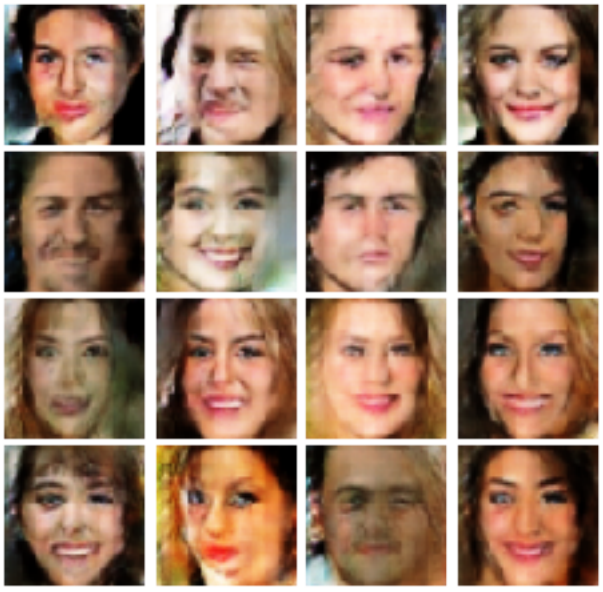
Iter: 3150, D: 0.1776, G:3.901

```
Iter: 3300, D: 0.5424, G:8.048
```

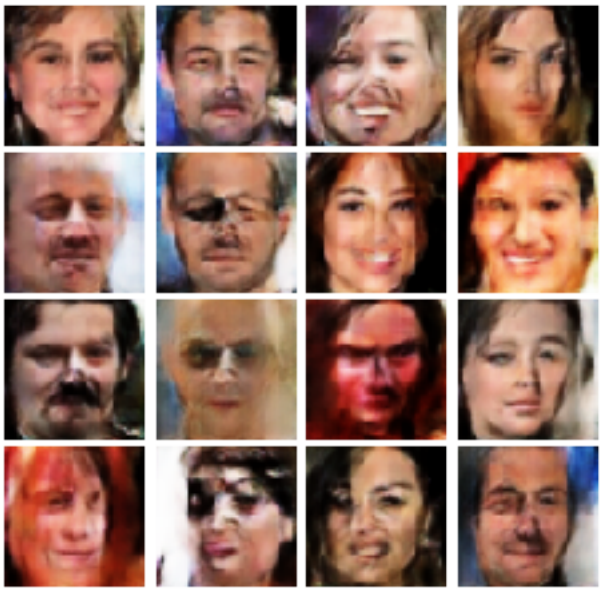
```
Iter: 3450, D: 1.133, G:2.46
```

Iter: 3600, D: 1.615, G:15.39

Iter: 3750, D: 0.6859, G:1.75



```
Iter: 3900, D: 0.5491, G:3.929
```

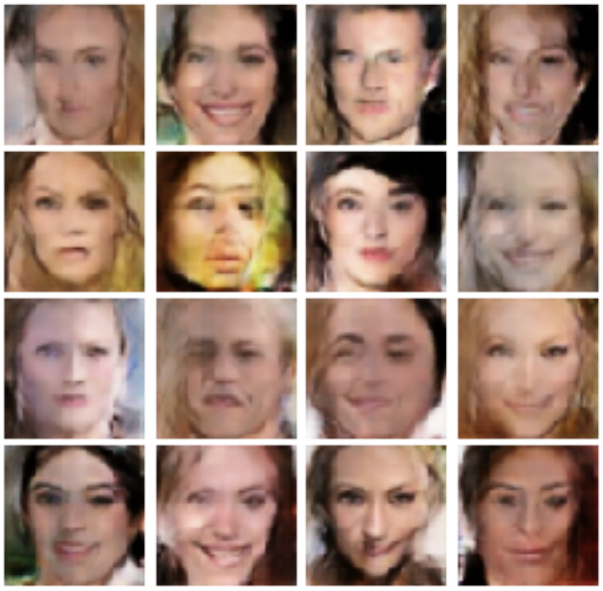


```
EPOCH:  5
Iter: 4050, D: 1.137, G:2.713
```

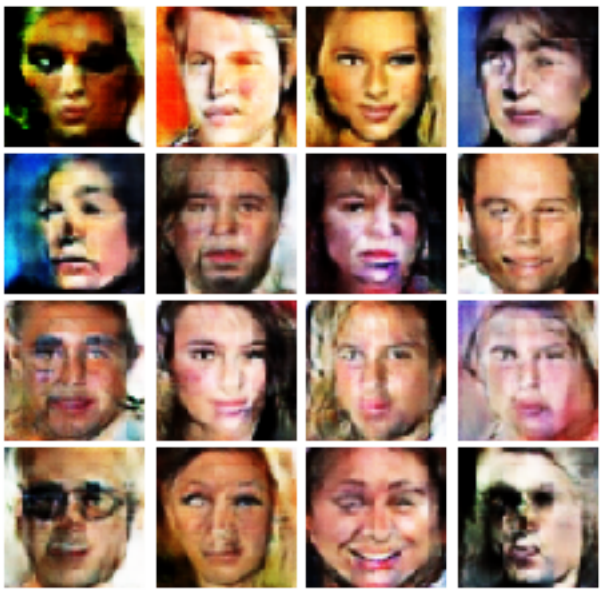
Iter: 4200, D: 0.1574, G:4.54

Iter: 4350, D: 0.7175, G:2.076

```
Iter: 4500, D: 0.2321, G:2.25
```



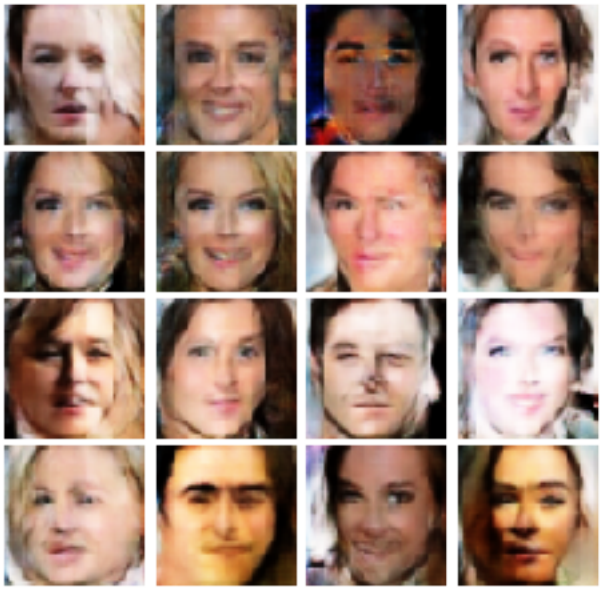
```
Iter: 4650, D: 0.1315, G:4.514
```



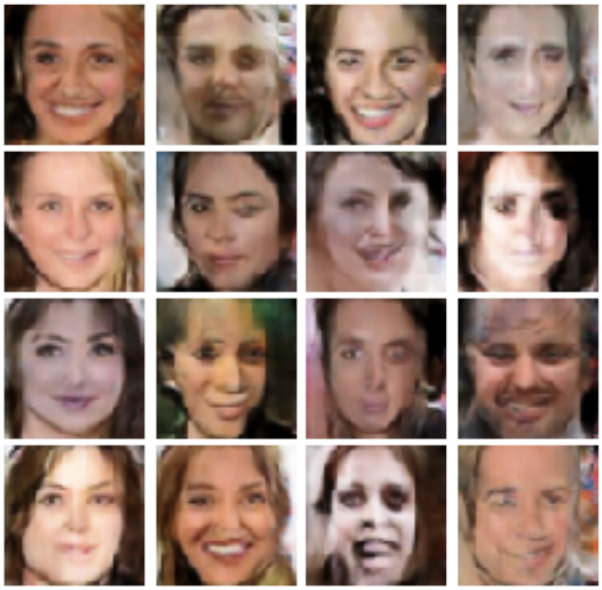
```
Iter: 4800, D: 0.2015, G:4.097
```

```
EPOCH:  6
Iter: 4950, D: 0.5905, G:8.918
```

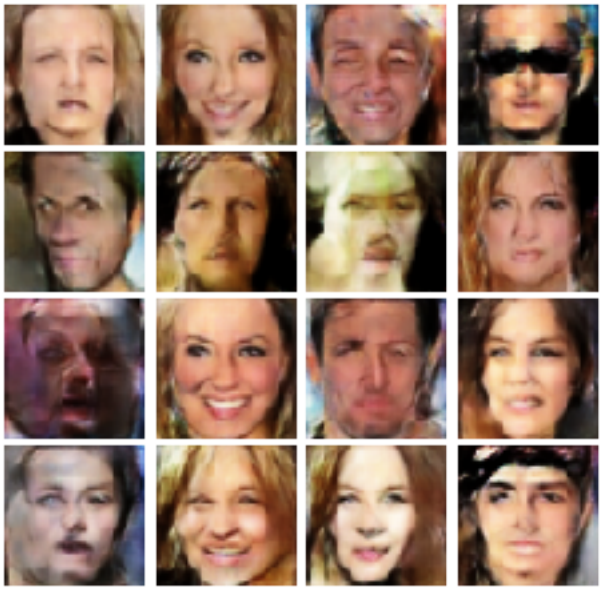
Iter: 5100, D: 1.165, G:7.783



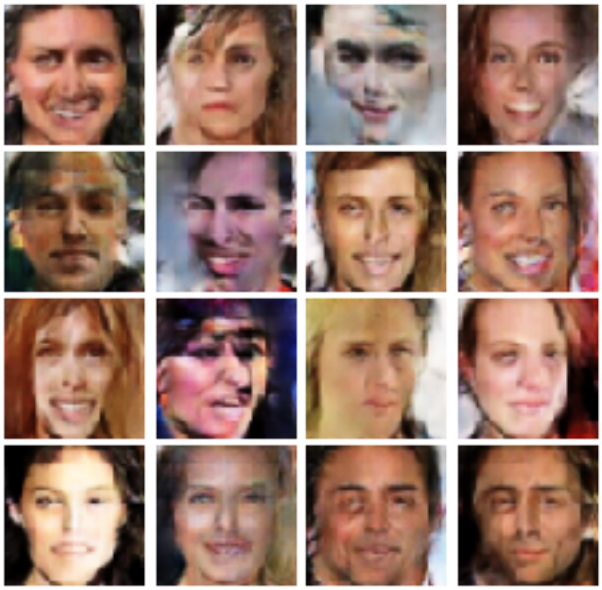
Iter: 5250, D: 2.397, G:11.24



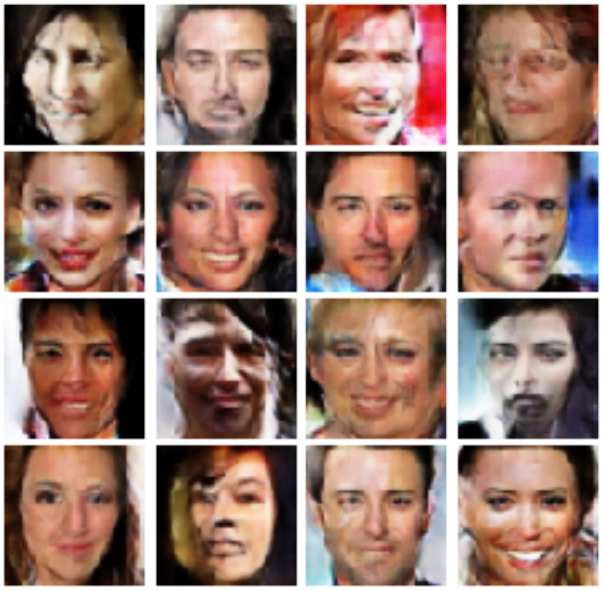
```
Iter: 5400, D: 0.2227, G:3.263
```



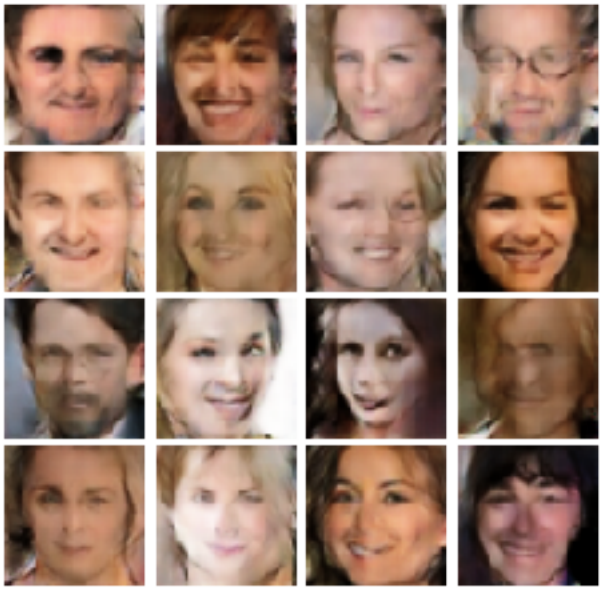
```
Iter: 5550, D: 0.224, G:3.16
```



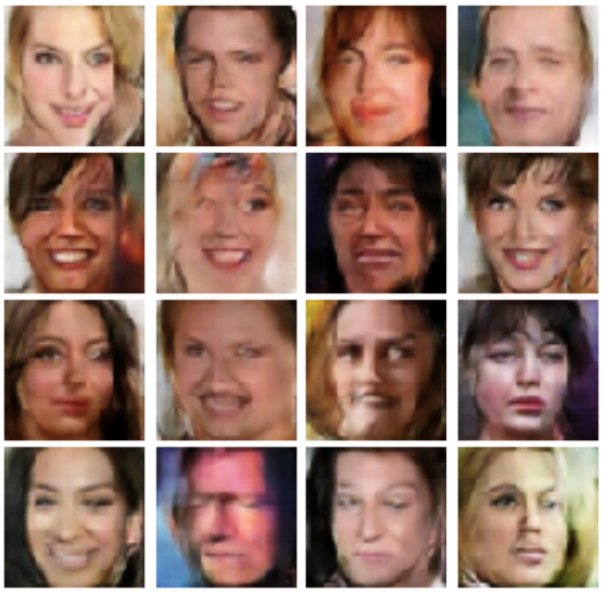
Iter: 5700, D: 0.3269, G:3.169



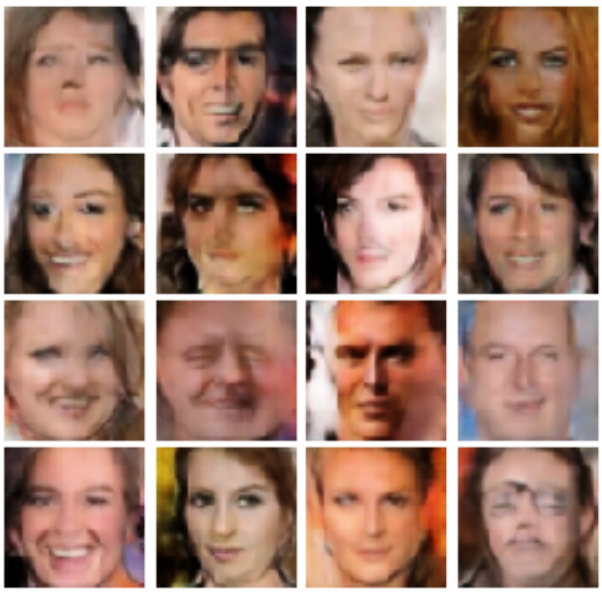
Iter: 5850, D: 0.3931, G:3.92



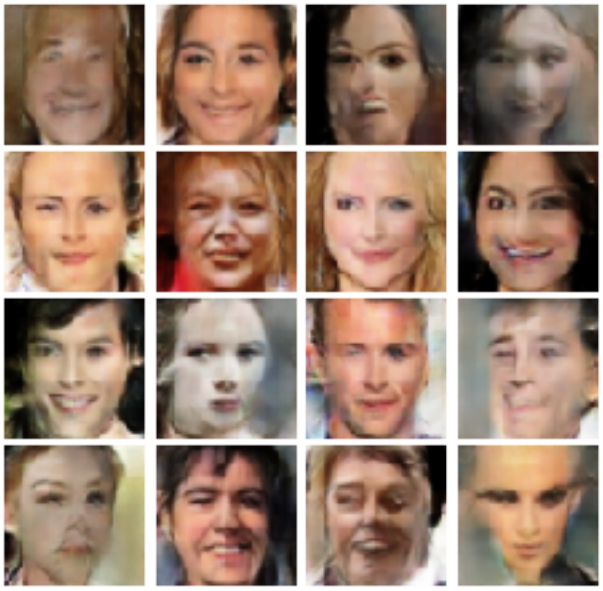
```
EPOCH:  7
Iter: 6000, D: 0.1538, G:3.64
```



```
Iter: 6150, D: 0.7282, G:6.024
```



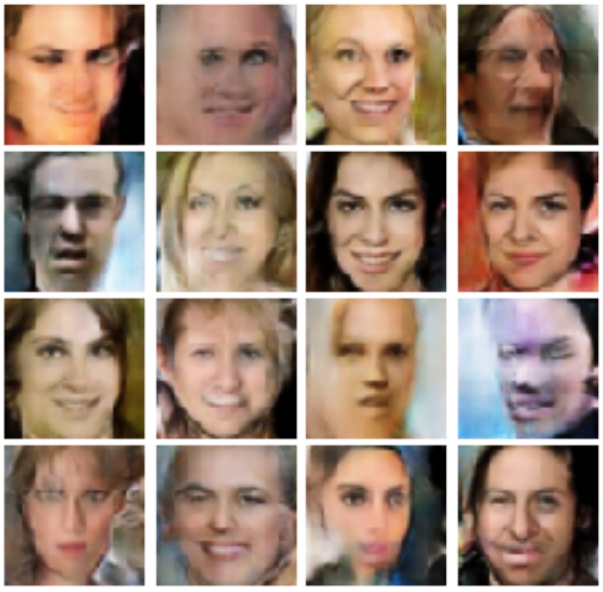
Iter: 6300, D: 0.1785, G:3.802



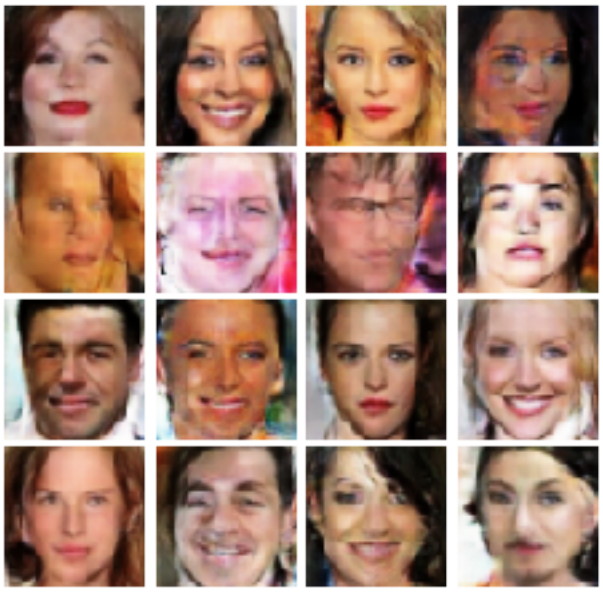
Iter: 6450, D: 0.124, G:2.671

Iter: 6600, D: 0.2779, G:3.952

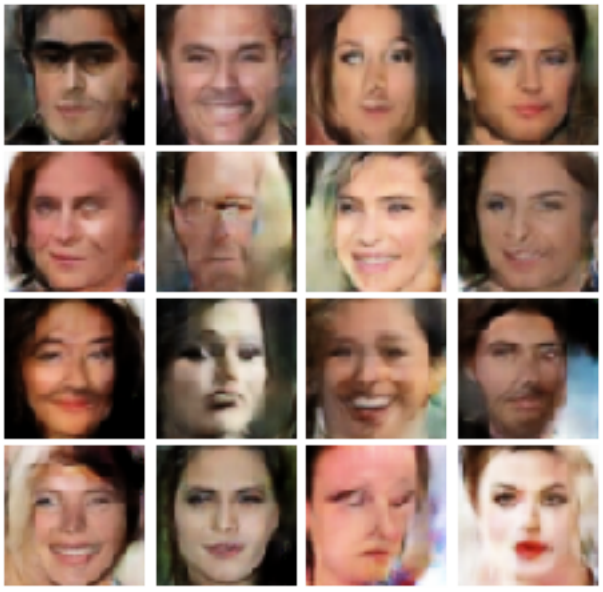
Iter: 6750, D: 0.3234, G:5.367



```
Iter: 6900, D: 0.1961, G:3.06
```

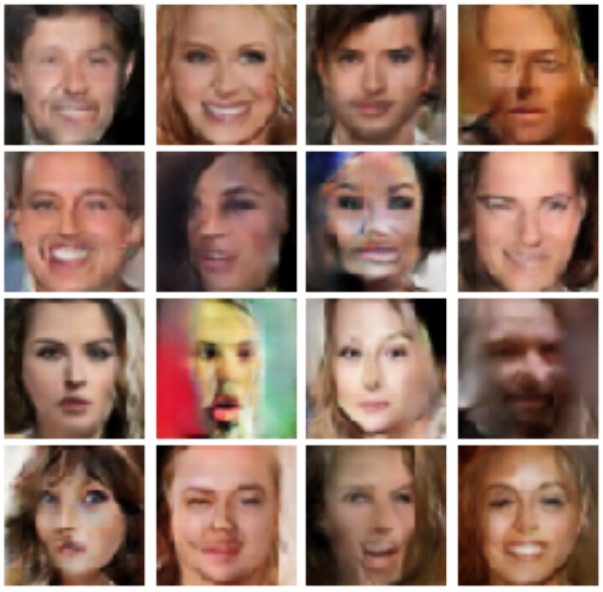


```
EPOCH:  8
Iter: 7050, D: 0.08181, G:7.152
```



```
Iter: 7200, D: 0.3232, G:4.937
```

```
Iter: 7350, D: 0.3526, G:2.444
```

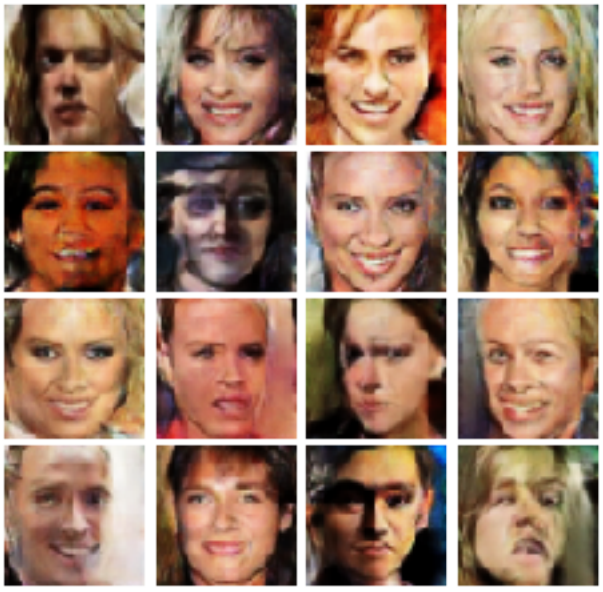


```
Iter: 7500, D: 0.05562, G:4.744
```

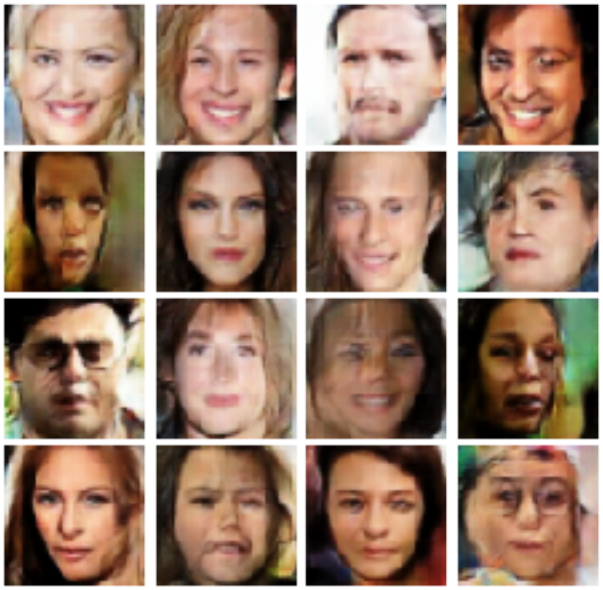
```
Iter: 7650, D: 0.1504, G:1.497
```

```
Iter: 7800, D: 0.04531, G:5.386
```

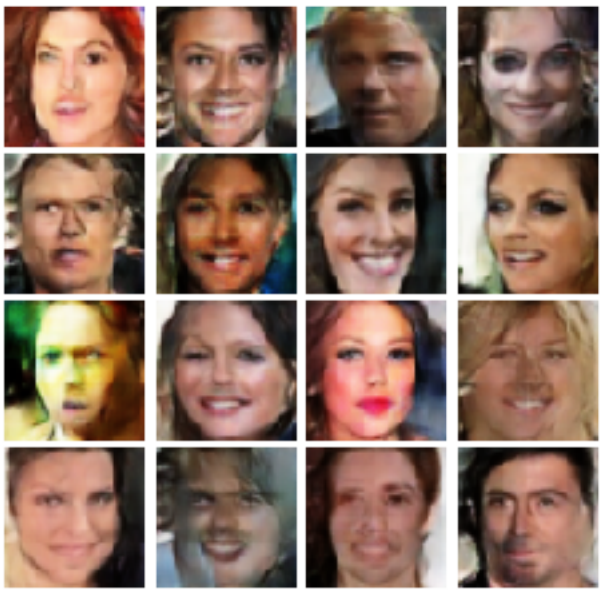
```
EPOCH:  9
Iter: 7950, D: 0.8338, G:13.14
```



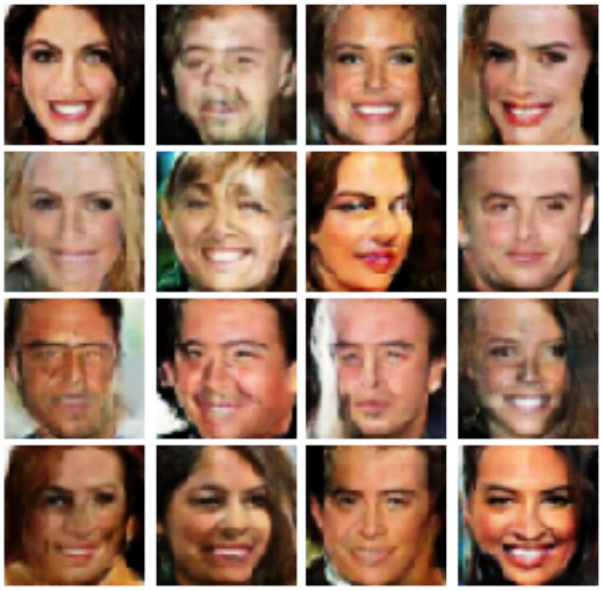
```
Iter: 8100, D: 0.1056, G:4.341
```



```
Iter: 8250, D: 0.08754, G:3.046
```

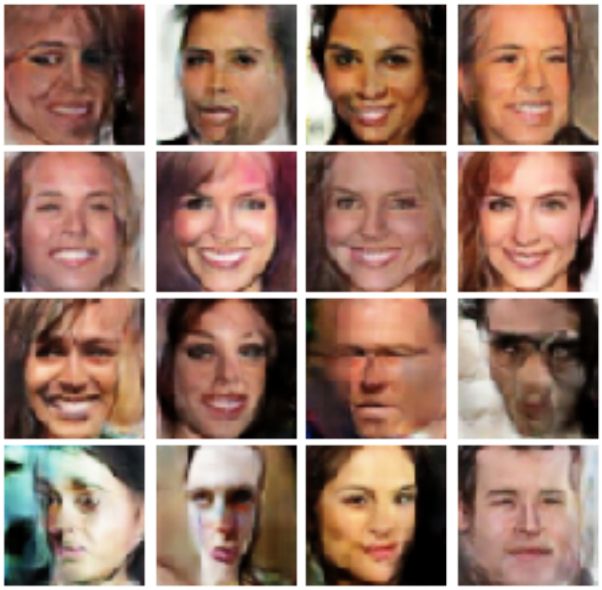


Iter: 8400, D: 0.1849, G:3.37

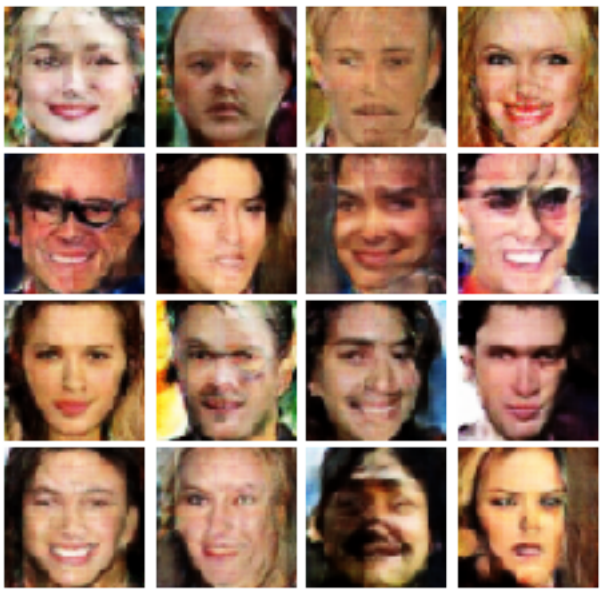


Iter: 8550, D: 0.2251, G:4.905

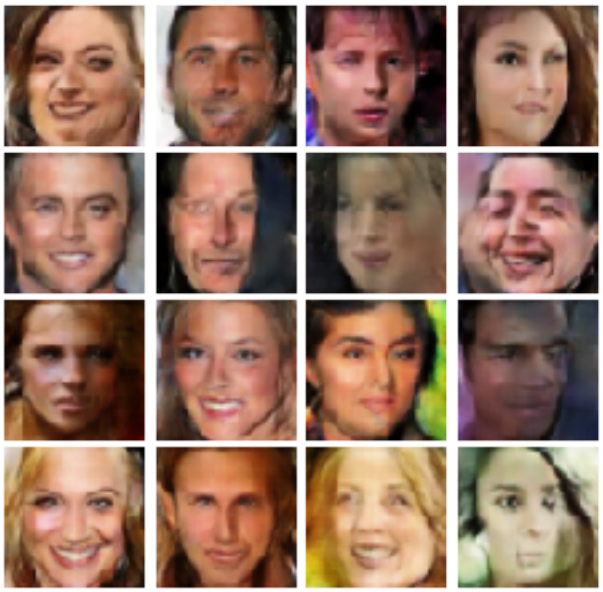
Iter: 8700, D: 0.7274, G:7.648



Iter: 8850, D: 0.106, G:4.847

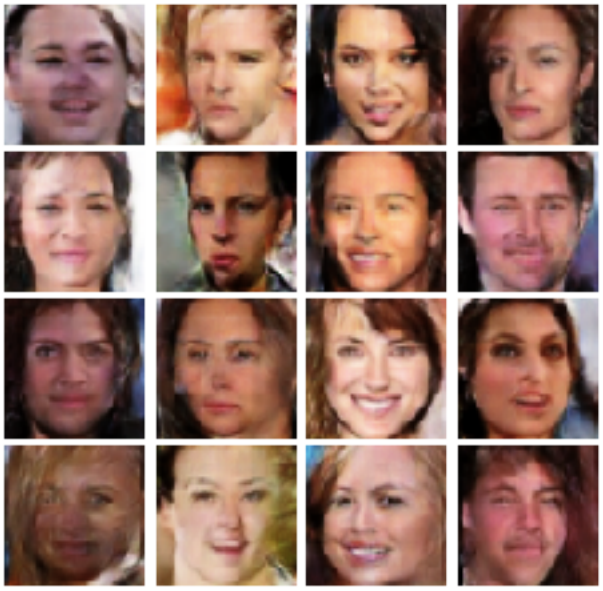


```
EPOCH:  10
Iter: 9000, D: 0.04889, G:2.622
```

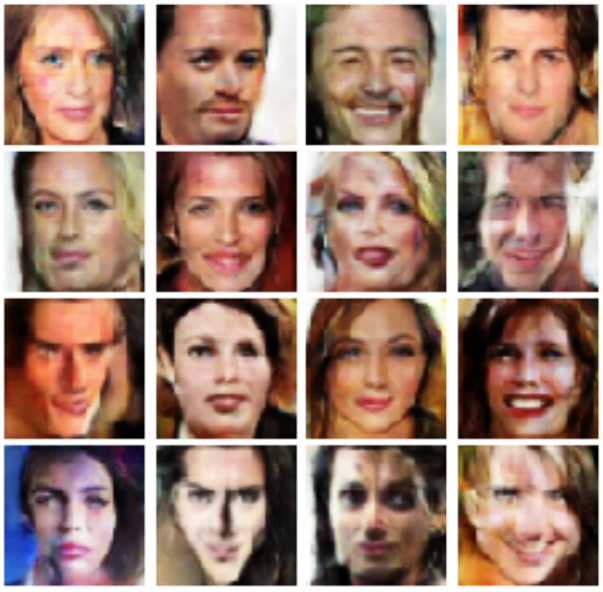


```
Iter: 9150, D: 0.2359, G:0.4249
```

```
Iter: 9300, D: 0.5293, G:4.905
```

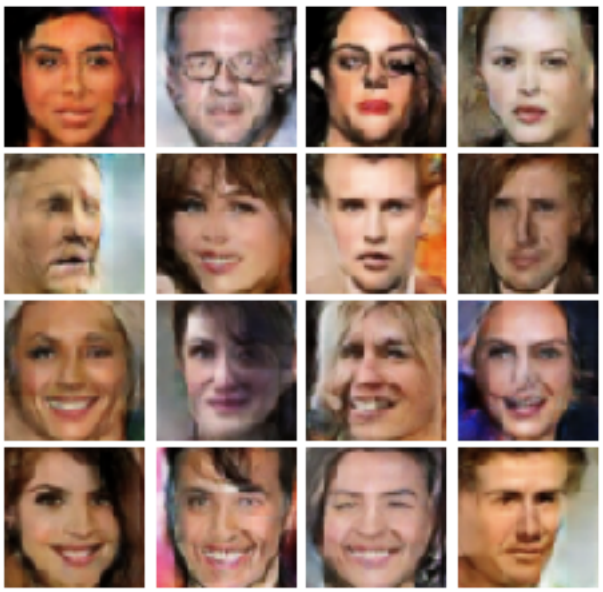


```
Iter: 9450, D: 0.1915, G:3.614
```

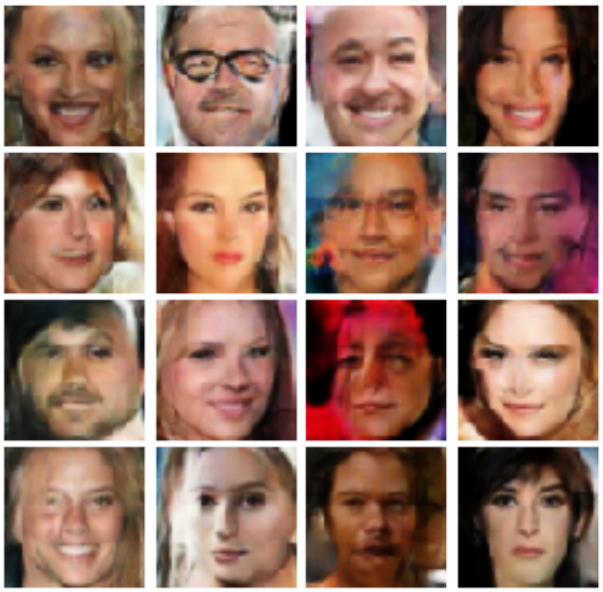


```
Iter: 9600, D: 0.3267, G:5.088
```

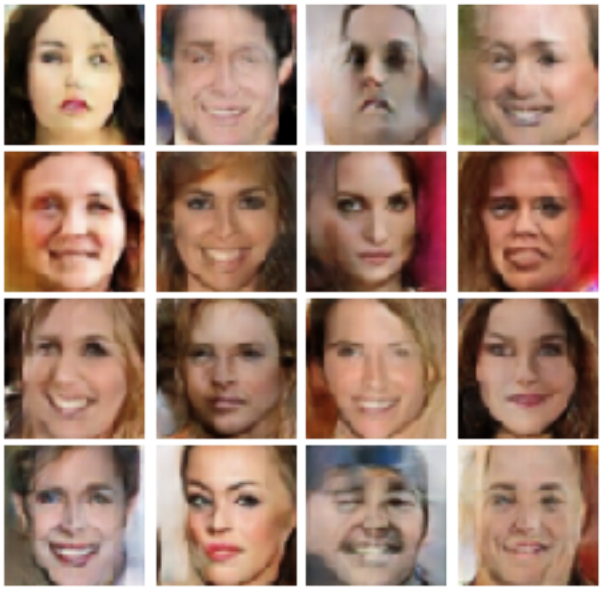
```
Iter: 9750, D: 0.09309, G:4.37
```



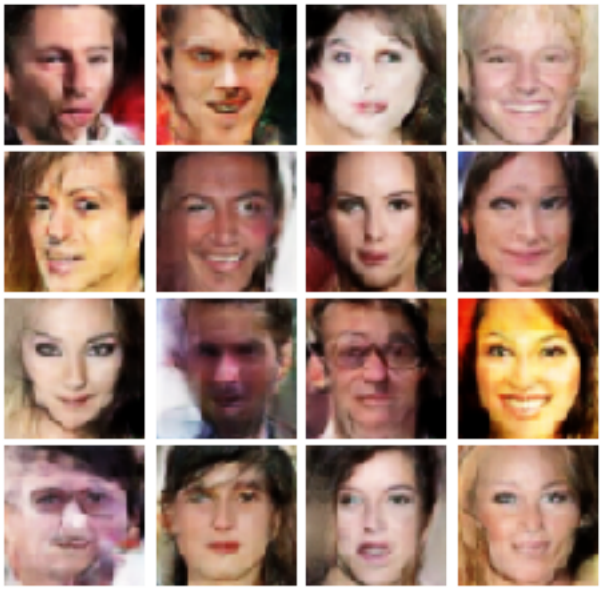
```
EPOCH:  11
Iter: 9900, D: 0.07503, G:4.611
```



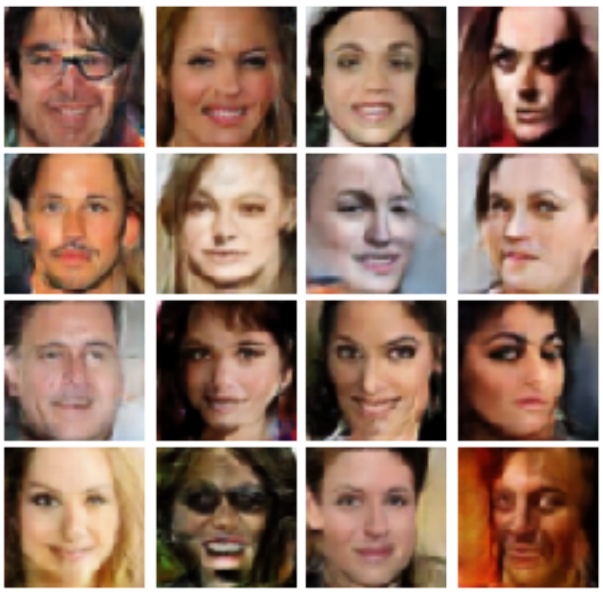
```
Iter: 10050, D: 0.1382, G:3.472
```



```
Iter: 10200, D: 0.8346, G:1.88
```



```
Iter: 10350, D: 0.2293, G:3.036
```



Iter: 10500, D: 0.3636, G:6.611

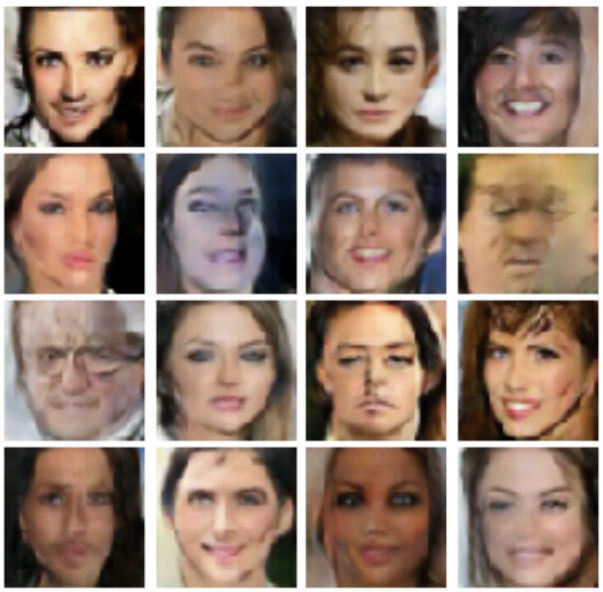
Iter: 10650, D: 0.2277, G:3.173

```
Iter: 10800, D: 0.2869, G:5.153
```

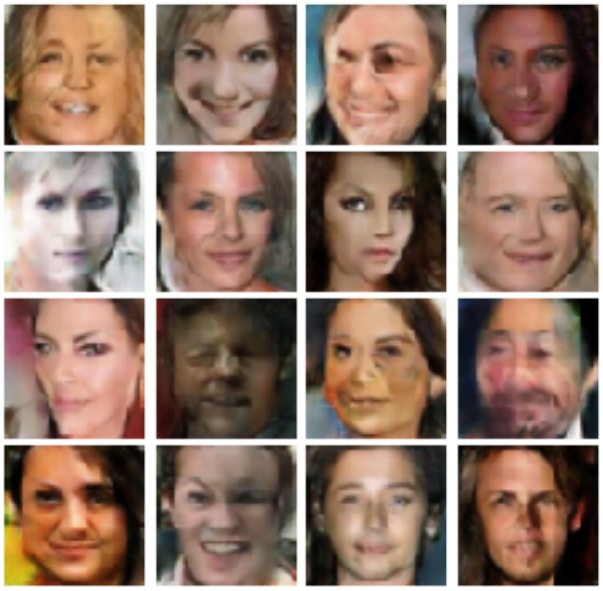
```
EPOCH:  12
Iter: 10950, D: 0.08887, G:2.443
```

Iter: 11100, D: 0.1534, G:5.224

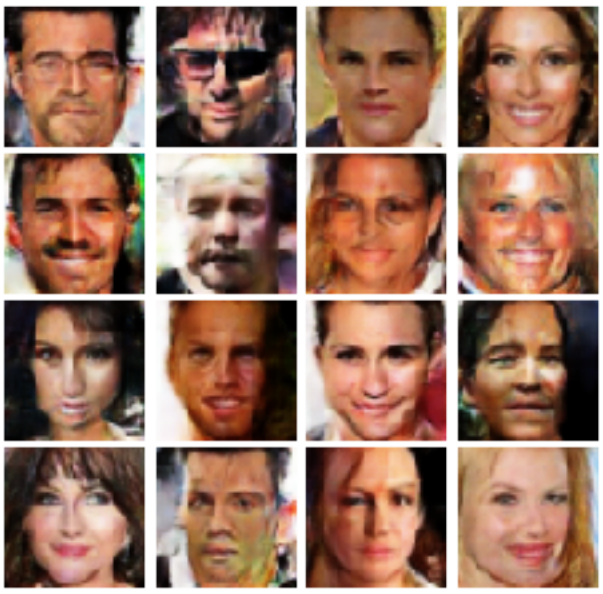
Iter: 11250, D: 0.2783, G:6.173



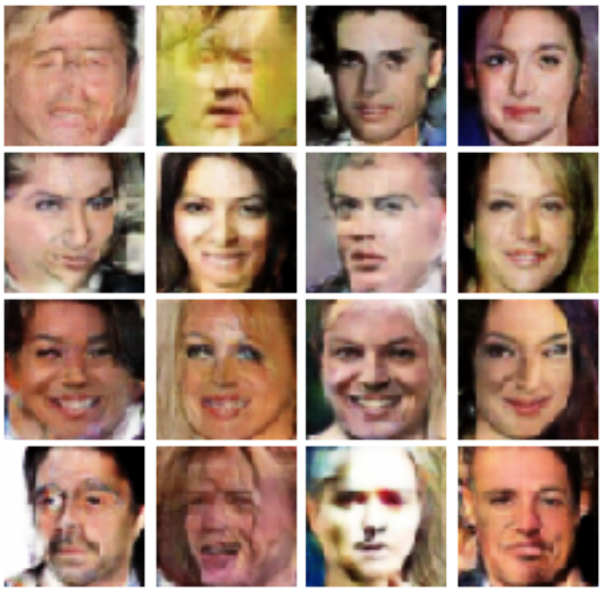
Iter: 11400, D: 0.1129, G:2.101



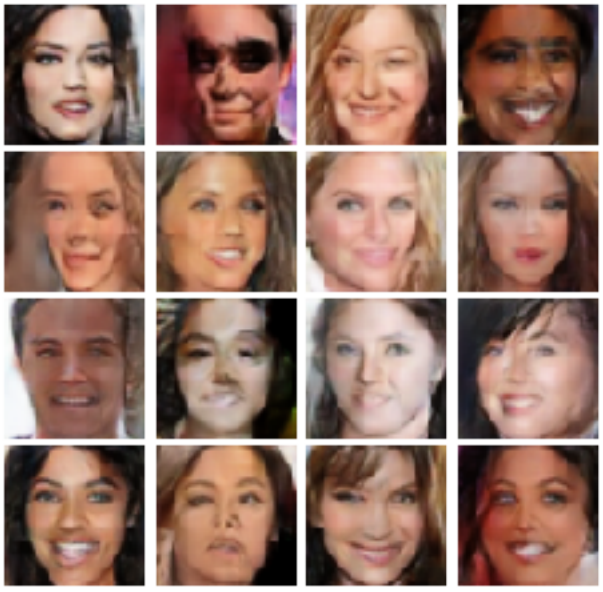
Iter: 11550, D: 0.07253, G:4.266



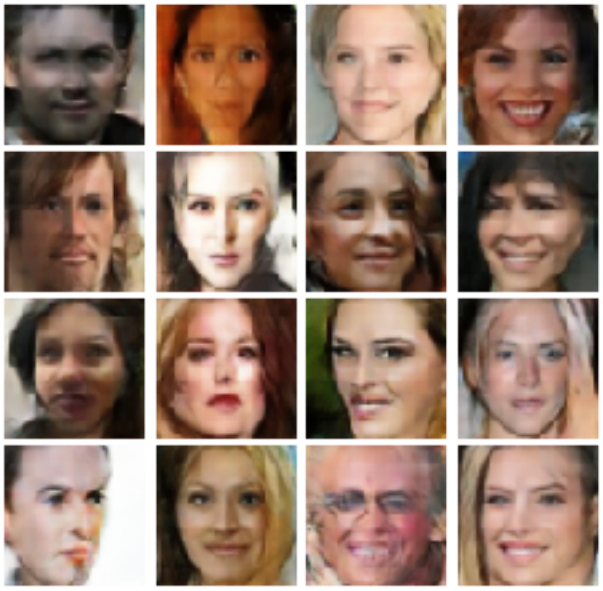
```
Iter: 11700, D: 0.03409, G:8.207
```



```
EPOCH:  13
Iter: 11850, D: 0.1043, G:4.642
```

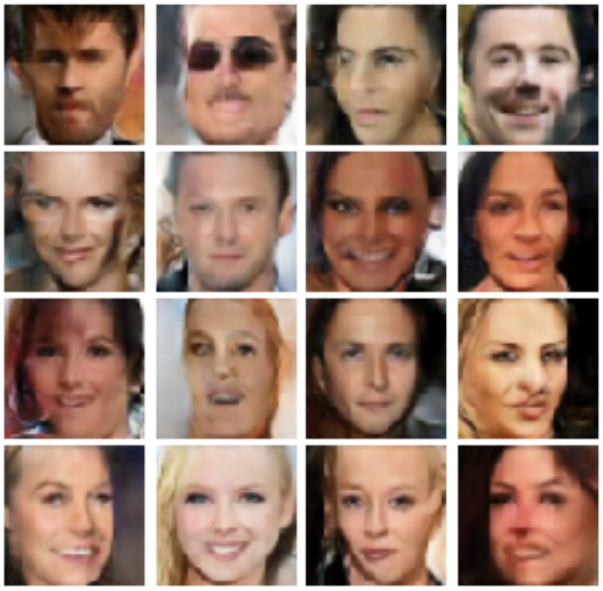


Iter: 12000, D: 0.05386, G:3.842

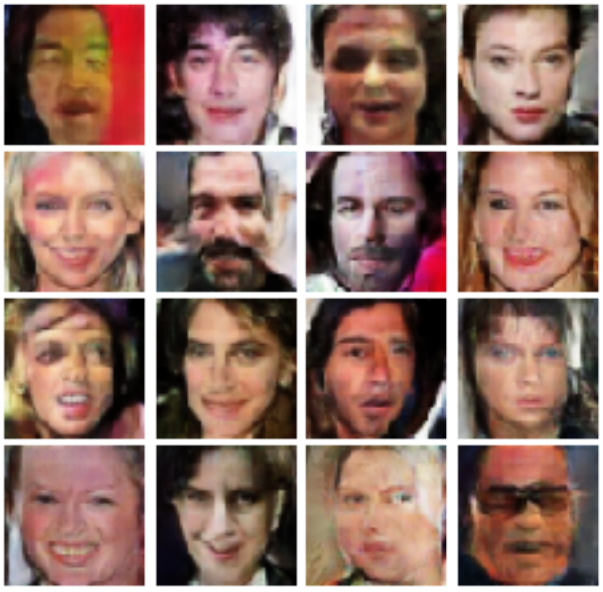


Iter: 12150, D: 0.213, G:5.146

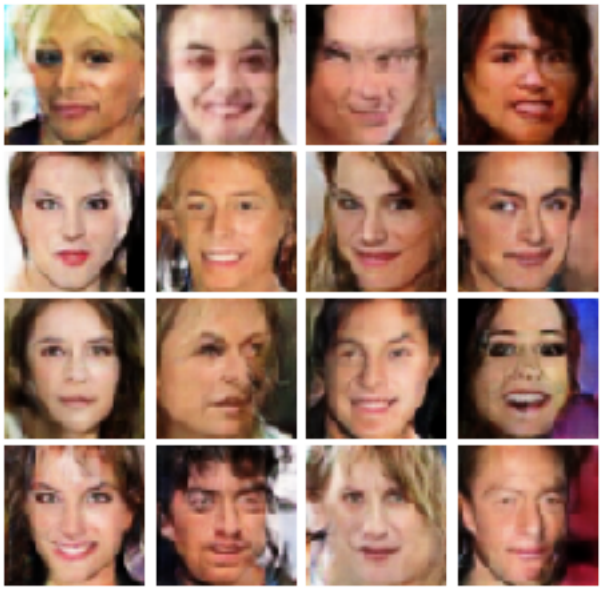
```
Iter: 12300, D: 0.1097, G:5.455
```



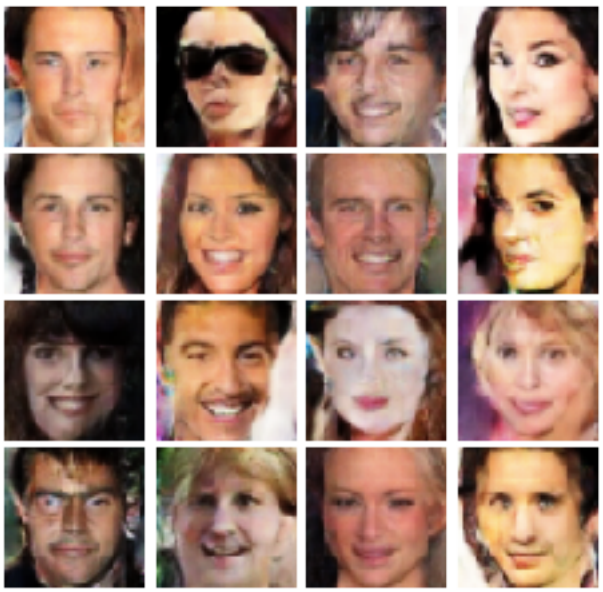
```
Iter: 12450, D: 0.139, G:5.997
```



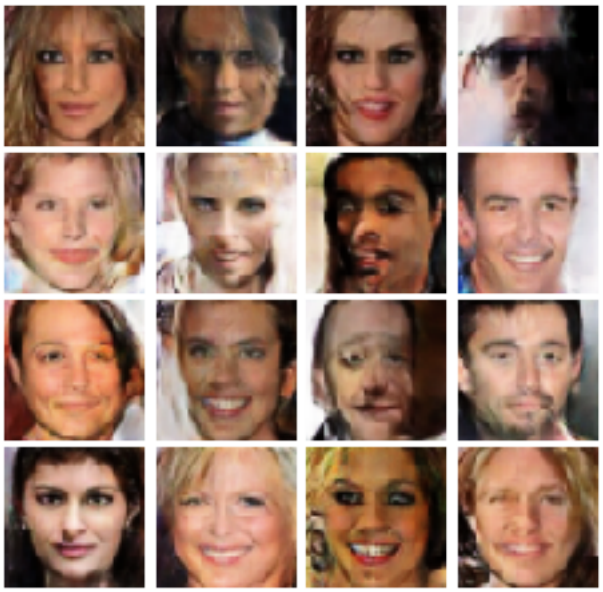
```
Iter: 12600, D: 0.2491, G:3.242
```



```
Iter: 12750, D: 0.06846, G:2.148
```

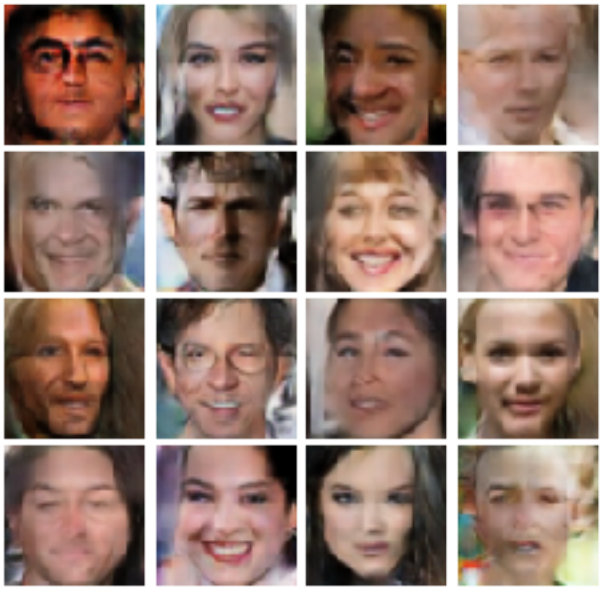


EPOCH: 14
Iter: 12900, D: 1.691, G:1.045



Iter: 13050, D: 0.1673, G:3.988

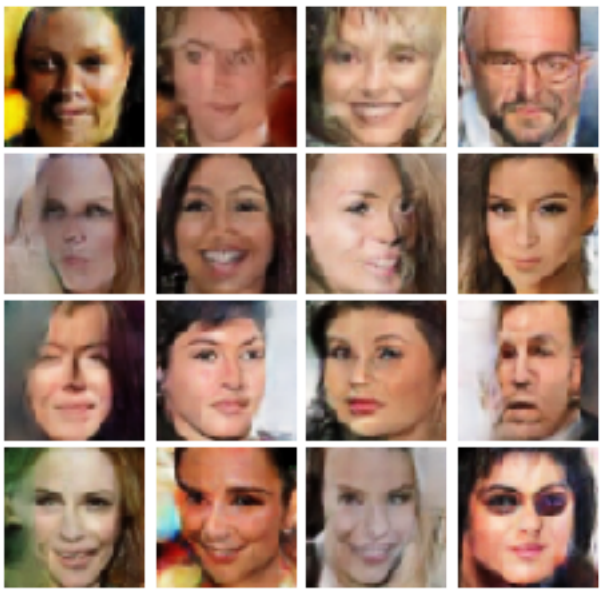
```
Iter: 13200, D: 0.7094, G:5.545
```



```
Iter: 13350, D: 0.103, G:4.158
```

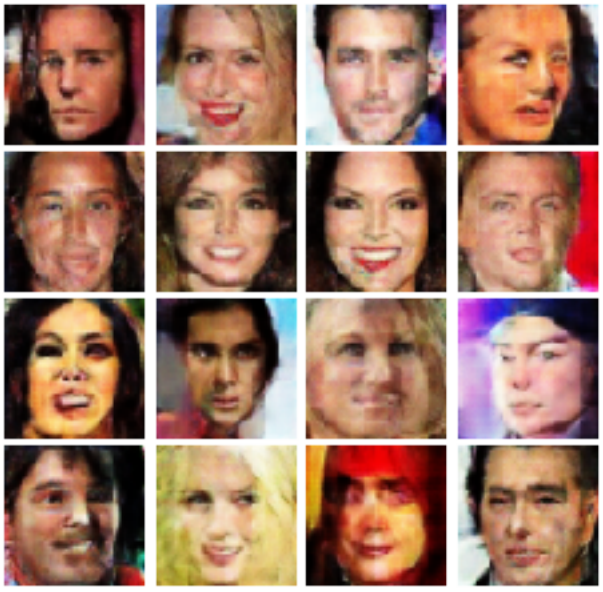
Iter: 13500, D: 0.08307, G:4.463

Iter: 13650, D: 0.05793, G:4.501

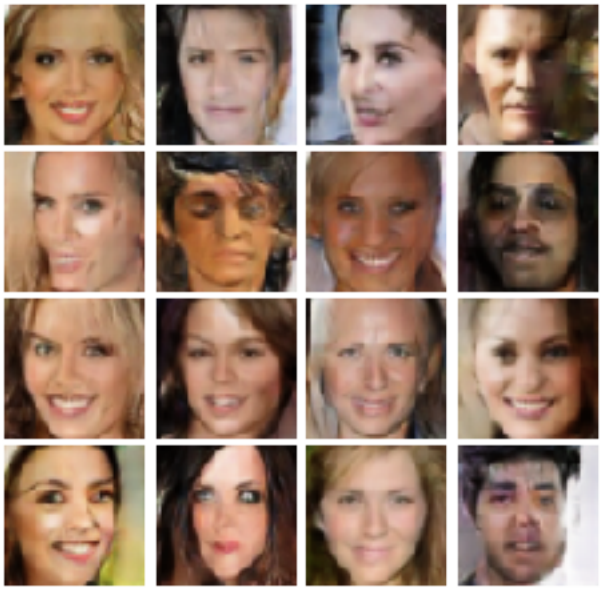


```
Iter: 13800, D: 0.02267, G:7.597
```

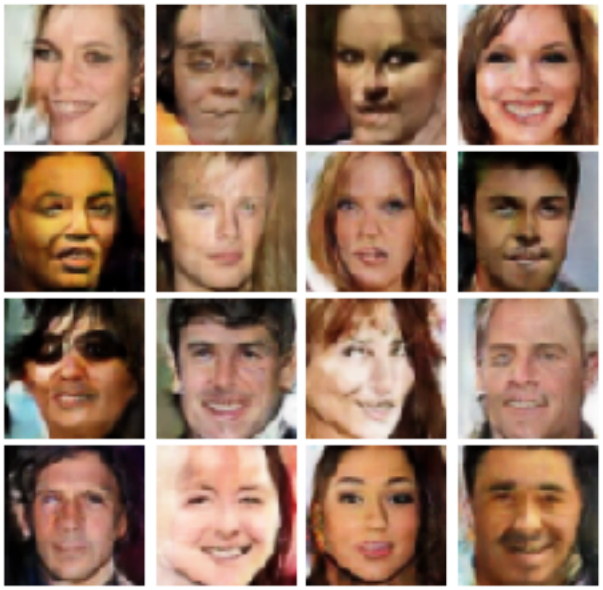
```
EPOCH:  15
Iter: 13950, D: 0.1752, G:8.92
```



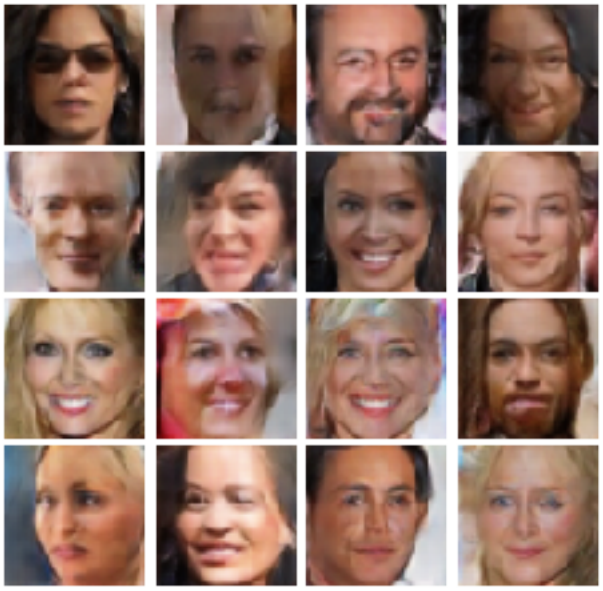
Iter: 14100, D: 0.1385, G:5.169



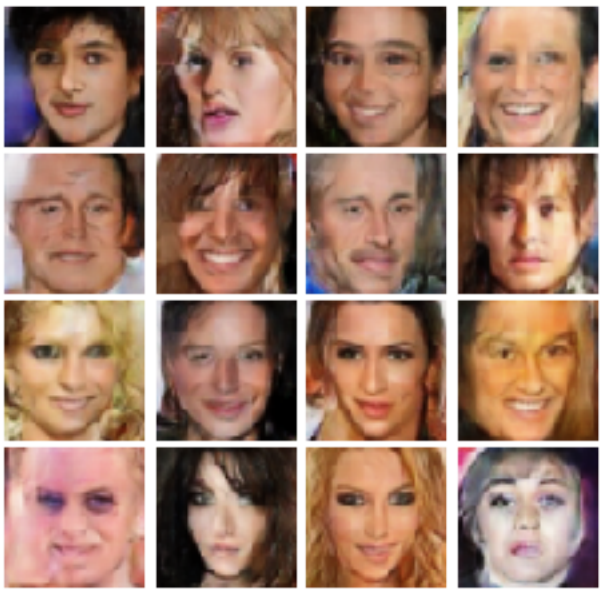
Iter: 14250, D: 0.3743, G:7.438



Iter: 14400, D: 0.4341, G:2.221

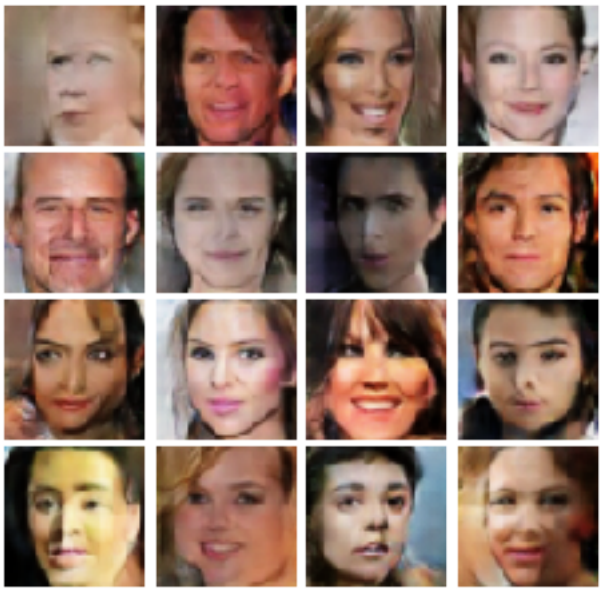


Iter: 14550, D: 0.218, G:4.549



Iter: 14700, D: 0.04625, G:5.338

Final image grid - Iter: 14790, D: 0.7842, G:6.971



[ ]:

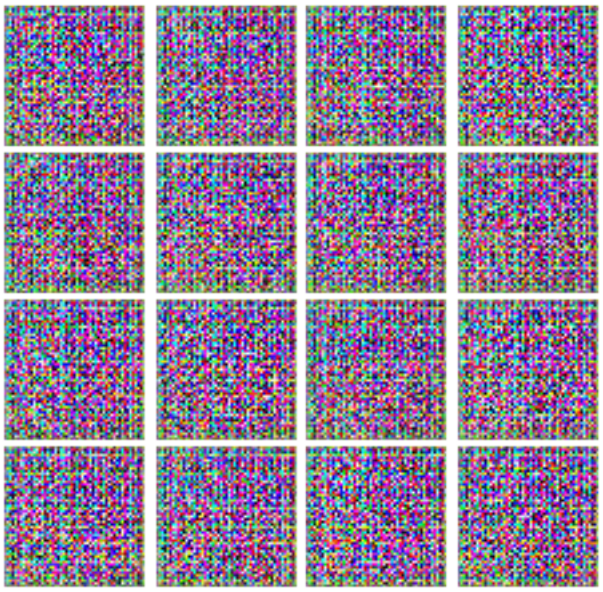
5.0.2 Train LS-GAN

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

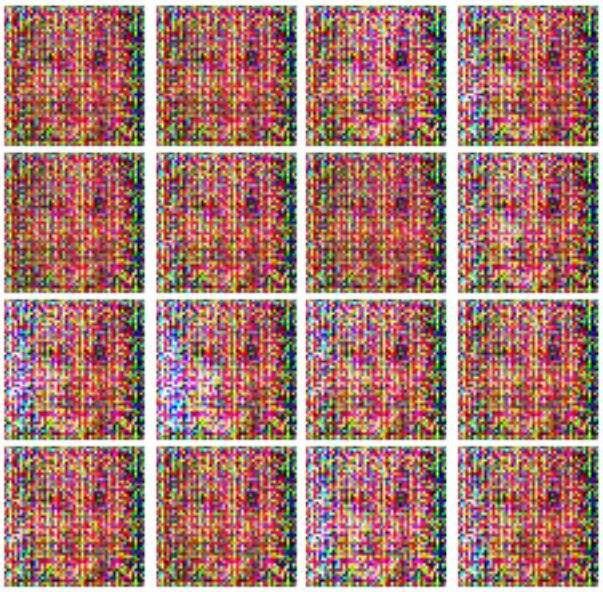
[ ]:
```
D_optimizer = torch.optim.Adam(D.parameters(), lr=learning_rate, betas = (0.5,␣
↪0.999))
G_optimizer = torch.optim.Adam(G.parameters(), lr=learning_rate, betas = (0.5,␣
↪0.999))
```

[ ]:
```
# ls-gan - spectral normaliztion
train(D, G, D_optimizer, G_optimizer, ls_discriminator_loss,
          ls_generator_loss, num_epochs=NUM_EPOCHS, show_every=200,
          train_loader=celeba_loader_train, device=device)
```

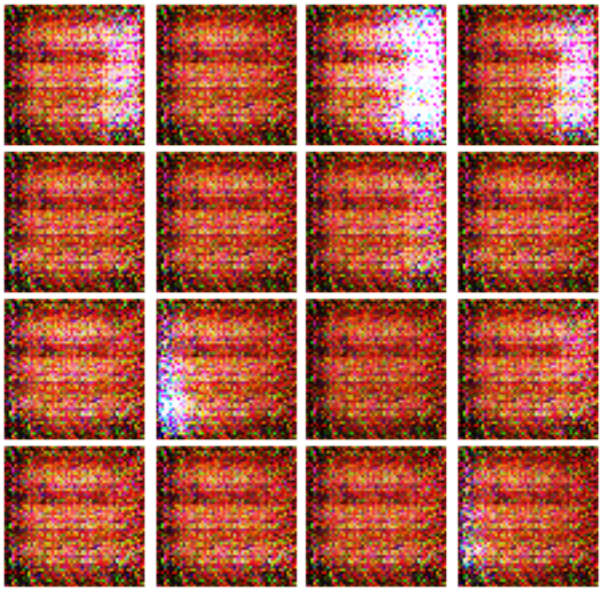
```
EPOCH:  1
Iter: 0, D: 0.3325, G:0.1793
```



```
Iter: 200, D: 0.1027, G:0.5327
```

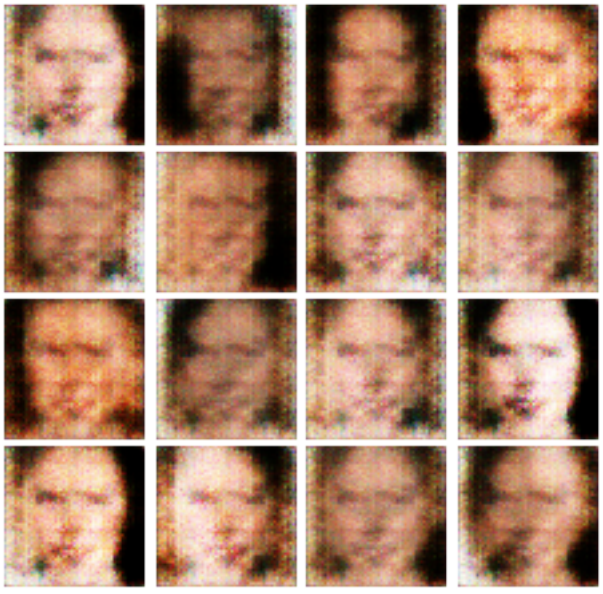


```
Iter: 400, D: 0.0109, G:0.5057
```

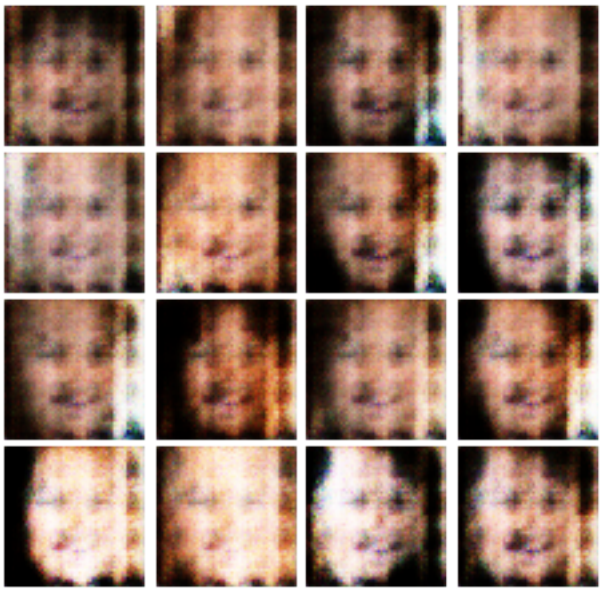


```
Iter: 600, D: 0.06717, G:0.5598
```

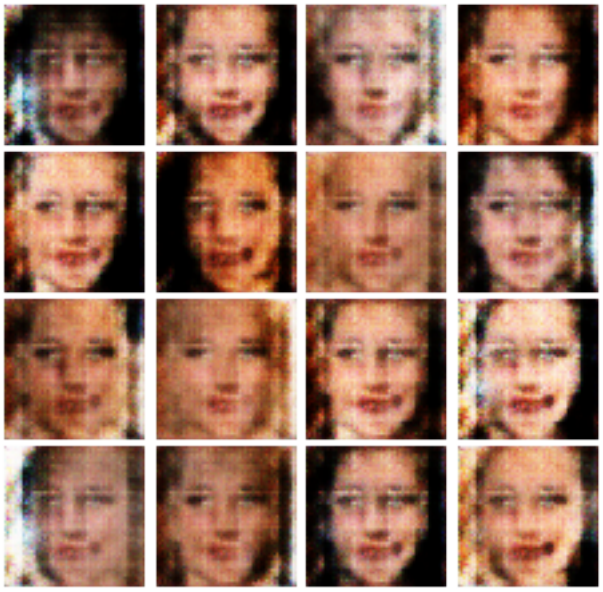
```
Iter: 800, D: 0.1666, G:0.5095
```



```
EPOCH:  2
Iter: 1000, D: 0.2161, G:0.1688
```



Iter: 1200, D: 0.08516, G:0.436



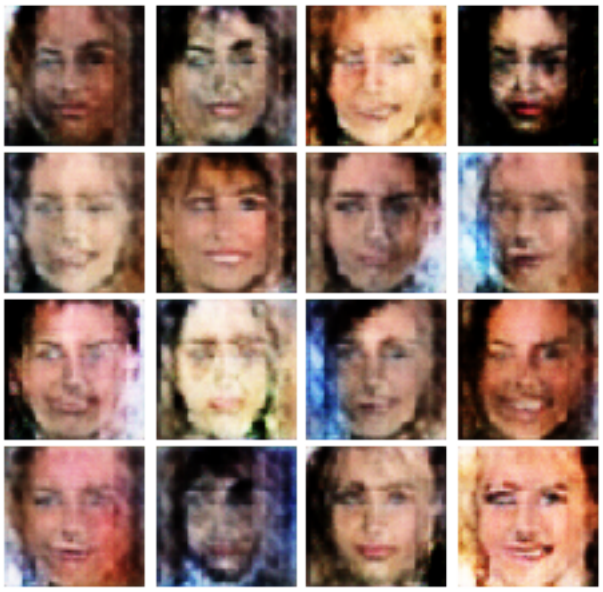
Iter: 1400, D: 0.08307, G:0.3749

Iter: 1600, D: 1.281, G:0.09224

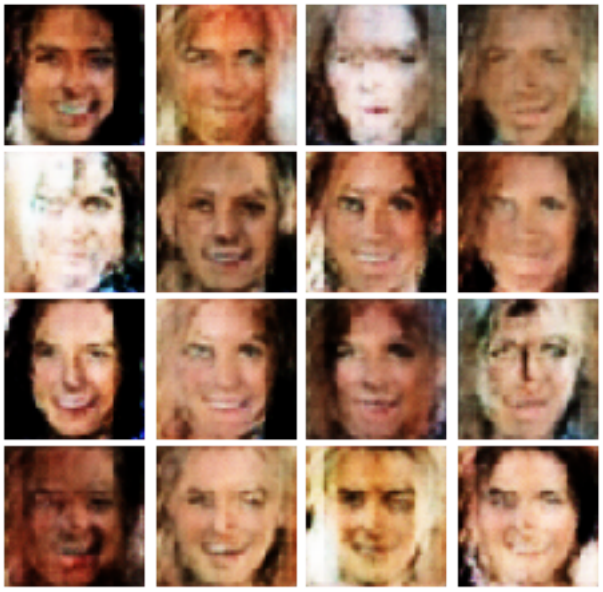
Iter: 1800, D: 0.137, G:0.2429

```
EPOCH:  3
Iter: 2000, D: 0.2557, G:0.2412
```

```
Iter: 2200, D: 0.3204, G:0.5532
```



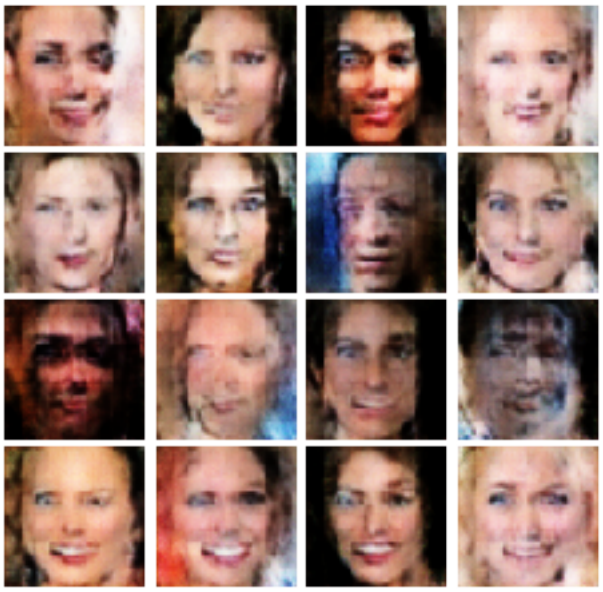
```
Iter: 2400, D: 0.1269, G:0.341
```



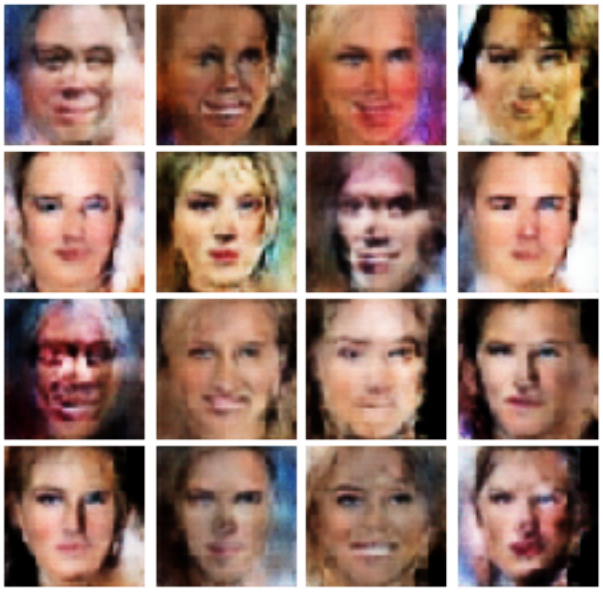
```
Iter: 2600, D: 0.4402, G:0.2344
```

```
Iter: 2800, D: 0.1316, G:0.3972
```

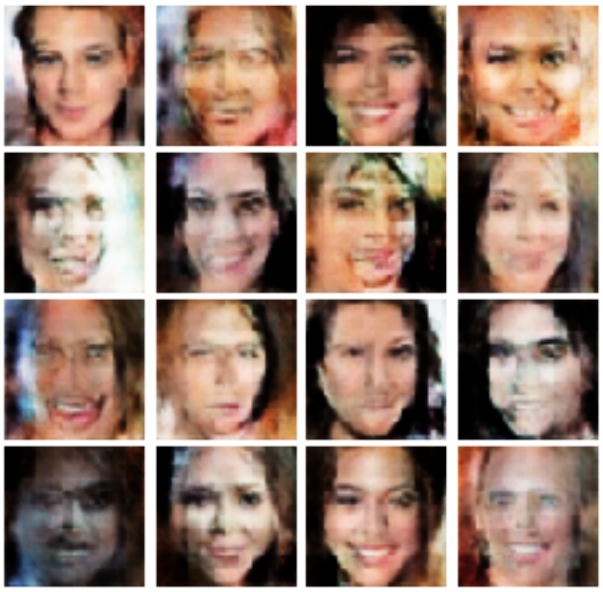
```
EPOCH:  4
Iter: 3000, D: 0.1446, G:0.5463
```



Iter: 3200, D: 0.131, G:0.3366



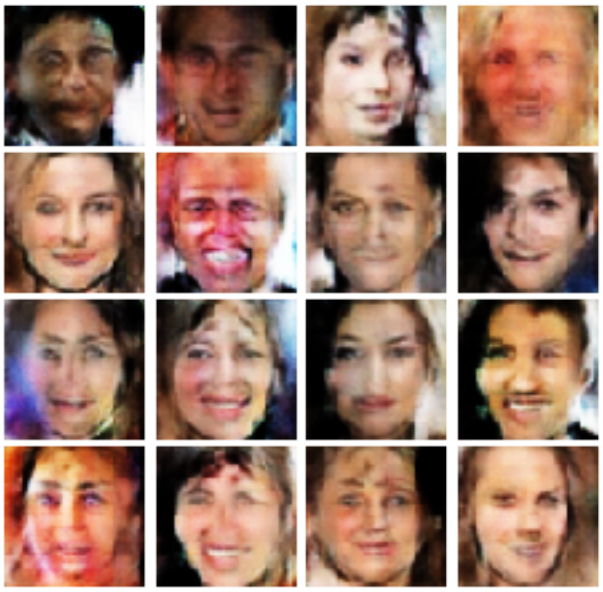
Iter: 3400, D: 0.1847, G:0.1646



Iter: 3600, D: 0.1742, G:0.4114

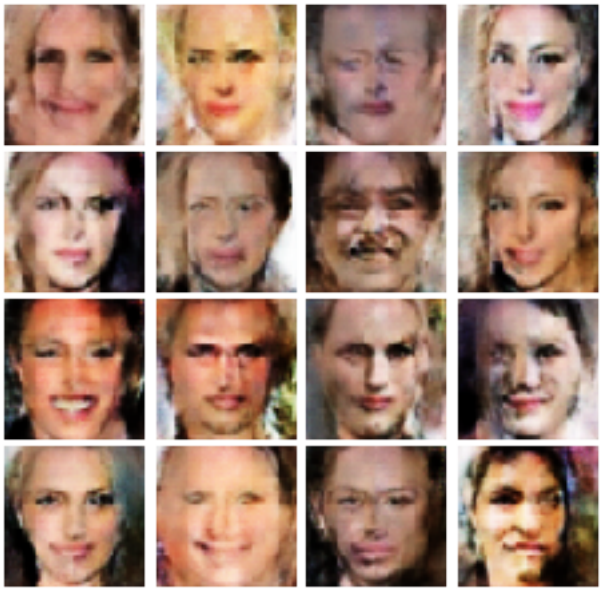
Iter: 3800, D: 0.1976, G:0.4303

```
EPOCH:  5
Iter: 4000, D: 0.4341, G:0.3214
```



```
Iter: 4200, D: 0.2254, G:0.9207
```

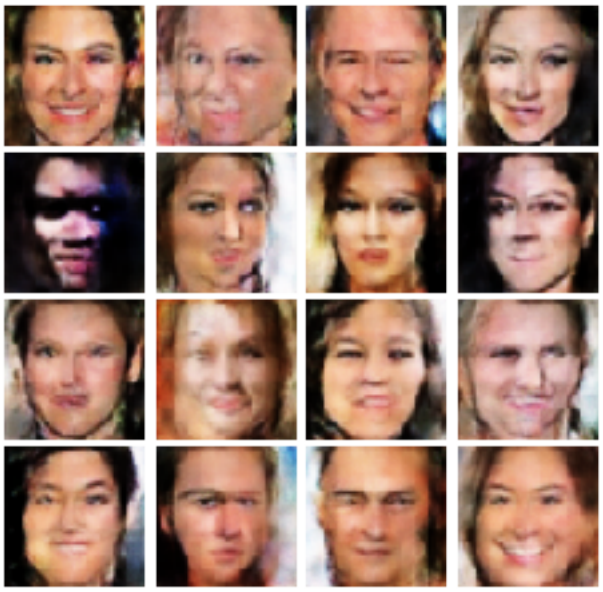
```
Iter: 4400, D: 0.1602, G:0.4675
```



```
Iter: 4600, D: 0.02914, G:0.485
```

```
Iter: 4800, D: 0.05084, G:0.7015
```

```
EPOCH:  6
Iter: 5000, D: 0.08078, G:0.3763
```

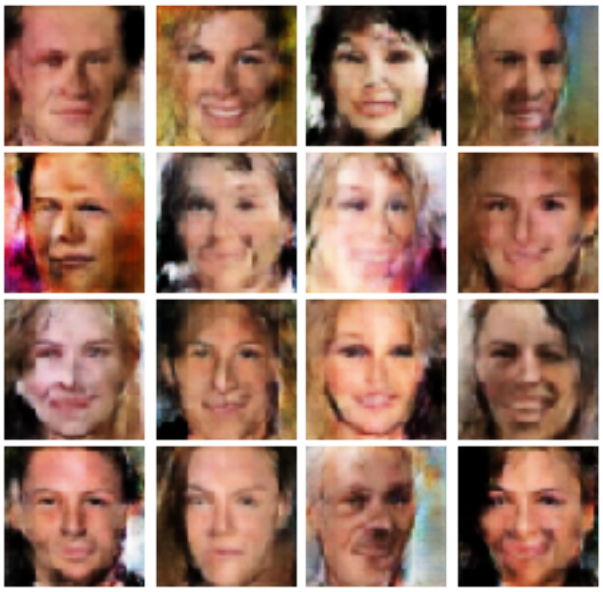


```
Iter: 5200, D: 0.1033, G:1.069
```

```
Iter: 5400, D: 0.1785, G:0.9083
```

```
Iter: 5600, D: 0.03581, G:0.4252
```

```
Iter: 5800, D: 0.1095, G:0.2689
```



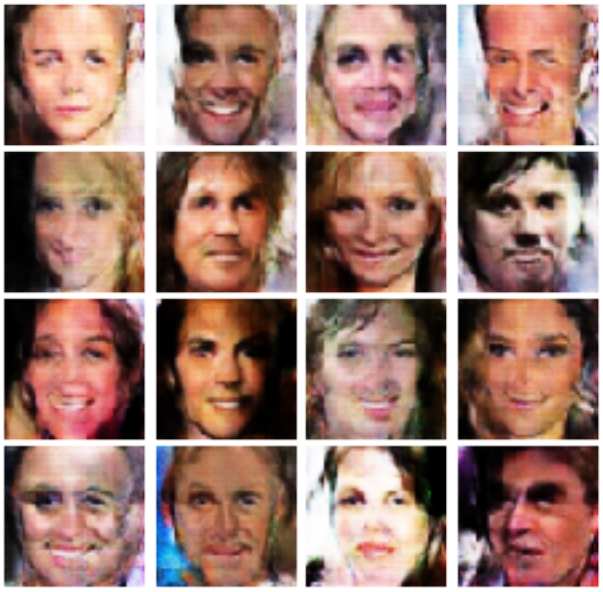
```
EPOCH:  7
Iter: 6000, D: 0.05748, G:0.5118
```

```
Iter: 6200, D: 0.06896, G:0.4717
```

Iter: 6400, D: 0.05794, G:0.4825

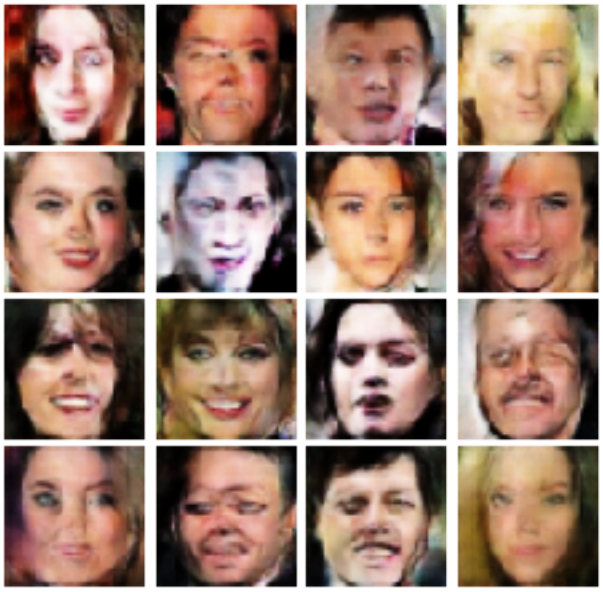
Iter: 6600, D: 0.06395, G:0.2904

```
Iter: 6800, D: 0.03844, G:0.4197
```



```
EPOCH:  8
Iter: 7000, D: 0.09331, G:0.3998
```

Iter: 7200, D: 0.05645, G:0.543



Iter: 7400, D: 0.03876, G:0.2877

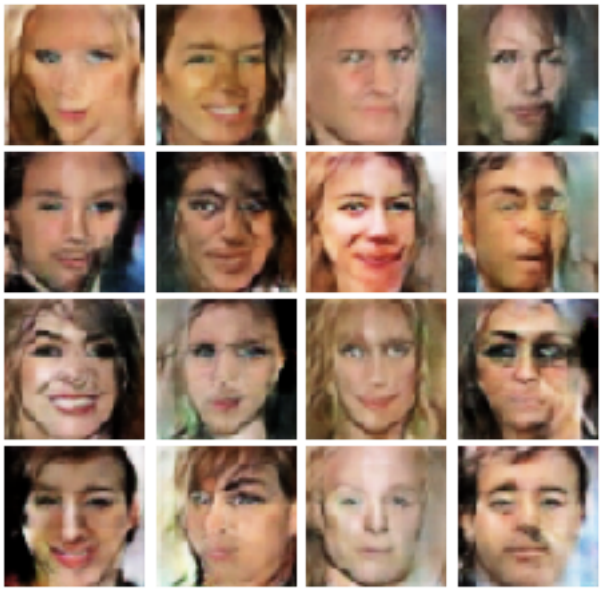
Iter: 7600, D: 0.07393, G:0.3771

Iter: 7800, D: 0.08661, G:0.3906

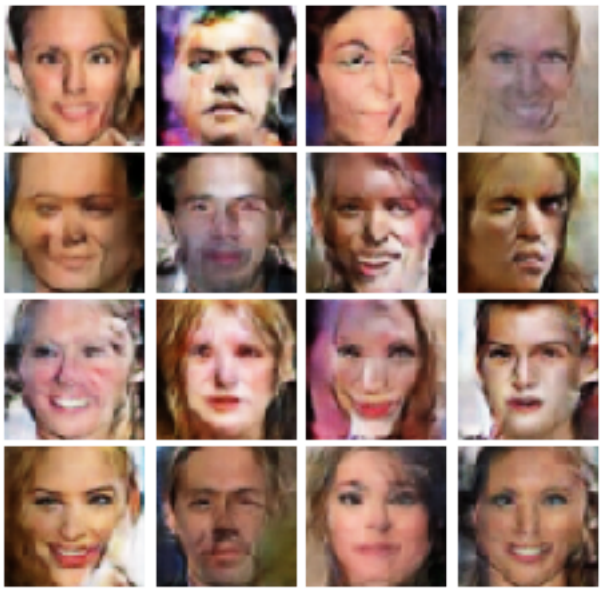
EPOCH: 9
Iter: 8000, D: 0.1779, G:0.2587

Iter: 8200, D: 0.03736, G:0.3427

Iter: 8400, D: 0.1345, G:0.3437

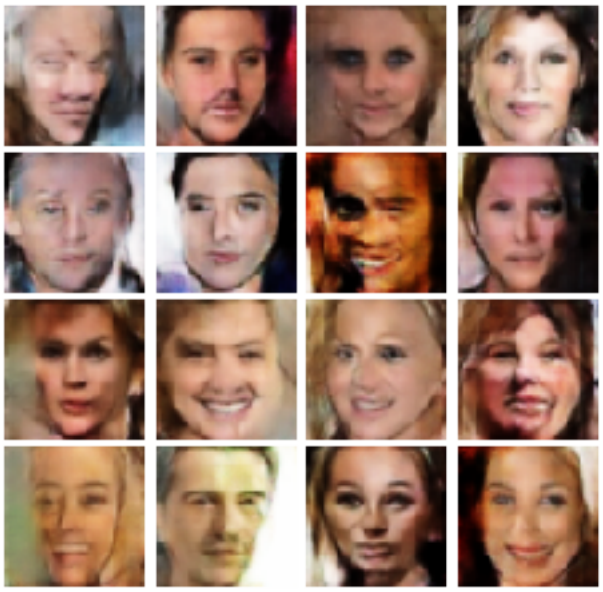


Iter: 8600, D: 0.3797, G:0.2068



```
Iter: 8800, D: 0.04186, G:0.7001
```

```
EPOCH:  10
Iter: 9000, D: 0.02014, G:0.4658
```

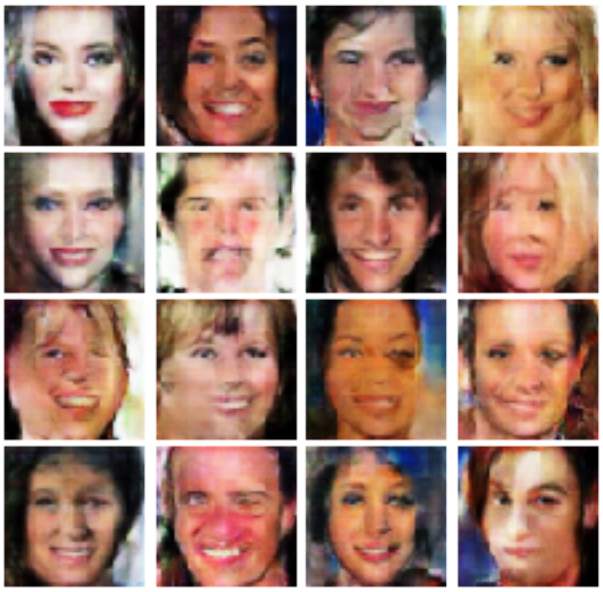


Iter: 9200, D: 0.07356, G:0.4286

Iter: 9400, D: 0.04038, G:0.5274

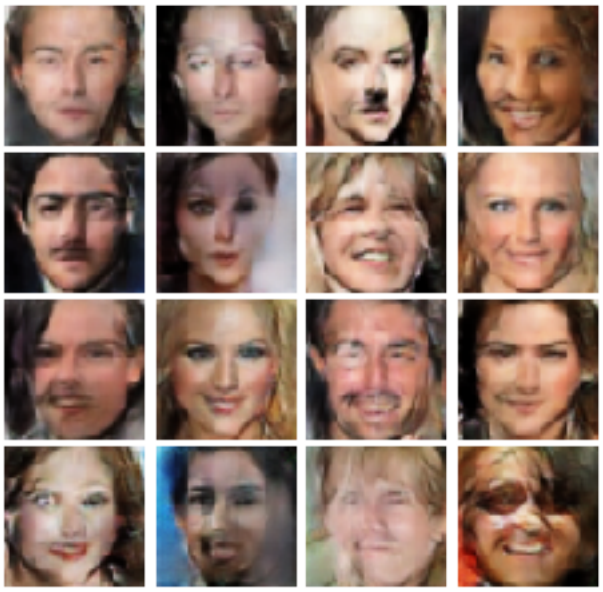
Iter: 9600, D: 0.02959, G:0.5609

Iter: 9800, D: 0.08957, G:0.7525

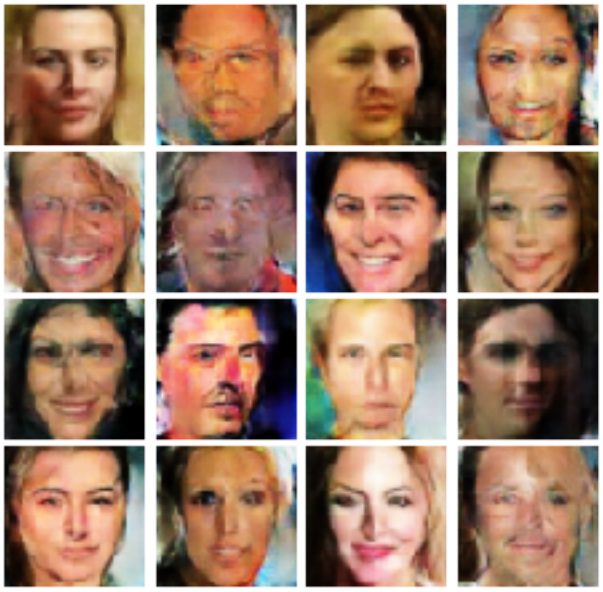


```
EPOCH:  11
Iter: 10000, D: 0.0333, G:0.255
```

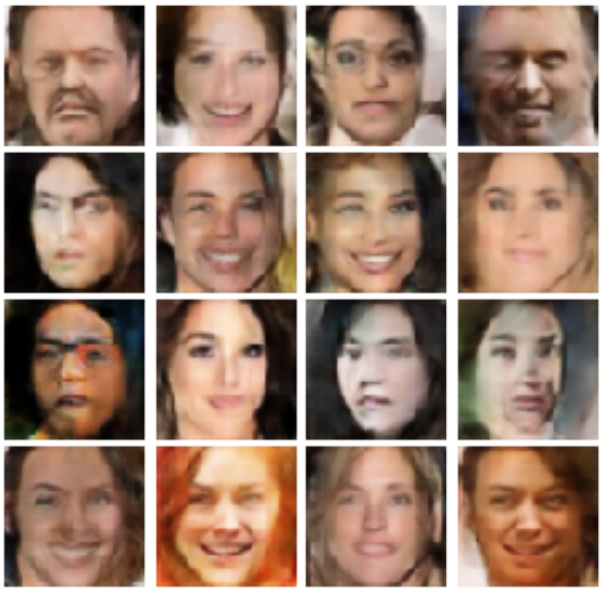
```
Iter: 10200, D: 0.07882, G:0.5127
```



Iter: 10400, D: 0.05538, G:0.6367

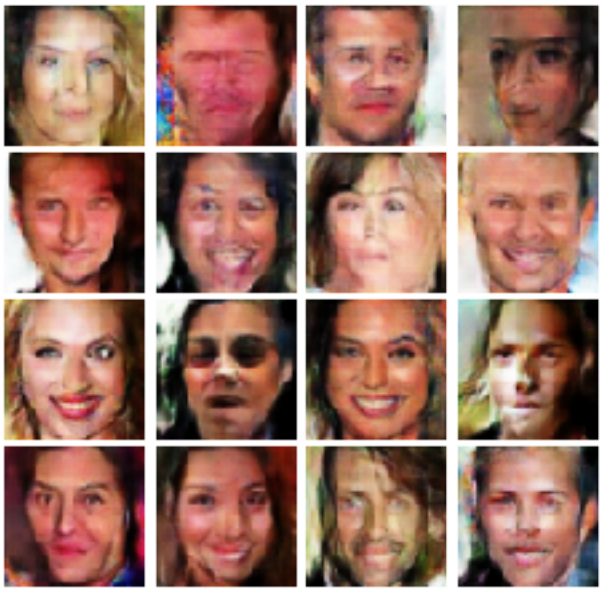


Iter: 10600, D: 0.09575, G:0.312

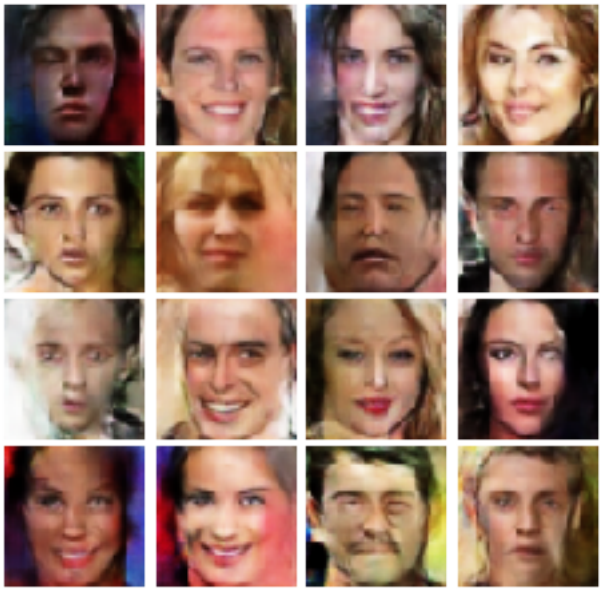


```
Iter: 10800, D: 0.02387, G:0.5169
```

```
EPOCH:  12
Iter: 11000, D: 0.07836, G:0.3941
```

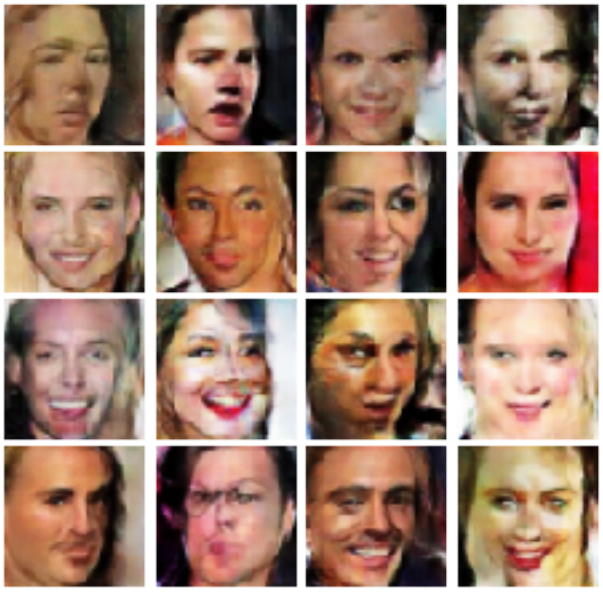


```
Iter: 11200, D: 0.03676, G:0.3295
```

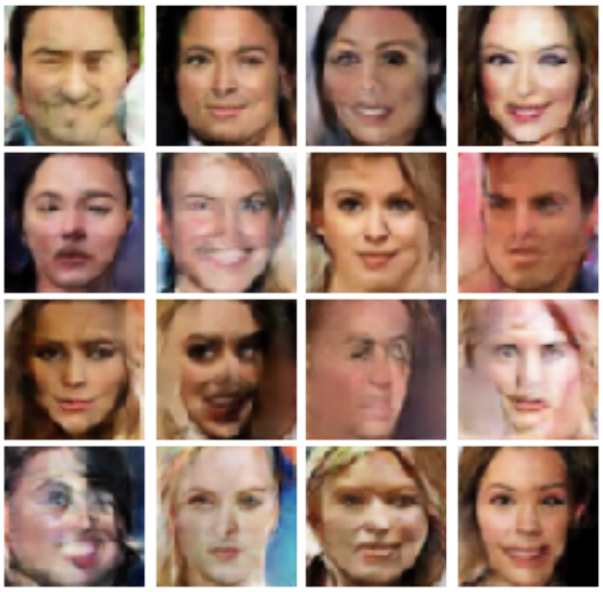


```
Iter: 11400, D: 0.0429, G:0.3114
```

Iter: 11600, D: 0.0984, G:0.7645

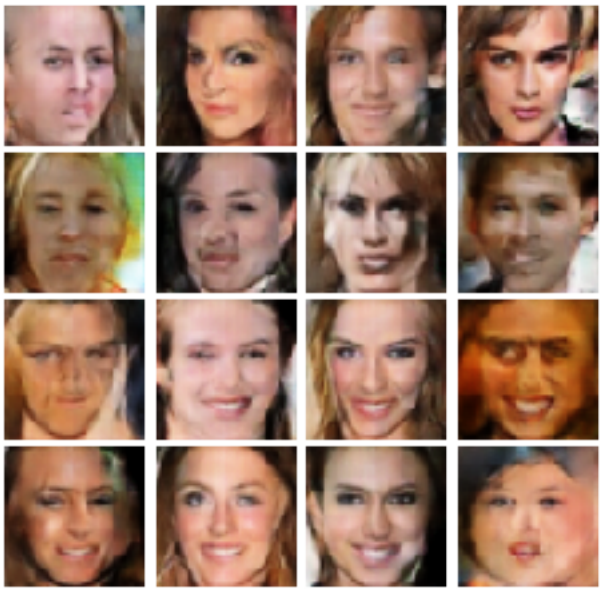


Iter: 11800, D: 0.03191, G:0.4425

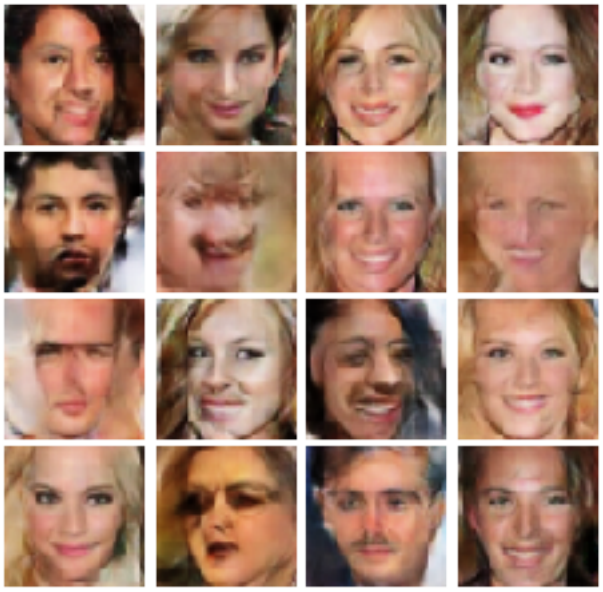


EPOCH: 13
Iter: 12000, D: 0.0214, G:0.5287

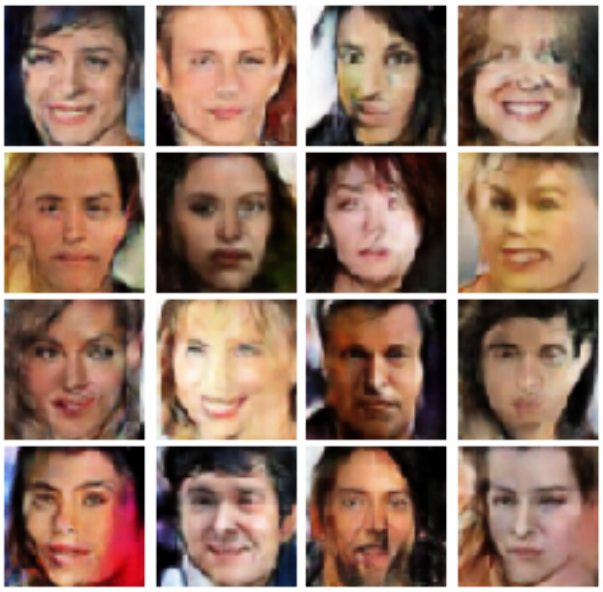
Iter: 12200, D: 0.0621, G:0.4768



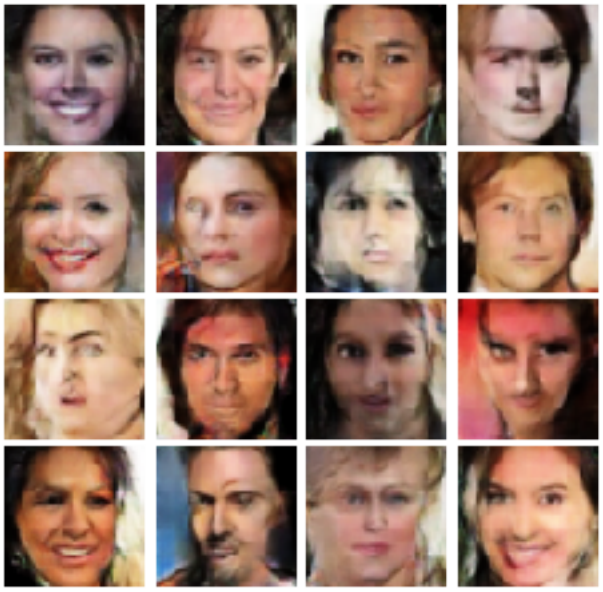
Iter: 12400, D: 0.07967, G:0.4969



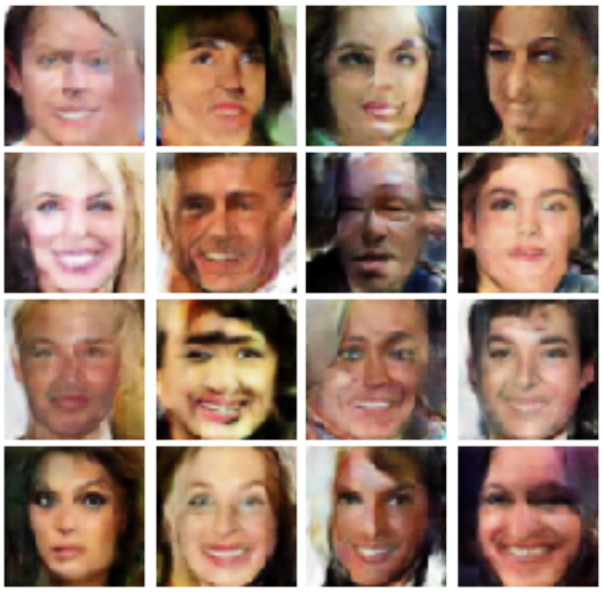
Iter: 12600, D: 0.05732, G:0.7469



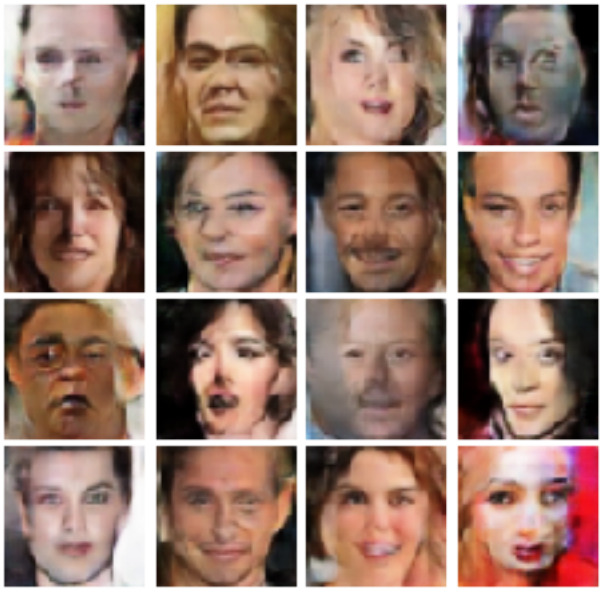
Iter: 12800, D: 0.03189, G:0.2925



EPOCH: 14
Iter: 13000, D: 0.02664, G:0.4641



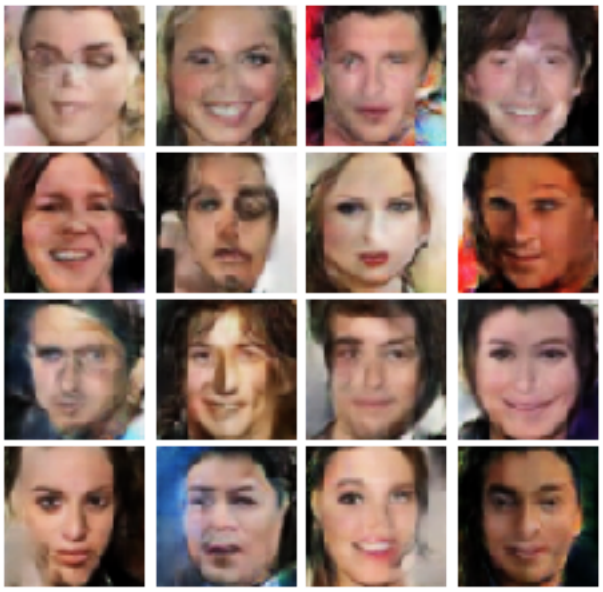
Iter: 13200, D: 0.03719, G:0.3895



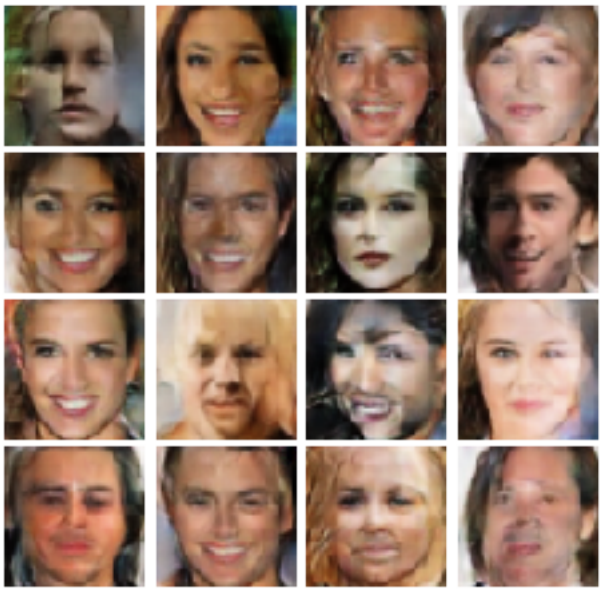
Iter: 13400, D: 0.07297, G:0.5169

Iter: 13600, D: 0.09203, G:0.6784

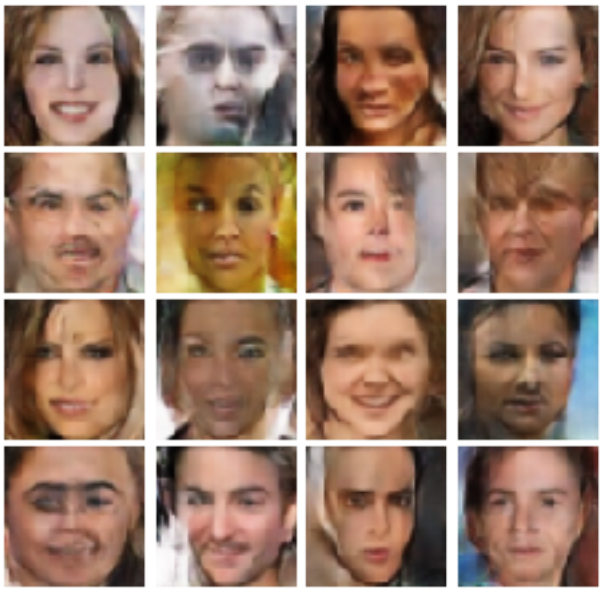
Iter: 13800, D: 0.0642, G:0.3263



EPOCH: 15
Iter: 14000, D: 0.05362, G:0.4279

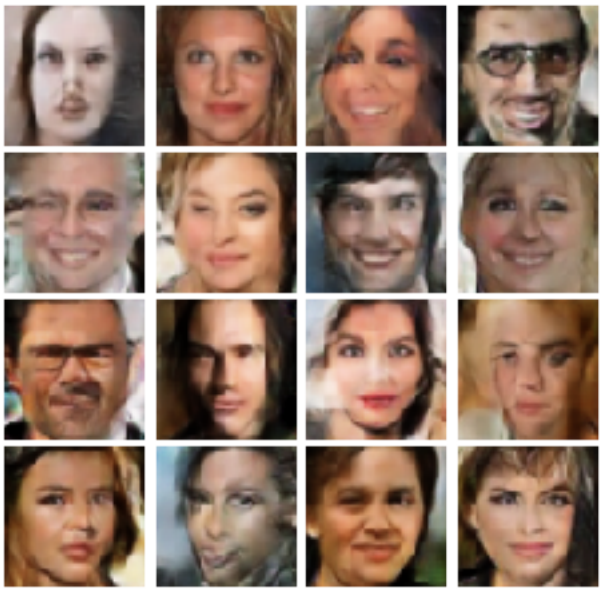


Iter: 14200, D: 0.04428, G:0.4293

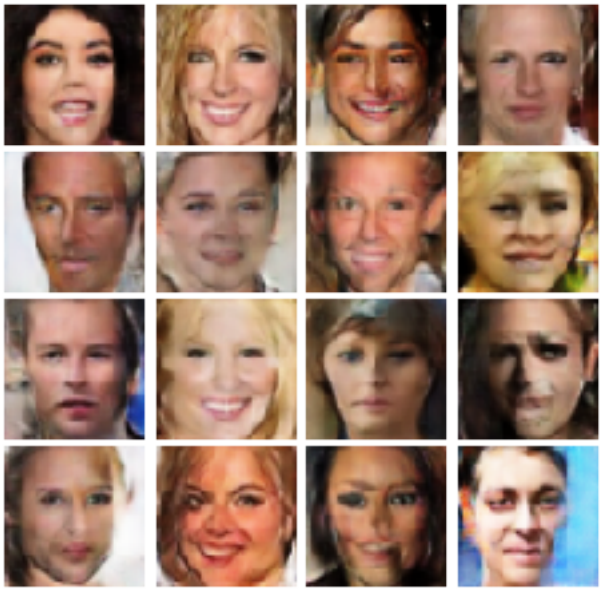


```
Iter: 14400, D: 0.04158, G:0.5897
```

```
Iter: 14600, D: 0.02051, G:0.4097
```



```
Final image grid - Iter: 14790, D: 0.02369, G:0.381
```



[ ]: