part1

May 4, 2024

1 CS 5814 Homework 4, Part 1: Generative Adversarial Networks

[121]:
```
import torch
import os
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision.datasets import ImageFolder
import matplotlib.pyplot as plt
import numpy as np
os.environ['TORCH_HOME'] = '/home/aishwaryak/.cache/torch'

%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
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[122]:
```
from gan.train import train
```

[123]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "gpu")
```

[124]:
```
print(device)
```

```
cuda:0
```

2 GAN loss functions

You'll need to implement two different loss functions. The first is the loss from the original GAN paper. The next is the loss from LS-GAN.

2.0.1 GAN loss

TODO: You need to 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:

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

Note that these equations are negated because we will be *minimizing* these losses.

HINTS: Use the `torch.nn.functional.binary_cross_entropy_with_logits` function to compute the binary cross entropy loss since it is more numerically stable than using a softmax followed by BCE loss.

We will be averaging over the elements of the minibatch instead of computing the expectation of $\log D(G(z))$, $\log D(x)$ and $\log\left(1 - D(G(z))\right)$.

[125]:
```
from gan.losses import discriminator_loss, generator_loss
```

2.0.2 Least Squares GAN loss

TODO: You need to implement the `ls_discriminator_loss` and `ls_generator_loss` functions in `gan/losses.py`.

Least Squares GAN is an alernative to the original GAN loss function. For this part, all you need to do change the loss function and retrain the model.

Specifically, you'll implement equation (9) in the paper:

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

and the discriminator loss:

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

[126]:
```
from gan.losses import ls_discriminator_loss, ls_generator_loss
```

3 GAN model architecture

TODO: Next, you'll need to implement the `Discriminator` and `Generator` networks in `gan/models.py`.

We'll be using the architecture from 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.

You can either use LeakyReLu throughout the discriminator (and a negative slope value of 0.2) or just use relu.

The discriminator will output a single score for each sample.The output of your discriminator should be a single value score corresponding to each input sample.

Generator:

Note: Here, you'll need to use transposed convolution (sometimes known as fractionally-strided convolution). 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 generator network will need to map the output values between -1 and 1 using a `tanh` nonlinearity. The output should be of size 64x64 with 3 channels for each sample (equal dimensions to the images from the dataset).

[127]:
```
from gan.models import Discriminator, Generator
```

4 Data loading

Our dataset is pretty small, so in order to prevent overfitting, we need to perform data augmentation.

TODO: You'll need to implement some data augmentation by adding new transforms to the cell below.

The easiest you can do is just use the RandomCrop and ColorJitter, but feel free to play around with other augmentations as well to see how it affects the performance of your model. See https://pytorch.org/vision/stable/transforms.html.

[139]:
```
batch_size = 32
imsize = 64
cat_root = './cats'

cat_train = ImageFolder(root=cat_root, transform=transforms.Compose([
  transforms.Resize(imsize),
  transforms.CenterCrop(imsize),
  transforms.RandomCrop(imsize, padding=4),
  transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5, hue=0.1),
  transforms.ToTensor(),
#   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
]))

cat_loader_train = DataLoader(cat_train, batch_size=batch_size, drop_last=True)
```

[140]:
```
from gan.utils import *
```

4.0.1 Visualize dataset

[141]:
```
from gan.utils import show_images

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

5 Training

TODO: You need to write the training loop in `gan/train.py`.

[142]:
```
NOISE_DIM = 100
NUM_EPOCHS = 50
learning_rate = 0.001
```

5.0.1 Train GAN

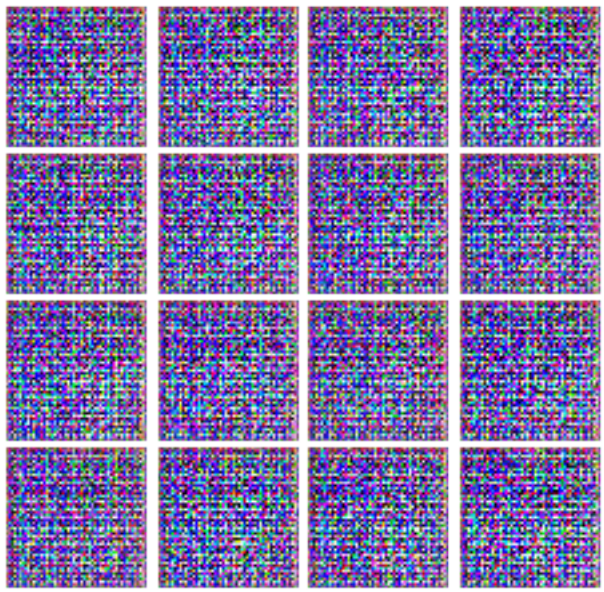
[143]:
```
D = Discriminator().to(device)
G = Generator(noise_dim=NOISE_DIM).to(device)
```

[144]:
```
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))
```

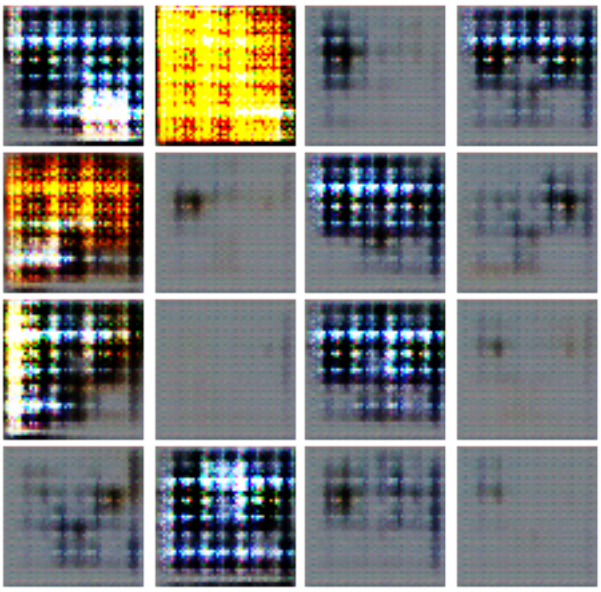
[145]:
```
from gan.utils import preprocess_img
```

[146]:
```
# original gan
train(D, G, D_optimizer, G_optimizer, discriminator_loss,
          generator_loss, num_epochs=NUM_EPOCHS, show_every=250,
          batch_size=batch_size, train_loader=cat_loader_train, device=device)
```

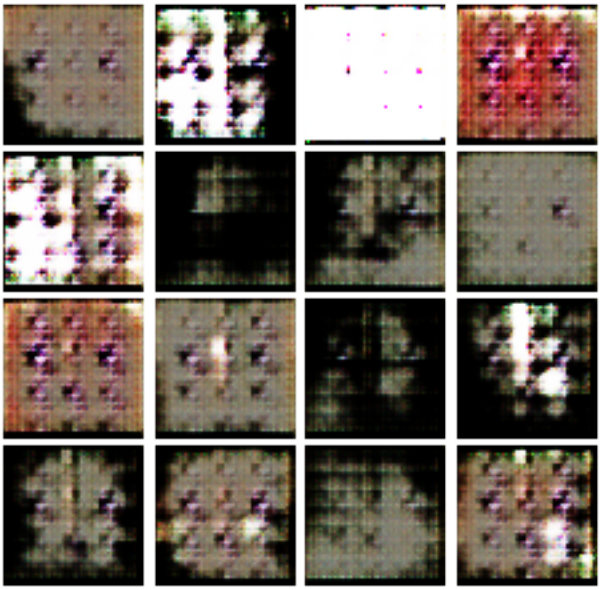
```
EPOCH:  1
Iter: 0, D: 0.7475, G:8.684
```



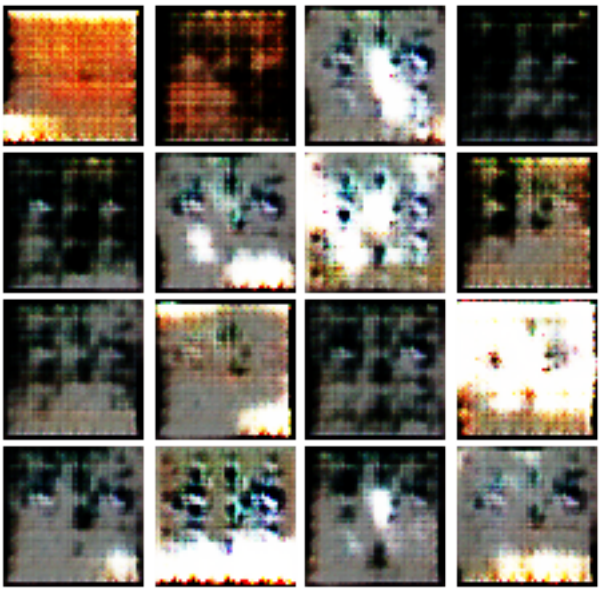
```
Iter: 250, D: 0.2529, G:4.715
```



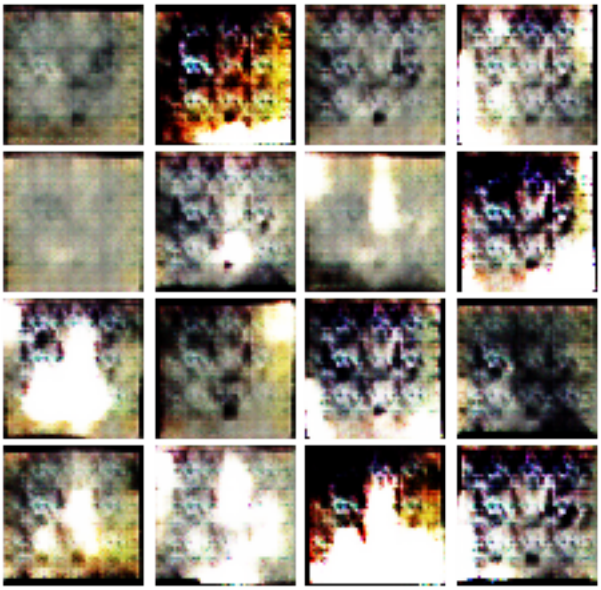
EPOCH: 2
Iter: 500, D: 1.281, G:8.908



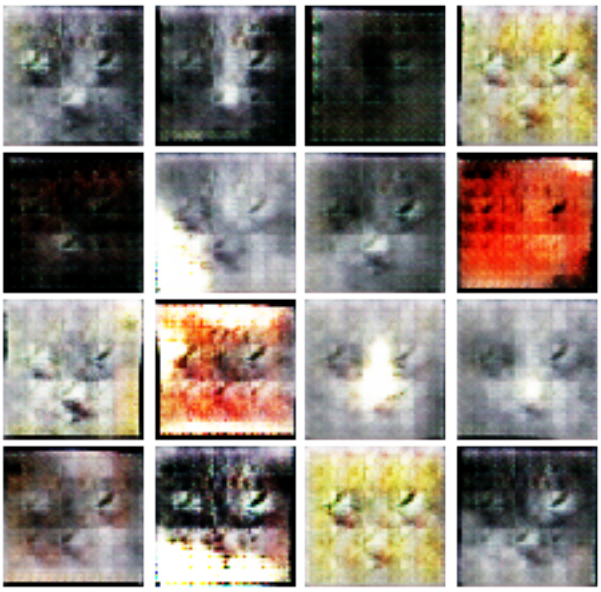
Iter: 750, D: 0.324, G:6.605



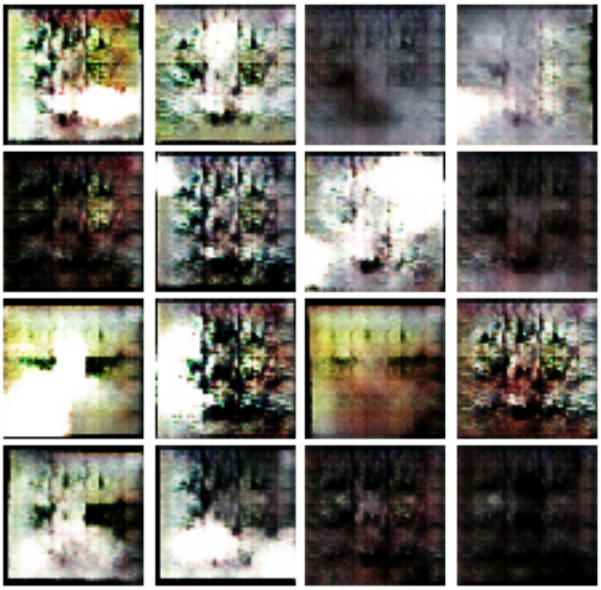
EPOCH: 3
Iter: 1000, D: 0.6689, G:3.135



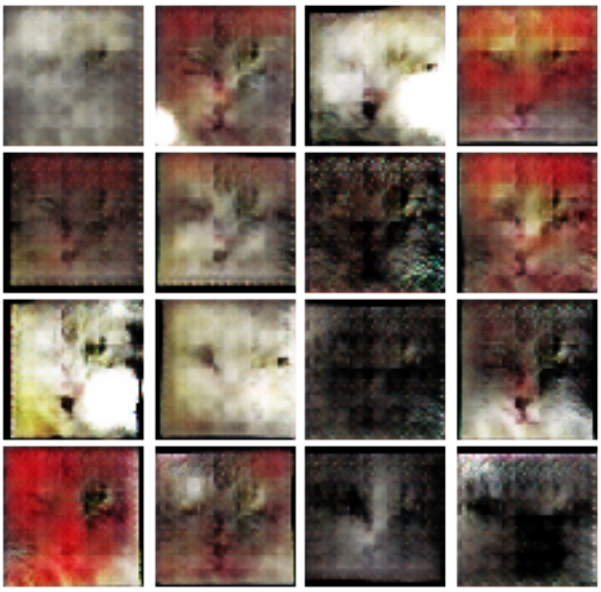
Iter: 1250, D: 0.3831, G:3.279



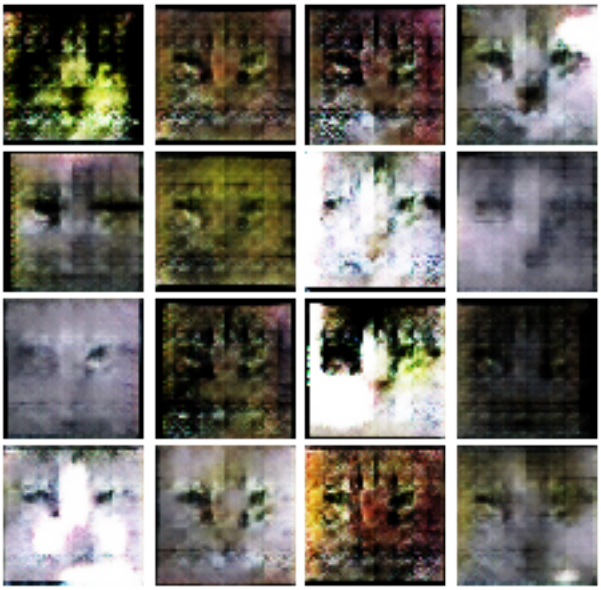
EPOCH: 4
Iter: 1500, D: 0.4782, G:5.237



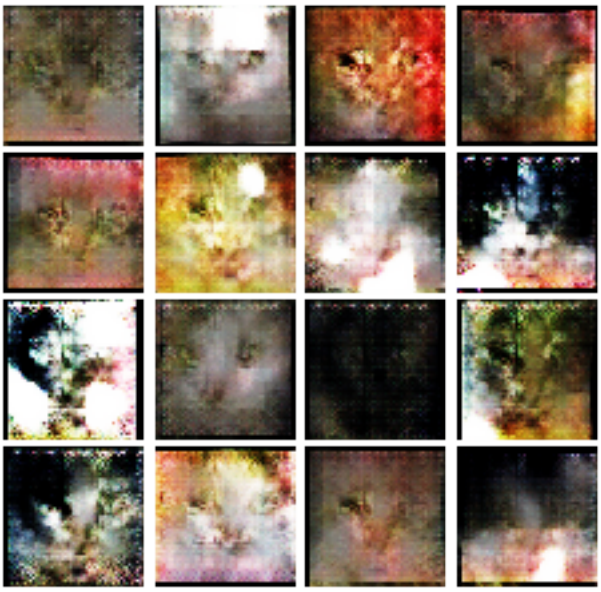
Iter: 1750, D: 0.3446, G:4.044



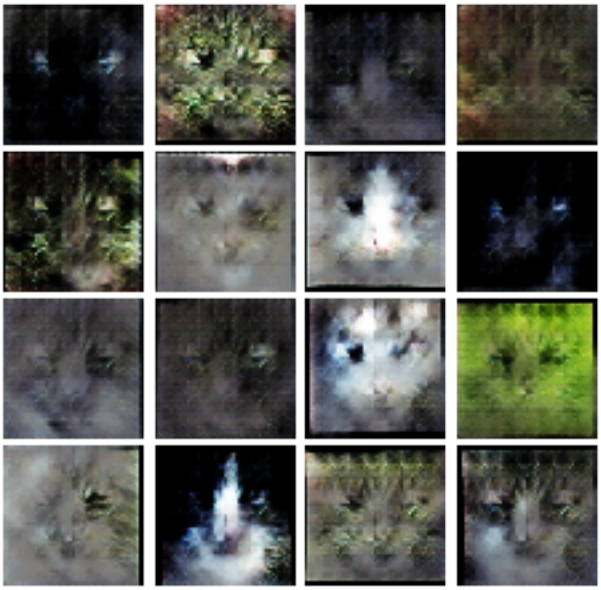
EPOCH: 5
Iter: 2000, D: 0.4857, G:3.462



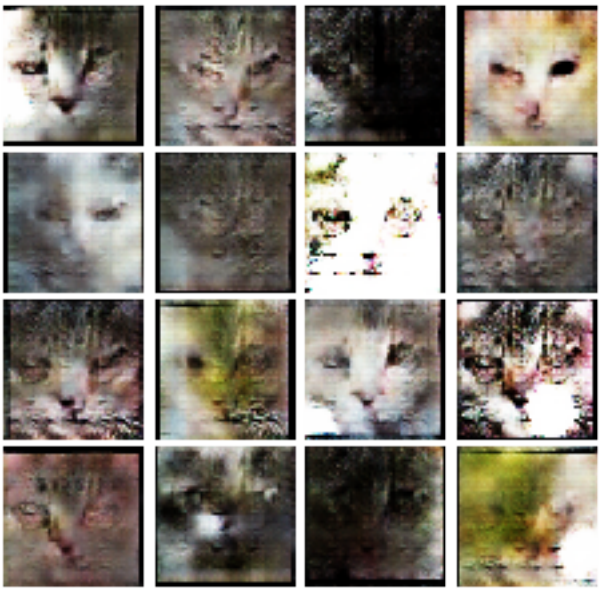
Iter: 2250, D: 0.5294, G:3.04



EPOCH: 6
Iter: 2500, D: 0.3816, G:2.435

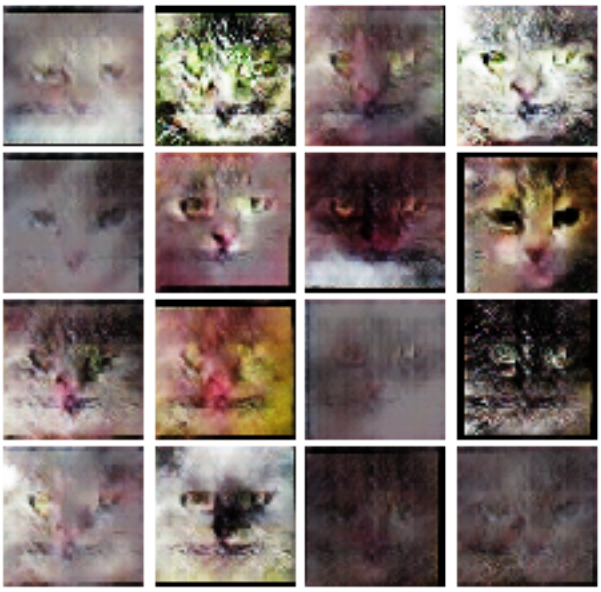


Iter: 2750, D: 0.2639, G:2.303



```
EPOCH:  7
Iter: 3000, D: 0.6637, G:3.232
```

```
Iter: 3250, D: 0.3997, G:1.424
```



EPOCH: 8
Iter: 3500, D: 0.4362, G:3.249

Iter: 3750, D: 0.4862, G:4.118

```
EPOCH:  9
Iter: 4000, D: 0.3758, G:2.504
```

```
Iter: 4250, D: 0.3411, G:4.129
```

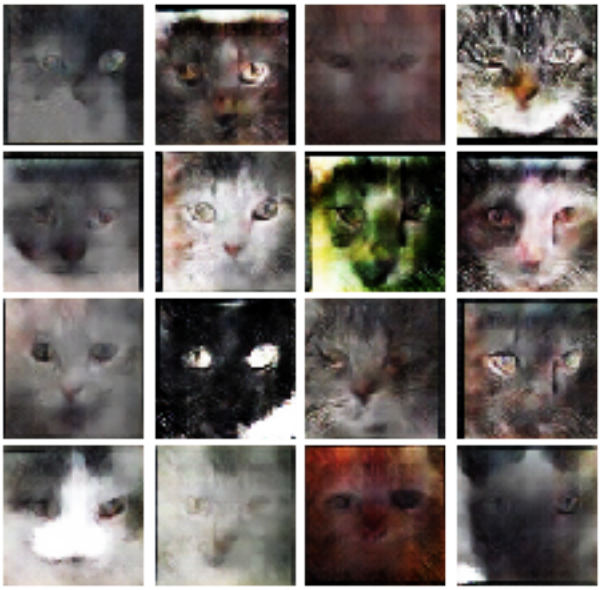
EPOCH: 10
Iter: 4500, D: 0.24, G:5.295

Iter: 4750, D: 0.4429, G:4.447

EPOCH: 11
Iter: 5000, D: 0.4726, G:3.083

Iter: 5250, D: 0.2, G:1.917

EPOCH: 12
Iter: 5500, D: 0.4175, G:3.602



Iter: 5750, D: 0.3426, G:2.888

EPOCH: 13
Iter: 6000, D: 0.2863, G:3.736

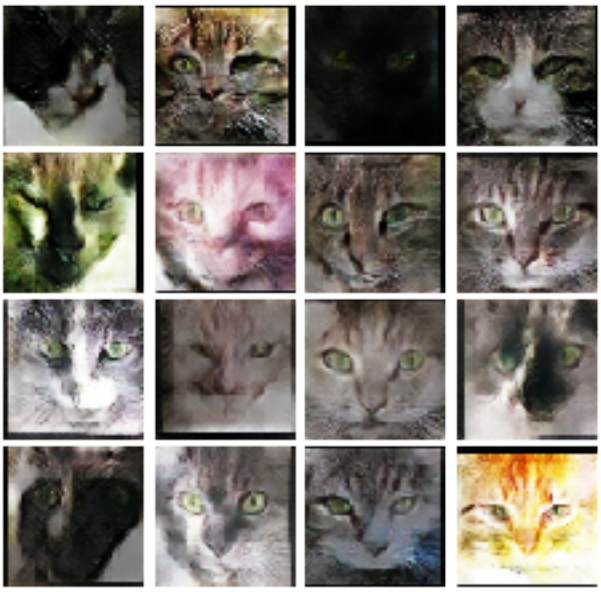
Iter: 6250, D: 0.4866, G:6.291

EPOCH: 14
Iter: 6500, D: 0.4211, G:2.707

Iter: 6750, D: 0.2686, G:3.078

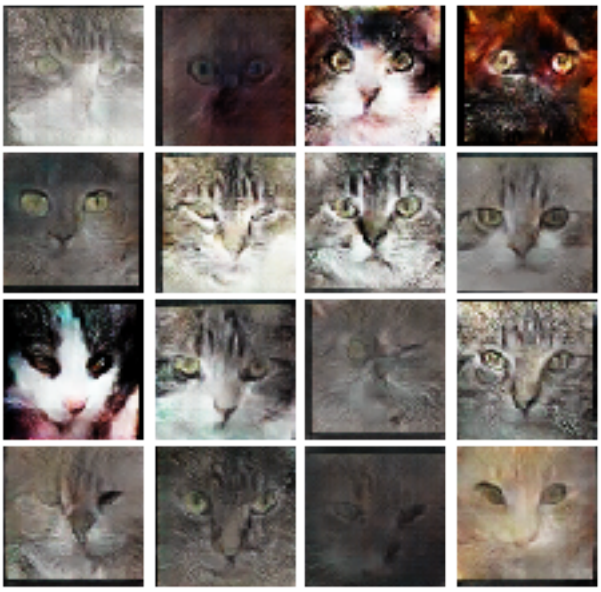
```
EPOCH:  15
Iter: 7000, D: 0.3435, G:2.27
```

```
Iter: 7250, D: 0.3964, G:1.348
```

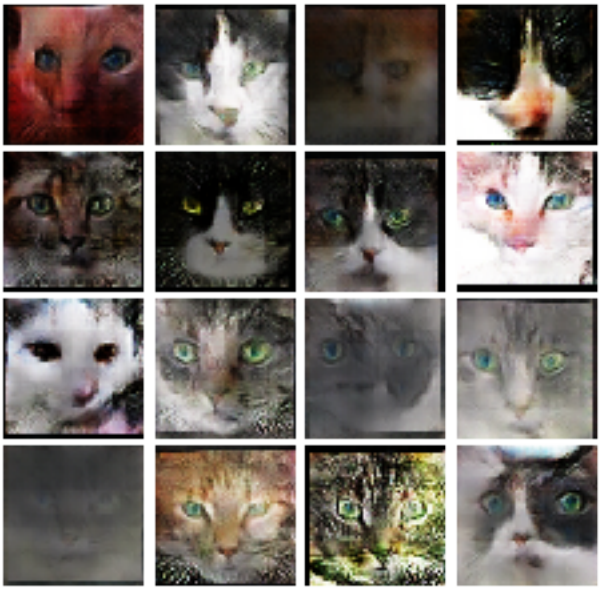


```
EPOCH:  16
Iter: 7500, D: 0.6159, G:1.621
```

```
Iter: 7750, D: 0.3116, G:3.0
```



```
EPOCH:  17
Iter: 8000, D: 0.4479, G:2.638
```

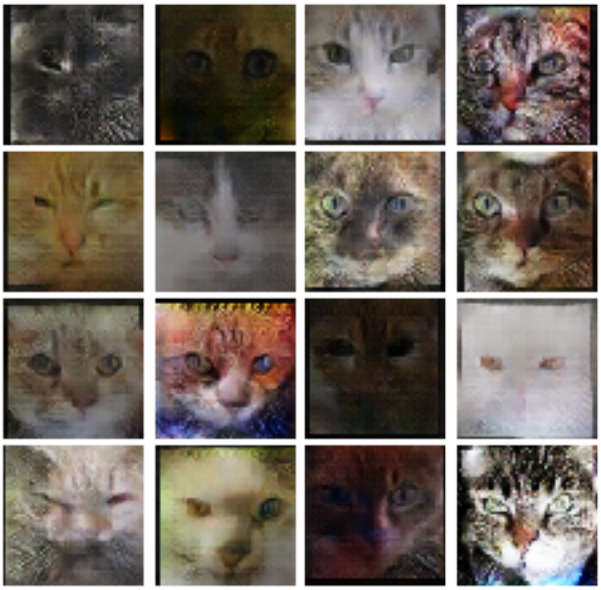


```
Iter: 8250, D: 0.5308, G:5.955
```

```
EPOCH:  18
Iter: 8500, D: 0.4354, G:5.261
```

```
Iter: 8750, D: 0.2245, G:3.253
```

```
EPOCH:  19
Iter: 9000, D: 0.4626, G:5.944
```



```
Iter: 9250, D: 0.2718, G:3.077
```

```
EPOCH:  20
Iter: 9500, D: 0.5082, G:3.147
```

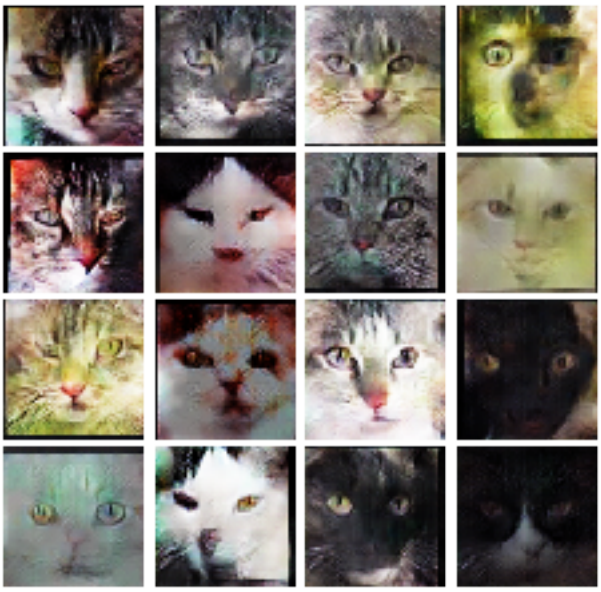
```
Iter: 9750, D: 0.3826, G:1.599
```

```
EPOCH:  21
Iter: 10000, D: 0.4216, G:3.268
```

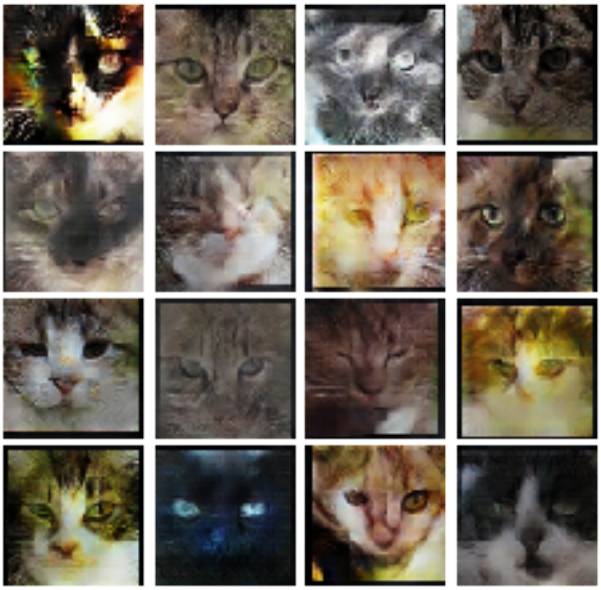
```
Iter: 10250, D: 0.1738, G:2.98
```

EPOCH: 22
Iter: 10500, D: 0.431, G:3.89

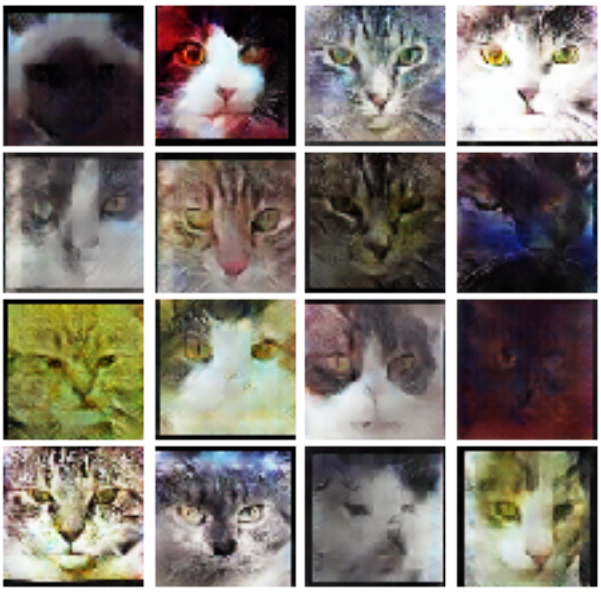
Iter: 10750, D: 0.375, G:2.247



```
EPOCH:  23
Iter: 11000, D: 0.4544, G:4.358
```



```
Iter: 11250, D: 0.1614, G:4.318
```

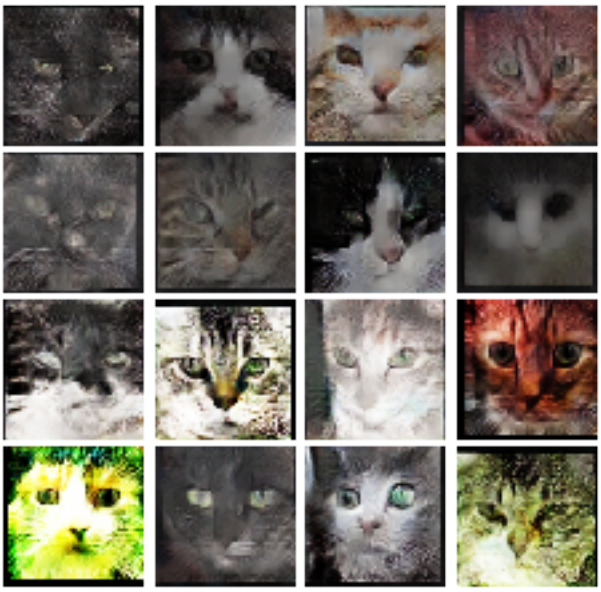


EPOCH: 24
Iter: 11500, D: 0.285, G:3.536

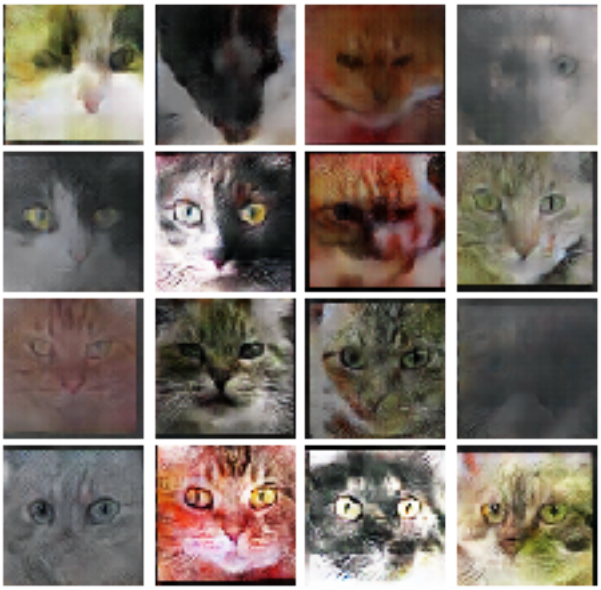
Iter: 11750, D: 0.2269, G:3.344

```
EPOCH:  25
Iter: 12000, D: 0.5639, G:1.606
```

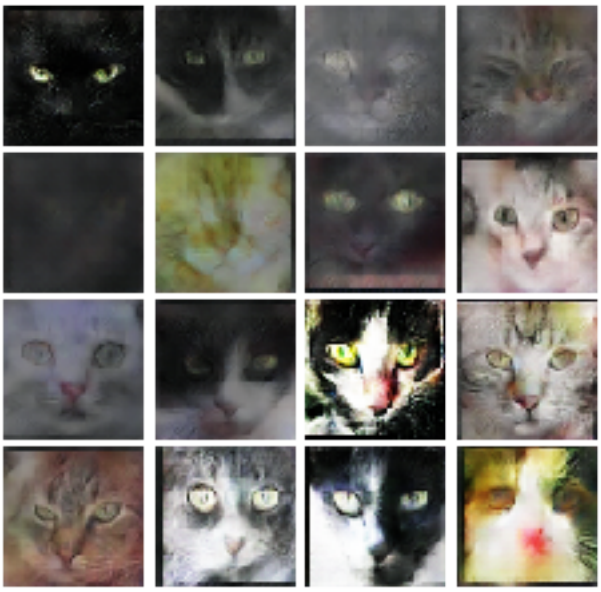
```
Iter: 12250, D: 0.4634, G:5.831
```



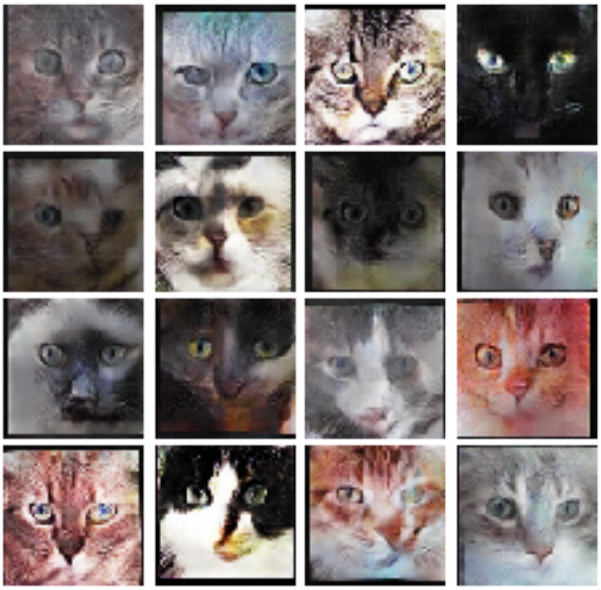
EPOCH: 26
Iter: 12500, D: 0.3039, G:2.5



Iter: 12750, D: 0.1328, G:5.758

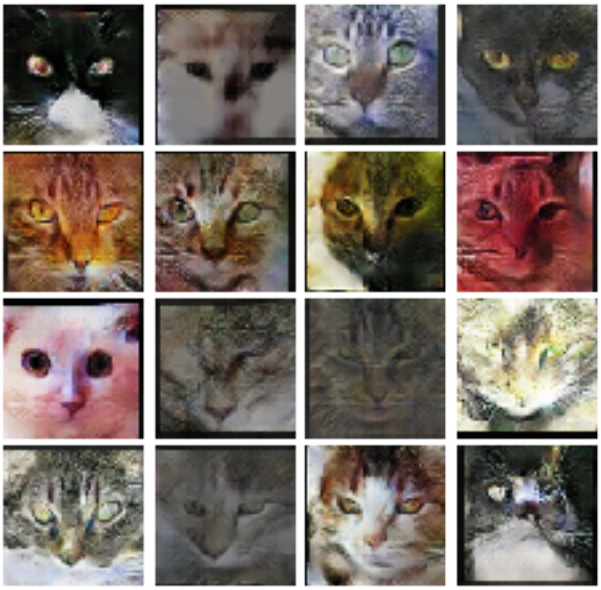


EPOCH: 27
Iter: 13000, D: 0.2435, G:2.621



Iter: 13250, D: 0.1644, G:2.596

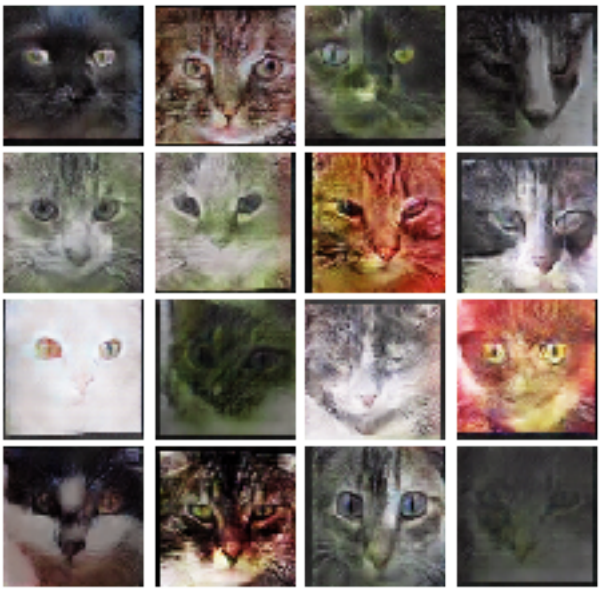
EPOCH: 28
Iter: 13500, D: 0.3818, G:1.693



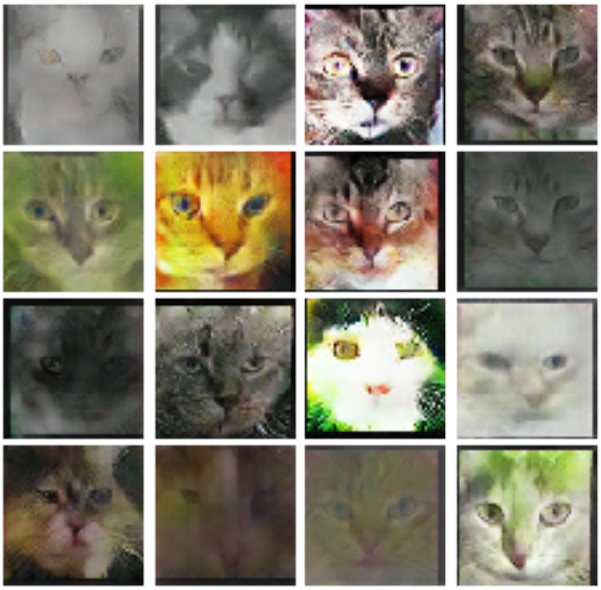
Iter: 13750, D: 0.3615, G:2.64

```
EPOCH:  29
Iter: 14000, D: 0.5978, G:5.653
```

```
Iter: 14250, D: 0.1068, G:4.929
```



EPOCH: 30
Iter: 14500, D: 0.2628, G:2.943



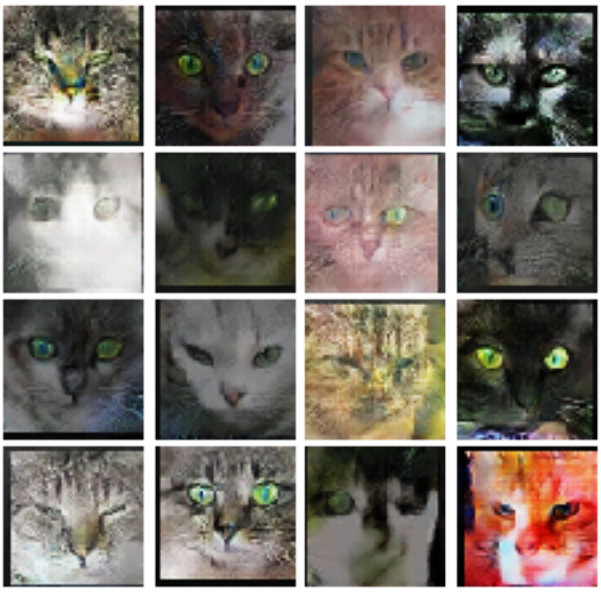
Iter: 14750, D: 0.2614, G:4.682

EPOCH: 31
Iter: 15000, D: 0.253, G:4.752

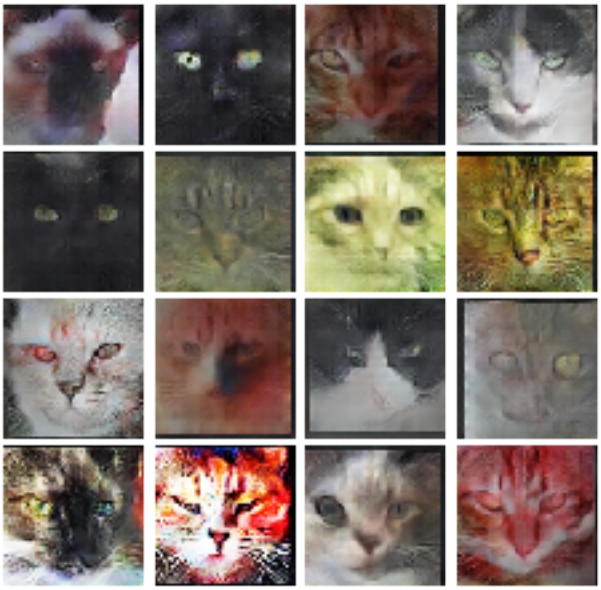
Iter: 15250, D: 0.1241, G:3.727

EPOCH: 32
Iter: 15500, D: 0.2055, G:5.874

EPOCH: 33
Iter: 15750, D: 0.1236, G:2.418



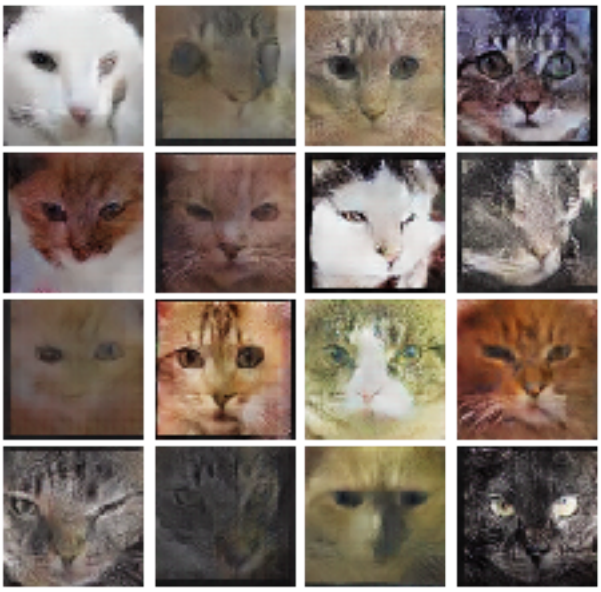
Iter: 16000, D: 0.142, G:3.573



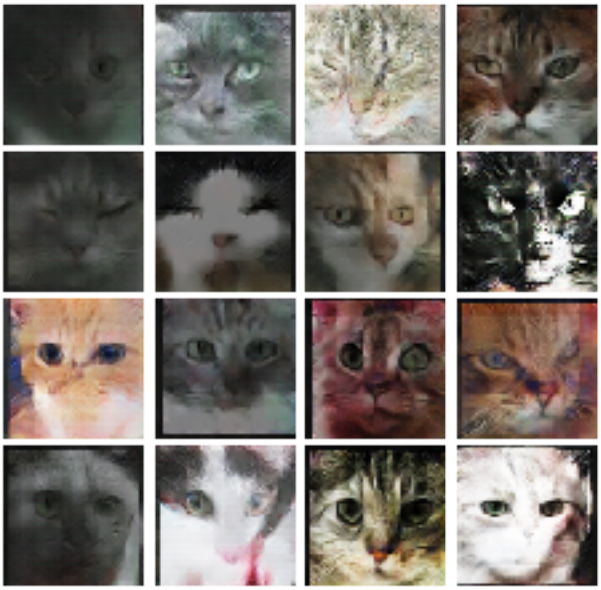
EPOCH: 34
Iter: 16250, D: 0.1861, G:2.288

Iter: 16500, D: 0.3603, G:3.165

EPOCH: 35
Iter: 16750, D: 0.165, G:3.403

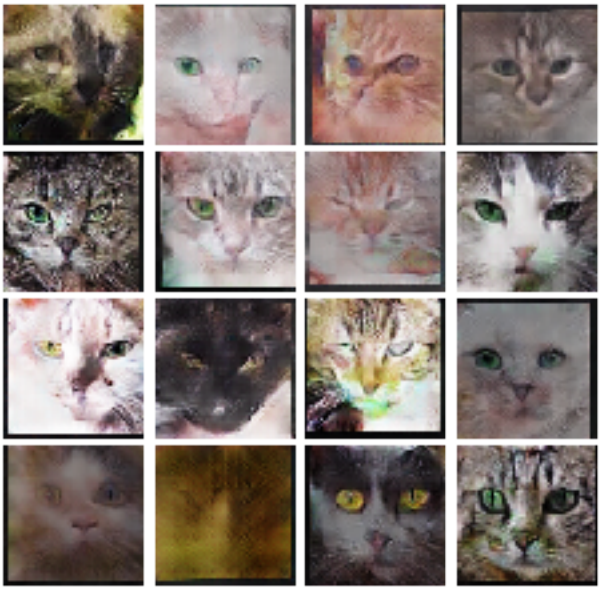


Iter: 17000, D: 0.3331, G:5.053



EPOCH: 36
Iter: 17250, D: 0.1967, G:4.02

Iter: 17500, D: 0.2666, G:3.826

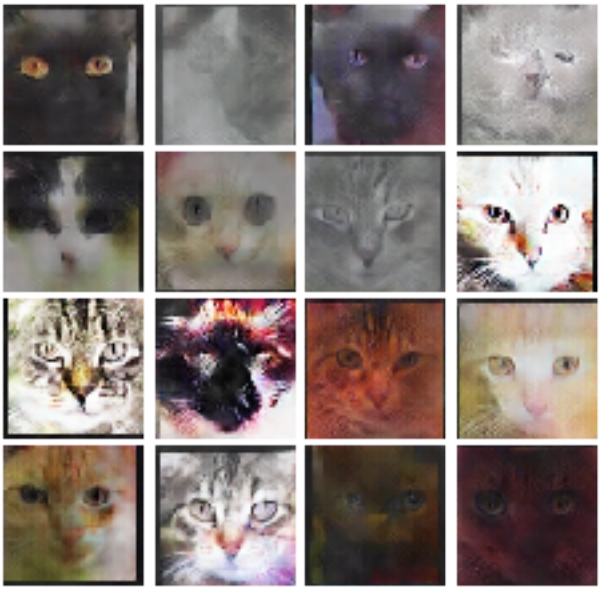


EPOCH: 37
Iter: 17750, D: 0.1198, G:5.174

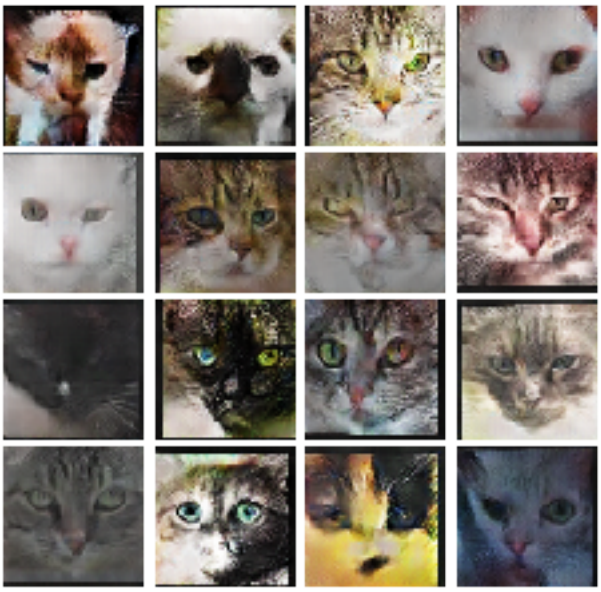
Iter: 18000, D: 0.0598, G:3.565

EPOCH: 38
Iter: 18250, D: 0.903, G:0.6797

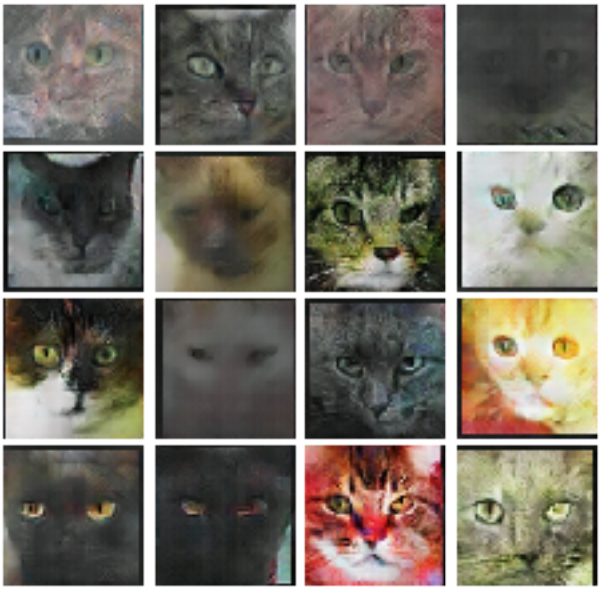
Iter: 18500, D: 0.2649, G:5.478



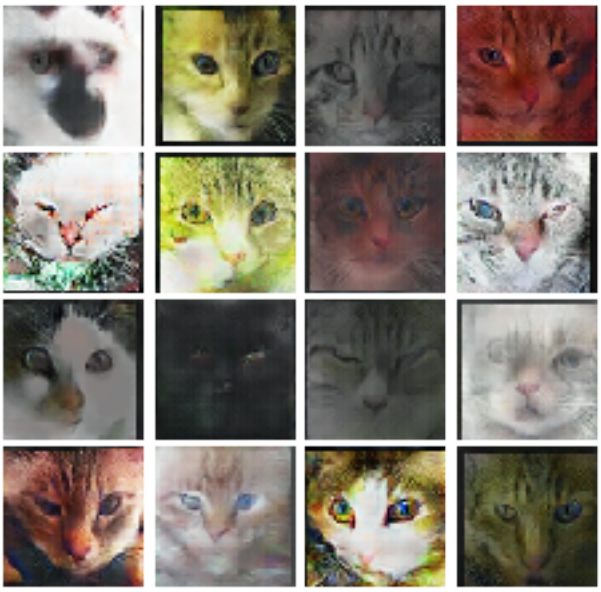
EPOCH: 39
Iter: 18750, D: 0.2381, G:4.314



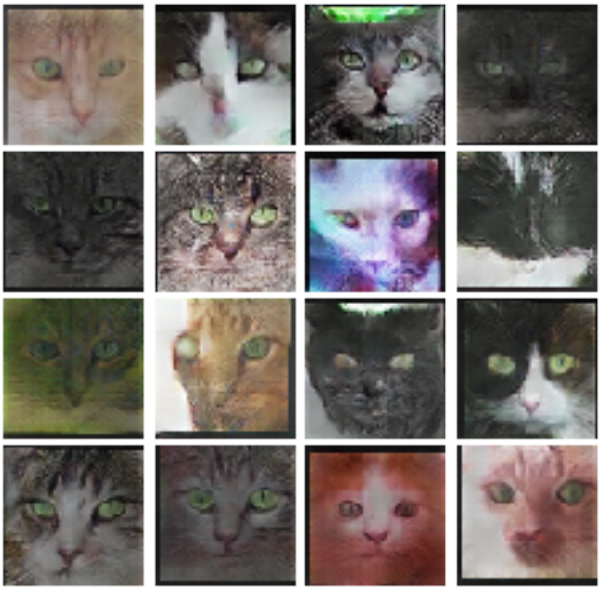
Iter: 19000, D: 0.09481, G:4.7



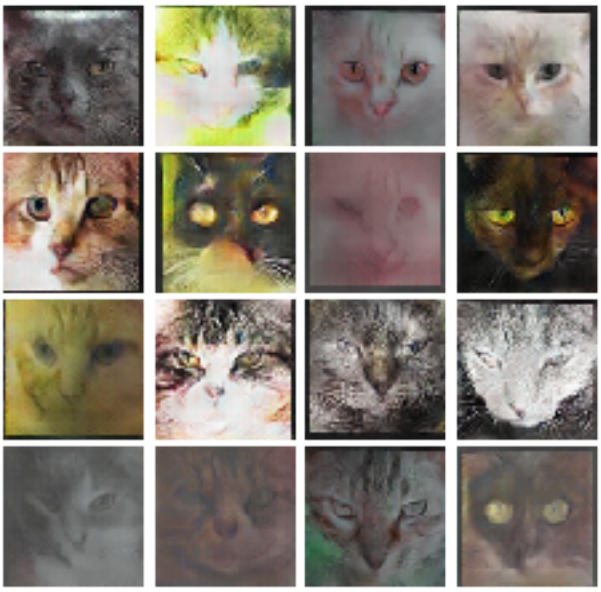
EPOCH: 40
Iter: 19250, D: 0.1536, G:5.823



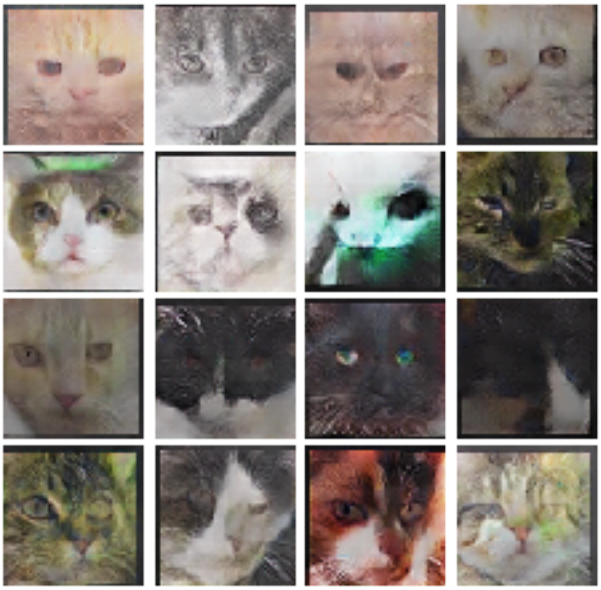
Iter: 19500, D: 0.5425, G:4.935



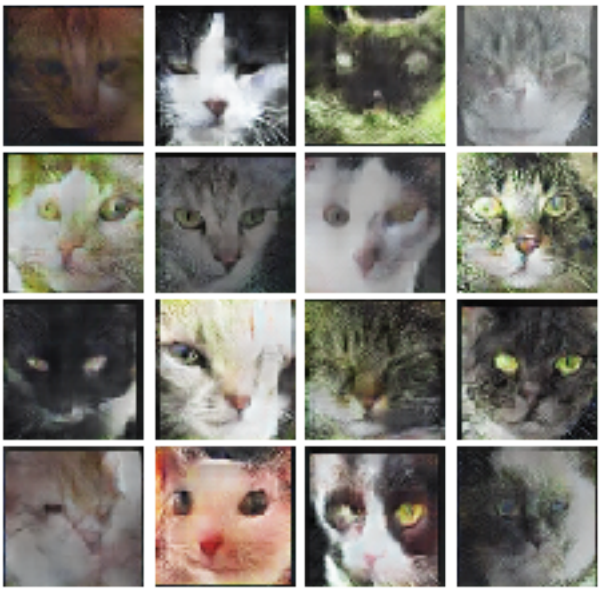
EPOCH: 41
Iter: 19750, D: 0.259, G:4.511



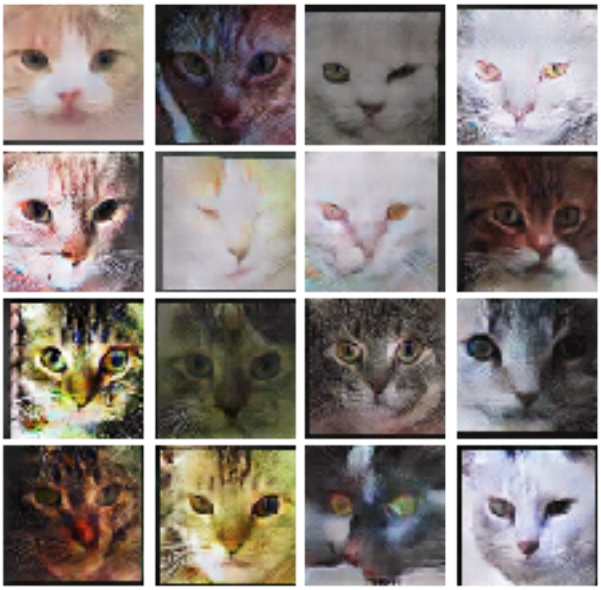
```
Iter: 20000, D: 0.2256, G:2.053
```



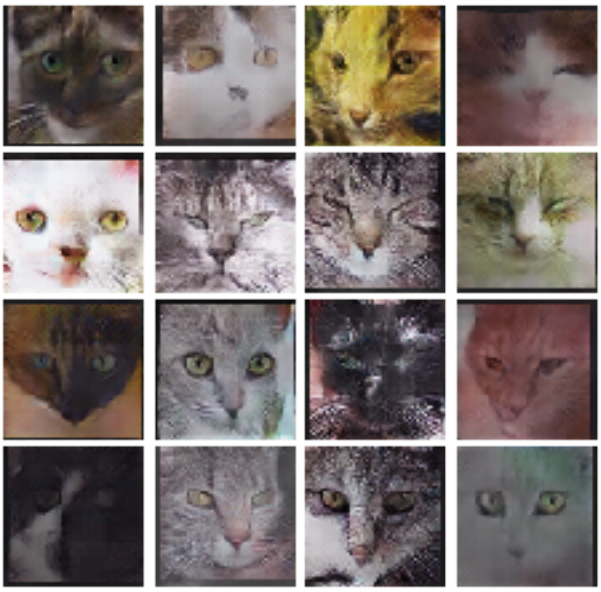
```
EPOCH:  42
Iter: 20250, D: 0.4427, G:6.017
```



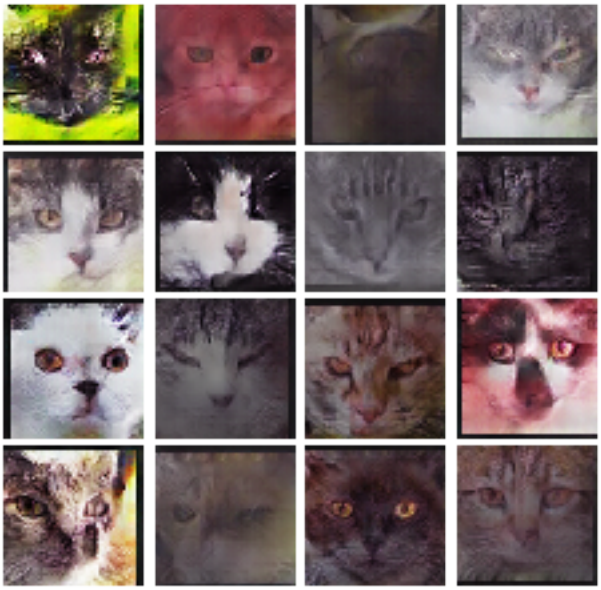
Iter: 20500, D: 0.2192, G:4.712



EPOCH: 43
Iter: 20750, D: 0.1773, G:4.62



Iter: 21000, D: 0.1502, G:3.659



EPOCH: 44
Iter: 21250, D: 0.439, G:1.544

Iter: 21500, D: 0.736, G:7.136

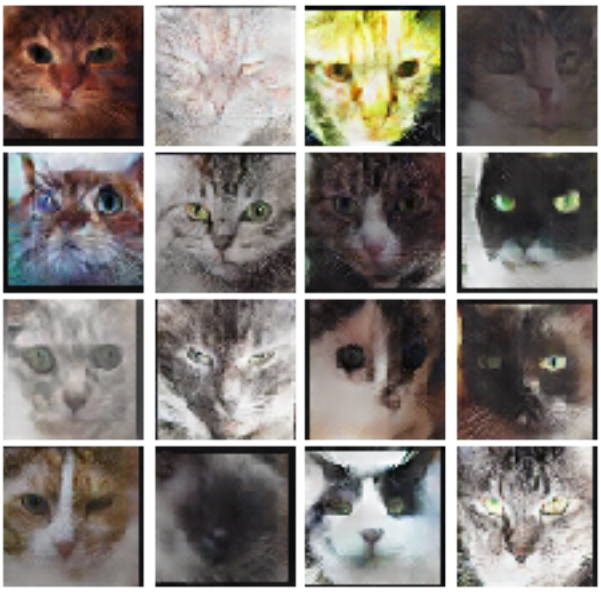
EPOCH: 45
Iter: 21750, D: 0.1022, G:5.818

Iter: 22000, D: 0.1786, G:6.652

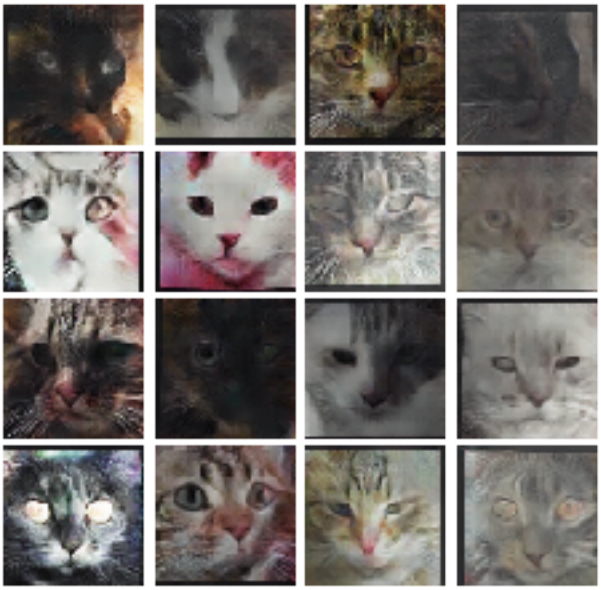
EPOCH: 46
Iter: 22250, D: 0.297, G:5.537

Iter: 22500, D: 0.2151, G:3.693

EPOCH: 47
Iter: 22750, D: 0.4593, G:7.484

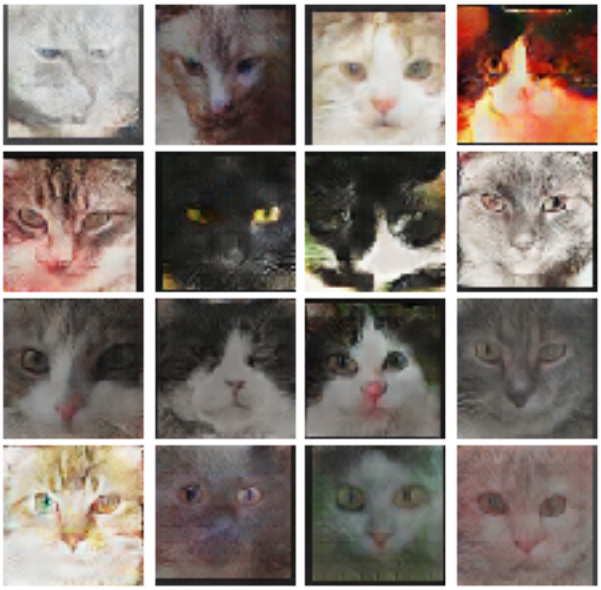


Iter: 23000, D: 0.1129, G:4.593

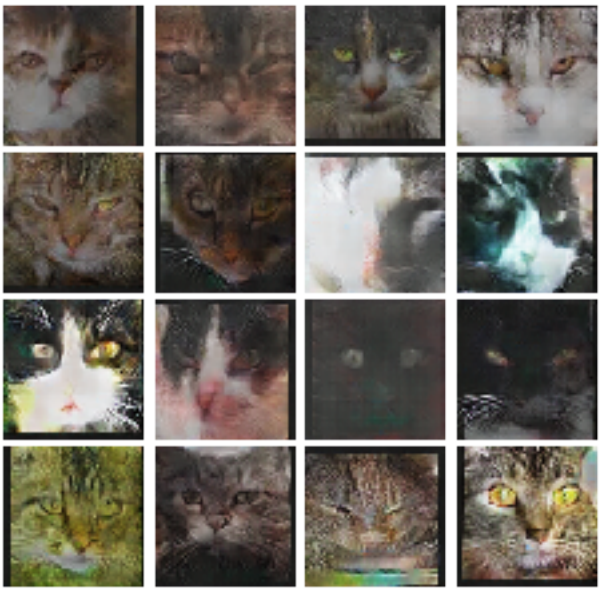


EPOCH: 48
Iter: 23250, D: 0.1097, G:4.201

Iter: 23500, D: 0.0601, G:4.423



EPOCH: 49
Iter: 23750, D: 0.2541, G:2.944



Iter: 24000, D: 0.209, G:4.51

EPOCH: 50
Iter: 24250, D: 0.1488, G:4.993

Iter: 24500, D: 0.344, G:5.752

5.0.2 Train LS-GAN

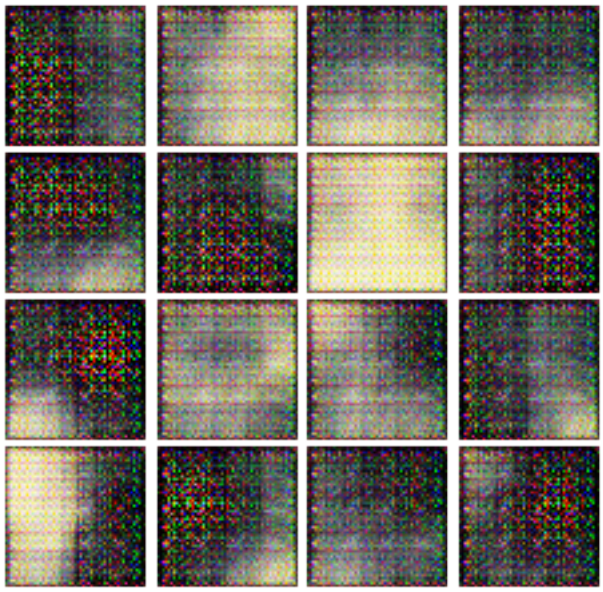
[147]:
```
D = Discriminator().to(device)
G = Generator(noise_dim=NOISE_DIM).to(device)
```

[148]:
```
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))
```

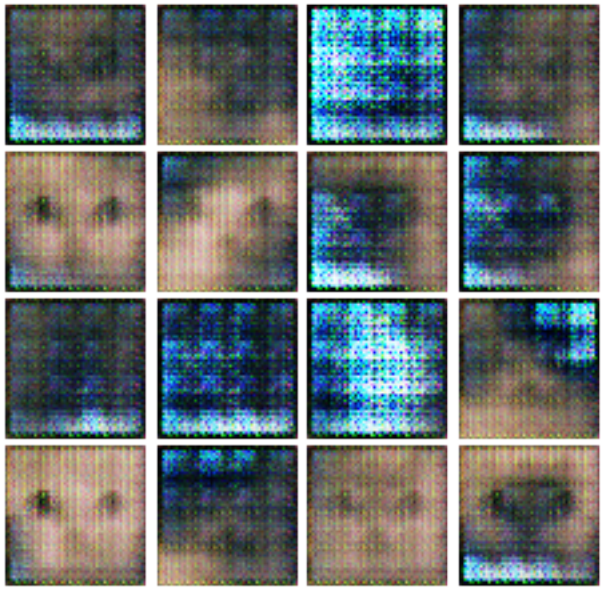
[149]:
```
# ls-gan
train(D, G, D_optimizer, G_optimizer, ls_discriminator_loss,
          ls_generator_loss, num_epochs=NUM_EPOCHS, show_every=250,
          batch_size=batch_size, train_loader=cat_loader_train, device=device)
```

```
EPOCH:  1
Iter: 0, D: 0.5372, G:24.74
```

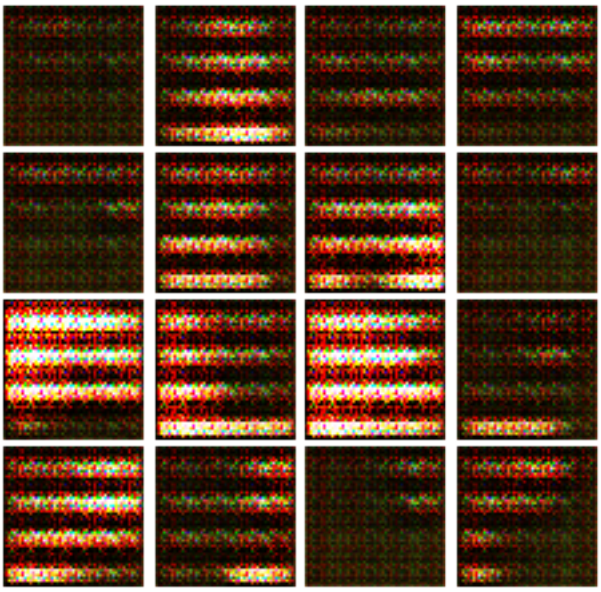
```
Iter: 250, D: 0.6101, G:0.03461
```



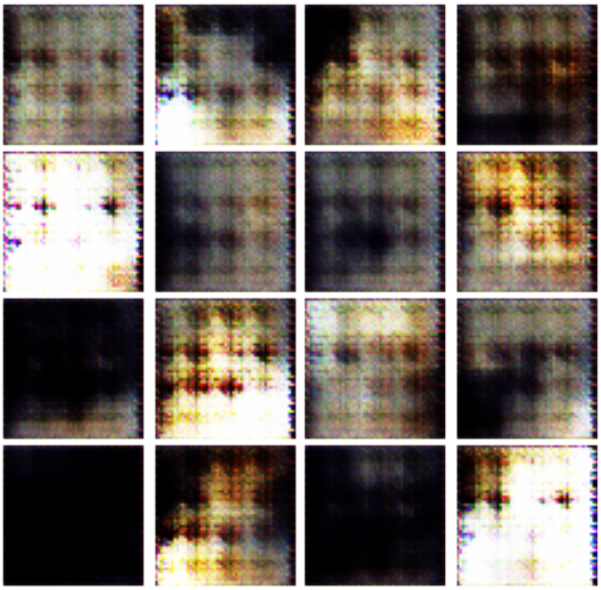
```
EPOCH:  2
Iter: 500, D: 0.3258, G:0.2167
```



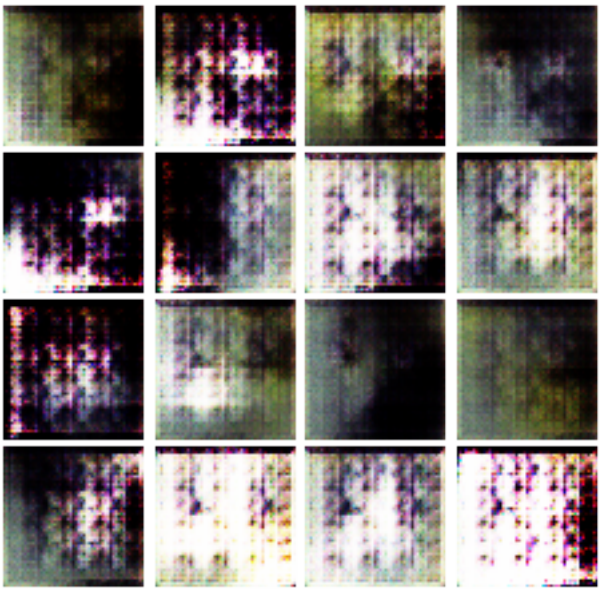
```
Iter: 750, D: 2.839, G:2.481
```



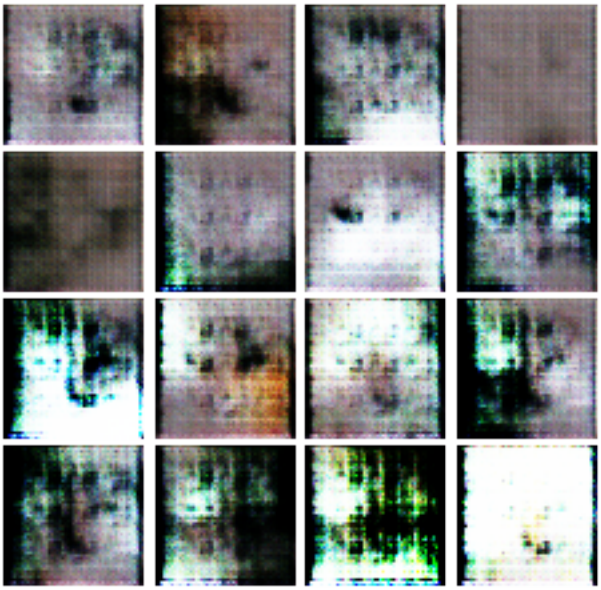
EPOCH: 3
Iter: 1000, D: 0.3082, G:0.8536



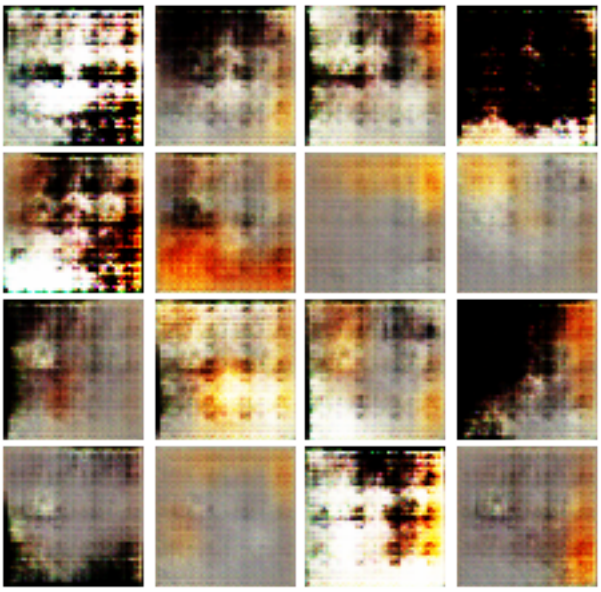
Iter: 1250, D: 0.286, G:0.8467



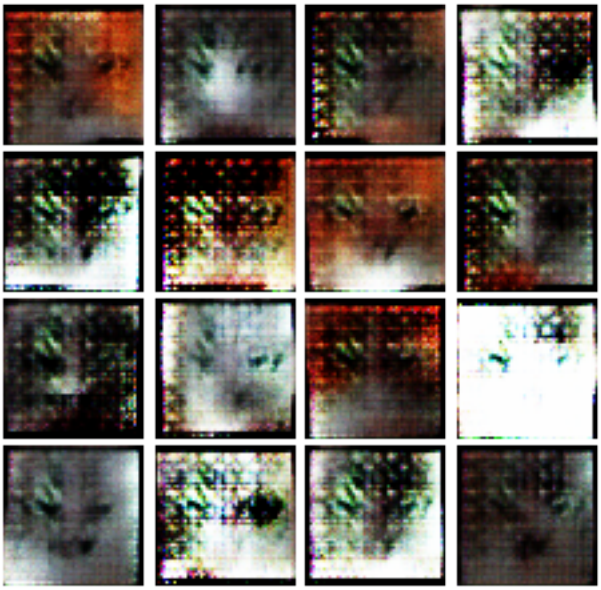
EPOCH: 4
Iter: 1500, D: 0.2049, G:0.1991



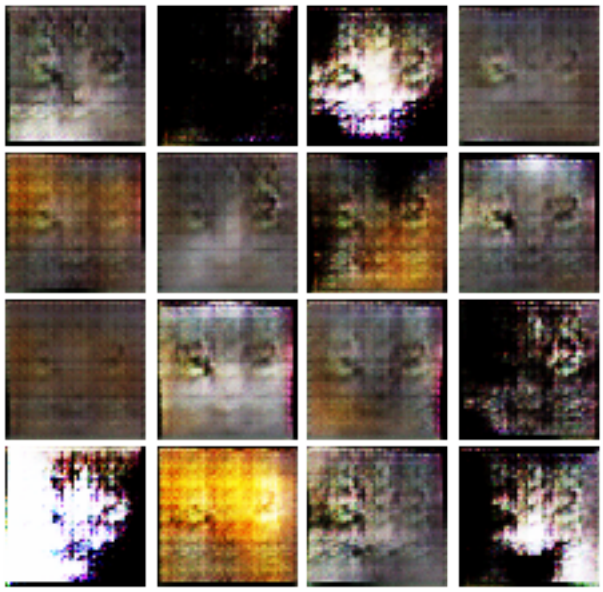
Iter: 1750, D: 0.2406, G:0.2438



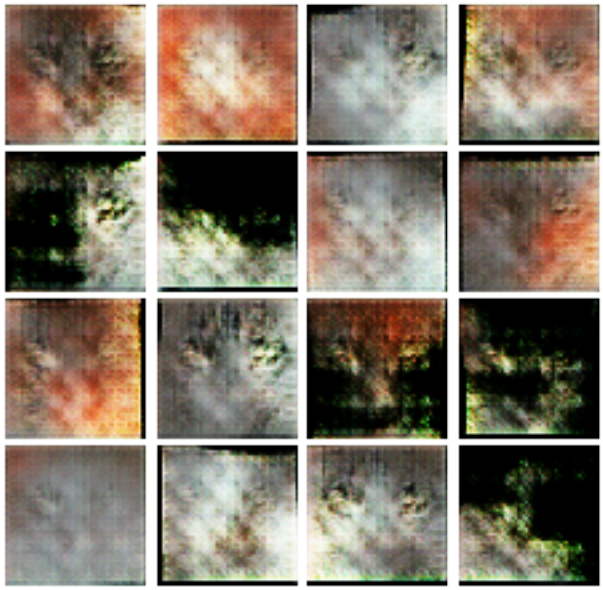
EPOCH: 5
Iter: 2000, D: 0.2837, G:0.8524



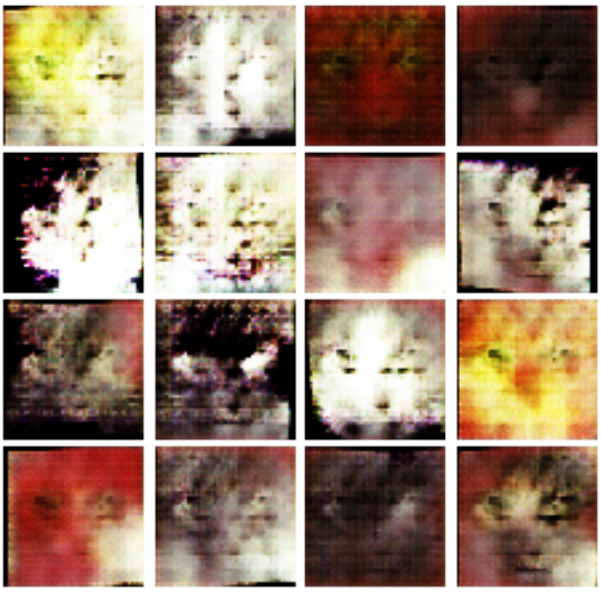
Iter: 2250, D: 0.1955, G:0.2059



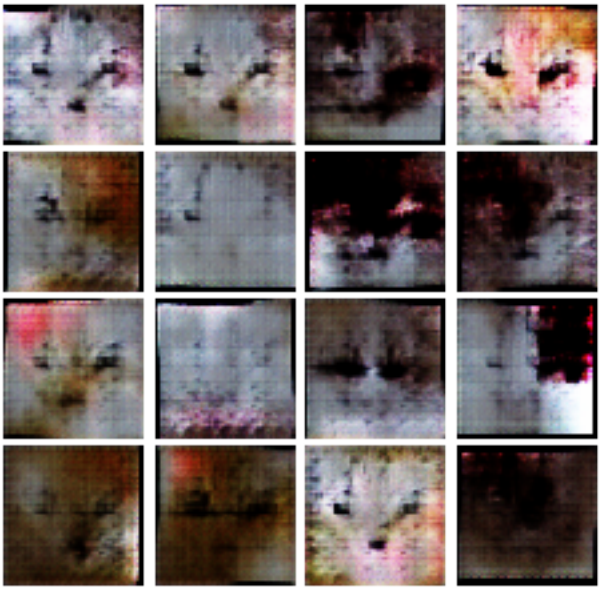
EPOCH: 6
Iter: 2500, D: 0.2724, G:0.5116



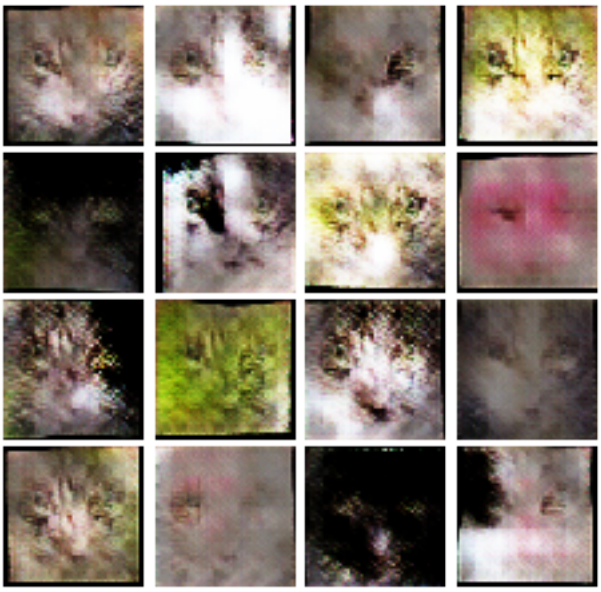
Iter: 2750, D: 0.1603, G:0.2243



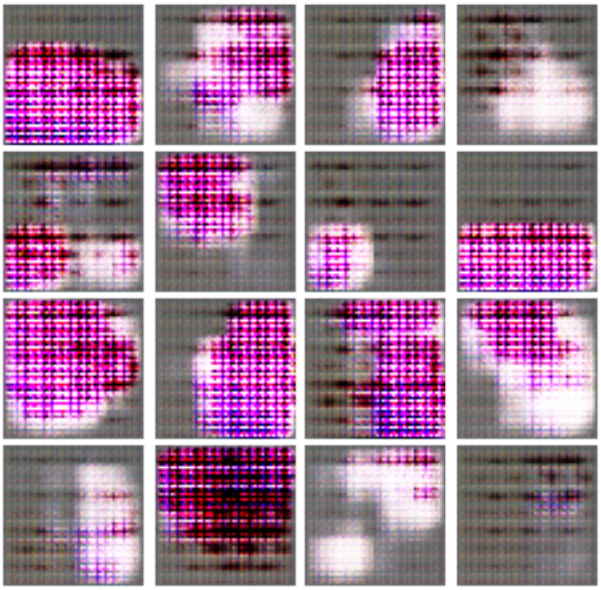
EPOCH: 7
Iter: 3000, D: 0.2336, G:0.5071



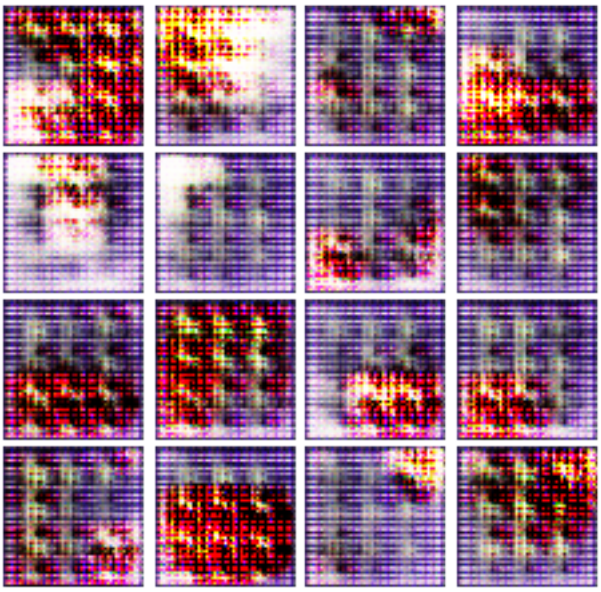
Iter: 3250, D: 0.2305, G:0.2478



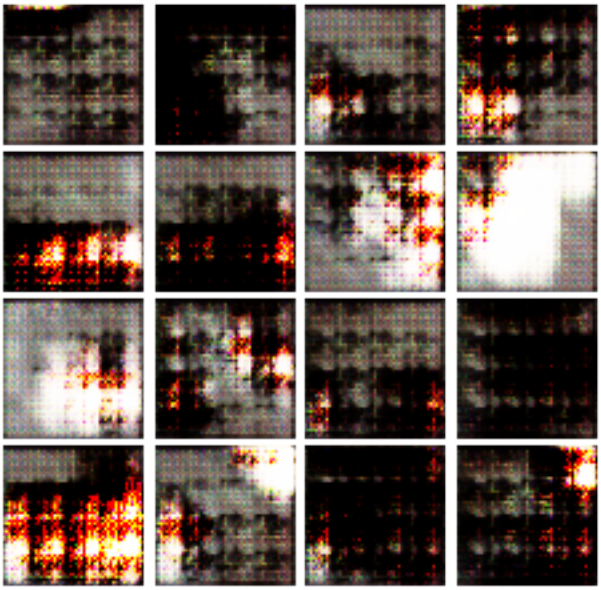
EPOCH: 8
Iter: 3500, D: 0.1575, G:0.1658



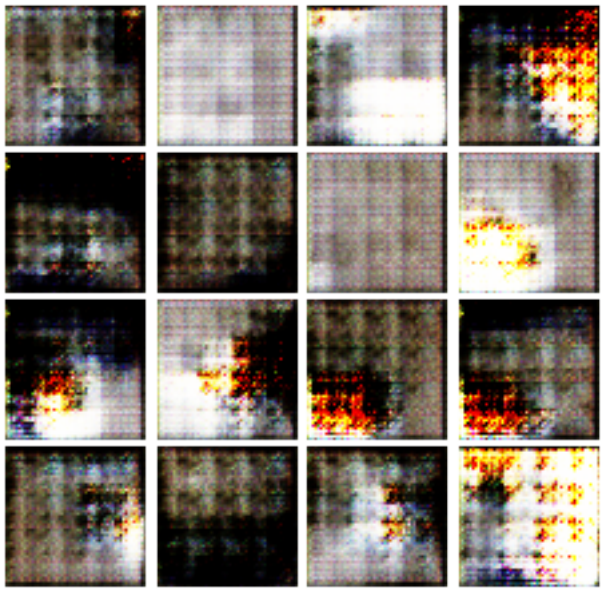
Iter: 3750, D: 0.06785, G:0.4133



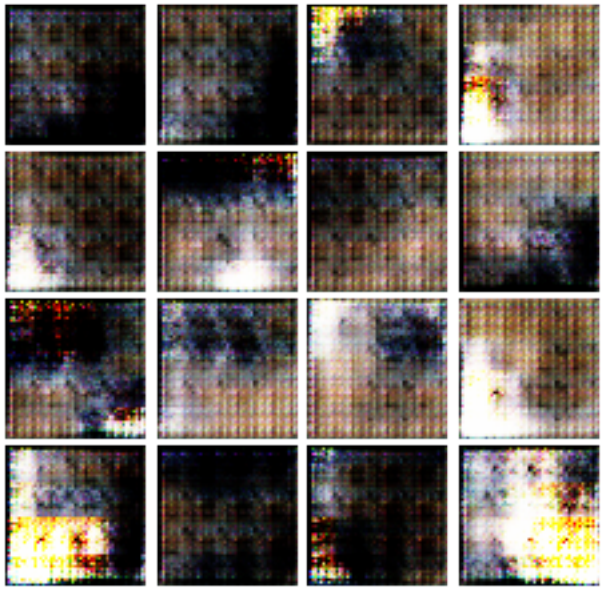
```
EPOCH:  9
Iter: 4000, D: 0.1166, G:0.2293
```



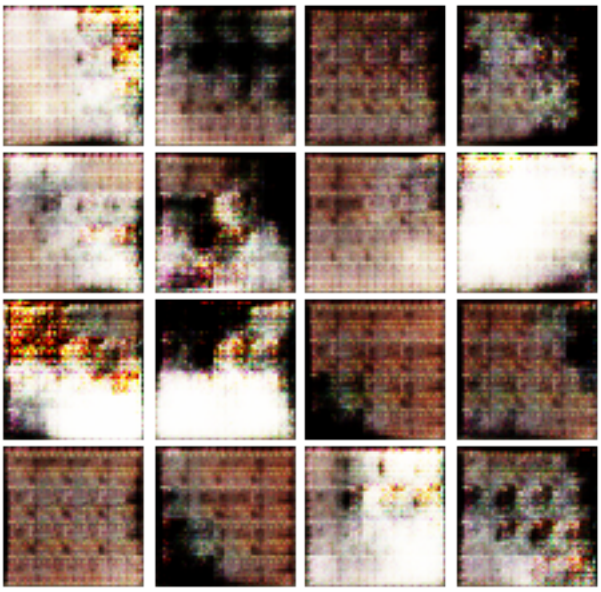
```
Iter: 4250, D: 0.2485, G:0.1928
```



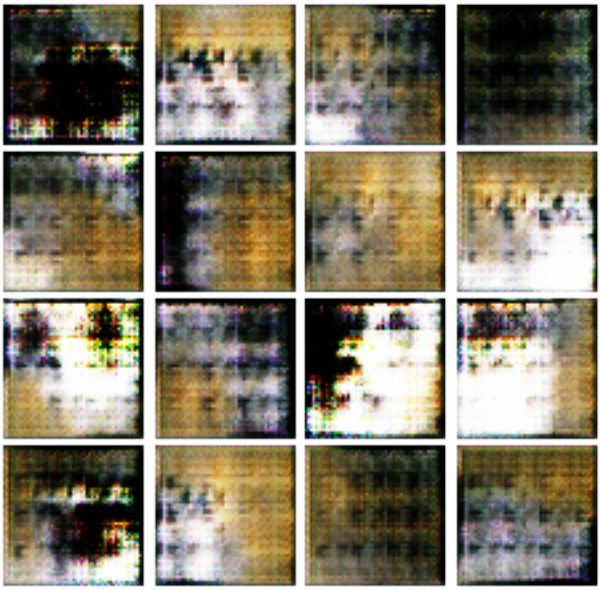
EPOCH: 10
Iter: 4500, D: 0.2772, G:0.304



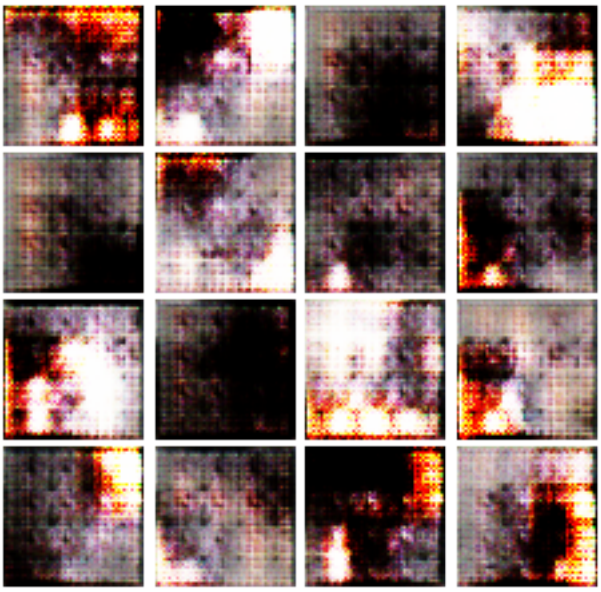
Iter: 4750, D: 0.6289, G:0.3724



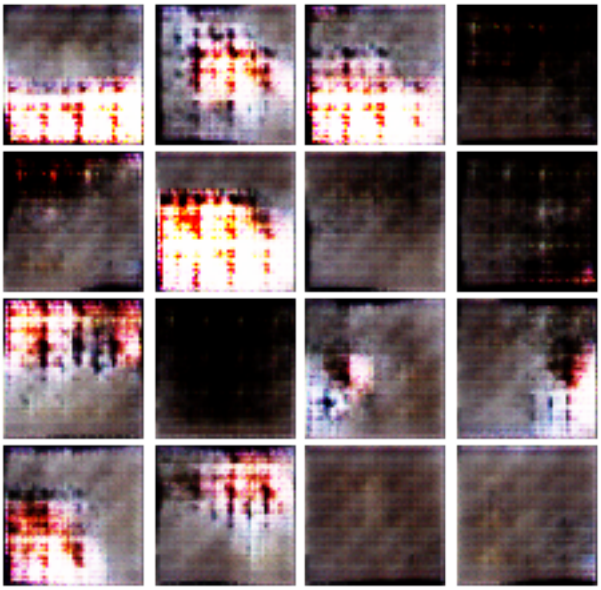
```
EPOCH:  11
Iter: 5000, D: 0.2753, G:0.1986
```



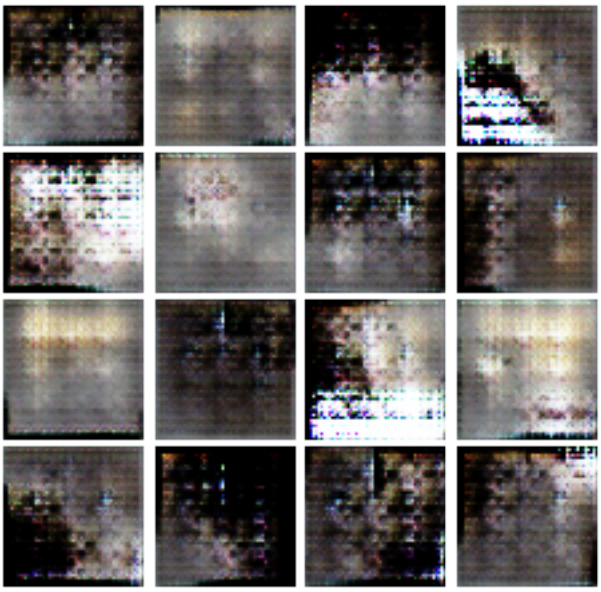
```
Iter: 5250, D: 0.3178, G:0.3021
```



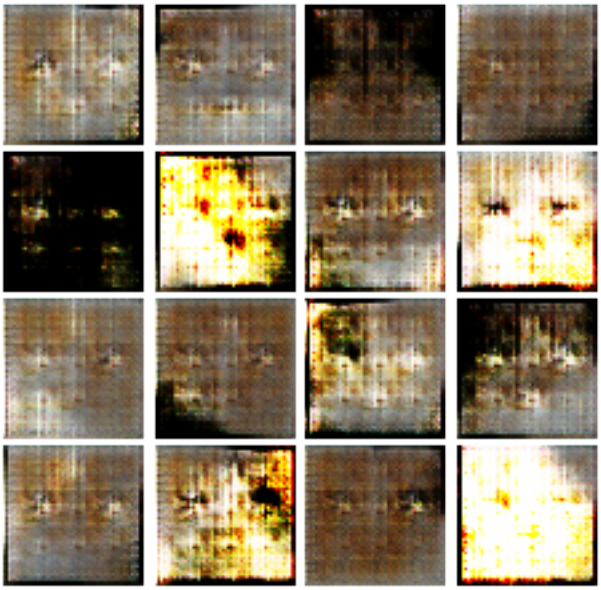
EPOCH: 12
Iter: 5500, D: 0.345, G:0.1693



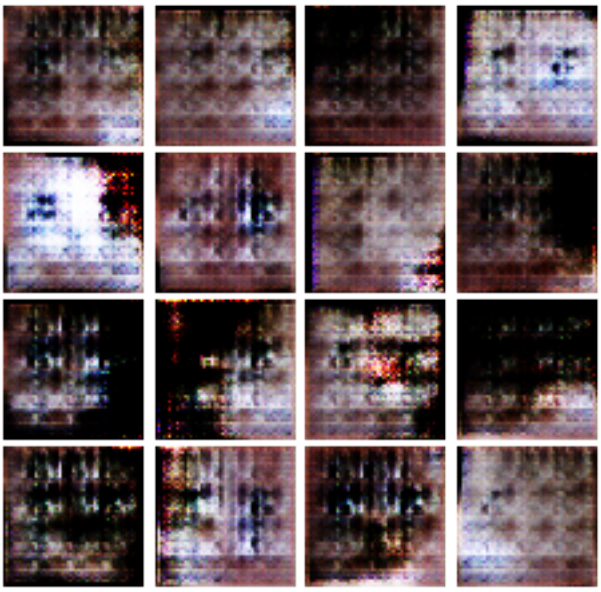
Iter: 5750, D: 0.1346, G:0.3482



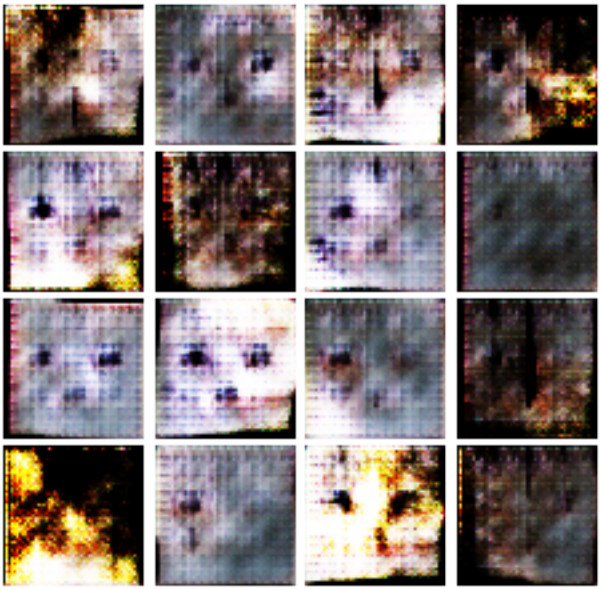
EPOCH: 13
Iter: 6000, D: 0.1465, G:0.3679



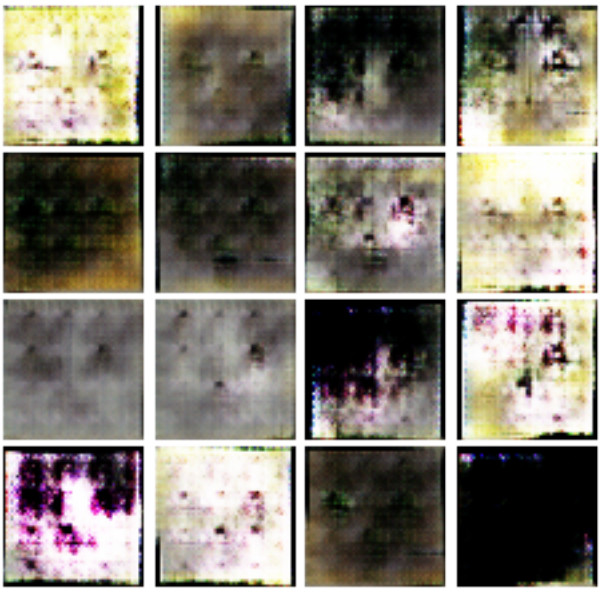
Iter: 6250, D: 1.244, G:1.153



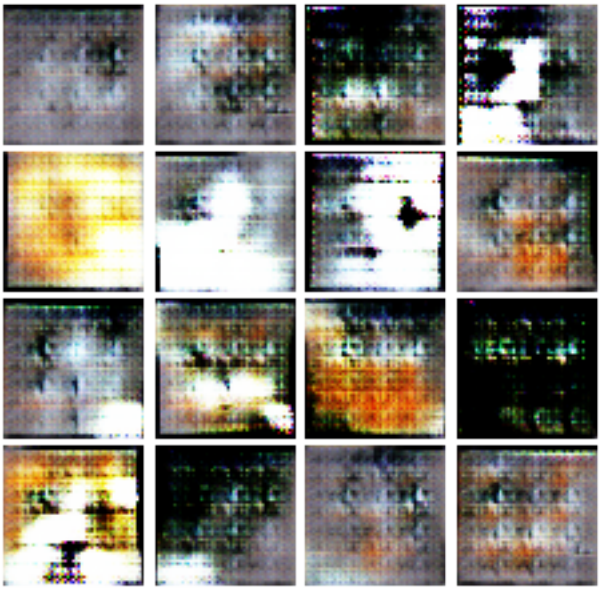
EPOCH: 14
Iter: 6500, D: 0.2524, G:0.6802



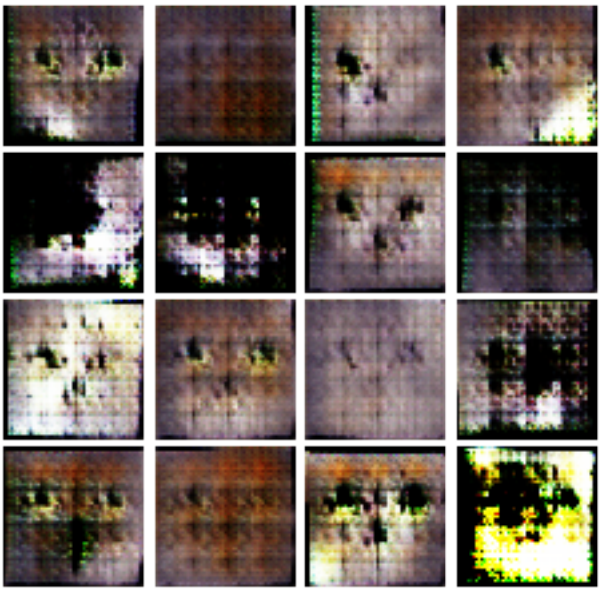
Iter: 6750, D: 0.229, G:0.2375



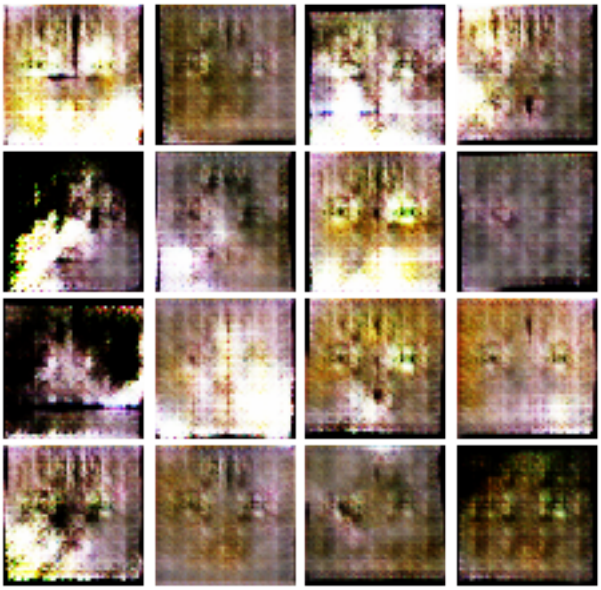
```
EPOCH:  15
Iter: 7000, D: 0.2712, G:0.2869
```



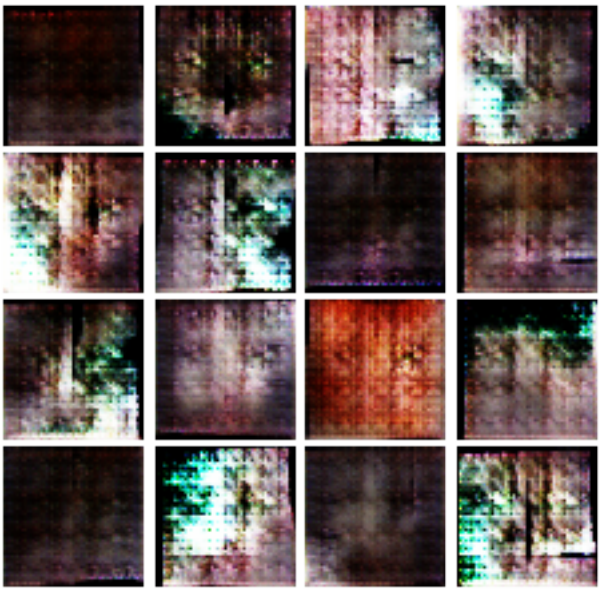
```
Iter: 7250, D: 0.3783, G:0.2255
```



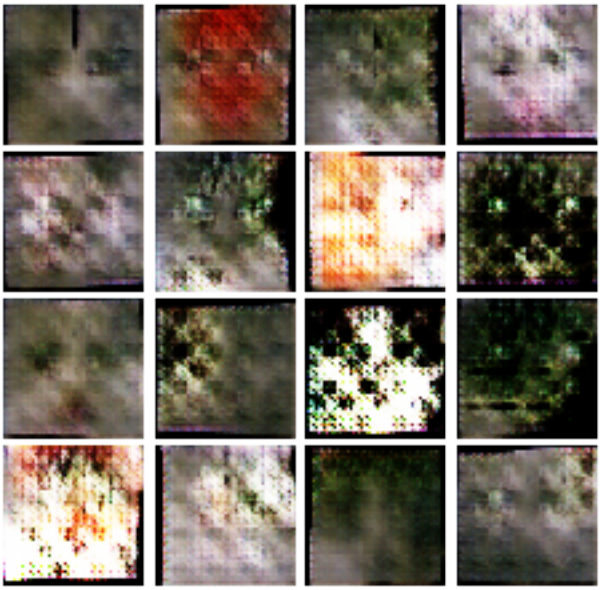
```
EPOCH:  16
Iter: 7500, D: 0.2836, G:0.2253
```



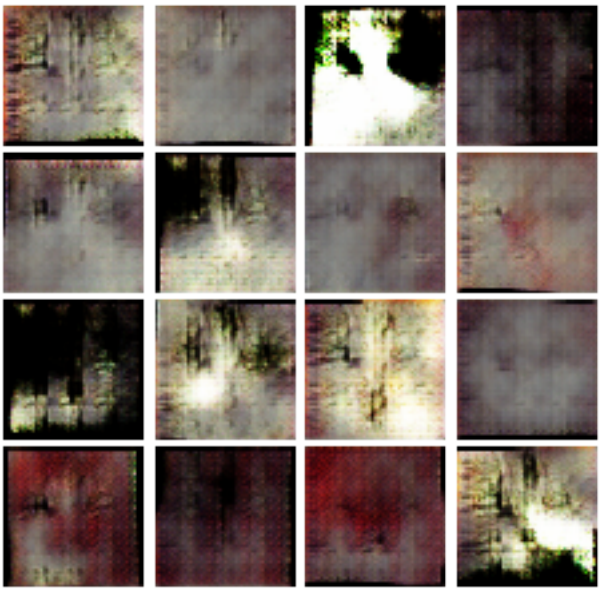
```
Iter: 7750, D: 0.204, G:0.2286
```



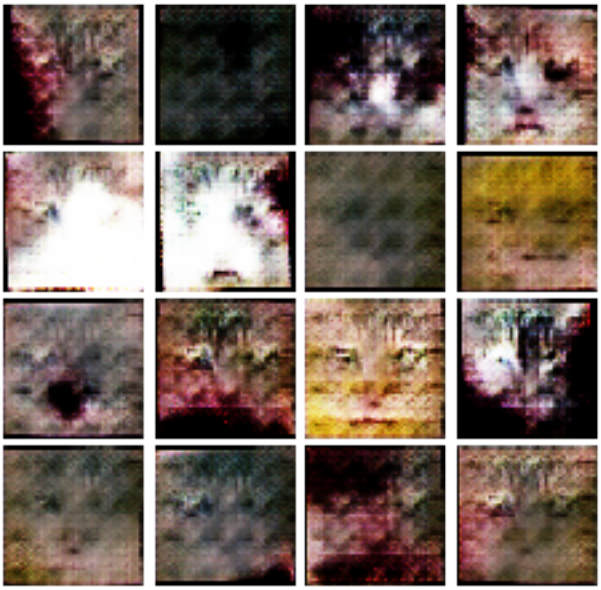
EPOCH: 17
Iter: 8000, D: 0.2062, G:0.3262



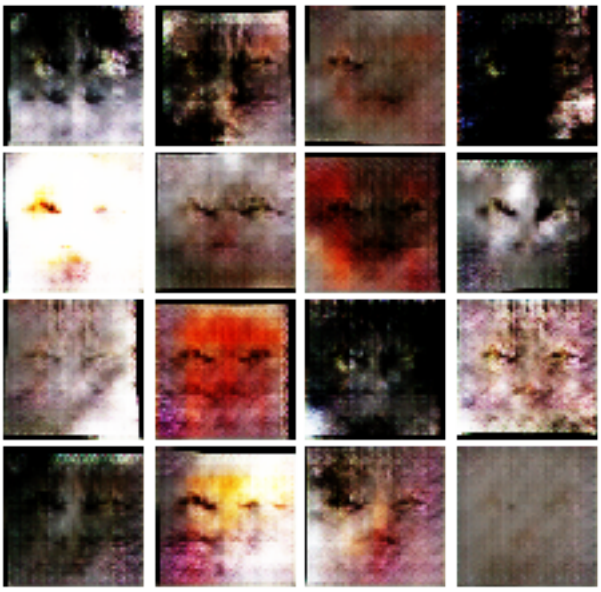
Iter: 8250, D: 0.2486, G:0.2358



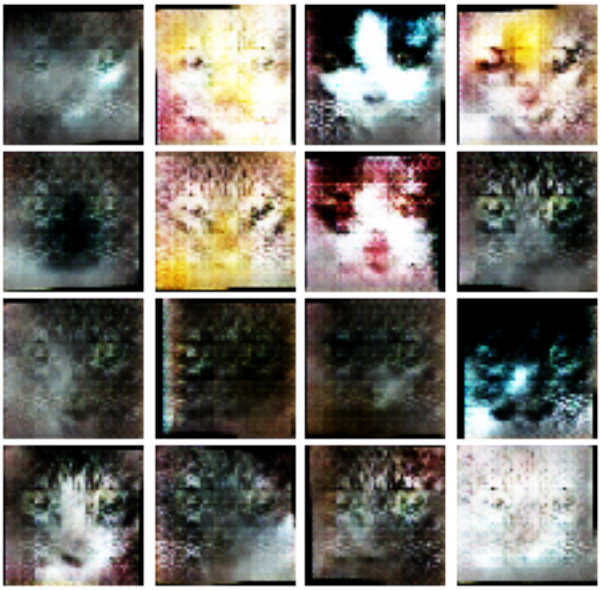
EPOCH: 18
Iter: 8500, D: 0.2311, G:0.2711



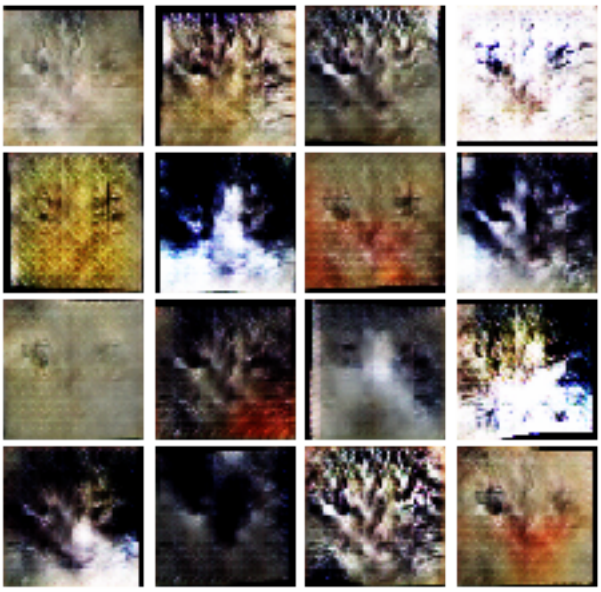
Iter: 8750, D: 0.2774, G:0.1286



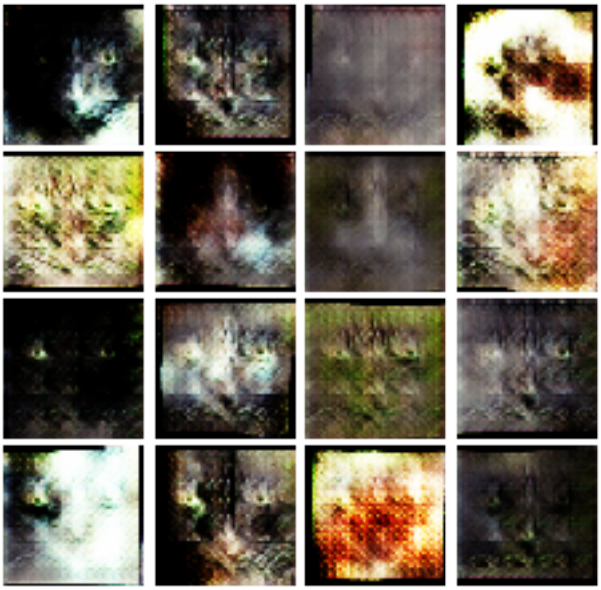
EPOCH: 19
Iter: 9000, D: 0.2266, G:0.2424



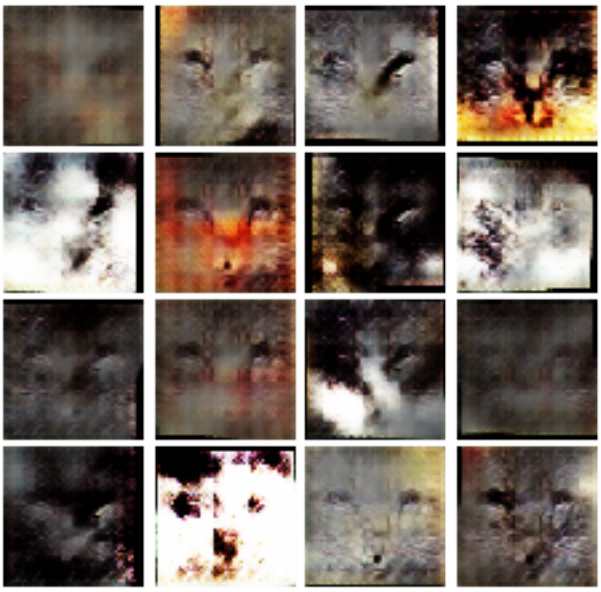
Iter: 9250, D: 0.2438, G:0.2233



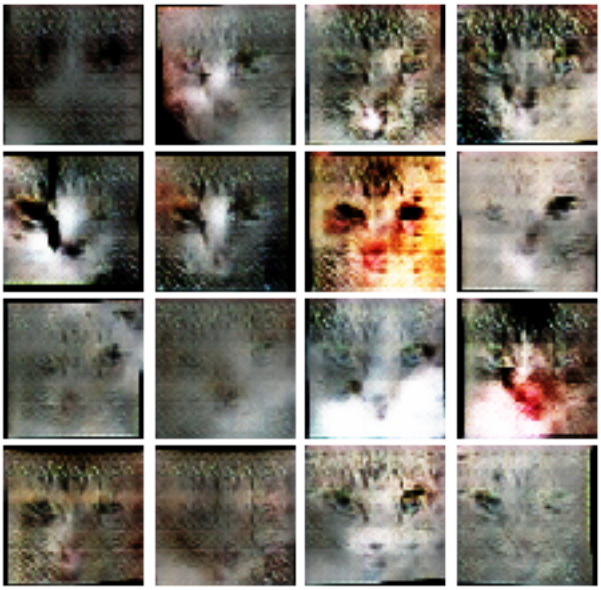
EPOCH: 20
Iter: 9500, D: 0.4825, G:0.2345



Iter: 9750, D: 0.3217, G:0.1281

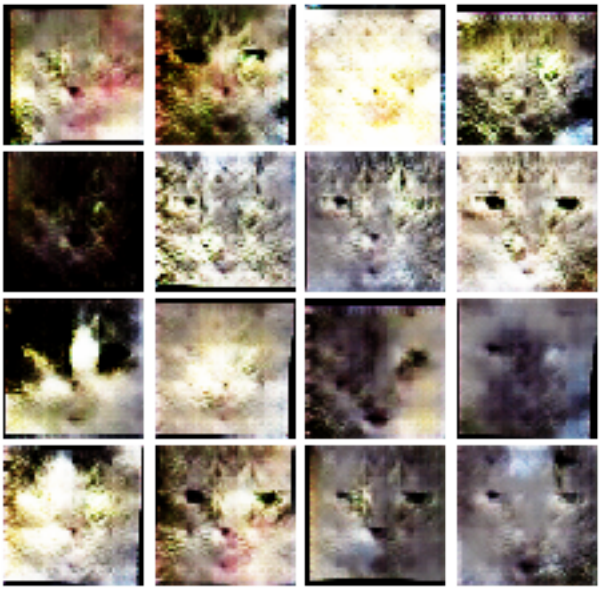


EPOCH: 21
Iter: 10000, D: 0.2514, G:0.208

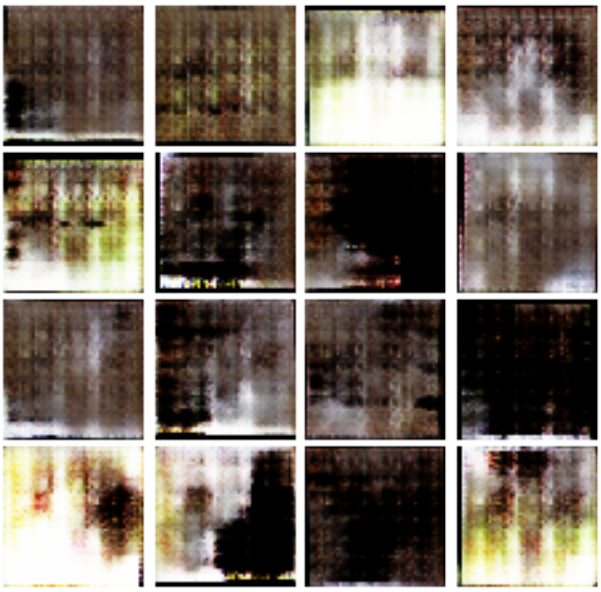


Iter: 10250, D: 0.3348, G:0.1403

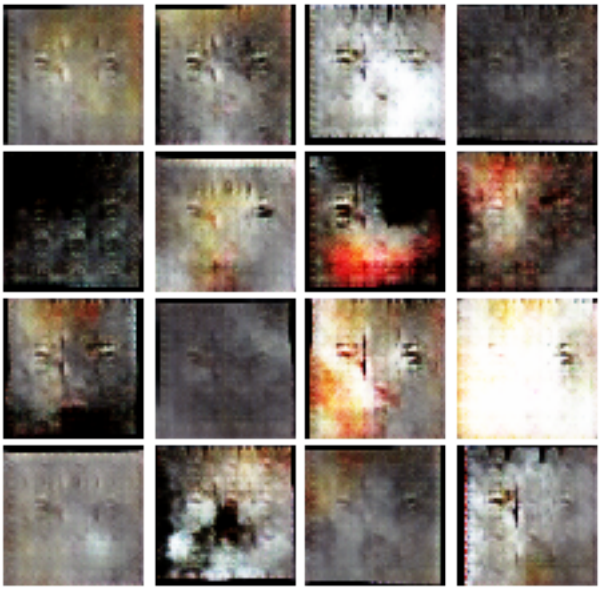
EPOCH: 22
Iter: 10500, D: 0.2386, G:0.3395



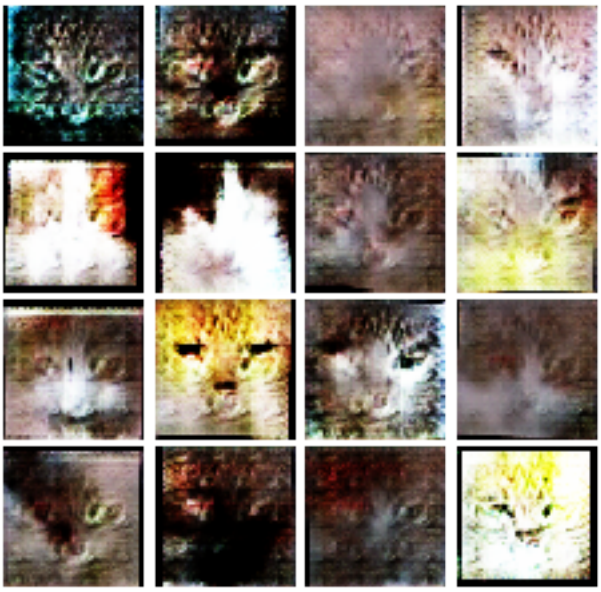
Iter: 10750, D: 0.2625, G:0.2963



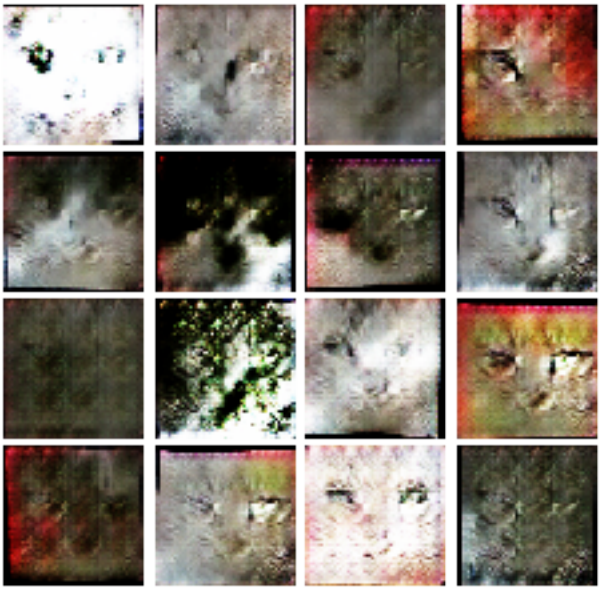
EPOCH: 23
Iter: 11000, D: 0.337, G:0.1213



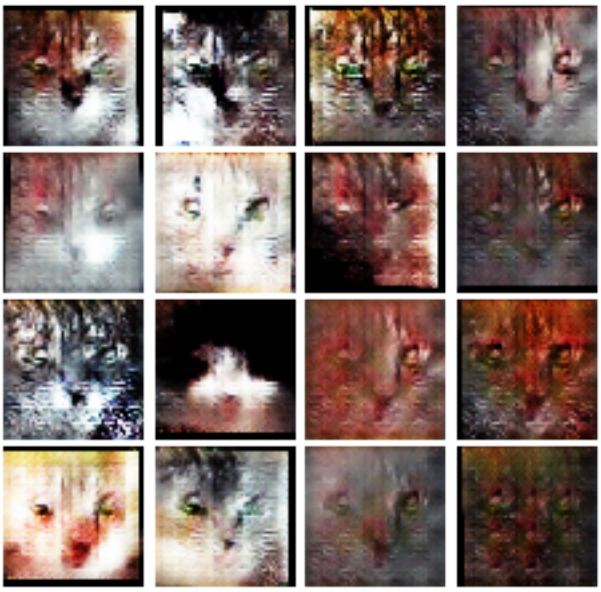
Iter: 11250, D: 0.2748, G:0.1902



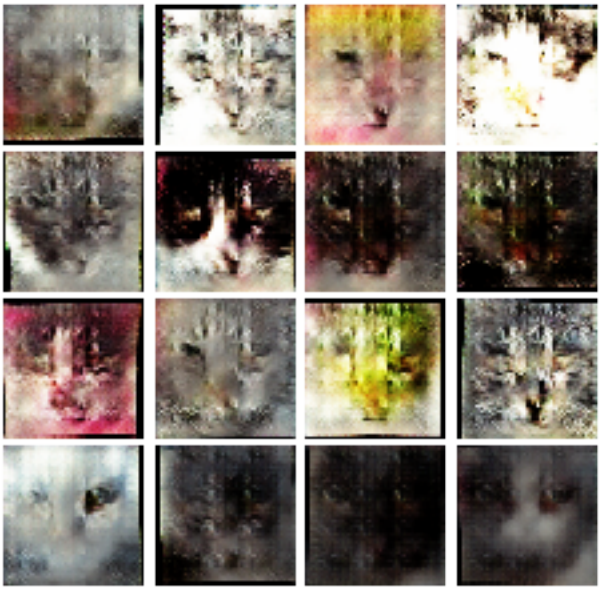
EPOCH: 24
Iter: 11500, D: 0.5811, G:0.6823



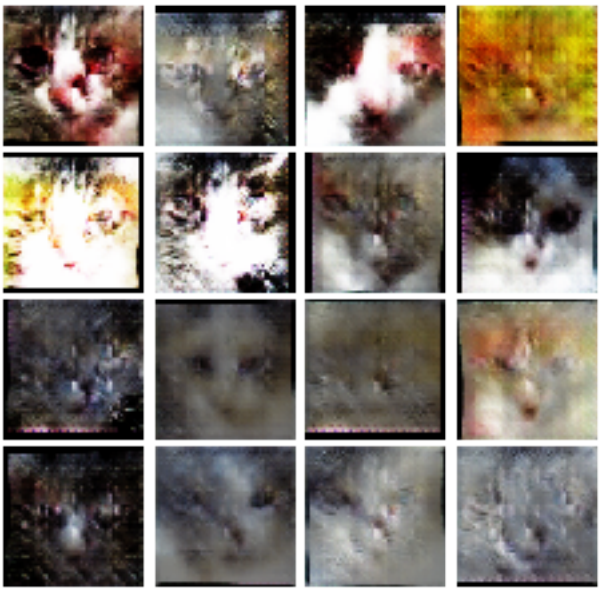
Iter: 11750, D: 0.2478, G:0.2435



EPOCH: 25
Iter: 12000, D: 0.2394, G:0.2512

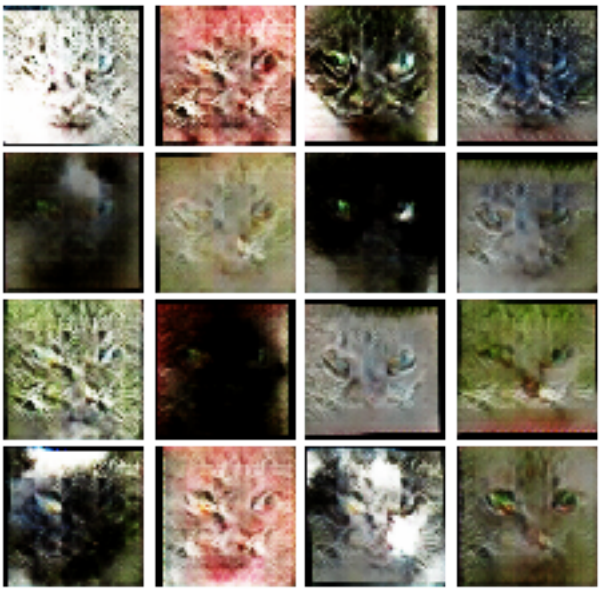


Iter: 12250, D: 0.8457, G:0.3271

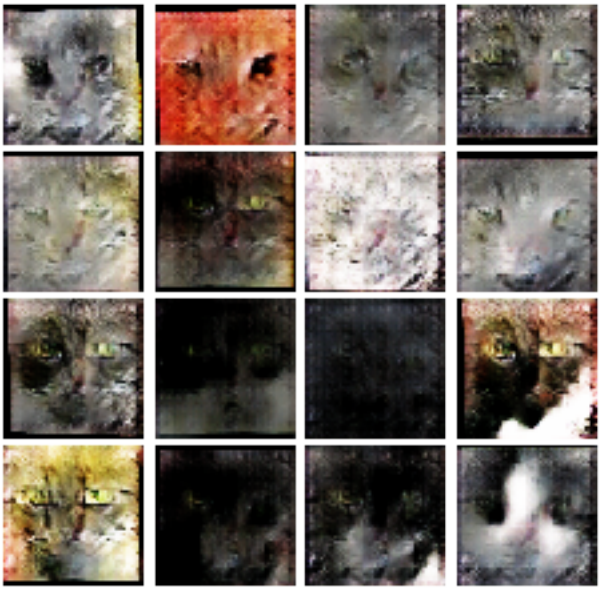


EPOCH: 26
Iter: 12500, D: 0.233, G:0.2767

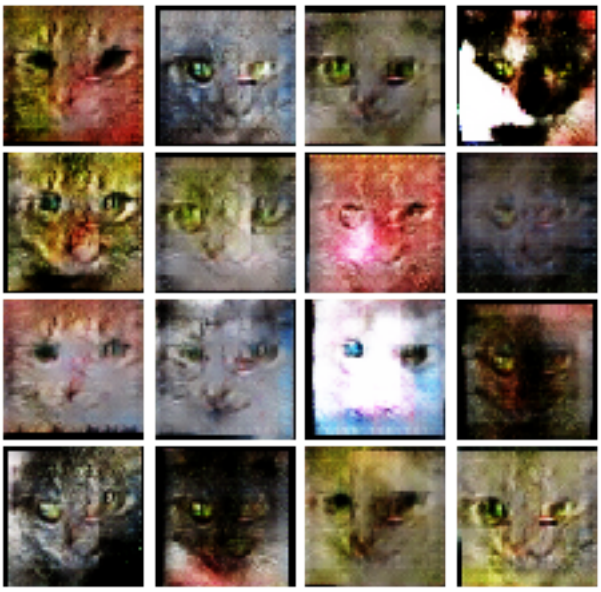
Iter: 12750, D: 0.1991, G:0.3223



EPOCH: 27
Iter: 13000, D: 0.282, G:0.4808

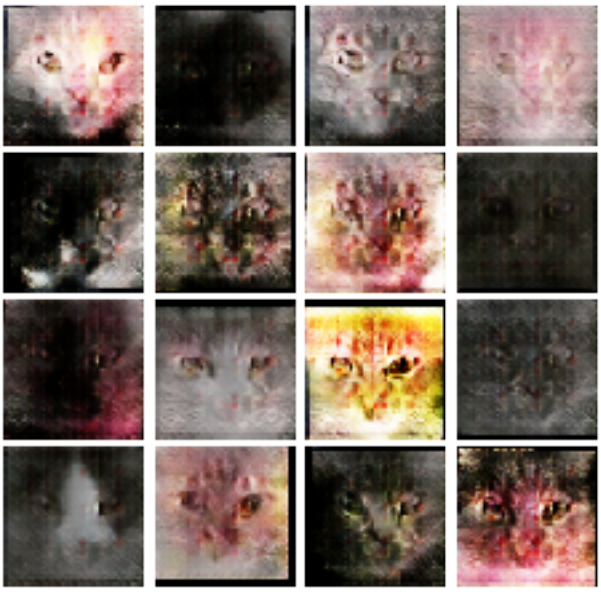


Iter: 13250, D: 0.2533, G:0.2752

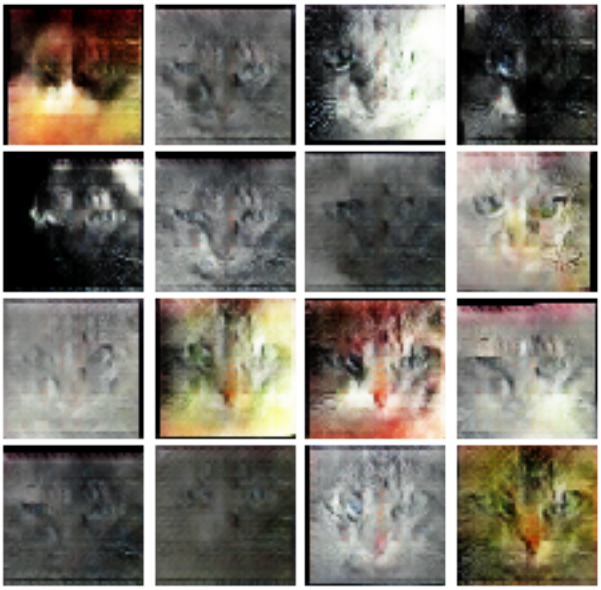


```
EPOCH:  28
Iter: 13500, D: 0.1563, G:0.2276
```

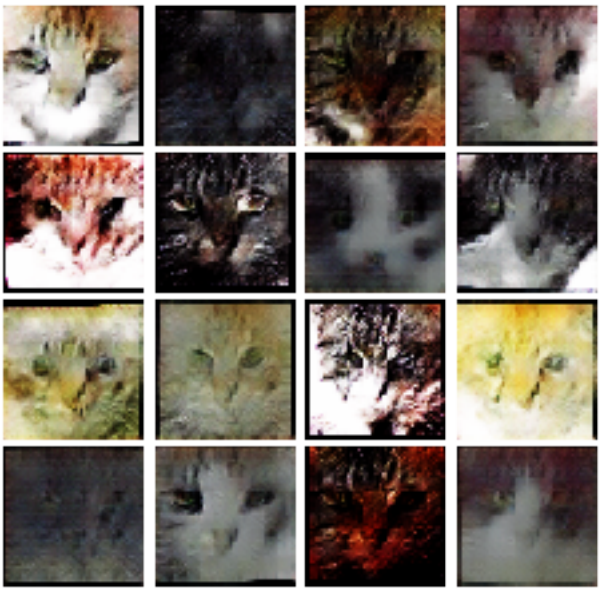
```
Iter: 13750, D: 0.1972, G:0.2149
```



EPOCH: 29
Iter: 14000, D: 0.3055, G:0.4638

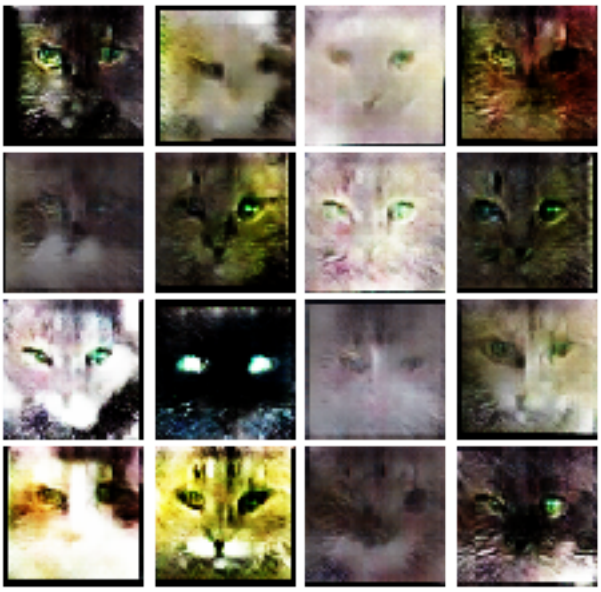


Iter: 14250, D: 0.1945, G:0.4129

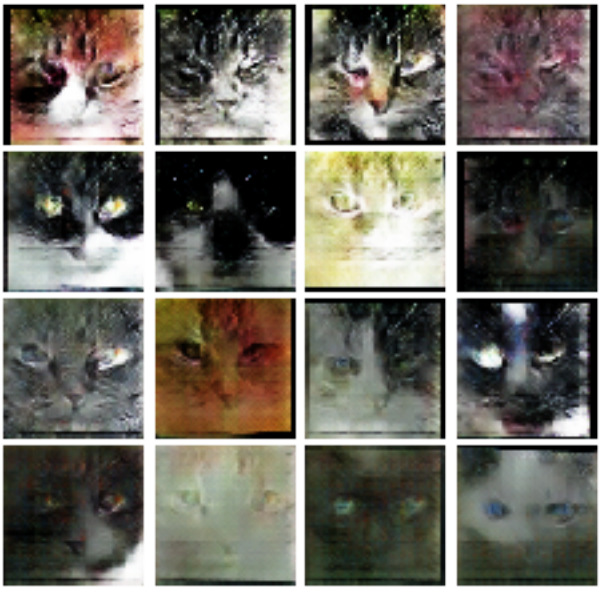


EPOCH: 30
Iter: 14500, D: 0.2831, G:0.1314

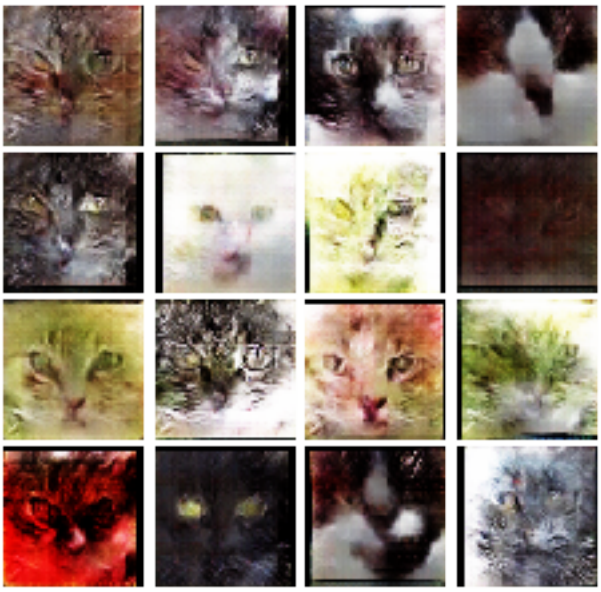
Iter: 14750, D: 0.1039, G:0.4137



```
EPOCH:  31
Iter: 15000, D: 0.1789, G:0.2493
```

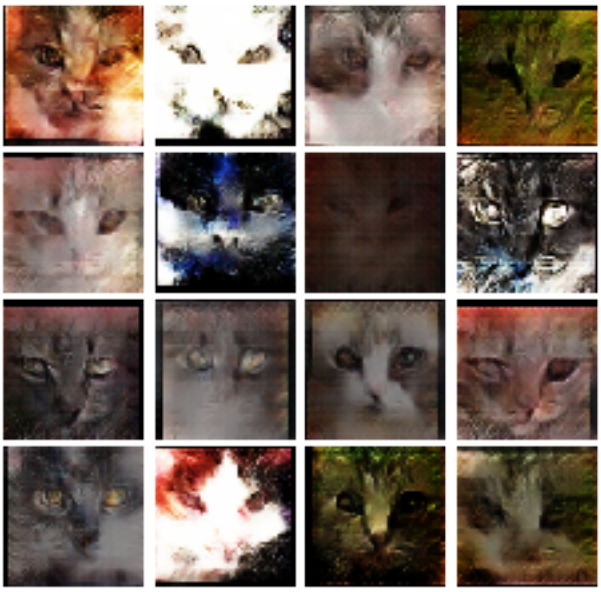


```
Iter: 15250, D: 0.2236, G:0.2336
```

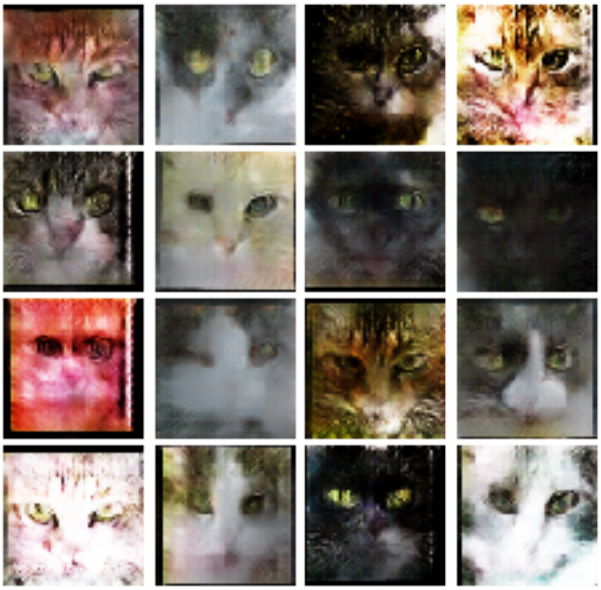


EPOCH: 32
Iter: 15500, D: 0.2092, G:0.1993

EPOCH: 33
Iter: 15750, D: 0.1577, G:0.2957



Iter: 16000, D: 0.2849, G:0.2428



EPOCH: 34
Iter: 16250, D: 0.1896, G:0.3592

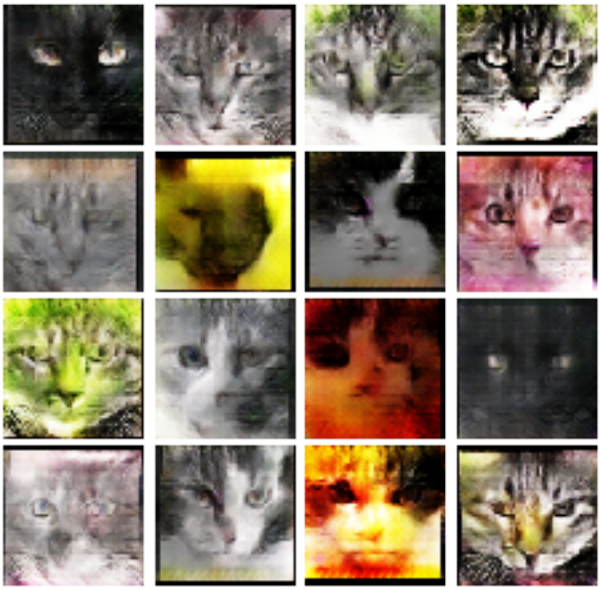
Iter: 16500, D: 0.1802, G:0.3017

EPOCH: 35
Iter: 16750, D: 0.2054, G:0.648

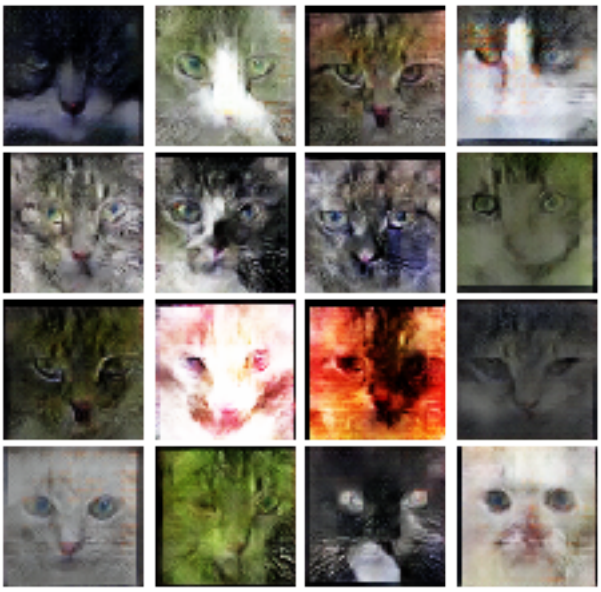
Iter: 17000, D: 0.4264, G:0.2593

EPOCH: 36
Iter: 17250, D: 0.08286, G:0.588

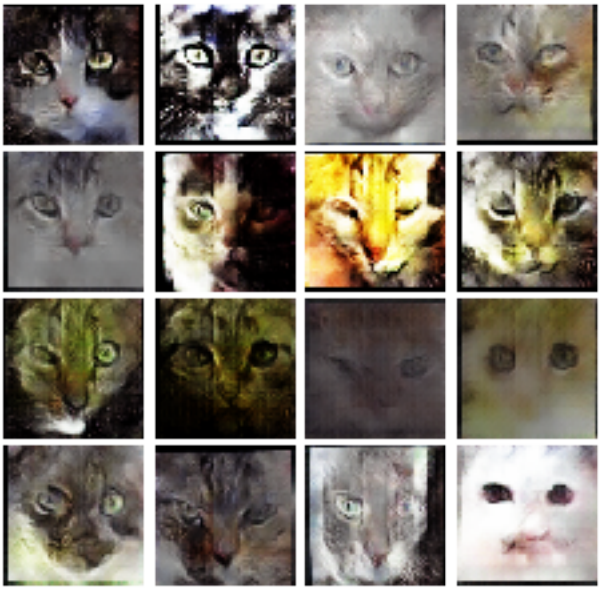
Iter: 17500, D: 0.2279, G:0.181



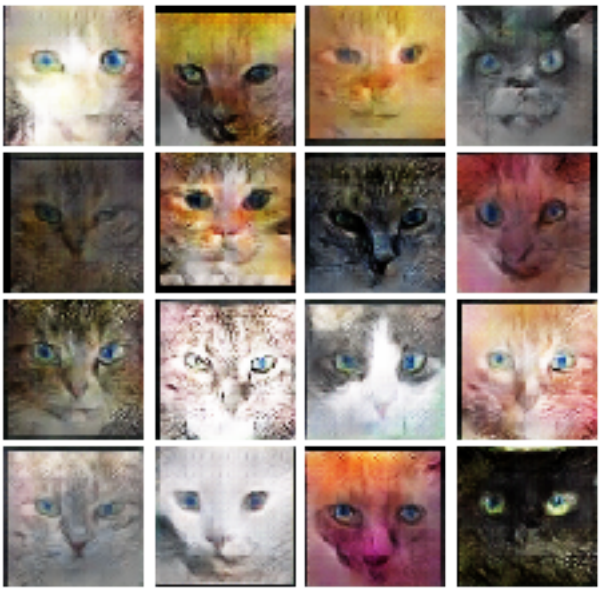
EPOCH: 37
Iter: 17750, D: 0.09918, G:0.3777



Iter: 18000, D: 0.1136, G:0.3387

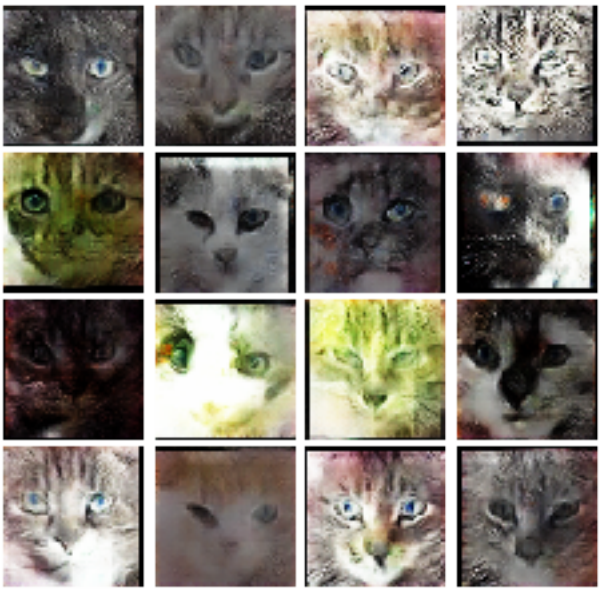


EPOCH: 38
Iter: 18250, D: 0.1886, G:0.2394

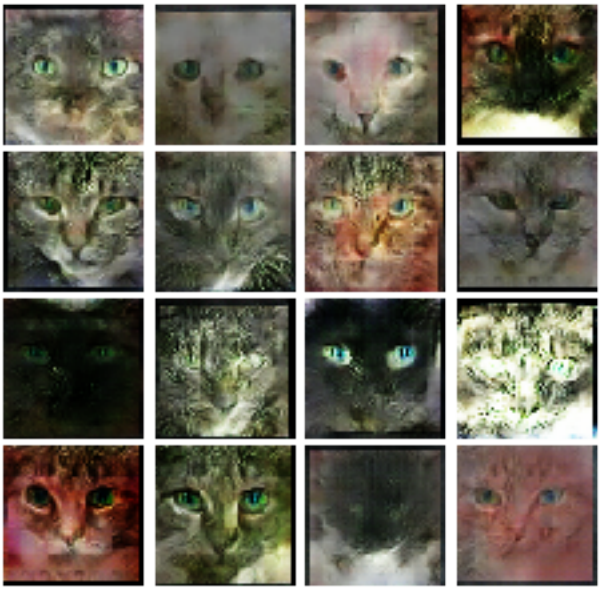


```
Iter: 18500, D: 0.1486, G:0.5488
```

```
EPOCH:  39
Iter: 18750, D: 0.1313, G:0.2739
```

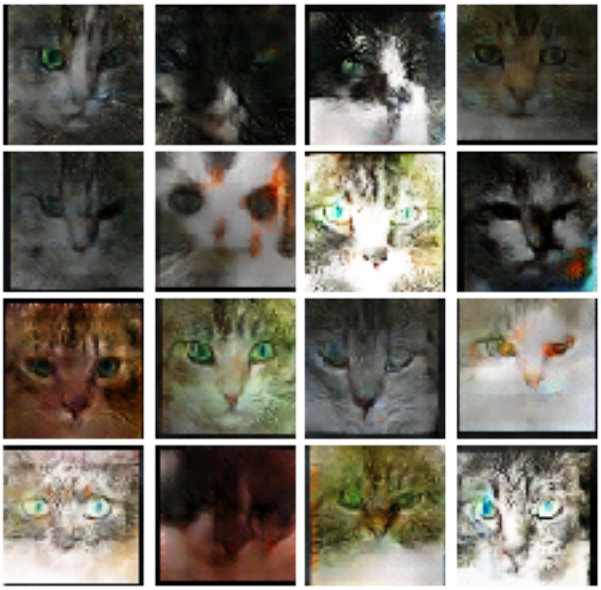


Iter: 19000, D: 0.245, G:0.4477



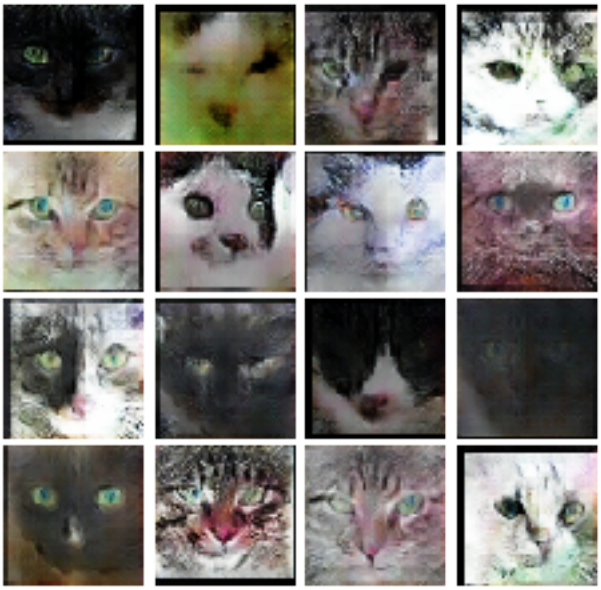
EPOCH: 40
Iter: 19250, D: 0.1639, G:0.3797

Iter: 19500, D: 0.2535, G:0.6392

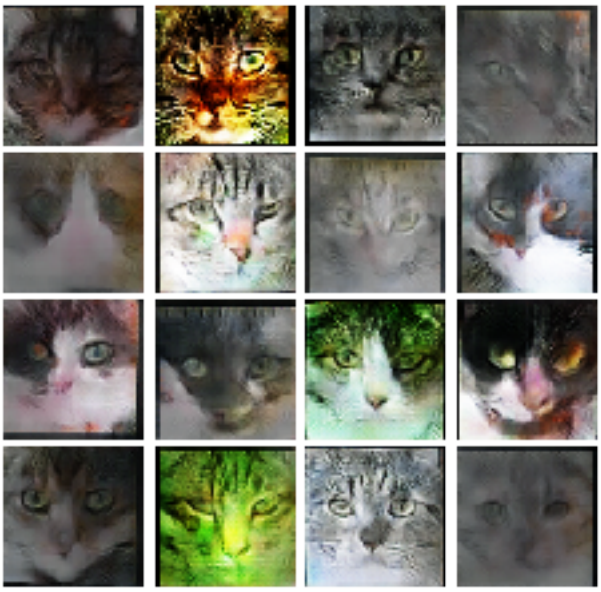


EPOCH: 41
Iter: 19750, D: 0.1919, G:0.2078

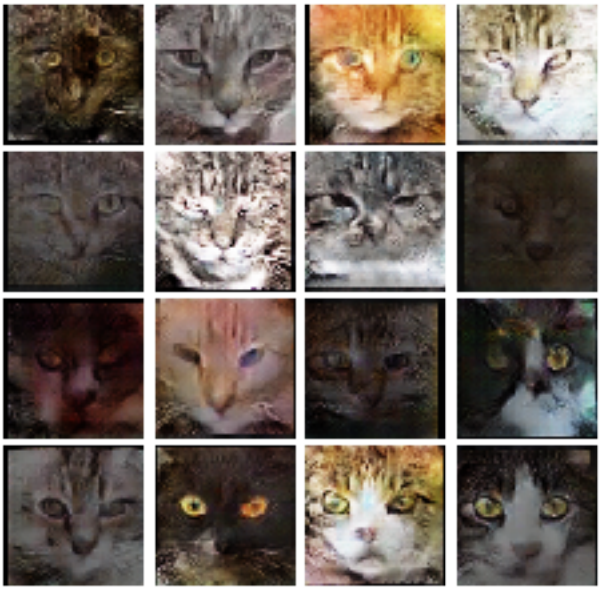
Iter: 20000, D: 0.1297, G:0.2076



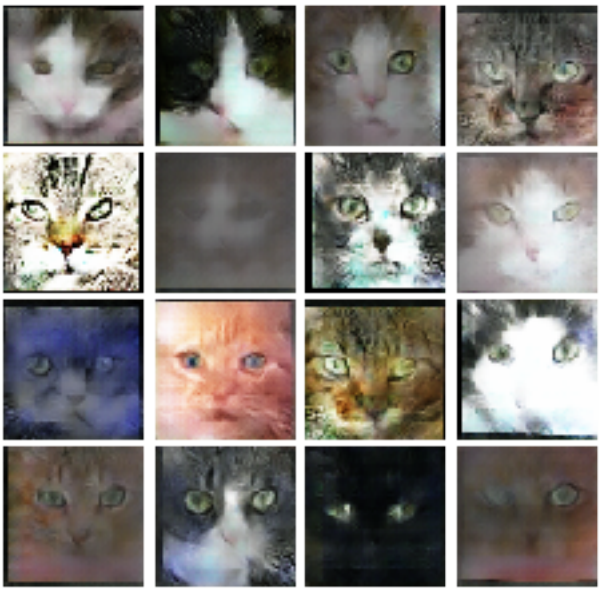
EPOCH: 42
Iter: 20250, D: 0.1377, G:0.2724



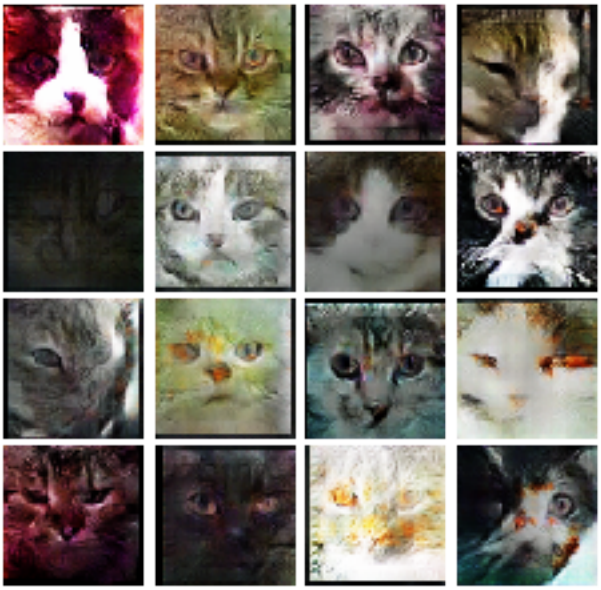
```
Iter: 20500, D: 0.2435, G:0.6204
```



```
EPOCH:  43
Iter: 20750, D: 0.1244, G:0.3809
```



Iter: 21000, D: 0.1453, G:0.3142



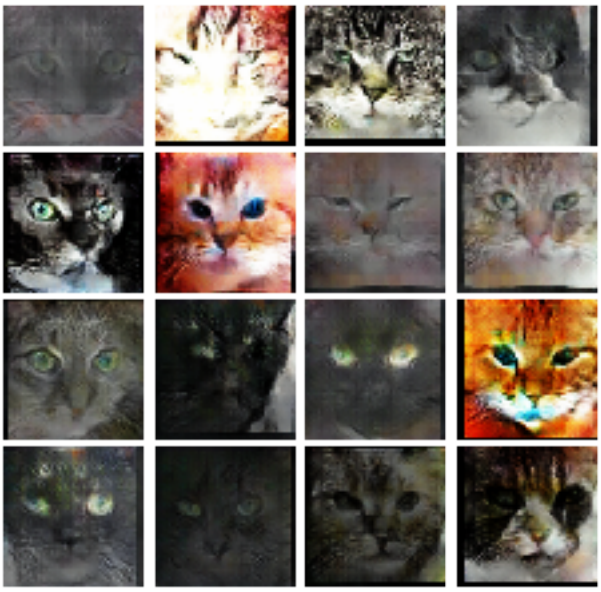
EPOCH: 44
Iter: 21250, D: 0.1048, G:0.3365

Iter: 21500, D: 0.09555, G:0.4174

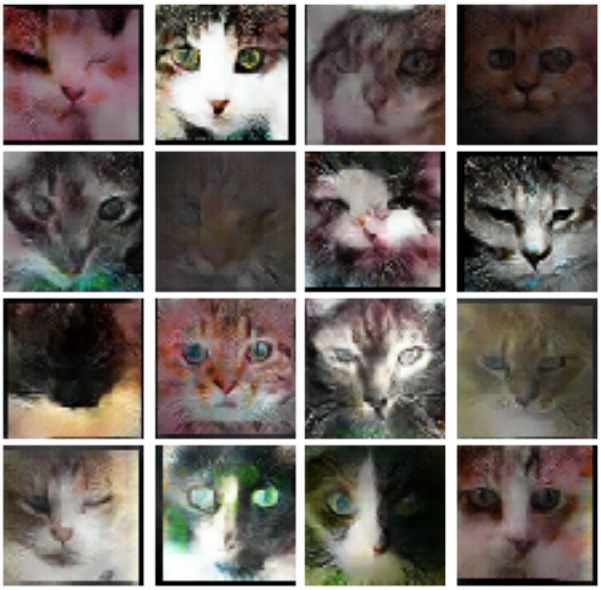
EPOCH: 45
Iter: 21750, D: 0.1418, G:0.4321

Iter: 22000, D: 0.1404, G:0.3123

EPOCH: 46
Iter: 22250, D: 0.1195, G:0.5211

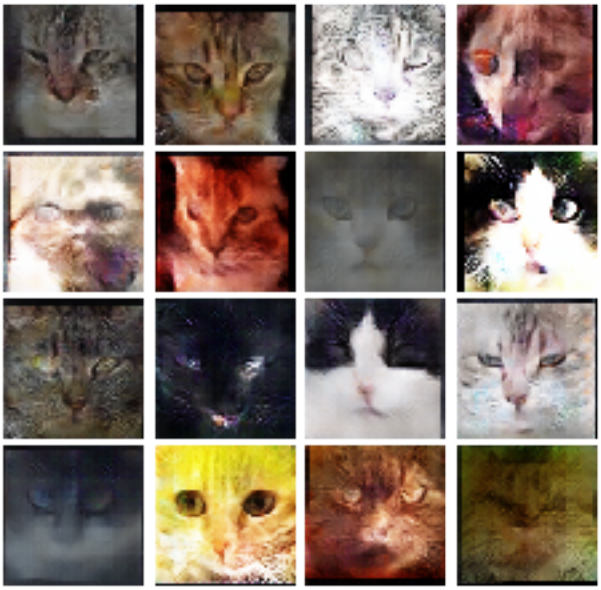


```
Iter: 22500, D: 0.1959, G:0.3301
```

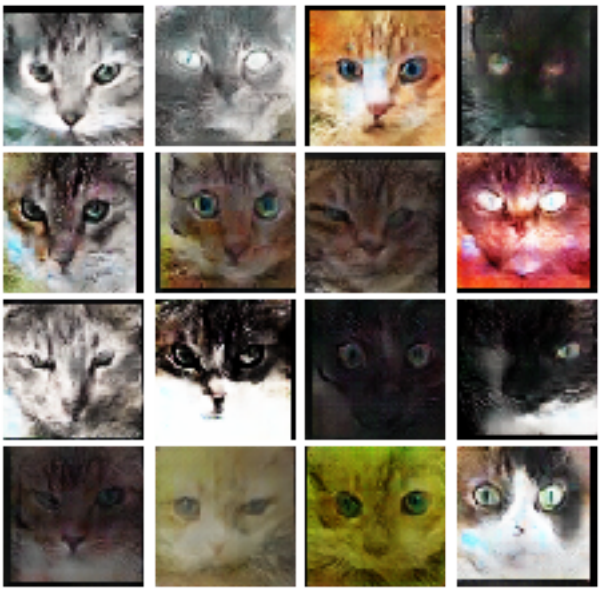


```
EPOCH:  47
Iter: 22750, D: 0.2284, G:0.6478
```

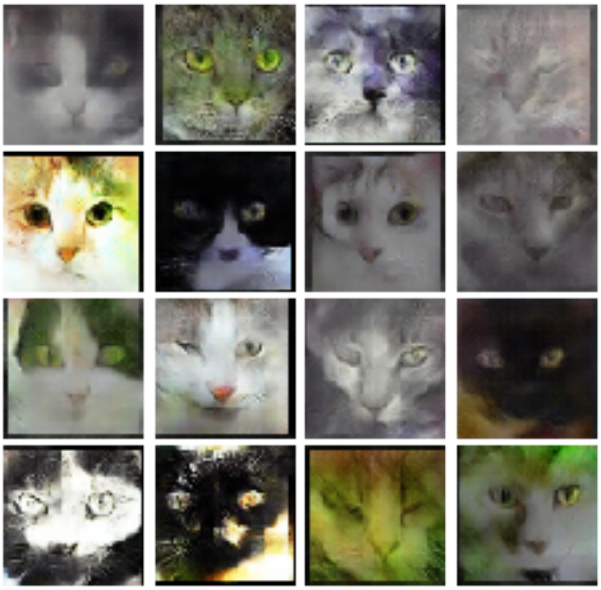
Iter: 23000, D: 0.1279, G:0.3845



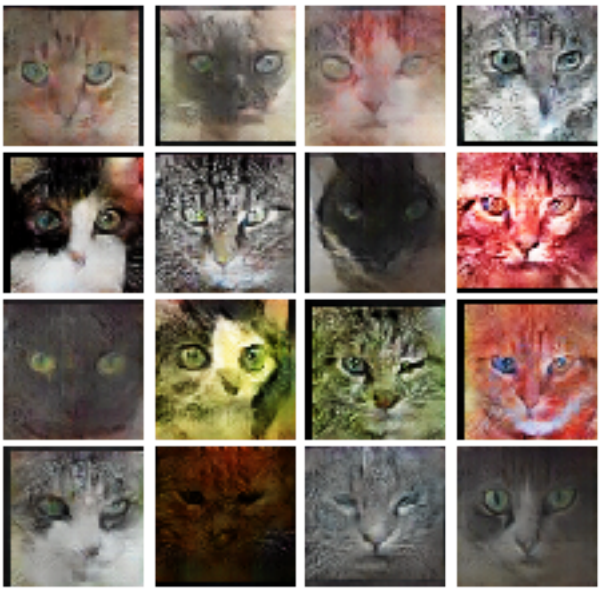
EPOCH: 48
Iter: 23250, D: 0.2364, G:0.1844



```
Iter: 23500, D: 0.129, G:0.2379
```

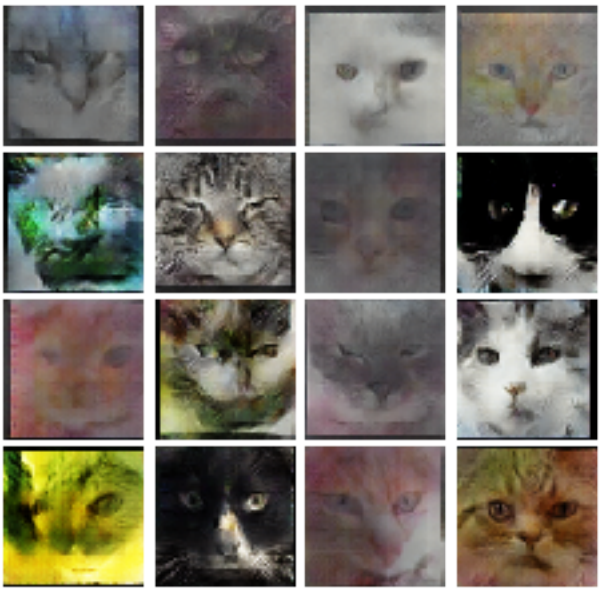


```
EPOCH:  49
Iter: 23750, D: 0.2295, G:0.2507
```



Iter: 24000, D: 0.163, G:0.3008

EPOCH: 50
Iter: 24250, D: 0.2197, G:0.2048



Iter: 24500, D: 0.1409, G:0.6342