MP4

April 12, 2024

1 (Optional) Colab Setup

If you aren't using Colab, you can delete the following code cell. This is just to help students with mounting to Google Drive to access the other .py files and downloading the data, which is a little trickier on Colab than on your local machine using Jupyter.

[1]:
```
# you will be prompted with a window asking to grant permissions
from google.colab import drive
drive.mount("/content/drive")
```

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

[ ]:
```
# fill in the path in your Google Drive in the string below. Note: do not␣
 ↪escape slashes or spaces
import os
datadir = "/content/assignment4"
if not os.path.exists(datadir):
  !ln -s "/content/drive/My Drive/CS444/assignment4" $datadir # TODO: Fill your␣
 ↪Assignment 4 path
os.chdir(datadir)
!pwd
```

2 Generative Adversarial Networks

For this part of the assignment you implement two different types of generative adversarial networks. We will train the networks on a dataset of cat 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
```

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

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

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

```
cuda:0
```

3 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.

3.0.1 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:

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

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 compute the binary cross entropy loss since it is more numerically stable than using a softmax followed by BCE loss. The BCE loss is needed to compute the log probability of the true label given the logits 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 $\log\left(1 - D(G(z))\right)$, we will be averaging over elements of the minibatch, so make sure to combine the loss by averaging instead of summing.

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

3.0.2 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
```

4 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
```

5 Data loading

The cat images we provide are RGB images with a resolution of 64x64. In order to prevent our discriminator from overfitting, we will need to perform some data augmentation.

TODO: Implement data augmentation by adding new transforms to the cell below. At the minimum, you should have a RandomCrop and a ColorJitter, but we encourage you to experiment with different augmentations to see how the performance of the GAN changes. See https://pytorch.org/vision/stable/transforms.html.

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

cat_train = ImageFolder(root=cat_root, transform=transforms.Compose([
  transforms.ToTensor(),

  # Example use of RandomCrop:
  transforms.Resize(int(1.15 * imsize)),
  transforms.RandomCrop(imsize),
]))

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

5.0.1 Visualize dataset

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

try:
  imgs = next(iter(cat_loader_train))[0].numpy().squeeze()
except:
```

```
  imgs = cat_loader_train.__iter__().next()[0].numpy().squeeze()

show_images(imgs, color=True)
```

6 Training

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

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

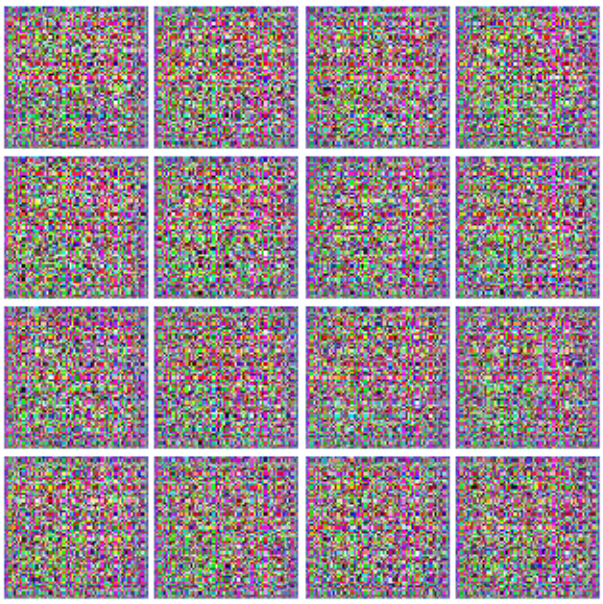
6.0.1 Train GAN

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

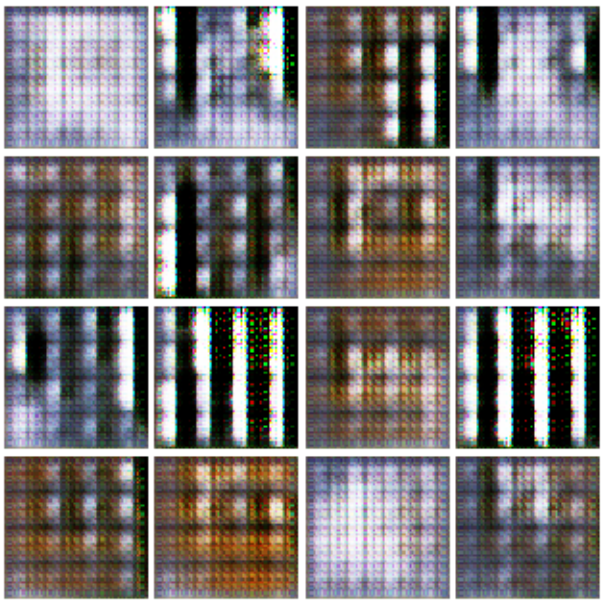
[ ]:
```
D_optimizer = torch.optim.Adam(D.parameters(), lr=learning_rate, betas = (0.5,␣
↪0.999))
G_optimizer = torch.optim.Adam(G.parameters(), lr=learning_rate, betas = (0.5,␣
↪0.999))
```

[ ]:
```
# original gan
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)
```

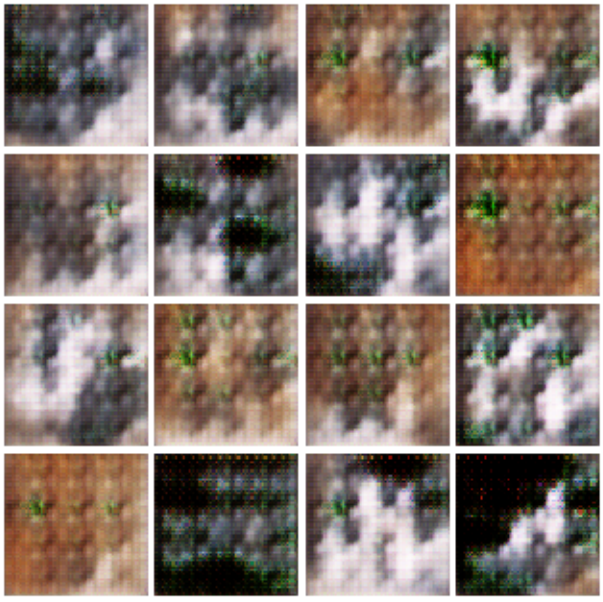
```
EPOCH:  1
Iter: 0, D: 1.434, G:6.76
```



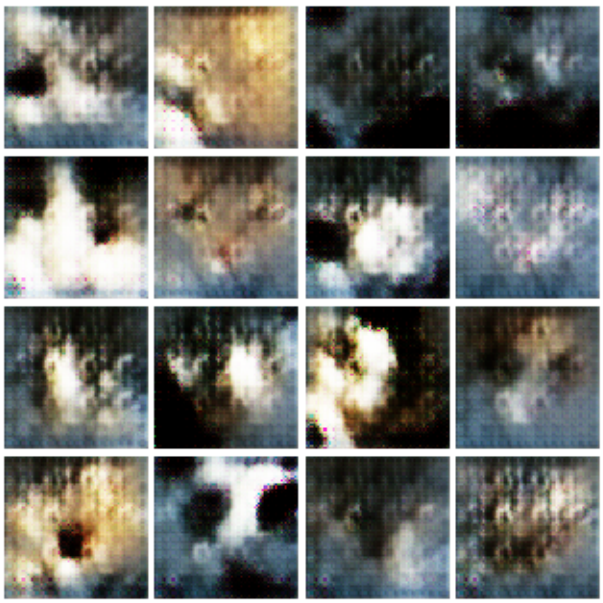
```
EPOCH:  2
Iter: 250, D: 0.4139, G:2.922
```



EPOCH: 3
Iter: 500, D: 1.321, G:0.954

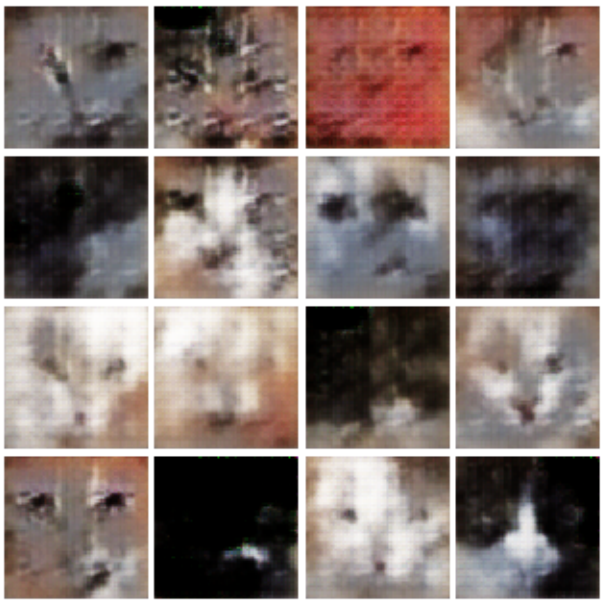


EPOCH: 4
Iter: 750, D: 1.81, G:2.555

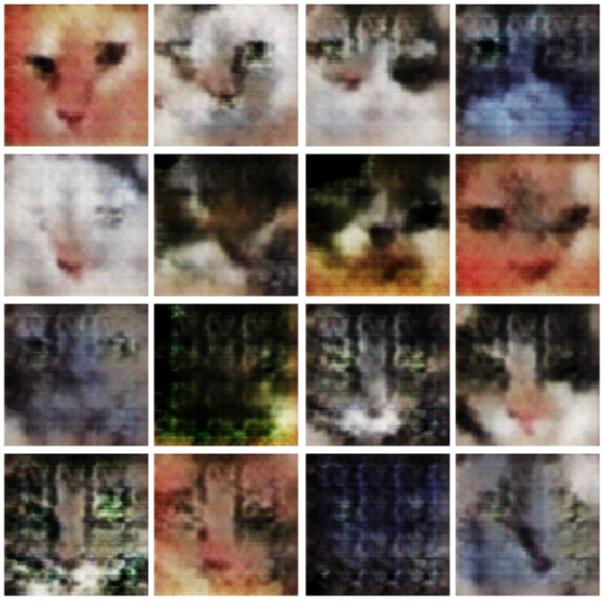


EPOCH: 5
Iter: 1000, D: 2.105, G:1.181

EPOCH: 6
Iter: 1250, D: 0.7123, G:1.995



EPOCH: 7
Iter: 1500, D: 1.116, G:2.779



EPOCH: 8
Iter: 1750, D: 1.118, G:1.565

EPOCH: 9
Iter: 2000, D: 0.5186, G:1.965

EPOCH: 10
Iter: 2250, D: 1.07, G:5.21

EPOCH: 11
Iter: 2500, D: 0.4391, G:3.272

EPOCH: 12
Iter: 2750, D: 0.6128, G:3.432

```
EPOCH:  13
Iter: 3000, D: 0.8975, G:1.864
```

```
EPOCH:  14
Iter: 3250, D: 0.4588, G:3.39
```

EPOCH: 15
Iter: 3500, D: 0.8431, G:6.011

EPOCH: 16
Iter: 3750, D: 0.6837, G:2.197

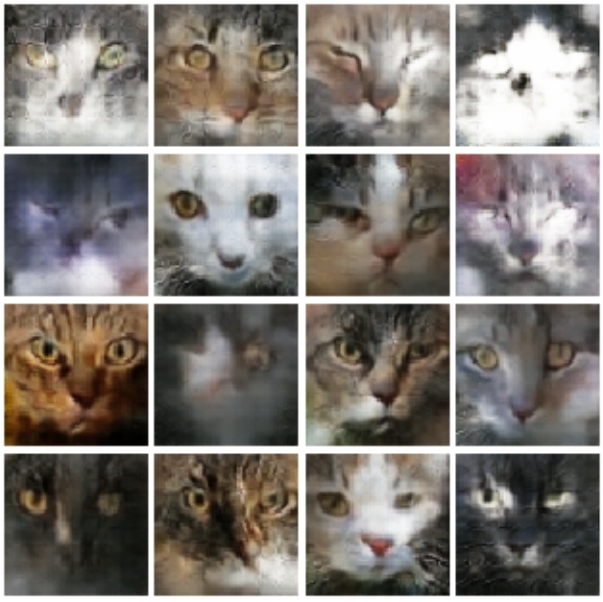
EPOCH: 17
Iter: 4000, D: 0.76, G:3.597

EPOCH: 18
Iter: 4250, D: 2.298, G:0.9215

EPOCH: 19
Iter: 4500, D: 0.4436, G:4.173

EPOCH: 20
Iter: 4750, D: 0.6505, G:2.678

EPOCH: 21
Iter: 5000, D: 2.109, G:1.606



EPOCH: 22
Iter: 5250, D: 0.5301, G:5.821

EPOCH: 23
Iter: 5500, D: 0.3504, G:3.502

EPOCH: 24
Iter: 5750, D: 0.3673, G:2.862

EPOCH: 25
Iter: 6000, D: 0.469, G:4.492

EPOCH: 26
Iter: 6250, D: 0.3664, G:3.605

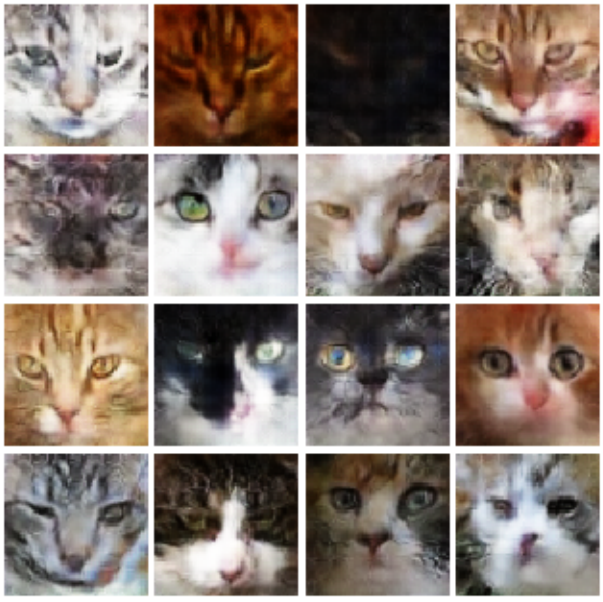
EPOCH: 27
Iter: 6500, D: 1.53, G:1.404

EPOCH: 28
Iter: 6750, D: 0.2472, G:2.748

```
EPOCH:  29
Iter: 7000, D: 1.059, G:6.211
```

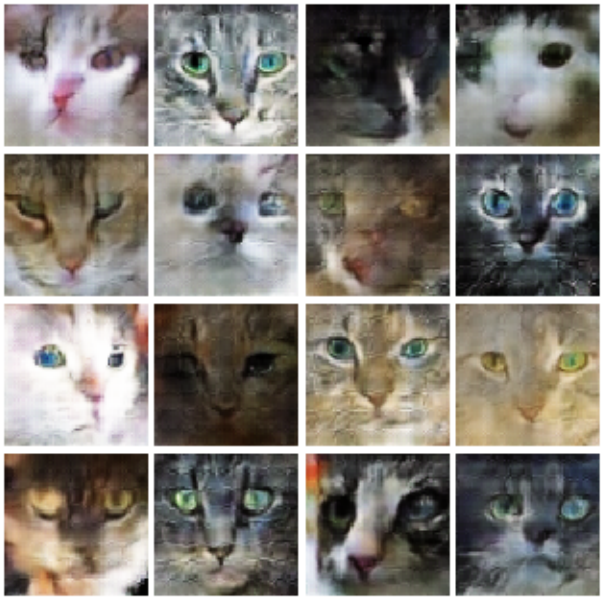
```
EPOCH:  30
Iter: 7250, D: 1.354, G:8.999
```

EPOCH: 31
Iter: 7500, D: 0.3022, G:4.046

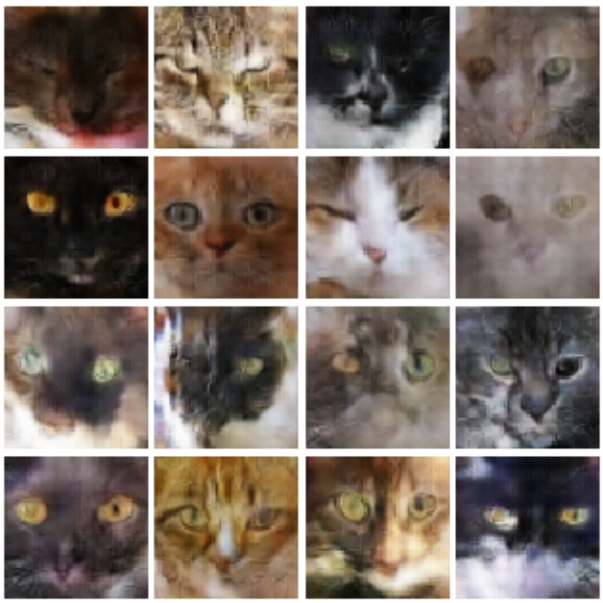


EPOCH: 32
Iter: 7750, D: 0.7092, G:5.825

EPOCH: 33
Iter: 8000, D: 1.464, G:1.967

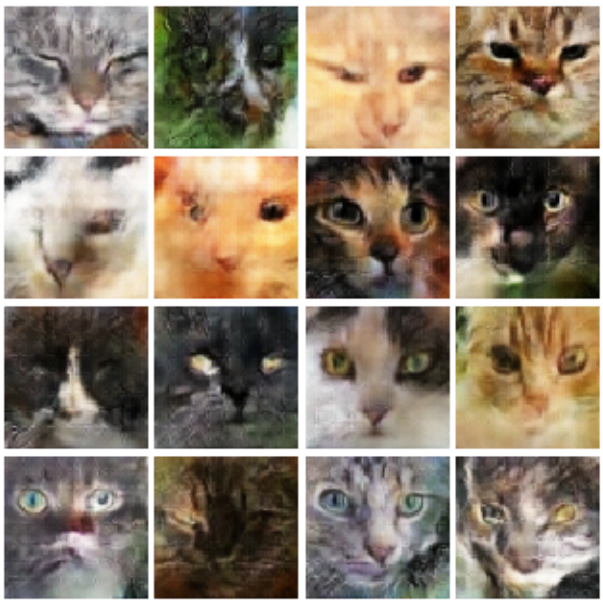


EPOCH: 34
Iter: 8250, D: 0.1711, G:4.863

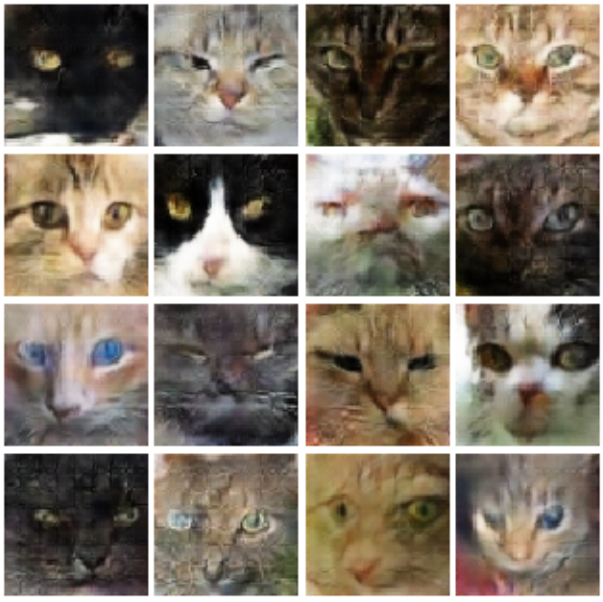


EPOCH: 35
Iter: 8500, D: 0.2343, G:4.1

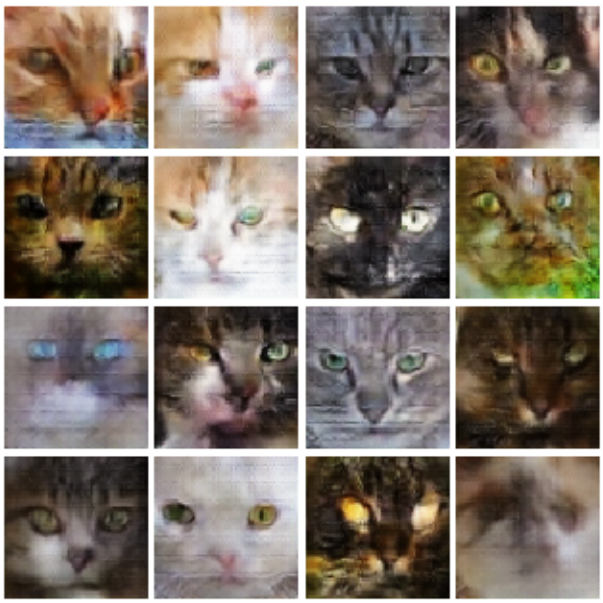
EPOCH: 36
Iter: 8750, D: 0.322, G:4.124



EPOCH: 37
Iter: 9000, D: 0.4345, G:6.169



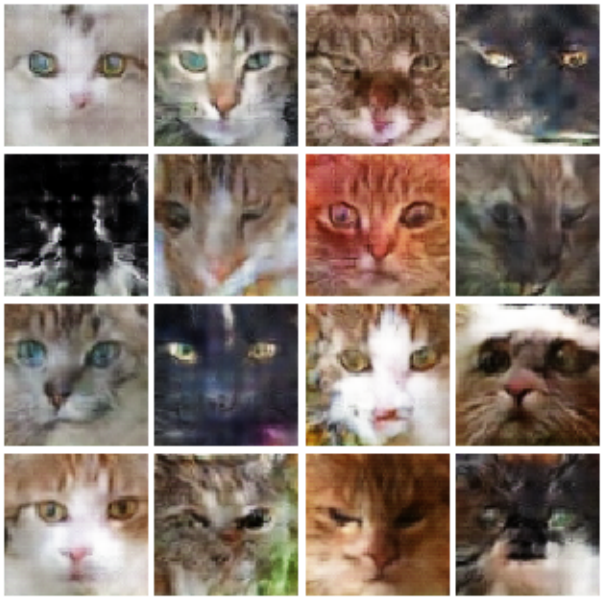
EPOCH: 38
Iter: 9250, D: 0.6034, G:5.286



EPOCH: 39
Iter: 9500, D: 0.9227, G:8.206

EPOCH: 40
Iter: 9750, D: 0.5856, G:3.554

```
EPOCH:  41
Iter: 10000, D: 0.3183, G:4.089
```

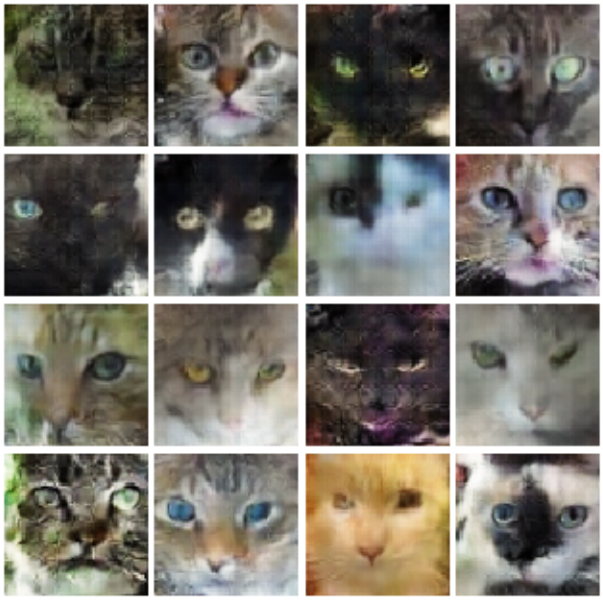


```
EPOCH:  42
Iter: 10250, D: 0.3083, G:3.383
```

EPOCH: 43
Iter: 10500, D: 0.1295, G:4.307

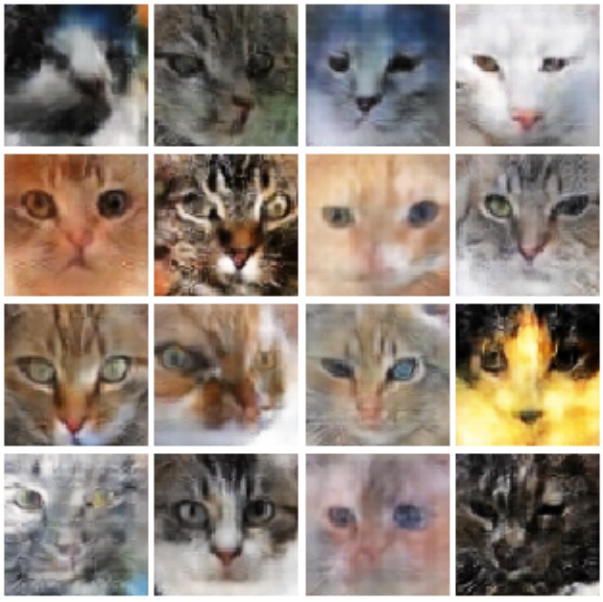
EPOCH: 44
Iter: 10750, D: 0.1736, G:4.867

```
EPOCH:  45
Iter: 11000, D: 0.5499, G:6.571
```

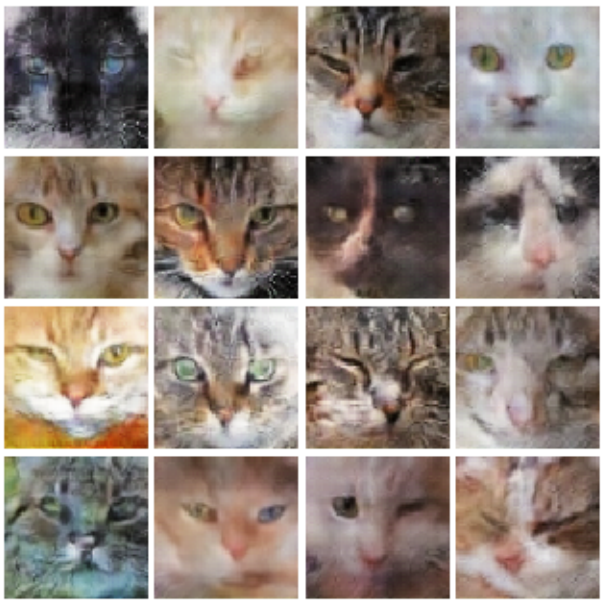


```
EPOCH:  46
Iter: 11250, D: 0.2223, G:6.768
```

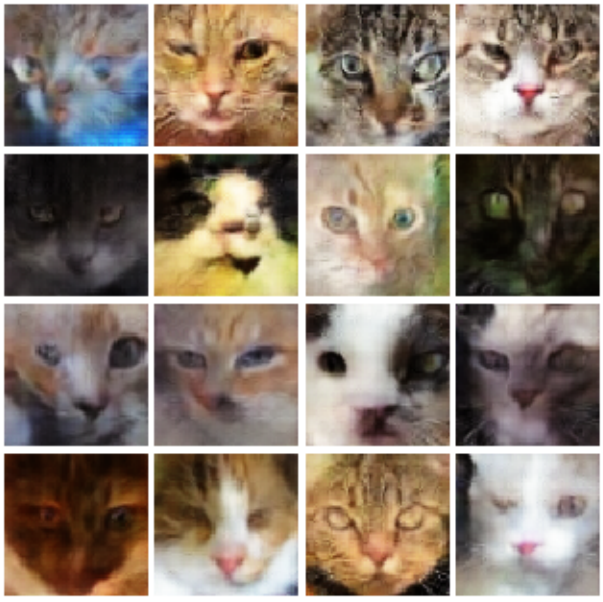
EPOCH: 47
Iter: 11500, D: 0.2271, G:5.016



EPOCH: 48
Iter: 11750, D: 0.5456, G:3.43



```
EPOCH:  49
Iter: 12000, D: 0.2531, G:3.519
```



```
EPOCH:  50
Iter: 12250, D: 5.078, G:2.906
```

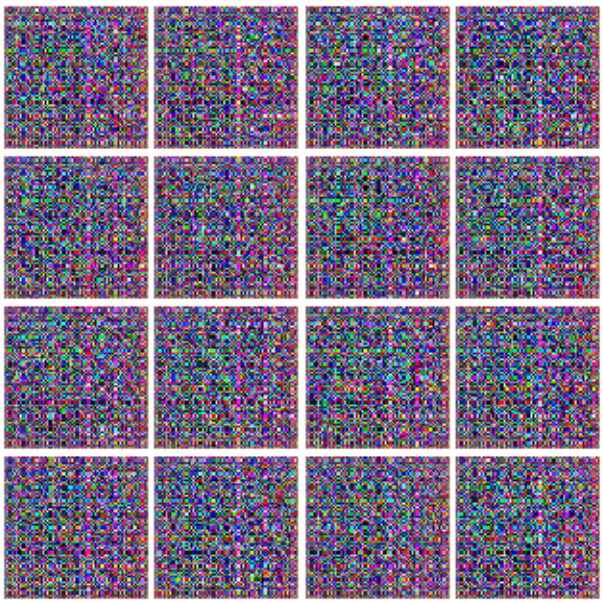
6.0.2 Train LS-GAN

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

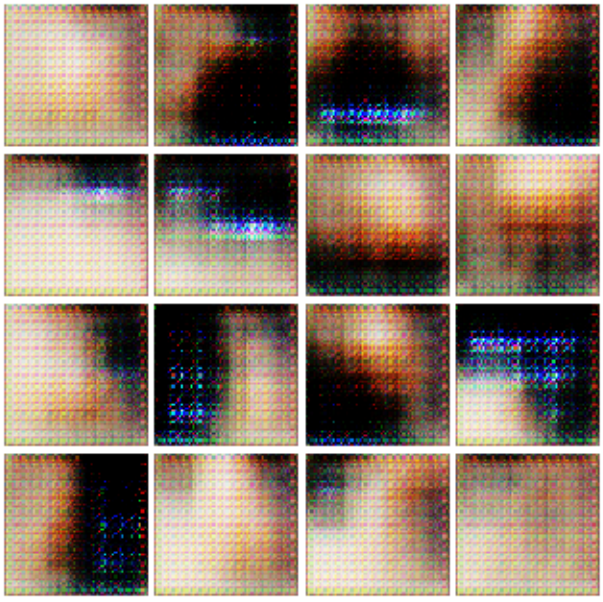
[ ]:
```
D_optimizer = torch.optim.Adam(D.parameters(), lr=learning_rate, betas = (0.5,␣
↪0.999))
G_optimizer = torch.optim.Adam(G.parameters(), lr=learning_rate, betas = (0.5,␣
↪0.999))
```

[ ]:
```
# ls-gan
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)
```

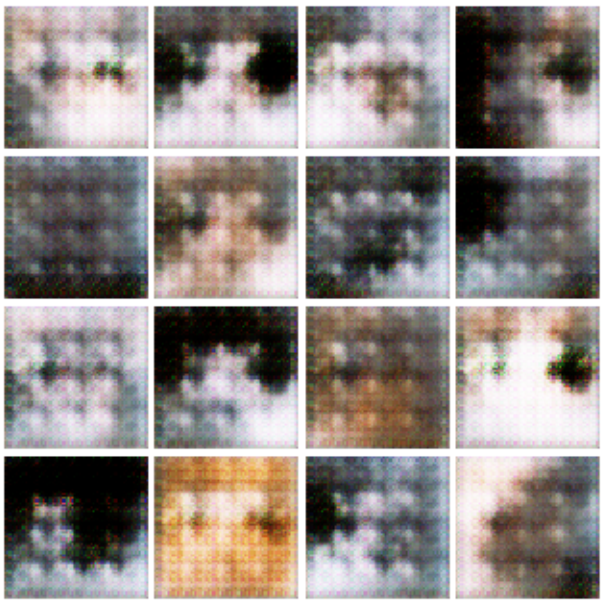
```
EPOCH:  1
Iter: 0, D: 0.4937, G:32.85
```



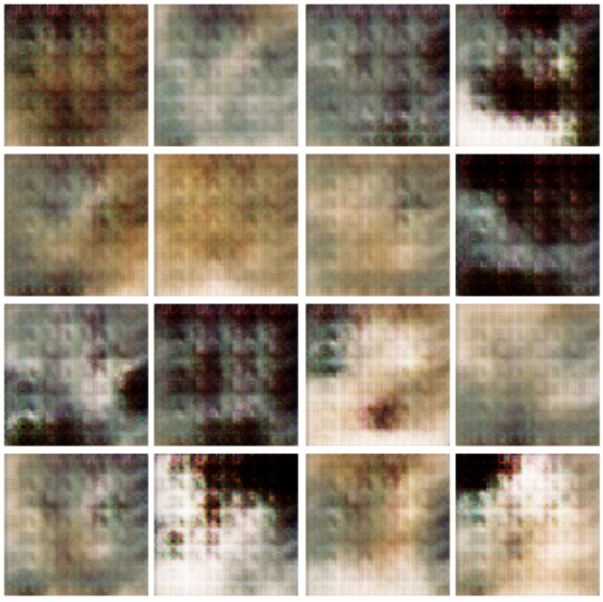
```
EPOCH:  2
Iter: 250, D: 0.2076, G:0.2198
```



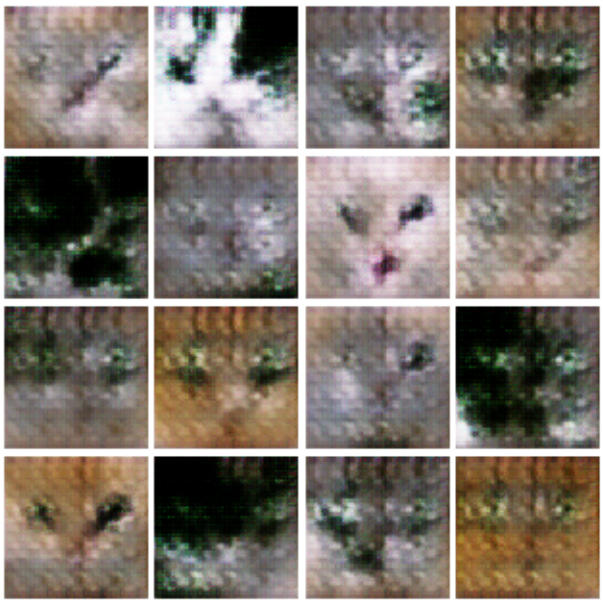
```
EPOCH:  3
Iter: 500, D: 0.7306, G:0.06443
```



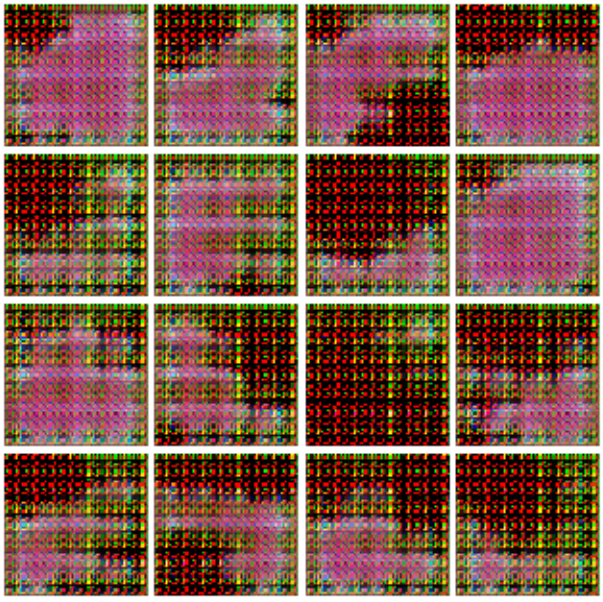
```
EPOCH:  4
Iter: 750, D: 0.1666, G:0.2884
```



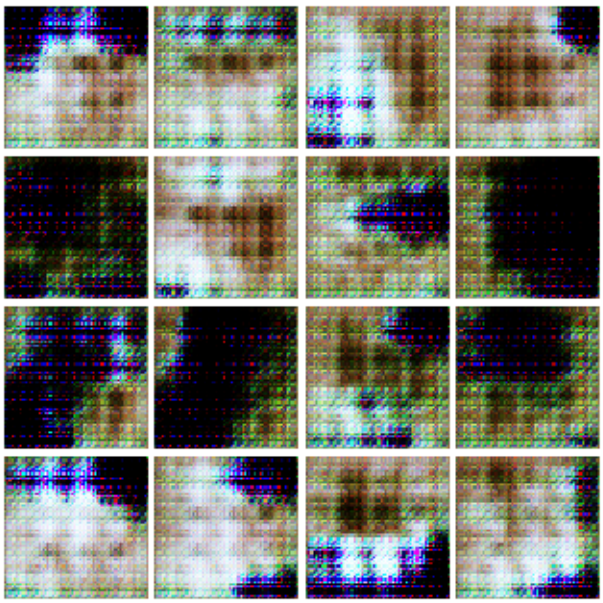
```
EPOCH:  5
Iter: 1000, D: 0.2997, G:0.6397
```



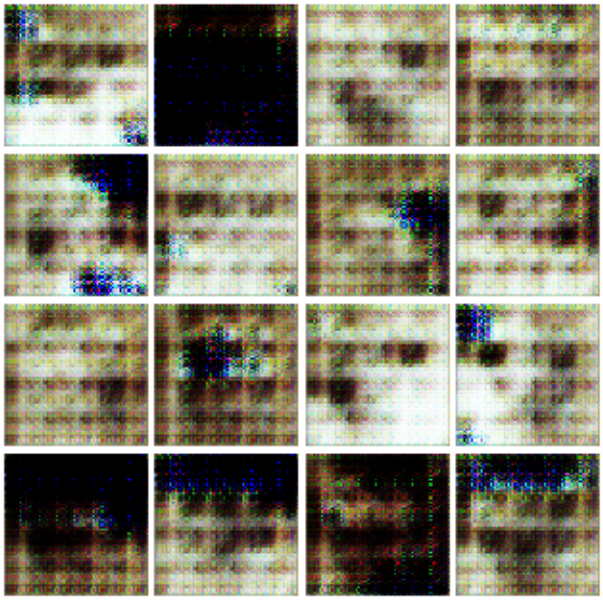
```
EPOCH:  6
Iter: 1250, D: 0.1226, G:0.2547
```



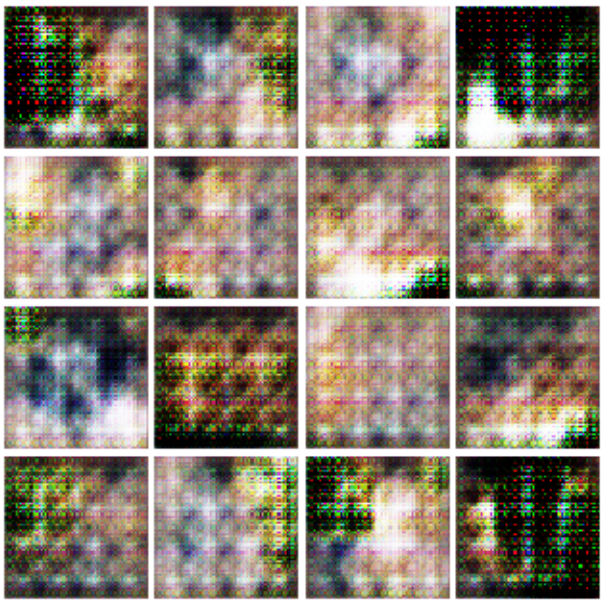
```
EPOCH:  7
Iter: 1500, D: 0.2253, G:0.1375
```



EPOCH: 8
Iter: 1750, D: 0.2253, G:0.1999

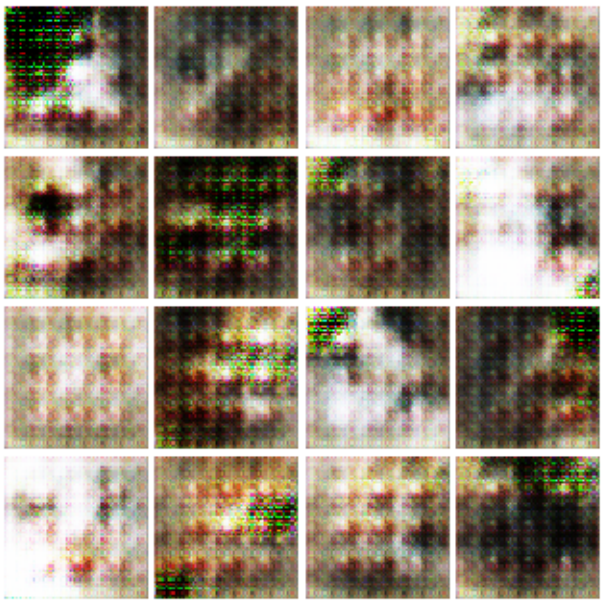


EPOCH: 9
Iter: 2000, D: 0.2585, G:0.1423

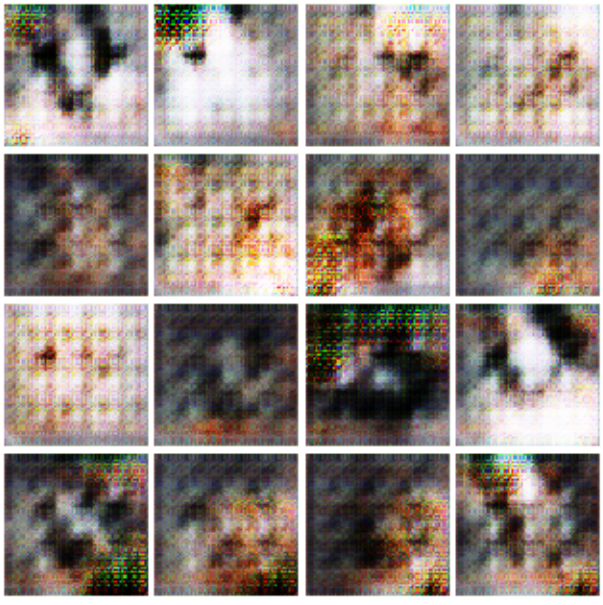


EPOCH: 10
Iter: 2250, D: 0.2479, G:0.1602

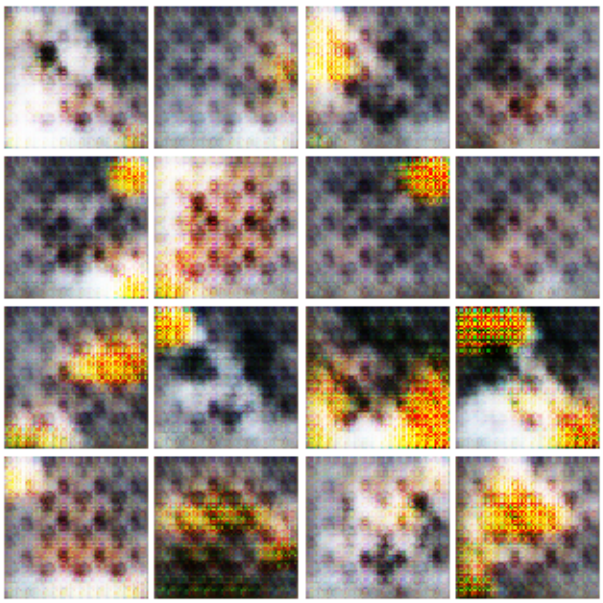
EPOCH: 11
Iter: 2500, D: 0.2509, G:0.1554



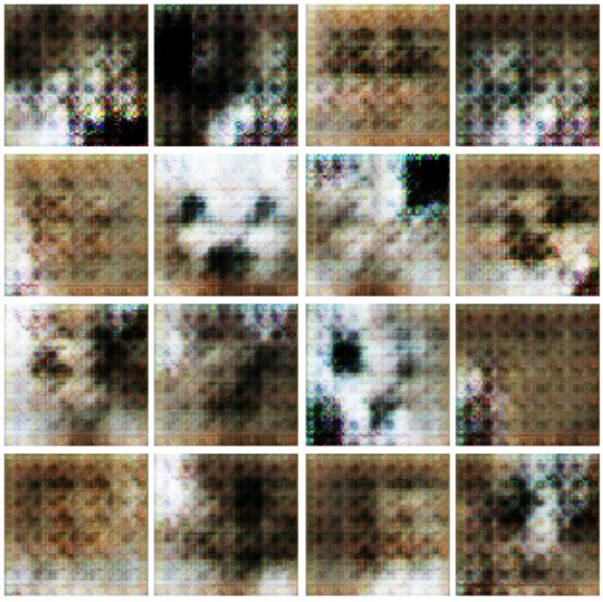
EPOCH: 12
Iter: 2750, D: 0.2521, G:0.1607



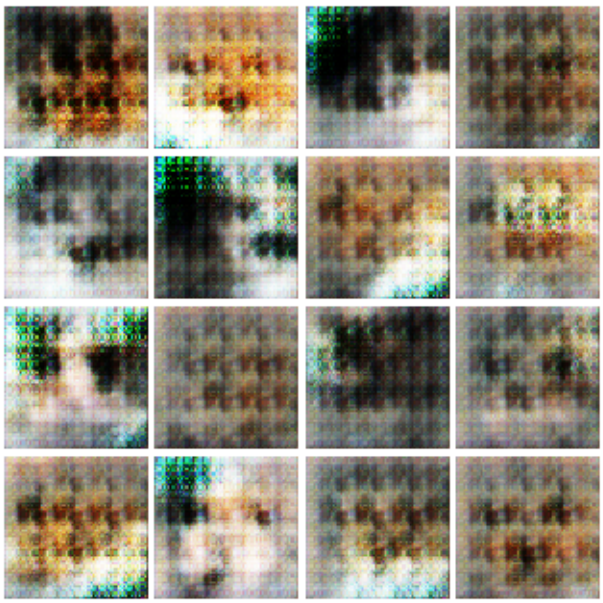
EPOCH: 13
Iter: 3000, D: 0.2594, G:0.1666



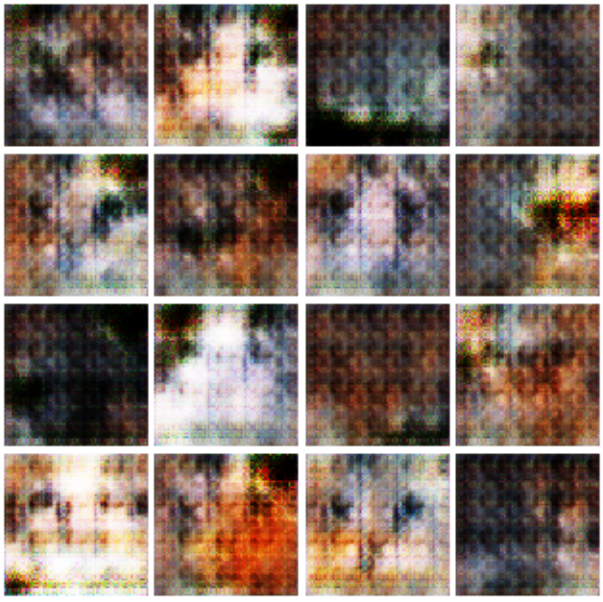
```
EPOCH:  14
Iter: 3250, D: 0.2588, G:0.1337
```



```
EPOCH:  15
Iter: 3500, D: 0.2274, G:0.1446
```

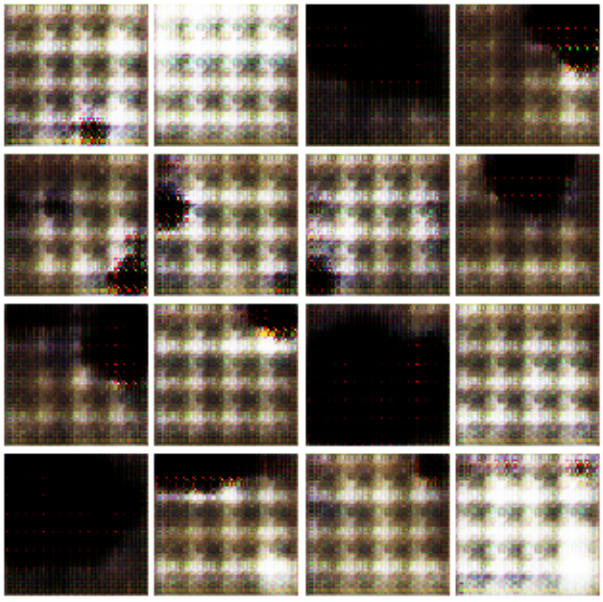


EPOCH: 16
Iter: 3750, D: 0.2629, G:0.1613

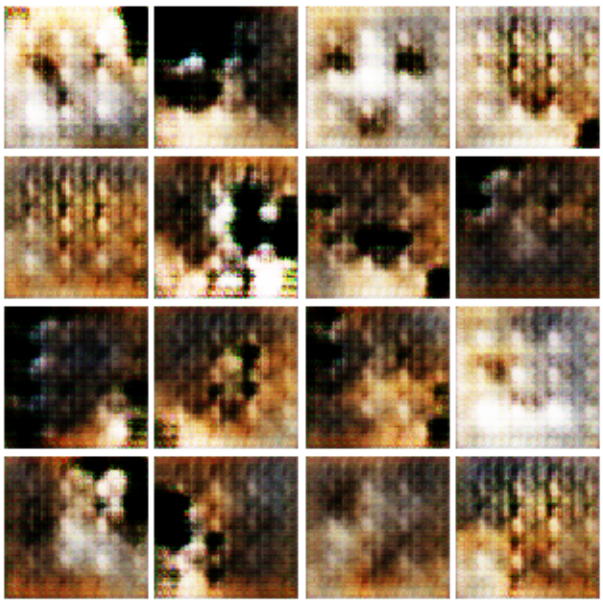


EPOCH: 17
Iter: 4000, D: 0.2655, G:0.1462

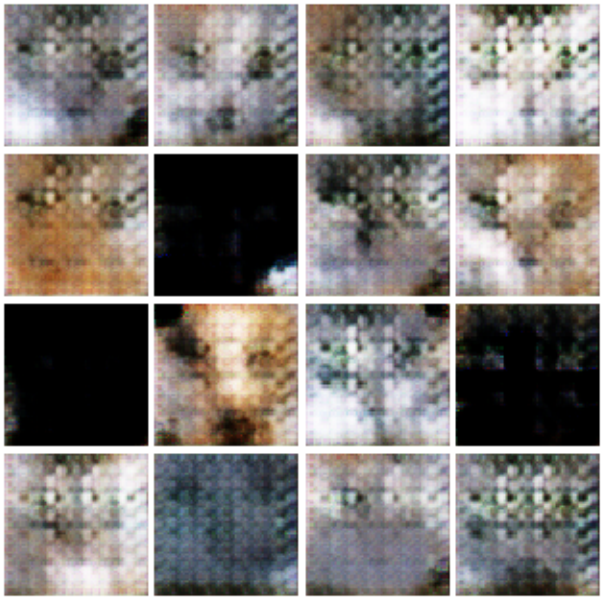
EPOCH: 18
Iter: 4250, D: 0.3798, G:0.4034



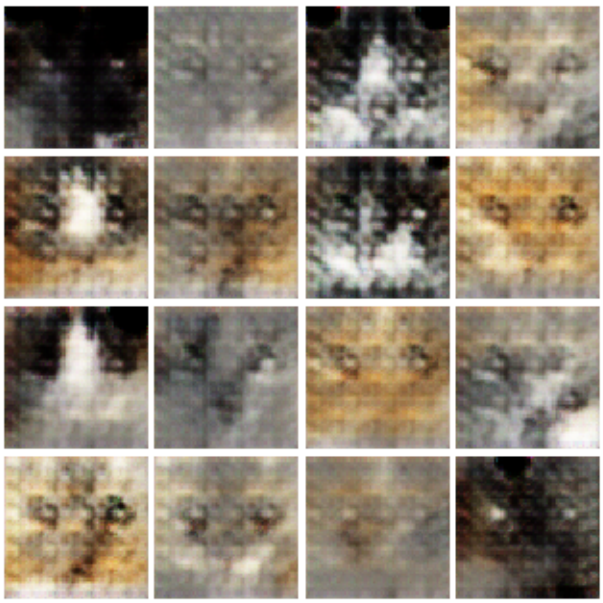
EPOCH: 19
Iter: 4500, D: 0.2048, G:0.1754



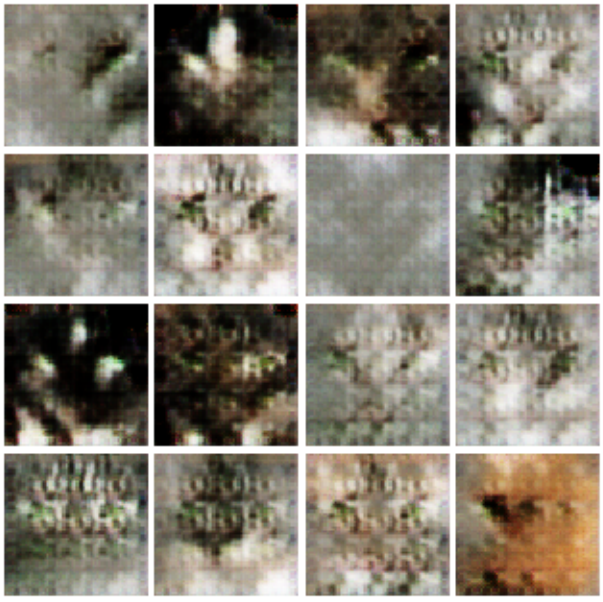
EPOCH: 20
Iter: 4750, D: 0.1935, G:0.3828



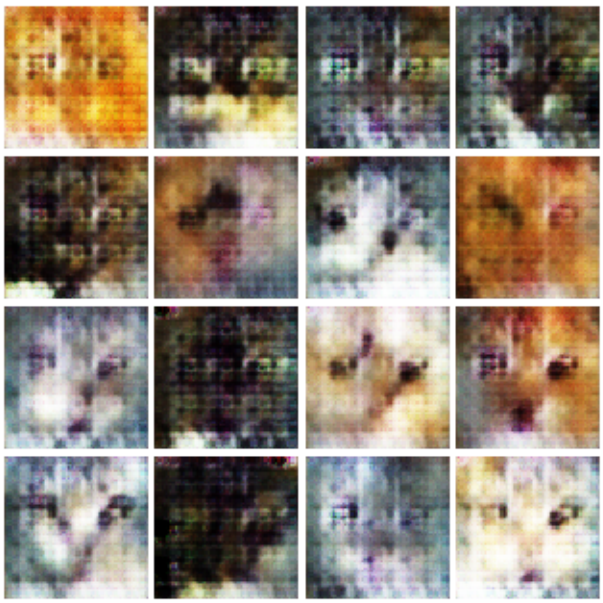
EPOCH: 21
Iter: 5000, D: 0.1796, G:0.1603



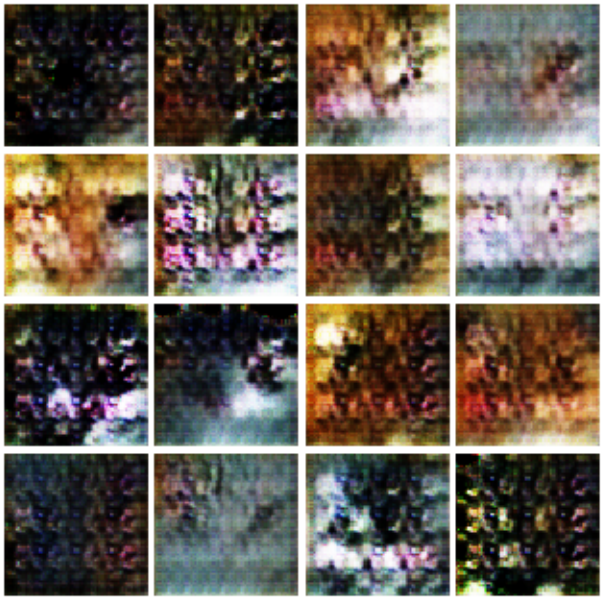
EPOCH: 22
Iter: 5250, D: 0.2654, G:0.5572



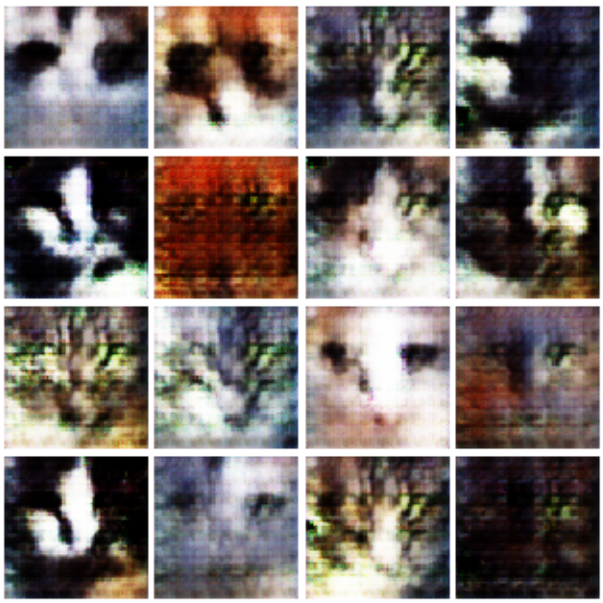
EPOCH: 23
Iter: 5500, D: 0.27, G:0.1135



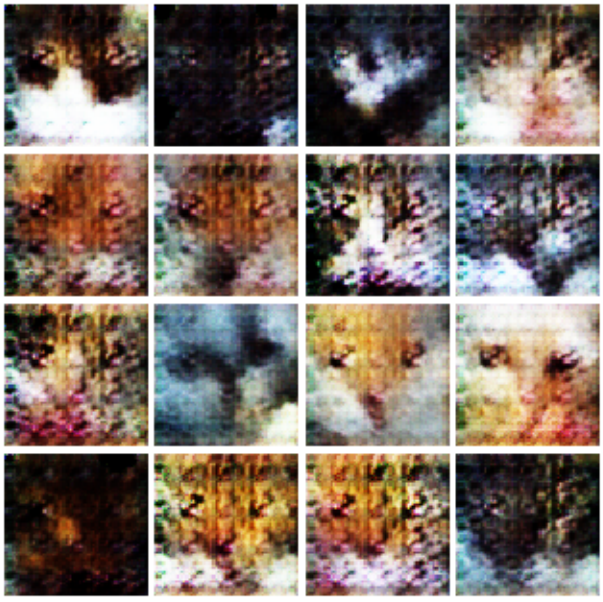
EPOCH: 24
Iter: 5750, D: 0.2412, G:0.2736



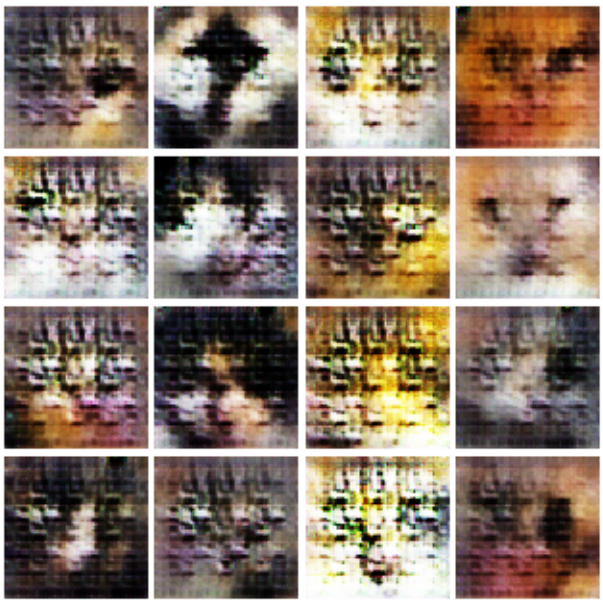
EPOCH: 25
Iter: 6000, D: 0.2285, G:0.3909



EPOCH: 26
Iter: 6250, D: 0.2662, G:0.2258



EPOCH: 27
Iter: 6500, D: 0.4462, G:0.3337



```
EPOCH:  28
Iter: 6750, D: 0.2506, G:0.1854
```

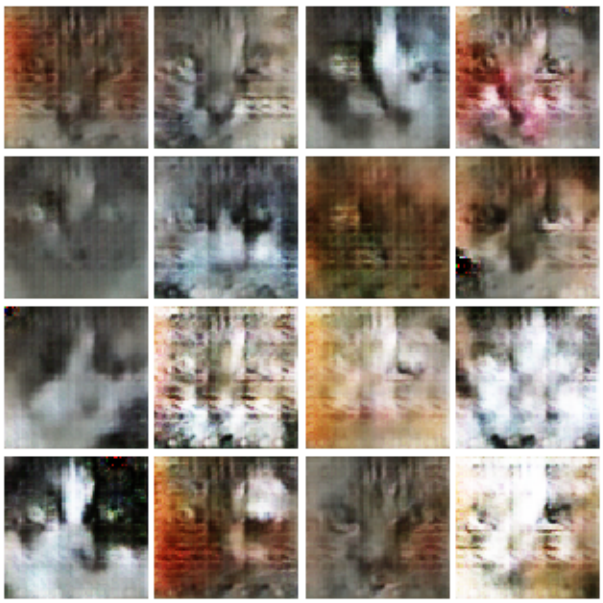
```
EPOCH:  29
Iter: 7000, D: 0.2969, G:0.2533
```

EPOCH: 30
Iter: 7250, D: 0.2189, G:0.1839

EPOCH: 31
Iter: 7500, D: 0.4495, G:0.2258

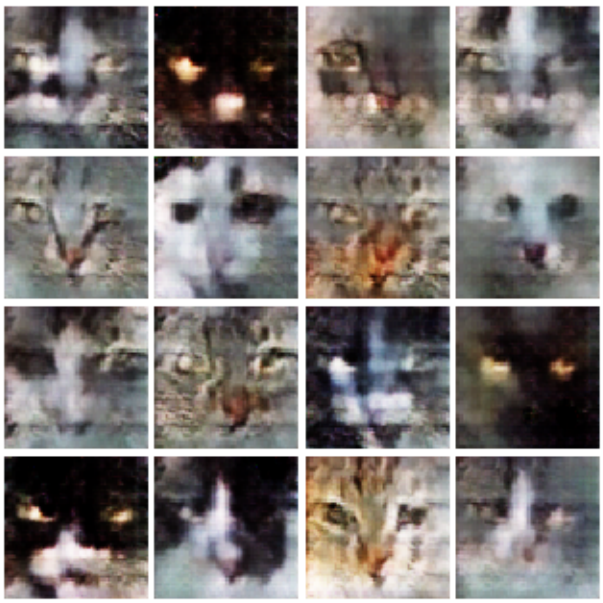
EPOCH: 32
Iter: 7750, D: 0.239, G:0.1689

EPOCH: 33
Iter: 8000, D: 0.1462, G:0.241



EPOCH: 34
Iter: 8250, D: 0.179, G:0.1969

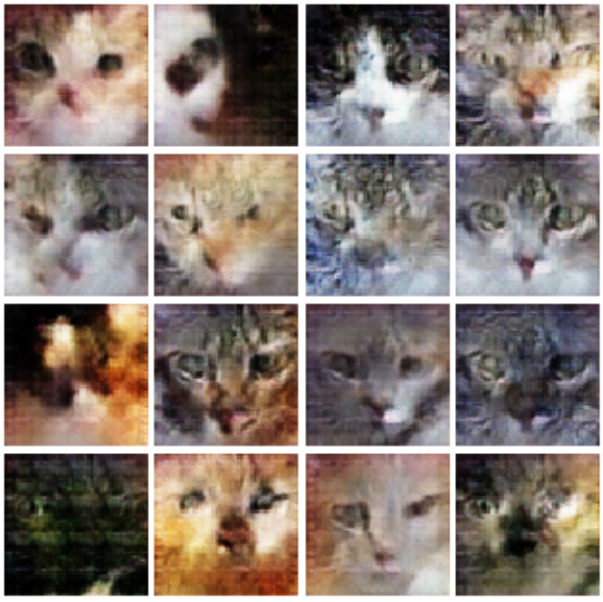
EPOCH: 35
Iter: 8500, D: 0.1802, G:0.4034



EPOCH: 36
Iter: 8750, D: 0.251, G:0.1944

EPOCH: 37
Iter: 9000, D: 0.221, G:0.6671

EPOCH: 38
Iter: 9250, D: 0.2752, G:0.1956



EPOCH: 39
Iter: 9500, D: 0.05867, G:0.2958

```
EPOCH:  40
Iter: 9750, D: 0.1881, G:0.2339
```

```
EPOCH:  41
Iter: 10000, D: 0.1298, G:0.2789
```

EPOCH: 42
Iter: 10250, D: 0.1615, G:0.271

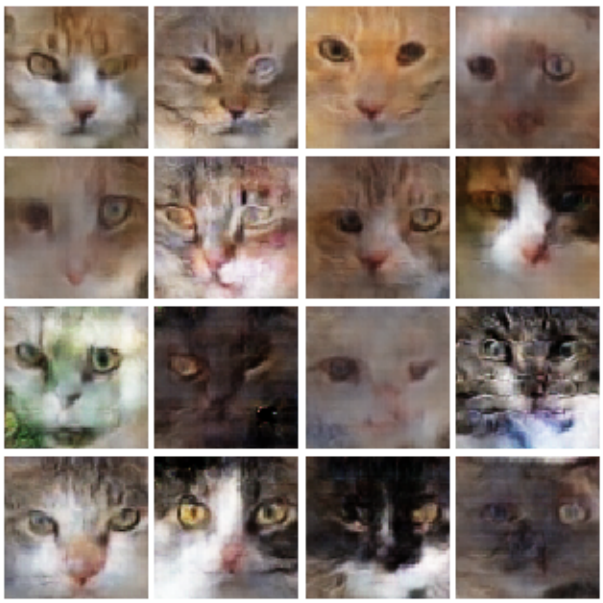
EPOCH: 43
Iter: 10500, D: 0.1891, G:0.4147

EPOCH: 44
Iter: 10750, D: 0.2499, G:0.1897

EPOCH: 45
Iter: 11000, D: 0.2359, G:0.4049

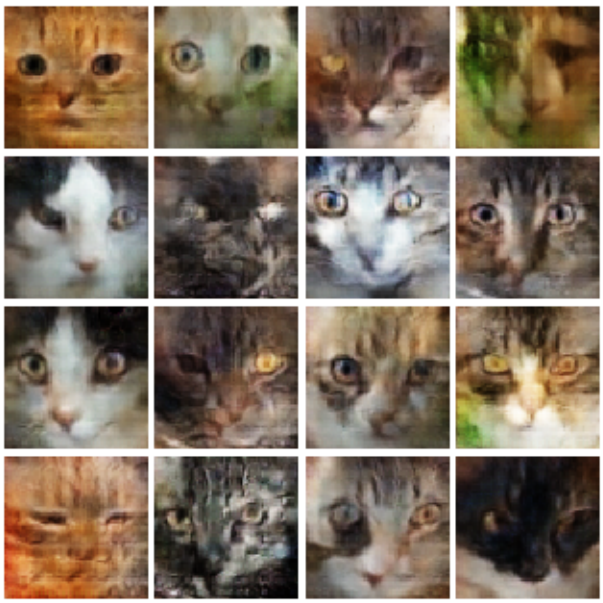
```
EPOCH:  46
Iter: 11250, D: 0.1909, G:0.2361
```

```
EPOCH:  47
Iter: 11500, D: 0.2203, G:0.162
```



EPOCH: 48
Iter: 11750, D: 0.2374, G:0.2051

EPOCH: 49
Iter: 12000, D: 0.1891, G:0.2963



```
EPOCH:  50
Iter: 12250, D: 0.2157, G:0.2054
```

7 Extra Credit 1: WGAN Loss

7.0.1 Train WGAN

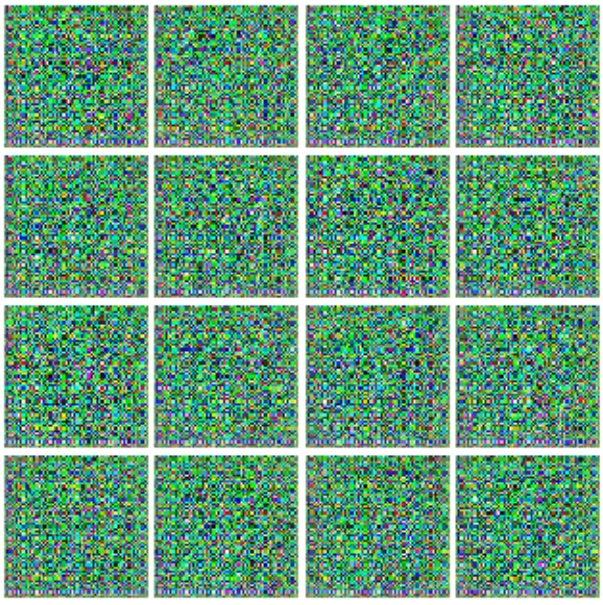
[ ]:
```
D = Discriminator().to(device)
G = Generator(noise_dim=NOISE_DIM).to(device)
```

[ ]:
```
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))
```

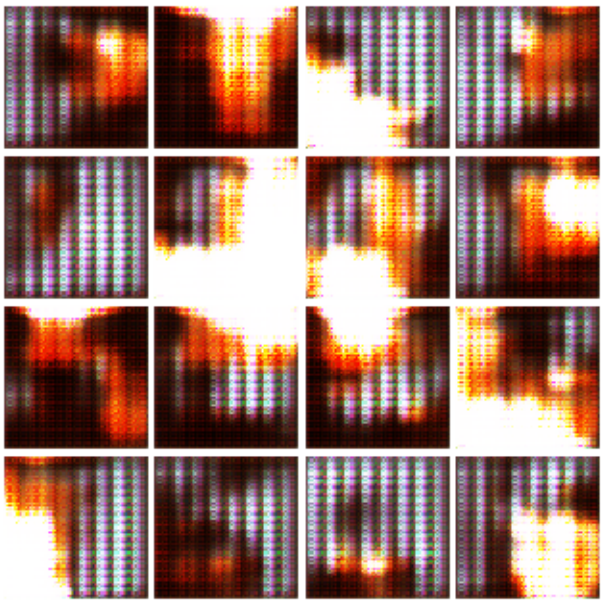
[ ]:
```
from gan.losses import w_discriminator_loss, w_generator_loss
```

[ ]:
```
# w-gan
train(D, G, D_optimizer, G_optimizer, w_discriminator_loss,
          w_generator_loss, num_epochs=NUM_EPOCHS, show_every=250,
          batch_size=batch_size, train_loader=cat_loader_train, device=device)
```

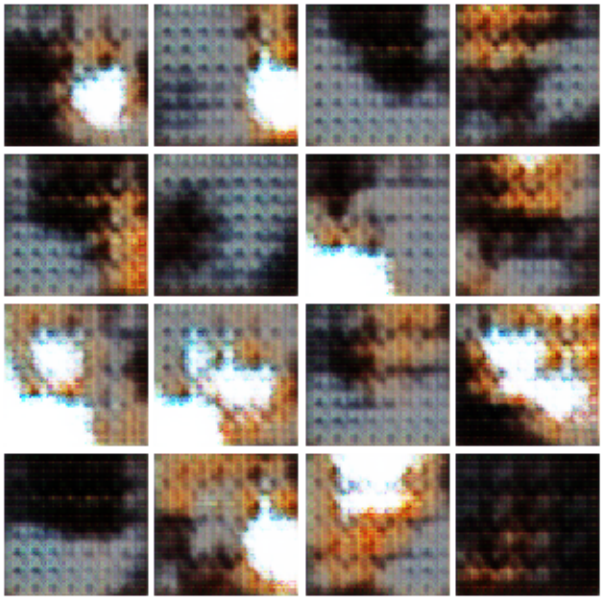
```
EPOCH:  1
Iter: 0, D: 0.0248, G:6.091
```



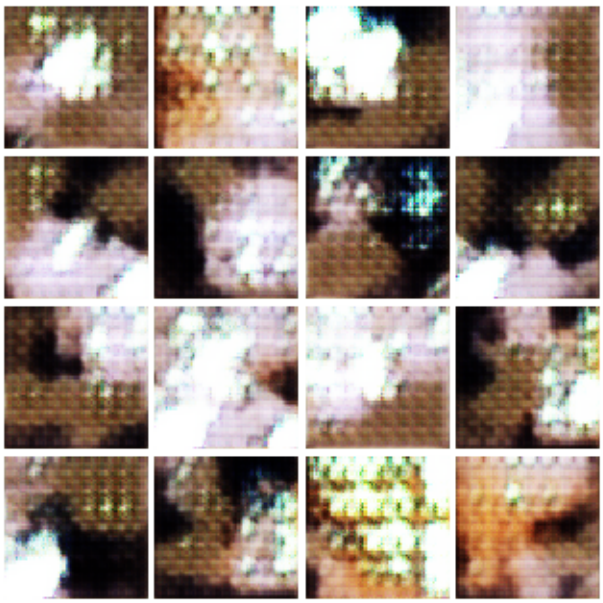
```
EPOCH:  2
Iter: 250, D: -1.148e+03, G:728.5
```



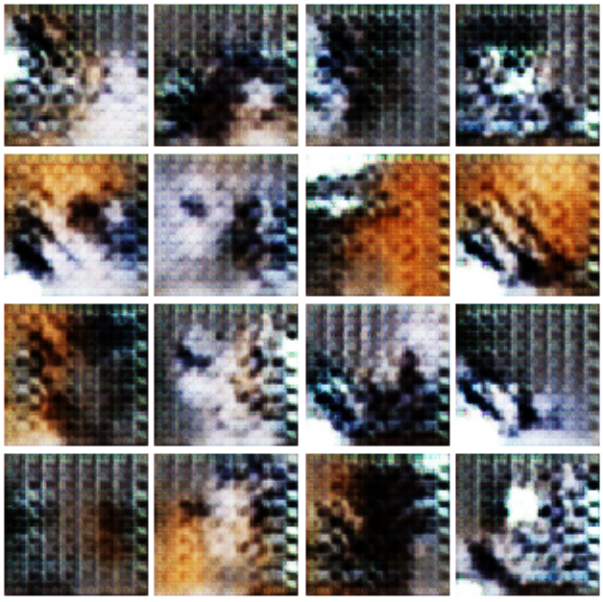
```
EPOCH:  3
Iter: 500, D: -4.819e+03, G:3.041e+03
```



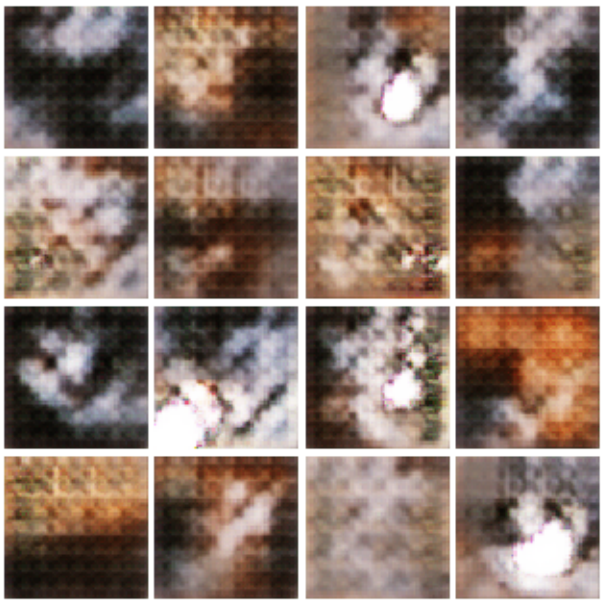
```
EPOCH:  4
Iter: 750, D: -5.589e+03, G:5.14e+03
```



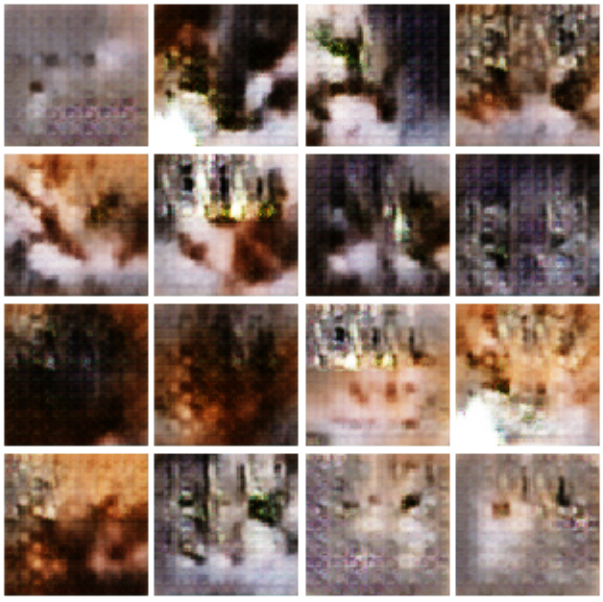
```
EPOCH:  5
Iter: 1000, D: -927.4, G:-7.76e+03
```



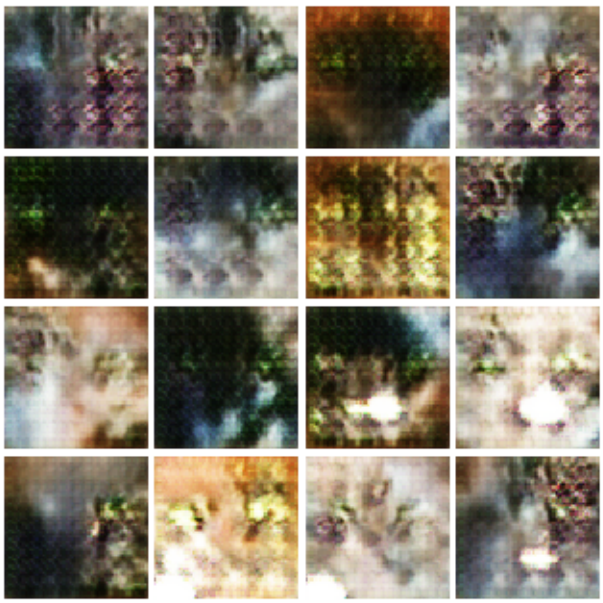
```
EPOCH:  6
Iter: 1250, D: -3.558e+03, G:2.24e+03
```



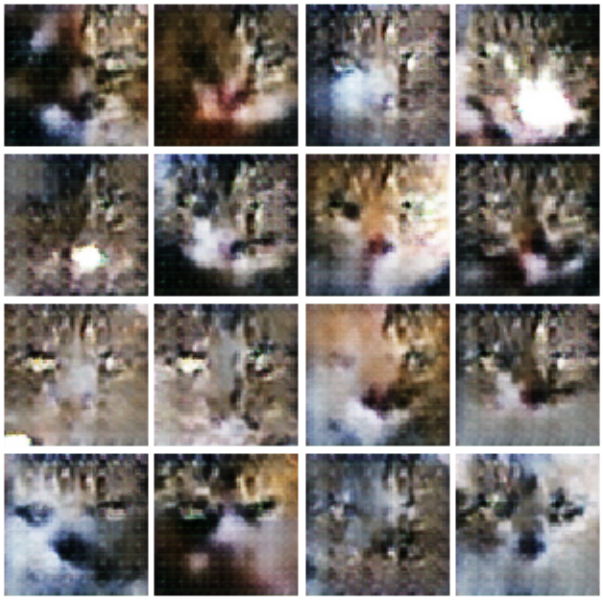
```
EPOCH:  7
Iter: 1500, D: -2.549e+04, G:1.537e+04
```



```
EPOCH:  8
Iter: 1750, D: -7.862e+03, G:-1.492e+03
```



```
EPOCH:  9
Iter: 2000, D: -8.144e+03, G:-414.2
```



```
EPOCH:  10
Iter: 2250, D: -3.693e+04, G:2.31e+04
```

```
EPOCH:  11
Iter: 2500, D: -4.357e+04, G:4.149e+04
```

```
EPOCH:  12
Iter: 2750, D: -6.45e+04, G:5.011e+04
```

EPOCH: 13
Iter: 3000, D: -1.128e+04, G:3.343e+03

EPOCH: 14
Iter: 3250, D: -7.664e+04, G:5.96e+04

```
EPOCH:  15
Iter: 3500, D: -1.028e+05, G:8.698e+04
```

```
EPOCH:  16
Iter: 3750, D: -8.201e+04, G:8.154e+04
```

```
EPOCH:  17
Iter: 4000, D: -7.555e+04, G:5.451e+04
```

```
EPOCH:  18
Iter: 4250, D: -1.335e+05, G:4.106e+04
```

EPOCH: 19
Iter: 4500, D: -9.836e+04, G:1.176e+05

EPOCH: 20
Iter: 4750, D: -2.165e+05, G:-7.68e+04

EPOCH: 21
Iter: 5000, D: -4.391e+05, G:2.159e+05

EPOCH: 22
Iter: 5250, D: -9.723e+04, G:1.565e+05

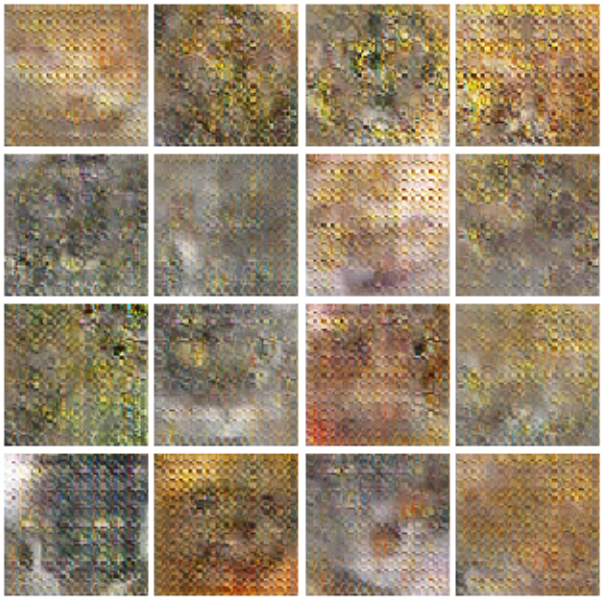
EPOCH: 23
Iter: 5500, D: -1.618e+05, G:1.882e+05

EPOCH: 24
Iter: 5750, D: -2.526e+04, G:-1.874e+05

```
EPOCH:  25
Iter: 6000, D: -1.642e+05, G:2.041e+05
```

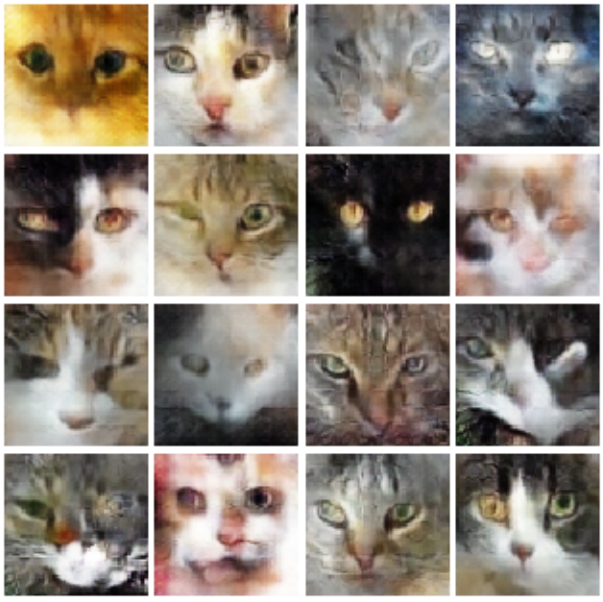
```
EPOCH:  26
Iter: 6250, D: -6.685e+05, G:3.294e+05
```

EPOCH: 27
Iter: 6500, D: -7.455e+05, G:3.499e+05



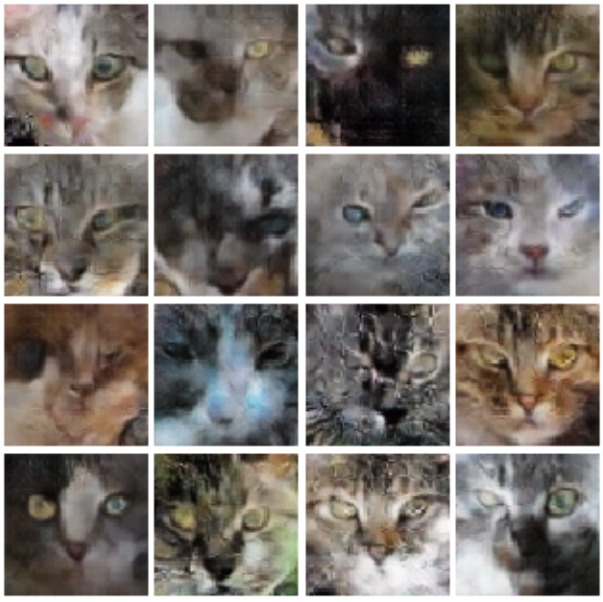
EPOCH: 28
Iter: 6750, D: -3.096e+05, G:2.773e+05

EPOCH: 29
Iter: 7000, D: -1.332e+05, G:3.551e+05



EPOCH: 30
Iter: 7250, D: -3.532e+05, G:1.486e+05

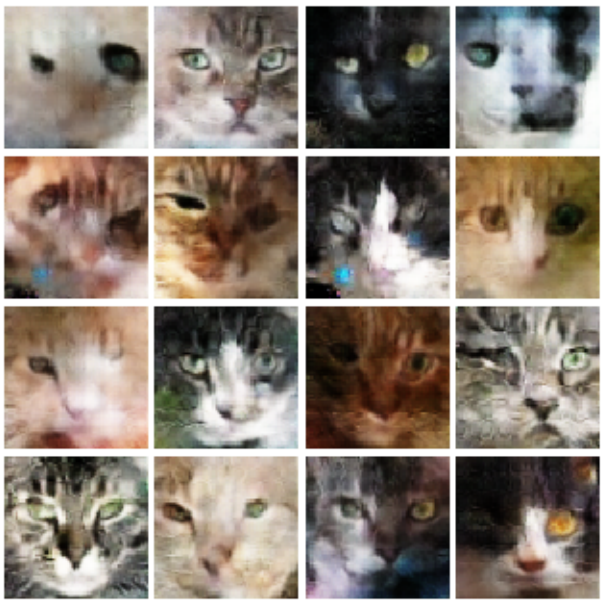
EPOCH: 31
Iter: 7500, D: -1.689e+05, G:2.7e+05



EPOCH: 32
Iter: 7750, D: -1.01e+06, G:4.815e+05

```
EPOCH:  33
Iter: 8000, D: -1.105e+06, G:5.479e+05
```

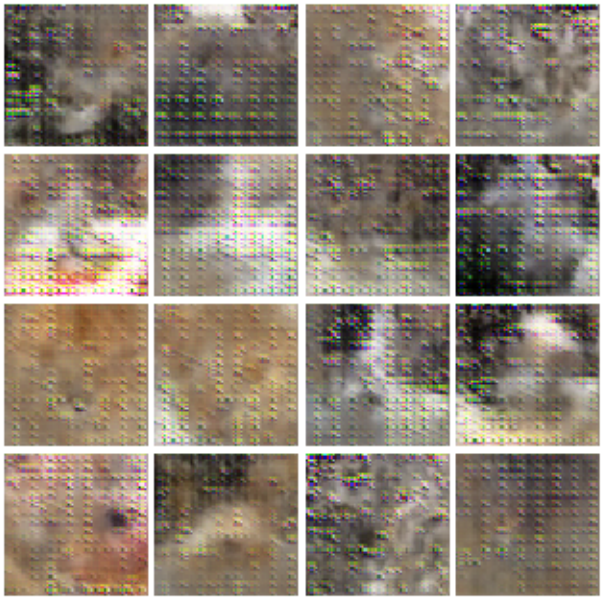
```
EPOCH:  34
Iter: 8250, D: -4.883e+05, G:3.625e+05
```



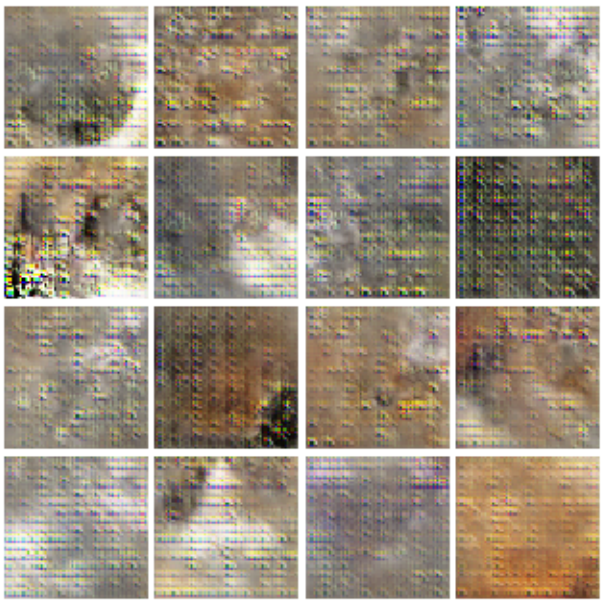
```
EPOCH:  35
Iter: 8500, D: -2.942e+05, G:4.262e+05
```

```
EPOCH:  36
Iter: 8750, D: -1.251e+06, G:6.217e+05
```

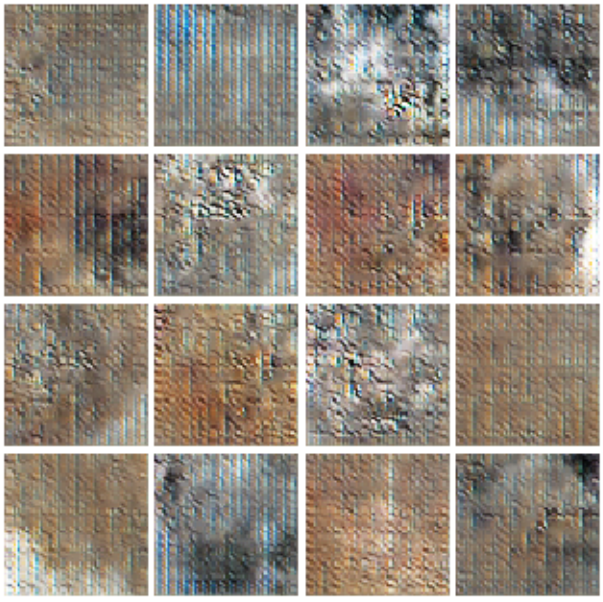
EPOCH: 37
Iter: 9000, D: -1.354e+06, G:6.386e+05



EPOCH: 38
Iter: 9250, D: -1.457e+06, G:6.874e+05

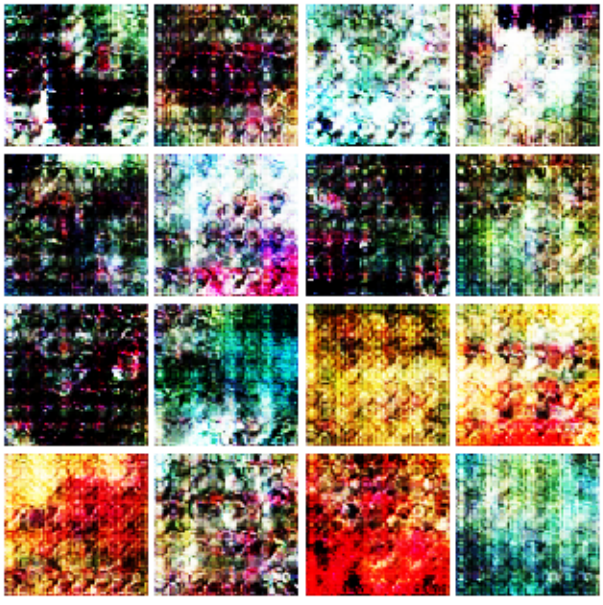


EPOCH: 39
Iter: 9500, D: -1.533e+06, G:7.188e+05

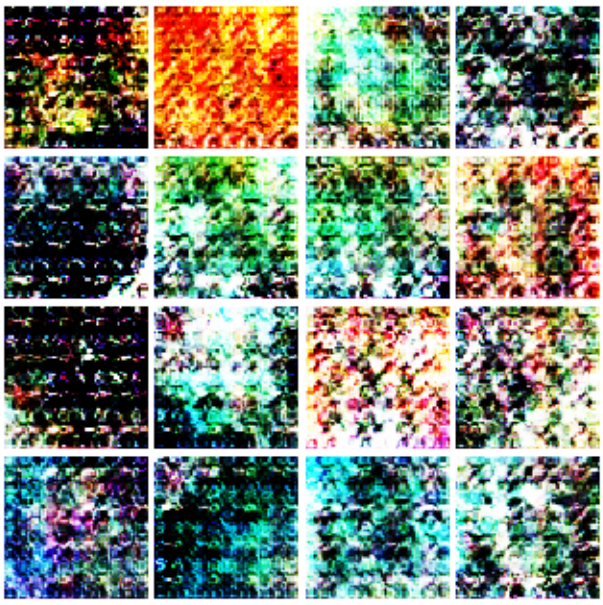


EPOCH: 40
Iter: 9750, D: -1.632e+06, G:7.716e+05

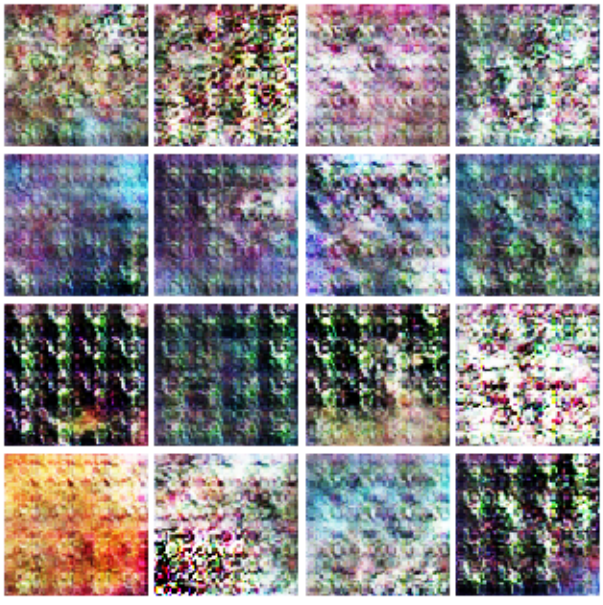
EPOCH: 41
Iter: 10000, D: -1.717e+06, G:8.128e+05



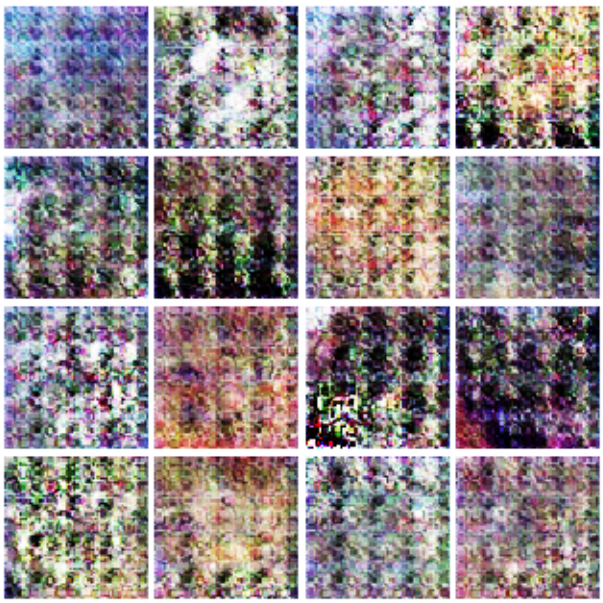
EPOCH: 42
Iter: 10250, D: -1.802e+06, G:8.5e+05



EPOCH: 43
Iter: 10500, D: -1.901e+06, G:8.912e+05



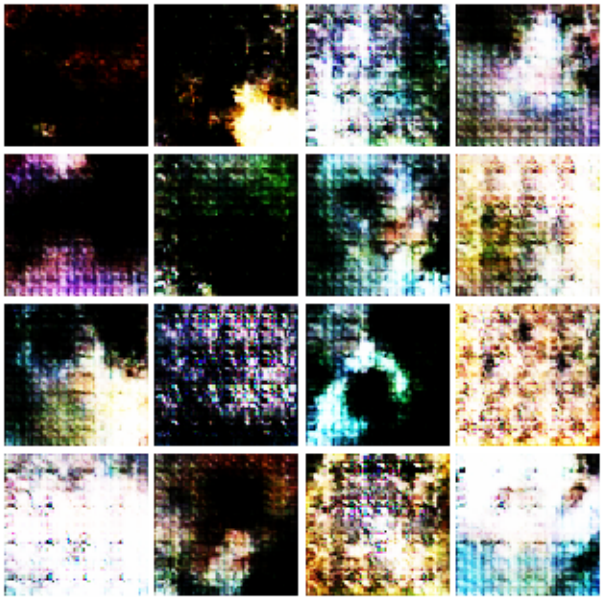
EPOCH: 44
Iter: 10750, D: -1.992e+06, G:9.374e+05



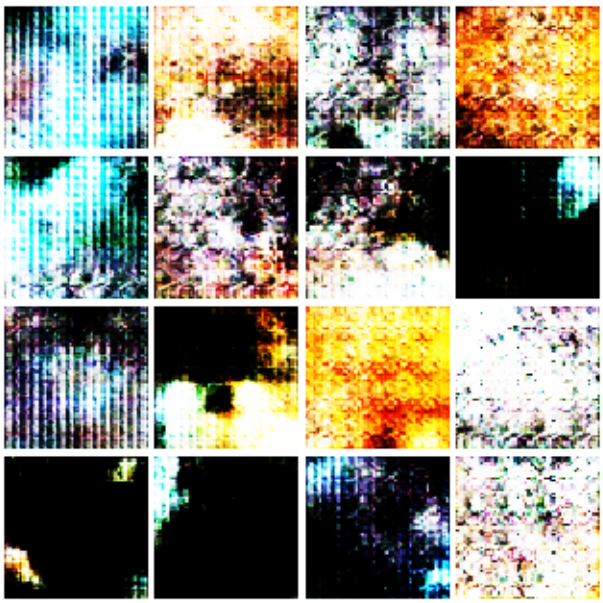
```
EPOCH:  45
Iter: 11000, D: -7.083e+04, G:-5.667e+05
```

```
EPOCH:  46
Iter: 11250, D: -2.11e+06, G:1.011e+06
```

```
EPOCH:  47
Iter: 11500, D: -2.211e+06, G:1.041e+06
```



```
EPOCH:  48
Iter: 11750, D: -2.303e+06, G:1.081e+06
```



```
EPOCH:  49
Iter: 12000, D: -2.156e+05, G:1.669e+04
```

```
EPOCH:  50
Iter: 12250, D: 9.797e+04, G:-6.357e+05
```

8 Extra Credit 2: Spectral Normalization

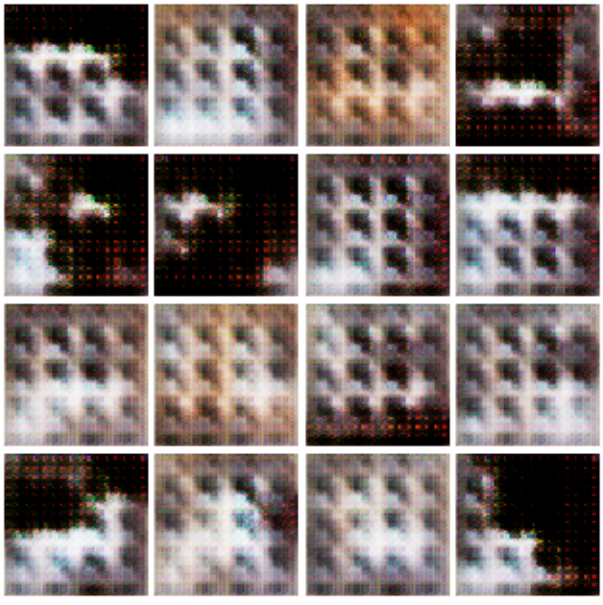
[ ]:
```
D = Discriminator().to(device)
G = Generator(noise_dim=NOISE_DIM).to(device)
```

[ ]:
```
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))
```

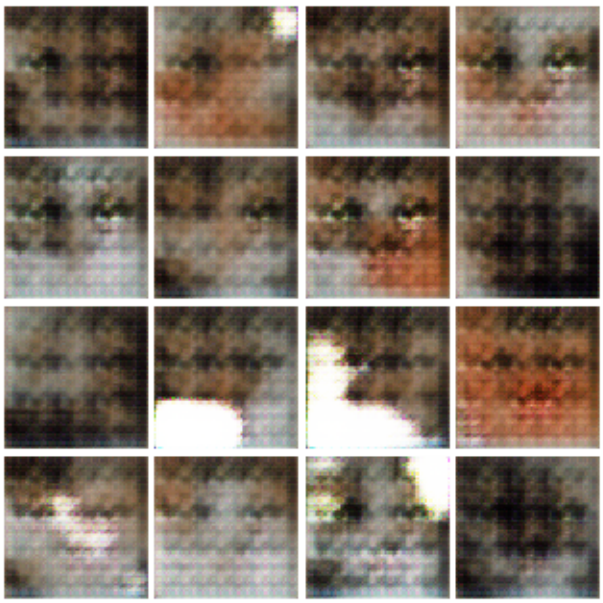
[ ]:
```
train(D, G, D_optimizer, G_optimizer, discriminator_loss,
          generator_loss, num_epochs=NUM_EPOCHS, show_every=250,
          batch_size=batch_size, train_loader=cat_loader_train, device=device)
```

```
EPOCH:  1
Iter: 0, D: 1.404, G:4.587
```

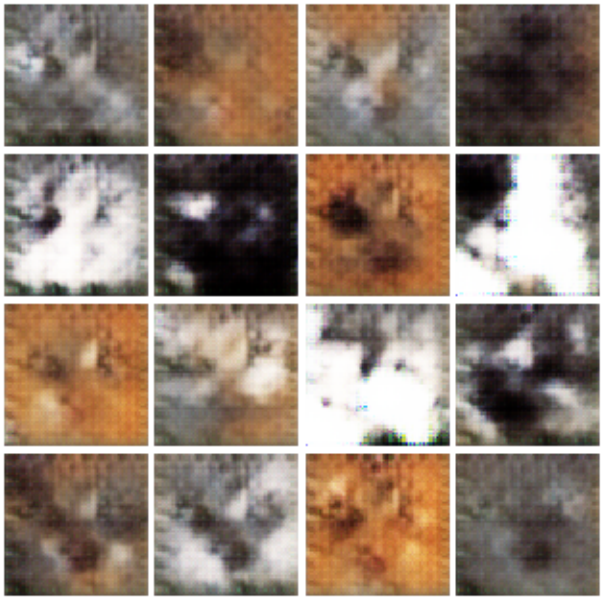
```
EPOCH:  2
Iter: 250, D: 1.468, G:3.173
```



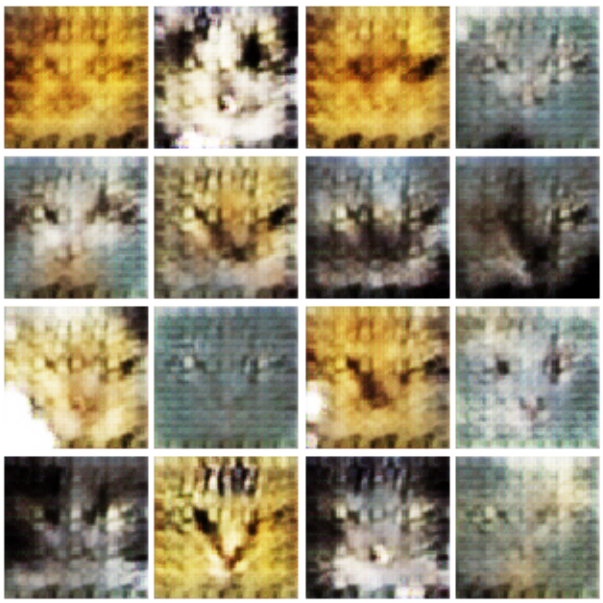
```
EPOCH:  3
Iter: 500, D: 1.285, G:2.781
```



```
EPOCH:  4
Iter: 750, D: 1.686, G:5.162
```



```
EPOCH:  5
Iter: 1000, D: 1.196, G:1.126
```



```
EPOCH:  6
Iter: 1250, D: 1.032, G:1.983
```

```
EPOCH:  7
Iter: 1500, D: 1.266, G:1.015
```

EPOCH: 8
Iter: 1750, D: 1.547, G:1.839

EPOCH: 9
Iter: 2000, D: 0.9348, G:1.46

```
EPOCH:  10
Iter: 2250, D: 1.001, G:1.955
```

```
EPOCH:  11
Iter: 2500, D: 0.738, G:2.569
```

EPOCH: 12
Iter: 2750, D: 1.719, G:1.755

EPOCH: 13
Iter: 3000, D: 0.4123, G:2.399

EPOCH: 14
Iter: 3250, D: 0.7801, G:3.544

EPOCH: 15
Iter: 3500, D: 0.6437, G:2.187

```
EPOCH:  16
Iter: 3750, D: 0.5209, G:3.502
```

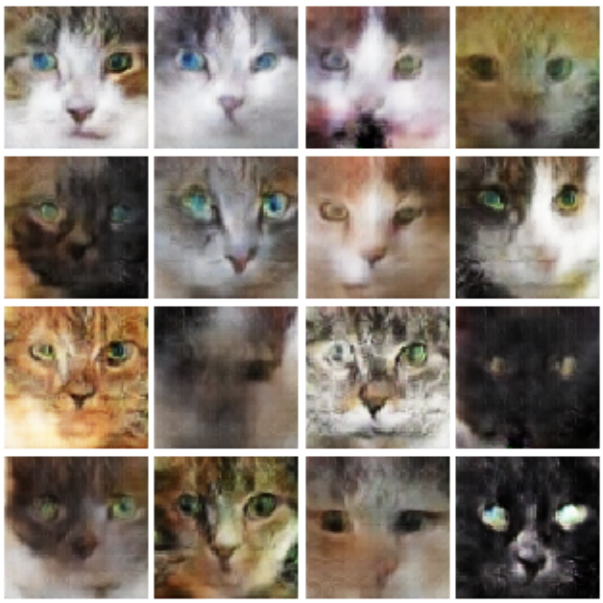
```
EPOCH:  17
Iter: 4000, D: 0.3493, G:3.831
```

```
EPOCH:  18
Iter: 4250, D: 0.3992, G:4.406
```

```
EPOCH:  19
Iter: 4500, D: 0.4835, G:3.697
```

```
EPOCH:  20
Iter: 4750, D: 0.6429, G:2.772
```

```
EPOCH:  21
Iter: 5000, D: 0.6385, G:6.887
```



EPOCH: 22
Iter: 5250, D: 0.246, G:3.861

EPOCH: 23
Iter: 5500, D: 0.3747, G:3.706

```
EPOCH:  24
Iter: 5750, D: 0.308, G:3.607
```

```
EPOCH:  25
Iter: 6000, D: 0.1213, G:4.931
```

```
EPOCH:  26
Iter: 6250, D: 0.505, G:8.478
```

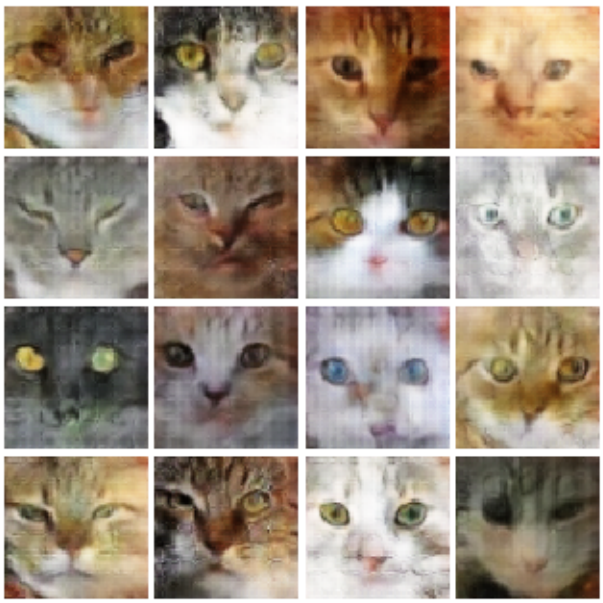
```
EPOCH:  27
Iter: 6500, D: 0.3444, G:3.813
```

```
EPOCH:  28
Iter: 6750, D: 0.3579, G:3.171
```

```
EPOCH:  29
Iter: 7000, D: 1.789, G:1.996
```

EPOCH: 30
Iter: 7250, D: 0.2094, G:4.823

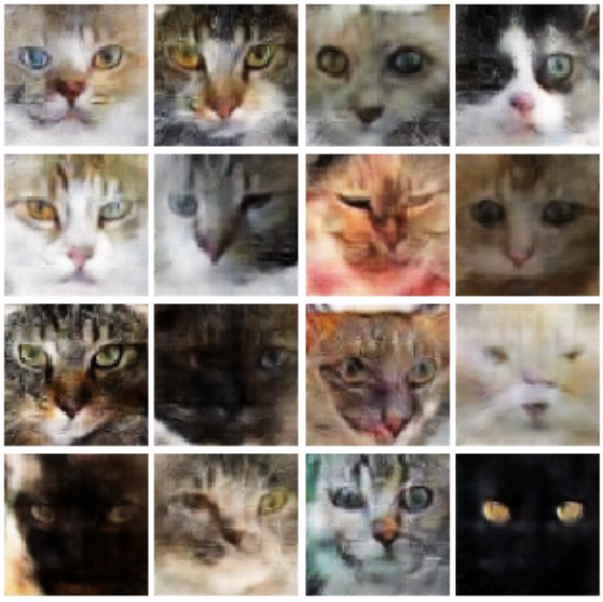
EPOCH: 31
Iter: 7500, D: 0.2543, G:4.393



EPOCH: 32
Iter: 7750, D: 1.084, G:7.711

EPOCH: 33
Iter: 8000, D: 0.1085, G:5.36

EPOCH: 34
Iter: 8250, D: 0.2684, G:3.894



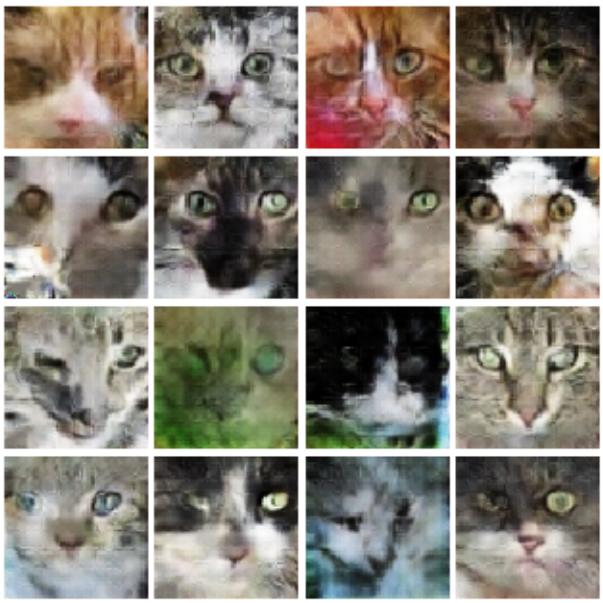
EPOCH: 35
Iter: 8500, D: 0.5873, G:1.972

```
EPOCH:  36
Iter: 8750, D: 0.2957, G:1.519
```

```
EPOCH:  37
Iter: 9000, D: 1.058, G:11.42
```

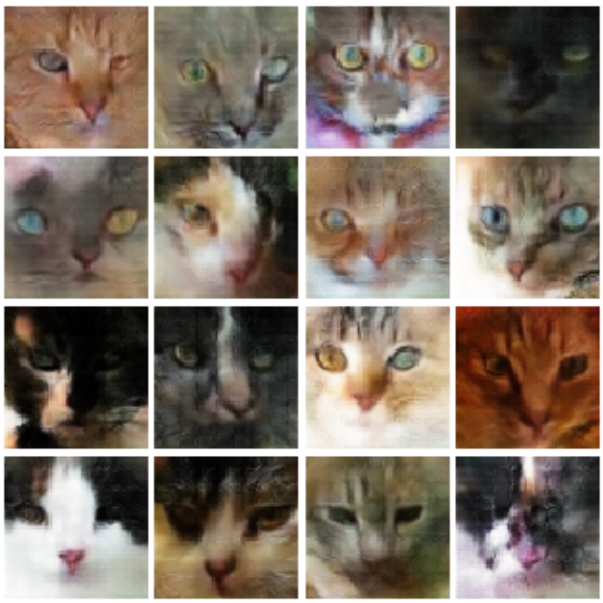
```
EPOCH:  38
Iter: 9250, D: 0.4308, G:4.428
```

```
EPOCH:  39
Iter: 9500, D: 0.3184, G:5.904
```



```
EPOCH:  40
Iter: 9750, D: 0.3222, G:6.536
```

```
EPOCH:  41
Iter: 10000, D: 0.3402, G:5.17
```

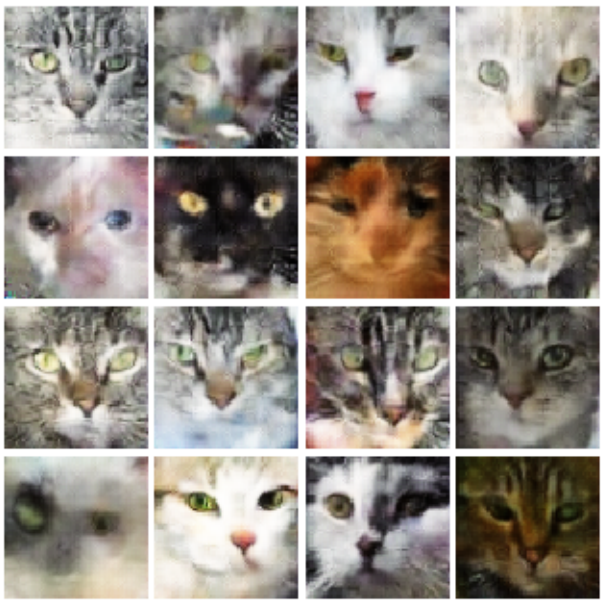


EPOCH: 42
Iter: 10250, D: 0.1742, G:5.425

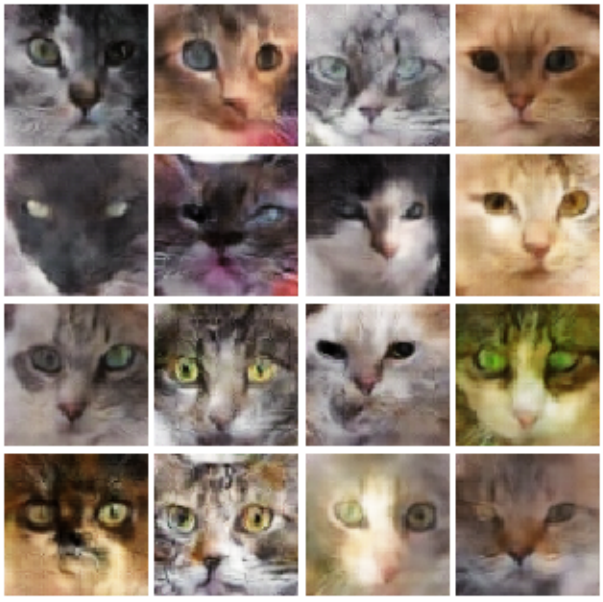
EPOCH: 43
Iter: 10500, D: 0.148, G:5.973

```
EPOCH:  44
Iter: 10750, D: 0.1794, G:6.006
```

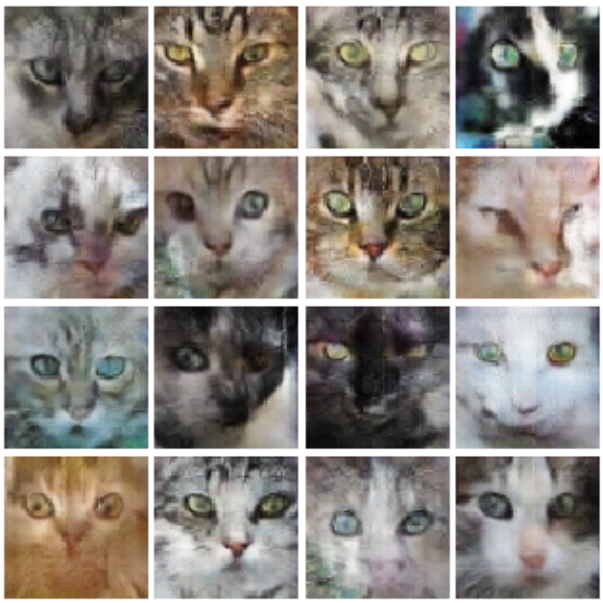
```
EPOCH:  45
Iter: 11000, D: 0.3815, G:4.111
```



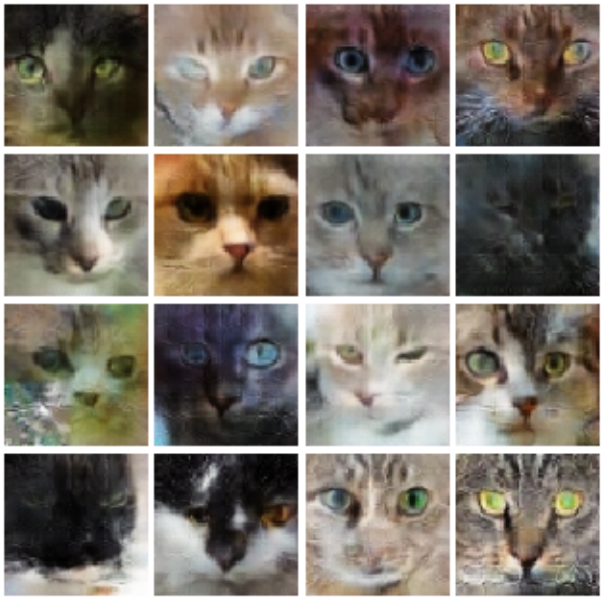
```
EPOCH:  46
Iter: 11250, D: 0.3606, G:4.85
```



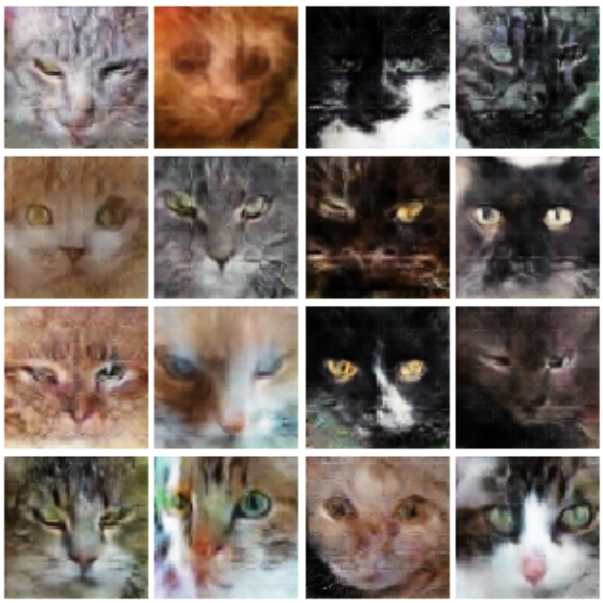
```
EPOCH:  47
Iter: 11500, D: 0.2876, G:4.475
```



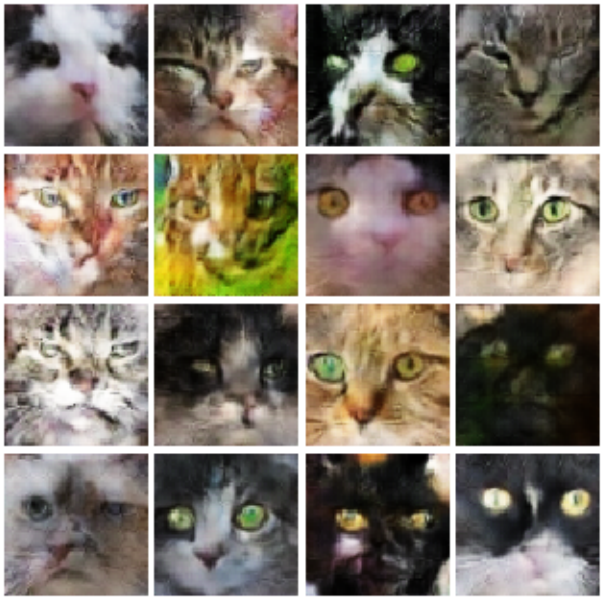
```
EPOCH:  48
Iter: 11750, D: 0.1511, G:5.33
```



```
EPOCH:  49
Iter: 12000, D: 0.2331, G:5.303
```



```
EPOCH:  50
Iter: 12250, D: 0.4051, G:4.777
```



9 Extra Credit 3: Another Dataset

[ ]:
```
### Load Dog Dataset ###
batch_size = 64
imsize = 64
dog_root = './dog'

dog_train = ImageFolder(root=dog_root, transform=transforms.Compose([
      transforms.ToTensor(),

      # Example use of RandomCrop:
      transforms.Resize(int(1.15 * imsize)),
      transforms.RandomCrop(imsize),
]))

dog_loader_train = DataLoader(dog_train, batch_size=batch_size, drop_last=True)
```

[ ]:
```
### Visualize ###
from gan.utils import show_images

try:
  imgs = next(iter(dog_loader_train))[0].numpy().squeeze()
except:
  imgs = dog_loader_train.__iter__().next()[0].numpy().squeeze()

show_images(imgs, color=True)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

9.0.1 Training

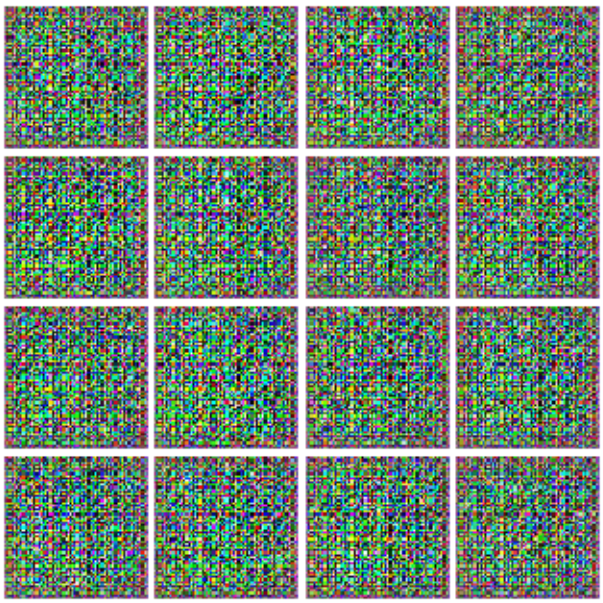
[ ]:
```
NOISE_DIM = 100
NUM_EPOCHS = 200
learning_rate = 0.001
```

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

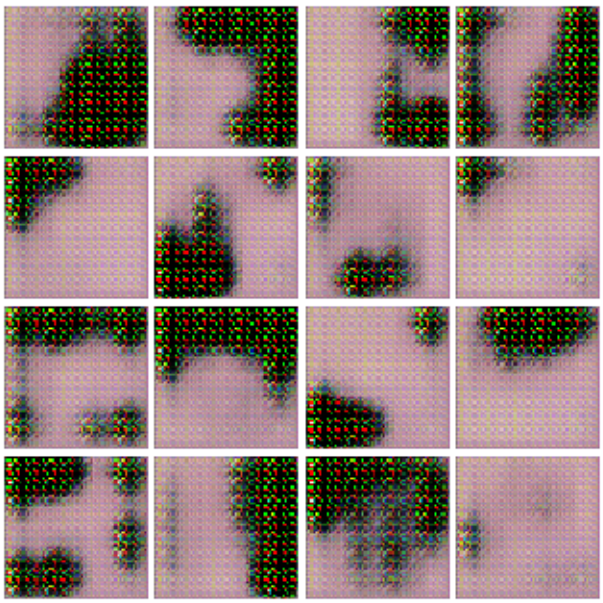
[ ]:
```
D_optimizer = torch.optim.Adam(D.parameters(), lr=learning_rate, betas = (0.5,␣
↪0.999))
G_optimizer = torch.optim.Adam(G.parameters(), lr=learning_rate, betas = (0.5,␣
↪0.999))
```

[ ]:
```
train(D, G, D_optimizer, G_optimizer, discriminator_loss,
          generator_loss, num_epochs=NUM_EPOCHS, show_every=50,
          batch_size=batch_size, train_loader=dog_loader_train, device=device)
```

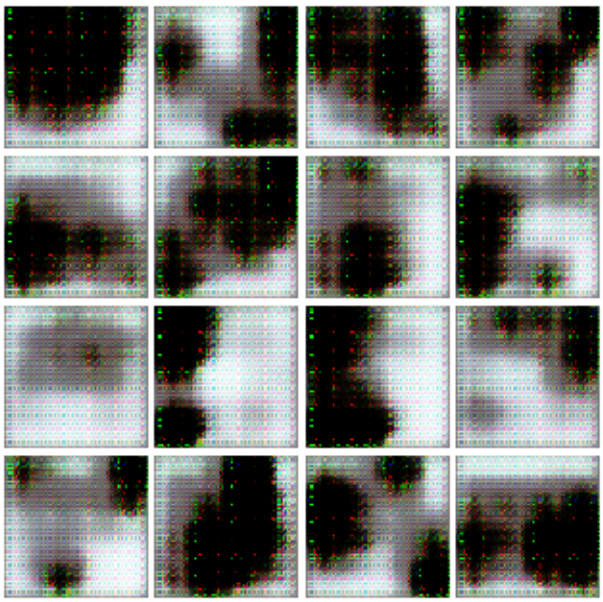
```
EPOCH:  1
Iter: 0, D: 1.455, G:8.388
```



```
EPOCH:  2
EPOCH:  3
Iter: 50, D: 1.382, G:1.601
```

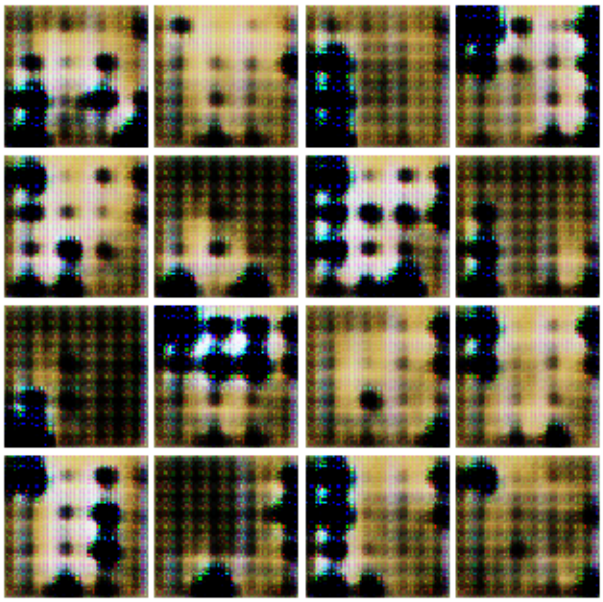


```
EPOCH:  4
EPOCH:  5
Iter: 100, D: 0.8467, G:5.018
```

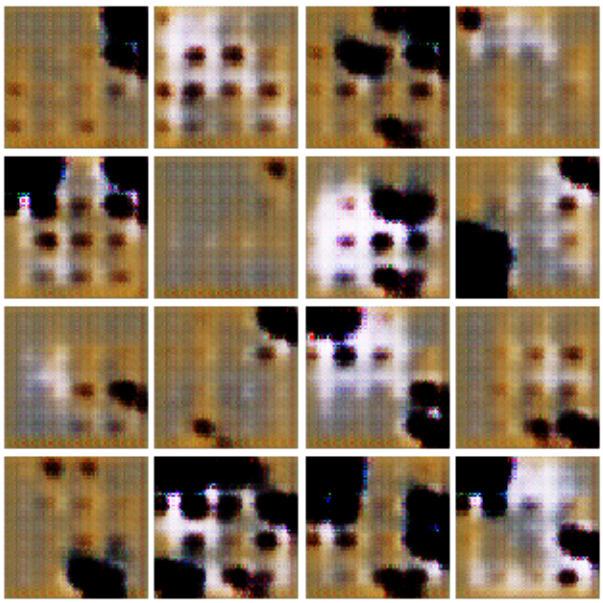


```
EPOCH:  6
```

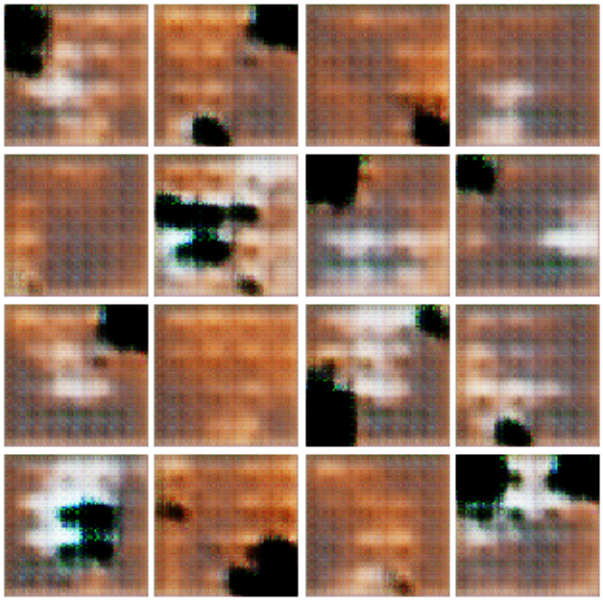
```
EPOCH:  7
Iter: 150, D: 0.9229, G:1.974
```



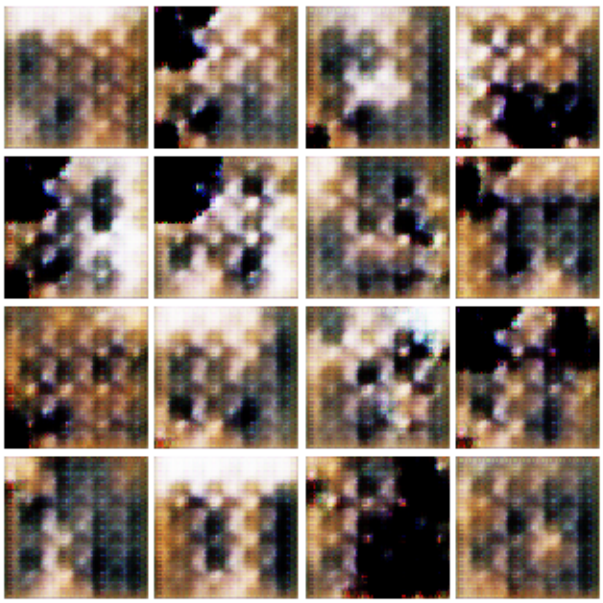
```
EPOCH:  8
EPOCH:  9
EPOCH:  10
Iter: 200, D: 0.9069, G:1.367
```



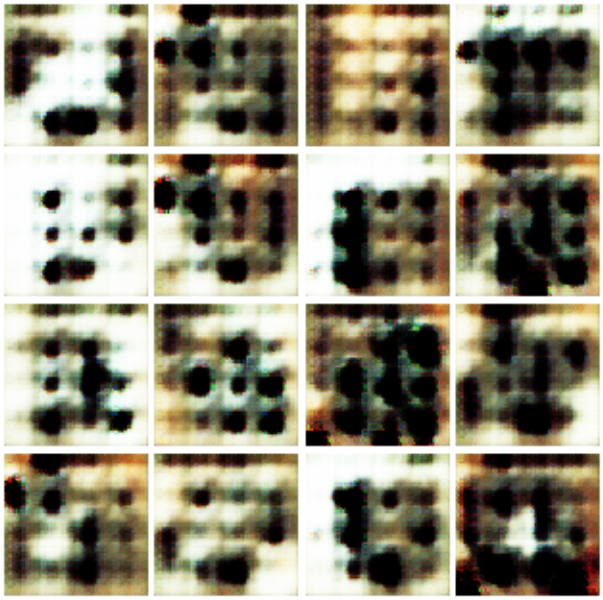
```
EPOCH:  11
EPOCH:  12
Iter: 250, D: 0.9697, G:2.671
```



```
EPOCH:  13
EPOCH:  14
Iter: 300, D: 0.2901, G:2.823
```

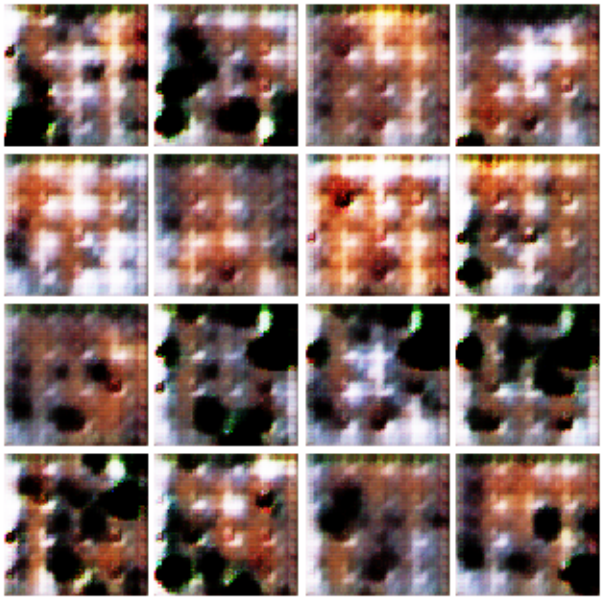


```
EPOCH:  15
EPOCH:  16
Iter: 350, D: 0.6936, G:3.675
```

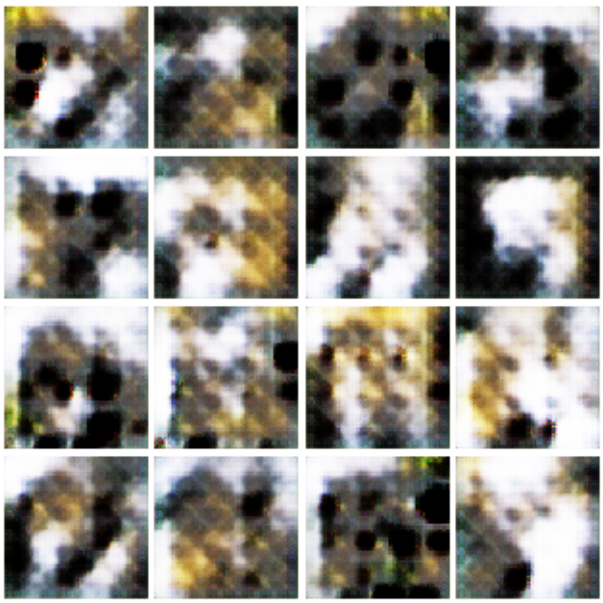


```
EPOCH:  17
```

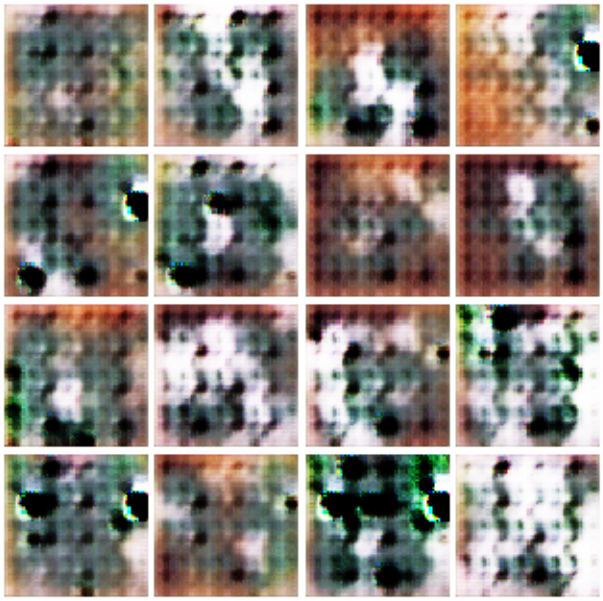
```
EPOCH:  18
EPOCH:  19
Iter: 400, D: 0.8892, G:4.959
```



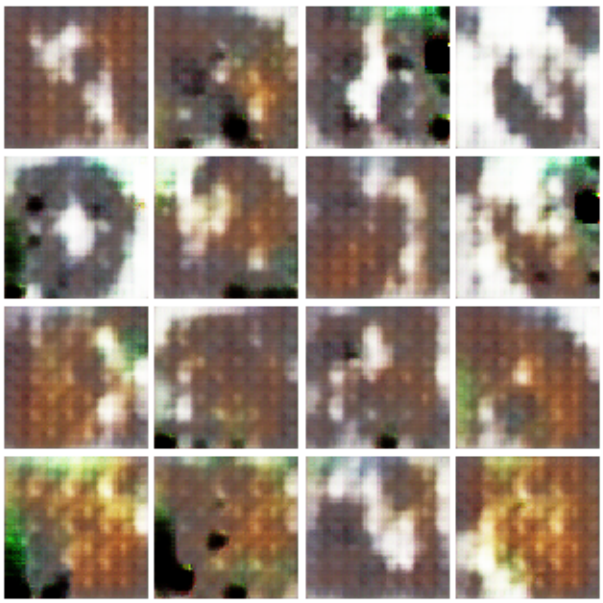
```
EPOCH:  20
EPOCH:  21
Iter: 450, D: 0.8315, G:1.899
```



```
EPOCH:  22
EPOCH:  23
Iter: 500, D: 0.5564, G:4.372
```



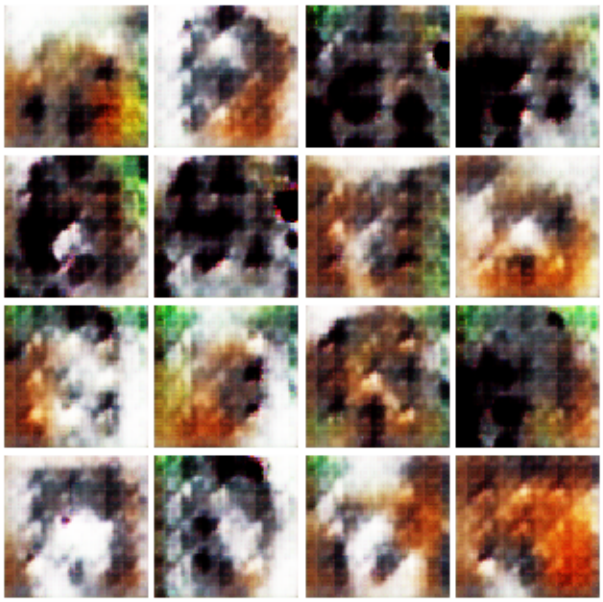
```
EPOCH:  24
EPOCH:  25
EPOCH:  26
Iter: 550, D: 0.8471, G:2.774
```



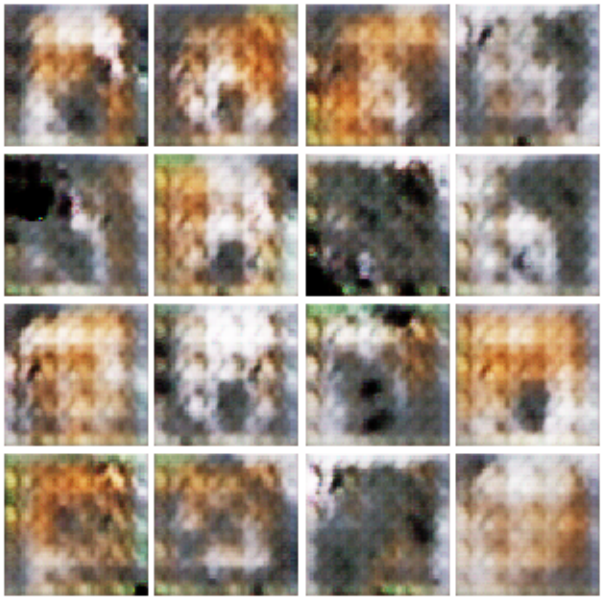
```
EPOCH:  27
EPOCH:  28
Iter: 600, D: 0.4462, G:3.297
```

```
EPOCH:  29
```

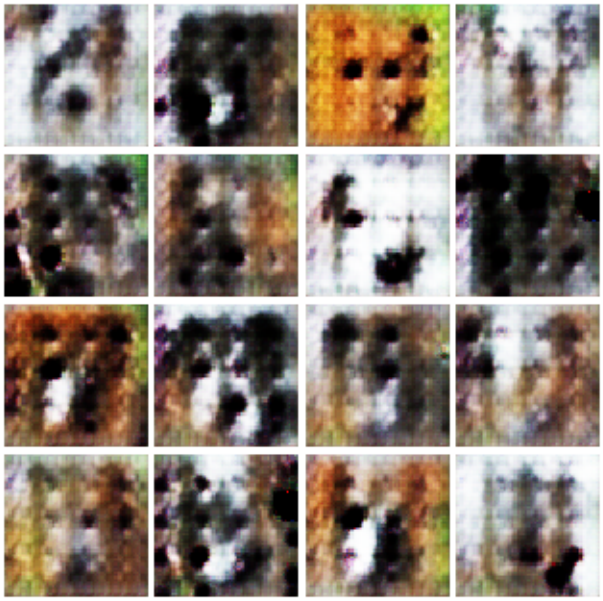
```
EPOCH:  30
Iter: 650, D: 0.5287, G:2.792
```



```
EPOCH:  31
EPOCH:  32
Iter: 700, D: 1.807, G:3.081
```

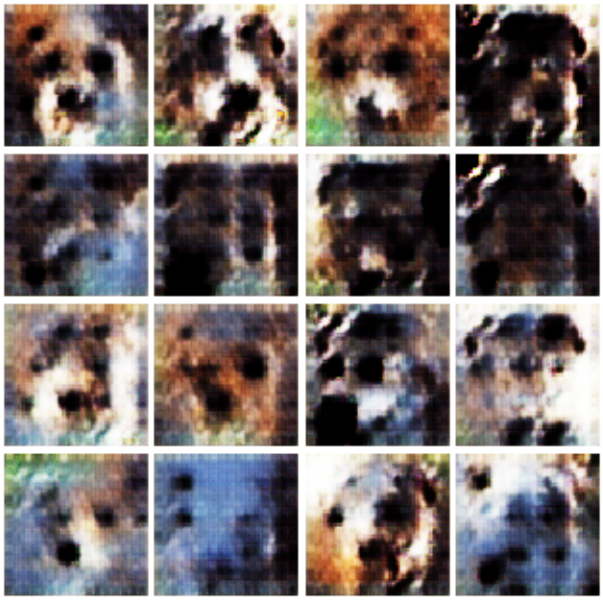


```
EPOCH:  33
EPOCH:  34
EPOCH:  35
Iter: 750, D: 0.7818, G:1.552
```



```
EPOCH:  36
EPOCH:  37
Iter: 800, D: 0.4931, G:4.509
```

```
EPOCH:  38
EPOCH:  39
Iter: 850, D: 0.8428, G:3.586
```



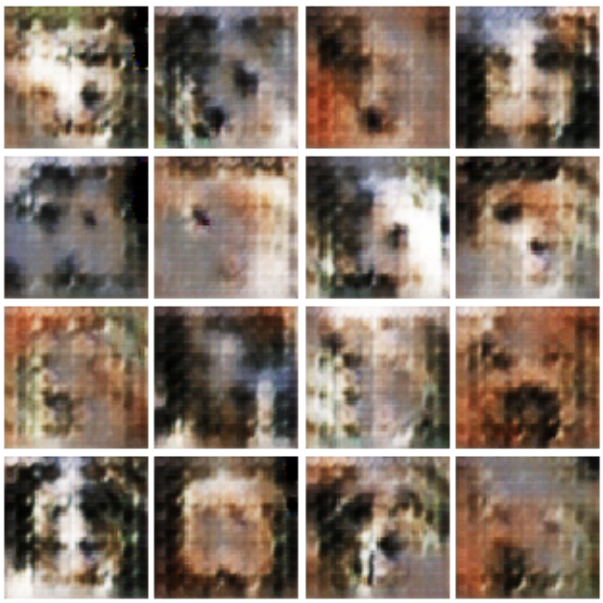
```
EPOCH:  40
```

```
EPOCH:  41
Iter: 900, D: 1.053, G:0.9006
```

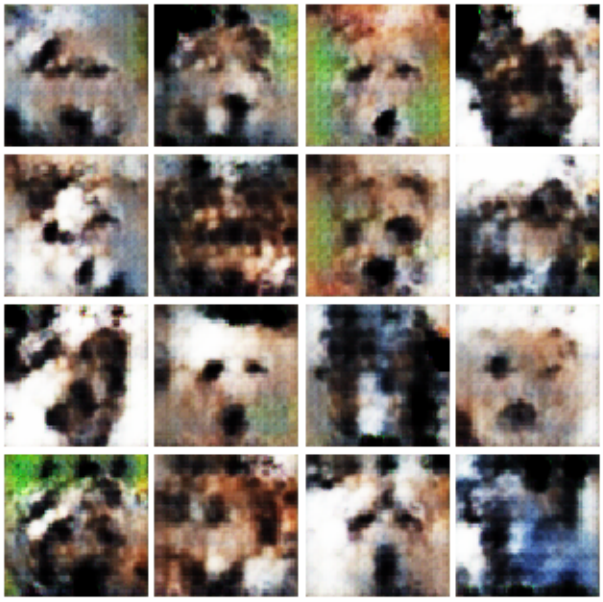
```
EPOCH:  42
EPOCH:  43
EPOCH:  44
Iter: 950, D: 0.4252, G:3.805
```

```
EPOCH:  45
EPOCH:  46
Iter: 1000, D: 1.081, G:2.533
```

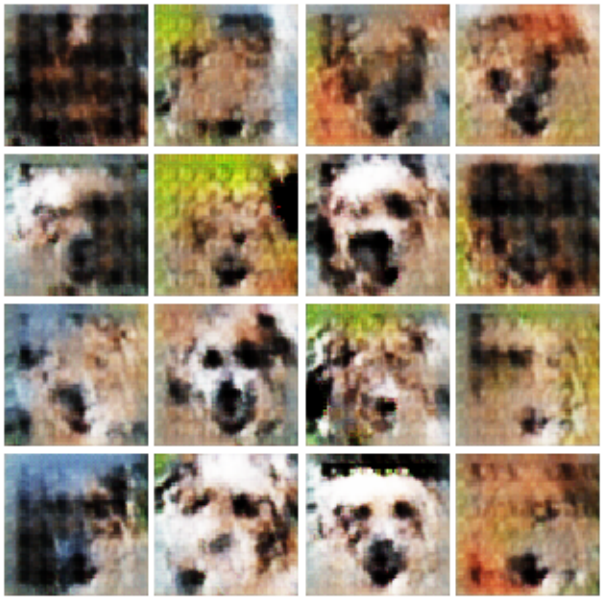
```
EPOCH:  47
EPOCH:  48
Iter: 1050, D: 1.886, G:4.949
```



```
EPOCH:  49
EPOCH:  50
EPOCH:  51
Iter: 1100, D: 0.6, G:2.044
```



```
EPOCH:  52
EPOCH:  53
Iter: 1150, D: 0.4634, G:1.609
```



```
EPOCH:  54
EPOCH:  55
Iter: 1200, D: 0.6041, G:4.063
```

```
EPOCH:  56
EPOCH:  57
Iter: 1250, D: 0.3431, G:1.242
```

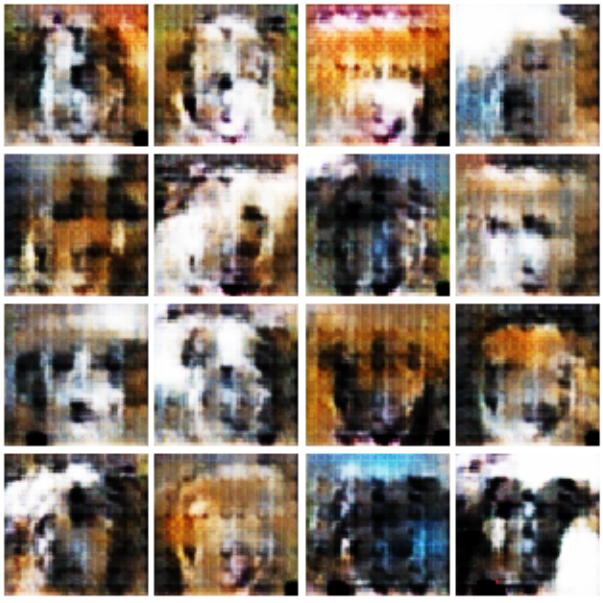
```
EPOCH:  58
EPOCH:  59
EPOCH:  60
Iter: 1300, D: 0.603, G:0.9618
```

```
EPOCH:  61
EPOCH:  62
Iter: 1350, D: 0.3746, G:1.257
```

```
EPOCH:  63
```

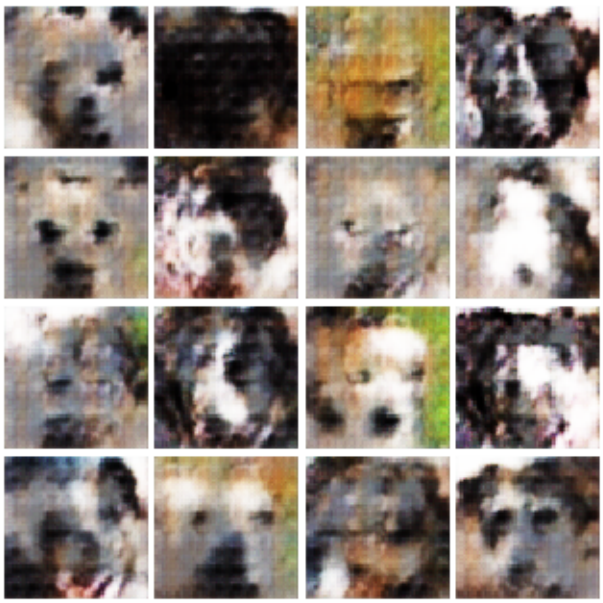
```
EPOCH:  64
Iter: 1400, D: 0.8191, G:3.073
```

```
EPOCH:  65
EPOCH:  66
Iter: 1450, D: 0.6467, G:2.953
```



```
EPOCH:  67
EPOCH:  68
EPOCH:  69
Iter: 1500, D: 0.9179, G:2.315
```

```
EPOCH:  70
EPOCH:  71
Iter: 1550, D: 0.8369, G:1.526
```



```
EPOCH:  72
EPOCH:  73
Iter: 1600, D: 0.4643, G:2.914
```

```
EPOCH:  74
```

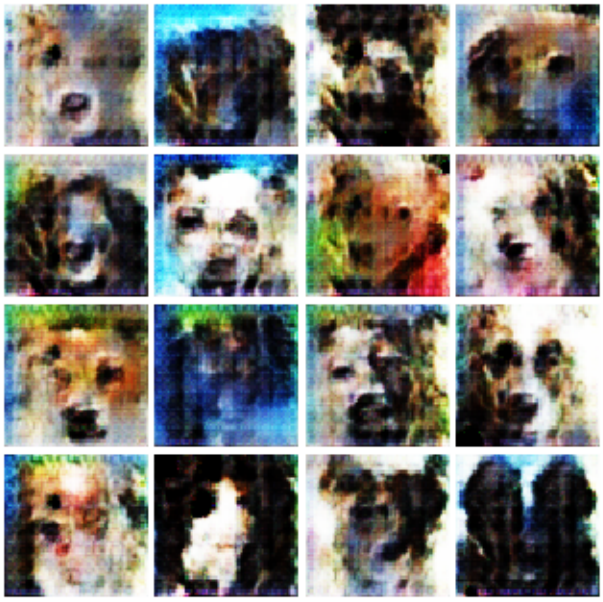
```
EPOCH:  75
EPOCH:  76
Iter: 1650, D: 0.6531, G:2.01
```

```
EPOCH:  77
EPOCH:  78
Iter: 1700, D: 0.5539, G:3.16
```

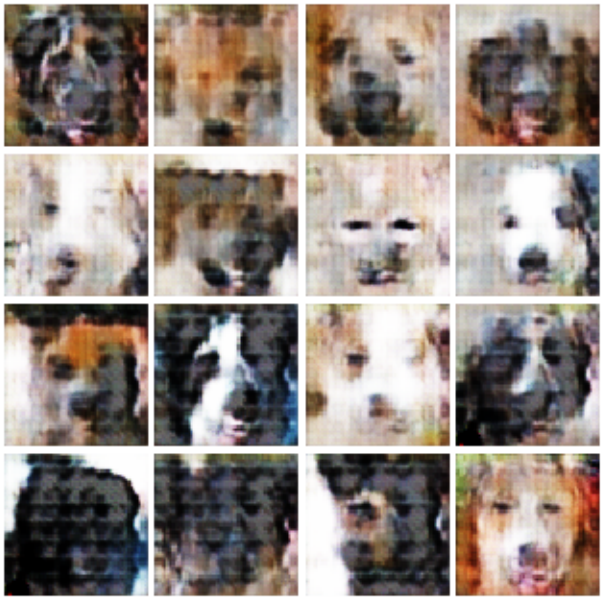
```
EPOCH:  79
EPOCH:  80
Iter: 1750, D: 0.3326, G:2.7
```

```
EPOCH:  81
EPOCH:  82
Iter: 1800, D: 0.408, G:3.018
```

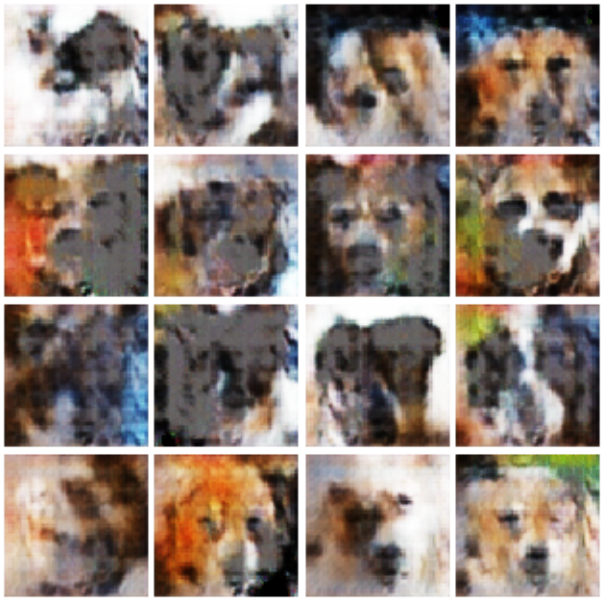
```
EPOCH:  83
EPOCH:  84
EPOCH:  85
Iter: 1850, D: 0.7723, G:6.317
```



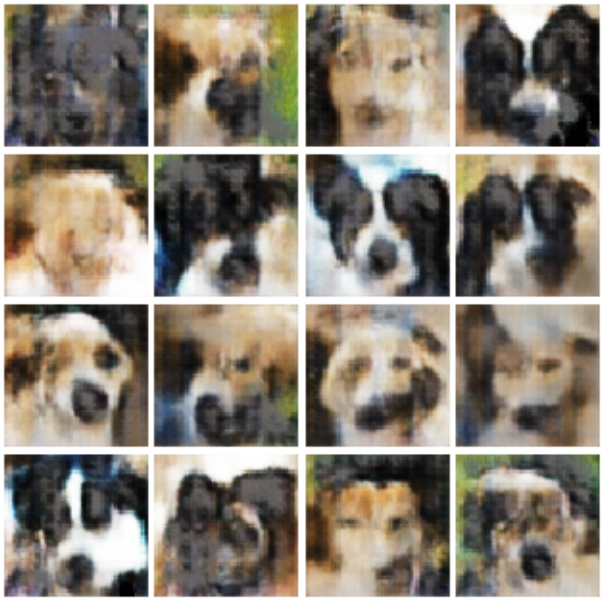
```
EPOCH:  86
EPOCH:  87
Iter: 1900, D: 0.5553, G:2.233
```



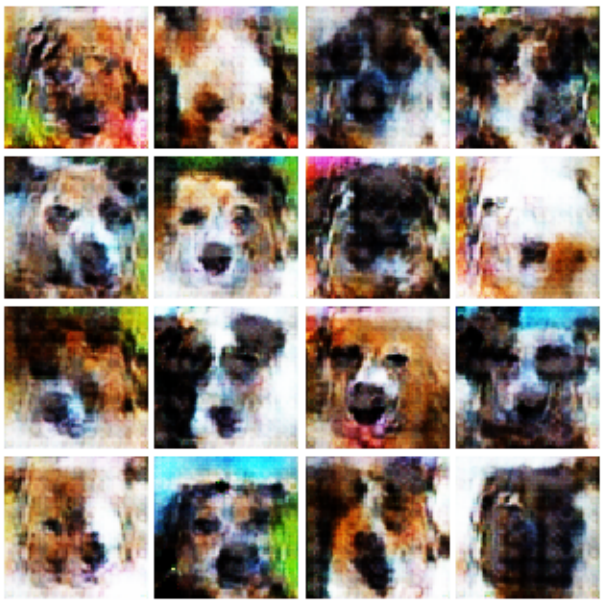
```
EPOCH:  88
EPOCH:  89
Iter: 1950, D: 0.7804, G:2.607
```



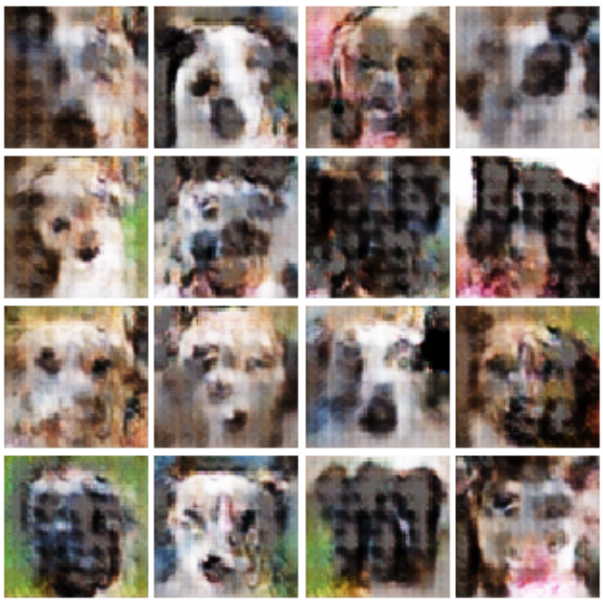
```
EPOCH:  90
EPOCH:  91
Iter: 2000, D: 0.6392, G:2.428
```



```
EPOCH:  92
EPOCH:  93
EPOCH:  94
Iter: 2050, D: 1.206, G:6.978
```



```
EPOCH:  95
EPOCH:  96
Iter: 2100, D: 0.6119, G:1.355
```



```
EPOCH:  97
```

```
EPOCH:  98
Iter: 2150, D: 0.646, G:2.051
```

```
EPOCH:  99
EPOCH:  100
EPOCH:  101
Iter: 2200, D: 0.3058, G:2.554
```

```
EPOCH:  102
EPOCH:  103
Iter: 2250, D: 0.3878, G:4.774
```

```
EPOCH:  104
EPOCH:  105
Iter: 2300, D: 1.085, G:3.305
```

```
EPOCH:  106
EPOCH:  107
Iter: 2350, D: 0.3186, G:2.482
```

```
EPOCH:  108
```

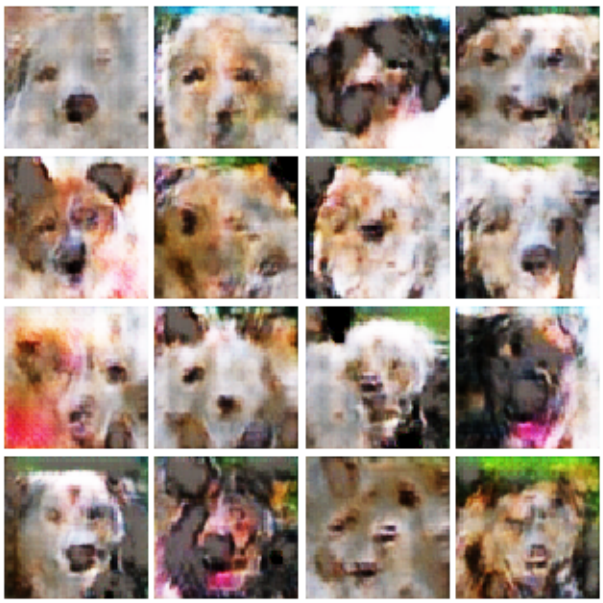
```
EPOCH:  109
EPOCH:  110
Iter: 2400, D: 0.751, G:1.894
```

```
EPOCH:  111
EPOCH:  112
Iter: 2450, D: 0.7278, G:5.115
```

```
EPOCH:  113
EPOCH:  114
Iter: 2500, D: 0.4007, G:2.998
```

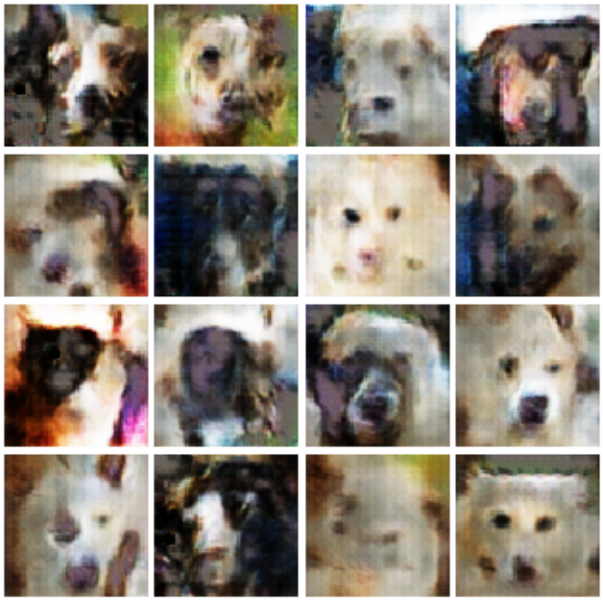
```
EPOCH:  115
EPOCH:  116
Iter: 2550, D: 0.5138, G:2.863
```

```
EPOCH:  117
EPOCH:  118
EPOCH:  119
Iter: 2600, D: 0.7732, G:4.785
```



```
EPOCH:  120
EPOCH:  121
Iter: 2650, D: 0.4183, G:2.374
```

```
EPOCH:  122
EPOCH:  123
Iter: 2700, D: 0.5076, G:4.684
```



```
EPOCH:  124
EPOCH:  125
EPOCH:  126
Iter: 2750, D: 0.2939, G:4.975
```

```
EPOCH:  127
EPOCH:  128
Iter: 2800, D: 0.477, G:4.369
```

```
EPOCH:  129
EPOCH:  130
Iter: 2850, D: 0.4775, G:2.355
```

```
EPOCH:  131
```

```
EPOCH:  132
Iter: 2900, D: 0.3916, G:4.001
```

```
EPOCH:  133
EPOCH:  134
EPOCH:  135
Iter: 2950, D: 0.562, G:5.788
```

```
EPOCH:  136
EPOCH:  137
Iter: 3000, D: 0.3671, G:3.164
```

```
EPOCH:  138
EPOCH:  139
Iter: 3050, D: 1.058, G:4.071
```

```
EPOCH:  140
EPOCH:  141
Iter: 3100, D: 0.6665, G:4.569
```

```
EPOCH:  142
```

```
EPOCH:  143
EPOCH:  144
Iter: 3150, D: 0.6846, G:3.193
```

```
EPOCH:  145
EPOCH:  146
Iter: 3200, D: 0.5569, G:2.004
```

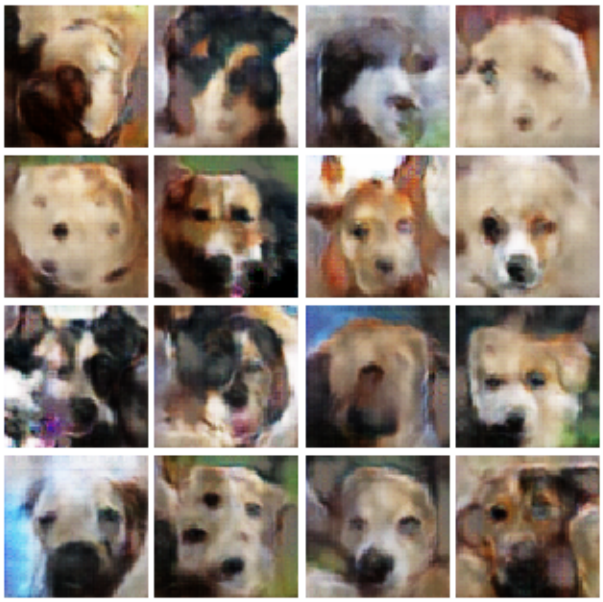
```
EPOCH:  147
EPOCH:  148
Iter: 3250, D: 0.4629, G:3.221
```

```
EPOCH:  149
EPOCH:  150
EPOCH:  151
Iter: 3300, D: 0.4111, G:3.769
```

```
EPOCH:  152
EPOCH:  153
Iter: 3350, D: 0.764, G:5.756
```

```
EPOCH:  154
```

```
EPOCH:  155
Iter: 3400, D: 0.4272, G:3.783
```



```
EPOCH:  156
EPOCH:  157
Iter: 3450, D: 1.101, G:1.684
```

```
EPOCH:  158
EPOCH:  159
EPOCH:  160
Iter: 3500, D: 0.5408, G:2.534
```

```
EPOCH:  161
EPOCH:  162
Iter: 3550, D: 0.605, G:3.486
```

```
EPOCH:  163
EPOCH:  164
Iter: 3600, D: 0.6403, G:3.928
```

```
EPOCH:  165
```

```
EPOCH:  166
Iter: 3650, D: 0.52, G:2.047
```

```
EPOCH:  167
EPOCH:  168
EPOCH:  169
Iter: 3700, D: 1.784, G:5.401
```

```
EPOCH:  170
EPOCH:  171
Iter: 3750, D: 0.3819, G:3.795
```

```
EPOCH:  172
EPOCH:  173
Iter: 3800, D: 0.7515, G:2.263
```

```
EPOCH:  174
EPOCH:  175
EPOCH:  176
Iter: 3850, D: 1.114, G:2.539
```

```
EPOCH:  177
EPOCH:  178
Iter: 3900, D: 0.6965, G:4.483
```

```
EPOCH:  179
EPOCH:  180
Iter: 3950, D: 1.082, G:2.021
```

```
EPOCH:  181
EPOCH:  182
Iter: 4000, D: 2.558, G:4.945
```

```
EPOCH:  183
EPOCH:  184
EPOCH:  185
Iter: 4050, D: 0.3962, G:2.803
```

```
EPOCH:  186
EPOCH:  187
Iter: 4100, D: 0.5238, G:2.256
```

```
EPOCH:  188
```

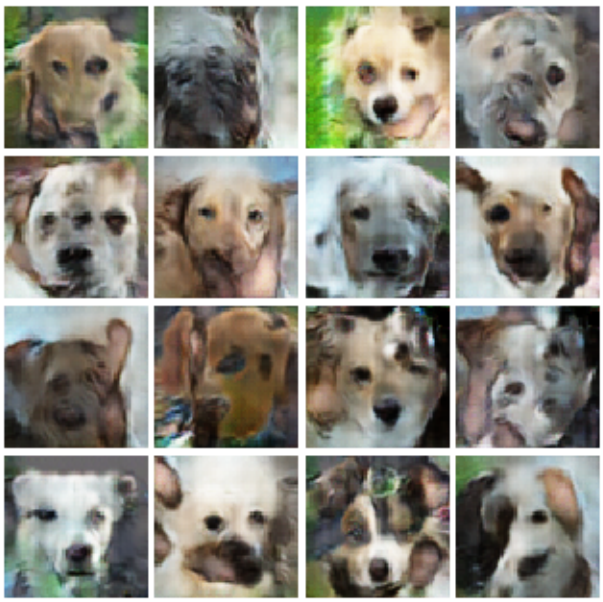
EPOCH: 189
Iter: 4150, D: 0.3937, G:3.496

EPOCH: 190
EPOCH: 191
Iter: 4200, D: 0.7048, G:3.374

```
EPOCH:  192
EPOCH:  193
EPOCH:  194
Iter: 4250, D: 0.4764, G:2.677
```

```
EPOCH:  195
EPOCH:  196
Iter: 4300, D: 0.3797, G:3.075
```

```
EPOCH:  197
EPOCH:  198
Iter: 4350, D: 0.4082, G:3.572
```



```
EPOCH:  199
```

```
EPOCH:  200
```

[ ]:
```
train(D, G, D_optimizer, G_optimizer, discriminator_loss,
          generator_loss, num_epochs=NUM_EPOCHS, show_every=50,
          batch_size=batch_size, train_loader=dog_loader_train, device=device)
```

```
EPOCH:  1
Iter: 0, D: 0.7443, G:4.264
```

```
EPOCH:  2
EPOCH:  3
Iter: 50, D: 0.5131, G:2.586
```

```
EPOCH:  4
EPOCH:  5
Iter: 100, D: 1.279, G:3.748
```

```
EPOCH:  6
```

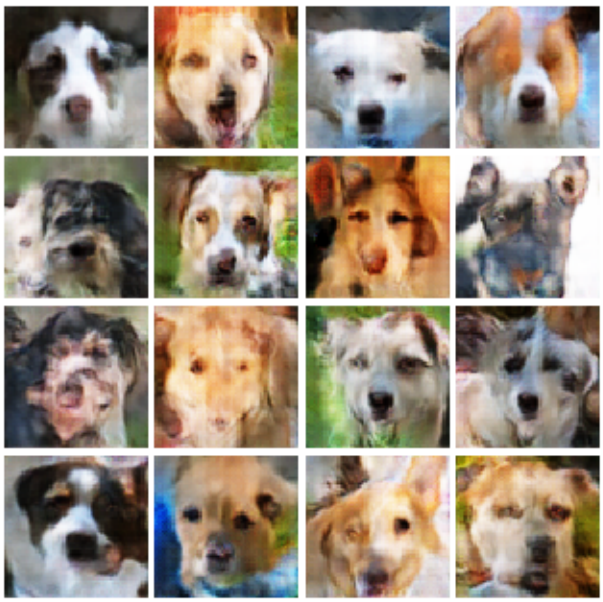
```
EPOCH:  7
Iter: 150, D: 0.5254, G:3.507
```

```
EPOCH:  8
EPOCH:  9
EPOCH:  10
Iter: 200, D: 0.4195, G:3.065
```

```
EPOCH:  11
EPOCH:  12
Iter: 250, D: 0.3574, G:2.902
```

```
EPOCH:  13
EPOCH:  14
Iter: 300, D: 0.5297, G:1.432
```

```
EPOCH:  15
EPOCH:  16
Iter: 350, D: 0.6446, G:3.379
```



```
EPOCH:  17
```

```
EPOCH:  18
EPOCH:  19
Iter: 400, D: 0.3581, G:3.601
```

```
EPOCH:  20
EPOCH:  21
Iter: 450, D: 0.6807, G:2.878
```

```
EPOCH:  22
EPOCH:  23
Iter: 500, D: 0.6603, G:1.655
```

```
EPOCH:  24
EPOCH:  25
EPOCH:  26
Iter: 550, D: 0.3963, G:3.366
```

```
EPOCH:  27
EPOCH:  28
Iter: 600, D: 1.226, G:1.32
```

```
EPOCH:  29
```

```
EPOCH:  30
Iter: 650, D: 0.3361, G:4.004
```

```
EPOCH:  31
EPOCH:  32
Iter: 700, D: 0.4197, G:3.824
```

```
EPOCH:  33
EPOCH:  34
EPOCH:  35
Iter: 750, D: 0.4287, G:3.131
```

```
EPOCH:  36
EPOCH:  37
Iter: 800, D: 0.6259, G:5.687
```

```
EPOCH:  38
EPOCH:  39
Iter: 850, D: 0.7528, G:3.018
```

```
EPOCH:  40
```

```
EPOCH:  41
Iter: 900, D: 0.6188, G:2.393
```

```
EPOCH:  42
EPOCH:  43
EPOCH:  44
Iter: 950, D: 0.6318, G:1.753
```

```
EPOCH:  45
EPOCH:  46
Iter: 1000, D: 0.6878, G:2.114
```

```
EPOCH:  47
EPOCH:  48
Iter: 1050, D: 0.3919, G:2.479
```

```
EPOCH:  49
EPOCH:  50
EPOCH:  51
Iter: 1100, D: 0.8548, G:3.186
```

```
EPOCH:  52
EPOCH:  53
Iter: 1150, D: 0.4915, G:3.245
```

```
EPOCH:  54
EPOCH:  55
Iter: 1200, D: 0.3002, G:3.114
```

```
EPOCH:  56
EPOCH:  57
Iter: 1250, D: 0.3677, G:3.624
```

```
EPOCH:  58
EPOCH:  59
EPOCH:  60
Iter: 1300, D: 0.3864, G:2.004
```

```
EPOCH:  61
EPOCH:  62
Iter: 1350, D: 0.3095, G:3.226
```

```
EPOCH:  63
```

```
EPOCH:  64
Iter: 1400, D: 1.028, G:0.7783
```

```
EPOCH:  65
EPOCH:  66
Iter: 1450, D: 1.317, G:1.13
```

```
EPOCH:  67
EPOCH:  68
EPOCH:  69
Iter: 1500, D: 0.3897, G:2.675
```

```
EPOCH:  70
EPOCH:  71
Iter: 1550, D: 0.396, G:2.067
```

```
EPOCH:  72
EPOCH:  73
Iter: 1600, D: 0.3293, G:2.819
```

```
EPOCH:  74
```

```
EPOCH:  75
EPOCH:  76
Iter: 1650, D: 0.6036, G:4.496
```

```
EPOCH:  77
EPOCH:  78
Iter: 1700, D: 0.4071, G:3.638
```

```
EPOCH:  79
EPOCH:  80
Iter: 1750, D: 0.4616, G:4.671
```

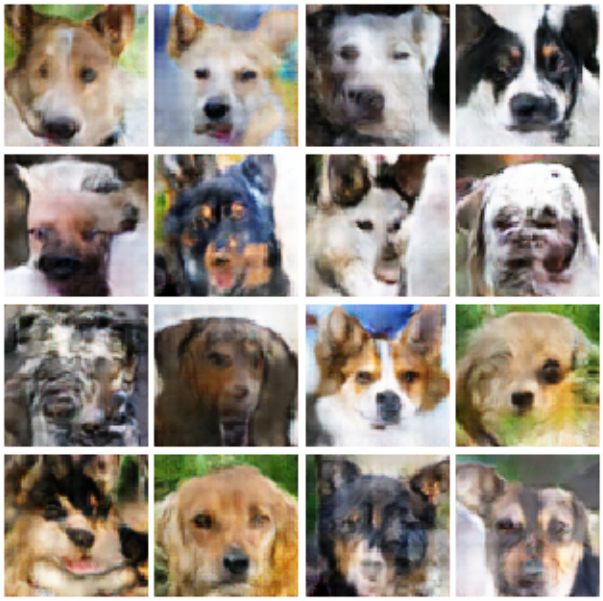
```
EPOCH:  81
EPOCH:  82
Iter: 1800, D: 0.3699, G:3.5
```

```
EPOCH:  83
EPOCH:  84
EPOCH:  85
Iter: 1850, D: 0.3505, G:1.984
```

```
EPOCH:  86
EPOCH:  87
Iter: 1900, D: 0.6654, G:1.913
```

```
EPOCH:  88
EPOCH:  89
Iter: 1950, D: 0.5576, G:3.593
```

```
EPOCH:  90
EPOCH:  91
Iter: 2000, D: 0.5075, G:2.305
```



```
EPOCH:  92
EPOCH:  93
EPOCH:  94
Iter: 2050, D: 0.2616, G:2.831
```

```
EPOCH:  95
EPOCH:  96
Iter: 2100, D: 0.4052, G:3.696
```

```
EPOCH:  97
```

```
EPOCH:  98
Iter: 2150, D: 0.7685, G:2.676
```

```
EPOCH:  99
EPOCH:  100
EPOCH:  101
Iter: 2200, D: 1.677, G:1.508
```

```
EPOCH:  102
EPOCH:  103
Iter: 2250, D: 0.7784, G:4.412
```

```
EPOCH:  104
EPOCH:  105
Iter: 2300, D: 0.2383, G:4.078
```

```
EPOCH:  106
EPOCH:  107
Iter: 2350, D: 0.2653, G:3.742
```

```
EPOCH:  108
```

```
EPOCH:  109
EPOCH:  110
Iter: 2400, D: 0.2858, G:3.129
```

```
EPOCH:  111
EPOCH:  112
Iter: 2450, D: 4.503, G:9.163
```

```
EPOCH:  113
EPOCH:  114
Iter: 2500, D: 1.642, G:0.923
```

```
EPOCH:  115
EPOCH:  116
Iter: 2550, D: 1.464, G:1.076
```

```
EPOCH:  117
EPOCH:  118
EPOCH:  119
Iter: 2600, D: 1.294, G:1.957
```

```
EPOCH:  120
EPOCH:  121
Iter: 2650, D: 0.7849, G:2.133
```

```
EPOCH:  122
EPOCH:  123
Iter: 2700, D: 0.4893, G:2.687
```

```
EPOCH:  124
EPOCH:  125
EPOCH:  126
Iter: 2750, D: 0.6195, G:5.036
```

```
EPOCH:  127
EPOCH:  128
Iter: 2800, D: 0.4943, G:2.019
```

```
EPOCH:  129
EPOCH:  130
Iter: 2850, D: 0.3445, G:4.07
```

```
EPOCH:  131
```

```
EPOCH:  132
Iter: 2900, D: 0.3882, G:4.418
```

```
EPOCH:  133
EPOCH:  134
EPOCH:  135
Iter: 2950, D: 0.5226, G:3.061
```

```
EPOCH:  136
EPOCH:  137
Iter: 3000, D: 0.682, G:4.569
```

```
EPOCH:  138
EPOCH:  139
Iter: 3050, D: 0.4632, G:1.829
```

```
EPOCH:  140
EPOCH:  141
Iter: 3100, D: 0.3096, G:2.905
```

```
EPOCH:  142
```

```
EPOCH:  143
EPOCH:  144
Iter: 3150, D: 0.2037, G:5.173
```

```
EPOCH:  145
EPOCH:  146
Iter: 3200, D: 0.2066, G:3.42
```

```
EPOCH:  147
EPOCH:  148
Iter: 3250, D: 1.697, G:0.955
```

```
EPOCH:  149
EPOCH:  150
EPOCH:  151
Iter: 3300, D: 1.528, G:0.7102
```

```
EPOCH:  152
EPOCH:  153
Iter: 3350, D: 1.475, G:0.8106
```

```
EPOCH:  154
```

```
EPOCH:  155
Iter: 3400, D: 1.497, G:0.8644
```

```
EPOCH:  156
EPOCH:  157
Iter: 3450, D: 1.463, G:0.7675
```

```
EPOCH:  158
EPOCH:  159
EPOCH:  160
Iter: 3500, D: 1.461, G:0.9261
```

```
EPOCH:  161
EPOCH:  162
Iter: 3550, D: 1.432, G:0.8991
```

```
EPOCH:  163
EPOCH:  164
Iter: 3600, D: 1.402, G:0.7002
```

```
EPOCH:  165
```

```
EPOCH:  166
Iter: 3650, D: 1.368, G:0.8707
```

```
EPOCH:  167
EPOCH:  168
EPOCH:  169
Iter: 3700, D: 1.398, G:0.7297
```

```
EPOCH:  170
EPOCH:  171
Iter: 3750, D: 1.41, G:0.845
```

```
EPOCH:  172
EPOCH:  173
Iter: 3800, D: 1.384, G:0.8024
```

```
EPOCH:  174
EPOCH:  175
EPOCH:  176
Iter: 3850, D: 1.339, G:0.756
```

```
EPOCH:  177
EPOCH:  178
Iter: 3900, D: 1.394, G:0.8869
```

```
EPOCH:  179
EPOCH:  180
Iter: 3950, D: 1.304, G:0.9702
```

```
EPOCH:  181
EPOCH:  182
Iter: 4000, D: 1.372, G:0.8883
```

```
EPOCH:  183
EPOCH:  184
EPOCH:  185
Iter: 4050, D: 1.422, G:0.9508
```

```
EPOCH:  186
EPOCH:  187
Iter: 4100, D: 1.275, G:0.8154
```

```
EPOCH:  188
```

```
EPOCH:  189
Iter: 4150, D: 1.477, G:0.9051
```

```
EPOCH:  190
EPOCH:  191
Iter: 4200, D: 1.374, G:1.002
```

```
EPOCH:  192
EPOCH:  193
EPOCH:  194
Iter: 4250, D: 1.453, G:1.264
```

```
EPOCH:  195
EPOCH:  196
Iter: 4300, D: 1.16, G:2.844
```

```
EPOCH:  197
EPOCH:  198
Iter: 4350, D: 1.261, G:4.243
```

```
EPOCH:  199
```

```
EPOCH:  200
```

10 Save to PDF

[ ]:
```
%%capture

from google.colab import drive
drive.mount('/content/drive')
# install tex; first run may take several minutes
! apt-get install texlive-xetex
# file path and save location below are default; please change if they do not␣
  ↪match yours
! jupyter nbconvert --output-dir='/content/drive/MyDrive/' '/content/drive/
  ↪MyDrive/CS444/assignment4/MP4.ipynb' --to pdf
```