CS 5814 Homework 4, Part 1: Generative Adversarial Networks

```
from google.colab import files
import zipfile
import os
# Create a zip file containing the folder you want to upload
folder_to_upload = "/path/to/your/folder"
zip_file_name = "folder_to_upload.zip"
with zipfile.ZipFile(zip_file_name, 'w', zipfile.ZIP_DEFLATED) as zipf:
    for root, dirs, files in os.walk(folder_to_upload):
        for file in files:
            zipf.write(os.path.join(root, file), os.path.relpath(os.path.join(root, file), f
# Upload the zip file
uploaded = files.upload()
```

```
Choose Files cats.zip
• cats.zip(application/x-zip-compressed) - 49750115 bytes, last modified: 4/27/2024 - 100% done
Saving cats.zip to cats.zip
```

```
import zipfile
import os

# Specify the path to the zip file you uploaded
zip_file_path = "/content/cats.zip"

# Specify the directory where you want to extract and save the contents
extracted_folder_path = "/content/cats"

# Create a directory to extract the contents if it doesn't exist
if not os.path.exists(extracted_folder_path):
    os.makedirs(extracted_folder_path)

# Extract and save the contents of the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extracted_folder_path)

print("Files extracted and saved successfully to:", extracted_folder_path)
```

```
Files extracted and saved successfully to: /content/cats
```

```
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
```

```
# %pip install torchvision
```

```
WARNING: The directory '/home/jovyan/.cache/pip' or its parent directory is not owned or
Defaulting to user installation because normal site-packages is not writeable
Collecting torchvision
  Downloading torchvision-0.17.2-cp38-cp38-manylinux1_x86_64.whl (6.9 MB)
     |████████████████████████████████| 6.9 MB 3.3 MB/s eta 0:00:01
Requirement already satisfied: numpy in /opt/conda/lib/python3.8/site-packages (from tor
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /opt/conda/lib/python3.8/site-pa
Requirement already satisfied: torch==2.2.2 in /home/stanleygabriel97/.local/lib/python3
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105; platform_system == "Lir
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106; platform_system == "Linux
Requirement already satisfied: nvidia-nccl-cu12==2.19.3; platform_system == "Linux" and
Requirement already satisfied: triton==2.2.0; platform_system == "Linux" and platform_ma
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106; platform_system == "Lir
Requirement already satisfied: sympy in /opt/conda/lib/python3.8/site-packages (from tor
Requirement already satisfied: networkx in /opt/conda/lib/python3.8/site-packages (from
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54; platform_system == "Linux"
Requirement already satisfied: typing-extensions>=4.8.0 in /home/stanleygabriel97/.local
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105; platform_system == "Linux" ar
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.8/site-packages (from to
Requirement already satisfied: filelock in /home/stanleygabriel97/.local/lib/python3.8/s
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1; platform_system == "Linux"
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107; platform_system == "Lir
Requirement already satisfied: fsspec in /opt/conda/lib/python3.8/site-packages (from to
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105; platform_system == "L
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26; platform_system == "Linux" a
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105; platform_system == "Lir
Requirement already satisfied: nvidia-nvjitlink-cu12 in /home/stanleygabriel97/.local/li
Requirement already satisfied: mpmath>=0.19 in /opt/conda/lib/python3.8/site-packages (f
Requirement already satisfied: decorator>=4.3.0 in /opt/conda/lib/python3.8/site-package
Requirement already satisfied: MarkupSafe>=0.23 in /opt/conda/lib/python3.8/site-package
Installing collected packages: torchvision
Successfully installed torchvision-0.17.2
Note: you may need to restart the kernel to use updated packages.
```

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

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

```
# import os
# cwd = os.getcwd()
```

```
# cwd
```

```
'/home/stanleygabriel97/cs5814/hw4/part1'
```

```
# import os
# datadir = "./part1/" # path to the homework
# if not os.path.exists(cwd):
#   !ln -s "" $datadir # path to the homework
# os.chdir(cwd)
# !pwd
```

```
/home/stanleygabriel97/cs5814/hw4/part1
```

```
import zipfile
import os

# Specify the path to the zip file you uploaded
zip_file_path = "/content/cats.zip"

# Specify the directory where you want to extract and save the contents
extracted_folder_path = "/content/cats"

# Create a directory to extract the contents if it doesn't exist
if not os.path.exists(extracted_folder_path):
    os.makedirs(extracted_folder_path)

# Extract and save the contents of the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extracted_folder_path)

print("Files extracted and saved successfully to:", extracted_folder_path)
```

```
Files extracted and saved successfully to: /content/cats
```

```
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
```

```
uploaded = files.upload()
```

Choose Files 4 files

- **losses.py**(text/x-python) - 3747 bytes, last modified: 5/1/2024 - 100% done
- **models.py**(text/x-python) - 2021 bytes, last modified: 5/1/2024 - 100% done
- **train.py**(text/x-python) - 4489 bytes, last modified: 5/1/2024 - 100% done
- **utils.py**(text/x-python) - 1609 bytes, last modified: 5/1/2024 - 100% done

```
Saving losses.py to losses.py
Saving models.py to models.py
Saving train.py to train.py
Saving utils.py to utils.py
```

```
from train import train
```

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

```
device
```

```
device(type='cuda', index=0)
```

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.

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:

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

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 $\log(1 - D(G(z)))$.

```
from losses import discriminator_loss, generator_loss
```

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:

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

```
from losses import ls_discriminator_loss, ls_generator_loss
```

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).

```
from models import Discriminator, Generator
```

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.

```
batch_size = 32
imsize = 64
cat_root = '/content/cats'

# Define your transformations
transform_augment = transforms.Compose([
    transforms.Resize((imsize, imsize)), # Ensures all images are the same size
    transforms.RandomCrop(imsize, padding=4),  # Pads and then crops
    transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(15),
    transforms.ToTensor(),  # Converts to tensor
])

# Create your ImageFolder dataset
cat_train = ImageFolder(root=cat_root, transform=transform_augment)

# Create DataLoader
cat_loader_train = DataLoader(cat_train, batch_size=batch_size, shuffle=True, drop_last=True
```

Visualize dataset

```
from gan.utils import show_images

for batch in cat_loader_train:
    imgs = batch[0].numpy().squeeze()
    show_images(imgs, color=True)
    break
```

Training

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

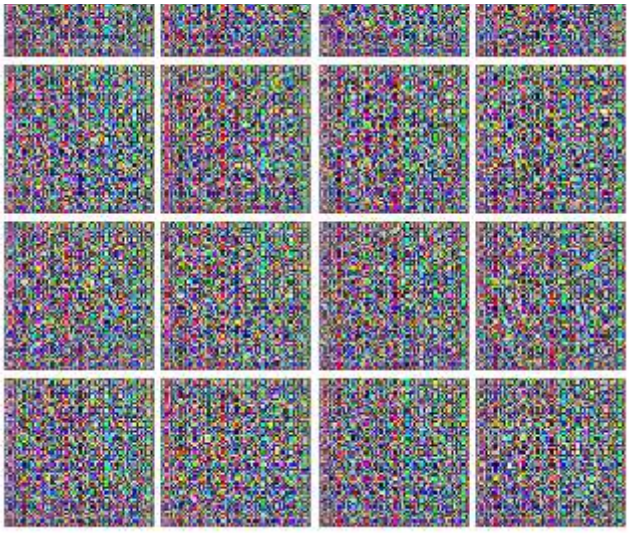
```
NOISE_DIM = 100
NUM_EPOCHS = 50
learning_rate = 0.001
```

Train GAN

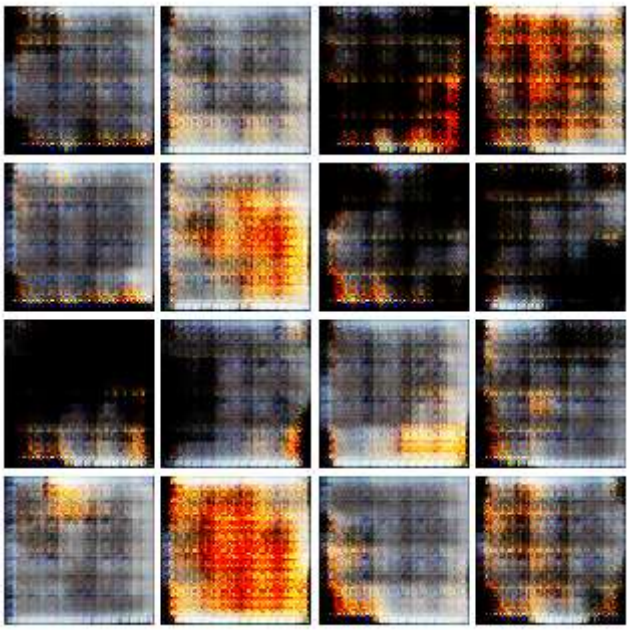
```
D = Discriminator().to(device)
G = Generator(noise_dim=NOISE_DIM).to(device)
```

```
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)
```

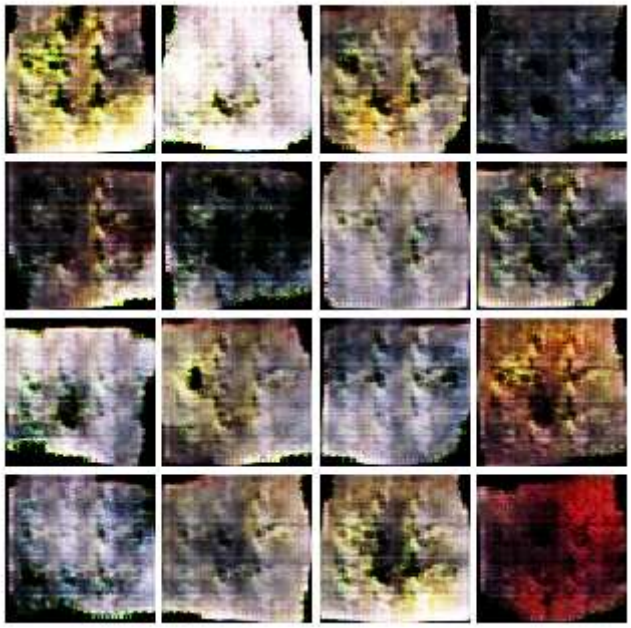


Iter: 250, D: 1.032, G:4.5

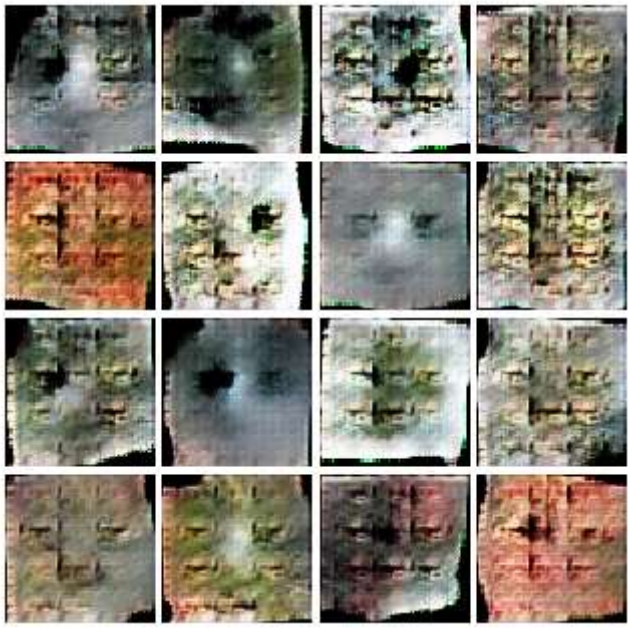


EPOCH: 2
Iter: 500, D: 0.8907, G:1.909

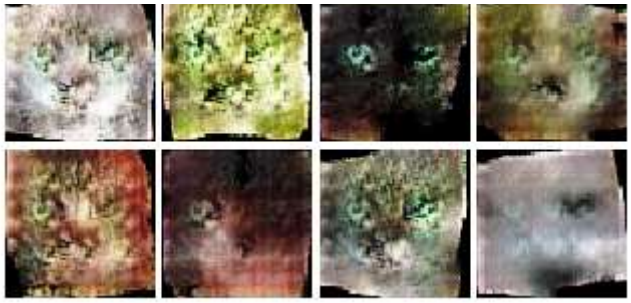
Iter: 750, D: 0.9202, G:3.247



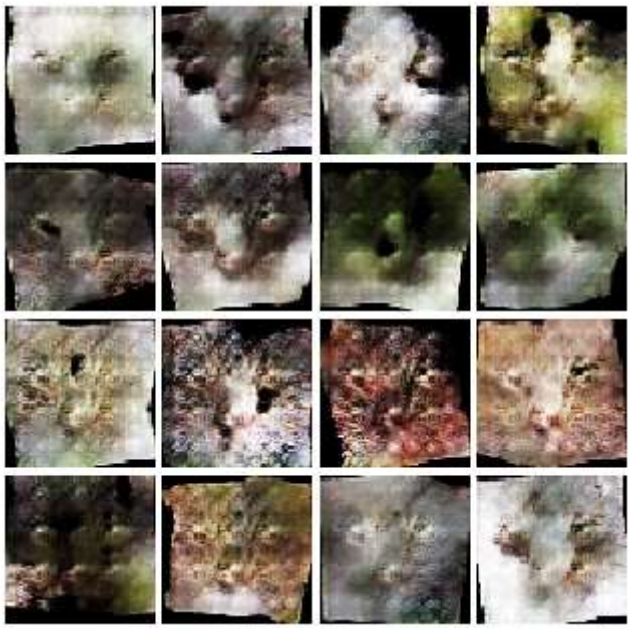
EPOCH: 3
Iter: 1000, D: 1.417, G:1.554



Iter: 1250, D: 1.204, G:3.537

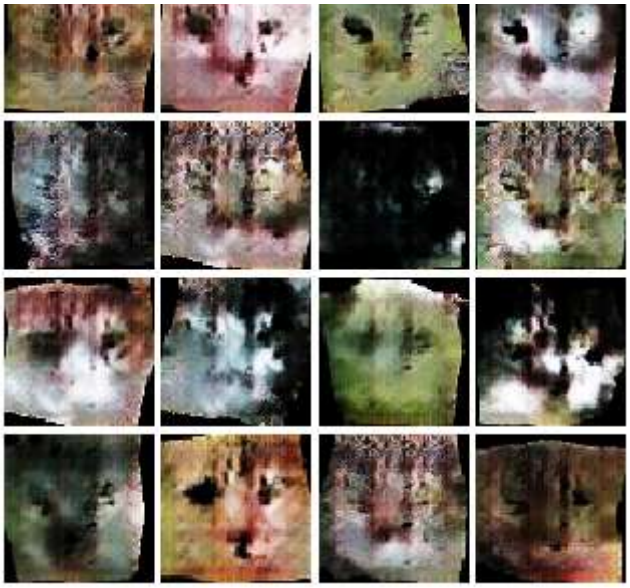


EPOCH: 4
Iter: 1500, D: 1.536, G:2.177



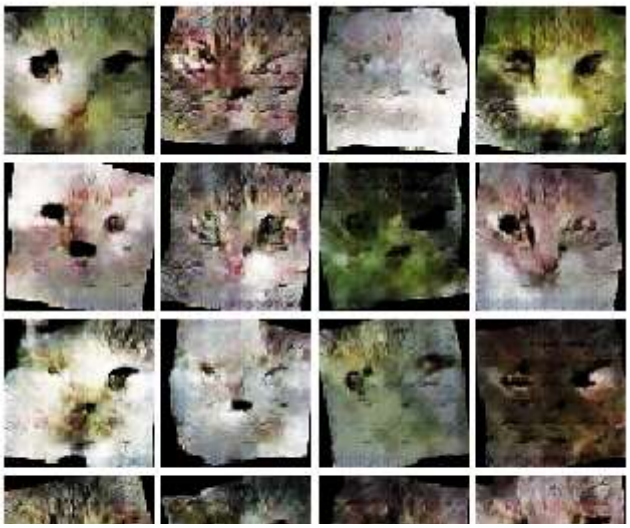
Iter: 1750, D: 1.267, G:4.24

EPOCH: 5
Iter: 2000, D: 1.007, G:1.771



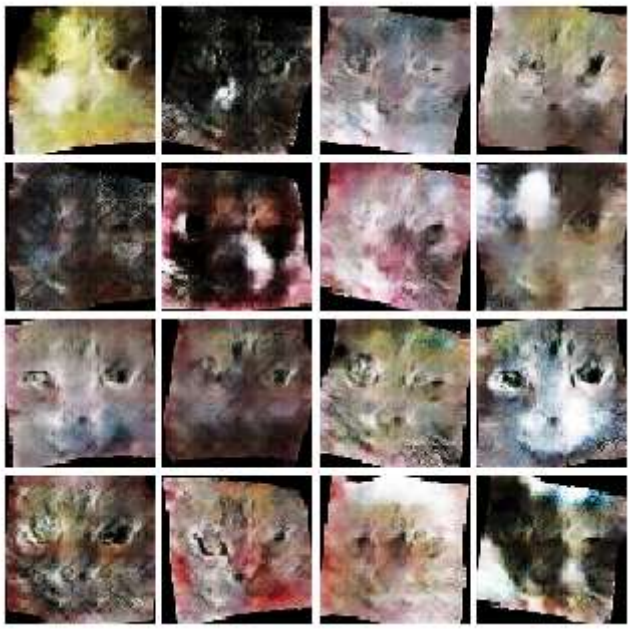
Iter: 2250, D: 1.022, G:2.23

EPOCH: 6
Iter: 2500, D: 1.305, G:3.021

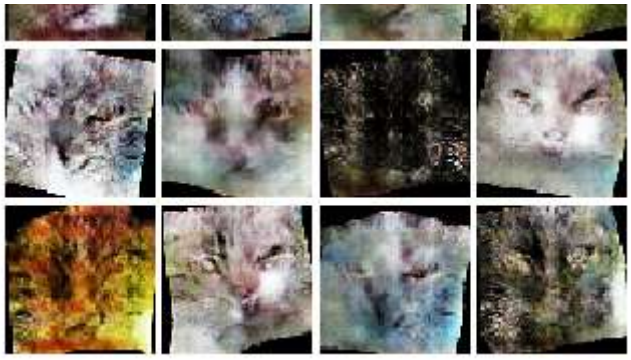


Iter: 2750, D: 0.9092, G:2.5

EPOCH: 7
Iter: 3000, D: 0.6024, G:3.235



Iter: 3250, D: 0.6535, G:3.135



EPOCH: 8
Iter: 3500, D: 0.6437, G:4.008

Iter: 3750, D: 1.176, G:5.605

EPOCH: 9
Iter: 4000, D: 0.6453, G:4.451

```
Iter: 4250, D: 0.7271, G:4.437
```

```
EPOCH:  10
Iter: 4500, D: 0.5916, G:3.872
```

Iter: 4750, D: 0.5339, G:3.419

EPOCH: 11
Iter: 5000, D: 0.5941, G:3.317

Iter: 5250, D: 1.549, G:2.761

EPOCH: 12
Iter: 5500, D: 0.3588, G:4.996

Iter: 5750, D: 0.6169, G:6.276

EPOCH: 13

EPOCH: 13
Iter: 6000, D: 0.6193, G:5.159

Iter: 6250, D: 0.681, G:10.27

EPOCH: 14
Iter: 6500, D: 0.3038, G:5.288

Iter: 6750, D: 0.1952, G:3.823

EPOCH: 15
Iter: 7000, D: 0.4229, G:6.833

Iter: 7250, D: 0.1874, G:5.001

EPOCH: 16
Iter: 7500, D: 0.1565, G:6.137

Iter: 7750, D: 0.9893, G:2.385

```
EPOCH:  17
Iter: 8000, D: 0.3265, G:6.547
```

```
Iter: 8250, D: 0.4314, G:6.103
```

```
EPOCH:  18
Iter: 8500, D: 0.06536, G:5.327
```

Iter: 8750, D: 0.2058, G:5.863

EPOCH: 19
Iter: 9000, D: 0.2069, G:8.763

Iter: 9250, D: 0.8755, G:11.01

EPOCH: 20
Iter: 9500, D: 0.3802, G:8.224

Iter: 9750, D: 0.03611, G:6.363

```
EPOCH:  21
Iter: 10000, D: 0.1062, G:7.084
```

```
Iter: 10250, D: 0.0477, G:6.773
```

```
EPOCH:  22
Iter: 10500, D: 0.2313, G:5.054
```

Iter: 10750, D: 0.07899, G:5.622

EPOCH: 23
Iter: 11000, D: 0.1112, G:6.658

Iter: 11250, D: 1.728, G:21.38

EPOCH: 24
Iter: 11500, D: 0.03007, G:6.568

Iter: 11750, D: 0.2212, G:5.852

EPOCH: 25
Iter: 12000, D: 0.09737, G:7.974

Iter: 12250, D: 0.06177, G:4.739

EPOCH: 26
Iter: 12500, D: 0.08106, G:6.075

Iter: 12750, D: 0.2877, G:6.02

EPOCH: 27
Iter: 13000, D: 0.04758, G:6.504

Iter: 13250, D: 0.08282, G:7.503

EPOCH: 28
Iter: 13500, D: 0.3406, G:9.794

Iter: 13750, D: 0.02785, G:7.07

EPOCH: 29
Iter: 14000, D: 0.02731, G:5.526

Iter: 14250, D: 0.3407, G:4.559

EPOCH: 30
Iter: 14500, D: 0.1133, G:4.504

Iter: 14750, D: 0.5945, G:3.644

EPOCH: 31
Iter: 15000, D: 0.246, G:7.006

```
Iter: 15250, D: 0.06481, G:8.699
```

```
EPOCH:  32
Iter: 15500, D: 0.09275, G:5.653
```

```
EPOCH:  33
Iter: 15750, D: 0.1036, G:10.19
```

Iter: 16000, D: 0.4116, G:9.544

EPOCH: 34
Iter: 16250, D: 0.08335, G:6.745

Iter: 16500, D: 1.241, G:3.693

EPOCH: 35
Iter: 16750, D: 0.03554, G:12.18

Iter: 17000, D: 0.6134, G:8.738

```
EPOCH:  36
Iter: 17250, D: 0.0218, G:7.417
```

```
Iter: 17500, D: 1.015, G:15.61
```

```
EPOCH:  37
Iter: 17750, D: 0.008697, G:11.27
```

Iter: 18000, D: 0.06906, G:6.486

EPOCH: 38
Iter: 18250, D: 0.01343, G:7.557

Iter: 18500, D: 0.2975, G:10.04

EPOCH: 39
Iter: 18750, D: 0.03125, G:7.357

Iter: 19000, D: 0.2797, G:6.413

EPOCH: 40
Iter: 19250, D: 0.04077, G:10.37

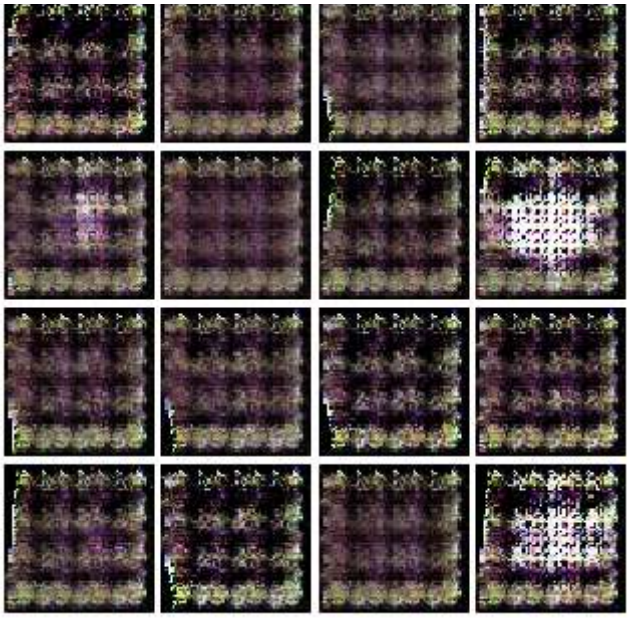
Iter: 19500, D: 0.04365, G:8.713

EPOCH: 41
Iter: 19750, D: 0.0787, G:6.169

Iter: 20000, D: 0.2463, G:4.271

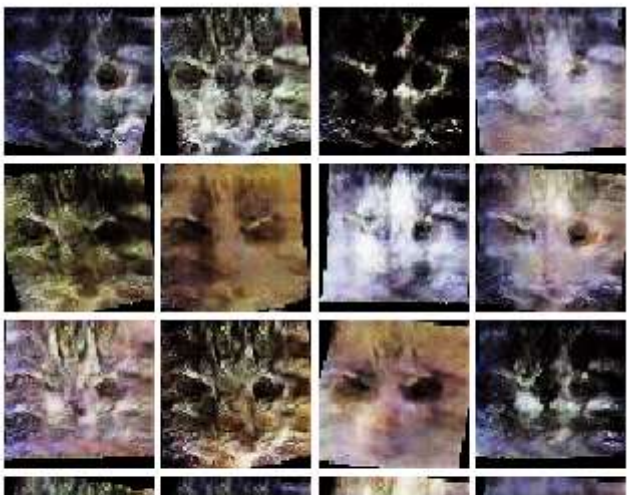
EPOCH: 42
Iter: 20250, D: 0.03496, G:22.24

Iter: 20500, D: 6.075e-05, G:28.46



EPOCH: 43
Iter: 20750, D: 0.03668, G:12.67

Iter: 21000, D: 0.08736, G:5.776



EPOCH: 44
Iter: 21250, D: 0.3015, G:5.206

Iter: 21500, D: 0.00915, G:7.408

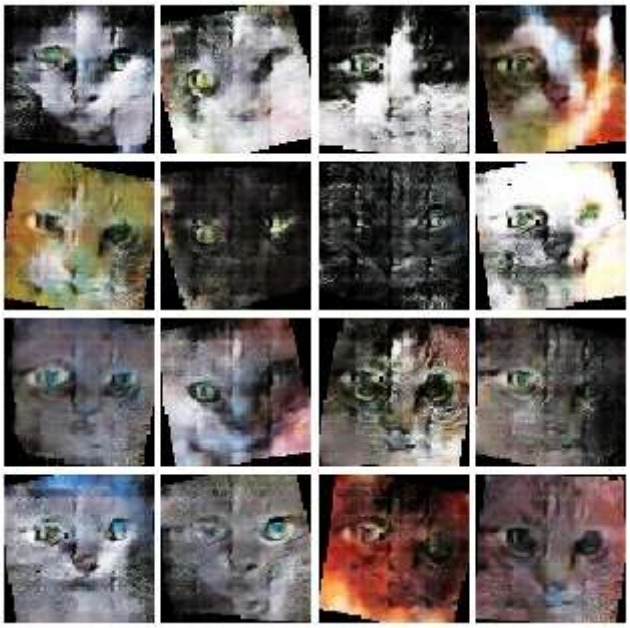
EPOCH: 45
Iter: 21750, D: 0.0346, G:7.252

```
Iter: 22000, D: 0.2835, G:8.013
```

```
EPOCH:  46
Iter: 22250, D: 0.09855, G:11.97
```

```
Iter: 22500, D: 0.04589, G:9.738
```

```
EPOCH:  47
Iter: 22750, D: 0.2167, G:10.64
```



```
Iter: 23000, D: 0.5244, G:16.5
```

```
EPOCH:  48
Iter: 23250, D: 0.2772, G:5.852
```

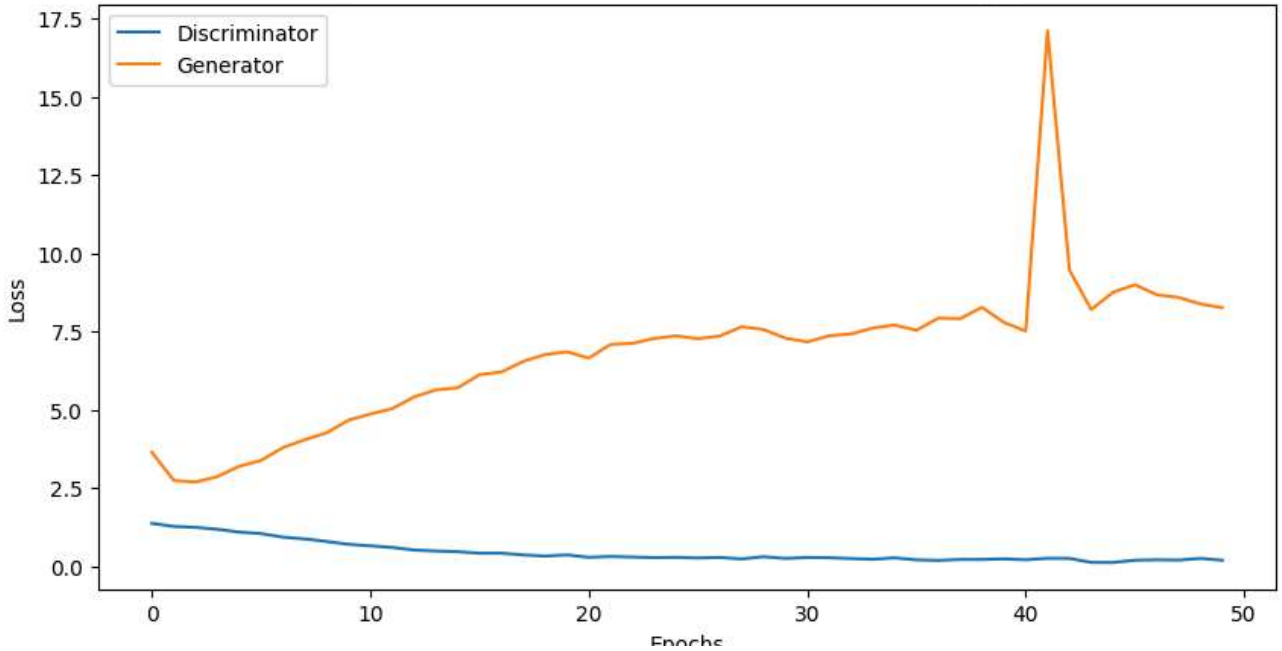
```
Iter: 23500, D: 0.0253, G:9.613
```

```
EPOCH:  49
Iter: 23750, D: 0.0905, G:4.809
```

```
Iter: 24000, D: 0.8134, G:18.92
```

```
EPOCH:  50
Iter: 24250, D: 0.00371, G:11.46
```

```
Iter: 24500, D: 0.4889, G:15.0
```

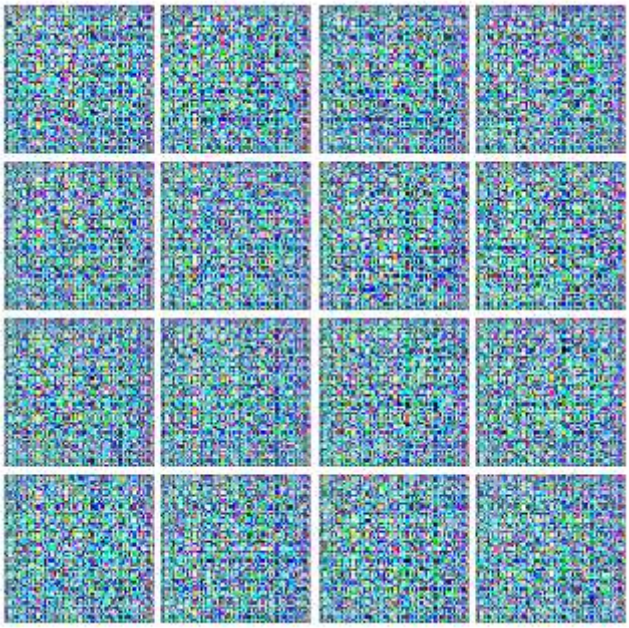


Train LS-GAN

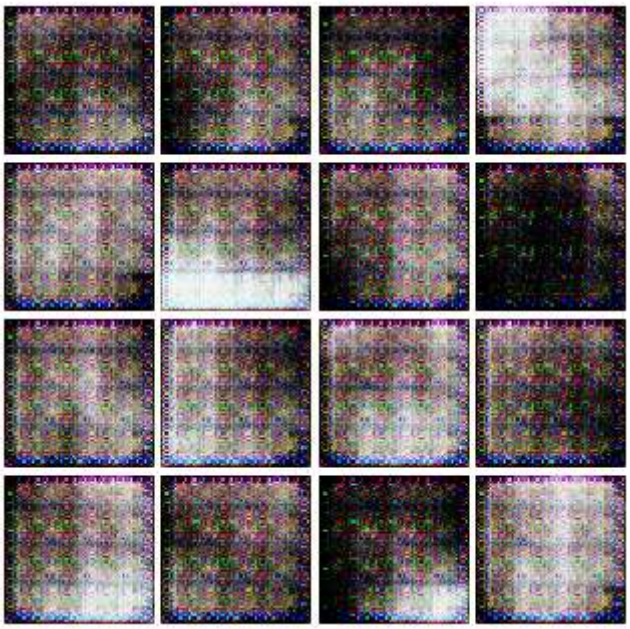
```
D = Discriminator().to(device)
G = Generator(noise_dim=NOISE_DIM).to(device)
```

```
    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)
```

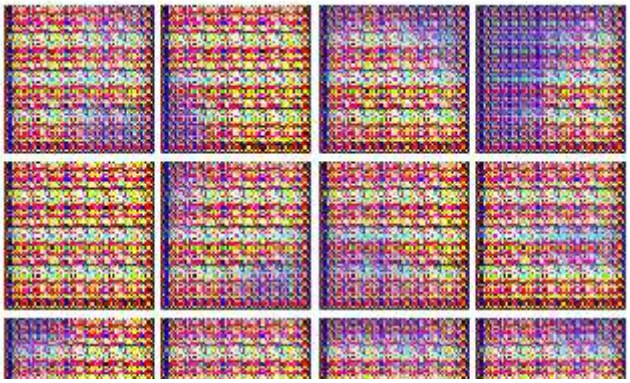
```
EPOCH:  1
Iter: 0, D: 0.7362, G:198.9
```

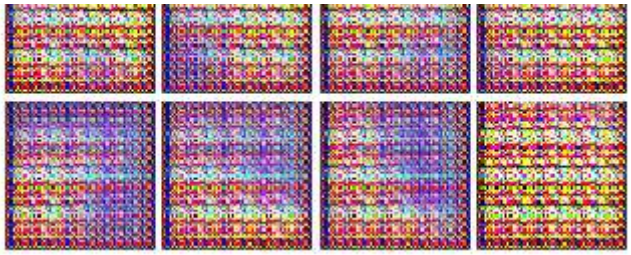


```
Iter: 250, D: 0.4907, G:0.5792
```

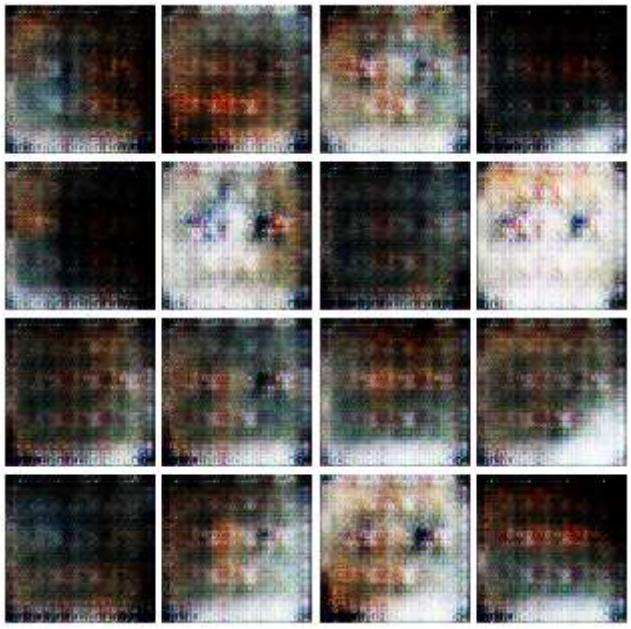


```
EPOCH:  2
Iter: 500, D: 0.1546, G:0.2421
```

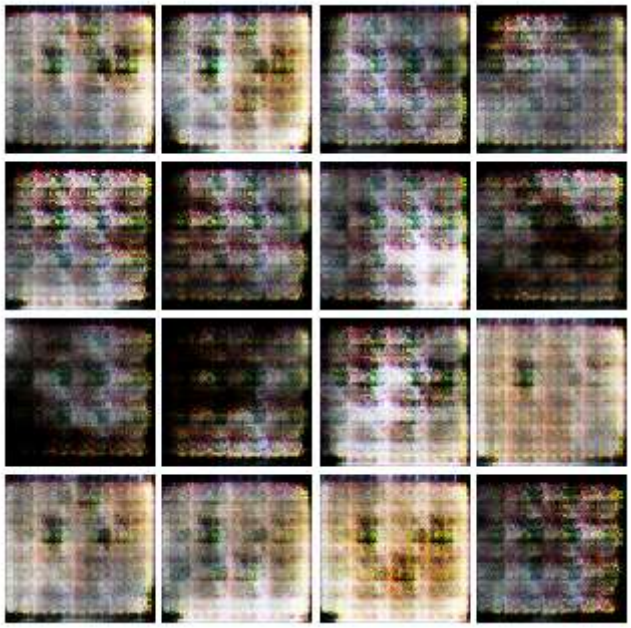




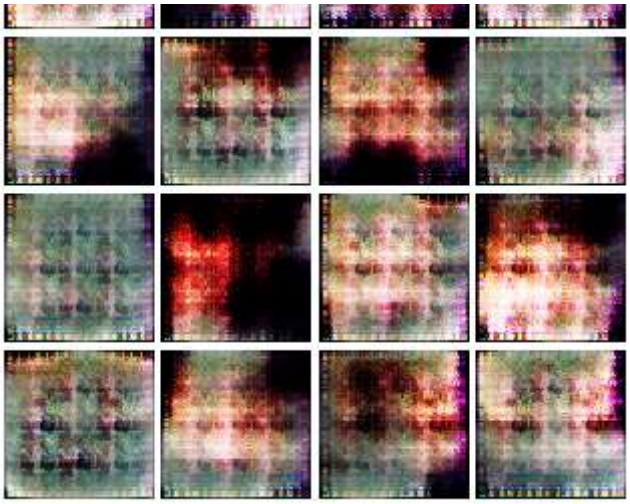
Iter: 750, D: 0.2104, G:0.5662



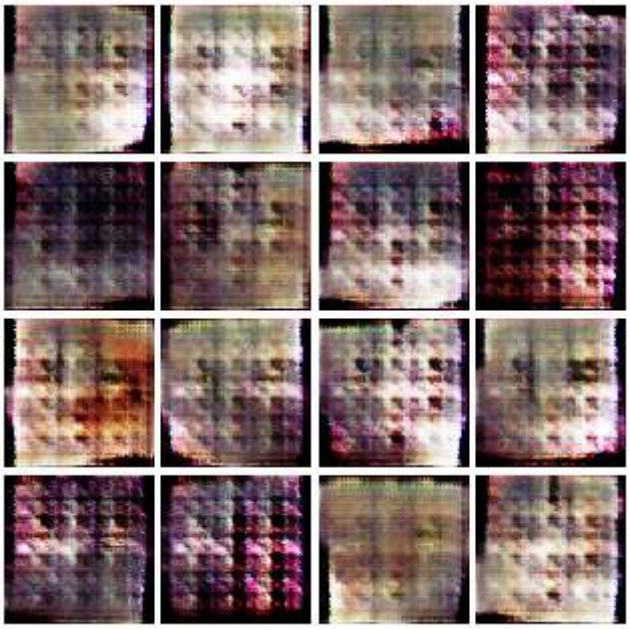
EPOCH: 3
Iter: 1000, D: 0.3954, G:0.9194



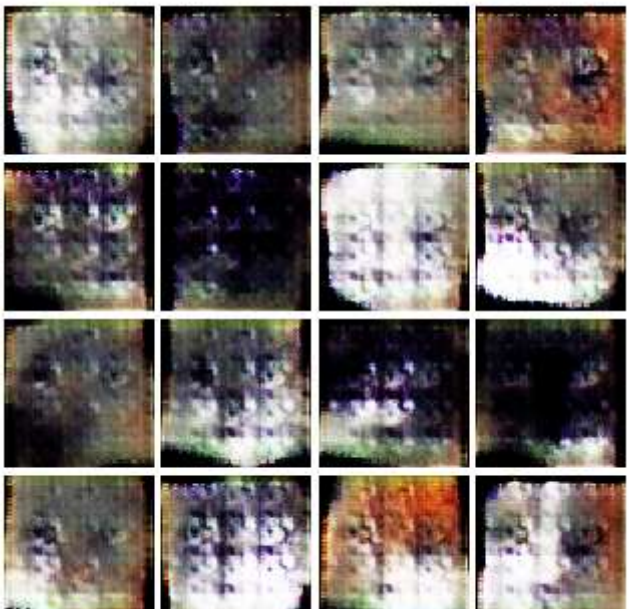
Iter: 1250, D: 0.3537, G:0.5337



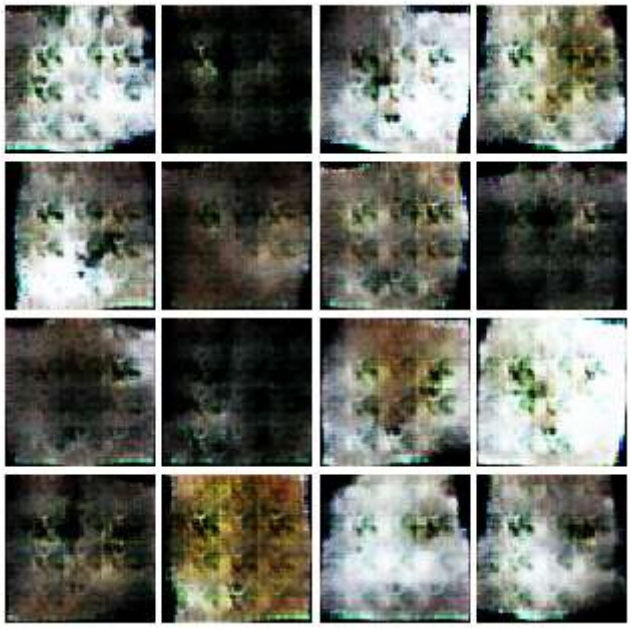
EPOCH: 4
Iter: 1500, D: 0.1804, G:0.3405



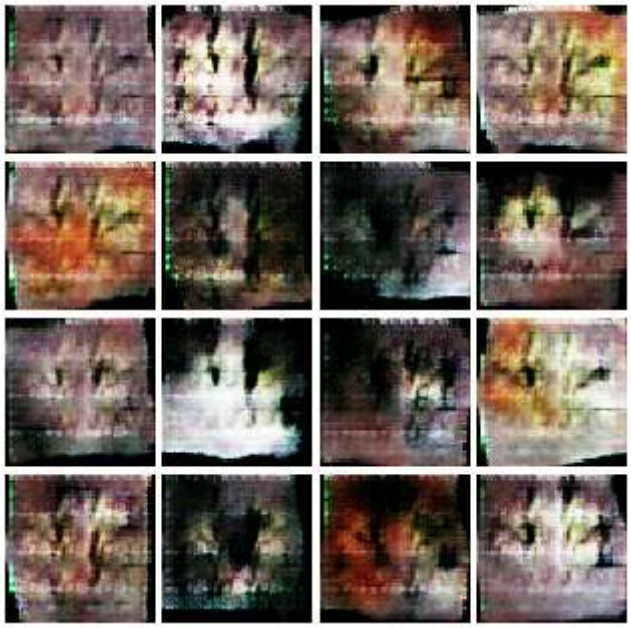
Iter: 1750, D: 0.1479, G:0.5585



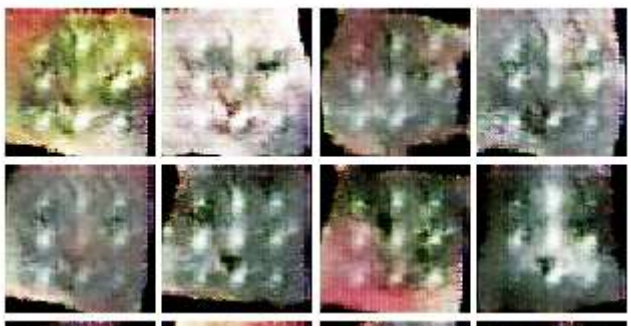
```
EPOCH:  5
Iter: 2000, D: 0.163, G:0.5274
```

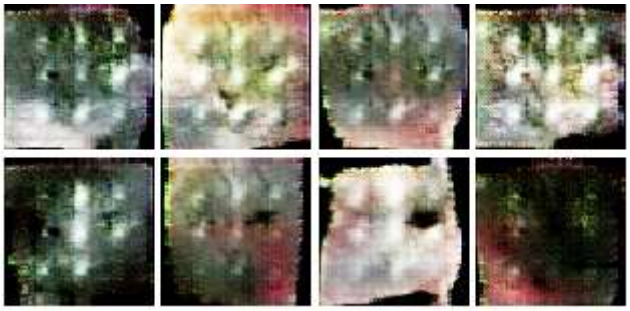


```
Iter: 2250, D: 0.5651, G:0.9845
```



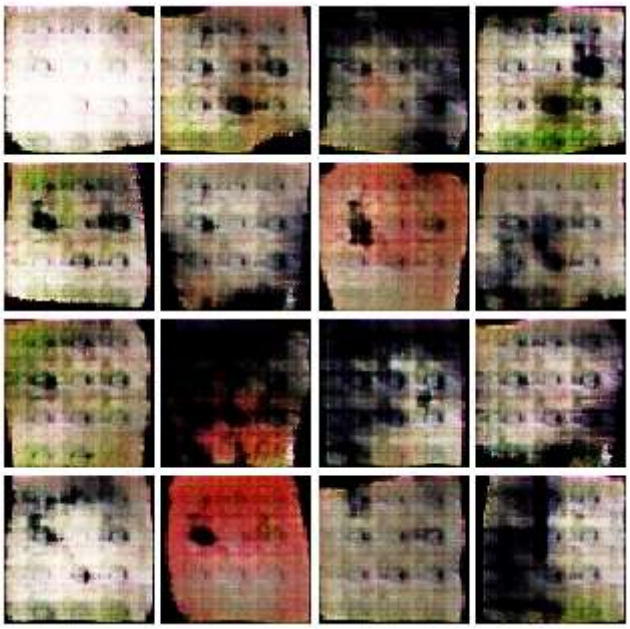
```
EPOCH:  6
Iter: 2500, D: 0.2795, G:0.1416
```



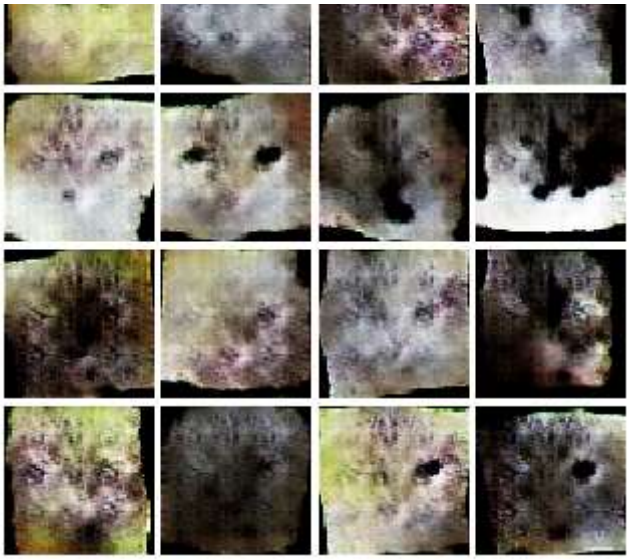


Iter: 2750, D: 0.2633, G:0.1754

EPOCH: 7
Iter: 3000, D: 0.2583, G:0.1023

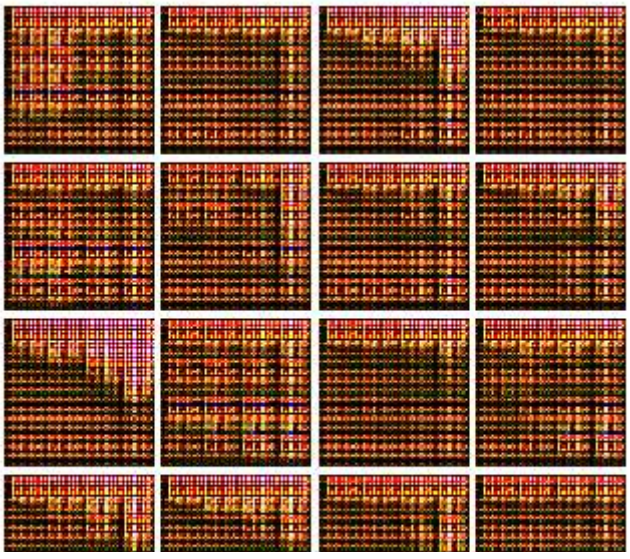


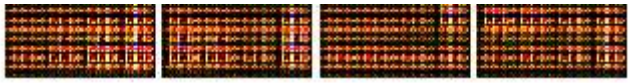
Iter: 3250, D: 0.2431, G:0.2854



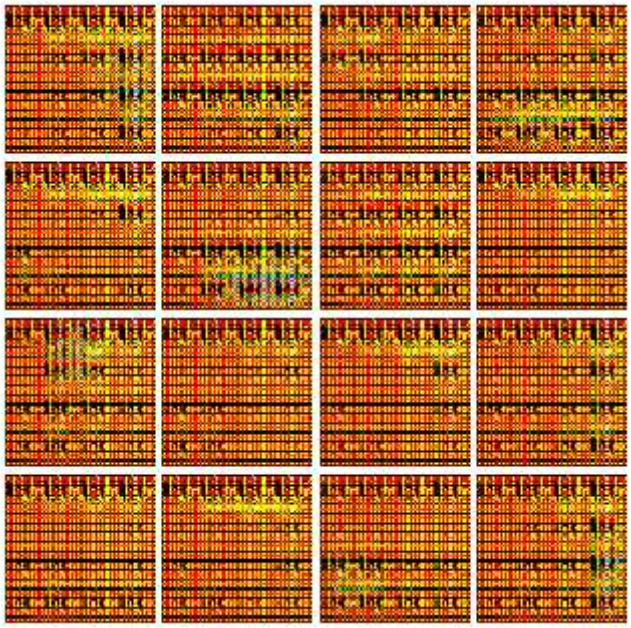
```
EPOCH:  8
Iter: 3500, D: 0.3165, G:0.5218
```

```
Iter: 3750, D: 0.2438, G:0.5267
```

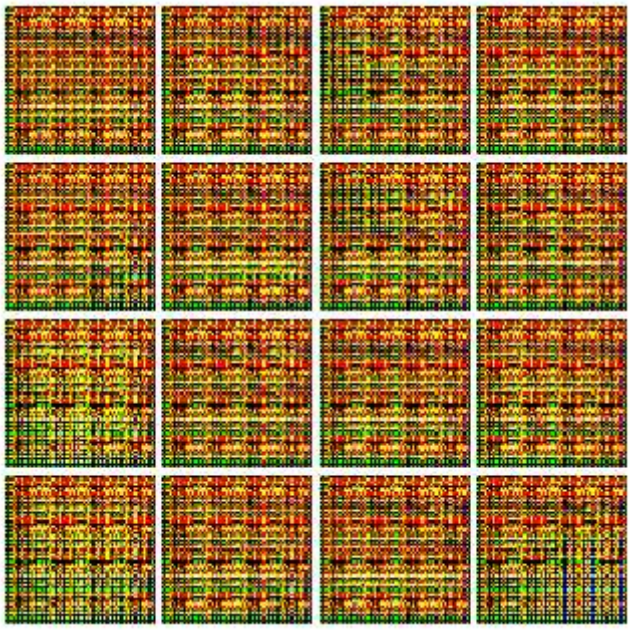




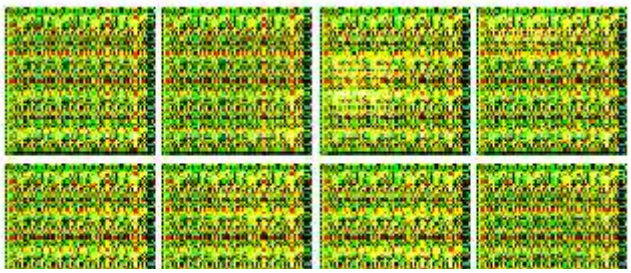
EPOCH: 9
Iter: 4000, D: 0.02441, G:0.5369

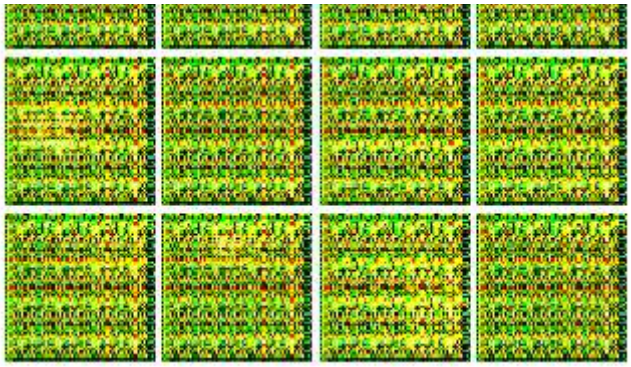


Iter: 4250, D: 0.02449, G:0.6189



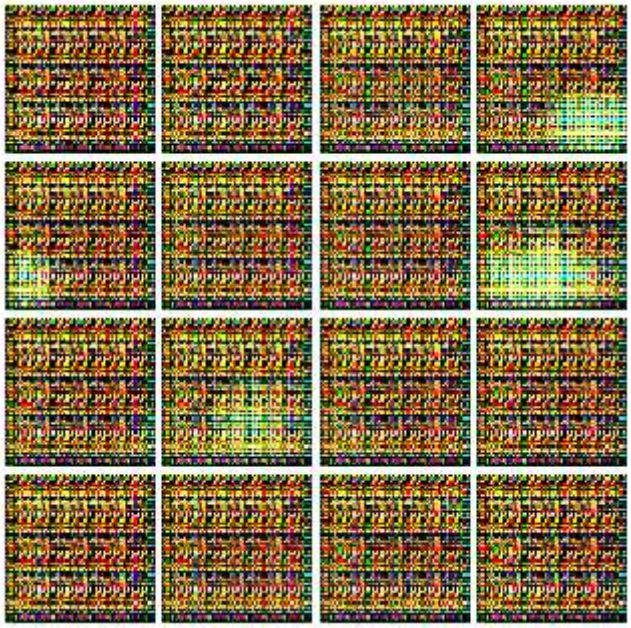
EPOCH: 10
Iter: 4500, D: 0.1512, G:0.3595





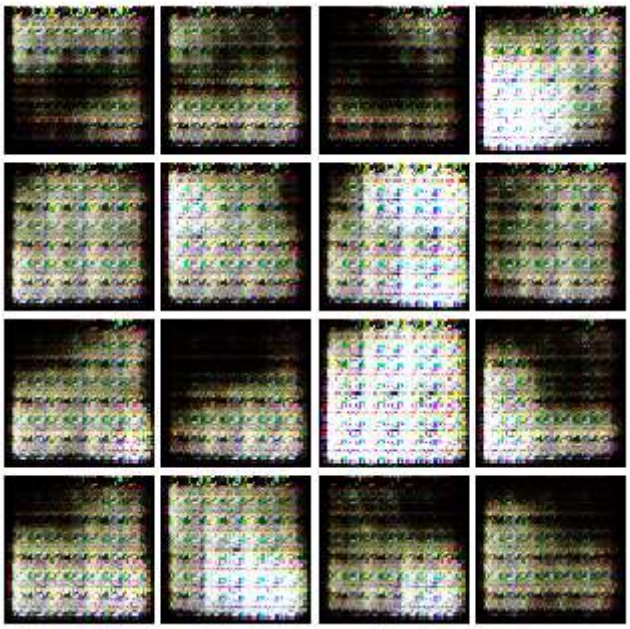
Iter: 4750, D: 0.01687, G:0.5119

EPOCH: 11
Iter: 5000, D: 0.02509, G:0.4689

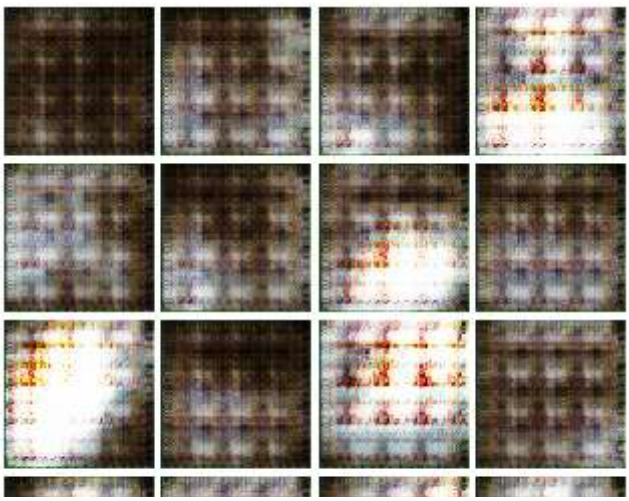


Iter: 5250, D: 0.05062, G:0.5853

EPOCH: 12
Iter: 5500, D: 0.1076, G:0.3877



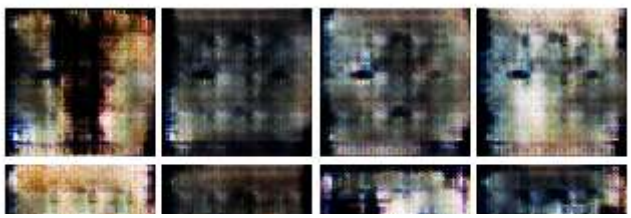
Iter: 5750, D: 0.1561, G:0.3191

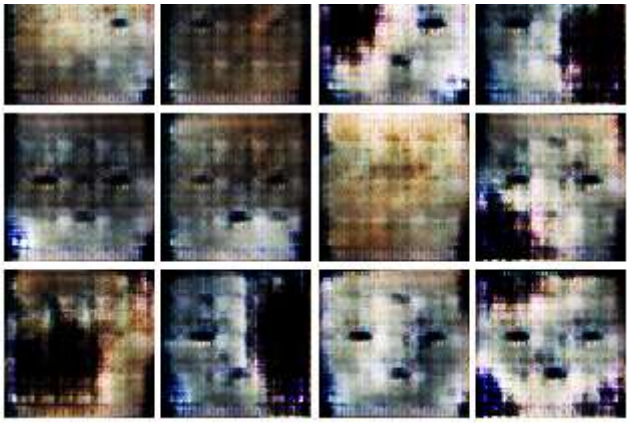


EPOCH: 13
Iter: 6000, D: 0.1195, G:0.4414

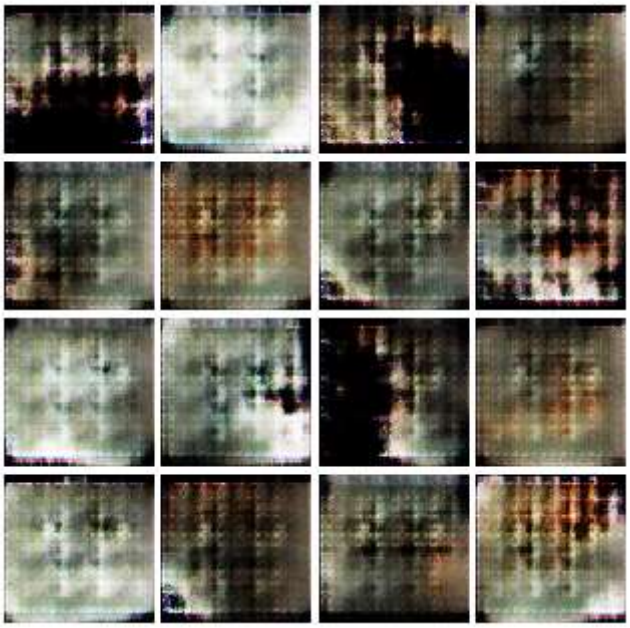
Iter: 6250, D: 0.1736, G:0.257

EPOCH: 14
Iter: 6500, D: 0.2123, G:0.279

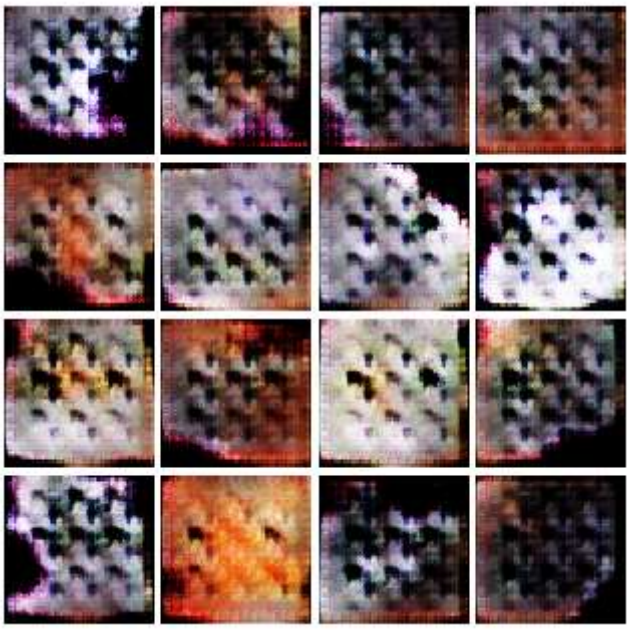




Iter: 6750, D: 0.2244, G:0.3181

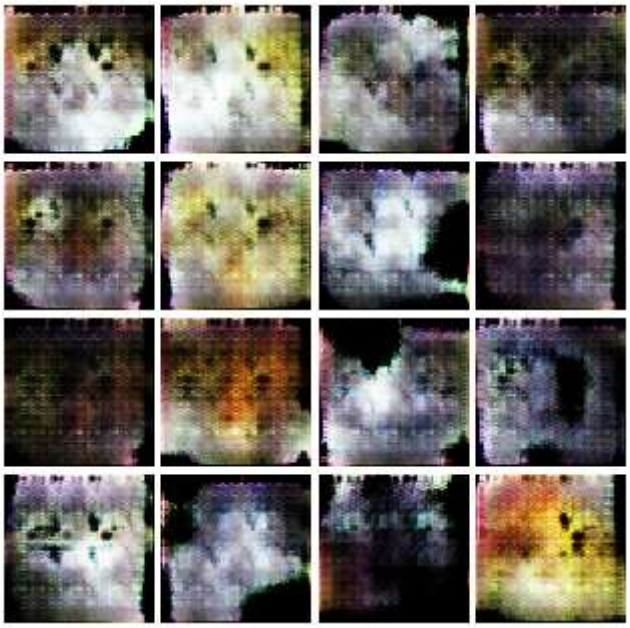


EPOCH: 15
Iter: 7000, D: 0.3048, G:0.5716

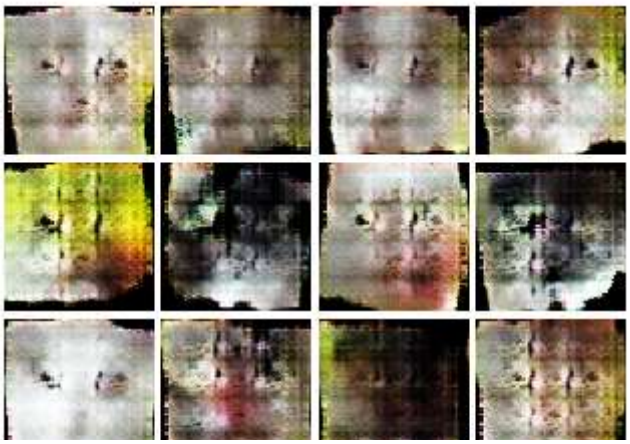


```
Iter: 7250, D: 0.2361, G:0.2867
```

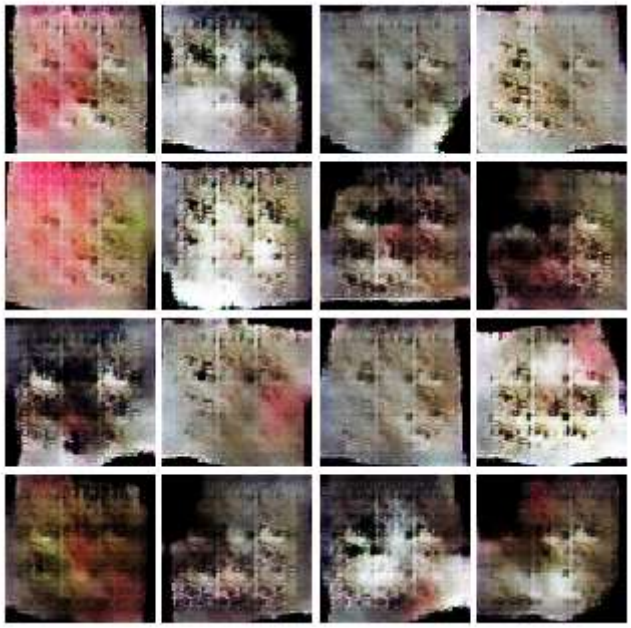
```
EPOCH:  16
Iter: 7500, D: 0.195, G:0.4263
```



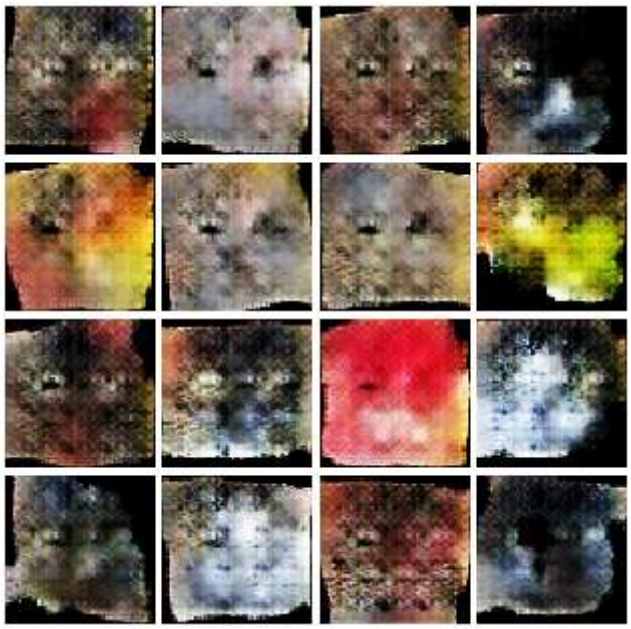
```
Iter: 7750, D: 0.1387, G:0.5116
```



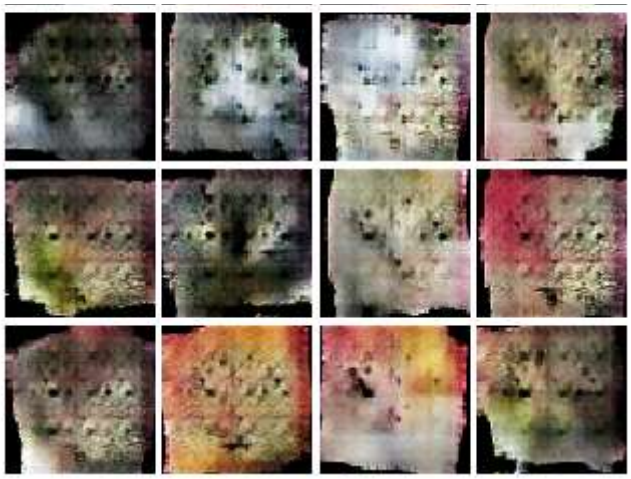
```
EPOCH:  17
Iter: 8000, D: 0.3599, G:0.2998
```



```
Iter: 8250, D: 1.251, G:0.368
```

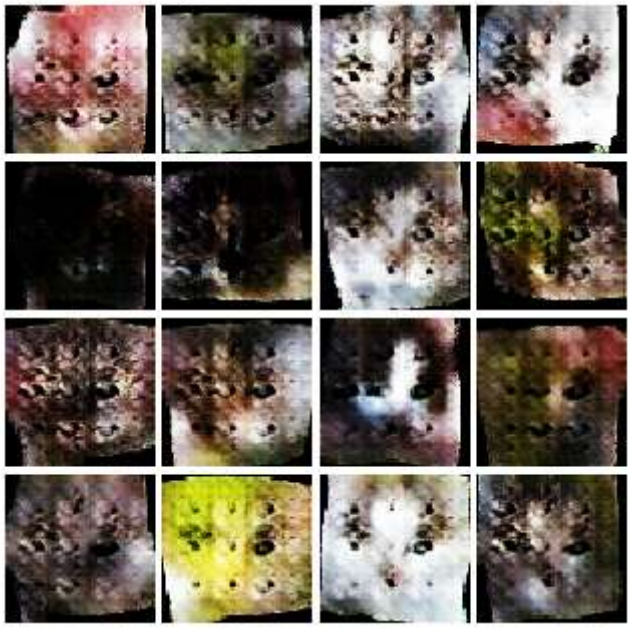


```
EPOCH:  18
Iter: 8500, D: 0.1721, G:0.3255
```

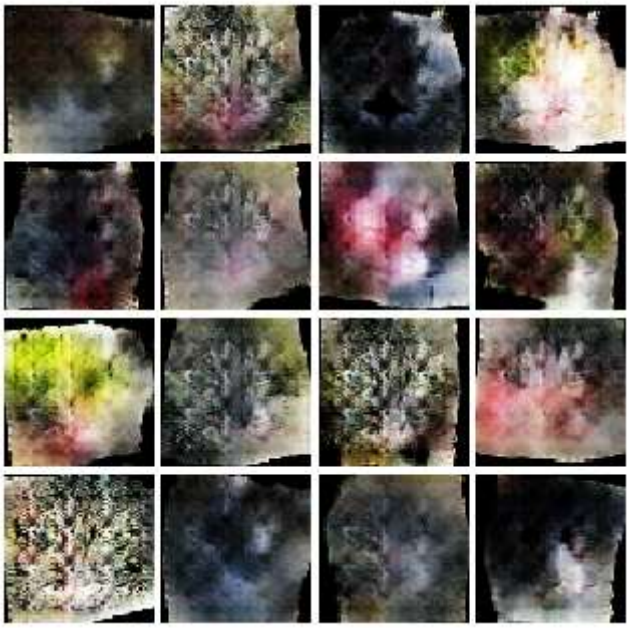


Iter: 8750, D: 0.5474, G:0.2671

EPOCH: 19
Iter: 9000, D: 0.1768, G:0.268

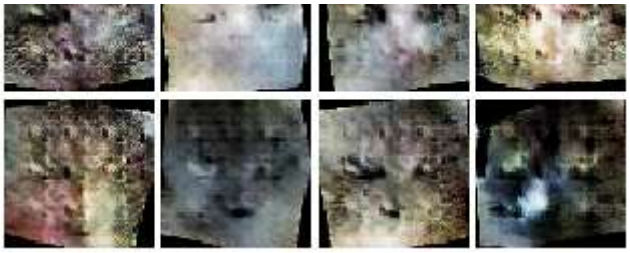


```
Iter: 9250, D: 0.2314, G:0.2239
```

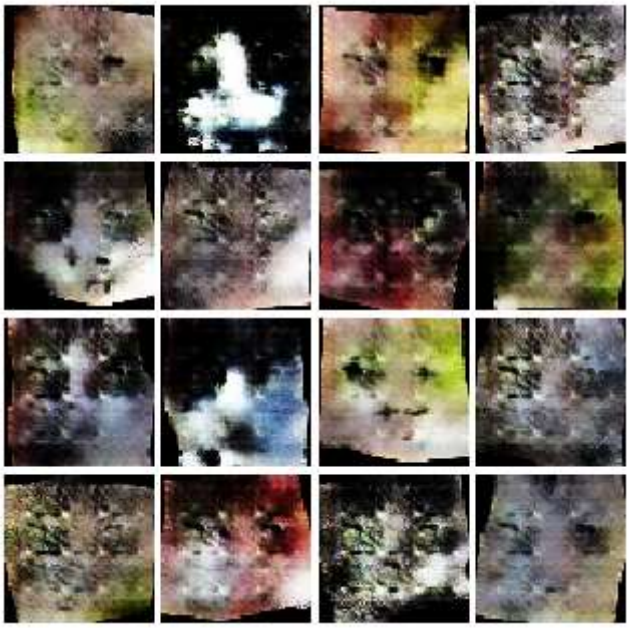


```
EPOCH:  20
Iter: 9500, D: 0.3198, G:0.2107
```

```
Iter: 9750, D: 0.2722, G:0.1819
```



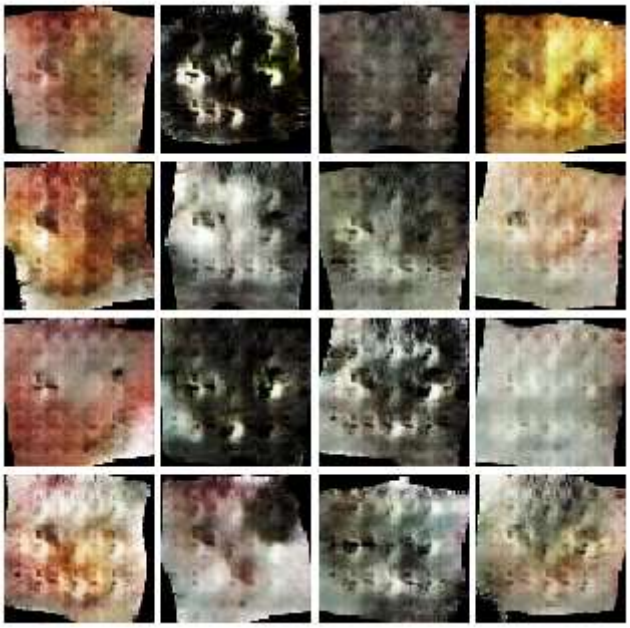
```
EPOCH:  21
Iter: 10000, D: 0.3473, G:0.2727
```



```
Iter: 10250, D: 0.2886, G:0.2131
```

```
EPOCH:  22
Iter: 10500, D: 0.2288, G:0.2429
```

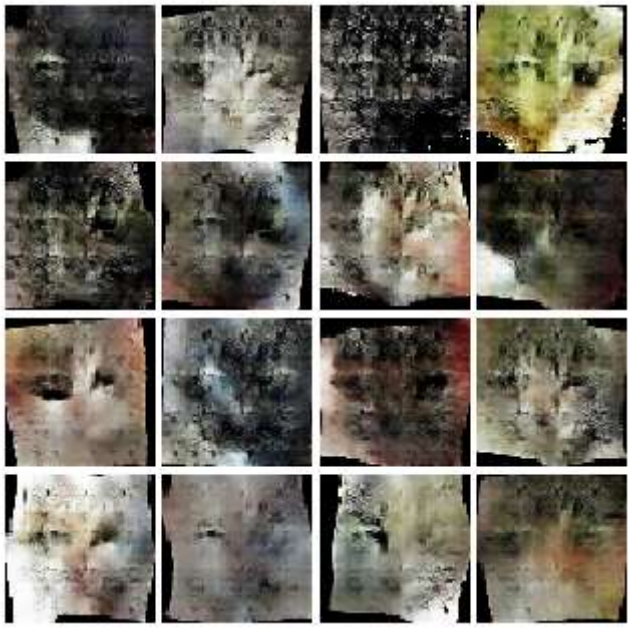
Iter: 10750, D: 0.2085, G:0.3348



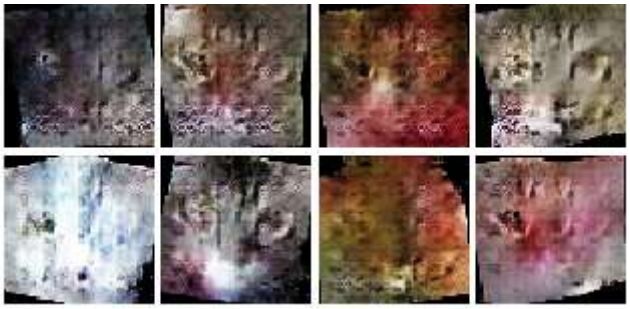
EPOCH: 23
Iter: 11000, D: 0.1942, G:0.1443

Iter: 11250, D: 0.2767, G:0.2884

EPOCH: 24
Iter: 11500, D: 0.1252, G:0.3581



Iter: 11750, D: 0.1488, G:0.2507



EPOCH: 25
Iter: 12000, D: 0.2228, G:0.3441

Iter: 12250, D: 0.1896, G:0.4519

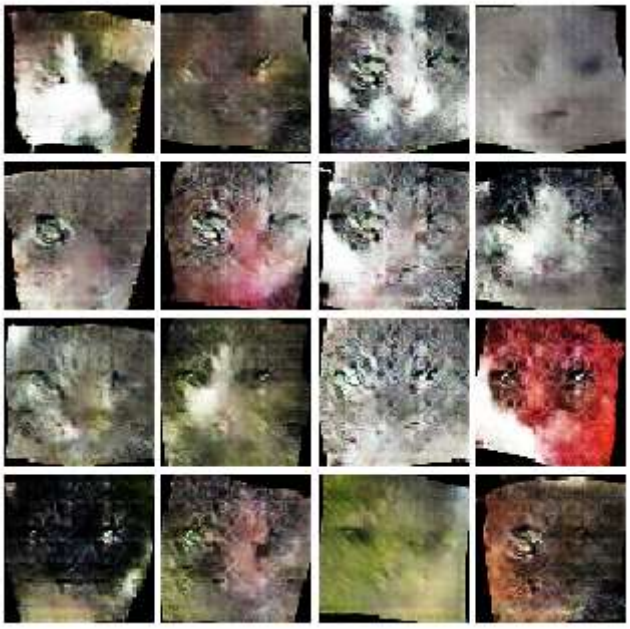
EPOCH: 26
Iter: 12500, D: 0.2261, G:0.7038

Iter: 12750, D: 0.2473, G:0.8095

EPOCH: 27
Iter: 13000, D: 0.1631, G:0.5141

Iter: 13250, D: 0.2342, G:0.4416

EPOCH: 28
Iter: 13500, D: 0.211, G:0.3536



Iter: 13750, D: 0.4228, G:1.301

EPOCH: 29
Iter: 14000, D: 0.31, G:0.2028

Iter: 14250, D: 0.1277, G:0.4235

EPOCH: 30
Iter: 14500, D: 0.26, G:0.799

```
Iter: 14750, D: 0.2193, G:0.6028
```

```
EPOCH:  31
Iter: 15000, D: 0.1245, G:0.5017
```

Iter: 15250, D: 0.2277, G:0.2731

EPOCH: 32
Iter: 15500, D: 0.1135, G:0.3613

EPOCH: 33
Iter: 15750, D: 0.07075, G:0.3894

Iter: 16000, D: 0.1484, G:0.6708

EPOCH: 34
Iter: 16250, D: 0.2082, G:0.133

```
Iter: 16500, D: 0.3036, G:0.2128
```

```
EPOCH:  35
Iter: 16750, D: 0.2497, G:0.3855
```

```
Iter: 17000, D: 0.176, G:0.4333
```

EPOCH: 36
Iter: 17250, D: 0.03817, G:0.4479

Iter: 17500, D: 0.07675, G:0.3079

EPOCH: 37
Iter: 17750, D: 0.08682, G:0.4113

Iter: 18000, D: 0.1552, G:0.2979

EPOCH: 38
Iter: 18250, D: 0.0697, G:0.4067

Iter: 18500, D: 0.1042, G:0.5755

EPOCH: 39
Iter: 18750, D: 0.09618, G:0.4593

Iter: 19000, D: 0.06422, G:0.4242

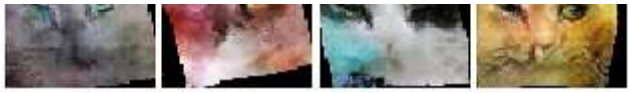
EPOCH: 40
Iter: 19250, D: 0.03282, G:0.4334

Iter: 19500, D: 0.04452, G:0.7536

EPOCH: 41
Iter: 19750, D: 0.06267, G:0.636

Iter: 20000, D: 0.1207, G:0.6796

EPOCH: 42
Iter: 20250, D: 0.04635, G:0.4948



```
Iter: 20500, D: 0.08243, G:0.4496
```

```
EPOCH:  43
Iter: 20750, D: 0.1925, G:0.9108
```

```
Iter: 21000, D: 0.1914, G:0.3235
```

EPOCH: 44
Iter: 21250, D: 0.09725, G:0.4987

Iter: 21500, D: 0.04754, G:0.5303

EPOCH: 45
Iter: 21750, D: 0.02453, G:0.3757

```
Iter: 22000, D: 0.02569, G:0.6016
```

```
EPOCH:  46
Iter: 22250, D: 0.1362, G:0.9052
```

Iter: 22500, D: 0.07824, G:0.4224

EPOCH: 47
Iter: 22750, D: 0.09832, G:0.3758

Iter: 23000, D: 0.08666, G:0.2883

EPOCH: 48
Iter: 23250, D: 0.09723, G:0.4557

Iter: 23500, D: 0.04062, G:0.7224

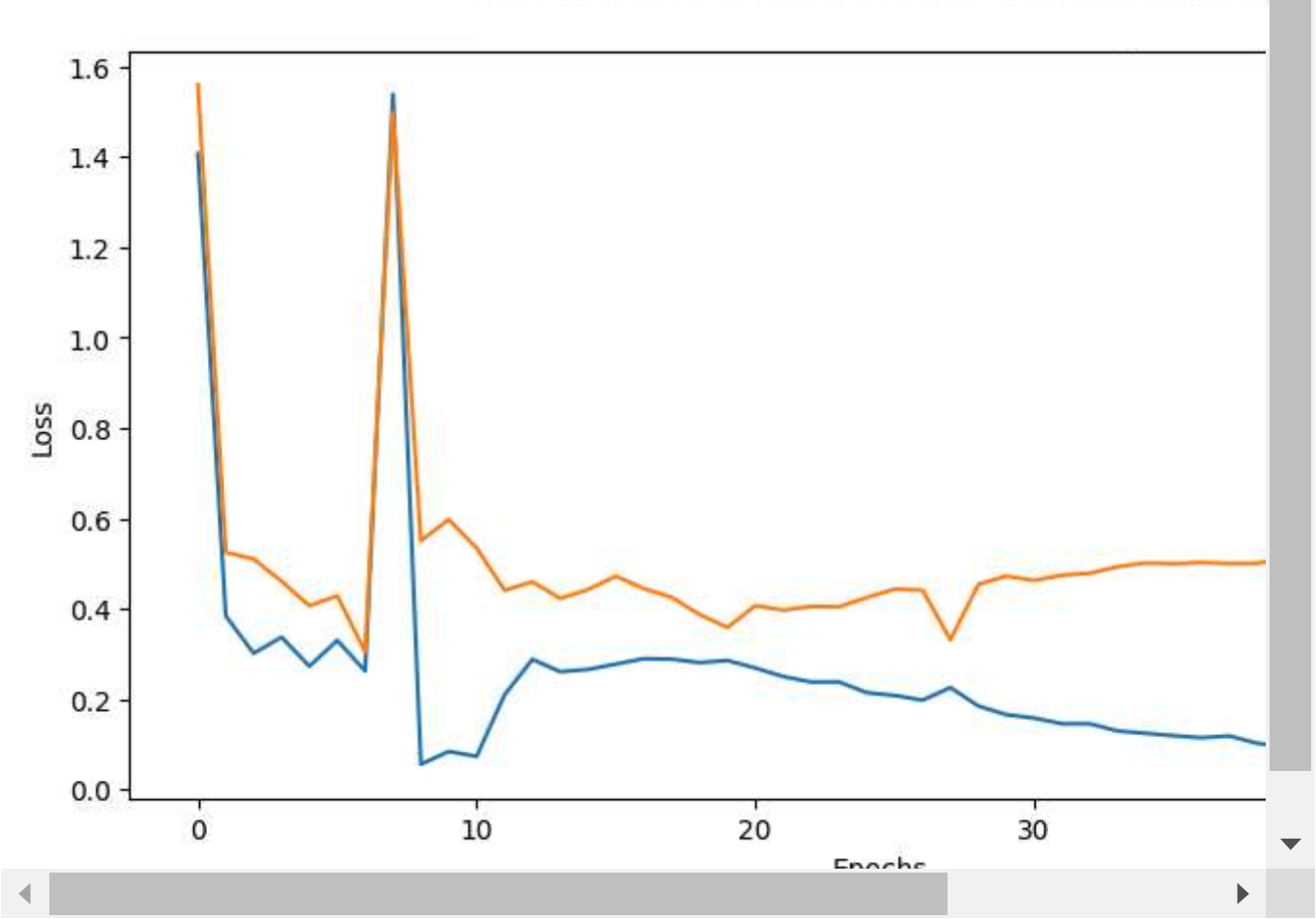
EPOCH: 49

```
Iter: 23750, D: 0.06536, G:0.4853
```

```
Iter: 24000, D: 0.03182, G:0.4533
```

```
EPOCH:  50
Iter: 24250, D: 0.07781, G:0.3245
```

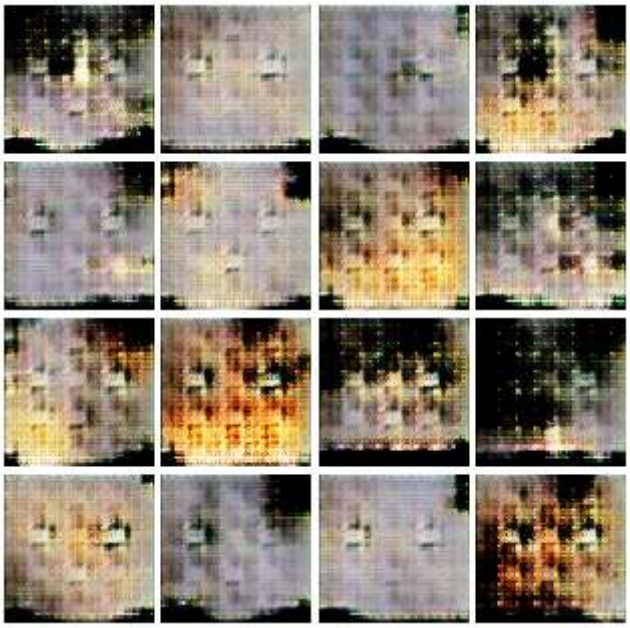
```
Iter: 24500, D: 0.07277, G:0.3537
```



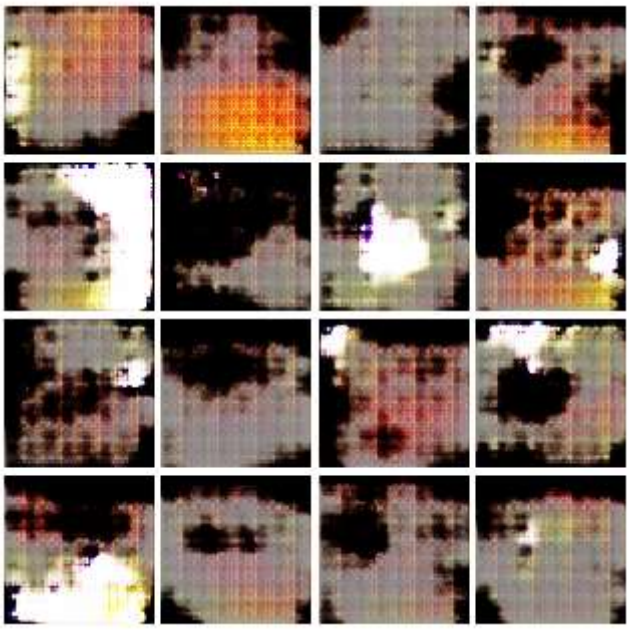
WGAN Loss

```
from losses import wg_generator_loss, wg_discriminator_loss


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, wg_discriminator_loss,
          wg_generator_loss, num_epochs=NUM_EPOCHS, show_every=250,
          batch_size=batch_size, train_loader=cat_loader_train, device=device)
```

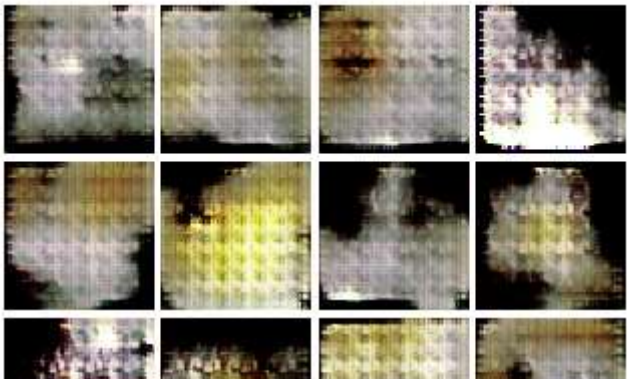
```
EPOCH:  1
Iter: 0, D: -1.06, G:6.693
```

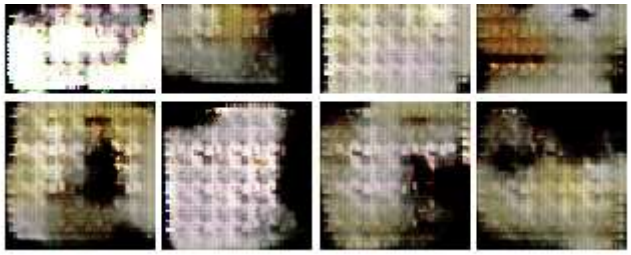


```
Iter: 250, D: -789.3, G:3.611e+03
```

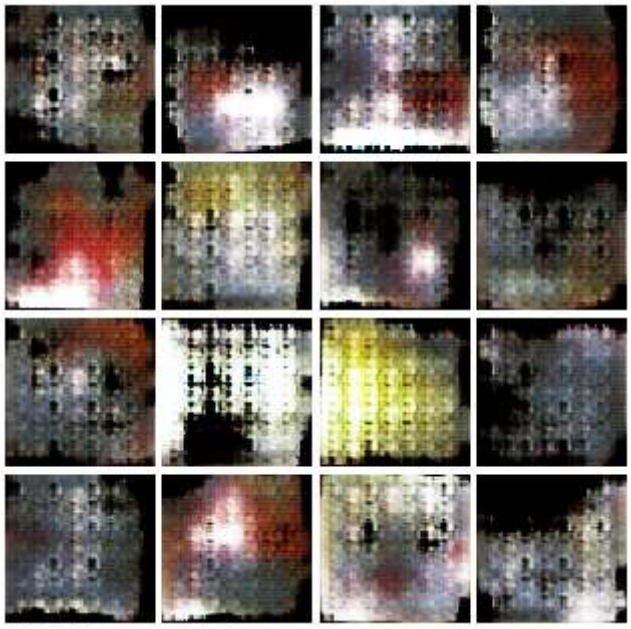


```
EPOCH:  2
Iter: 500, D: -1.763e+04, G:-1.548e+03
```

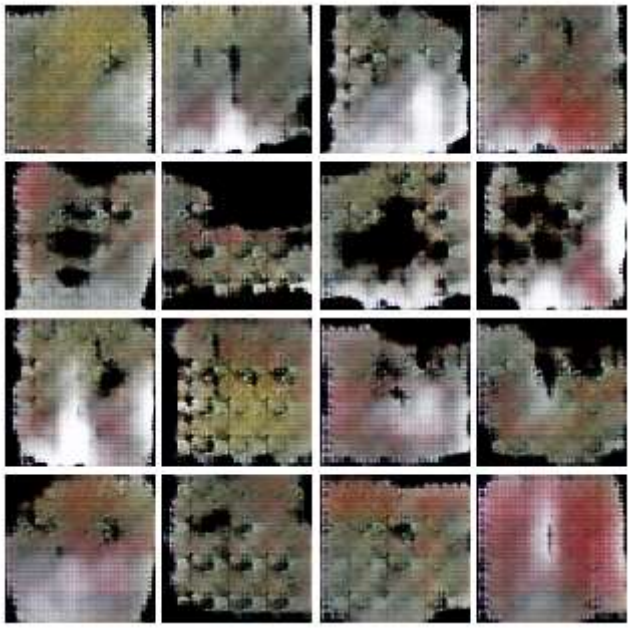




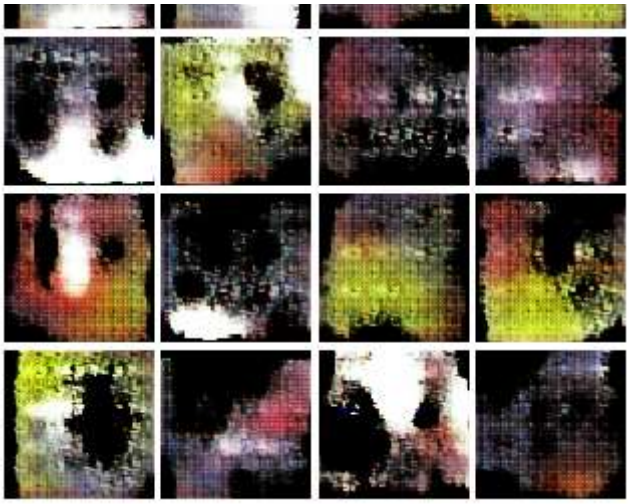
```
Iter: 750, D: -7.65e+03, G:2.425e+04
```



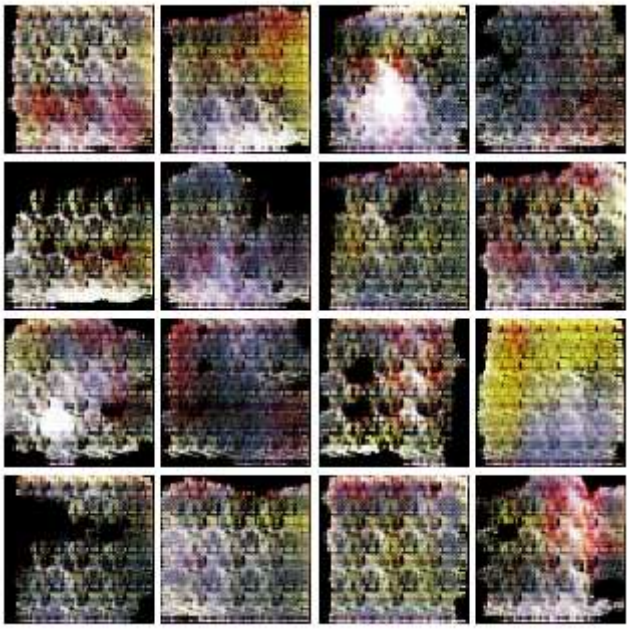
```
EPOCH:  3
Iter: 1000, D: -8.674e+04, G:4.58e+04
```



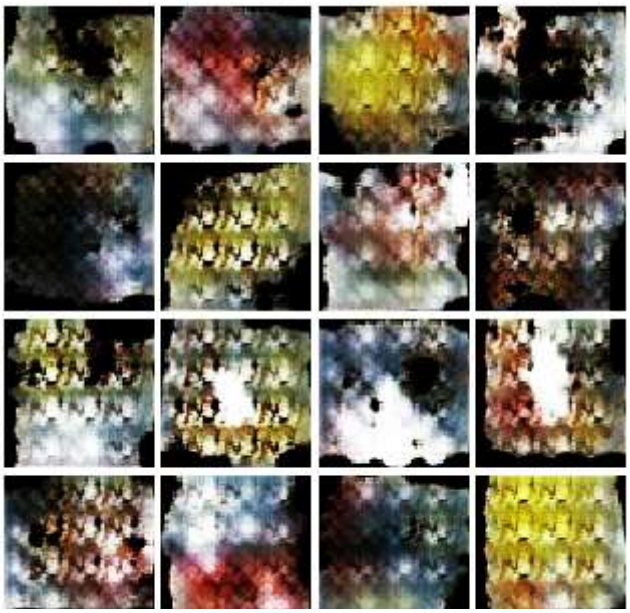
```
Iter: 1250, D: -1.3e+05, G:6.787e+04
```



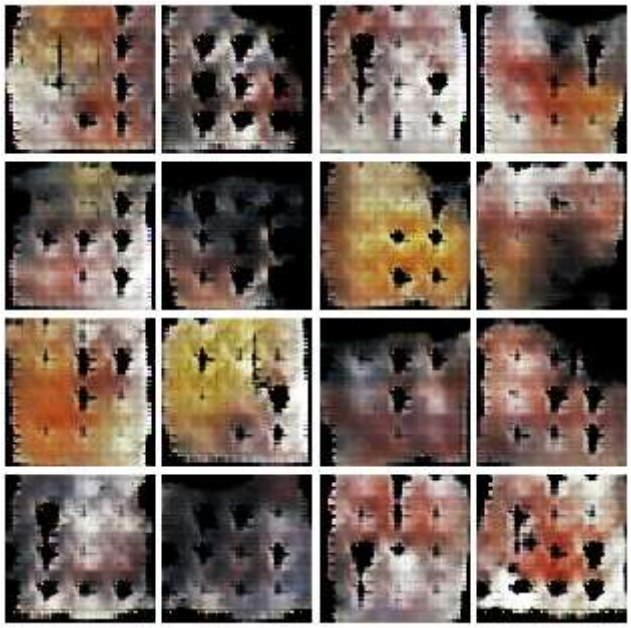
EPOCH: 4
Iter: 1500, D: -1.467e+04, G:-1.511e+04

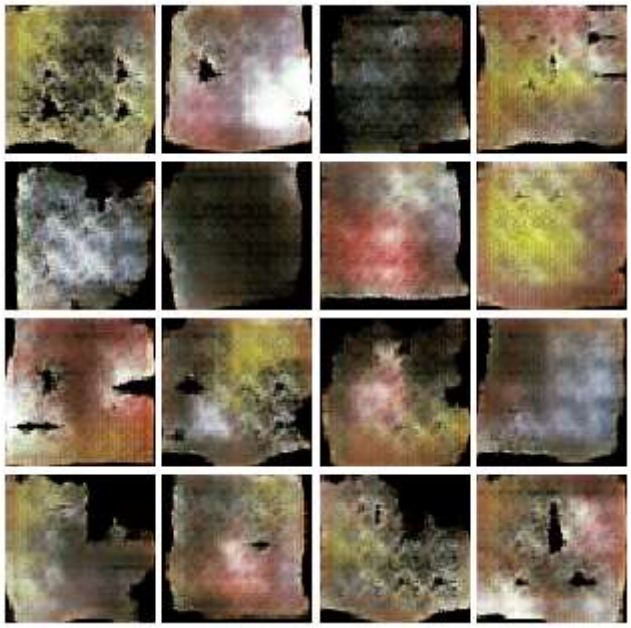


Iter: 1750, D: -2.216e+05, G:1.173e+05

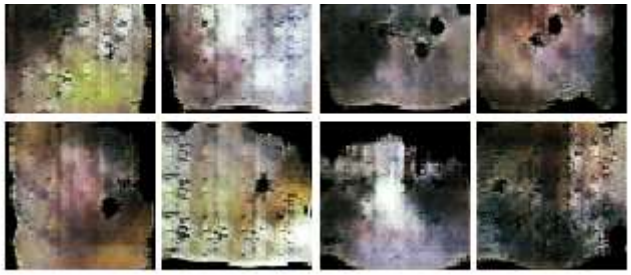


```
EPOCH:  5
Iter: 2000, D: -2.967e+05, G:1.557e+05
```

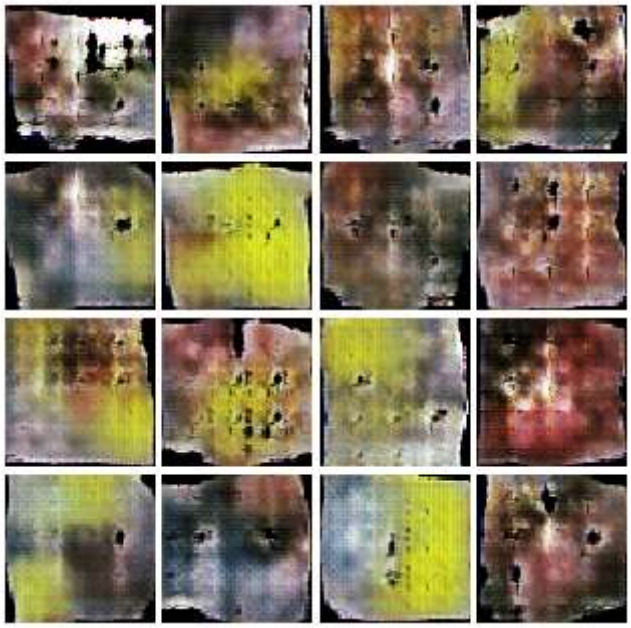




```
EPOCH:  6
Iter: 2500, D: -4.003e+05, G:2.146e+05
```

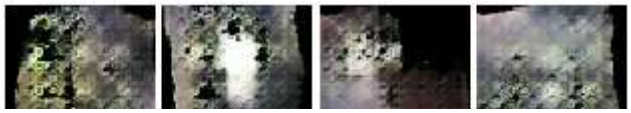


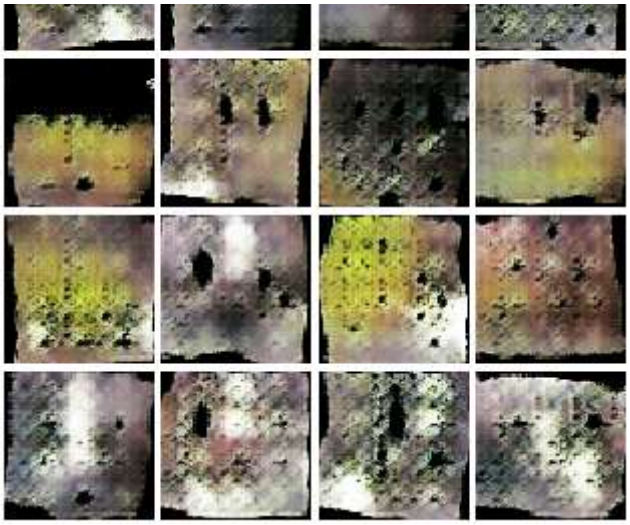
Iter: 2750, D: -4.452e+05, G:2.366e+05



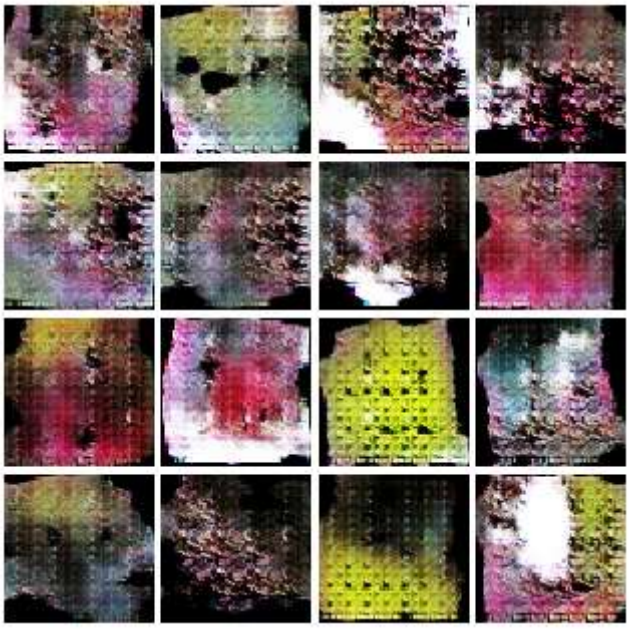
EPOCH: 7
Iter: 3000, D: -5.89e+05, G:3.091e+05

Iter: 3250, D: -6.439e+05, G:3.462e+05





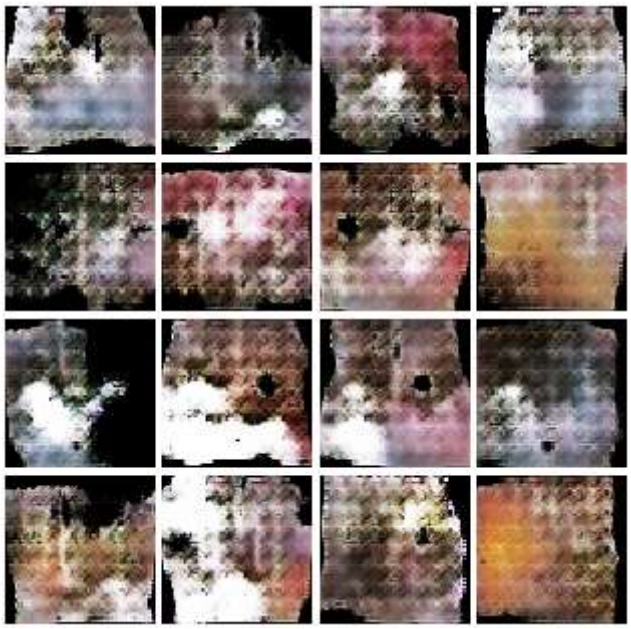
EPOCH: 8
Iter: 3500, D: -7.846e+05, G:4.099e+05



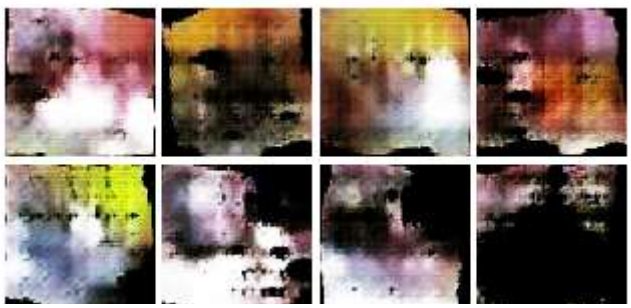
Iter: 3750, D: -8.375e+04, G:3.706e+05

```
EPOCH:  9
Iter: 4000, D: -9.725e+05, G:5.095e+05
```

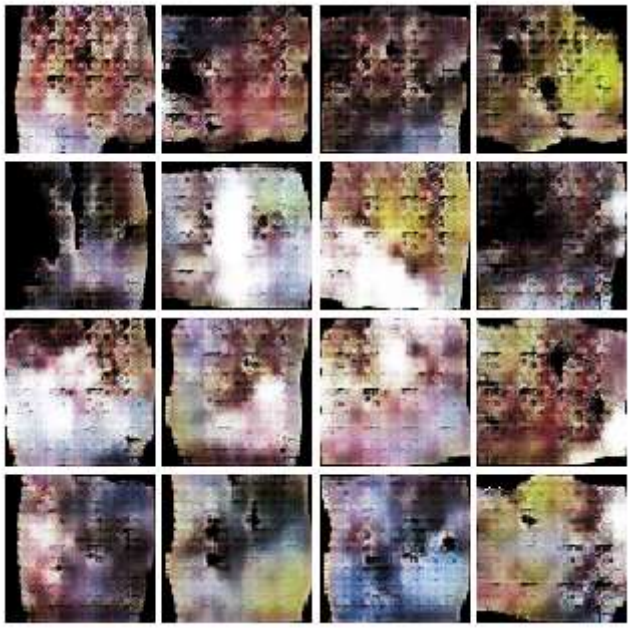
```
Iter: 4250, D: -1.067e+06, G:5.636e+05
```



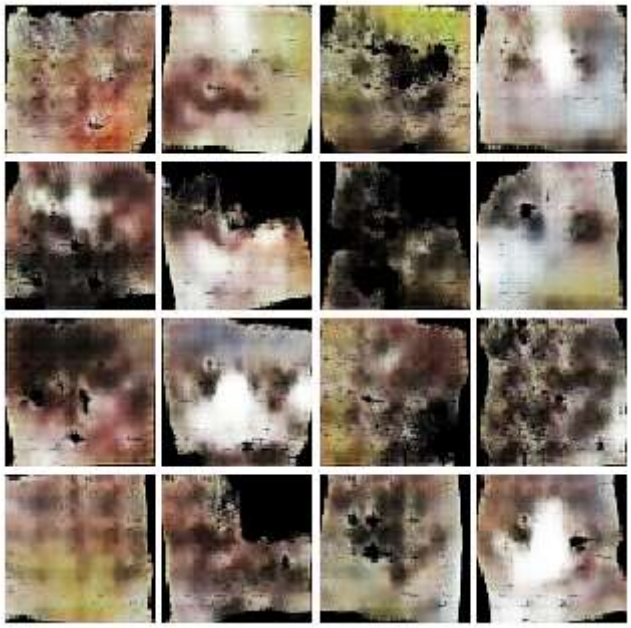
```
EPOCH:  10
Iter: 4500, D: -2.109e+05, G:4.849e+05
```



Iter: 4750, D: -1.302e+06, G:6.811e+05



EPOCH: 11
Iter: 5000, D: -1.448e+06, G:7.581e+05



Iter: 5250, D: -1.938e+05, G:1.64e+05

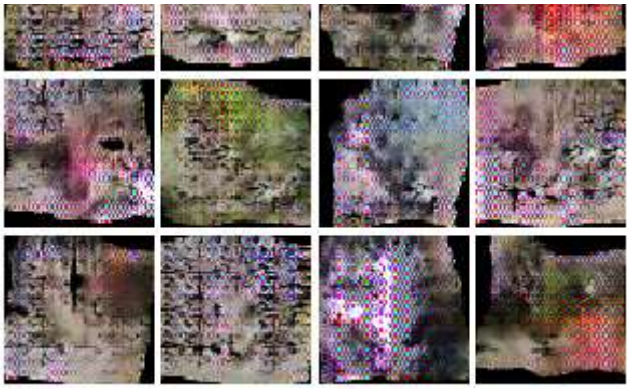
```
EPOCH:  12
Iter: 5500, D: -1.25e+06, G:7.536e+05
```

```
Iter: 5750, D: -6.318e+05, G:8.336e+05
```

EPOCH: 13
Iter: 6000, D: -1.79e+06, G:9.638e+05

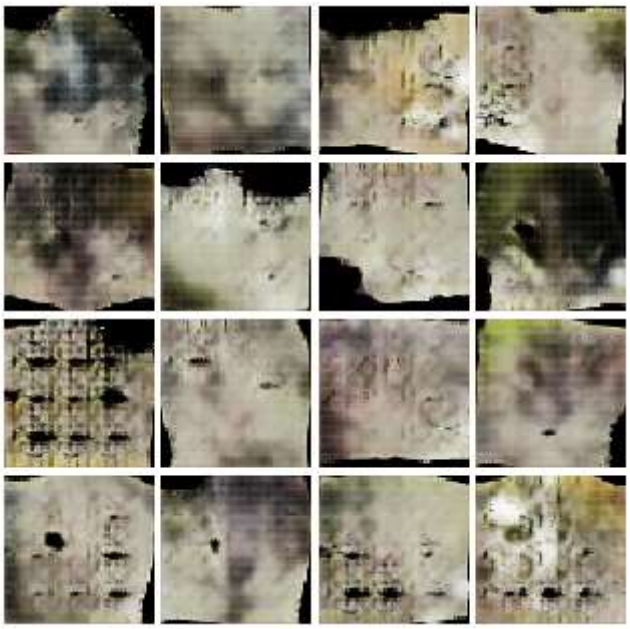
Iter: 6250, D: -2.076e+06, G:1.087e+06

EPOCH: 14
Iter: 6500, D: -2.247e+06, G:1.175e+06

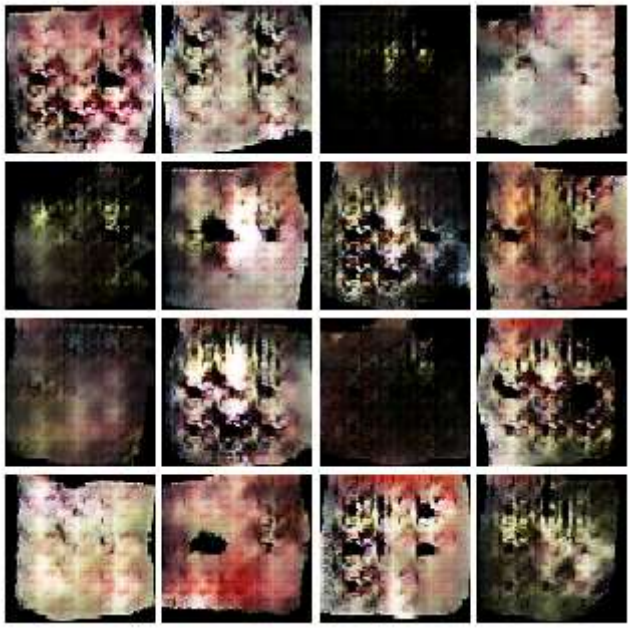


Iter: 6750, D: -2.416e+06, G:1.264e+06

EPOCH: 15
Iter: 7000, D: -2.588e+06, G:1.356e+06



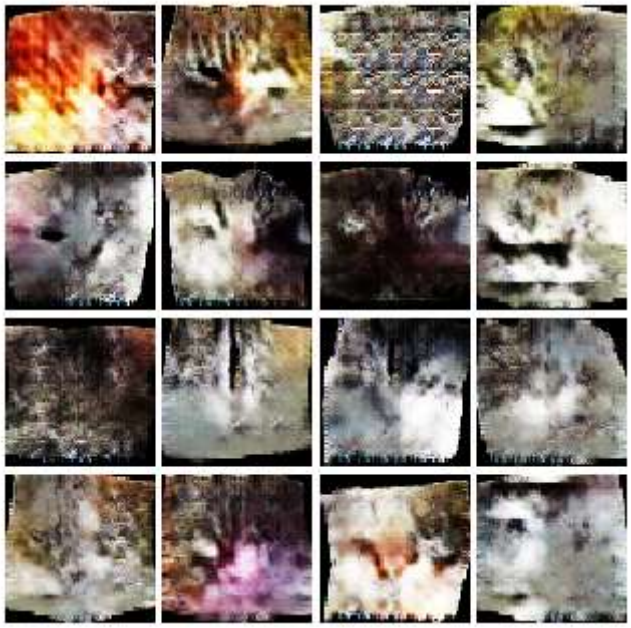
Iter: 7250, D: -1.511e+05, G:-6.935e+05



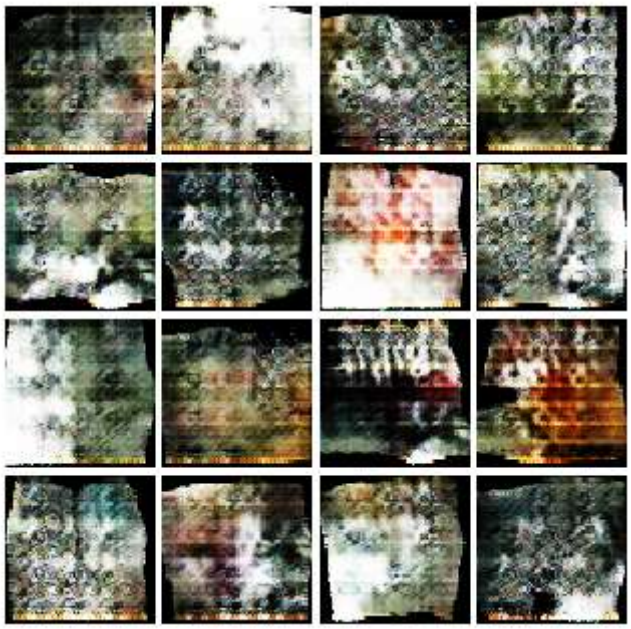
```
EPOCH:  16
Iter: 7500, D: -2.408e+06, G:1.416e+06
```

```
Iter: 7750, D: -2.776e+06, G:1.322e+06
```

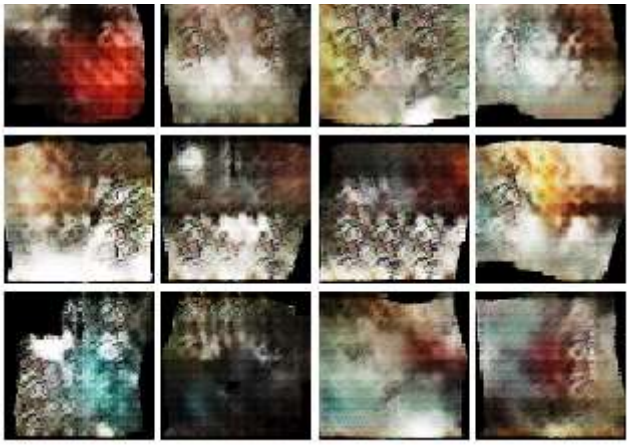
EPOCH: 17
Iter: 8000, D: -3.151e+06, G:1.654e+06



Iter: 8250, D: -3.345e+06, G:1.753e+06

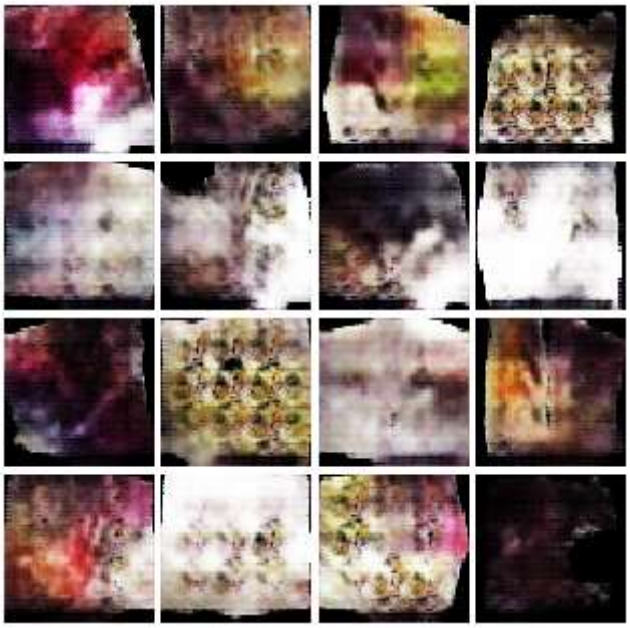


EPOCH: 18
Iter: 8500, D: -3.528e+06, G:1.861e+06



Iter: 8750, D: -3.756e+06, G:1.971e+06

EPOCH: 19
Iter: 9000, D: -3.941e+06, G:2.061e+06

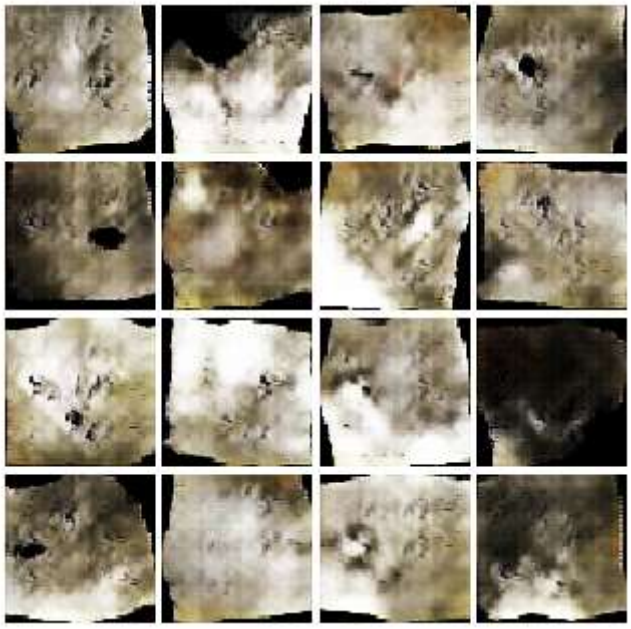


```
Iter: 9250, D: -4.173e+06, G:2.19e+06
```

```
EPOCH:  20
Iter: 9500, D: -4.382e+06, G:2.3e+06
```

```
Iter: 9750, D: -4.591e+06, G:2.409e+06
```

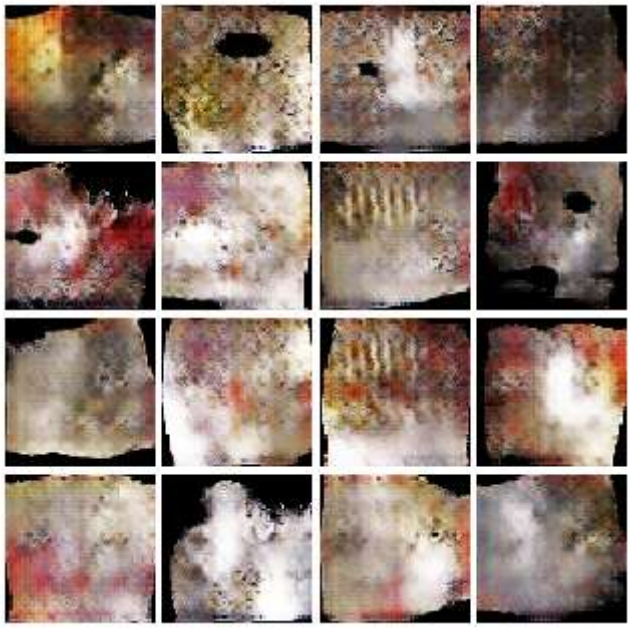
```
EPOCH:  21
Iter: 10000, D: -4.593e+06, G:2.434e+06
```



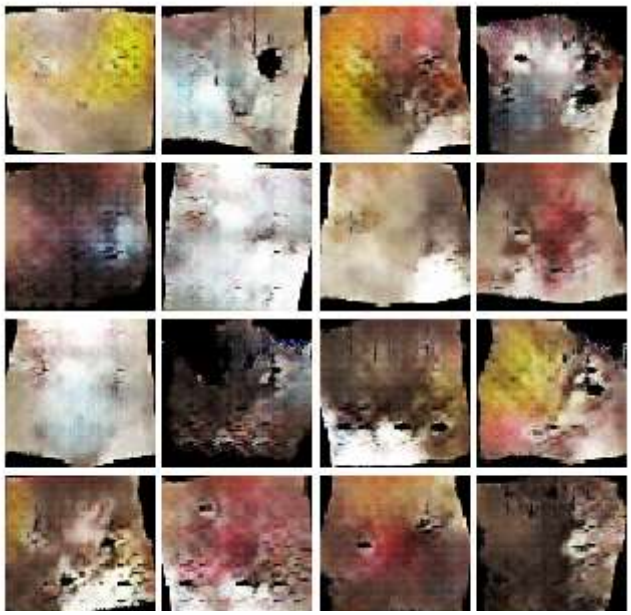
```
Iter: 10250, D: -4.895e+06, G:2.582e+06
```

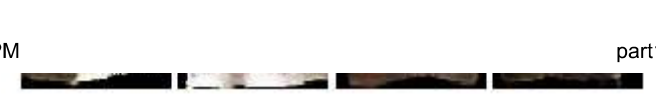
```
EPOCH:  22
Iter: 10500, D: -5.158e+06, G:2.711e+06
```

Iter: 10750, D: -5.311e+06, G:2.803e+06



EPOCH: 23
Iter: 11000, D: -1.321e+06, G:2.152e+06



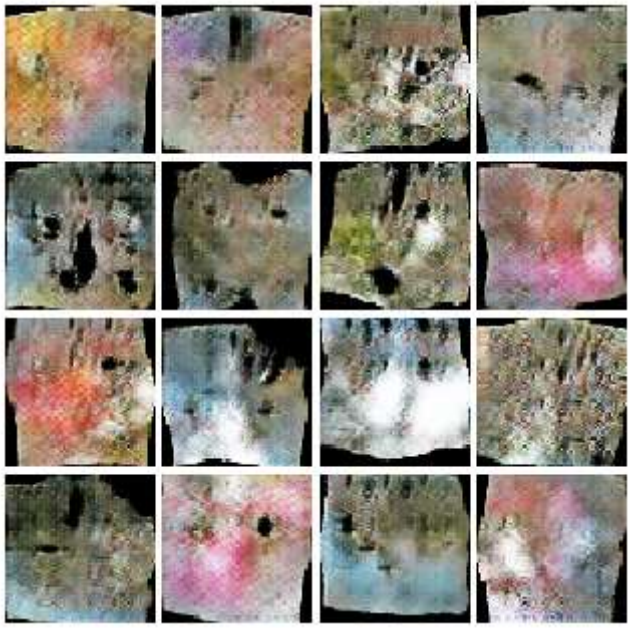


```
Iter: 11250, D: -4.984e+06, G:2.826e+06
```

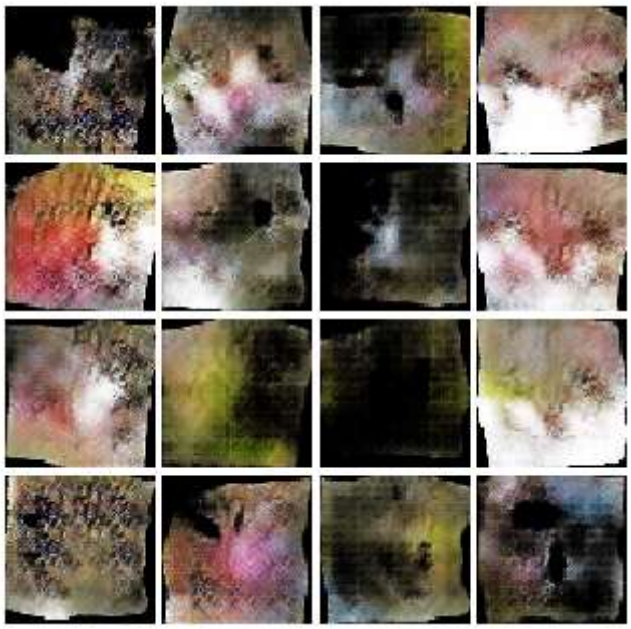
```
EPOCH:  24
Iter: 11500, D: -5.916e+06, G:3.113e+06
```

```
Iter: 11750, D: -6.187e+06, G:3.256e+06
```

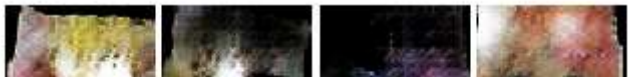
EPOCH: 25
Iter: 12000, D: -6.405e+06, G:3.382e+06

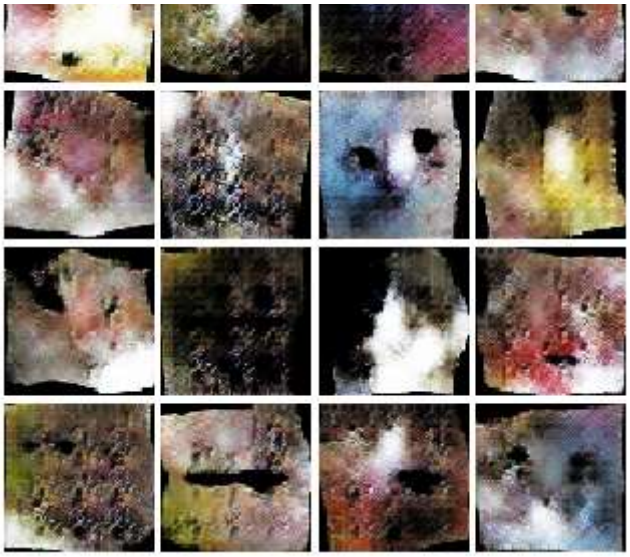


Iter: 12250, D: -6.733e+06, G:3.545e+06

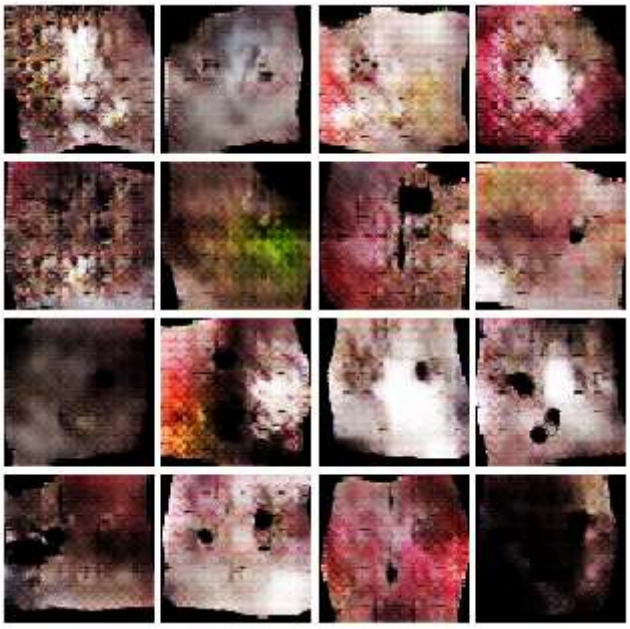


EPOCH: 26
Iter: 12500, D: -6.966e+06, G:3.681e+06

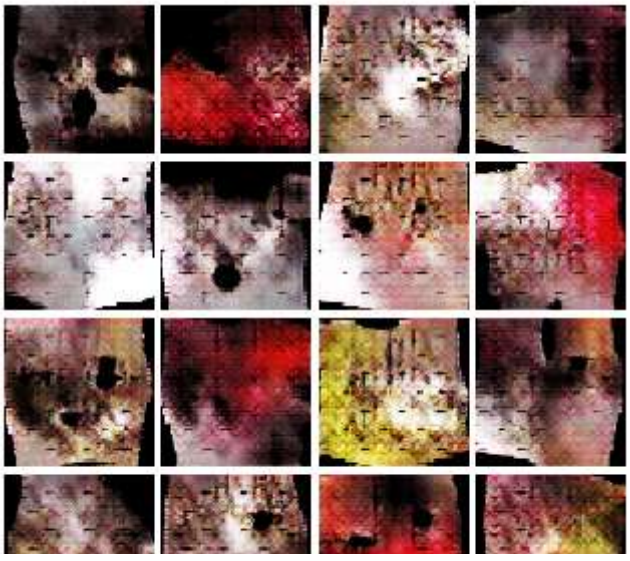




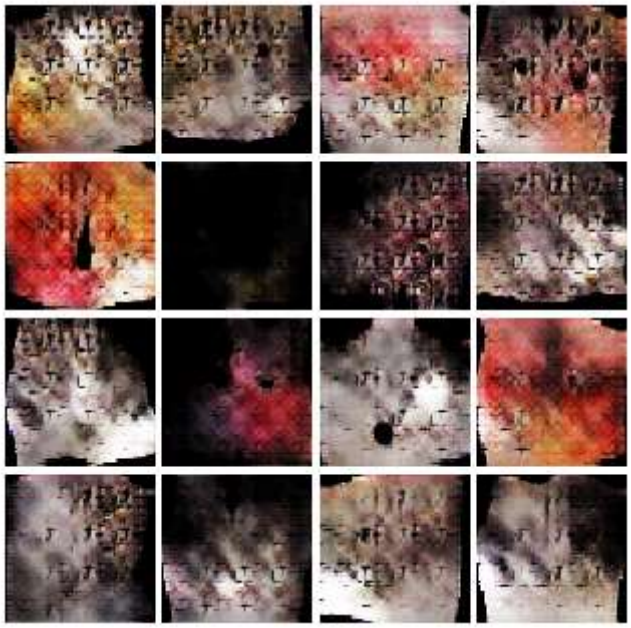
Iter: 12750, D: -7.249e+06, G:3.815e+06



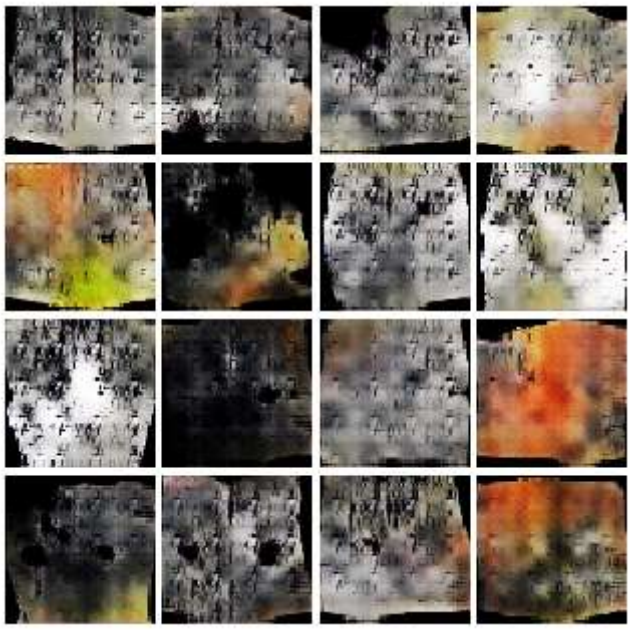
EPOCH: 27
Iter: 13000, D: -7.521e+06, G:3.957e+06



Iter: 13250, D: -7.767e+06, G:4.087e+06

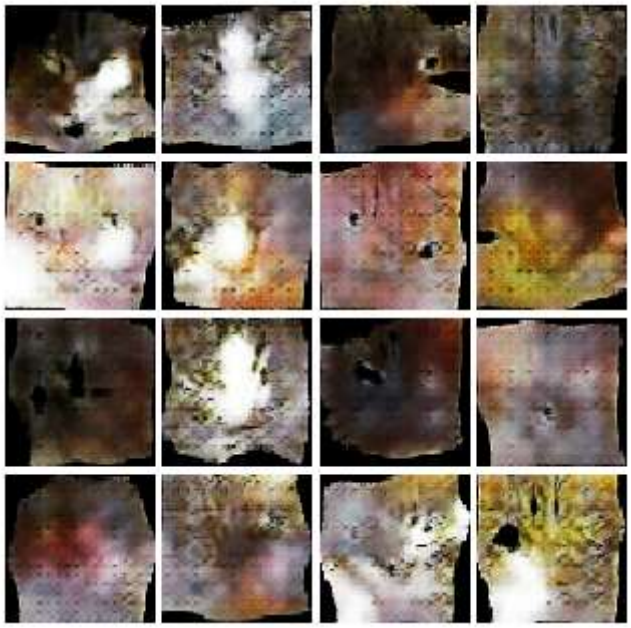


EPOCH: 28
Iter: 13500, D: -8.01e+06, G:4.225e+06

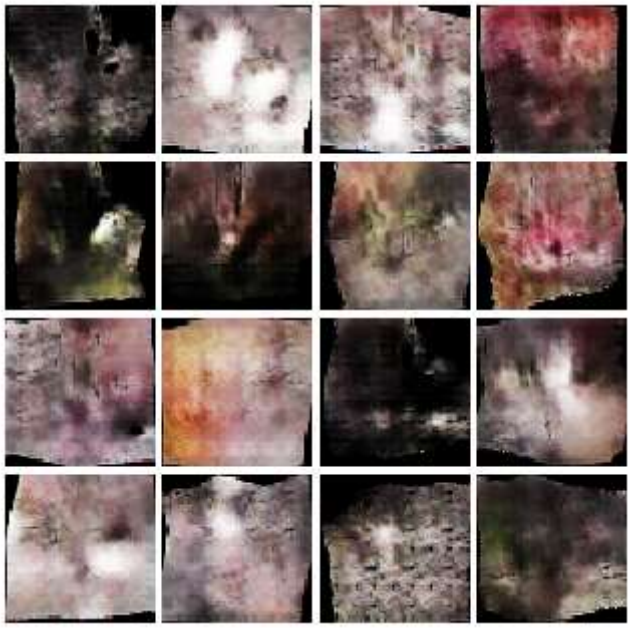


Iter: 13750, D: -2.116e+06, G:2.05e+06

EPOCH: 29
Iter: 14000, D: -7.864e+06, G:4.285e+06



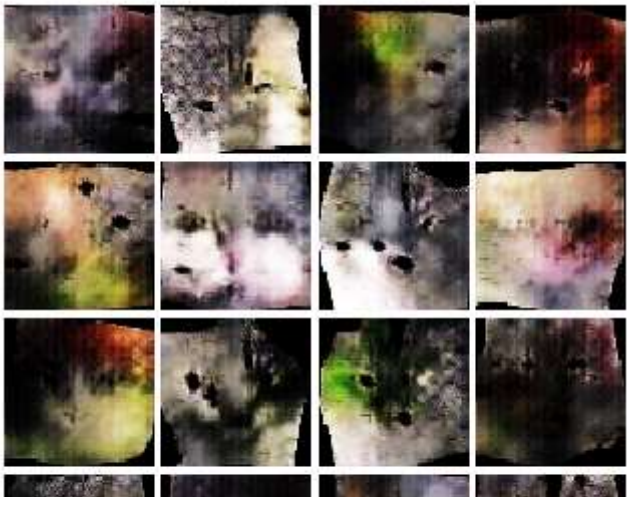
Iter: 14250, D: -8.628e+06, G:4.562e+06



EPOCH: 30
Iter: 14500, D: -8.967e+06, G:4.719e+06

Iter: 14750, D: -4.127e+06, G:3.559e+06

EPOCH: 31
Iter: 15000, D: -3.577e+06, G:-1.131e+06



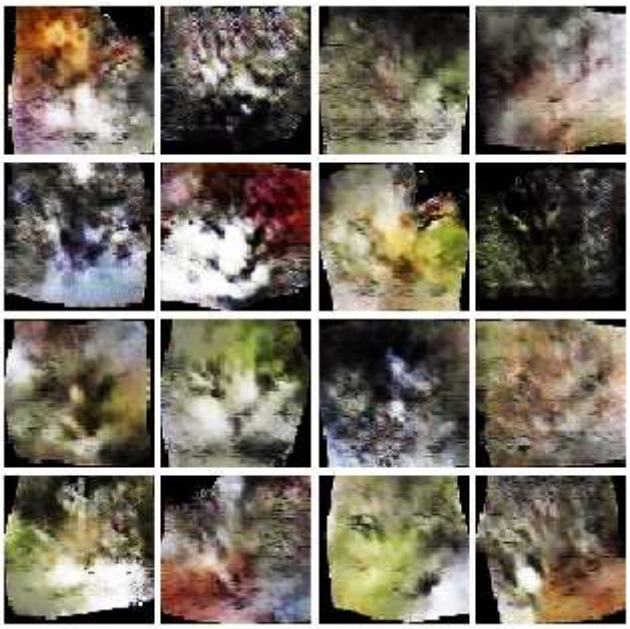
Iter: 15250, D: -8.457e+06, G:4.861e+06

EPOCH: 32
Iter: 15500, D: -9.983e+06, G:5.264e+06

EPOCH: 33
Iter: 15750, D: -1.032e+07, G:5.446e+06

Iter: 16000, D: -7.395e+06, G:4.499e+06

EPOCH: 34
Iter: 16250, D: -9.398e+06, G:4.252e+06

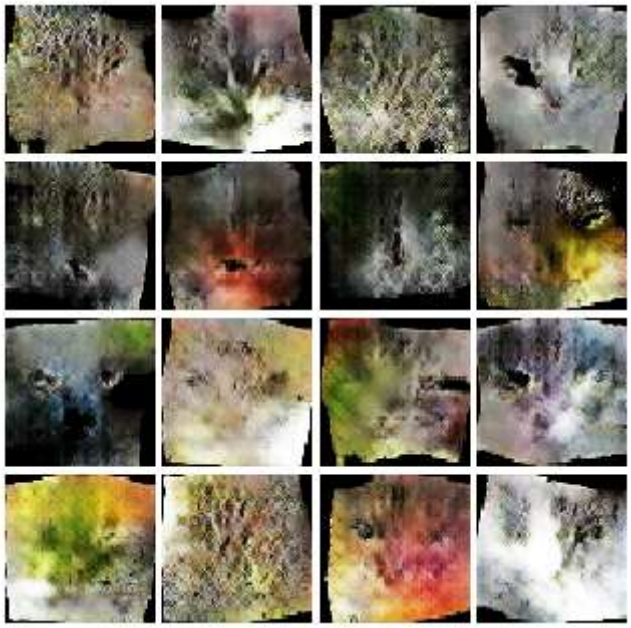


Iter: 16500, D: -1.116e+07, G:5.878e+06

EPOCH: 35
Iter: 16750, D: -5.908e+05, G:5.681e+06

Iter: 17000, D: -1.182e+07, G:6.252e+06

```
EPOCH:  36
Iter: 17250, D: -1.19e+07, G:6.329e+06
```

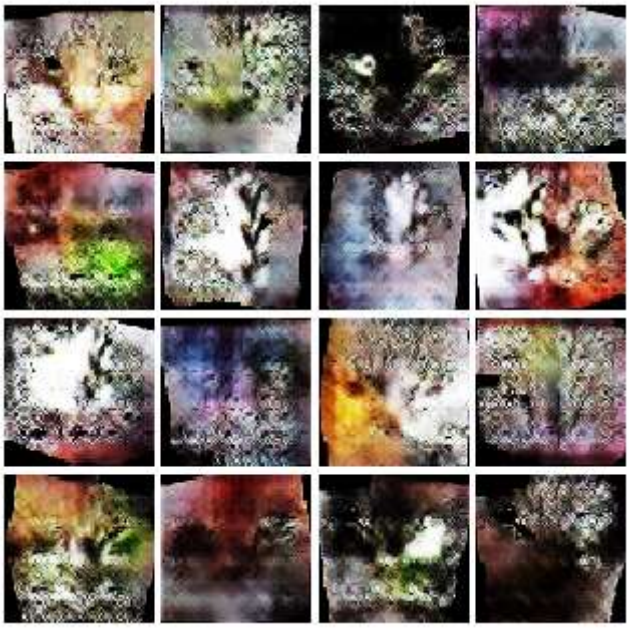


```
Iter: 17500, D: -1.255e+07, G:6.617e+06
```

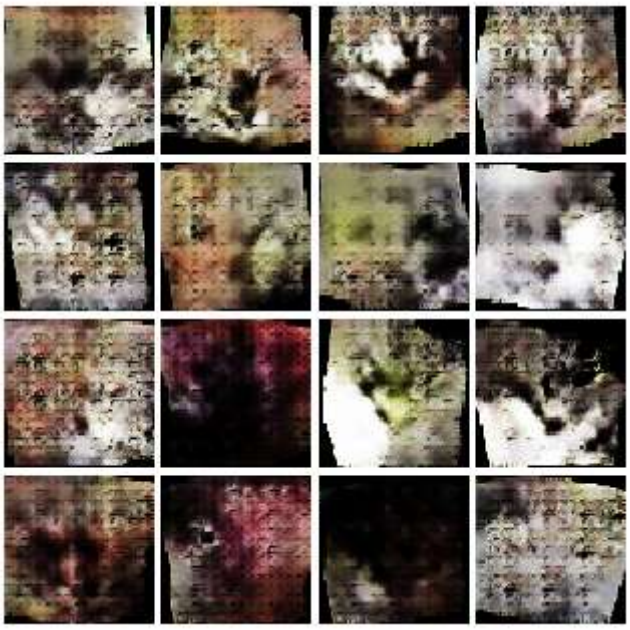
```
EPOCH:  37
Iter: 17750, D: -1.292e+07, G:6.81e+06
```

Iter: 18000, D: -1.328e+07, G:7.004e+06

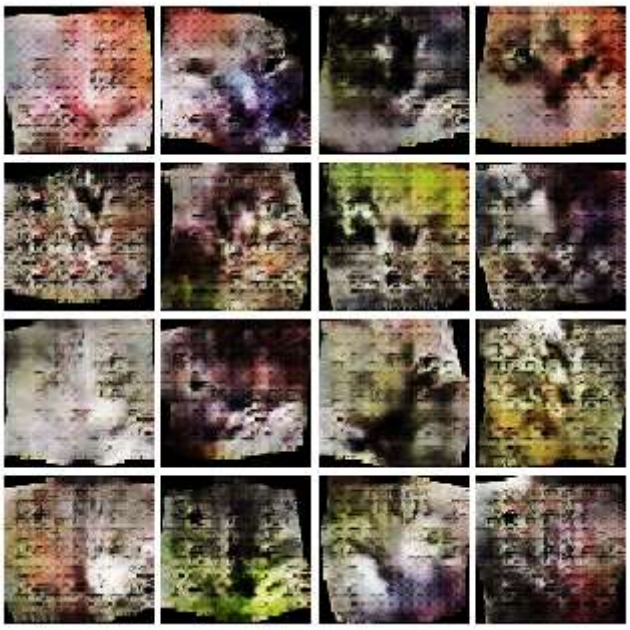
EPOCH: 38
Iter: 18250, D: -1.362e+07, G:7.186e+06



```
Iter: 18500, D: -1.401e+07, G:7.394e+06
```



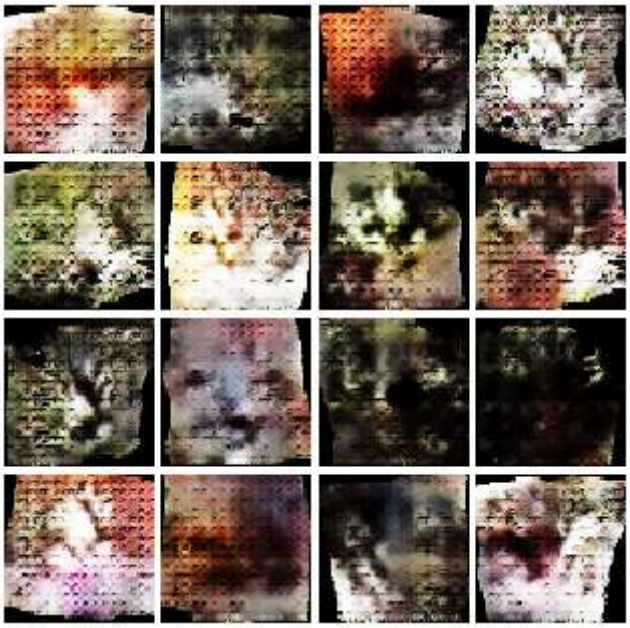
```
EPOCH:  39
Iter: 18750, D: -1.439e+07, G:7.588e+06
```



```
Iter: 19000, D: -1.475e+07, G:7.781e+06
```

EPOCH: 40
Iter: 19250, D: -1.511e+07, G:7.972e+06

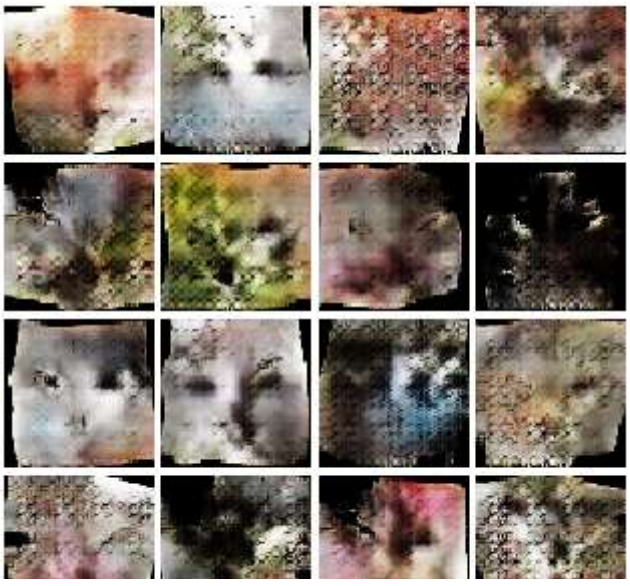
Iter: 19500, D: -1.548e+07, G:8.167e+06



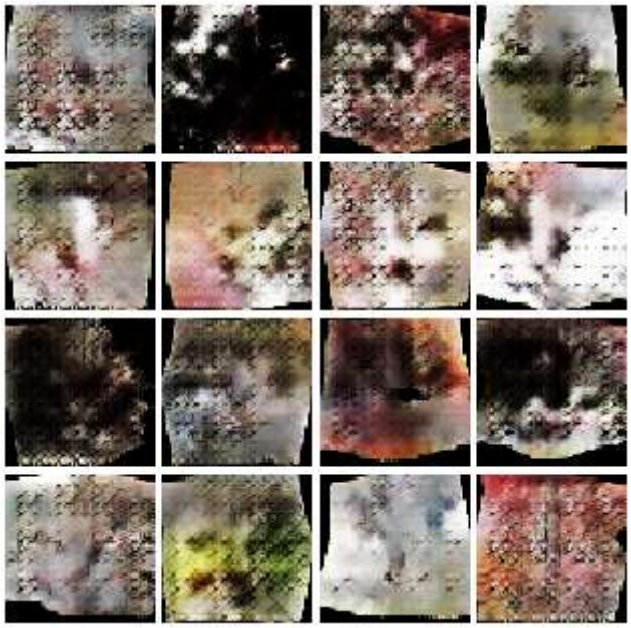
EPOCH: 41
Iter: 19750, D: -1.585e+07, G:8.362e+06

Iter: 20000, D: -1.619e+07, G:8.515e+06

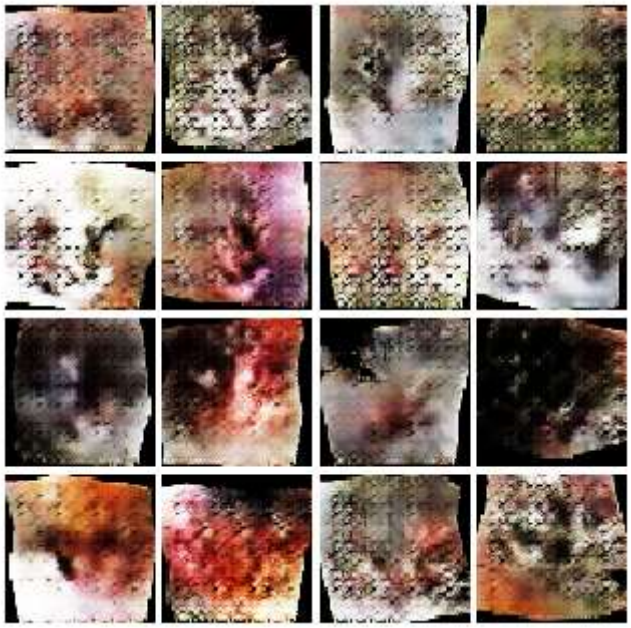
EPOCH: 42
Iter: 20250, D: -1.661e+07, G:8.767e+06



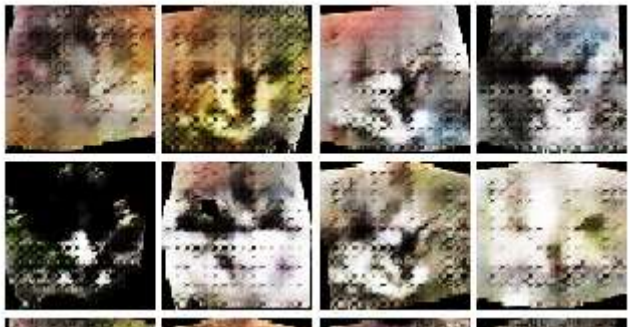
Iter: 20500, D: -1.7e+07, G:8.969e+06

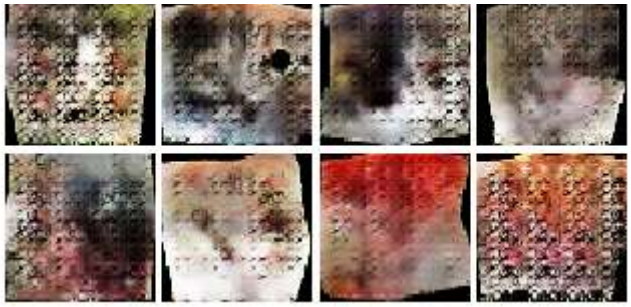


EPOCH: 43
Iter: 20750, D: -1.738e+07, G:9.174e+06

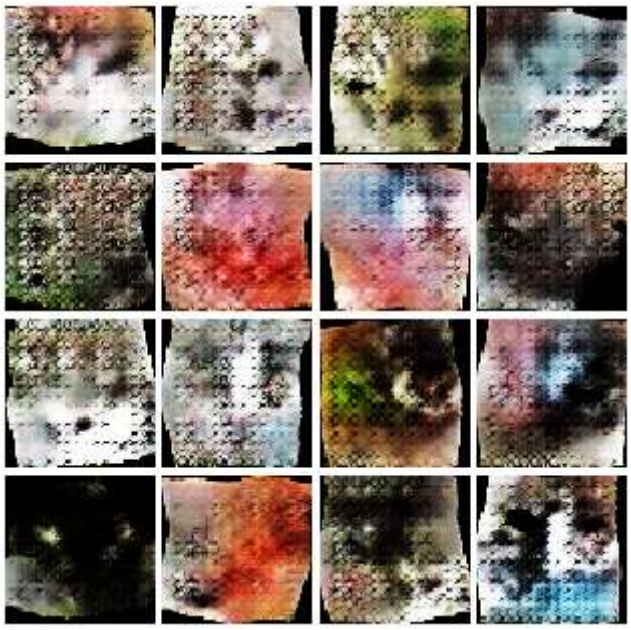


Iter: 21000, D: -1.777e+07, G:9.381e+06

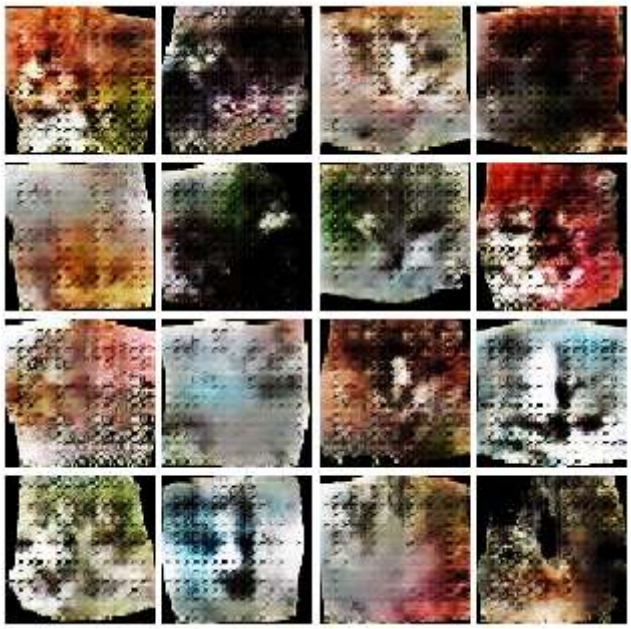




```
EPOCH:  44
Iter: 21250, D: -1.816e+07, G:9.589e+06
```

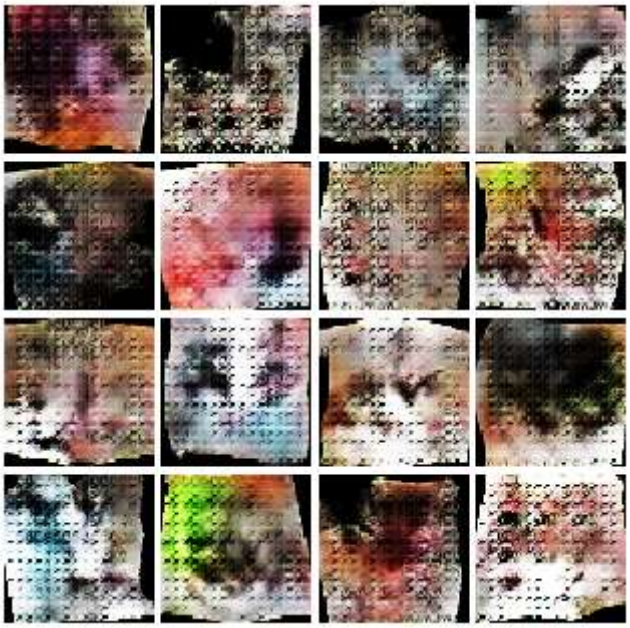


```
Iter: 21500, D: -1.856e+07, G:9.799e+06
```



```
EPOCH:  45
Iter: 21750, D: -1.897e+07, G:1.001e+07
```

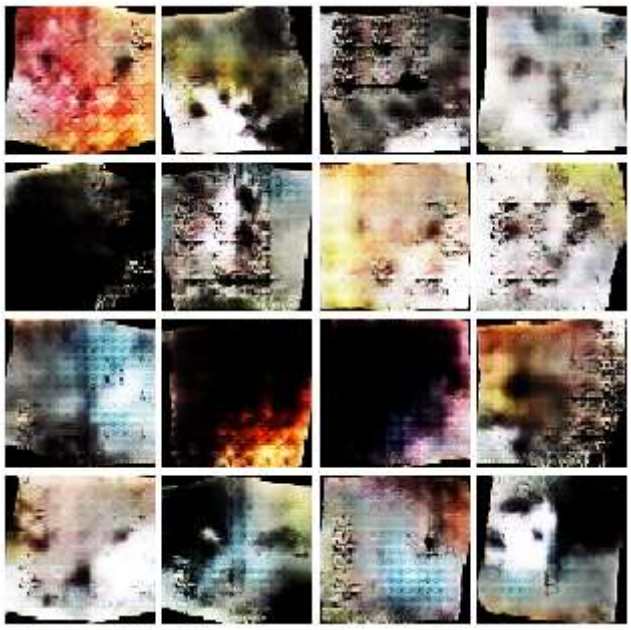
Iter: 22000, D: -1.937e+07, G:1.023e+07



EPOCH: 46
Iter: 22250, D: -1.978e+07, G:1.044e+07

Iter: 22500, D: -2.019e+07, G:1.066e+07

EPOCH: 47
Iter: 22750, D: -2.06e+07, G:1.088e+07



Iter: 23000, D: -2.101e+07, G:1.11e+07

```
EPOCH:  48
Iter: 23250, D: -2.145e+07, G:1.133e+07
```