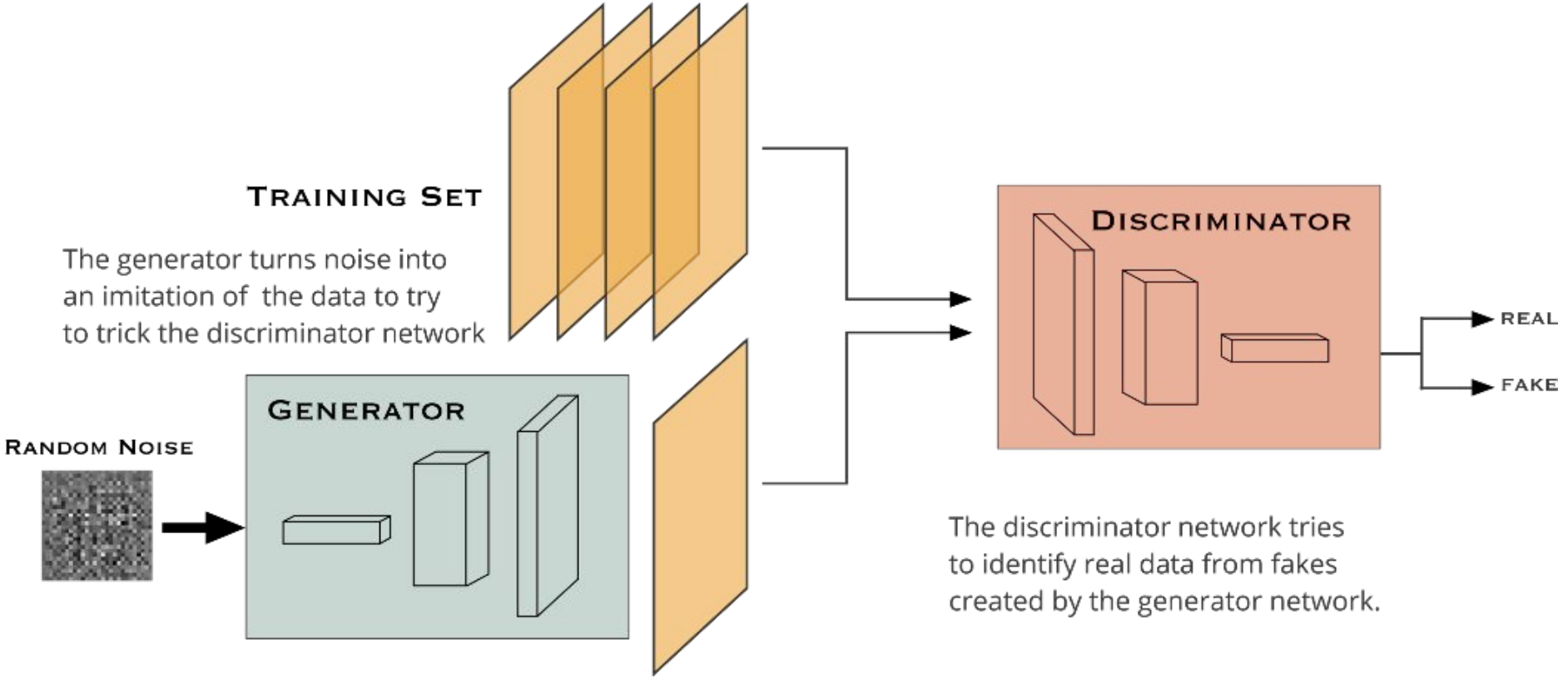
Exploring Image Generation

Harth Majeed
3546 Deep Learning

Quick Overview

We “generate” images to trick the “discriminator”.



Picture Source: https://towardsdatascience.com/image-generation-in-10-minutes-with-generative-adversarial-networks-c2afc56bfa3b

How does a GAN react to different types of Images?

- How does the network react to different types of images
- Does it react well to textures?
- Does it react well to shapes?
- Does it react well to detail?

We wish to understand the strength and weaknesses

Baseline Model - Generator

- Sequential()
- model.add(layers.Dense(8*8*512, use_bias=False))
- model.add(layers.Reshape((8, 8, 512)))
- assert model.output_shape == (None, 8, 8, 512)
- model.add(layers.Conv2DTranspose(256, (5, 5), strides=(1, 1), padding='same', use_bias=False))
- assert model.output_shape == (None, 8, 8, 256)
- model.add(layers.Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same', use_bias=False))
- assert model.output_shape == (None, 16, 16, 128)
- model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
- assert model.output_shape == (None, 32, 32, 64)
- model.add(layers.Conv2DTranspose(32, (5, 5), strides=(2, 2), padding='same', use_bias=False))
- assert model.output_shape == (None, 64, 64, 32)
- model.add(layers.Conv2DTranspose(16, (5, 5), strides=(2, 2), padding='same', use_bias=False))
- assert model.output_shape == (None, 128, 128, 16)
- model.add(layers.Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
- assert model.output_shape == (None, 256, 256, 3)

Baseline Model - Discriminator

- Sequential()
- model.add(layers.Conv2D(16, (5, 5), strides=(2, 2), padding='same', input_shape=[256, 256, 3]))
- model.add(layers.LeakyReLU())
- model.add(layers.Dropout(0.3))
- model.add(layers.Conv2D(32, (5, 5), strides=(2, 2), padding='same'))
- model.add(layers.LeakyReLU())
- model.add(layers.Dropout(0.3))
- model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same'))
- model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
- model.add(layers.Conv2D(256, (5, 5), strides=(2, 2), padding='same'))
- model.add(layers.Flatten())
- model.add(layers.Dense(1))

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32768)	3276800
batch_normalization (BatchNormalization)	(None, 32768)	131072
leaky_re_lu (LeakyReLU)	(None, 32768)	0
reshape (Reshape)	(None, 8, 8, 512)	0
conv2d_transpose (Conv2DTranspose)	(None, 8, 8, 256)	3276800
batch_normalization_1 (BatchNormalization)	(None, 8, 8, 256)	1024
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 256)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 16, 16, 128)	819200
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 32, 32, 64)	204800
batch_normalization_3 (BatchNormalization)	(None, 32, 32, 64)	256
leaky_re_lu_3 (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_transpose_3 (Conv2DTranspose)	(None, 64, 64, 32)	51200
batch_normalization_4 (BatchNormalization)	(None, 64, 64, 32)	128
leaky_re_lu_4 (LeakyReLU)	(None, 64, 64, 32)	0
conv2d_transpose_4 (Conv2DTranspose)	(None, 128, 128, 16)	12800
batch_normalization_5 (BatchNormalization)	(None, 128, 128, 16)	64
leaky_re_lu_5 (LeakyReLU)	(None, 128, 128, 16)	0
conv2d_transpose_5 (Conv2DTranspose)	(None, 256, 256, 3)	1200

Total params: 7,775,856
Trainable params: 7,709,328
Non-trainable params: 66,528

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 16)	1216
leaky_re_lu_6 (LeakyReLU)	(None, 128, 128, 16)	0
dropout (Dropout)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 64, 64, 32)	12832
leaky_re_lu_7 (LeakyReLU)	(None, 64, 64, 32)	0
dropout_1 (Dropout)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	51264
leaky_re_lu_8 (LeakyReLU)	(None, 32, 32, 64)	0
dropout_2 (Dropout)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	204928
leaky_re_lu_9 (LeakyReLU)	(None, 16, 16, 128)	0
dropout_3 (Dropout)	(None, 16, 16, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	819456
leaky_re_lu_10 (LeakyReLU)	(None, 8, 8, 256)	0
dropout_4 (Dropout)	(None, 8, 8, 256)	0
flatten (Flatten)	(None, 16384)	0
dense_1 (Dense)	(None, 1)	16385

Total params: 1,106,081
Trainable params: 1,106,081
Non-trainable params: 0

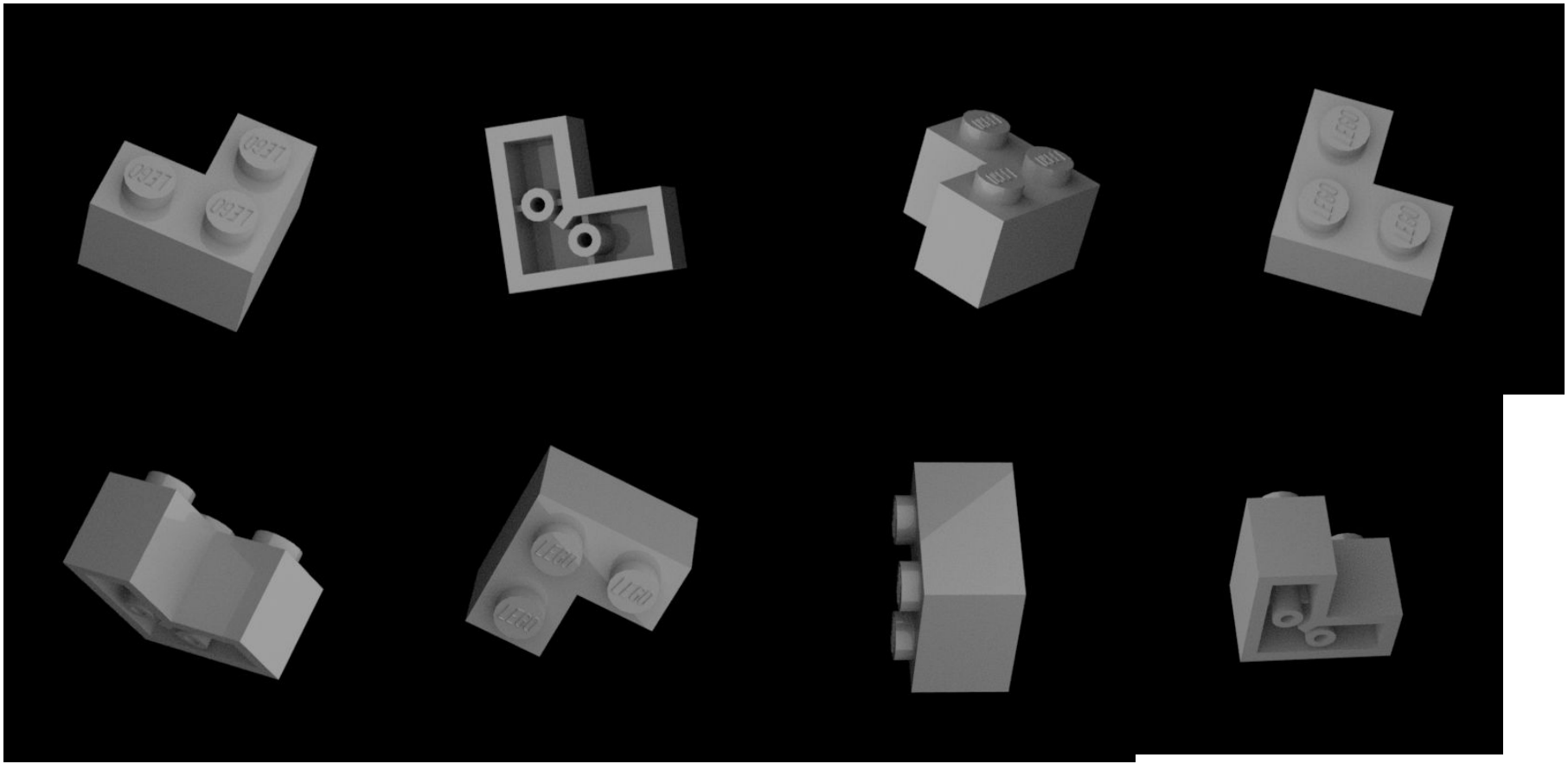
The Plan

- The baseline model will be used across all datasets, will not be changed
- Epoch = 5000

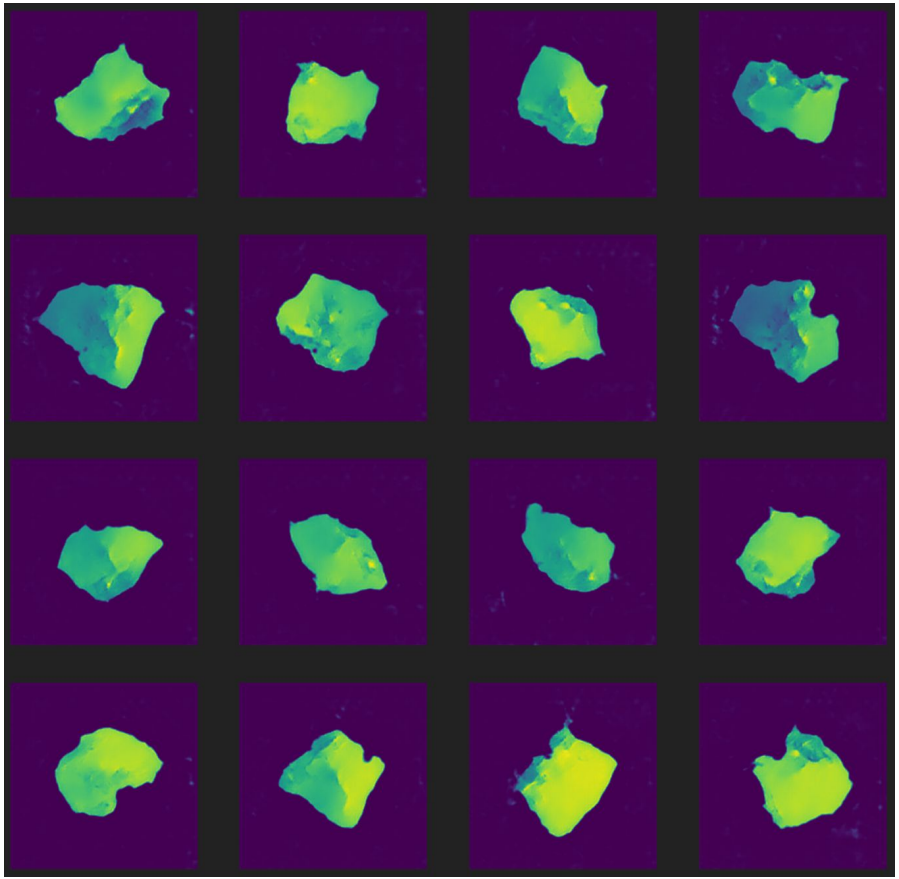
Datasets

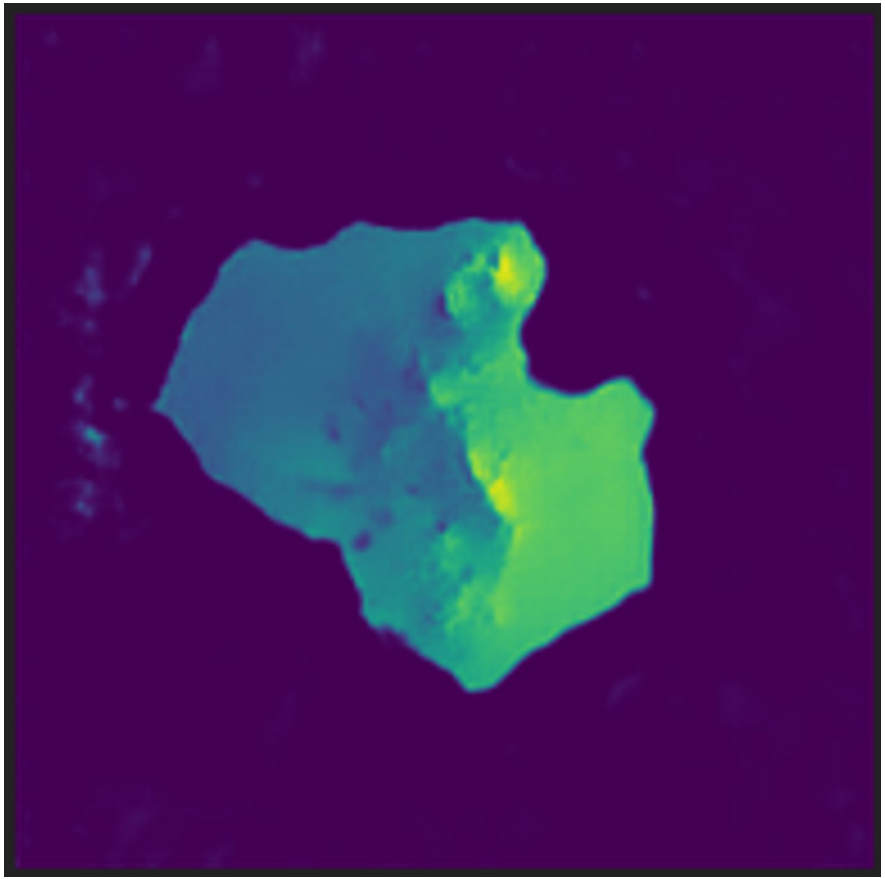
- Lego, corner piece (800 images)
- Cutlery (spoons, knifes, forks, etc) (870 images)
- Marble textures (120 images)
- Cats (500, 4096 images)

Lego Results - Input Images (800)

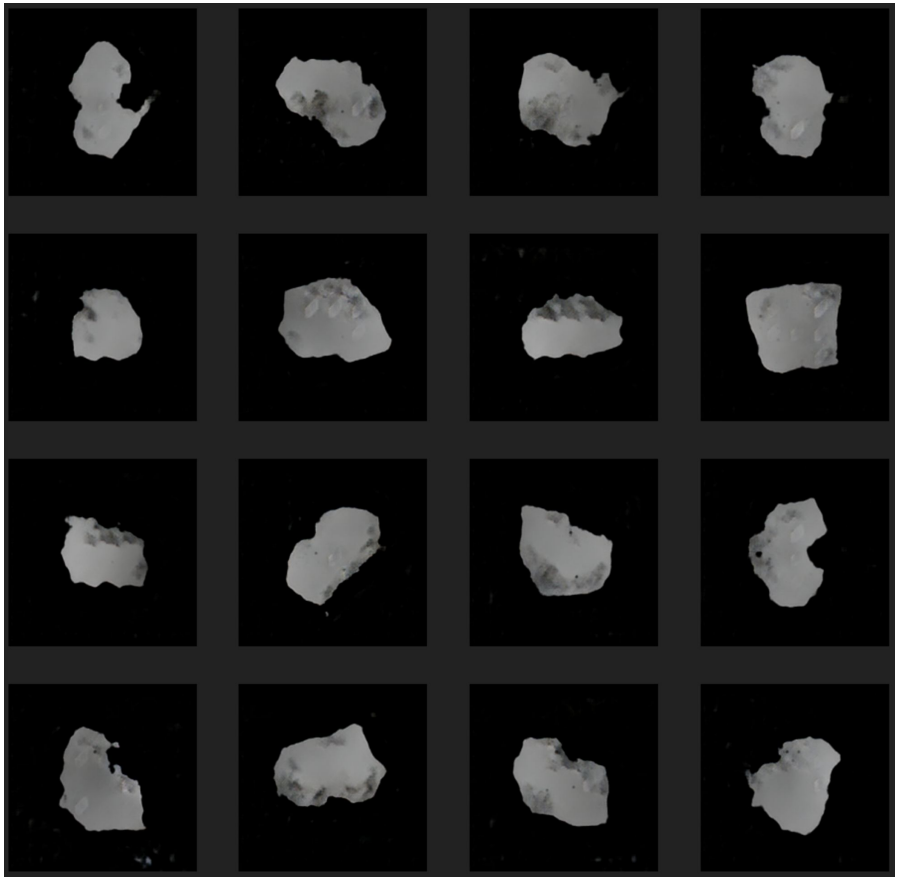


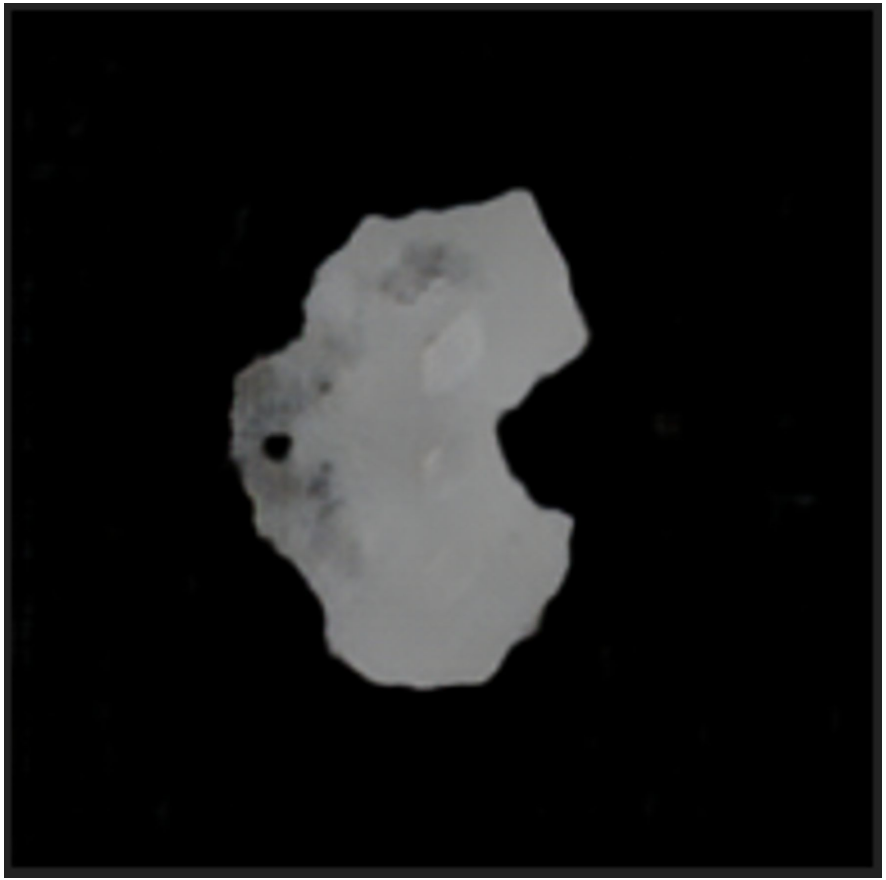
Lego Results - Output Images (800 Images) 5k epoch



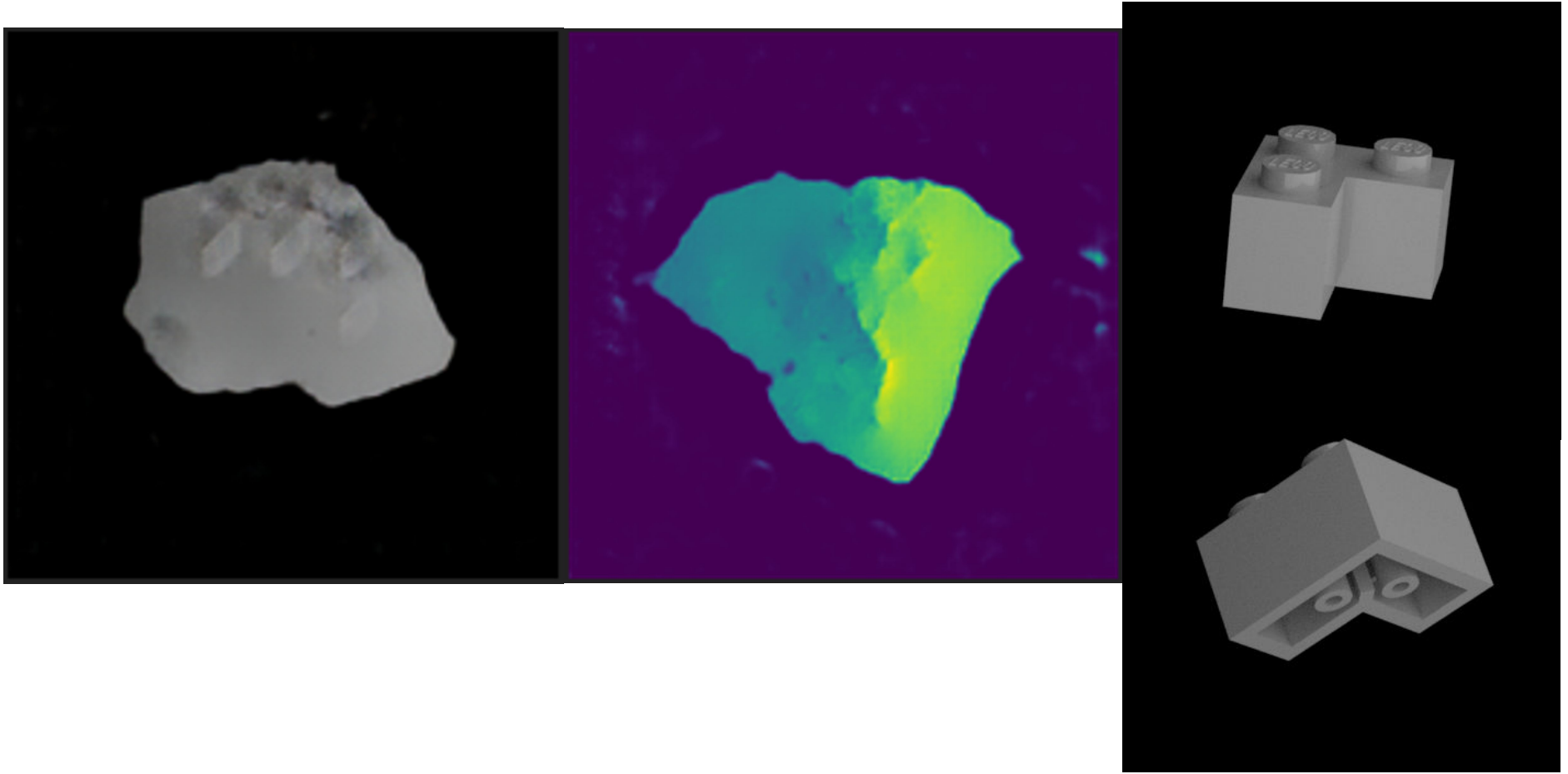


Lego Results - Output Images (800 Images) 5k epoch

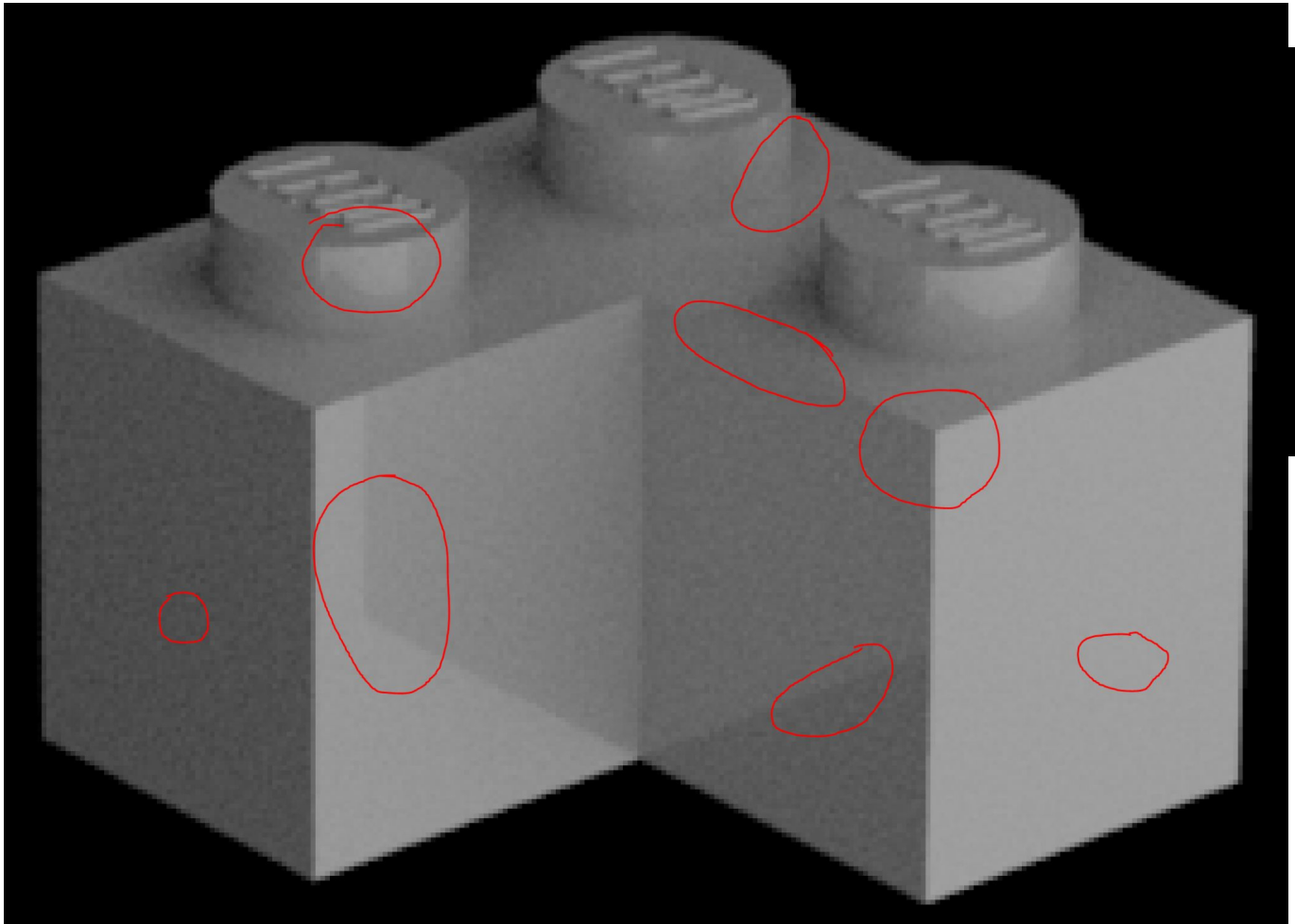


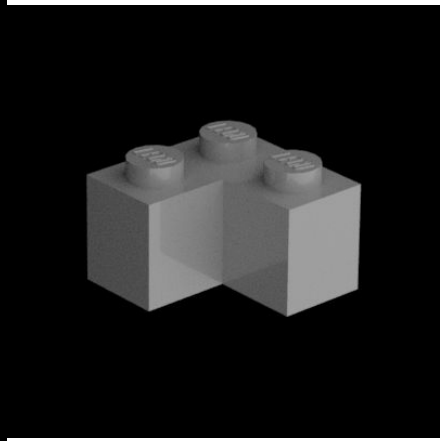


Lego Results - Output Images (800 Images)

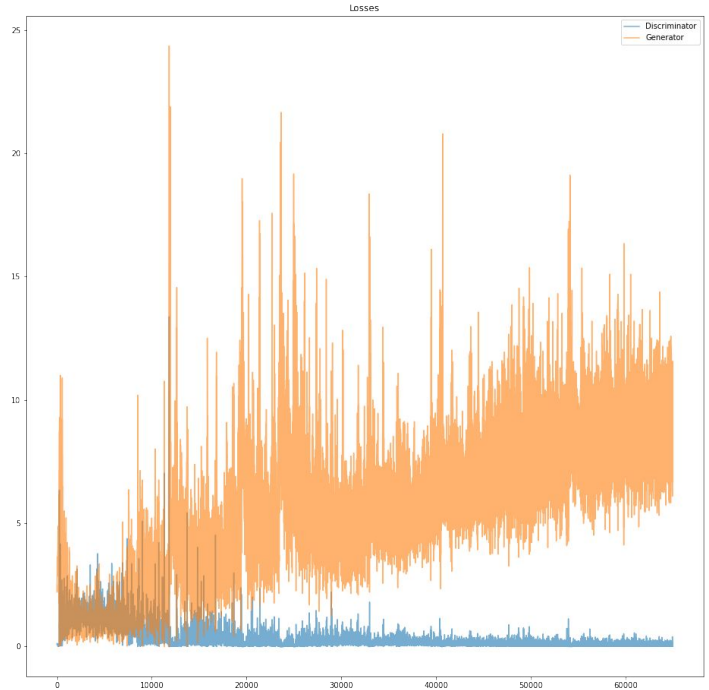


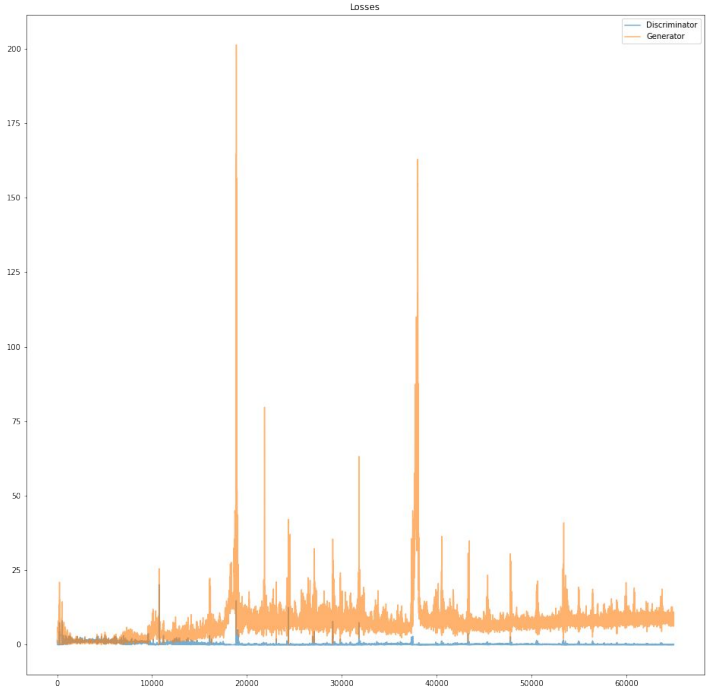
Lego Results - Analyzing the image more (800 Images)





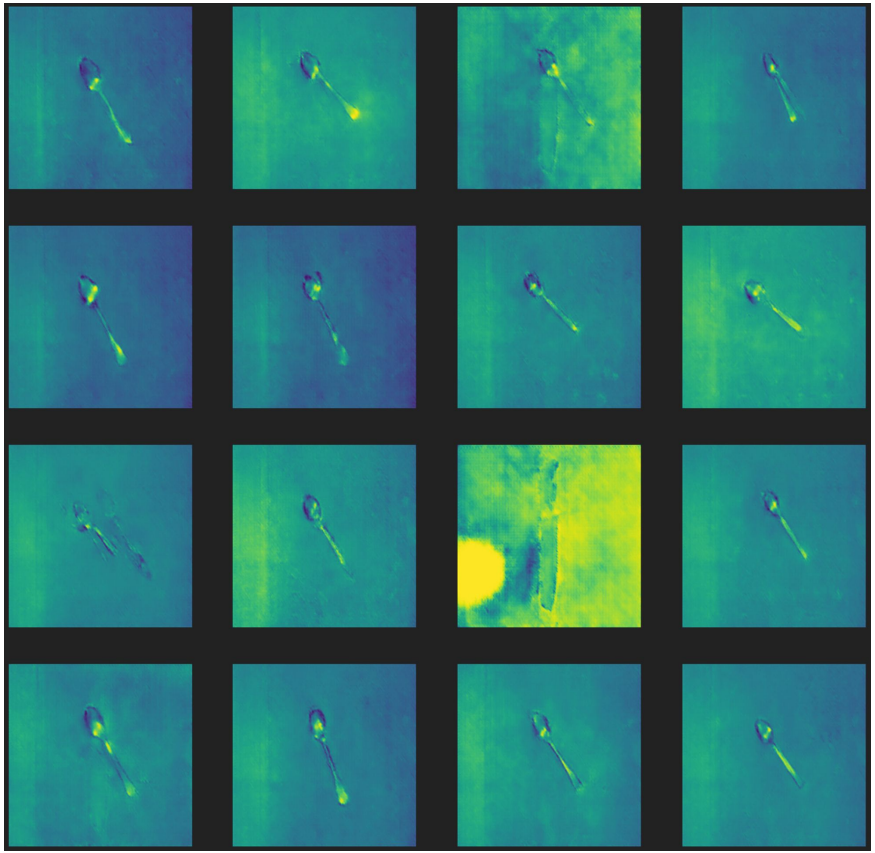
Lego Results - Graphs (800 Images) 5k epoch

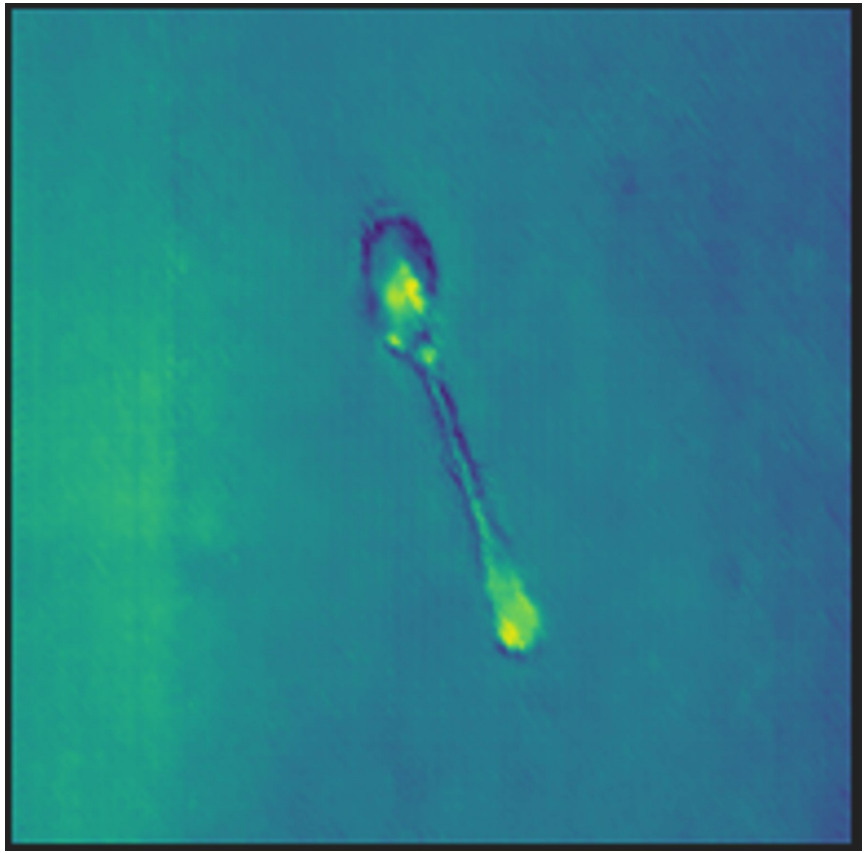




Cutlery Results - Input Images (870)

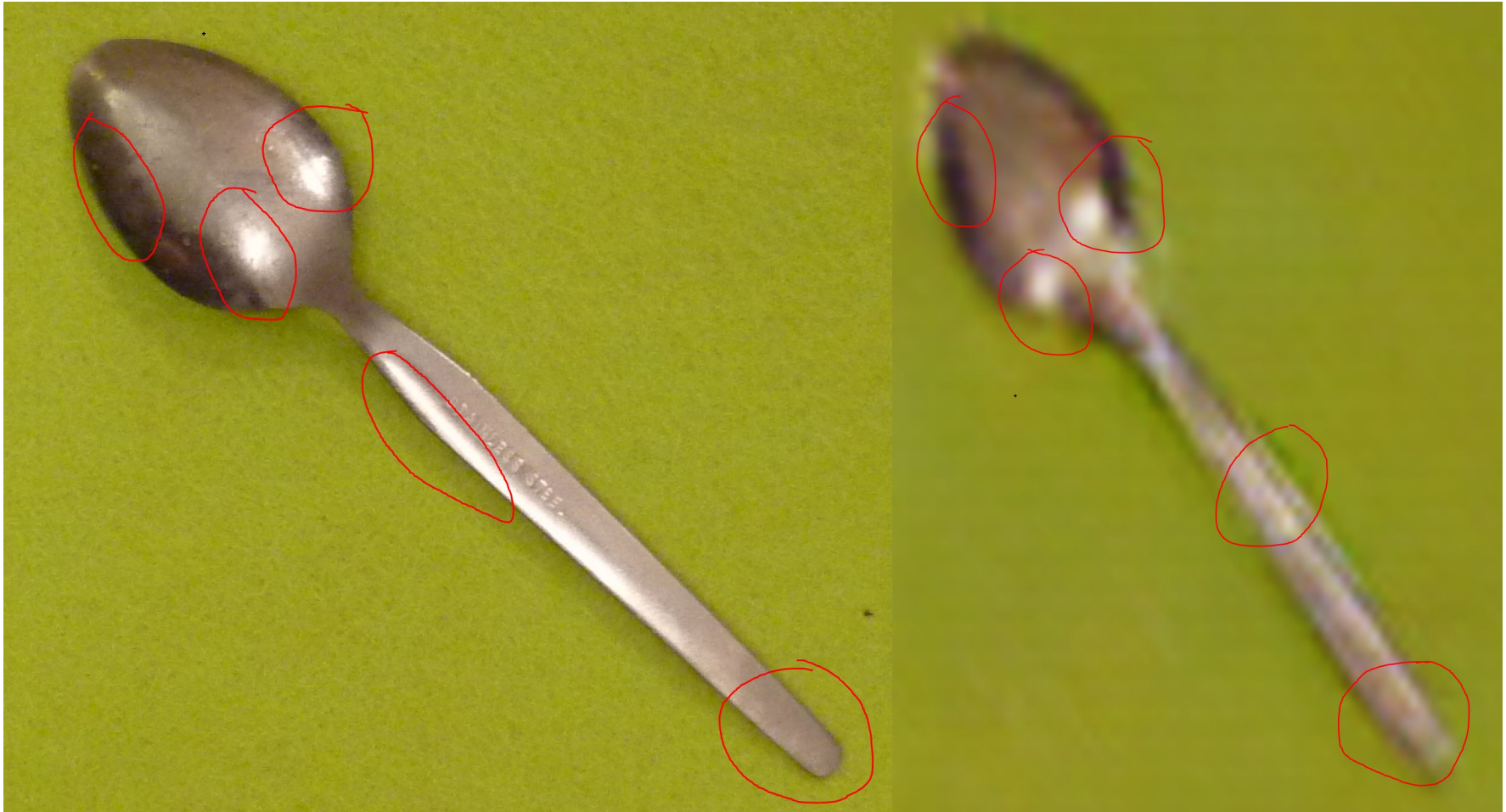
Cutlery Results - Output Images (870) 5k epoch



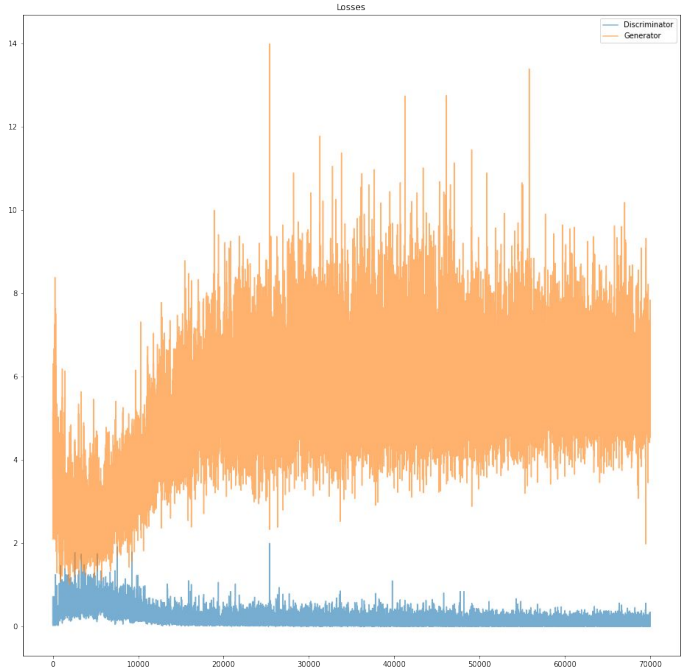


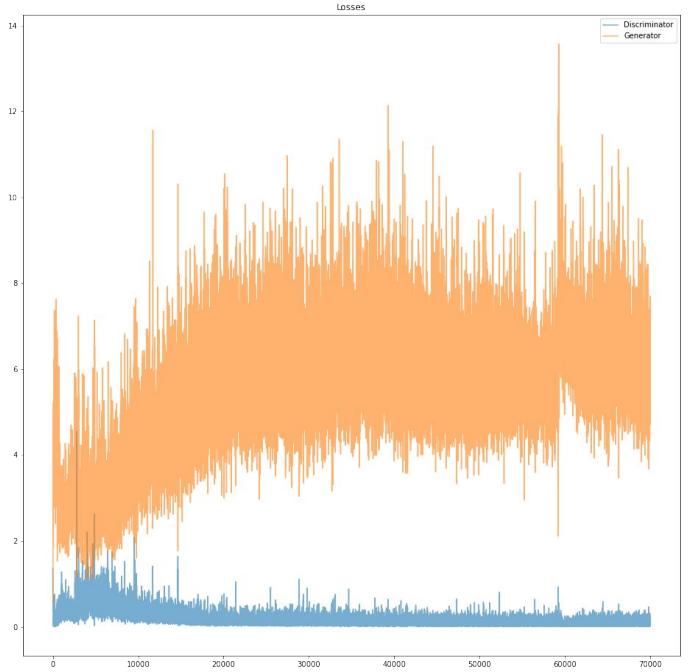
Cutlery Results - Output Images (870) 5k epoch

Cutlery Results - Analyzing the images (870)

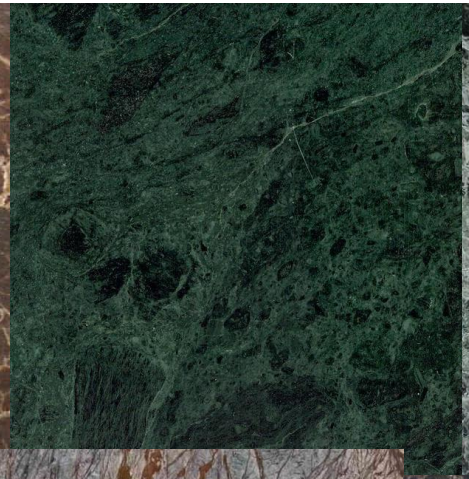


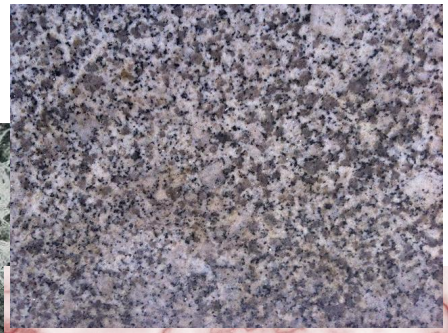
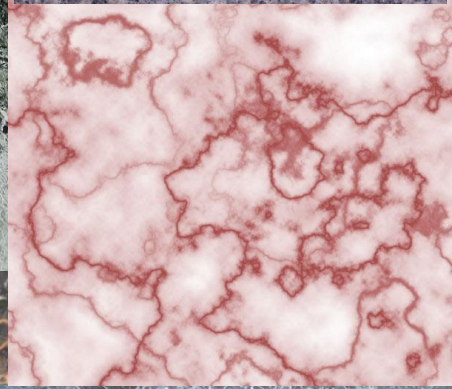
Cutlery Results - Output Images (870) 5k epoch

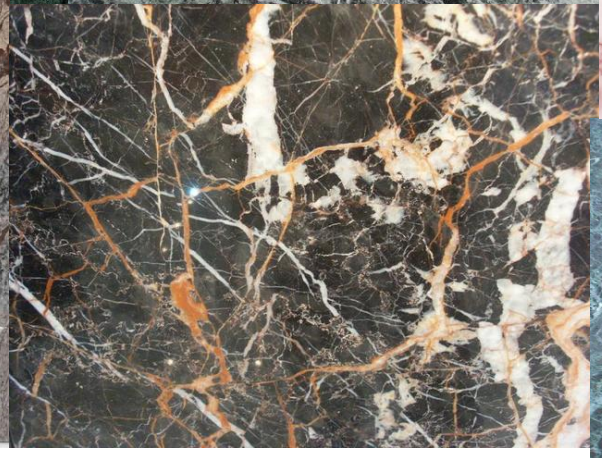
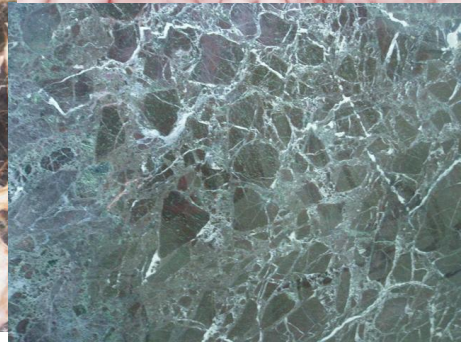




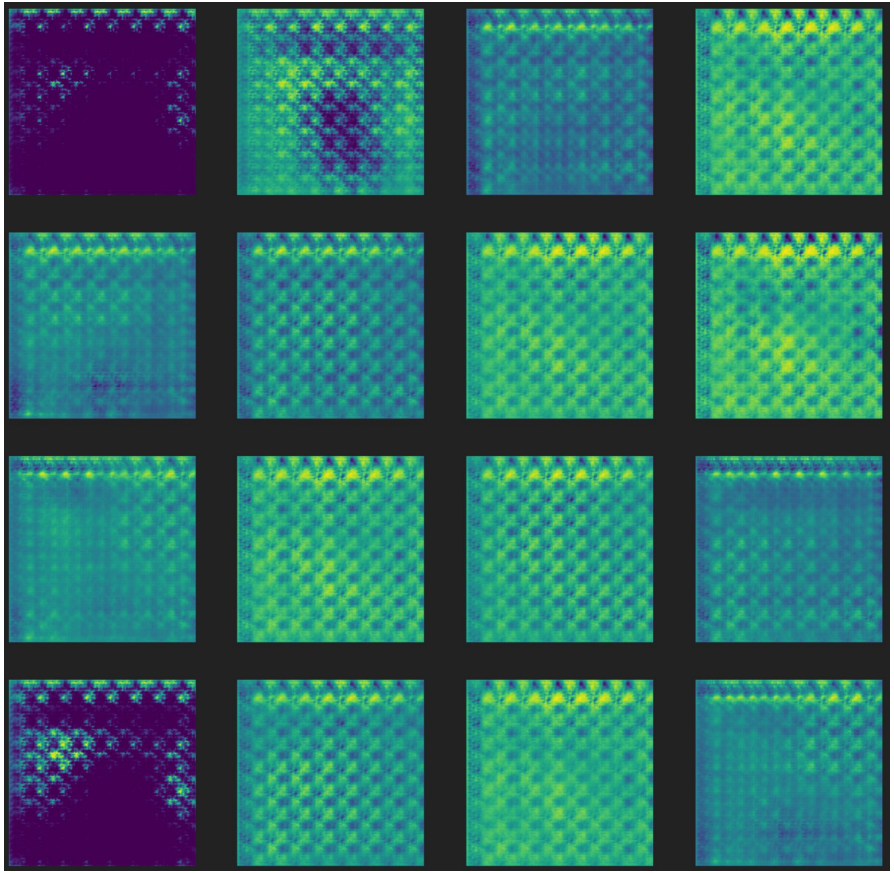
Marble Texture Results - Input Images (120)

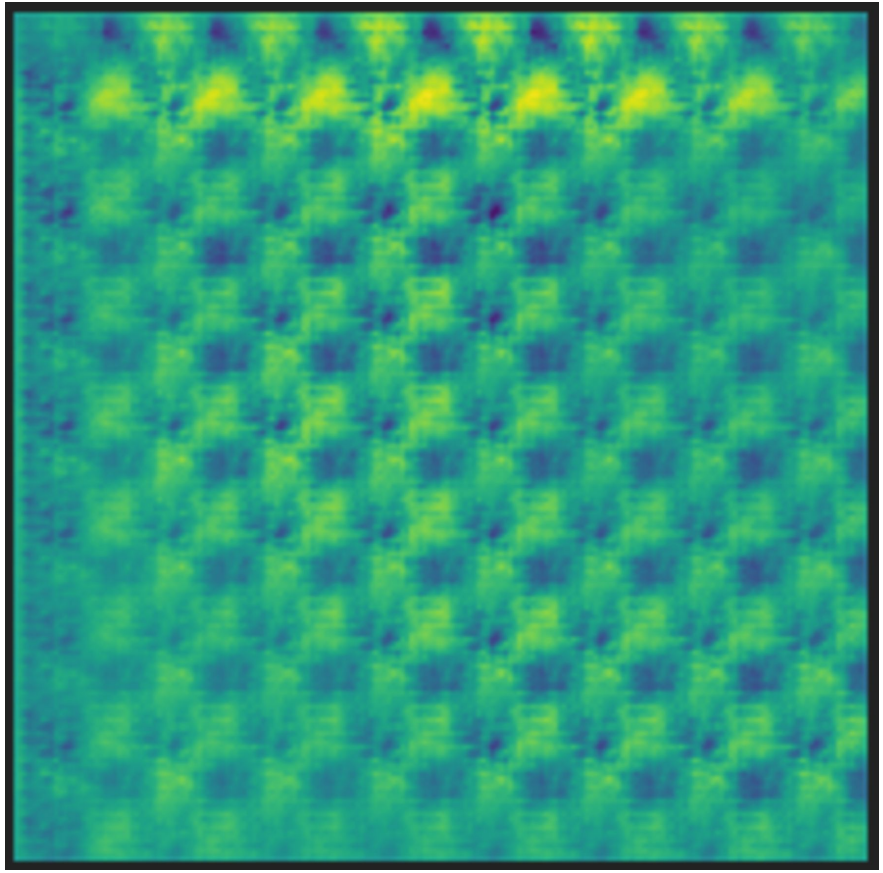



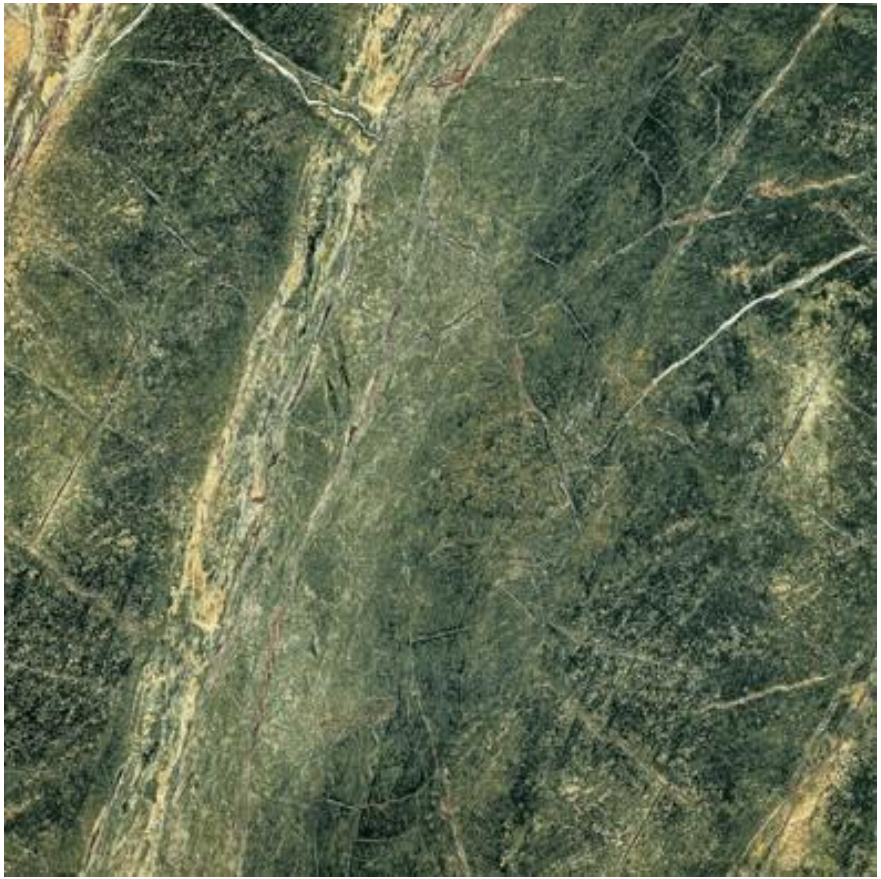




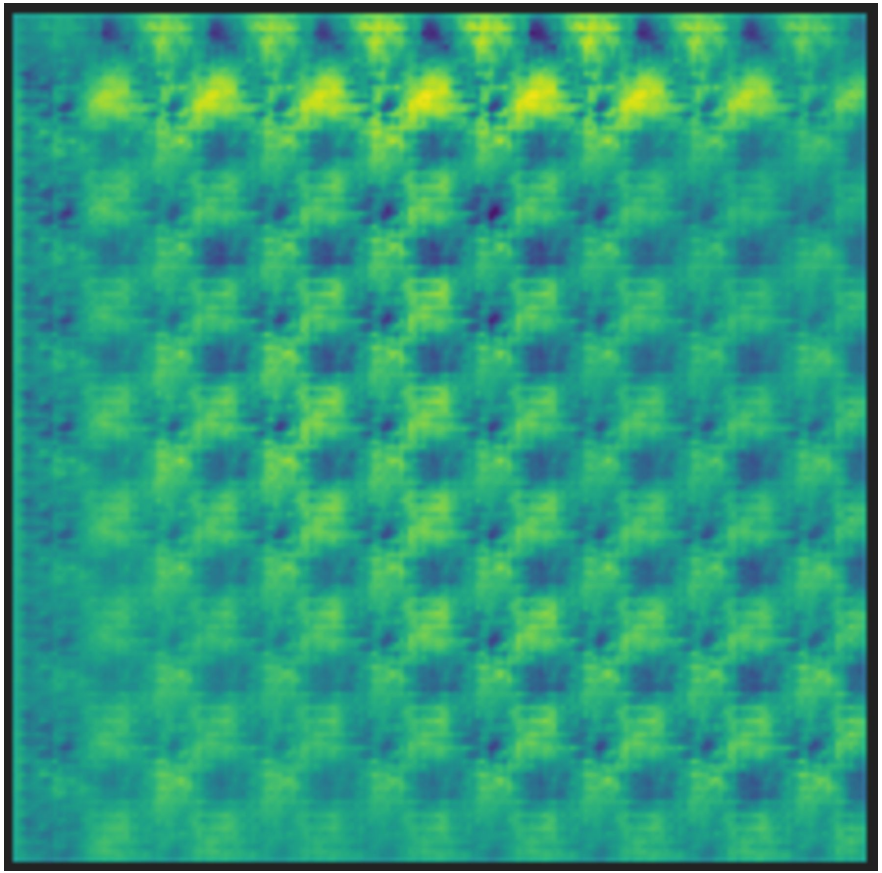
Marble Texture Results - Output Images (120) 5k epoch



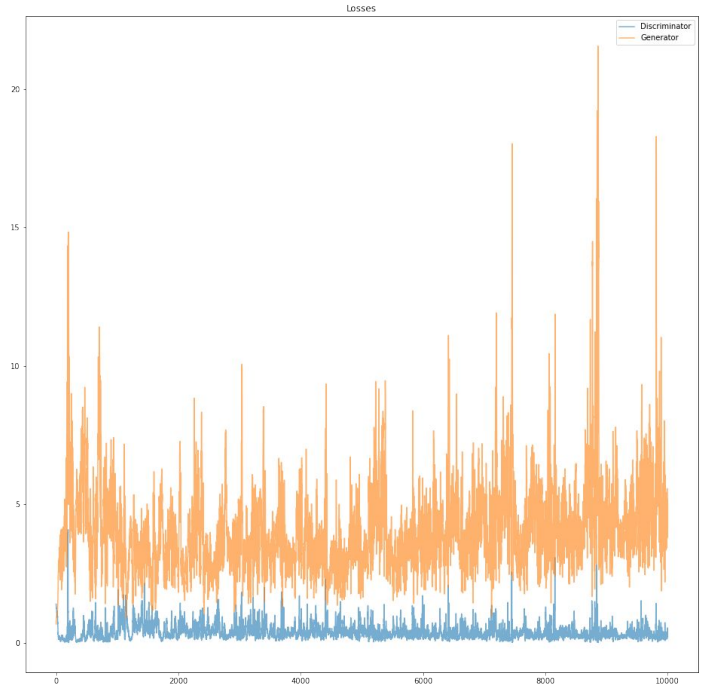


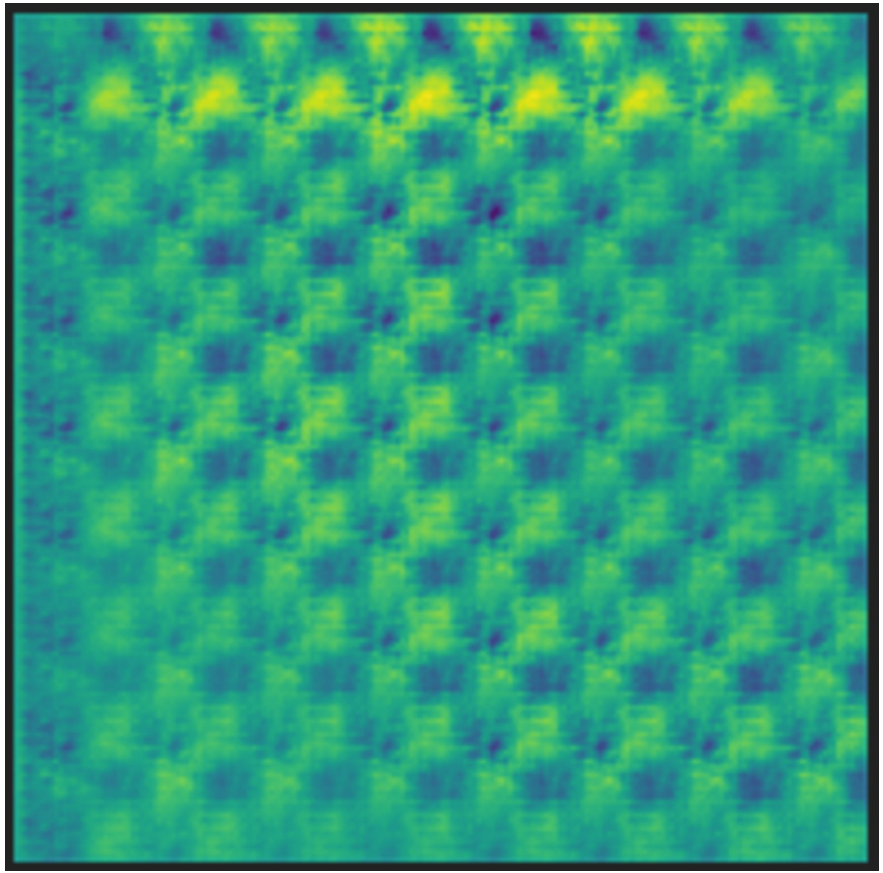
Marble Texture Results - Output Images (120) 5k epoch





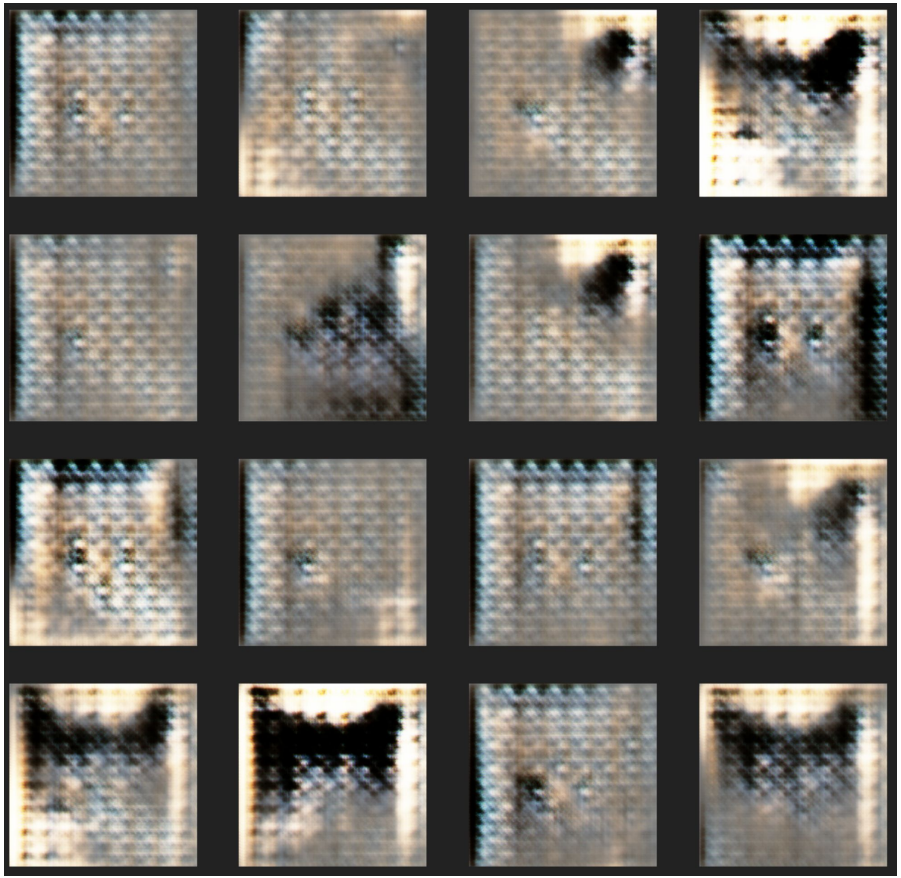
Marble Texture Results - Graphs (120) 5k epoch

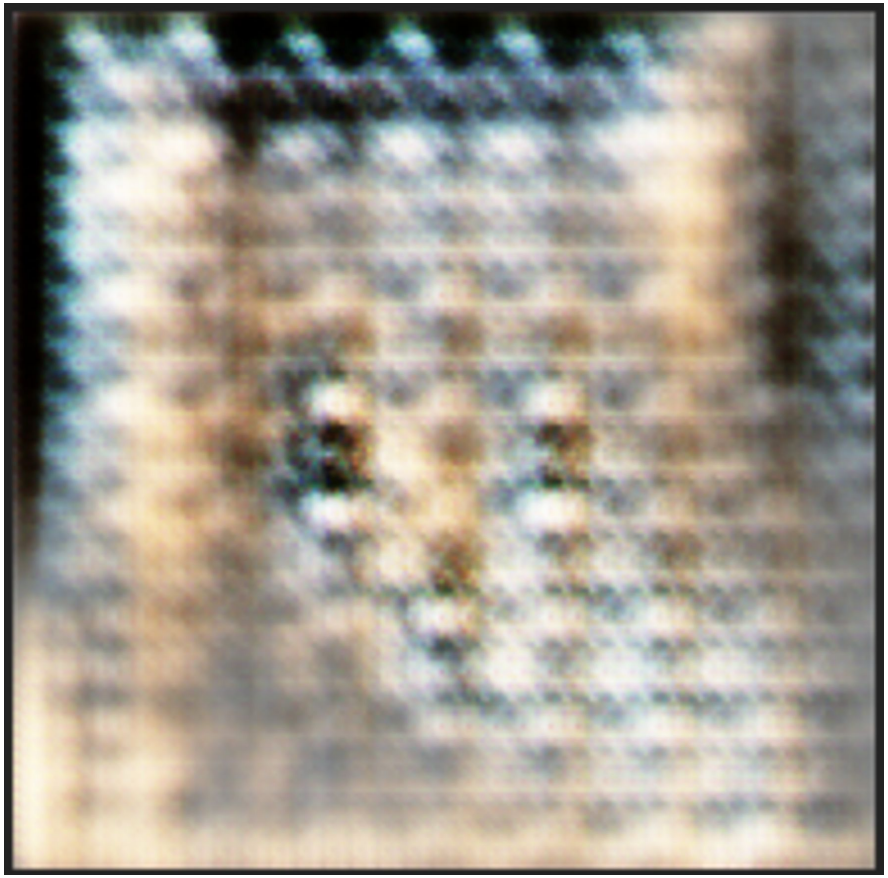




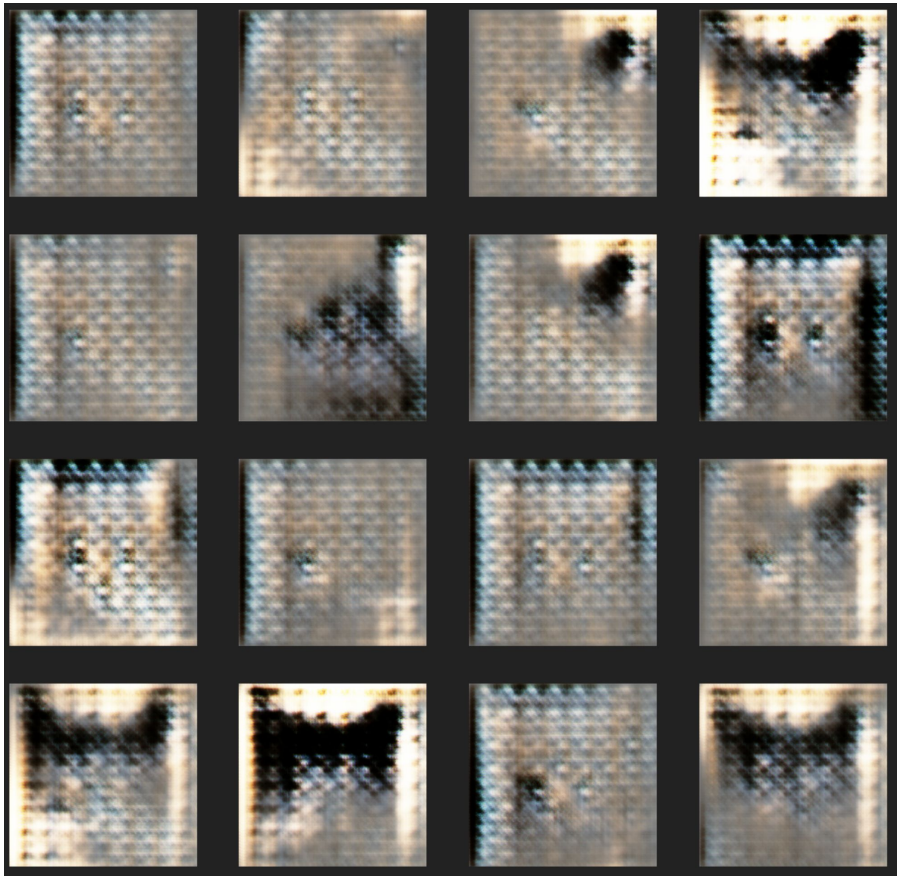
Cats Results - Input Images (500, 4096) 1k epoch

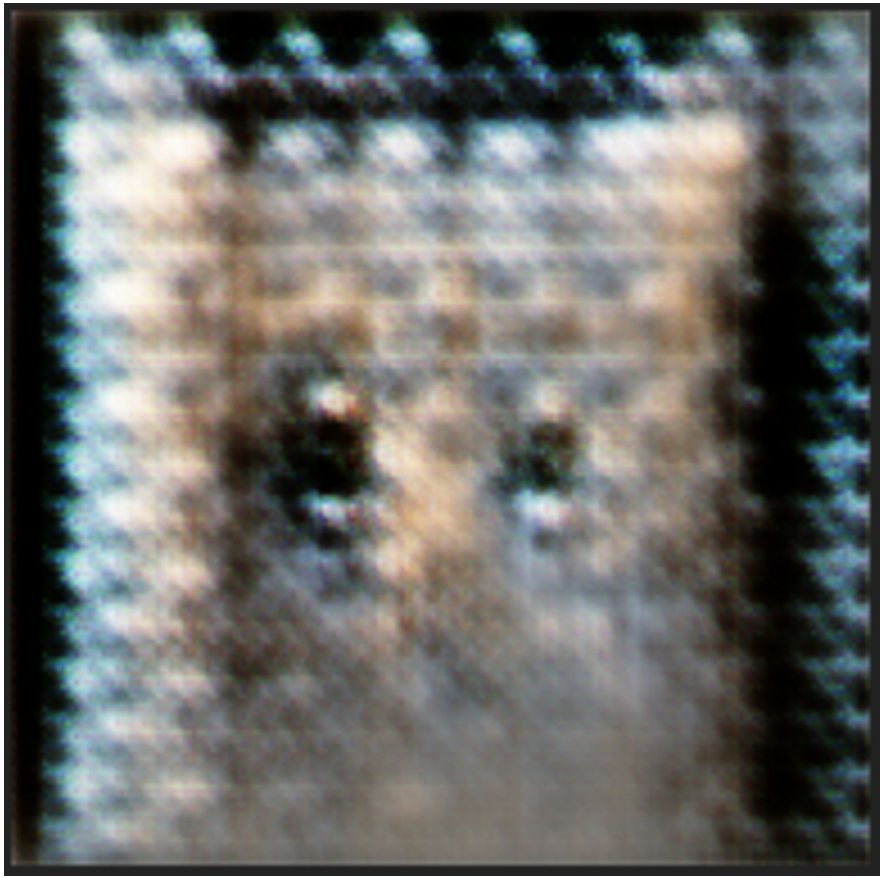
Cats Results - Output Images (500 images) 1k epoch



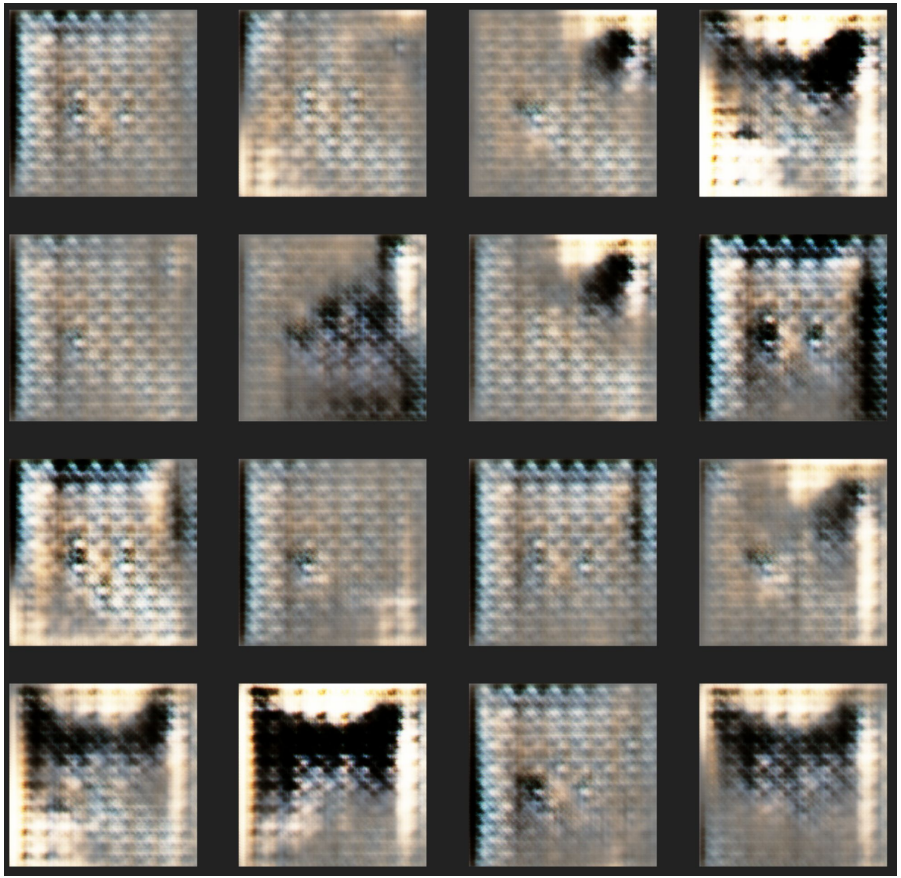


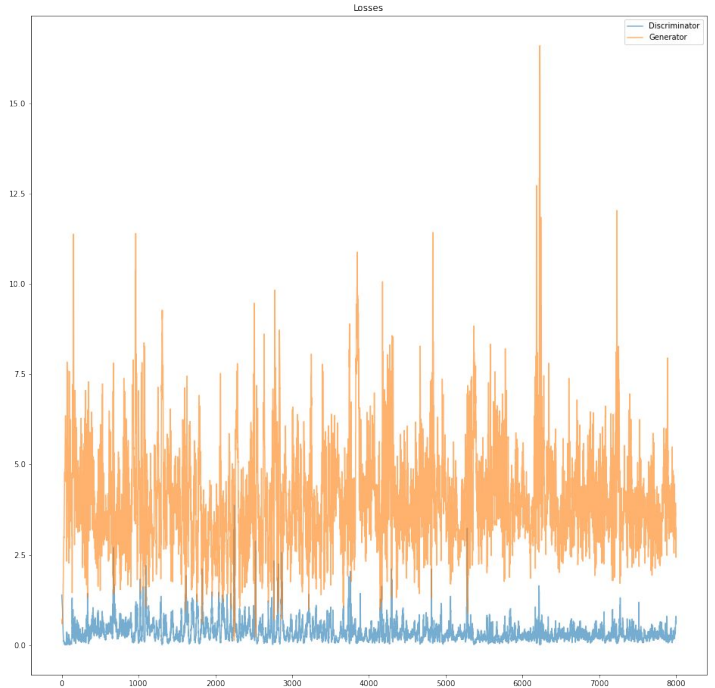
Cats Results - Output Images (500 images) 1k epoch



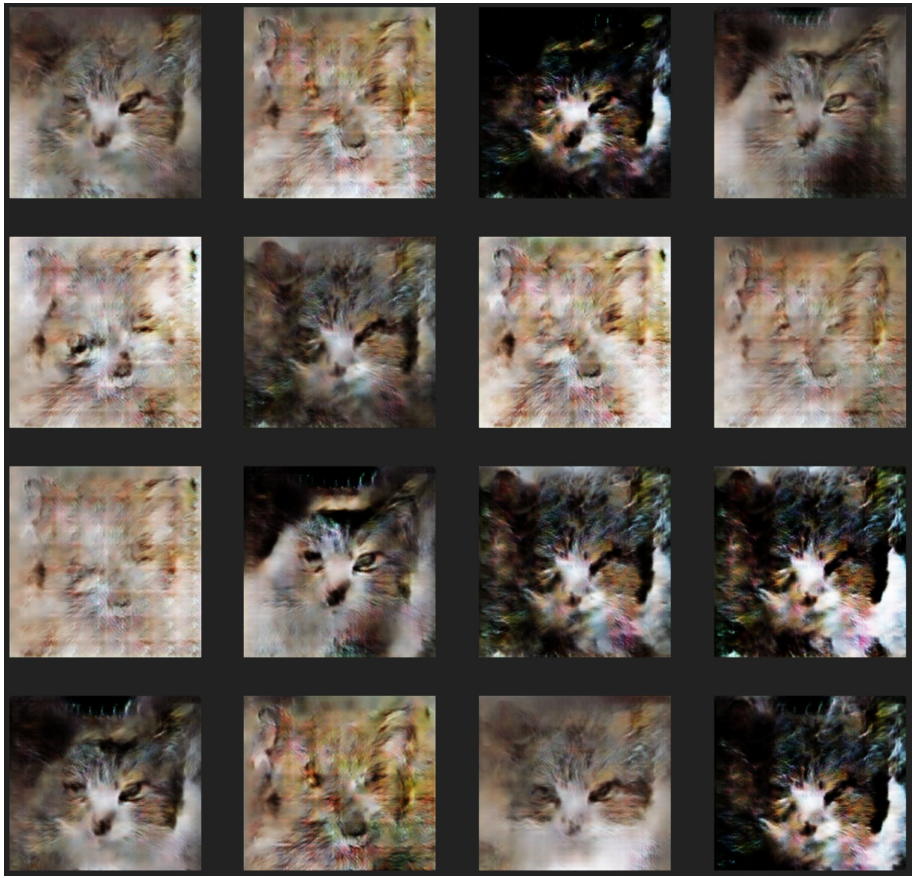


Cats Results - Graphs (500 images) 1k epoch

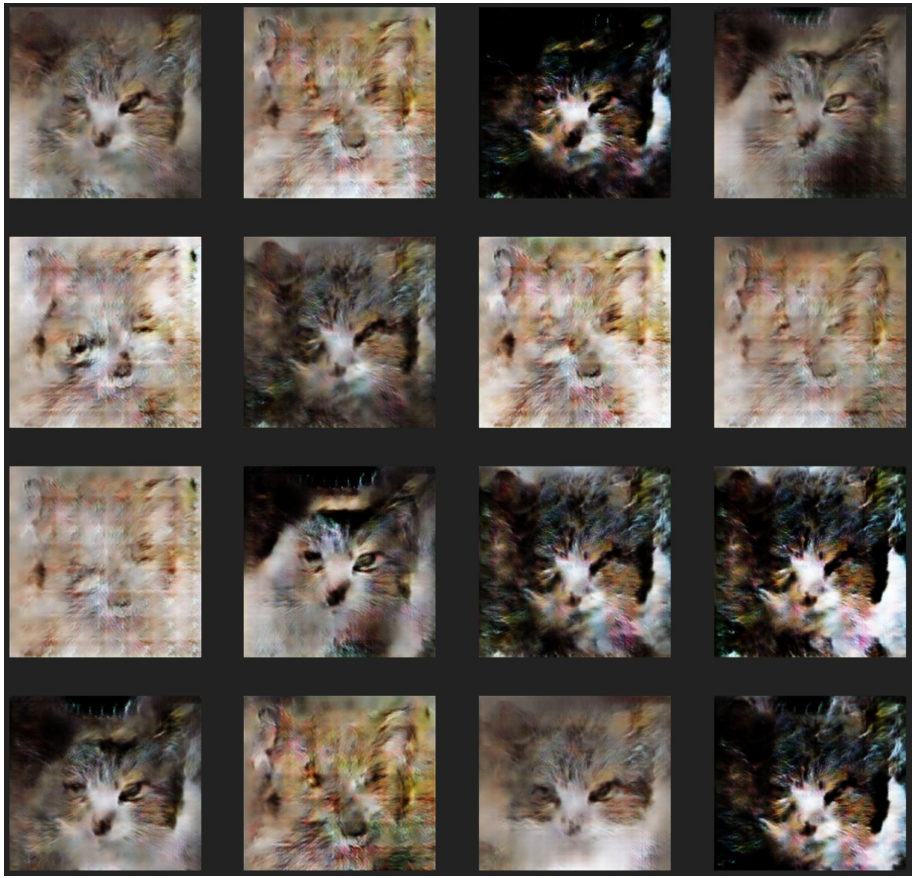




Cats Results - Output Images (4096 images) 1k epoch

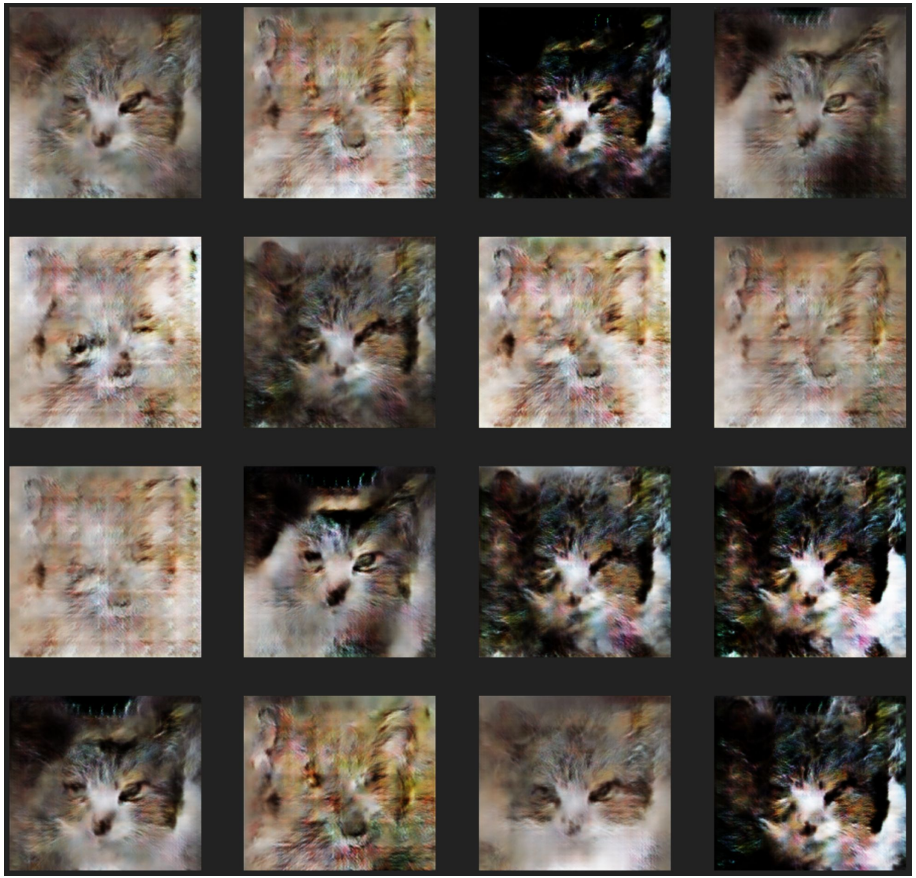


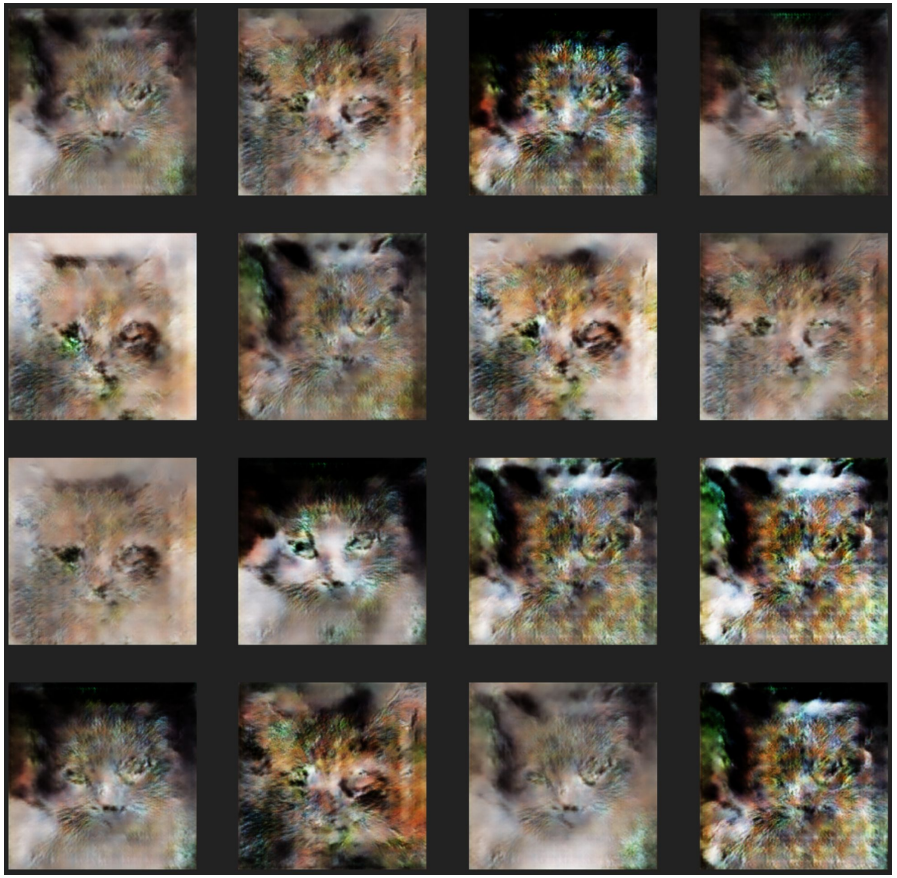
Cats Results - Output Images (4096 images) 1k epoch



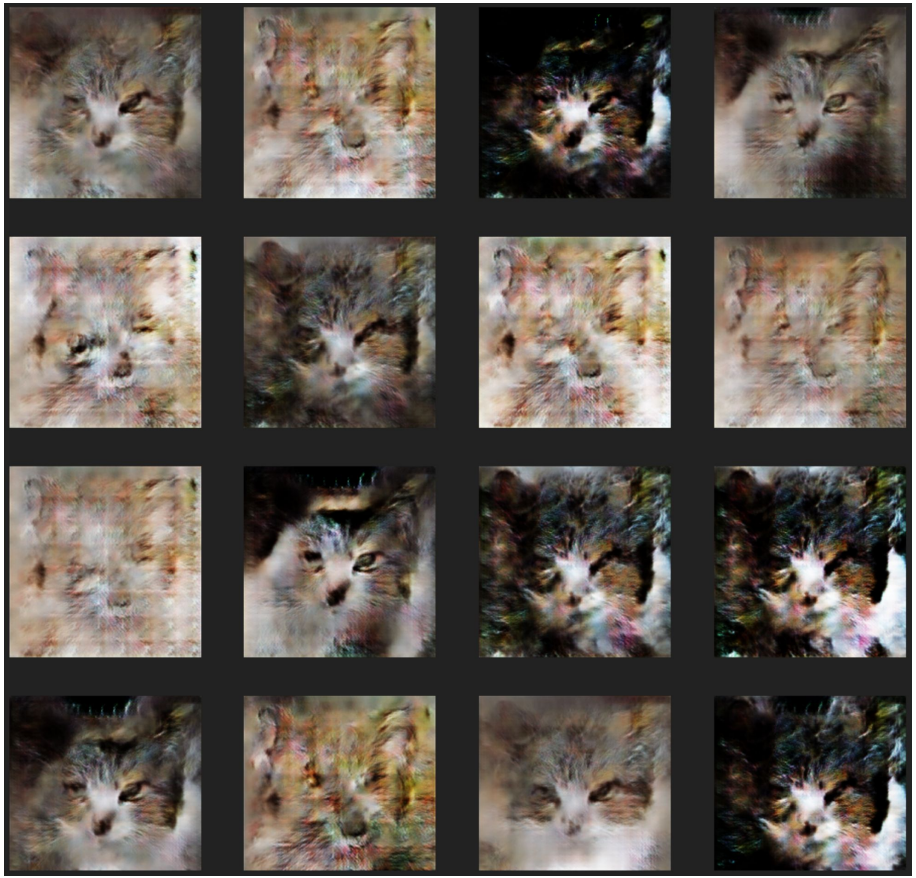
Cats Results - Output Images (4096 images) 1k epoch

900 epoch

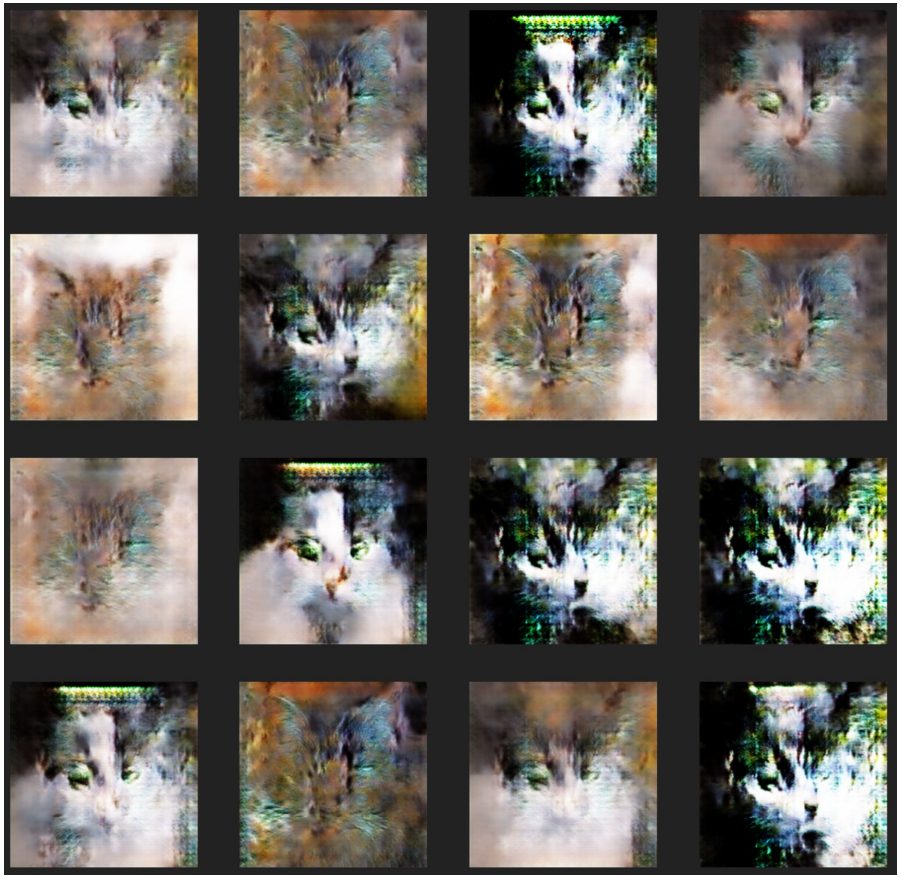




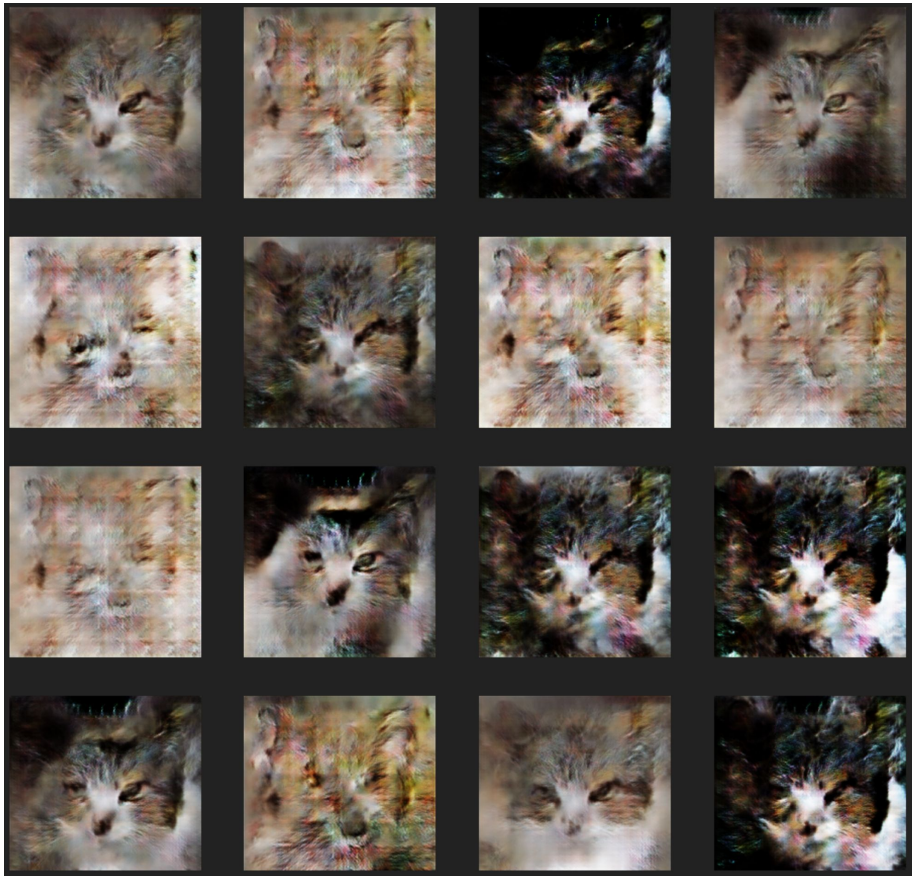
Cats Results - Output Images (4096 images) 1k epoch



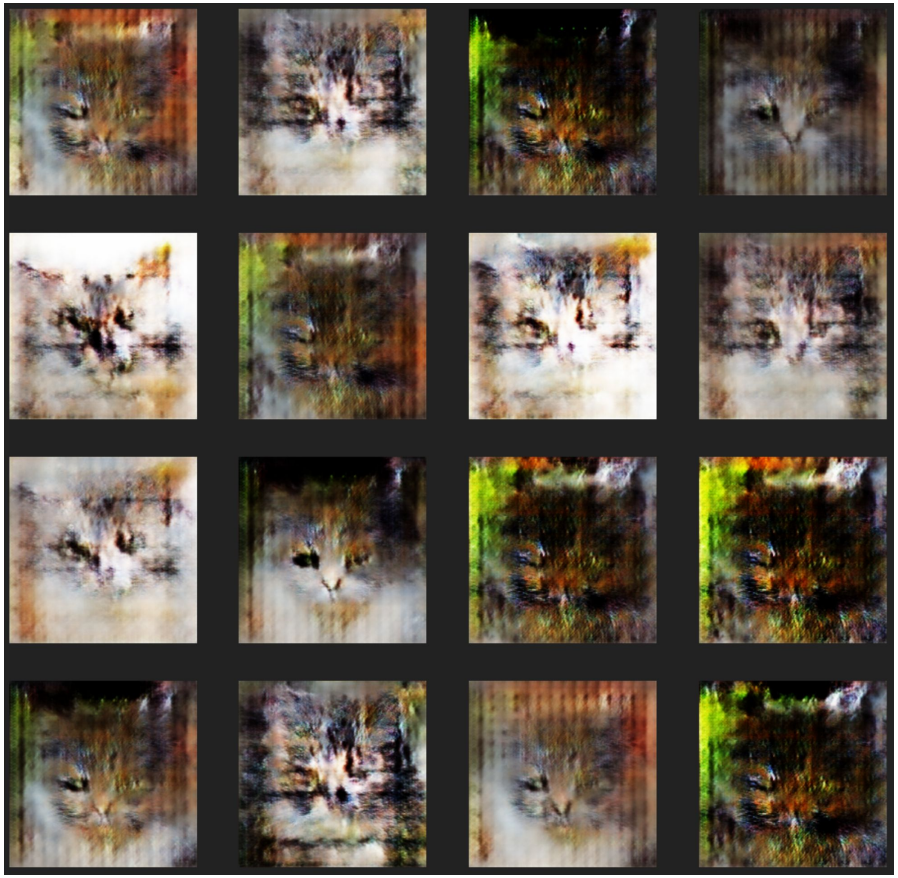
800 epoch



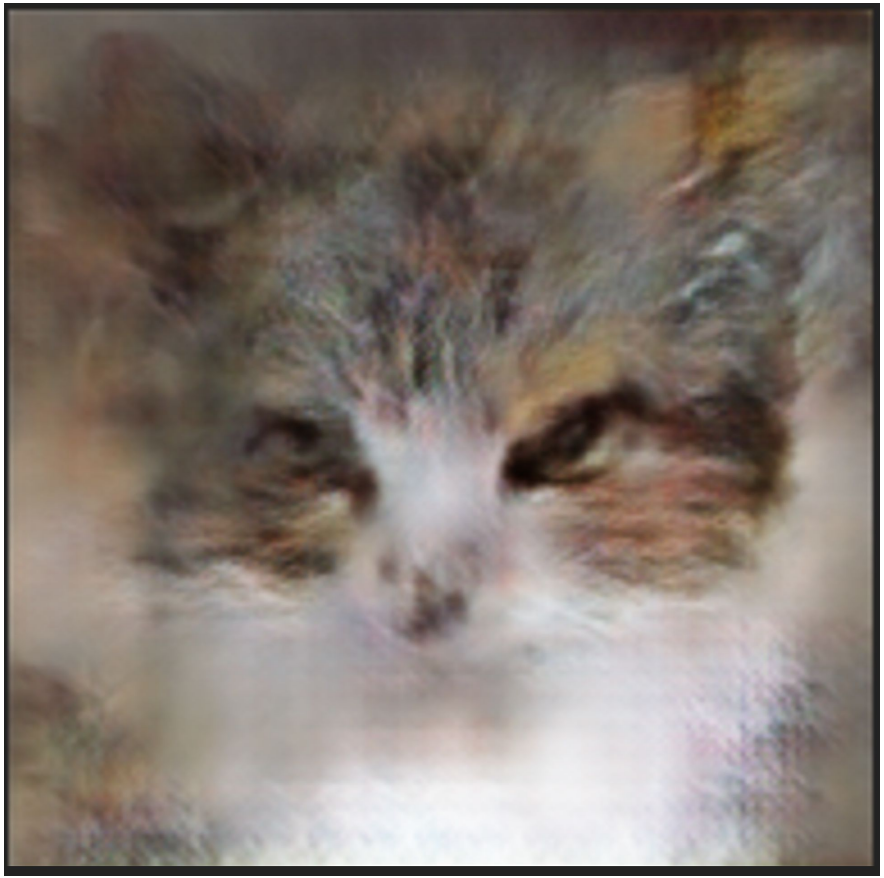
Cats Results - Output Images (4096 images) 1k epoch



700 epoch

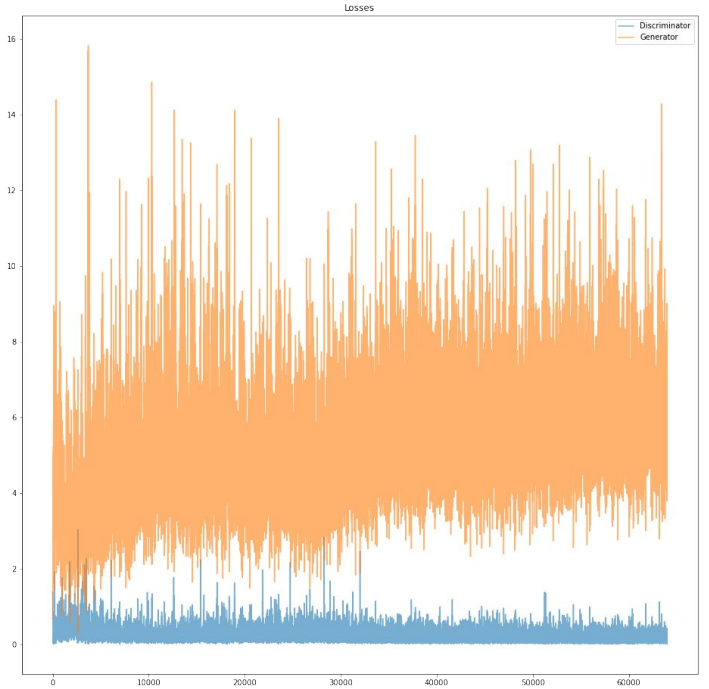


Cats Results - Analyzing Images (4096 images) 1k epoch



Cats Results - Analyzing Images (4096 images) 1k epoch

Cats Results - Graphs (4096 images) 1k epoch



Conclusion of this adventure

- Stick to simple images
- Simpler images = better results sooner
- Avoid images rich in detail, such as 3d images
- Around 1000 images or more
- More images = more epoch
- Diverse pictures work great, such as dogs, cats, cars, horses

Complex images result in poor performance, simpler images results in better performance sooner. Will need to apply different baseline models to see if this still holds true.

Data Preparation

- Finding the data
- Find the “right” data and enough of it
- For example sea shells (shapes and texture)
- Sometimes the data fails to import because one file is bad

Statistics

If your interested:

- Ryzen 7 2700X with 16GB, RTX 3080 Aorus Master, 1TB NVMe
  - FYI, the time per epoch goes up the more we train, thus the range
- Lego, 800 images, 2.2s-2.8s per epoch, ran 5k epoch in 2.5-3 hours
- Cutlery, 870 images, 2.2s-2.8s per epoch, ran 5k epoch in 2.5-3 hours
- Marble, 120 images, 1.2s-1.45s per epoch, ran 5k epoch in 1.5 hours
- Cat, 500 images, 2.3s-3.0s per epoch, ran 1k epoch in 1.5 hours
- Cat, 4096 images, 7.8s-8.6s per epoch, ran 1k in 5 hours

- Have lots of RAM, more than 16GB, with 16GB max ingestion was 4k images.
- Enable GPU with Tensorflow (www.youtube.com/watch?v=IubEtS2JAiY)

References

Datasets

- Cat Dataset: https://www.kaggle.com/andrewmvd/animal-faces
- Marble Textures: https://www.robots.ox.ac.uk/~vgg/data/dtd/
- Lego Bricks: https://www.kaggle.com/joosthazelzet/lego-brick-images
- Cutlery: https://homepages.inf.ed.ac.uk/rbf/UTENSILS/

GAN Tutorials and Help

- Tensorflow Tutorial: https://www.tensorflow.org/tutorials/generative/dcgan
- Machine Learning Mastery: https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-cifar-10-small-object-photographs-from-scratch/
- Truly Artificial: https://simran-tinani.medium.com/truly-artificial-generating-and-experimenting-with-artificial-flower-images-using-deep-d11808193e3b
- Towards Data Science: https://towardsdatascience.com/image-generation-in-10-minutes-with-generative-adversarial-networks-c2afc56bfa3b