DCGAN: Deep Convolutional Generative Adversarial Network

Install important packages

```
!pip install tensorflow-gpu==2.0.0-alpha0
# To generate GIFs
!pip install imageio
```

```
Collecting tensorflow-gpu==2.0.0-alpha0
-manylinux1_x86_64.whl (332.1MB)
ent already satisfied: gast>=0.2.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflow-gpu==2.0.0-alpha0) (0.2.2)
Collecting tf-estimator-
nightly<1.14.0.dev2019030116,>=1.14.0.dev2019030115
ator_nightly-1.14.0.dev2019030115-py2.py3-none-any.whl (411kB)
ent already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflow-gpu==2.0.0-alpha0) (1.12.0)
Requirement already satisfied: wheel>=0.26 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (0.33.6)
Requirement already satisfied: grpcio>=1.8.6 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (1.15.0)
Requirement already satisfied: google-pasta>=0.1.2 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (0.1.8)
Requirement already satisfied: keras-preprocessing>=1.0.5 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (1.1.0)
Requirement already satisfied: absl-py>=0.7.0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (0.9.0)
Requirement already satisfied: astor>=0.6.0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (0.8.1)
Collecting tb-nightly<1.14.0a20190302,>=1.14.0a20190301
ent already satisfied: protobuf>=3.6.1 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (3.10.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (1.1.0)
Requirement already satisfied: keras-applications>=1.0.6 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (1.0.8)
```

```
Requirement already satisfied: numpy<2.0,>=1.14.5 in
/usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-
alpha0) (1.17.5)
Requirement already satisfied: werkzeug>=0.11.15 in
/usr/local/lib/python3.6/dist-packages (from tb-
nightly<1.14.0a20190302,>=1.14.0a20190301->tensorflow-gpu==2.0.0-
alpha0) (0.16.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.6/dist-packages (from tb-
nightly<1.14.0a20190302,>=1.14.0a20190301->tensorflow-gpu==2.0.0-
alpha0) (3.1.1)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.6/dist-packages (from protobuf>=3.6.1-
>tensorflow-gpu==2.0.0-alpha0) (42.0.2)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-
packages (from keras-applications>=1.0.6->tensorflow-gpu==2.0.0-
alpha0) (2.8.0)
Installing collected packages: tf-estimator-nightly, tb-nightly,
tensorflow-gpu
Successfully installed tb-nightly-1.14.0a20190301 tensorflow-gpu-
2.0.0a0 tf-estimator-nightly-1.14.0.dev2019030115
Requirement already satisfied: imageio in
/usr/local/lib/python3.6/dist-packages (2.4.1)
Requirement already satisfied: pillow in
/usr/local/lib/python3.6/dist-packages (from imageio) (6.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-
packages (from imageio) (1.17.5)
```

Import TensorFlow and other important libraries

```python
from __future__ import absolute_import, division, print_function,
unicode_literals
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow.keras import layers
import time

from IPython import display
```

Load and prepare the dataset

```python
# Load the MNIST Dataset
(train_images, train_labels), (_, _) =
tf.keras.datasets.mnist.load_data()
```

```
train_images = train_images.reshape(train_images.shape[0], 28, 28,
1).astype('float32')
train_images = (train_images - 127.5) / 127.5 # Normalize the images
to [-1, 1]

BUFFER_SIZE = 60000
BATCH_SIZE = 256

# Batch and shuffle the data
train_dataset =
tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).
batch(BATCH_SIZE)
```

Model Creation

The Generator Model

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False,
input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256) # Note: None is the
batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),
padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2),
padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

Sampling from Generator

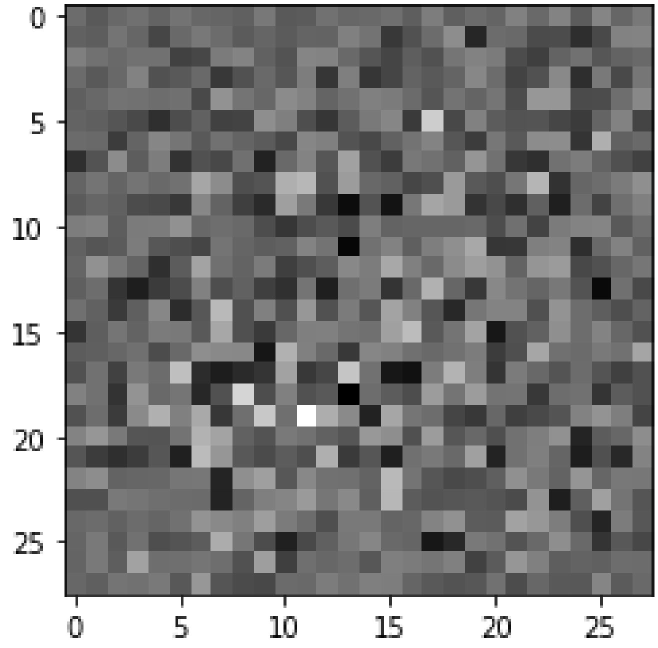
```
generator = make_generator_model()
generator.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 12544)             1254400   
_________________________________________________________________
batch_normalization_v2_3 (Ba (None, 12544)             50176     
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 12544)             0         
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 256)         0         
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 7, 7, 128)         819200    
_________________________________________________________________
batch_normalization_v2_4 (Ba (None, 7, 7, 128)         512       
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 7, 7, 128)         0         
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 14, 14, 64)        204800    
_________________________________________________________________
batch_normalization_v2_5 (Ba (None, 14, 14, 64)        256       
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 28, 28, 1)         1600      
=================================================================
Total params: 2,330,944
Trainable params: 2,305,472
Non-trainable params: 25,472
_________________________________________________________________
```

```
noise = tf.random.normal([1, 100])
print(noise)
```

```
tf.Tensor(
[[-0.71972084 -0.68301564 -1.2953588   1.5932783   0.1587864
1.4874604
  -1.0031837   0.37651387  1.0444032  -0.82308906 -0.60667413
0.51768786
  -1.1837319  -1.0357522  -0.43154112  1.3142896  -1.0219003
1.138651
  -1.1692301   0.9991749  -0.57195437 -1.1872257  -0.98843026 -
0.07879474
   1.16171    -2.1973832   0.17024942  0.85251063  0.78433764
0.69545275
```

```
  -0.4422542  -0.30634564  0.5728824   0.23832941 -0.86591804
1.3435823
   0.55046695  0.42850563  1.1854291  -0.40157956 -0.03179322 -
1.0930563
   0.29245377  0.6465509   1.1045731   0.96080214  0.43721426
1.6654477
   0.60754037  0.4770089  -1.2128993  -1.1565721  -0.3364298
0.19228469
  -0.71483696 -0.12038109  0.24392122  0.30132553 -0.40010163
1.0083213
   0.30977964  0.9936133  -1.8183966  -1.0528294  -1.5051688
0.5815504
  -1.3307948   0.65801656 -0.33610865  2.4291966   0.5598288
0.20861173
  -0.1989685   0.8157344   2.1324925   0.81441444  1.6171186
0.7037948
   1.3942616  -0.5958636  -0.40955848  1.8595178   2.3997855
0.12822227
  -0.94522095 -1.9171835   1.7463205  -1.8636442  -0.12970157 -
2.7207518
   0.73160243 -0.6739517  -0.70676595 -1.120187    0.2649668  -
0.86355174
   1.2634548   1.3374666   0.3613912   0.21842301]], shape=(1, 100),
dtype=float32)

generated_image = generator(noise, training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')

<matplotlib.image.AxesImage at 0x7efdd9263c18>
```



The Discriminator Model

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2),
padding='same',
                                     input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2),
padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

Discriminator Functionality

```
discriminator = make_discriminator_model()
discriminator.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 14, 14, 64)        1664      
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 14, 14, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 128)         204928    
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 7, 7, 128)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 6273      
=================================================================
Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0
_________________________________________________________________
```

```
decision = discriminator(generated_image)
print (decision)

tf.Tensor([[0.000309]], shape=(1, 1), dtype=float32)
```

Loss for Generator and Discriminator

```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

Optimizer for Generator and Discriminator

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

Save checkpoints

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = 
tf.train.Checkpoint(generator_optimizer=generator_optimizer,

discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
```

Experimental Setup

```
EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

# We will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

Training Loop

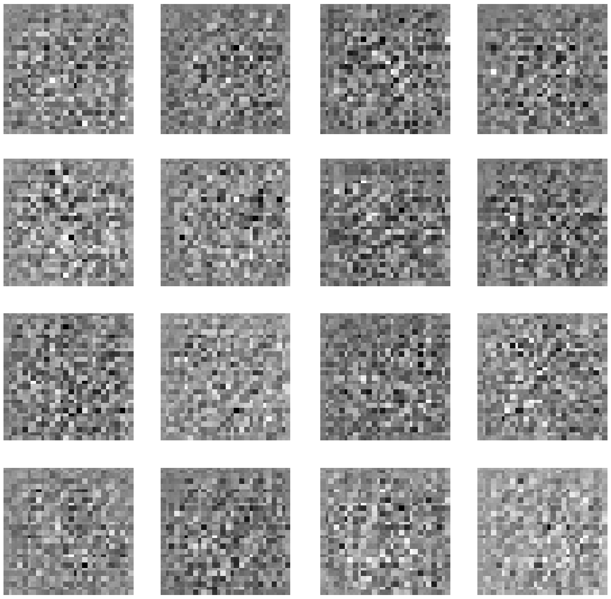
```
# Notice the use of `tf.function`
# This annotation causes the function to be "compiled".
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as
disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss,
discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))

discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator
, discriminator.trainable_variables))

def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    # Produce images for the GIF as we go
    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
                             seed)

    # Save the model every 15 epochs
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1,
time.time()-start))

  # Generate after the final epoch
  display.clear_output(wait=True)
  generate_and_save_images(generator,
                           epochs,
                           seed)
```

Generate and save images

```
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4,4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

Train the model

```
%%time
train(train_dataset, EPOCHS)
```



```
Time for epoch 11 is 11.935768127441406 sec

----------------------------------------------------------------------
-----
```

```
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-38-3bfe38106dd7> in <module>()
----> 1 get_ipython().run_cell_magic('time', '', 'train(train_dataset,
EPOCHS)')

/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.p
y in run_cell_magic(self, magic_name, line, cell)
   2115             magic_arg_s = self.var_expand(line, stack_depth)
   2116             with self.builtin_trap:
-> 2117                 result = fn(magic_arg_s, cell)
   2118             return result
   2119

</usr/local/lib/python3.6/dist-packages/decorator.py:decorator-gen-60>
in time(self, line, cell, local_ns)

/usr/local/lib/python3.6/dist-packages/IPython/core/magic.py in
<lambda>(f, *a, **k)
    186     # but it's overkill for just that one bit of state.
    187     def magic_deco(arg):
--> 188         call = lambda f, *a, **k: f(*a, **k)
    189
    190         if callable(arg):

/usr/local/lib/python3.6/dist-packages/IPython/core/magics/execution.p
y in time(self, line, cell, local_ns)
   1187         if mode=='eval':
   1188             st = clock2()
-> 1189             out = eval(code, glob, local_ns)
   1190             end = clock2()
   1191         else:

<timed eval> in <module>()

<ipython-input-36-802af7bf198a> in train(dataset, epochs)
      4
      5     for image_batch in dataset:
----> 6       train_step(image_batch)
      7
      8     # Produce images for the GIF as we go

/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/def_fun
ction.py in __call__(self, *args, **kwds)
    412       # In this case we have created variables on the first
call, so we run the
    413       # defunned version which is guaranteed to never create
variables.
--> 414       return self._stateless_fn(*args, **kwds)  # pylint:
disable=not-callable
```

```
    415     elif self._stateful_fn is not None:
    416       # In this case we have not created variables on the
first call. So we can

/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/functio
n.py in __call__(self, *args, **kwargs)
   1286     """Calls a graph function specialized to the inputs."""
   1287     graph_function, args, kwargs =
self._maybe_define_function(args, kwargs)
-> 1288     return graph_function._filtered_call(args, kwargs)  #
pylint: disable=protected-access
   1289 
   1290   @property

/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/functio
n.py in _filtered_call(self, args, kwargs)
    572     """
    573     return self._call_flat(
--> 574         (t for t in nest.flatten((args, kwargs))
    575          if isinstance(t, (ops.Tensor,
    576 
resource_variable_ops.ResourceVariable))))

/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/functio
n.py in _call_flat(self, args)
    625     # Only need to override the gradient in graph mode and
when we have outputs.
    626     if context.executing_eagerly() or not self.outputs:
--> 627       outputs = self._inference_function.call(ctx, args)
    628     else:
    629       self._register_gradient()

/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/functio
n.py in call(self, ctx, args)
    413             attrs=("executor_type", executor_type,
    414                    "config_proto", config),
--> 415             ctx=ctx)
    416       # Replace empty list with None
    417       outputs = outputs or None

/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/execute
.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     tensors = pywrap_tensorflow.TFE_Py_Execute(ctx._handle,
device_name,
     59                                                op_name,
inputs, attrs,
---> 60                                                num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:
```

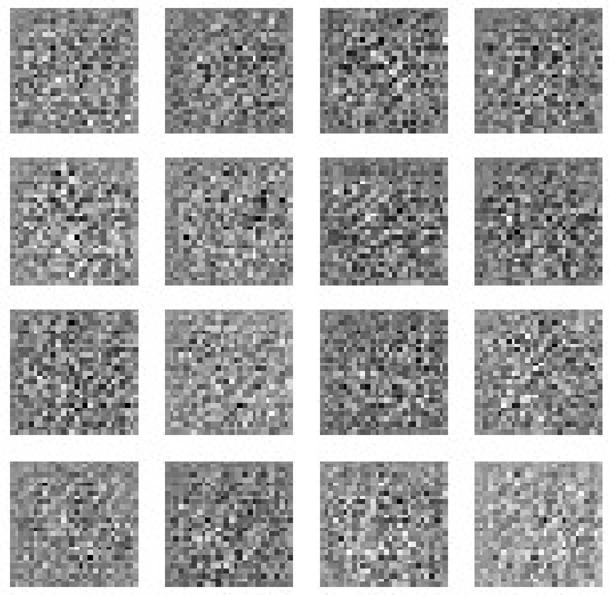
```
KeyboardInterrupt:
```

Restore the latest checkpoint

```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))

<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7efe2a4f9208>
```

Visualize the output

```
# Display a single image using the epoch number
def display_image(epoch_no):
  return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))

EPOCH_NUM = 5
display_image(EPOCH_NUM)
```



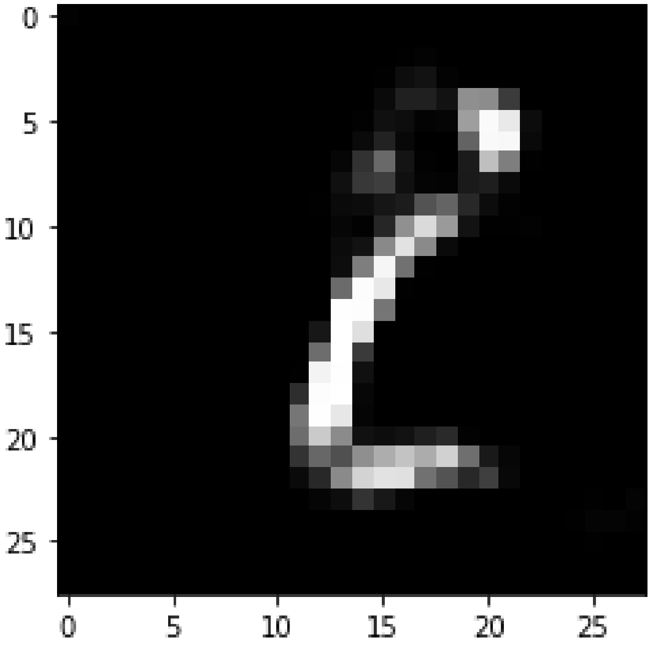
Create a GIF

```
anim_file = 'DCGAN_Animation.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
  filenames = glob.glob('image*.png')
  filenames = sorted(filenames)
  last = -1
  for i,filename in enumerate(filenames):
    frame = 2*(i**0.5)
    if round(frame) > round(last):
      last = frame
    else:
      continue
    image = imageio.imread(filename)
    writer.append_data(image)
  image = imageio.imread(filename)
  writer.append_data(image)

import IPython
if IPython.version_info > (6,2,0,''):
  display.Image(filename=anim_file)
```

Sampling new data

```
# Everytime it will generate new data

noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7efdd99d8c18>
```


0
5
10
15
20
25
0
5
10
15
20
25